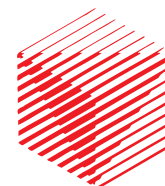


European Research Consortium
for Informatics and Mathematics

ERCIM

www.ercim.org



Proceedings of the
**First ERCIM Workshop
on Software-Intensive
Dependable Embedded Systems**

Porto, Portugal, 30 August - 1 September 2005

Editors: Amund Skavhaug and Erwin Schoitsch



in conjunction with the
31st EUROMICRO conference

DECOS

EU-FP6-511764

ERCIM Workshop Reports

**Proceedings of the
First ERCIM Workshop
on Software-Intensive
Dependable Embedded Systems**

held in conjunction with the
31st EUROMICRO Conference
on Software Engineering
and Advanced Applications (SEAA)

Porto, Portugal, 30 August - 1 September 2005

**Editors:
Amund Skavhaug and Erwin Schoitsch**

Foreword

(a workshop in retrospect)

A few months have passed since we arranged the inaugural ERCIM workshop on Software Intensive Dependable Embedded Systems. The event took place in Porto, Portugal, in cooperation with EUROMICRO SEAA, EUROMICRO DSD, the European Integrated Project DECOS (Dependable Embedded Components and Systems, FP6-IST-511764) and its DECOS Interest Group (DIG).

One of the most irritating aspects of many conferences is the lack of discussion following presentations. Usually this is because audiences are not given access to the papers prior to the conference, and therefore have insufficient time to fully digest the work presented. For this workshop we intended things to be different. Participants were given advance access to all the workshop papers, and asked as their ‘homework’ to prepare some questions. In addition, we reduced the number of papers presented so that there was enough time for each to be treated properly. To further promote the idea that we were going to do some work (workshop...), we promised to include a transcript with ideas and questions in the final proceedings.

We spent two long half-days for the ten presentations. By dividing the day in two, we had enough time to thoroughly discuss each paper, but were not too exhausted at the end of the day to participate. In addition we were able to attend the invited keynote speeches at SEAA/DSD. Most importantly however, there were plenty of opportunities for socializing and building a group spirit.

Work after the event is also important, and one presenter was unfortunately unable to make the presentation while he was in Porto. We tried very hard later to remedy this by arranging a video conference. This solution has a great deal of potential, and we will use it again where necessary. Further, in order to complete our discussions, correct our transcripts and so forth, we set up a ‘WIKI’-style Web site. Unfortunately this fell victim to computer-generated attacks, but with a better technical set-up, it is definitely something that will be employed in the next event.

Not only participants at the ERCIM workshop, but also those present at the EUROMICRO SEAA and DSD have received the booklet of proceedings. It is our hope that this will improve the dissemination of our research and that cooperation and the sharing of knowledge between ERCIM and EUROMICRO will be enhanced.

For the ERCIM Working Group on Software Intensive Dependable Embedded Systems, and serving as your humble organizers of the workshop,

Amund Skavhaug and Erwin Schoitsch

Contents

Foreword (a workshop in retrospect)	3
Message form the Euromicro chairman	7
Papers	
The Integrated Project DECOS: From a Federated to an Integrated Architecture for Dependable Safety-Critical Embedded Systems – an Overview by Erwin Schoitsch, ARC Seibersdorf research, Austria	9
Improving the Performance of Embedded Superscalar Microprocessors by Adding Partial Pipelines by Huibin Shi, Chris Bailey, Department of Computer Science, University of York, UK; Glenn Farrall, Neil Hastie and Sam Jenkins, TriCore Architecture, Infineon Technologies UK Ltd, Bristol, UK.....	15
The PISA Architecture: A Viable Platform for the Superscalar Execution of Statically Scheduled Stack Code by Soyeb Alli and Chris Bailey, Department of Computer Science, University of York, UK	23
SCTL: A StateChart Transformation Language for Test Sets Reduction by Nicolas Guelfi and Benoît Ries, University of Luxembourg, Luxembourg	27
Test Method for Embedded Real-Time Systems by J.P.Bodeveix, R.Bouaziz and O.Koné Université Paul Sabatier - IRIT, Toulouse, France	35
From Message Queue to Ready Queue — Case study of a small, dependable synchronous blocking channels API — “Ship & forget rather than send & forget” by Øyvind Teig, Autronica Fire and Security, Trondheim, Norway	39
An Embedded Future for Distributed System Architectures by Trygve Lunheim and Amund Skavhaug, Department of Engineering Cybernetics, Norwegian University of Technology and Science, Trondheim, Norway	45
Allocation of Dependable Software Modules under Consideration of Replicas by Georg Weissenbacher, Wolfgang Herzner and Egbert Althammer, ARC Seibersdorf research, Austria.....	51
Assessing Reliability of Real-Time Distributed Systems by Åsmund Tjora and Amund Skavhaug, Department of Engineering Cybernetics, Norwegian University of Technology and Science, Trondheim, Norway	59
Assessment of Safety Critical Systems with COTS Software and Software of uncertain Pedigree (SOUP) by Torbjørn Skramstad, Norwegian University of Science and Technology, Trondheim, Norway, and Det Norske Veritas Research, Høvik, Norway	65
Component-Based Context-Dependent Hybrid Property Prediction by Anders Möller, Mikael Nolin, MRTC, Mälardalen University, Västerås, Sweden and CC Systems, Uppsala, Sweden; Johan Fredriksson, Mälardalen University, Västerås, Sweden; Ian Peake and Heinz Schmidt, Monash University, Melbourne, Australia	69
Annex	
Presentation slides	76
List of Participants	124



Message from the Euromicro Chairman

Euromicro is an international scientific, engineering and educational organization dedicated to advancing the arts, sciences and applications of Information Technology and Microelectronics. Euromicro's contributions to the progress in this field are understood as a unique European focal point of activities. In pursuing the mission the highest professional and ethical standards are observed; Euromicro is a non-profit organization.

Euromicro was founded in 1975, in response to, and inspired by, the emerging microprocessor technology. Since that time, Euromicro has been devoted to promoting, discussing, disseminating knowledge, information and skills, in academia, industry, government and in education. As a truly international society Euromicro strictly adheres to impartiality in national and international affairs.

A major focus of Euromicro's activities was to organize conferences for the area of Information Technology and Microelectronics. Actually Euromicro is running four annual international conferences:

- the Euromicro Conference on Software Engineering and Advanced Applications (SEAA);
- the Euromicro Conference on Digital System Design (DSD);
- the Euromicro Conference on Parallel, Distributed and Network-based Processing (PDP);
- the Euromicro Conference on Real-Time Systems (ECRTS).

The Conference Proceedings are published through IEEE Computer Press. Moreover, Euromicro publishes the Journal of Systems Architecture (JSA), through Elsevier.

The Euromicro activities inherently involve co-operation with other European scientific organisations. So, we welcomed and appreciated the possibility of a joint effort with ERCIM, in organizing this Workshop on Dependable Software Intensive Embedded Systems, in connection with our SEAA 2005 / DSD 2005 conference event in Porto, Portugal.

I would like to thank Amund Skavhaug and all others involved in the preparation of this workshop, for their work to make this event a success. I would also like to express my hope that - in addition to the scientific success - this workshop has also increased interest and motivation to participate in future Euromicro conference events.

Erwin Grosspietsch
Chairman of the Euromicro Board of Directors

<http://www.euromicro.org>

The Integrated Project DECOS¹

From a Federated to an Integrated Architecture for Dependable Safety-Critical Embedded Systems – an Overview

Erwin Schoitsch
ARC Seibersdorf research
erwin.schoitsch@arcs.ac.at

Abstract

In dependable embedded systems, currently each application function is assigned to a separate ECU (embedded control unit). Multi-tier supply chains as in automotive industry even enforce this approach to protect the IPs (Intellectual Property) of each supplier. With growing complexity of distributed systems and expanding functionality, this approach tends to become critical because of integration problems. In the federated approach, integration requires all subsystems to inherit the highest SIL (safety integrity level), too many ECUs, buses and connectors decrease dependability considerably. Therefore, research activities aim at an integrated approach, allowing the deployment of multiple application subsystems on a single distributed computer system [1]. DECOS is developing a generic architecture (middleware) based on time-triggered core technology, guaranteeing high dependability for critical embedded systems and allowing integration of safety-critical and non-safety-critical subsystems. It covers the full life-cycle, from the top-level platform independent models of the distributed application subsystems down to the actual deployment [5], taking into account software, hardware and validation/certification issues as well. DECOS is domain- and platform independent, as long as certain requirements are fulfilled by the core platform (e.g. predictable temporal behavior). This is shown by demonstrators from the automotive, aerospace and industrial control area, and by implementation on three different core platforms: TTA (TTP/C, time triggered architecture), Layered FlexRay and TT-Ethernet.

1. Introduction

“Smart Systems” are based on intelligent embedded control systems, which are distributed within the application systems, and often hidden to the every-day life user. E.g., more and more functions in today’s cars are realized by electronics and software, 80-90% of the new innovative features are realized by distributed embedded systems. Eventually, even highly safety critical mechanical and hydraulic control systems will be replaced by electronic components. Value of electronics in cars will increase beyond 40% of the total value. Even today, upper class cars contain up to 80 ECUs, several bus systems, and about 55% of failures are caused by electronics, software, cables and connectors.

The DECOS project [2] aims at making a significant contribution to the safety of dependable embedded systems by facilitating the systematic design and deployment of integrated systems [1]. In federated systems, each application subsystem is located on a dedicated processor. The federated approach provides natural separation of application functions, but causes increased weight, electric energy consumption and cost due to resource duplication and the large number of wires, buses and connectors. Integrated systems not only help to alleviate this problem, they also permit communication among application functions. A remarkable feature of the integrated DECOS architecture is that hardware *nodes* are capable of executing several *tasks* of application subsystems of different criticality (see fig. 4, fig. 5). Throughout this paper, we will use the notion of a *node* instead of processor or component.

¹ Research supported in part by EC IST FP6 IP DECOS No. 511764

An integrated architecture provides a fixed number of nodes, each of which has certain properties (e.g., size of memory, computational power, I/O resources). All tasks have to be allocated such that given functional and dependability constraints are satisfied. This is discussed in detail in [5].

DECOS started July 1st, 2004, and is planned to last for 3 years (June 30, 2007). The budget is about 14,3 Mio € funding about 9 Mio €. The project is performed by 19 partners, which are:

- ARCS (Austrian Research Centers Seibersdorf research), coordinator.
- Universities: TU Vienna, TU Darmstadt, Budapest University of Technology and Economics, University of Kassel, University of Kiel, TU Hamburg-Harburg
- TTTech, Esterel, Infineon München (Technology Providers)
- Audi AEV, Hella, Fiat Research Center CRF (Automotive Demonstrator)
- Airbus Deutschland, EADS, Thales Avionics, Liebherr Aerospace Lindenberg (Aerospace Demonstrator)
- Profactor (Industrial Control Demonstrator)
- Swedish Test- and Research Center (SP)

The project is subdivided into subprojects, which are lead by core partners, each supported by a group of partners:

- Architecture Design (TU Darmstadt, TU Vienna)
- Component Design and Integration (TTTech)
- Silicon Infrastructure (Infineon)
- Validation and Certification (ARCS)
- Automotive Application (Demonstrator) (Audi)
- Aerospace Application (Airbus)
- Industrial Control Application (Profactor)
- Training, Dissemination, Standardization, Policy and Gender issues (ARCS)
- IP Management and Co-ordination (ARCS)

2. DECOS Motivation

DECOS methodically targets, investigates, and develops approaches to significantly alleviate - elimination would be an idealised goal - the identified five key obstacles to the (mass) deployment of advanced electronic functions in distributed, hard real-time and critical embedded systems by choosing an integrated architecture approach:

- Electronic Hardware Cost (fewer ECUs, cables, connectors)

- Enhanced Dependability by Design (clear partitioning of safety-critical and non-safety-critical subsystems),
- Development Cost (modular certification, reuse of software components, structured integration of communication and computational elements)
- Diagnosis and Maintenance (diagnosis of transient and intermittent component failures)
- Intellectual Property (IP) Protection (allocating software-jobs as “black-box” building blocks on target hardware in an encapsulated execution environment and communication via virtual communication links)

The intent is to provide an integrated distributed execution platform and a set of pre-validated *hardware components* and *software modules* and tools for the design of dependable embedded systems. Generic design solutions for integrated dependable systems will be developed such that the invariance of the design strategies and technology neutral interfaces are considered upfront as a design objective. System design approaches that are applicable to diverse application domains will be considered, target areas are (but not restricted to) automotive, aerospace, industrial control, railways, medical devices and systems, robotics, autonomous systems (other approaches, e.g. the Integrated Modular Avionics System IMA in Airbus 380 are domain specific).

DECOS builds on the substantial results of previous European research projects (NextTTA, FIT, TTA, SETTA, RISE, X-By-Wire, PDCS, DEVA, DSOS). The components and tools (prototypes) developed within DECOS will cover: cluster design, middleware and code generators, validation and certification as well as systems-on-a-chip (SoC) for high dependability applications.

3. DECOS Core Services and Architecture

Applications are composed of subsystems of different levels of criticality, e.g. crash prevention system, engine control system, door lock system, driver assistance systems, comfort systems.

The advantages of a federated system are encapsulation and IPR protection by ECU separation, the clear disadvantages are cost, space, power consumption, complexity in cabling, connectors and buses (including higher EMC vulnerability).

The integrated approach has the advantage of cost reduction, less space and weight, less power consumption, software-only implementation, the dependability

benefit of fewer connectors, cables and ECUs. The research and design task is to cope with the disadvantages: separation of jobs (to prohibit interference between “jobs” (software components or building blocks) in time and space). Therefore the main requirements for a dependable integrated architecture are:

- Predictable allocation of time slots to jobs in DASs (Distributed Application Subsystems, allocated in nodes, see fig. 1, fig. 4)
- Encapsulated Execution Environment (EEE)
- Virtual Communication Links to embed jobs as software components in DECOS nodes despite the non-existence of the dedicated ECU and bus

A new challenge is the allocation of jobs, predefined at design time, to meet performance, resource and dependability constraints.

The properties (core services) required from any core platform are (see fig. 1):

- Deterministic and timely message transport
- Fault tolerant clock synchronization
- Strong fault isolation
- Consistent diagnosis of failing nodes

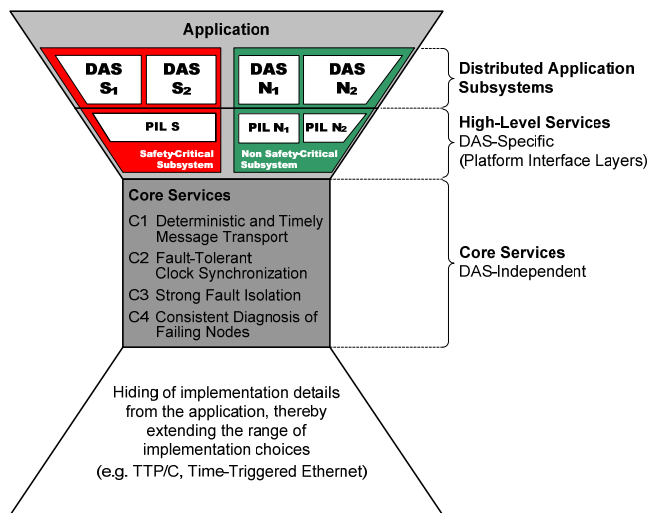


Fig. 1: Distributed Application Subsystems and Core Services

On top of the (existing/assumed) core services, the DECOS High Level Services are developed within the DECOS project, including tools for generation of applications (deployment) from the PIM (platform independent model) by transformations utilizing the hardware specific model (HSM, Resource Specification), the PI (Platform Interface) and the platform specific

model (PSM). The approach allows to reuse pre-developed models, specifications and platform interfaces (PI) (see fig. 2 and fig. 3).

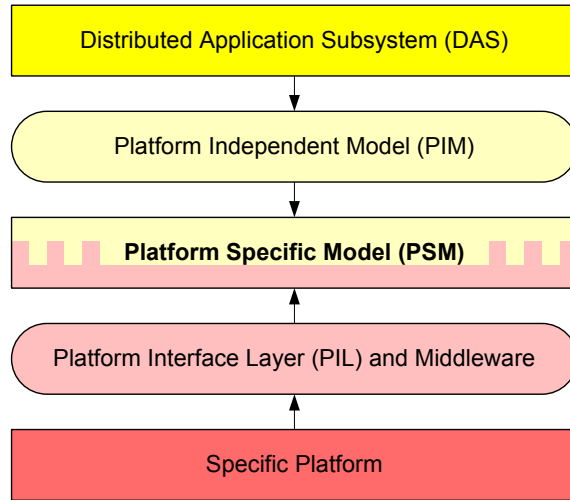


Fig. 2: DECOS Development Hierarchy

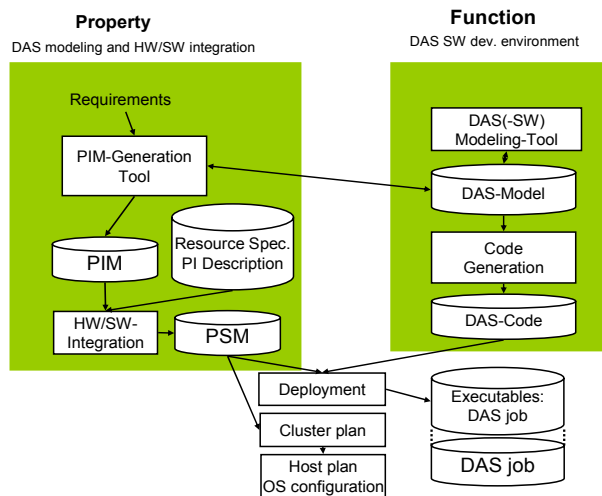


Fig. 3: Development from Model to Deployment

To reduce (mental) complexity, a DECOS application is composed of (nearly) independent subsystems with precisely defined LIFs (linking interfaces), the internals are hidden. This is achieved by a well defined DECOS architecture. A DECOS node is a FCU (Fault Containment Unit) (see fig. 4). Each job is loaded into a separated EEE (Encapsulated Execution Environment), realized in hardware because of the need for safe separation of EEEs from each other. Each EEE

can contain its partition OS (Operating System) as required by the job. The overall management of the EEE is performed by a Core Operating System taking care of the separation in space (memory) and time (fixed time slots are assigned to the job and partition, no other clock base is known and accessible to the job).

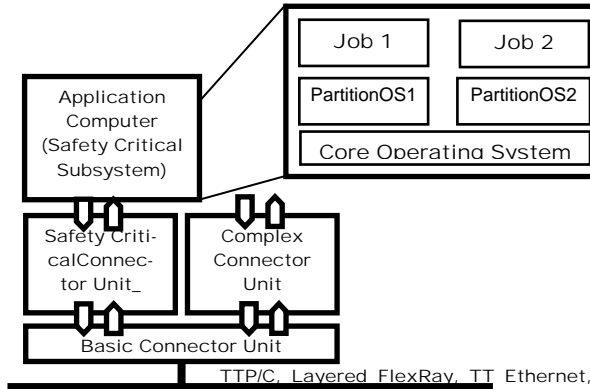


Fig. 4: DECOS node (FCU – fault containment unit)

The safety critical applications are connected to the underlying time-triggered bus infrastructure (network) via a basic connector unit and the safety critical connector unit (SCCU). The non-safety critical applications and legacy systems are connected to the basic connector unit via the complex connector unit (CCU) to absolutely inhibit control flow from the non-safety critical applications to the safety-critical part of the system. This allows the non-safety critical subsystems to be of a lower criticality class (Safety Integrity Level, SIL, according to IEC 61508) than the critical subsystems.

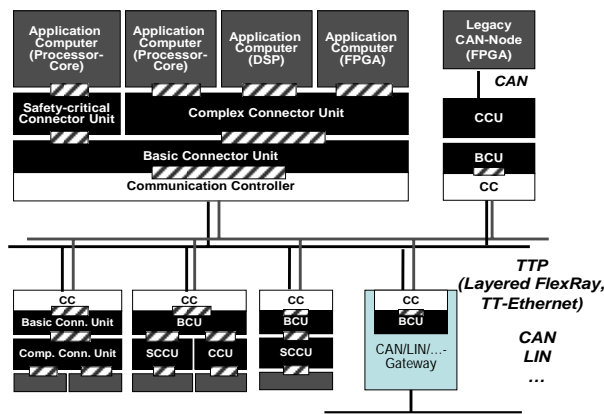


Fig. 5: Detailed structure of DECOS System

Following the same concept, gateways can be built to connect other buses in a safe manner to the time-triggered core services (detailed structure see fig. 5).

4. Component Design and Silicon Infrastructure

As part of the “Component Design” subproject, the Encapsulated Execution Environment is developed and implemented in hardware. Additionally, the virtual communication links and gateways, as described before, the extensive diagnostic services identifying all out-of-norm conditions, and the optimized fault tolerant layer are part of the work.

Within the project, each hardware node currently consists of a single processor, the Infineon Tri-Core1796, which will run the partition operating system. It will host the application jobs, all connector units and - presumably - the communication controller, as shown below. The hardware fault-tolerant layer (HFTL) and hardware event layer (HEVL) with high level services (middleware) are implemented in FPGA (prototype for later System-on-Chip implementation, which cannot be part of the research project)(fig.6)

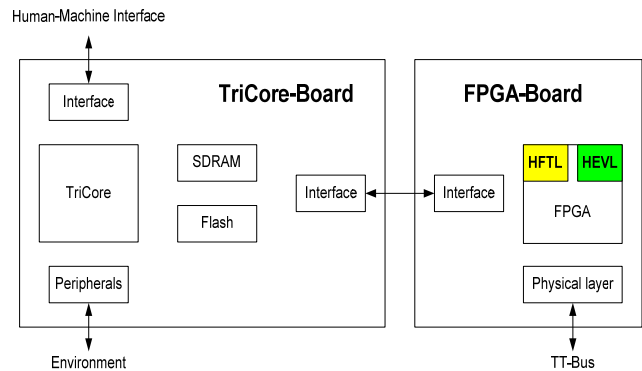


Fig.6: Silicon Infrastructure-Prototype Implementation

5. Towards Modular Certification: the Generic Safety Case

The subproject “Validation and Certification” has the goal to facilitate certification of DECOS-based systems in a modular (component based) manner, making use of properties of the DECOS core services, high level services, design and development process properties. Basis is the generic functional safety standard IEC 61508. A comparison and evaluation of several domain specific standards and IEC 61508 has shown, that systems conforming to higher SIL levels of IEC 61508 or related standards fulfill the major requirements of do-

main specific standards, such as IEC 50129 (railways), the evolving ISO 26262 automotive functional safety standard (the so-called FAKRA standard proposal) or RTCA/DO 178B for aircraft industry.

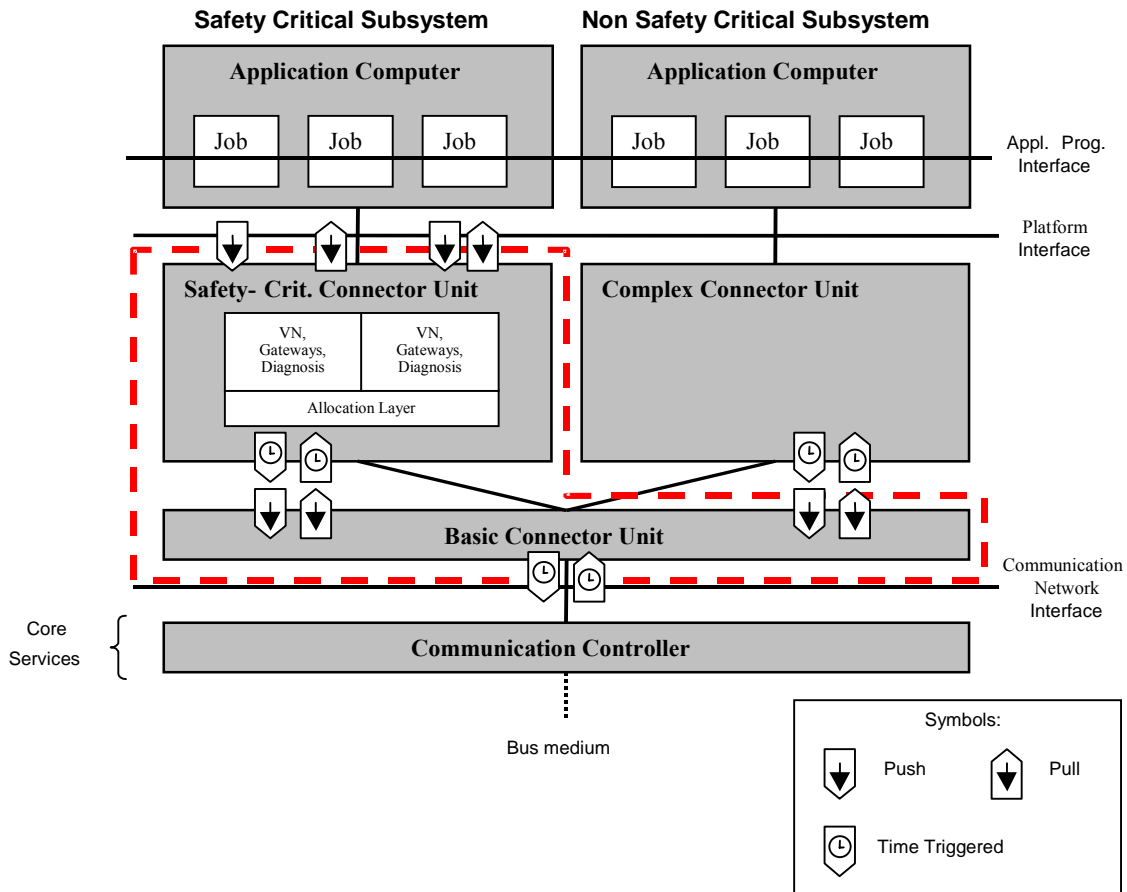
A generic Test bench is developed, accumulating all the tools and the experience of the partners for validation of DECOS components and systems and following the standards' requirements and design processes from PIM modeling down to deployment, initiating the performance of adequate tests available from a repository. The activities can be distributed and delegated to adequate test labs (e.g. EMC (Electromagnetic Compatibility) test labs for EMI (Electromagnetic Interference), or HIFI (Heavy Ion Fault Injection) to a partner with adequate equipment etc.), the results to be collected and a safety case being built from the documents for evaluation. A separate issue to be managed (only once per case) is the validation of tools and processes. Since DECOS is a research project and not for development of products, certain issues will not be tackled during the project (QM System, strict development processes), but advice will be given how the requirements of standards could be fulfilled in a production process.

In a first step towards certification, an approach to demonstrate certifiability of DECOS systems in a

modular, component based (“incremental”) manner is the development of a “Generic Safety Case”. This safety case builds at this stage on the inherent assumptions and the assumed fulfillment of the requirements defined for all the subprojects (architecture claims, core services, high level services). It is up to the subprojects to prove that the requirements have been met or can be met in a production process (“certifiability”).

The construct to be looked at is shown in fig. 7. It consists of the safety-critical part of a DECOS node, including the SCCU (safety-critical connector unit) and the PI (Platform Interface), but not the application, because only the generic part is looked at. The result is an evaluation and assessment of the contribution of the DECOS architecture to the safety of application systems, which is intended to be included in the safety case of DECOS-based applications and is expected to facilitate this part of the system certification.

Fig. 7: DECOS node, view for Generic Safety Case



6. Conclusions

DECOS will provide a set of methodologies and (prototype) tools for composable, integrated design of dependable, critical embedded systems, from top-level model (PIM) down to code generation and deployment, including validation, verification and certification support. The goal is to facilitate re-use of software-, hardware and middleware components by making use of the high-level services developed by DECOS. This will be achieved by a platform and domain independent integrated architecture of the DECOS high level services and by support through the DECOS tool-chain. The approach will be evaluated by industrial application subprojects (demonstrators) in the automotive, aerospace and industrial control domain.

In the automotive application, a mixed criticality approach is demonstrated by integrating a door-control system and a critical crash warning and avoidance system demonstrator (vehicle and environment simulator with DECOS hardware in-the-loop, based on layered FlexRay core technology).

The aerospace demonstrator is a flap control system for the Airbus outer flap control, a really critical application, with a gateway to the AFDX-bus of Airbus (see fig. 8).

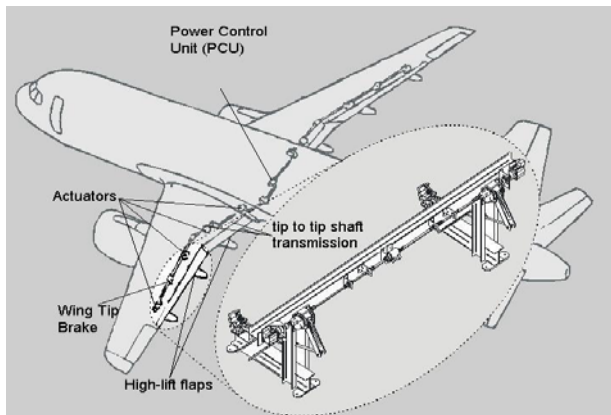


Fig.8: Airbus Outer Flap Control

The industrial control demonstrator is control of a production- and business critical vibration control system for high-end nano-imprinting machines, controlling piezo-electric sensor and actuator networks. The long term vision of this demonstrator is critical structural control of engineering structures (helicopter cabins, aircraft wings, buildings, noise suppression etc.).

In one of the experiments, some properties of time-triggered Ethernet as core service will be evaluated.

7. References

- [1] H. Kopetz, R. Obermaisser, P.Peti and N. Suri, "From a Federated to an Integrated Architecture for Dependable Embedded Real-Time Systems (draft)", Vienna University of Technology, Austria, and Darmstadt University of Technology, Germany, 2004.
- [2] Dependable Embedded Components and Systems, Integrated Project within the EU Framework Programme 6, <http://www.decocos.at>
- [3] N. Suri, A. Jhumka, M. Hiller, T. Marlowe, "A Software Integration Approach for Designing Dependable Distributed Systems" submitted to Elsevier Science, 2004.
- [4] S. Islam, G. Csertan, W. Herzner, "A Multi Variable Optimization Approach for SW-HW Integration", Fast Abstract to appear at HASE 05: High Assurance Systems Engineering Conference, Heidelberg, Germany, 2005
- [5] G. Weißenbacher, W. Herzner, E. Althammer, "Allocation of Dependable Software Modules under Consideration of Replicas", Proceedings of the ERCIM/DECOS Workshop on Dependable Software-Intensive Embedded Systems at Euromicro 2005, Aug. 31-Sept.1, 2005, Porto, Portugal.

Improving the Performance of Embedded Superscalar Microprocessors by Adding Partial Pipelines

Huibin Shi¹, Chris Bailey¹, Glenn Farrall², Neil Hastie² and Sam Jenkins²

¹ Department of Computer Science, University of York, York, YO10 5DD, U.K.

² TriCore Architecture, Infineon Technologies UK Ltd (Bristol), Hunts Ground Road, Stoke Gifford, Bristol, BS34 8HP, UK

¹ {shi, chris}@cs.york.ac.uk ² {Glenn.Farrall, Neil.Hastie, Sam.Jenkins}@infineon.com

Abstract

This paper introduces adding a “partial” pipeline to a base embedded superscalar microprocessor implementation to achieve cost effective performance improvements. This method is exemplified by adding a “partial” integer pipeline (IP) to the TriCore™ 2.0 MCU/DSP core. The “partial” IP pipeline, designed based on TriCore 2.0 simulation results of the EEMBC benchmark suite, executes a subset of TriCore 2.0 IP instructions. We used the basic block sampling and simulation technique to simulate enhanced TriCore 2.0 models, and obtained results indicating that adding the partial IP pipeline can achieve similar performance improvements to duplicating the full IP pipeline. Our approach can be applied to the early design stages of microprocessor development in order to explore design spaces.

1. Introduction

Duplicating an instruction pipeline of a base microarchitecture has been explored as an approach to improve the performance of a general purpose superscalar microprocessor[1]. The duplicated pipeline is implemented in hardware in the same way as the original pipeline of the base microarchitecture and can execute the same instruction set. Since not all instructions have the same execution probabilities, the additional silicon area of the duplicated pipeline required for those instructions with small execution probabilities can not achieve optimal cost effective performance improvements.

Unlike processors at the high end that can afford to pay a very high premium for a small performance improvement [2], the embedded microprocessor and system are very sensitive to silicon area and cost [3] [4]. We customize this general approach of duplicating the full original pipeline, and suggest adding a partial pipeline to achieve a cost effective performance enhancement. The partial pipeline is designed to execute

a subset of instructions of the original full pipeline which have the highest execution probabilities. This enables the partial pipeline to deliver more performance per unit of silicon area than an additional full pipeline.

Our method is generic and can also be applied to adding other types of partial pipelines in addition to those discussed. In this paper, we choose TriCore 2.0 [4] [5] as a commercial example of an embedded superscalar 32 bit MCU/DSP microprocessor core for System-on-Chip(SoC) automotive electronics to introduce and validate our approach. TriCore 2.0 has three pipelines: an integer pipeline (IP) for arithmetic and logic operations on data registers, a load store (LS) pipeline for address operations using address registers, and a loop (LP) pipeline for executing zero-overhead loops. We abstracted TriCore 2.0 and developed a 3-pipeline microprocessor model of the three pipelines, as the base microarchitecture in our study.

The partial pipeline aims to provide cost effective performance enhancement of the base microarchitecture. To explore the design space of cost/performance tradeoffs, we designed eight different partial IP pipelines executing various subsets of TriCore 2.0 IP instructions that the original full IP pipeline can also execute. Those instructions were selected based on the instruction execution probabilities obtained from the statistics of the TriCore 2.0 simulation results, using the EEMBC’s automotive/industrial embedded microprocessor benchmark suite [6] consisting of sixteen benchmarks. Since the implementation of the multiplier and divider could take considerable silicon area and be very expensive based on [7], we split those eight partial IP pipelines equally into two groups, half with and half without the implementation of the multiplier and divider. We design this range of partial IP pipelines to explore the design space in relating to cost/performance tradeoffs, which are analyzed in an analytic approach, instead of a pure quantitative way.

The partial pipelines described are used to enhance the base microarchitecture to form eight special 4-pipeline microarchitectures. In order to compare our approach with the conventional pipeline duplication approach, we also duplicate the full IP pipeline of the base

microarchitecture and form a standard 4-pipeline microarchitecture. For this comparison, we simulate those microarchitectures with the EEMBC automotive/industrial benchmark suite using the basic block sampling and simulation technique, since a cycle-accurate simulator could not be provided for all of those microarchitectures. Our simulation approach is simple and fast, through combining two relatively complicated techniques, the statistical simulation [8] and reduced sampling [9]. The simulation results are used to measure the relative performance of the nine enhanced microarchitectures over that of the base 3-pipeline microarchitecture, instead of representing the actual performance of each microarchitecture.

In order to provide visual verification of the correct simulation of each microarchitecture, the basic block simulation approach extends the method in [10] and generates a graph representing the issuing results of each basic block. Experimental results show that a partial IP pipeline can provide performance results up to that generated by duplicating the full IP pipeline as well as a range of design space cost/performance tradeoffs .

2. Research methodology

2.1. Computing the weighted execution probability of an instruction

In our study, the whole of the EEMBC automotive/industrial benchmark suite was regarded as the aggregate workload and each benchmark was a part of it [11]. The overall execution probability of an instruction is computed as the total number of times that the instruction is executed, i.e. the total number of occurrences in the trace file, over the total number of times of all instructions executed in all the benchmarks, based on TriCore 2.0 simulation results.

Let $OverallP_i$ be the overall execution probability of the i -th instruction of the whole instruction set and T_{ij} be the total number of execution times of the i -th instruction in the j -th benchmark of the benchmark suite, $OverallP_i$ can be computed with the following equation:

$$OverallP_i = \frac{\sum_{j=1}^{16} T_{ij}}{\sum_{i=1}^m \sum_{j=1}^{16} T_{ij}} \quad (1)$$

where ‘16’ represents the total sixteen benchmarks of the EEMBC automotive/industrial benchmark suite and ‘ m ’ represents the total instruction numbers in the TriCore 2.0 instruction set.

2.2. Computing the performance of the microarchitectures

We use the overall weighted arithmetic mean (WAM), of the instructions per cycle (IPC) for the whole EEMBC automotive/industrial benchmark suite to represent the performance of each microarchitecture, based on [11]. The weight of each benchmark is based on its execution cycles over that of the whole benchmark suite.

Let j represent the j -th benchmark of the EEMBC automotive/industrial benchmark suite, overall WAM of IPC of a microarchitecture is computed with the following formula, based on the proof in [11]:

$$WAMofIPC = \frac{\sum_{j=1}^{16} I_j}{\sum_{j=1}^{16} C_j} \quad (2)$$

where I_j and C_j respectively represent the total instruction count and the total execution cycles of the j -th benchmark.

The speedup is used to represent the relative performance factor of the standard and enhanced 4-pipeline microarchitectures over the 3-pipeline base microarchitecture. It is computed with the following formula, based on [1].

$$Speedup = \frac{WAMofIPC(enhanced)}{WAMofIPC(base)} \quad (3)$$

2.3. The overall research flow

Figure 1 displays the overall flow of the partial IP pipeline generation and the performance analysis process in our study, which consists of three stages. In Figure 1, each stage is represented by a rectangle with a dashed line, in which the main operation is represented by the rectangle with a solid line. The thin arrowed line represents the output of one operation being input into another operation pointed to by the arrow. The large dashed arrow and the large solid arrow represent the input and the output of the three stages, respectively. Figure 1 shows that some of the outputs of Stages 1 and 2 are inputs into Stage 3.

Figure 1 shows that in Stage 1 the EEMBC benchmarks, after being compiled and linked into the assembly list file, are input into the TriCore 2.0 simulator to perform the ‘Instruction set simulation’ operation. The operation generates a trace file for each benchmark, which includes the execution frequency of each instruction and basic block. Then the ‘Instruction execution trace profiling’ operation processes the trace file and outputs instruction execution statistics, through

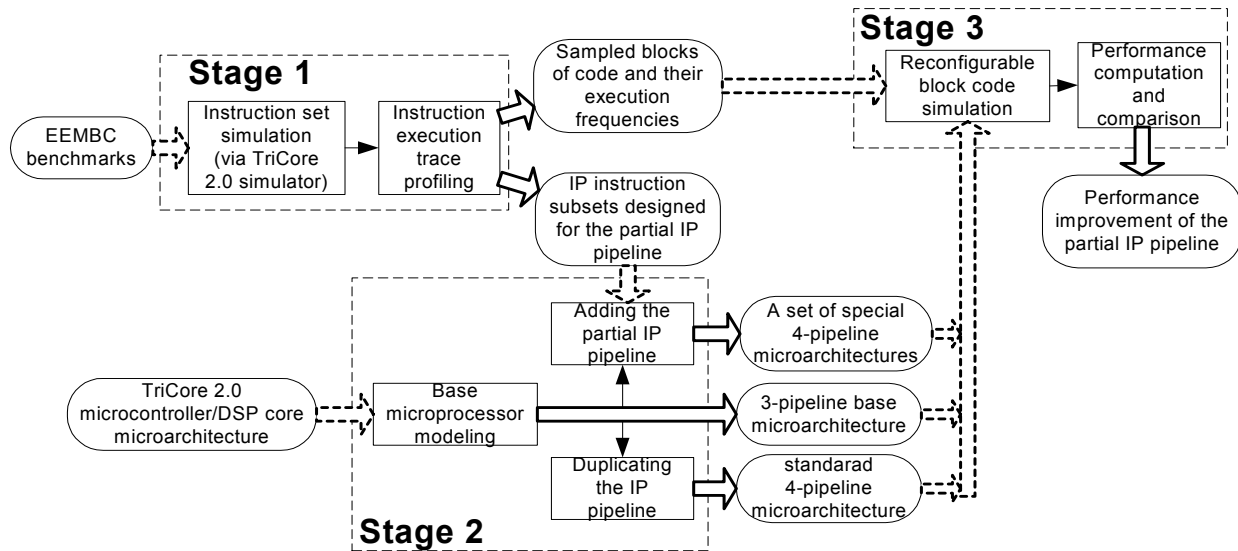


Figure 1. The process of partial pipeline generation and performance analysis with the reconfigurable block code simulation tool

analyzing the body of the benchmark and excluding initialization and other housekeeping code.

Stage 1 outputs the instruction execution statistics in two parts. The first part consists of sampled blocks of instruction code, which are basic blocks of TriCore 2.0 instructions executed at least once in the execution trace, together with their execution frequencies. This part is fed into Stage 3 for the simulation of various microarchitectures.

The second part comprises of the overall execution probabilities of all instructions, which are computed based on the equation (1) and are used to select IP instructions with the highest execution probabilities. Those selected IP instructions are divided into eight groups for different partial IP pipelines, in order to explore the design space for different silicon costs. Those partial IP pipelines are late input into Stage 2.

Figure 1 shows that in Stage 2, the first operation, ‘Base microprocessor modeling’ models TriCore 2.0 and outputs the 3-pipeline base microarchitecture model with the IP, LS and LP pipelines. Since TriCore 2.0 supports an optional floating point unit (FPU) with a coprocessor interfacing in the IP pipeline, FPU instructions are issued to the IP pipeline in the model. Since our research method does not require the support of special pipeline stage structures, we do not model stage details of those pipelines. The memory and branch characteristics of TriCore 2.0 are not modeled, which does not affect the generality of our research method on measuring the relative performance improvements of enhanced microarchitectures.

The ‘Duplicating the IP pipeline’ operation duplicates the IP pipeline of the 3-pipeline base microarchitecture model and outputs the standard 4-pipeline microarchitecture model. In Stage 2, the ‘Adding partial

IP pipeline’ operation takes a set of partial IP pipelines output from Stage 1, adds them into the 3-pipeline base microarchitecture model and outputs a set of special 4-pipeline microarchitecture models, with the ‘special’ expressing a partial IP pipeline in them. The partial IP pipeline has the same pipeline stages as a full IP pipeline.

Figure 1 shows that in Stage 3, the ‘Reconfigurable block

code simulation’ takes as the input the sampled blocks of code, output from Stage 1, and the microarchitecture configurations, output from Stage 2, to simulate individual microarchitectures. It simulates the execution of each basic block and generates the graph representing the instruction issuing result for each microarchitecture. The execution cycle count of each basic block is generated and multiplied with the block execution frequency to report the total number of execution cycles in a benchmark.

After that, the overall WAM of IPC, the overall performance of a microarchitecture, is computed based on equation (2). The speedups of the enhanced microarchitectures are determined based on equation (3) and compared to show the relative performance improvement of adding the partial IP pipeline.

3. Experimental results

3.1. Overall instruction execution probability results and the partial IP pipeline design

Table 1 lists IP instructions with the overall execution probability of at least 0.02%, in descending order in the column ‘Freq.’. In the TriCore 2.0 ISA, some instructions have both 16 bit version, represented by appending ‘16’ to the end of the opcode, e.g. ‘add16’ in Table 1, and 32

bit version without appending of ‘16’, e.g. ‘add’ in Table 1. In Table 1, the execution probability of an opcode appended with ‘(16)’, e.g. ‘add (16)’ represents the sum of the probabilities of both versions, and that of a bare opcode represents the probability of its 32 bit version only.

Table 1: IP instructions with high execution probabilities

Opcode	Operation type	Freq.	Group(without mul/div operations)	Group(with mul/div operations)
add (16)	addition	7.70%	A	E
sha (16)	arithmetic shift	6.50%	A	E
jlt	cond. jump	2.37%		
mul (16)	multiplication	2.12%		E
sub (16)	subtraction	2.05%	A	E
madd	multiply add	1.34%		E
msub	multiply sub	0.99%		E
mov (16)	move	0.95%	B	F
jz (16)	jump if zero	0.84%		
jne (16)	cond. jump	0.51%		
dextr	extract data	0.43%	B	F
sh (16)	shift	0.39%	B	F
jge	cond. jump	0.39%		
jz.t (16)	cond. jump	0.38%		
lt (16)	compare data	0.35%	B	F
insert	insert bit	0.27%	C	G
or (16)	logic or	0.23%	C	G
sel	select data	0.19%	C	G
and (16)	logic and	0.18%	C	G
jeq (16)	cond. jump	0.14%		
rsub (16)	reverse subtract	0.12%	C	G
movh	move data	0.12%	C	G
jnz (16)	cond. jump	0.10%		
extr.u	extract bits	0.09%	C	G
subc	subtract+carry	0.09%	C	G
subx	subtract extended	0.09%	C	G
addc	add+carry	0.07%	C	G
mul.u	multiply unsign	0.07%		
dvstep	divide step	0.06%		H
andn.t	bit and-not	0.05%	D	H
addx	add extend	0.04%	D	H
and.eq	equal accumulate	0.04%	D	H
jlez (16)	cond. jump	0.04%		
xor.t	bit xor	0.04%	D	H
addi	add immediate	0.04%	D	H
eq (16)	equal	0.03%	D	H
clz	count lead zero	0.03%	D	H
jnz.t (16)	cond. jump	0.03%		
ge	greater/equal	0.02%	D	H
mov.u	move unsign	0.02%	D	H
lt.u	less than unsign	0.02%	D	H

Other IP instructions with smaller execution probabilities are not included in Table 1 and will not be considered in the partial IP pipeline design, since they do not contribute much to the overall performance improvement. However, the original IP pipeline of the base microarchitecture can still execute those instructions.

In Table 1, instructions selected to be duplicated in the partial IP pipeline are assigned with one or two group names for two types of partial IP pipelines, which represents the second part output of Stage 1 in Figure 1. In Table 1, instructions of group A to D in the column ‘Group (without mul/div operations)’ are the partial IP pipelines without the multiplier and divider implemented.

Instructions of group E to H in the column ‘Group (with mul/div operations)’ are the partial IP pipelines with the multiplier and divider. We design these two types of partial IP pipelines in order to explore the relative performance gains of the multiplier and divider and evaluate architectural design decisions.

Table 2 lists all ten microarchitectures in our study with the number of each type of the pipeline, which represents the output of Stage 2 in Figure 1. Table 2 also illustrates the groups of IP instructions, defined in the Table 1, that each of the eight partial IP pipelines can execute. In those groups, when one instruction has 16 bit and 32 bit versions, both versions of the instruction are included.

Table 2. Microarchitectures in our study

Architecture name	Full IP pipeline	Full LS pipeline	Full LP pipeline	Partial IP pipeline	Group of instructions for partial IP
4-pipeline (A)	1	1	1	1	A
4-pipeline (B)	1	1	1	1	A, B
4-pipeline (C)	1	1	1	1	A, B, C
4-pipeline (D)	1	1	1	1	A, B, C, D
4-pipeline (E)	1	1	1	1	E
4-pipeline (F)	1	1	1	1	E, F
4-pipeline (G)	1	1	1	1	E, F, G
4-pipeline (H)	1	1	1	1	E, F, G, H
3-pipeline (base)	1	1	1		
4-pipeline	2	1	1		

For each microarchitecture in Table 2, instructions are issued out of order, and the maximum width of the issue packet is equal to the maximum issue width [1]. When a loop instruction is within the packet, the maximum issue width is the same as the total number of pipelines. When the packet does not have a loop instruction, the maximum issue width is one less than the total number of pipelines. [see note A at the end of the paper] All instructions are assumed to have one cycle latency, and their operation results are readily available in the next cycle through data forwarding.

3.2. Graphical representation of the instruction issuing results

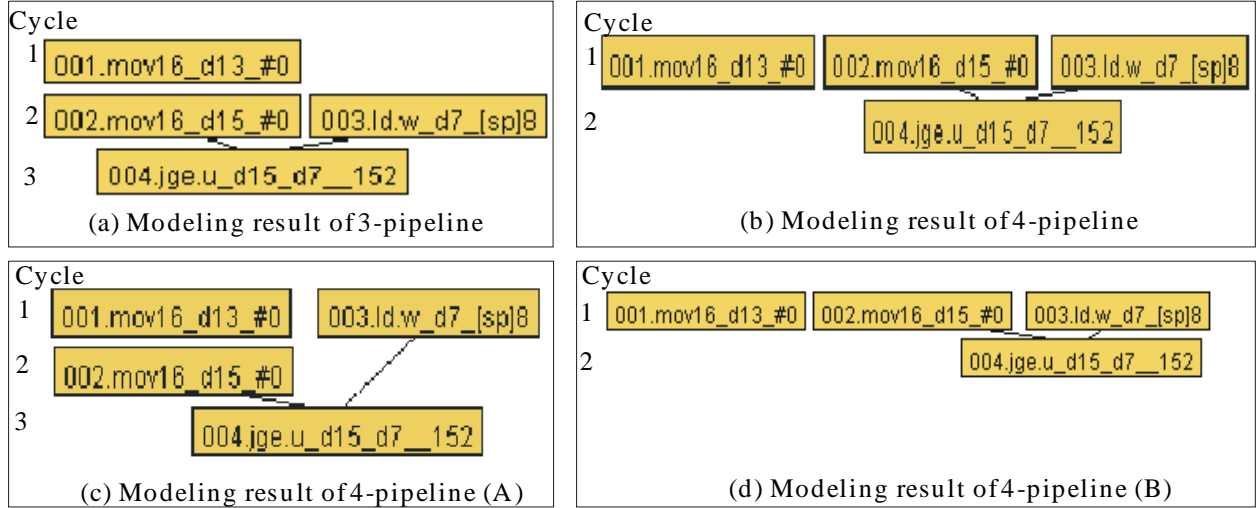


Figure 2. Graphs representing the issuing results of the block ‘120_001’ of the EEMBC benchmark ‘aiffr01’ for four different microarchitectures

In our research, the ‘Reconfigurable block code simulation’ of Stage 3 in Figure 1 can generate a graph that represents the instruction issuing results of the basic block under each microarchitecture. As an example, Figure 2 includes four subfigures with a graph representing the instruction issuing results of the block ‘120_001’ of the EEMBC benchmark ‘aiffr01’ for one specific microarchitecture. In each subfigure, each node of the graph represents an instruction, in which the first three digits represent its position in the original program sequence, followed by opcode and operands. The opcode followed by ‘16’ indicates a 16 bit instruction. Otherwise it is a 32 bit instruction. Each arc connects two instruction nodes with a true data dependency [1]. The instruction node at the vertically low end of the arc is a child instruction and has the true data dependency on the parent instruction at the high end of the arc.

In Figure 2, each graph in the subfigure also represents the relative timing of the instruction issuing. Instruction nodes aligned at the same horizontal level are issued on the same cycle and the issuing cycle time is ordered at the left side of the graph. Subfigures 2.c and 2.d show that the 4-pipeline (A) and 4-pipeline (B) microarchitectures take three and two cycles to execute the block ‘120_001’, respectively. This is because the partial IP pipeline in the 4-pipeline (B) microarchitecture can execute more instruction than that in the 4-pipeline (A).

Subfigure 2.b and 2.d show that both the 4-pipeline and 4-pipeline (B) microarchitectures take two cycles. In this case, the added partial IP pipeline of the 4-pipeline (B) microarchitecture can perform the same as the duplicated full IP pipeline of the 4-pipeline microarchitecture.

3.3. Overall performance results and discussions

Figure 3 displays the performance of all microarchitectures in the overall *WAM of IPC*, and the speedup of the enhanced microarchitectures with the empty square and solid diamond notations, respectively. The speedup is represented as percentage to one decimal place. We explore the design space of the partial IP pipeline by comparing the performance improvements with that achieved by the duplicated full IP pipeline.

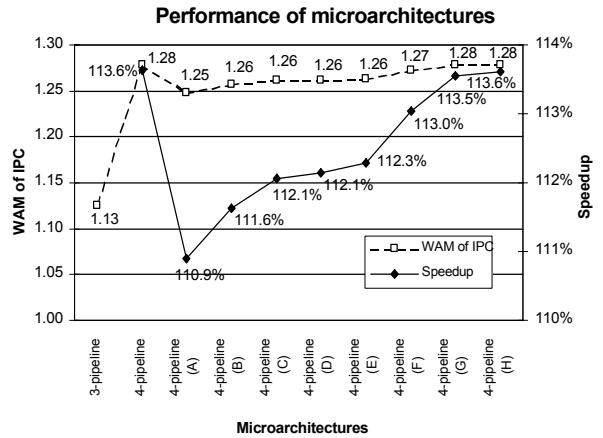


Figure 3. The overall performance results of all microarchitectures in our study

Figure 3 shows that the 4-pipeline (H) and the 4-pipeline microarchitectures achieve the same speedup of 113.6% after the rounding up. This indicates that adding a partial pipeline could achieve the performance improvement similar to duplicating a full pipeline.

Figure 3 also shows that the 4-pipeline (D) microarchitecture, with the best of the four partial IP

pipelines without implementing the multiplication and duplication operations, achieves 112.1% speedup. It achieves only 1.5% less speedup than the 4-pipeline (H) microarchitecture. When the hardware implementation cost of the multiplication and division operations in the partial IP pipeline of the 4-pipeline (H) microarchitecture are taken into account, the 4-pipeline (D) microarchitecture could also be a design choice under a stringent cost budget.

Under a more stringent cost budget, the 4-pipeline (A) version, implementing three instructions in the partial IP pipeline, could also be a design choice. It achieves only 1.2% less speedup than the 4-pipeline (D) Microarchitecture, while it implements seventeen instructions less. One reason of the small speedup differences between those two microarchitectures could be that the instructions executable in the partial IP pipeline of the 4-pipeline (A) and (D) microarchitectures respectively achieve 16.25% and 19.95% execution probabilities.

In our study, one set of EEMBC benchmark assembly code is used for the simulation of all microarchitectures. It was compiled with some optimisations specifically targeted at the performance of the 3-pipeline base microarchitecture, which is out of our control. We were unable to optimise the code for the performance of the standard 4-pipeline and the special 4-pipeline microarchitectures. When the set of the assembly code could receive similar optimisations for those enhanced microarchitectures, they could potentially achieve better performance results than reported in the paper. However it does not affect our research methods as presented.

4. Conclusion and future works

In this paper, we have presented a customized approach, in three stages, of designing an additional partial IP pipeline and adding it to the base embedded superscalar microprocessor to improve its performance. We developed the 3-pipeline base microprocessor model based on TriCore 2.0 32 bit MCU/DSP core. Eight partial IP pipelines were designed and added respectively to the base microarchitecture to form eight special 4-pipeline microarchitectures. The standard 4-pipeline microarchitecture was constructed by duplicating the full IP pipeline of the base microarchitecture. We used the basic block sampling and simulation approach to simulate the performance of those microarchitectures.

Our experimental results show that adding the extra partial IP pipeline to the base architecture can achieve cost effective performance improvements, close to that of duplicating the full IP pipeline. Performance results of the partial IP pipeline also pose more choices of the cost/performance tradeoffs compared with conventional pipeline duplication for the general purpose

microprocessor. Our approach, as presented, can be generally applied to embedded systems and general purpose microprocessors as well.

This paper can conclude that it is possible to add the partial pipeline to the base microprocessor for the SoC embedded system to achieve better cost effective performance improvements. Based on different levels of the cost constraints and silicon area vs. performance tradeoffs, different partial pipelines can be added.

As part of future work, it would be interesting to study the relative performance improvements of enhanced microarchitectures with the real TriCore 2.0 instruction latency. Future work could also focus on the choice of the subset of IP instructions for the partial IP pipeline, by using the different compiler to generate the benchmark suite assembly code, so the instruction subset of the partial IP pipelines as presented could be confirmed.

References

- [1] J. L. Hennessy and D. A. Patterson, *Computer Architecture A Quantitative Approach*, 3rd ed. San Francisco, California, USA: Morgan Kaufmann Publishers, Inc., 2003.
- [2] J. A. Fisher, "Customized instruction-sets for embedded processors," *Proceedings of the 36th Design Automation Conference(DAC 99)* New Orleans, Louisiana USA, pp.253–257, June 21-25 1999.
- [3] P. Koopman, "Embedded system design issues the rest of story," *Proceedings of the International conference on Computer Sesign, ICCD 96*, October 1996.
- [4] P. Hoogenboom, "Tasking helps siemens with 32-bit tricore™ architecture design," *Embedded Systems Programming Europ*, November 1997.
- [5] the Infineon Technologies (UK) Limited, "Tricore architecture," [Online]. Available: <http://www.infineon.com/tricore>, March 2005.
- [6] the Embedded Microprocessor Benchmark Consortium, "Eembc benchmark," [Online]. Available: <http://www.eembc.org/Benchmark/automotive.asp>.
- [7] T. Cuatto, C. Passerone, L. Lavagno, A. Jrreckska, A. Damiano, C. Sansoe, and A. Sangiovanni-Vincentelli, "A case study in embedded system design: an engine control unit," *DAC 1998*, vol. 6, June 1998.
- [8] V. E. Malyshev, Ed., *Parallel Computing Technologies, 6th International Conference, PaCT 2001*, Novosibirsk, Russia, September 3-7, 2001, *Proceedings*, ser. *Lecture Notes in Computer Science*, vol. 2127. Springer, 2001.
- [9] T. M. Conte, M. A. Hirsch, and K. N. Menezes, "Reducing state loss for effective trace sampling of superscalar processors," *Proceedings of the International conference on Computer Sesign, ICCD 96*, October 1996.
- [10] C. Bailey and H. Shi, "Analysing available instruction level parallelism in stack based code : A preliminary approach," *Proceedings of PREP*, April 2003.
- [11] L. K. John, "More on finding a single number to indicate overall performance of a benchmark suite," *ACM Computer Architecture News*, vol. 32, no. 1, March 2004.

Transcript from the discussion

Comment; references to speedup of 112% etc should really read as 12% speedup, looking at definition for speedup (3)

Q(AS): any other architectures using this already?

A: I do not know, not sure why, but I suspect that one reason is.. that we assume one cycle per instruction.

Q(AS): What is the effect of adding pipeline to the control unit, does it affect chip dye? It seems that with the partial pipeline it takes more logic to schedule the instructions, and the trend has been that control units have grown.

A: We have not looked into this yet. But compared with the control unit needed in dies with an additional pipeline, the partial pipeline should save some space. Because it needs to consider less instructions.

Q(ES): Did you do some real measurements on how good the simulation fits to reality?

A: We did some, but the company involved could not disclose certain results.

After this is developed with real instruction latencies, then we can compare our results.

Q (AS): Concerning page 3, first column: "The memory and branch characteristics of the TriCore 2.0 are not modeled, which does not affect the generality.." : With this in mind, what kind of penalty is expected when branches occur, in this extra partial pipeline version.

A: We did not consider the branch prediction penalties.

But performance could be worse, because more instructions could be in pipeline, compared to not having an extra pipeline.

Q (AS): Do you have access to any Cycle accurate simulator?

A: Yes, at infineon I had access to this.

Q (Ian): Explain the speedup, upper graph (Fig 3)

A: The curve represents the absolute performance of each architecture.

It shows the WAMofIPC of enhanced architectures compared vs the baseline 3-pipeline architecture.

ES: How relevant are benchmarks to other applications, e.g. DECOS, with respect to i.e. WCET?

A: The EEMBC automotive/industrial embedded microprocessor benchmark suite has several benchmarks that are special to the application, e.g. speed of wheels in a car.

Not sure about application of other benchmarks to DECOS application.

Q: What about dependencies in dye used for particular instructions? I.e. by taking away an instruction will it affect other instructions?

A: I have no quantitative answer to that.

The PISA Architecture: A Viable Platform for the Superscalar Execution of Statically Scheduled Stack Code

Soyeb Alli Chris Bailey
Department of Computer Science, University of York,
Heslington, York, YO10 5DD, UK.
soyeb.alli@cs.york.ac.uk

August 18, 2005

Abstract

Stack processors offer a level of code density unrivalled by that of register file-based architectures. This asset can be attributed directly to the fact that operands are addressed implicitly in stack programs. However, this implicit addressing also enforces a serial execution model in the program and the full impact of this constraint is only realised when an attempt is made at building superscalar implementations of a stack processor. This paper introduces a new architectural paradigm that tries to alleviate the impact of this constraint whilst retaining much of the code compactness of stack processors.

1 Introduction

Superscalar processors use dynamic scheduling to increase throughput by attempting to uncover independent instructions for execution every clock cycle. This out-of-order scheduling of instructions enables the processor to hide latencies of long running operations (e.g. multiply, load etc) and/or utilise functional units more efficiently[1]. The number of instructions considered by the processor for out-of-order issue is known as the scheduling window and larger windows increase the likelihood of the processor succeeding in finding independent instructions. However, increasing the size of the window also increases the hardware requirements of the processor.

Every instruction in the window compares its own

operand fields against those of its predecessors in order to uncover any potential hazards. For example, if an instruction j shares the same destination field as a preceding instruction i , then issuing j ahead of i would introduce a write-after-write(WAW) or write-after-read(WAR) hazard. The number of comparators required for this purpose can be calculated using the following formula:

$$2(n-1) + 2(n-2) + 2(n-3) + \dots + 4 + 2 = n^2 - n$$

In reality, the number of comparators required would not be as high as this because the processor only considers a subset of the scheduling window in any given cycle. However, the size of the scheduling window may have a direct impact on the cycle time (if the instruction scheduler lies in the critical path)[2]. Then deciding upon the size of the can become a trade-off between reducing cycle time and reducing CPI. Furthermore, the complexity of the scheduling logic makes it accountable for significant percentage of a processor's power budget[3].

Compilers can help alleviate the detrimental effects of reducing window size by scheduling code statically; compile time scheduling effectively extends the look-ahead capability of the processor. Unfortunately, stack code is not amenable to scheduling at compile time (due to the manner by which operands are addressed implicitly). However, the architectural techniques about to be introduced will improve the scheduling potential of stack code by exploiting certain properties intrinsic to stack ma-

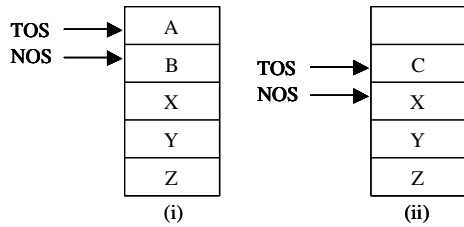


Figure 1: The state of memory (i) before and (ii) after executing the operation $C = A + B$

chines. It will be shown that these techniques can enable the compiler to schedule stack code in much the same way that register file-based code is scheduled. Furthermore, these techniques will permit optimisations that are not feasible in register file-based code.

2 Preliminaries

All operations are performed on the stack in a stack processor. The top two elements of the stack comprise the source operands¹ and the result is written back to the top of the stack, by default. All reads are destructive so the result of an operation would overwrite one of its own source operands (see figure 1). Sometimes stack operands need to be rearranged so that they appear in the correct order. This can be achieved by executing swap and/or rotate instructions [4].

3 Partially implicit stack addressing

In the canonical stack model, all instructions consume the top two operands on the stack. However, this is a rather superficial observation about the properties of the stack. In reality, the source operands of an instruction are the operands residing at the instruction's destination address and the cell above it. For example, if the destination address of a particular instruction's result is `0X00FC3E08` then the first and second source operands can be found at addresses `0X00FC3E04`

¹These entries are commonly referred to as the top-of-stack (TOS) and next-on-stack (NOS) entries.

and `0X00FC3E08`, respectively. Thus, by specifying the destination address alone, the processor has enough information to be able to execute the instruction.

Specifying the destination address in this way enables the compiler to reorder instructions without changing program semantics (c.f. RISC). However, specifying a 32 bit address within every instruction is impractical as it will significantly increase code size. Instead, it is better to specify the destination address as an offset relative to some pointer; the stack pointer is a suitable candidate for this purpose. If the stack pointer points to the first free cell then the processor can calculate the destination address by adding the offset to the current value of the stack pointer. Figure 2 illustrates this.

The processor is required to keep track of the value of the stack pointer after every instruction execution. Furthermore, the offset should be a 2's complement number because the stack can shrink as well as grow. Thus, a 5 bit address field can provide an offset ranging from -16 to +15. Any processor supporting this form of addressing will be said to implement the partially implicit stack addressing (PISA) architectural paradigm. The PISA paradigm does not rule out the possibility of supporting a dynamic instruction scheduler. Instead, the two techniques could be used in concert to reduce the complexity of the scheduler. Alternatively, the architecture could be used in combination with a VLIW approach.

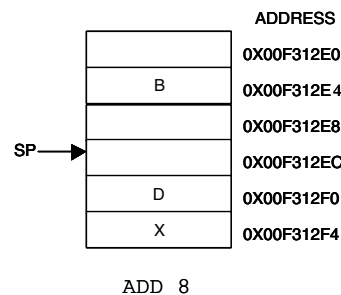


Figure 2: The ADD instruction will write its result to address ($0X00F312EC + 8 =$) $0X00F312F4$ and consume D and X during execution.

REGISTER-RENAMING TABLE			REGISTER-RENAMING TABLE			REGISTER-RENAMING TABLE		
ADDRESS	MAPPING		ADDRESS	MAPPING		ADDRESS	MAPPING	
0X00124804	PR27		0X00124804	PR27		0X00124804	PR27	
0X00124808	PR25		0X00124808	PR25		0X00124808	PR25	
0X0012480C	PR12		0X0012480C	PR12	PR8	0X0012480C	PR8	
(i)			(ii)			(iii)		

Figure 3: (i) DIV writes its result to address 0x00124802 so the register-renaming table queues the register mapping in the corresponding entry (ii) ADD writes its result to the same address so its mapping is queued at the same entry (iii) The first instruction to try to read the operand at this address will be redirected to the physical register holding the result of DIV (i.e. PR12) before this mapping is deleted from the renaming table. The next instruction to try to read the operand at this address will be re-directed to the physical register holding the result of the ADD (i.e. PR8).

Partially implicit addressing alone only permits overlapping the execution of instructions that write their results to different locations on the stack. This can be prohibitive as stack locations are often reused. However, the next section introduces a technique that circumvents this impediment.

4 Further enhancing static scheduling

Consider executing the following code sequence on a register file-based machine:

```
DIV R1, R2, R5
ADD R6, R4, R1
SUB R1, R12, R13
```

The compiler would like to schedule the SUB ahead of the ADD in order to hide some of the latency of the DIV instruction. However, if the compiler tries to do this then the ADD will get the wrong value for R1 (i.e. a RAW hazard will result). This is because the result of the SUB instruction will overwrite the result of the DIV instruction. However, operands cannot get overwritten in a stack machine but instead, are destroyed upon being read. The compiler can improve static

scheduling by exploiting this property.

In order for the compiler to be able to issue the SUB ahead of the ADD, the processor must provide some hardware mechanism that will enable the result of the DIV instruction to persist after the ADD has been executed. A register renaming table constructed from a set of first in, first out (FIFO) structures will perform this function. Figure 3 illustrates the operation of this mechanism.

The size of the FIFOs will dictate the degree to which the compiler can reorder instructions. Note that this mechanism is not invisible to the compiler and, therefore, this size cannot be reduced in future implementations of a particular instruction set architecture if backward compatibility is an issue.

5 Evaluation

The PISA architecture greatly enhances the ability of the compiler to schedule instructions over the canonical stack model. It also permits optimisations that are not feasible in register-file based machines. However, PISA-based code will be greater in size than stack-based code. This may result in poorer cache performance, though this performance should still be better than in a RISC machine.

The task of compiling code for the PISA architecture is more arduous than in the case of either the RISC or canonical stack machines. In order to compile code for a PISA machine, the compiler must first reduce the code into a format suitable for execution on a stack processor. Subsequently, the compiler must compute the destination addresses of each and every operation before finally scheduling any delays. This is clearly unsuitable in cases where fast compile time is a requirement.

In reality, the performance of the PISA architecture depends greatly on the amount of parallelism *statically* available in the program being executed. For programs with frequent branches, this figure can be quite small. However, for programs with infrequent branches, the amount of parallelism available can be quite significant (conditional branches constitute only 1% of the instruction mix in the EEMBC consumer benchmark for the TM32 CPU[5]). When coupled with the code density of stack machines, the performance of the PISA architecture would be unparalleled in such cases.

6 Conclusion

The PISA architecture enhances the ability of the compiler to schedule code statically and, consequently, reduces hardware requirements. However, this improvement comes at the expense of increased compilation time and reduced cache performance. A future study would be able to evaluate the performance of the PISA architecture in relation to the stack and RISC architectures supporting both dynamic scheduling and VLIW approaches.

7 Acknowledgements

This work was supported in part by a grant provided by EPSRC, Doctoral Training Account (DTA).

References

[1] R. Tomasulo. An efficient algorithm for exploiting multiple arithmetic units. *IBM Journal of Research and Development*, 11(1):25–33, January 1967.

- [2] S. Palacharla, N.P. Jouppi, and J.E. Smith. Compelxty-Effetive Superscalar Processors. In *Proceedings of the 24th International Symposium on Computer Architecture*, pages 206–218, June 1997.
- [3] R. Gonzalez and M. Horowitz. Energy dissipation in general purpose microprocessors. *IEEE Journal of Solid-State Circuits*, 31(9):1277–1284, September 1996.
- [4] P. Koopman. *Stack Computers: The New Wave*. Mountain View Press, 1989.
- [5] J.L. Hennessy and D.A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kauffman Publishers, third edition, 2003.

Transcript from the discussion

Q (ES): what is new?

A: stack architecture

Ø. Teig comments that a similar architecture has been used in transputers stack machine for pascal?

AS: knowledge of any machine

know of some machine by Jack(?) Moore

AS: could this work be used to also improve the java machine?

yes, think so..

SCTL: A StateChart Transformation Language for Test Sets Reduction

Nicolas Guelfi, Benoît Ries
University of Luxembourg
6, Rue Coudenhove-Kalergi
L-1359 Luxembourg-Kirchberg, Luxembourg

Abstract

Specification and testing activities are key phases in embedded systems development life cycles. In specification-based testing approaches, test cases are solely generated from the system specification. Test cases are often too numerous to be executed exhaustively. In practice, test engineers often select test cases based on informal approximations. We aim at improving these activities by formulating abstraction hypotheses on system specifications, to reduce generated test sets. Our general application framework is the automotive industry, and we aim the specific domain of small-sized real-time embedded systems that must be highly reliable. The main result presented in this paper is a model transformation language that helps test engineers to select test cases based on system specifications.

1. Introduction

Nowadays, embedded systems are everywhere, from washing machines to cars, trains, and planes, etc. The approach in this paper aims small-size embedded systems that have a restricted computation capacity as well as storage capacity, typically, electronic control units (ECU) present in cars. For instance, responsible for airbag deployment, car breaking (ABS), etc.

System specification activities (among other activities of the system development life cycle) are human activities. As such they are error-prone, that is why verification activities are necessary in the development process. Verification activities are classically divided in two categories, dynamic verification (test), and static verification (analysis). In our study, we are interested in the dynamic verification of systems based on their specifications; commonly called *specification-based testing* [Richardson89] (or conformance testing). During this phase, system's specifications are used as a reference to verify the implemented system, and test cases are exclusively derived from the specification of the system

under development. For many industrials, which subcontract system implementation activities to external companies, these tests are the only kind of tests that can be accomplished. Thus are of particular importance. One of the advantages resulting from specification-based testing is that test cases are created earlier in the development process, thus will be ready for execution **before** the end of the system's implementation. Moreover, when test cases are generated earlier, test engineers may find inconsistencies and ambiguity in the specifications, either by automatic analysis, or by informal inspection of the generated test cases, which allow improving the system's specification before it is finished.

Several modeling languages may be used to specify behavior of reactive systems [Harel85], we choose the statecharts' language [Harel87], because it has been formally defined [Harel96, Damm98], and is well spread in the industry (STATEMATE is a lodestar-tool of systems' behavior specifications in the automobile industry). In this paper, we focus on the behavioral specification of the embedded software.

Testing is a critical activity of systems development life cycle. As such, it must be performed confidently. Selection and generation of test cases are crucial activities of specification-based testing. Thus, it is important to have methodologies allowing systematic derivation of test cases from specification models. When performing tests based on specifications, generated test cases are frequently too numerous to be completely executed often due to a lack of time. Consequently it is needed to select what test cases to be performed to keep a reasonable test execution time. Practically, test cases selection is usually performed based on informal approximations.

In this paper, we define a model transformation language that reduces systems specifications. This simplification step is meant to be combined with automatic test case generation techniques [Gnesi04] in order to, reduce the number of test cases generated, and perform the activity of test cases selection in a systematic

way. The main advantages of our approach are (a) the specification of the system is the main artifact to decide for further test activities; (b) reduced specifications are specifications of the tests effectively performed; (c) reduced specifications describe a subset of the system’s functionalities to be tested.

In Section 2, we present our approach to the reduction of statecharts in the context of specification-based testing activities. We present the syntax and semantics of our statechart model in Section 3. We define the transformation language that allows characterizing abstraction rules on statecharts in Section 4. Section 5 gives a short description of our prototype implementation of our transformation language, which is integrated in the MagicDraw UML CASE tool.

2. Approach

The approach that we suggest is based on model transformation languages [Heckel00]. It offers applying abstraction rules on systems specifications in order to generate reduced specifications. These reduced specifications are used as input for test cases generation techniques, resulting in a smaller amount of test cases (i.e. a reduced test set) to perform.

Concretely, our approach will be composed of a model and a model transformation language. The model is based on statecharts and used for the specification of small embedded systems. The model transformation language is used for the reduction of specifications (written with the aforementioned model). The syntax and semantics of our model of specifications and our specifications transformation language being formally defined, our approach permits the verification of specification reductions, both on the syntax and semantics.

Our approach is illustrated by Figure 1 which shows a reduction (illustrated by $reduce(sc1)$) of a system’s specification written with RTSL statecharts, named $sc1$, in order to generate ($specTestGen(sc2)$) a reduced test set, named $ts2$, from reduced statecharts $sc2$. The semantics of the statechart before transformation is $sem(sc1)$ and after transformation is $sem(sc2)$. One of the important issues to address in our approach is to know to what extent there is an equivalence between the different paths of the figure going from $sc1$ to $ts2$. For instance, how are the two following paths $specTestGen(reduce(sc1))$ and $reduce(specTestGen(sc1))$ linked? Are they similar, if yes to what extent? Another question that we are asking ourselves is whether the reduction of the semantics of the statechart $sc1(reduce(sem(sc1)))$ is coherent with the semantics of reduced statechart $sc2(sem(reduce(sc1)))$? In this paper, we will not address these issues, but we think that it is fundamental to keep in mind the impact of the reductions of

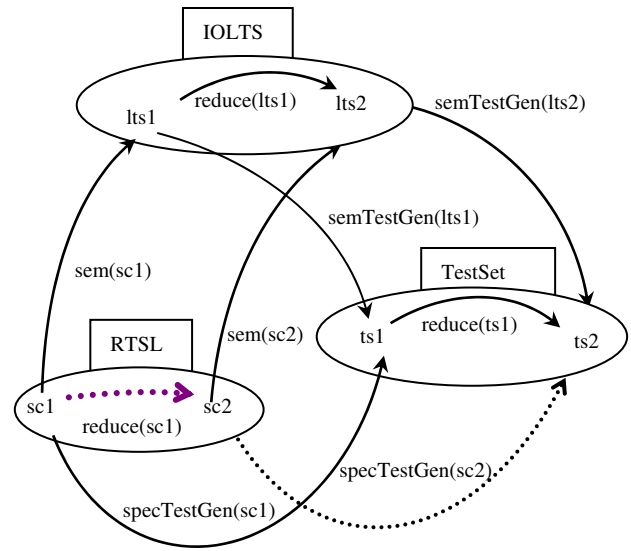


Figure 2 - Overview: Specification, Semantics and Test

specifications with respect to their semantics in order to define properly the model transformation language. In this paper, we focus on the reduction of statecharts (arrow $reduce(sc1)$) and we also tackle the semantics links between RTSL statecharts and IOLTS. For test case generation and test execution, we refer the reader to existing approaches, as for instance, Gnesi et al. [Gnesi04], or Tretmans [Tretmans99].

The major contribution of this paper is the definition of a statechart transformation language based on abstraction rules. These transformations aim at reducing systems’ specifications, so that the generated test cases from these reduced specifications are also reduced.

Numerous reduction approaches exist, but few are specifically oriented to visual models, most of them targeting textual specifications. Based on the study of Prenninger et al. [Prenninger04] we suggest a categorization of abstractions, applicable to the reduction of tests based on specifications that contain 5 categories:

- *Functional abstraction*: the aim of a functional abstraction is to focus on a precise system's functionality to verify. The application of this kind of abstraction lies on the fact that the system under test is not going to behave entirely the same way, but will only preserve the significant chosen aspects.
- *Structural abstraction*: the abstraction of syntactical elements of the specification models.
- *Data abstraction*: the idea behind data abstraction is to link concrete data types to abstract data types. For instance, to reduce the size of the representation or to reduce the data complexity.
- *Communication abstractions* aim to abstract complex communications into less complex communication. For example, a complex communication may be an interaction with a number of messages exchanged that

could be abstracted in a unique message starting and ending at the same location.

- *Temporal abstractions* are based on the idea that only the sequence of events/actions is pertinent; i.e. the precise times when events appear are not taken into account. These temporal abstractions may be used to modify the granularity of time. For example, a system may have clock cycles of 10ms, but we could wish to abstract time, in the way that clock cycles will remain unchanged but we will take snapshots of the system every second.

3. Prototype of RTSL: a Real-Time Statechart Language

Due to the proliferation of statechart variants [Beeck94], it is necessary to choose which statechart variant to use (Harel [Harel87], STATEMATE [Harel96], UML [OMG04], or other variants), and possibly adapt it to our specific needs. Thus, one of the results of our study is the definition of a model of statecharts, which we name RTSL, which fits to the further definition of our statecharts transformation language.

Most of the existing variants of statecharts give in details their semantics, but few linger on the precise description of the abstract syntax and the related syntactic properties. In our work, the precise description of the abstract syntax of RTSL statechart is paramount, as we describe the semantics of our transformation language at the level of the abstract syntax of RTSL statecharts.

In this paper, we only focus on basic concepts of statecharts that we think are fundamental to the definition of our transformation language (states, transitions, events, actions, hierarchy, and concurrence), and we give only a prototype of the **Real-Time Statechart Language**, which does not support time-related concepts, because we think that the basic concepts cited above are sufficient, in a first step, to establish a proof of concept of our approach, i.e. these concepts are sufficient to define a first version of our transformation language.

Equation 3 - Syntactical States Accessibility Relation of RTSL Statecharts

$$\begin{aligned}
(a) \quad & \forall s. (s \in S_i \implies s \xrightarrow{\epsilon}_i s) \\
(b) \quad & \forall (s, s', e, a). ((s, e, a, s') \in T_i \implies s \xrightarrow{(e,a)}_i s') \\
(c) \quad & \forall (s, s', s'', e, a, \gamma). (\gamma \in (E_i \times A_i)^* \wedge s \xrightarrow{\gamma}_i s' \wedge (s', e, a, s'') \in T_i \implies s \xrightarrow{\gamma \cdot (e,a)}_i s'') \\
(d) \quad & \forall (s, s'). (s \in S_i \wedge s \neq root_i \implies \exists s'. (s \in sub_i(s') \wedge s \xrightarrow{\epsilon}_i s')) \\
(e) \quad & \forall s. (s \in S_i \wedge type_i(s) = OR \implies s \xrightarrow{\epsilon}_i default_i(s)) \\
(f) \quad & \forall s. (s \in S_i \wedge type_i(s) = AND \implies \forall s'. (s' \in sub_i(s) \wedge s \xrightarrow{\epsilon}_i default_i(s')))
\end{aligned}$$

The description of the abstract syntax of statecharts, as given by Jansen [Jansen02], is adapted to our needs; we complete it by describing the informal properties of Jansen with logical formula, and we restrict this syntax by not taking into account guards on transitions.

The semantics of RTSL statecharts is given as Gnesi et al. [Gnesi04], i.e. we give RTSL statecharts semantics as input/output labeled transitions systems (IOLTS). To achieve it, we provide an algorithm [Ries05] that transforms RTSL statecharts into IOLTS.

Our model of statecharts RTSL is constituted of the set of statecharts SC compliant with the formalization, we give in the following, an extract of the formalization.

A statechart i , noted sc_i , is a 7-uplet

$$sc_i = (S_i, E_i, A_i, T_i, sub_i, default_i, type_i)$$

Where S_i is a finite non-empty set of states, containing at least the root state, noted $root_i$. E_i and A_i , are finite sets of events and actions characterized by their names.

$T_i \subseteq S_i \times (E_i \cup \mathcal{E}) \times (A_i \cup \mathcal{A}) \times S_i$ is the transition relation. Informally, for a transition $(s, e, a, s') \in T_i$: s is the source state, e is the event that enables the transition, a is the action performed when the transition is taken, and s' is the target state of the transition.

The refinement function $sub_i \subseteq S_i \rightarrow P(S_i)$ associates to a state its direct substates, it is restricted not to contain cycles, and that each state owns a unique direct super-state (forming a hierarchy).

Function $type_i: S_i \rightarrow \{OR, AND, BASIC\}$ associates to each state its type. Function $default_i: S_i \dashrightarrow S_i$ is a partial function which associates to every OR-state exactly one of its direct substates, which is its default state.

We define an accessibility relation between two states at the **syntax level**, which will be of great interest for the formalization of our transformation language. It allows us to ignore all inaccessible states from the root state. It is important to differentiate this accessibility relation at the syntax level from accessibility relations defined at the semantics level.

In this paper, we define the accessibility by simple deduction of the order and hierarchy of syntactical elements of statecharts. This relation is formally described by assertion of Equation 3.

Let $\xrightarrow{i}: S_i \times (E_i \times A_i)^* \times S_i$ be an accessibility relation between two states through a finite sequence of events/actions couples, formally defined by the 6 assertions below. We note $s_0 \xrightarrow{\gamma} s_1$ meaning that state s_1 may be reached from state s_0 with the sequence of events/actions $\gamma \in (E_i \times A_i)^*$. Intuitively, this relation represents two accessible states with a series of transitions and/or hierarchical links.

4. SCTL: a StateChart Transformation Language

The main contribution of this paper is the definition of SCTL. SCTL is a language that offers transformations of RTSL statecharts based on abstraction rules presented shortly in Section 2. The following section only presents in details a few transformations of SCTL; a technical report [Ries05] gives its complete definition.

These transformations aim at reducing the specifications written with RTSL statecharts. The transformations are to be used by test engineers who chose the abstractions they want to apply on the system specifications and then use the reduced specifications for test case generation that will result in a reduced amount of test to perform.

Our approach is in line with the numerous work on model transformations [Heckel00, Avgeriou04], initiated by the recent MDA standard from OMG [OMG03] and, more generally, by the existing model-based methods. Results on model transformations come mainly from the domain of graph theory [Heckel00].

SCTL offers transformations based on structural and functional abstractions rules. As variables are not taken in account in the abstract syntax of RTSL, our SCTL does

not offer any transformation based on data abstractions. Communication and temporal abstractions rules are also not taken into account for the current definition of SCTL. SCTL provides the 8 following transformations:

1. Transformation `delUnreach` deletes all inaccessible states from the root state. To preserve the coherence of the statechart, it also deletes transitions whose source state or target state has been deleted. It may be useful to apply `delUnreach` at the end of every transformation or sequence of transformations.
2. Transformation `exState` extracts a given state of a statechart, i.e. after transformation, only the behavior of the state, including its internal behavior is kept. This transformation and the following `abState` take advantage of the hierarchical structure of statecharts.
3. Transformation `abState` abstracts the behavior of a given state of a statechart, i.e. after transformation, internal behaviors of the state will be ignored.
4. Transformation `delState` deletes a state of a statechart.
5. Transformation `modifDefault` modifies a default state of a statechart.
6. Transformation `delTrans` deletes a transition of a statechart.
7. Transformation `startWith` allows specifying which state will be part of the initial states of the statechart.
8. Transformation `ignoreEvt` ignores a given event of the statechart.

SCTL semantics We define an axiomatic semantics for SCTL, thus the definition of each transformation is given by an axiom. Equation 4.1 is the assertion that defines the `abState` transformation whose purpose is to abstract a given state of a statechart. The transformation can be applied to any state except the root state. All states are kept except for all the substates of the state to abstract.

Equation 4.1 – Axiom for `abState` Transformation

$$\begin{aligned}
& \forall (sc_i, sc_j, s). (sc_i \in SC \wedge sc_j \in SC \wedge s \in S_i \wedge s \neq root_i) \\
\implies & ((sc_i, sc_j, s) \in abState \Leftrightarrow \\
& S_j = \{s' \mid s' \in S_i \wedge s' \notin sub_i^+(s)\} \\
& \wedge T_j = \{t' \mid t' \in T_i \wedge src_i(t') \notin sub_i^+(s) \wedge tgt_i(t') \notin sub_i^+(s)\} \\
& \quad \cup \{(s', e, a, s) \mid \exists s''. ((s', e, a, s'') \in T_i \wedge s' \in S_j \wedge s'' \in sub_i^+(s))\} \\
& \quad \cup \{(s, e, a, s'') \mid \exists s'. ((s', e, a, s'') \in T_i \wedge s' \in sub_i^+(s) \wedge s'' \in S_j)\} \\
& \wedge sub_j = \{(s', stSet') \mid s' \in S_j \wedge sub_i(s') = stSet' \wedge s' \neq s\} \cup \{(s, \emptyset)\} \\
& \wedge type_j = \{(s', ty') \mid s' \in S_j \wedge type_i(s') = ty' \wedge s' \neq s\} \cup \{(s, BASIC)\} \\
& \wedge default_j = \{(s', s'') \mid s' \in S_j \wedge s' \neq s \wedge default_i(s') = s''\})
\end{aligned}$$

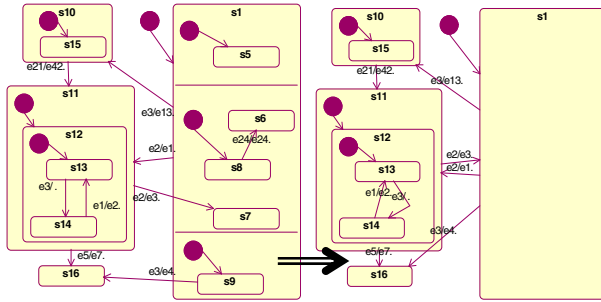


Figure 4.2 - Transformation abState

All transitions are kept except for the strictly internal transitions of the state to abstract, i.e. whose source and target state are both substates of the state to abstract. The transitions, whose either source state or target state is a substate of the state to abstract, are modified so that the source state (resp. target state) is replaced by the state to abstract. The state to abstract does not have any substate anymore, and thus becomes a basic state.

Figure 4.2 illustrates the application of an *abState* transformation on state s_1 of the left statechart; result of it is the right statechart. The substates of state s_1 are deleted, which are the states of the following set: $\{s_5, s_6, s_7, s_8, s_9\}$, as well as the three direct substates of state s_1 . The internal transition $(s_8, e_{24}, e_{24}, s_6)$ is deleted. As well as the transition arriving (resp. leaving) to (resp. from) a substate of state s_1 : (s_{11}, e_2, e_3, s_7) (resp. (s_9, e_3, e_4, s_{16})) now arrives (resp. leaves) directly to (resp. from) state s_1 .

Limitations It is important to notice that the resulting RTSL statechart of this transformation may contain behaviors that were not behaviors of the statechart before transformations. On Figure 4.3, for example, we can establish that the following sequence of transitions $\langle (s_0, e_1, \epsilon, s_1) (s_1, e_3, \epsilon, s_2) (s_2, e_1, \epsilon, s_3) \rangle$ is not a valid sequence of transition of the initial statechart sc_3 . This transformation may thus introduce additional traces (undesirable). It is important to be aware of consequences on the semantics of statecharts traces due to transformations. In order to test only original behaviors, we should delete the undesirable behaviors. This can be done in two different ways:

- By asking the user to delete the test case corresponding to the undesirable traces.
 - By automatic analysis of the generated test cases.
- A solution not to introduce undesirable tests would be to avoid using inter-level transitions, whose source and target states do not have the same direct superstate.

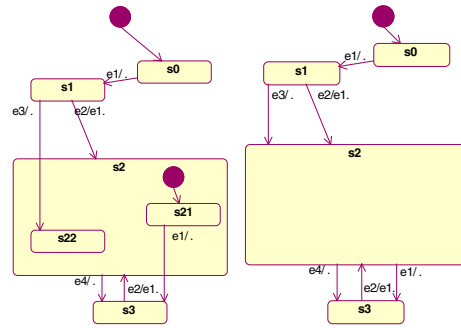


Figure 4.3 - Transformation abState (limitations)

5. Prototype

SCTT stands for **StateChart Transformation Tool**. The goal of SCTT is to provide a concrete way of applying SCTL transformations. In the context of this paper, the implementation of a prototype of SCTT aims at providing a proof of concept of our approach. Figure 5.1 shows the prototype of SCTT integrated as a menu in the UML MagicDraw modeling tool.

The design of the prototype of SCTT has been facilitated by the precise formalization of the concepts of RTSL and SCTL. Therefore, the design activity of SCTT has been straightforward. Each SCTL transformation is implemented as an object, and a RTSL statechart is an object composed of 7 attributes (states, events, actions, transitions, substates, types, defaults) corresponding to its abstract syntax elements (S, E, A, T, sub, default, type).

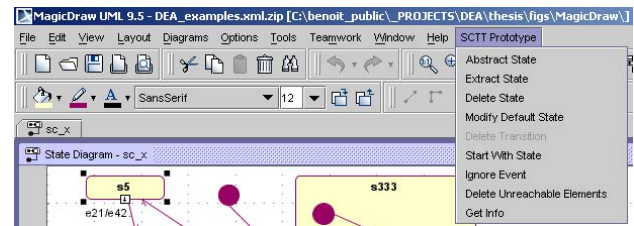


Figure 5.1 – Screenshot of SCTT in MagicDraw

The prototype of SCTT is implemented in Java as a plugin for the UML modeling tool MagicDraw v9.5. The prototype consists of 13 Java classes, for an average total of 2000 lines of code (including comments and diverse declarations), and has been developed with the Eclipse 3.0 development platform. SCTT prototype offers the following features:

- Transform a statechart modeled in MagicDraw with one of the 8 SCTL transformations.
- Print the description of the abstract syntax of the active state diagram, with the help of the *Get Info* command

The small-sized embedded system chosen as a case study for this paper is a system that allows deactivating the front passenger's airbag of cars depending on the presence of a front passenger, and the kind of passenger. This case

study is directly inspired from a real project from the automotive industry, and is voluntarily simplified in order to concentrate on the interesting parts for our statecharts transformation language. This embedded system aims at warning the general electronic module (GEM) of cars when the front passenger is a child seat (a child seated in a child seat), a child (not seated in a child seat), or an adult. The front passenger is the person seated next to the driver. The system does not care about the deployment of the driver's airbag. The system must also warn the GEM when the front passenger's head is too close to the airbag. Thanks to a 3D camera, this system classifies the passenger into different categories and measures the distance between the passenger's head and the airbag. This system may be used in two different use modes, the service mode in which it performs passenger classification and head detection, or the production mode in which the system sends diagnostics information.

Figure 5.2 shows the statechart that models the behavior of this small-sized embedded system that has been modeled with the UML tool MagicDraw, and that consists of 20 states (of which 12 basic states, 1 AND-state, 7 OR-states), 15 transitions, 12 events and 2 actions. Transformations realized in the context of this work have all been performed with our prototype SCTT.

The applications of SCTL reductions help targeting the tests to be performed. Let's make the assumption that at a certain moment of the test phase, test engineers want to focus on the test of the service mode of the system. The SCTL *exState* transformation can be applied on the *serviceMode* state. The application of the transformation will result in a consistent statechart composed of the *serviceMode* state with a root state. Let's take the four following traces which are part of the possible system executions before application of the aforementioned transformation:

```
<(contactKeyInserted, ε) (enterProdMod, ε) (sendPassengerStatus, ε)>
<(contactKeyInserted, ε) (enterProdMod, ε) (sendDiagnostic, ε)>
<(contactKeyInserted, ε) (childSeatDetected, switchLightOff) >
<(contactKeyInserted, ε) (passengerNotinOFF, switchLightOff) >
```

After application of the transformation this subset of possible execution is reduced to:

```
<(childSeatDetected, switchLightOff) >
<(passengerNotinOFF, switchLightOff) >
```

Another example of applications of SCTL transformations may be that engineers just want to test the service mode but also want to test the switching on of the car contact. In this case they can apply *delState* on the state *productionMode*. Or they could also apply *ignoreEvt* on the action *enterProdMod*, followed by an application of *delUnreach* in order to delete unreachable state, i.e. state *productionMode*. In this precise case, the resulting statecharts of these two applications are identical.

We also applied SCTL transformations with this case study to discover potential specifications errors. For

instance, by applying the SCTL transformation *startWith* on state *productionMode* followed by an application of the *delUnreach* transformation. We can notice, by looking at the resulting statechart that when the system is in production mode, then it cannot switch back to service mode anymore, which is potentially an undesirable behavior of the system. SCTT allowed us to exhibit this behavior that is probably a specification error.

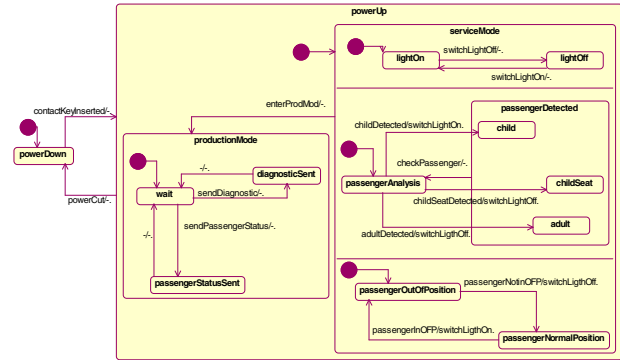


Figure 5.2 – Case Study

6. Conclusion

The main contribution of this paper is the definition of the SCTL model transformation language allowing the abstraction of specifications expressed with RTSL statecharts in order to simplify them. The advantages of this language are: (a) abstract statecharts represent simplifications of statecharts. These simplifications are mainly structural. Our approach does not aim to guarantee that all behaviors are preserved after transformation; (b) the formal description of SCTL has helped us to precisely automate its transformations.

We have defined a formalization of Harel's statecharts, that we have called RTSL, in order to create a model transformation language for RTSL statecharts. This formalization contains some restrictions, among others, transitions have one source state, one target state, at most one event, and at most one action. These simplifications allow us, while keeping fundamental concepts of statecharts (states, transitions, hierarchy, and concurrence) to define the basis of our transformation language: SCTL.

SCTL defines 8 transformations belonging to two categories of transformations (structural and functional). Semantics of these transformations is expressed at the level of the abstract syntax of RTSL statecharts. SCTL transformations described in chapter 4 are the following ones: (1) abstraction, (2) extraction, (3) deletion of a state, (4) modification of a default state, (5) deletion of a transition, (6) modification of the initial states of a RTSL statechart, (7) deletion of an event in the statechart, and (8) deletion of inaccessible states.

SCTT, a prototype implemented in the MagicDraw UML modeling tool has been realized to establish a proof of concept of our approach and particularly our transformation language SCTL. All the illustrations of transformations of this paper have been performed with this prototype tool.

Several perspectives can be considered.

- Definition of the semantics of SCTL transformations at the level of the semantics of RTSL statecharts. This will allow us to validate SCTL transformations with respect to the semantics of our statechart. This validation will help estimating more precisely which behaviors are lost during the transformation.
- Take into account real-time concepts: extension of the syntax of RTSL, proposition of temporal abstraction rules specific to embedded systems.
- Extension of the supported UML concepts by RTSL: addition of variables, guards, and actions on variables at the level of transitions of RTSL statecharts. This will allow us to define additional SCTL transformations based, for instance, on data abstractions.

References

- [Avgeriou04] P. Avgeriou, N. Guelfi, and G. Perrouin. Evolution through architectural reconciliation. In Workshop on Software Evolution through Transformations: Model-based vs. Implementation-level Solutions, SETra 2004.
- [Beeck94] M. von der Beeck. A comparison of statecharts variants. In Formal Techniques in Real-Time and Fault-Tolerant Systems, pages 128-148, 1994.
- [Damm98] W. Damm, B. Josko, H. Hungar, and A. Pnueli. A compositional real-time semantics of STATEMATE designs. In COMPOS'97: Revised Lectures from the International Symposium on Compositionality: The Significant Difference, pages 186-238. Springer-Verlag, 1998.
- [Gnesi04] S. Gnesi, D. Latella, and M. Massink. Formal test-case generation for UML statecharts. In ICECCS, pages 75-84, 2004.
- [Harel85] D. Harel and A. Pnueli. On the development of reactive systems. Logics and models of concurrent systems, pages 477-498, 1985.
- [Harel87] D. Harel. Statecharts: A visual formalism for complex systems. Science of Comp. Prog., 8(3):231-274, 1987.
- [Harel96] D. Harel and A. Naamad. The STATEMATE semantics of statecharts. ACM Trans. Softw. Eng. Methodol., 5(4):293-333, 1996.
- [Heckel00] R. Heckel and G. Engels. Graph transformation and visual modeling techniques. Bulletin of the European Association for Theoretical Computer Science, (72), 2000.
- [Jansen02] D. N. Jansen, H. Hermanns, and J.-P. Katoen. A probabilistic extension of UML statecharts. In FTRTFT, pages 355-374, 2002.
- [OMG03] Mda guide v1.0.1. Tech. report, OMG, 2003.
- [OMG04] UML 2.0 superstructure adopted specification. Technical report, OMG, 2004.
- [Prenninger04] W. Prenninger and A. Pretschner. Abstractions for model-based testing. In M. Pezze, editor, TACoS'04: Test and Analysis of Component-based Systems, 2004.
- [Richardson89] D. Richardson, O. O'Malley, and C. Tittle. Approaches to specification-based testing. In TAV3: Proceedings of the ACM SIGSOFT '89 third symposium on Software testing, analysis, and verification, pages 86-96. ACM Press, 1989.
- [Ries05] B. Ries. On the characterization of abstraction rules for the automatic transformation of small-sized embedded systems specifications. SE2C technical report TR-SE2C-05-05, University of Luxembourg, 2005.
- [Tretmans99] J. Tretmans. Testing concurrent systems: A formal approach. In CONCUR '99: Proceedings of the 10th International Conference on Concurrency Theory, pages 46-65, London, UK, 1999. Springer-Verlag.

Transcript from the discussion

Q (ES): is there some control or transformation to make sure which properties are lost in the abstraction?

A: For now, we can not guarantee what information is lost, which properties.

It may be a problem in some applications to make time abstractions, this is especially important in e.g. real-time systems.

Q (Teig): how much of added code is transformations, and how much is handling graphics

A: integration into tools is quite well done, so no need to do graphics.

Q: Have you thought about using existing tools/frameworks for approximations?

A: That is interesting, we should look into it.

Q (Kone): Given a statechart model can the method be used for other areas, to find some abstraction.

A: The approach is aimed at a specific domain of systems with few states, could be used for anything which is a statechart, to be transformed in a specific domain.

Test method for embedded real-time systems

J.P.Bodeveix, R.Bouaziz and O.Koné

Université Paul Sabatier - IRIT

118, Route de Narbonne 31062 Toulouse - France

Email: {bodeveix,bouaziz,kone}@irit

Abstract—Testing is an ultimate phase of product life cycle to which particular attention is paid, namely when dependability is of great importance. This work is concerned with the checking of dependability requirements by means of tests experiments. Here we address real-time embedded systems for which validation encounters a complexity problem due to real-time requirements. In this paper we propose a partial technique aimed at avoiding this problem and illustrate it with an example.

I. INTRODUCTION

The interest for *embedded systems* in every day life is now well recognized. The ERCIM issue [6] is a good reference that highlights the importance of dependability issues for this type of systems. Formal validation plays a major role here since this activity contributes to the development of reliable systems. Model-based testing is one of the formal techniques used for the validation of software/hardware systems. It consists in applying a set of experiments, generated a priori from the formal model of the specification, to a system *implementation*, with the intention of finding and discovering errors. The set of experiments (also called test suites) should be of reasonable size, but exhaustive in the sense that for each faulty system a set of experiment generated is able to detect potential errors. To ensure this, a strategy of selection is required. In the case of ERTSs (*Embedded Real Time Systems*) the test selection problem is worsened because a huge number of time instances are relevant to test. To achieve a good selection, care must be taken to define when to deliver an input to the system under test and when to expect an output.

In our framework, systems are modeled with *Timed automata*[1]. This popular model has been widely used as a basis for the *verification* of timed systems (real-time model checking) [11], [7] ... For *tests* selection from timed automata, there are currently two main approaches in the literature. The first one consists in selecting tests patterns from the so-called region graph by Alur and Dill[1]. But the combinatory explosion involved in the computation of the region graph constitutes a limitation of this approach [5], [3], [10]. The second approach is based on the so-called *symbolic* computation which enables a smaller graph, compared to the region graph. The concerned authors use the symbolic techniques for testing a specific class of non deterministic specifications [8], [9]. But the determinisation is realized “on the fly”, during test generation and execution. This task is often complex and time consuming. It can disturb the tester while the latter must react quickly to the actions performed by the system under test. In our work we do not address determinisation. We consider

specifications that are deterministic, but we use symbolic techniques for the selection of specific purpose tests that correspond to some expected *dependability requirements*. We argue that the resulting approach is a pragmatic and efficient way of computing timed tests suites with low complexity, since it combines symbolic technique with the exploration of specific/subset part of the specification model.

The remainder of the paper is organized as follows. Section 2 describes the time based formalism we use for describing embedded systems. Section 3 presents the features of our test computation method. Section 4 concludes the paper.

II. MODEL FOR EMBEDDED REAL-TIME SYSTEMS

We consider the functional and abstract behavior of embedded system that may be for instance controlling some process while interacting with its environment through inputs and outputs. Further testing of such embedded system will concern only functional and behavioral requirements [2], [4]. Here, we assume that the embedded system model can be directly provided in terms of input/output transition system, or compiled/translated into such model from another one (*model translation* takes part of our research). We need to model real-time mechanisms and we use clocks for that. The TIOSM model (Timed Input Output State Machine) is a particular kind of timed automaton [1] where *inputs* and *outputs* are modeled in an explicit manner.

Definition 1: TIOSM.

A TIOSM is a tuple

$M = (S(M), L(M), C(M), s_0(M), T(M))$ where:

- $S(M)$ is a finite non-empty set of states (also named *locations*);
- $L(M)$ is a finite non-empty set of interactions;
- $C(M)$ is a finite set of clocks;
- $s_0(M)$ is the initial state in which all the clocks are initially reset to zero;
- $T(M)$ is a finite set of transitions.

A transition $t \in T(M)$ is a tuple $t = (s, \mu, D, Z, d)$ where s and d are the starting state and the destination state of t . μ represents the interaction executed during the transition. μ may be some input (denoted $?a$) or output ($!a$). Z is the set of clocks to be reset to 0. D is a set of real time constraints of the form $Ci \in [a_i, b_i]$ ($Ci \in C(M)$; $a_i, b_i \in \mathbb{R} \cup \{\infty\}$) (\mathbb{R} is the set of real numbers). The transition can be fired only if the conjunction of the time constraints in D is true.

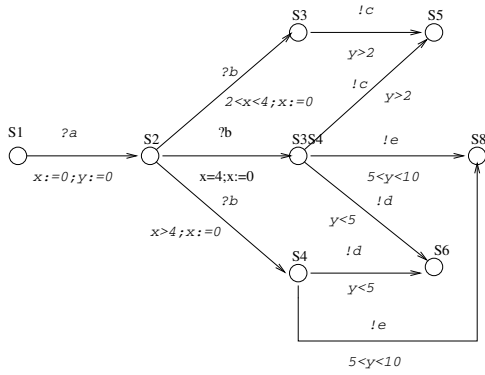


Fig. 1. (Deterministic) model of the embedded system

Example.: The example below is the description of a generic system that is embedded in a real-time environment. We model the system interactions with a set of labels $L = \{a, b, c, d, e\}$. Executions are constrained by timing requirements defined with clocks $C = \{x, y\}$. The formal model of our system is depicted with figure 1. The system is initialized with input a while the clocks x, y are reset to zero. After that, the behavior of the system will depend on the instant when it will receive input interaction b . Globally, if b is received early (between 2 and 4 time units), the system will perform output interaction c , at least 2 time units after a . If b could not be received before 4 time units, a warning notification action, let's say d must be sent, before 5. If this notification could not be computed on time, some alarm e must be triggered, but no later than 10. The reader may report to the model in figure 1, for the precise behavior of the embedded system.

III. TEST CASE DESIGN

Overview.: There are different test selection strategies that one could apply : random behavior testing, state checking etc. In our framework, we consider that a given test case must address a specific goal, related to a specific requirement. Therefore, our approach consists in computing one test case for one specific requirement (once a time). Technically, this enables us to restrict the computation to the subset part of the system model which is involved in the considered specific requirement. This strategy avoids to handle the whole specification model for computation, and thus reduces the computation complexity.

Dependability requirements are those requirements that are considered as particularly important (and *critical*) in the system design. We considered that system analysis must be focused on such requirements with a high priority.

A. Dependability requirement

A *Dependability requirement* characterizes some particular feature that one would like to check on the embedded system. The following is an example : “After input a is computed, the subsequent computation scheme of the embedded system *MUST* output b BEFORE some timeout occurs”. Such requirements can easily be formalized with temporal logics (with

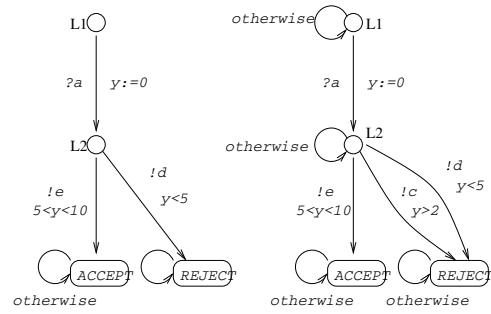


Fig. 2. Example of Dependability requirement

time) or with a TIOSM. Such TIOSM must be supplied with some means to check the requirement.

Definition 2: Dependability requirement.

A *Dependability requirement*, say D_R , is a deterministic and acyclic TIOSM with a distinguished non empty set of states. This set of states is denoted by $Accept(D_R)$ (resp. $Reject(D_R)$)

$Accept(D_R) \subset S(D_R)$ (resp. $Reject(D_R) \subset S(D_R)$).

$Accept(D_R)$ defines a set of *accepting states*. If the computation scheme reaches one of these states, then the requirement is exhausted. $Reject(D_R)$ defines a set of *rejecting states*. In the definition of the dependability requirement, these states are used to represent the paths or behavior that we would like to exclude from the search. For instance, in the example of figure 2, we would like to check the ability of the system to send alarm e , when this is desired. Here we do not consider experimenting the warning computation d . The requirement is modeled with a TIOSM which states (or locations) are $\{L1, L2, ACCEPT, REJECT\}$. In the figure 2, the left part is a standard version (while the right part shows how one could express default behavior with additional transitions). *otherwise* is a short representation of all alternatives.

The test case computation algorithm has two main inputs: The embedded system TIOSM model, denoted by $Spec$ and a given dependability requirement D_R . The dependability requirement model is a kind of observer with real-time features which keeps track of the walk through $Spec$, until the expected test case is effectively checked (i.e. an accepting state of the dependability requirement is reached). The analysis is restricted to the subset of $Spec$ checked by the observer. In automata theory, this can be formalized as a kind of synchronous product of $Spec$ and D_R . Here in our case, a transition is fireable in the synchronous product, if it is fireable in $Spec$, or it is fireable in both $Spec$ and D_R .

B. Partial computation

Let $Spec$ be some embedded system model (or specification), and D_R be a given dependability requirement. Both $Spec$ and D_R are formalised with TIOSM as we defined before. The partial computation of D_R within $Spec$ is characterized by a TIOSM SP obtained with the synchronous product of $Spec$ and D_R . The definition of SP follows.

Definition 3: SP.

- $L(SP) = L(Spec) \cup L(D_R)$;
- $C(SP) = C(Spec) \cup C(D_R)$;
- $S(SP) \subset S(Spec) \times S(D_R)$.

$S(SP)$ and $T(SP)$ are the smallest relations defined by the rules below:

- (Rule R_0):
 $s_0(SP) = (s_0(Spec), s_0(D_R)) \in S(SP)$.

- (Rule R_1):
 A transition is fired in $Spec$, but not in D_R , which remains in the same state

$$\frac{(s_1, s_2) \in S(SP) \wedge (s_1, \mu, D_1, Z_1, s'_1) \in T(Spec)}{(s'_1, s_2) \in S(SP) \wedge ((s_1, s_2), \mu, D_1, Z_1, (s'_1, s_2)) \in T(SP)}$$

- (Rule R_2):
 A transition is fired in both $Spec$ and D_R . The two automata move to their next states.

$$\frac{(s_1, s_2) \in S(SP) \wedge (s_1, \mu, D_1, Z_1, s'_1) \in T(Spec) \wedge (s_2, \mu, D_2, Z_2, s'_2) \in T(D_R)}{(s'_1, s'_2) \in S(SP) \wedge ((s_1, s_2), \mu, D_1 \cup D_2, Z_1 \cup Z_2, (s'_1, s'_2)) \in T(SP)}$$

The previous rules formalise the skeleton of the behaviour to be tested. Remember that the tester interacts with the implementation under test, while observing its behaviour. Rule R_1 defines the general walk through $Spec$, while Rule R_2 defines when the firing of some expected transition in $Spec$ is checked in D_R .

Now, we extend the definition of *Accept* for the synchronous product: $Accept(SP) \subset S(Spec) \times Accept(D_R)$. The states of $Accept(SP)$ are elements from $S(SP)$ of the form (s_1, s_2) , where s_2 is an accepting state of D_R .

Symbolic test paths computation: Now, test case computation turns to the selection of *executable* patterns that lead to the *Accept* set of SP . Time semantic based analysis can usually be borrowed from the so-called *state class graph* or *region graph*. This behavior graph captures all possible executable traces of the system. The time executability constraints of the embedded system can easily be stated by a conjunction of linear inequalities relating the execution instants of the test case transitions that lead to the *Accept* set. These inequalities are *symbolic* representations of the time constraints of the system. The reader may report to the references [1], [8] for classical symbolic graph computation algorithms of timed automata. The feature that we have added in our framework is the partial (or local) exploration of the graph with a standard Depth First Search algorithm. Figure 3 attempts to illustrate the local exploration: The search is performed depth-wise, so that only a local branch of the graph is memorized, until the expected behaviour is found. For instance, assume that, according to a given dependability requirement, we are interested in checking the part of the behaviour where some interaction 'f' is followed by some interaction 'g' ($\xrightarrow{f, g}$). Then, in the figure 3, the bold part corresponds to the path searched. The thin parts are gave up during the search.

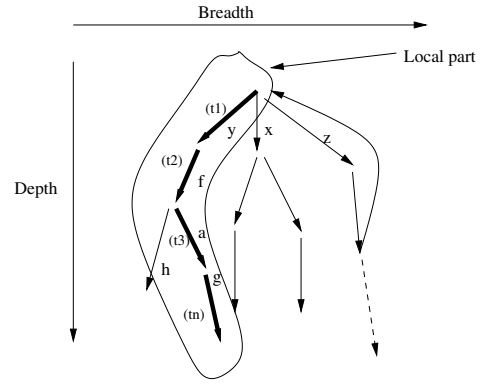


Fig. 3. Partial exploration approach

Our approach to the design test pattern implements this exploration technique. It is based on the joint computation of dependability requirement and specification model as it was formalised with definition 3. The symbolic test path search stops when an accepting state is reached. Consequently, we compute the expected test pattern while the whole symbolic graph need not to be explored.

Test system structure: The test graph computed from SP characterizes the static behaviour to be checked. It defines the subsequent structure of the tester. Since a test system interacts with some implementation under test (IUT), the design of the tester goes through inverting the inputs and outputs of the system model. An input of the system becomes an output of the tester (and vice versa). If the test system receives a wrong output, it produces a *fail* verdict (and enters a *fail* state). During a test campaign, three possible verdicts can be assigned according to the different observations:

Let μ be some action and τ be its execution instant.

- *FAIL* verdict: either the action μ or the time τ does not match the specification.
- *PASS* verdict: the action μ and the time τ match the specification and the dependability requirement.
- *INCONCLUSIVE* verdict: the action μ and the time τ match the specification, but they do not match the dependability requirement.

The figure 4 depicts the structure of the test system obtained from the specification model of figure 1 and the dependability requirement of figure 2.

To recap, our test design approach consists in the following steps :

- SP Computation of the product of the system model and the dependability requirement.
- Symbolic graph computation with a partial approach.
- Inverting of inputs and outputs of SP
- Add of verdict assignments and the corresponding transitions
- A test case is then an instance of the tester symbolic execution.

In our example, the symbolic constraints are straightforward since after initialisation, only clock x is reset to zero once (on

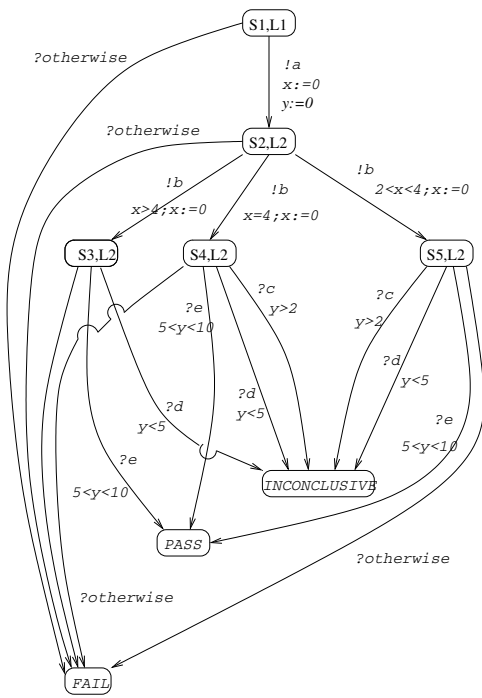


Fig. 4. The resulting test system structure

reception of *b*). All the execution patterns that lead to *PASS* correspond to successful test experiments showing that the expected dependability requirement is met.

Finally, in our work example, a successful test case (as expected from the dependability requirement), is for instance the following sequence:

$(!a; x := 0; y := 0) \rightarrow (!b, x = 4; x := 0) \rightarrow$
 $(?e, 5 < y < 10) \rightarrow PASS.$

An example of failure execution is :

$(!a; x := 0; y := 0) \rightarrow (!b, x > 4; x := 0) \rightarrow$
 $(?e, y = 10) \rightarrow FAIL.$

In our framework, the test design approach illustrated with the previous example can be fully automated (we are currently implementing prototypes for that) and repeated for each dependability requirement considered. The inputs of the method being one specification model and one dependability requirement modelled as shown in the previous paragraphs.

IV. CONCLUSION

Checking dependability requirements remains a development phase of high importance for embedded systems. Real-time features involves additional constraints on time management since a sound test system must compute when to send an output to implementation, and when to assign a given test verdict. The use of formal model enables the design of sound tests, but formal analysis of behaviour suffers from the complexity involved in the exploration of the behaviour graph. The test design approach presented here proposes a partial symbolic exploration which promises good results for the testing of complex embedded real-time systems.

Next step of the work is the full implementation of the approach presented. We need a testing tool as we are currently working on the experimentation of our work with industrial specifications from aerospace standards.

REFERENCES

- [1] R. Alur and D. timed Dill. A theory of timed automata. In *Theoretical Computer Science*, volume 126, pages 183–235, 1994.
- [2] B. Beizer. *Black-box testing: techniques for functional testing of software and systems*. Wiley, New York, 1995.
- [3] R. C Oliver and T. Glover, *A Practical and Complete Algorithm for Testing Real Time Systems* In *FTRFT1998 - Formal Techniques for Real-Time Fault Tolerant Systems*, Lygby, Danmark.1998
- [4] O.Koné. *Conformance testing to real-time communications systems*. Computer Communications, Volume 25. Elsevier Science, 2002.
- [5] A. En-Nouaary, R. Dssouli, F. Khendek, *Timed Wp-Method : Testing Real Time Systems*.*IEEE Transaction on Software Engineering*, November 2002.
- [6] ERCIM News N.52 . Special issue Embedded Systems. January 2003.
- [7] T. Hinzinger, X. Nicollin, J. Sifakis, and S. Yovine, *Symbolic Model Checking for Real-Time Systems*. *Information and Computation*. 111(2): 193-244, June1994.
- [8] K. Larsen, M. Mikucionis, and B. Nielsen, *Online Testing of Real-Time Systems, Formal Approaches To Testing of Software*, Link2, Austria. September 2004.
- [9] M. Krichen, and S. Tripakis, *Blac-Box Conformance Testing for Real-Time Systems*, In *SPIN 2004*. Spring-Verlag Heidelberg, 109-126.2004.
- [10] J. Springintveld, F. Vaandrager and P. D’Argenio, *Testing Timed Automata*, *Theoretical Computer Science*. 254, 2001.
- [11] K. Larsen and W. Yi, *Timed Abstracted Bisimulation: Implicite Specification and Decidability*. In *Proceedings Mathsatical Foundations of Programming Semantics (MFPS 9)*. April 2001.

Transcript from the discussion

Q: You’re specifying properties as an automata, find a witness and transform this to test case?
A: the model is also an automata, and the test case is the part of the model which meet
Q: Did you look at model-checking tools, and did they perform well enough?
A: Model-checking is different from testing, because you try to check the model by extensively checking states. But when testing you consider the system through interacting with it.
Thus a testing tool was developed. The input is the same, but when you have testing you have some non-deterministic properties..

From message queue to ready queue

Case study of a small, dependable synchronous blocking channels API
“Ship & forget rather than send & forget”

Øyvind Teig

Aurionica Fire and Security, Trondheim (A UTC Fire and Security company)

http://home.no.net/oyvteig

Abstract

This case study shows CSP style synchronous inter-process communication on top of a run-time system supporting SDL asynchronous messaging, in an embedded system. Unidirectional, blocking channels are supplied. Benefits are no runtime system message buffer overflow and "access control" from inside of a process of client processes in need of service. A pattern to avoid deadlocks is provided with an added asynchronous data-less channel. Even if still present here, the message buffer is obsolete, and a ready queue only could be asked for. An architecture built this way may be formally verified with the CSP process algebra.

1. Introduction

This "industrial" paper assumes that asynchronous interprocess communication is known by the reader. It describes a case study where the "opposite" – synchronous interprocess communication – was a viable solution, even for a small embedded system. A reader should hopefully be triggered to investigate further, as this short six page format implies.

The starting point for this now "work done" case was an in-house process/task non-preemptive run-time system written in C, compiled for an Atmel AVR processor, which contained 128 KB FLASH for program code and 32 KB external RAM. Several products built on this architecture had been successfully shipped. Messages were always asynchronous, meaning that any sender process would "send & forget" and go on.

But there were aspects where that design could be enhanced, like 1.) the system message buffer could in theory overflow, 2.) pointer movement between

processes (and possible race conditions) is difficult to handle and 3.) incoming messages could arrive in a process unprotected: regardless of its internal state.

However, we did close these cases by careful design and field trials.

Still, in another product, we decided to build a layer of synchronous, blocking and unidirectional channels on top of the asynchronous system. Having also shipped a product with this paradigm and implementation, with no new software release after a year's use (also thanks to stable functional requirements), we decided to continue and use it in a second product.

The concern here had initially been to select a dependable software pattern to avoid deadlocks. Interestingly, the selected pattern includes data-less asynchronous signal channels (later).

2. SDL and CSP

The two "competing" paradigms here are SDL ("the asynchronous") and CSP ("the synchronous" in this context). The edit-by-anyone *Wikipedia* dictionary [1] (also pointing to more academic sources) has entries of both:

SDL: "SDL (short for Specification and Description Language) is a specification language targeted at the unambiguous specification and description of the behaviour of reactive and distributed systems. It is defined by the ITU-T (Recommendation Z.100.) Originally focused on telecommunication systems, its current areas of application include process control and real-time applications in general."

CSP [2]: "'Communicating Sequential Processes' which was published in 1985. In May 2003, that book was the third-most cited computer science reference of all time according to CiteSeer (albeit a very unreliable source due to the nature of it sampling). ... As its name suggests, CSP allows us to describe systems as a

number of components (processes), which operate independently and communicate with each other solely over well-defined channels. CSP introduces a process algebra which is used to describe a process' communications with its environment."

Some languages influenced by CSP are occam [3] and Ada [4].

Synchronous systems may be built with asynchronous components, by adding some kind of handshake (like, waiting for a reply or building mechanisms into the run-time system). Likewise, asynchronous systems may be built with synchronous components, by adding some kind of overflow handling (like, overflow buffer processes.)

Most embedded system would probably need to use both paradigms. If we build with solely synchronous primitives, no input should allow the software or system to malfunction (therefore, have control on input, so that loosing data is a conscious action). If we build solely on asynchronous primitives, no inter-process communication should be allowed to crash the system (therefore, have control on it and introduce some kind of handshake or synchronism).

3. Blocking

When a process waits for a reply or access to other processes (below), it *blocks* "on the synchronous channel". Think of blocking as equal to what a calling function does when it waits for a called subroutine to return. There is no other thing to do than "wait".

Opposite, an asynchronous sending will *not* block.

Starting new processes from within a process may have "blocking fork/join semantics", depending on the operating system (occam blocks). In our system processes are only spawned from "main", before the scheduler is entered (the spawning in itself will not block, so that more than one process may be started).

With blocking semantics, *parallel slackness* becomes an issue – that we have enough processes to get things done, with a goal to handle all necessary I/O activity. Total work done should not be less.

Observe that passive waiting is indeed used when "delay" is wanted.

4. Access control of other processes

With blocking semantics, we will also be able to have others hang while waiting for *this* process. This could be busy processing or busy with another session. Then, in due course, we would open access to this from others, and process their messages, one in turn.

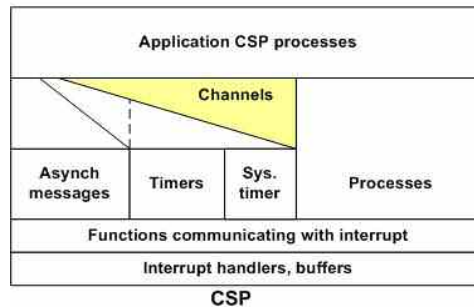
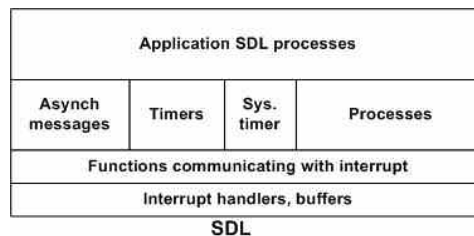
A side effect of this is that we may *choose whom we want to serve*, so that the server (this) could serve the clients (them) in a *fair* manner – or in the manner it chooses.

The term most often used for this is *selective waiting*. Both occam and Ada (and UML 2.0 [5]) have structures for it, and we have implemented it in the channel layer. It is called ALT for ALTERNative.

Another facet is that processes built with this scheme certainly need to obey the protocol semantics in interchange with other processes, but they do not need to know the semantics of the other processes' internal behaviour. As some times with asynchronous systems, *when*, and perhaps even *if* a message is put in a common message queue, need not be known. Not needing to know another party's semantics has been referred to as WYSIWYG semantics [6]. This also makes processes become dependable software components. It also shows the compositional semantics of CSP.

Observe that this is not the same as the *monitor* mechanism implemented in Java, where some sort of queuing of the blocking threads is done. However, one can build channels with monitors as building blocks in Java, .NET and Posix [7]. At least one major operating system bases interprocess communication on synchronous, blocking, rendezvous (later) type communication from bottom and up, and the vendor argue strongly about the safety perspective of this solution [8].

5. The layered architecture



The channel abstraction API resides on top of the SDL run-time system. Only one SDL function is not abstracted, the `_FSM_Init_`, which initialises a process. "SDL processes" may coexist with "CSP processes", but any communication between the two worlds would have to be done as asynchronous messages, with channels not used by the CSP process in charge during such a session.

6. The channels C API abstraction

All C code use of this API is by macros. A channel is a global data structure containing id of the *first* process, one half of the memcopy parameters, and states in the channel. For debug purposes we optionally insert intended sender and receiver identification.

```
#define CHAN_INIT_F
(CHAN,SENDER,RECEIVER,ALTTAKEN)
#define CHAN_IN_F
(CHAN,DATA,EVENT)
#define CHAN_IN_VARLEN_F
(CHAN,LEN,DATA,EVENT)
#define ALT_CHAN_IN_F
(GUARD,CHAN,DATA,EVENT,ALTTAKEN)
#define ALT_CHAN_IN_VARLEN_F
(GUARD,CHAN,LEN,DATA,EVENT,ALTTAKEN)
#define ALT_CHAN_IN_ASYNC_SIGNAL_F
(GUARD,CHAN,EVENT,ALTTAKEN)
#define CHAN_OUT_F
(CHAN,DATA,EVENT)
#define CHAN_OUT_VARLEN_F
(CHAN,LEN,DATA,EVENT)
#define CHAN_OUT_ASYNC_SIGNAL_F
(CHAN,RESCHEDULEME)
#define CHAN_IN_ASYNC_SIGNAL_F
(CHAN,EVENT)
#define ALT_TIMER_IN_F
(GUARD,TIME,UNIT,EVENT,ALTSTATE,ALTTAKEN)
#define FSM_RESCHEDULE_F
(EVENT)
```

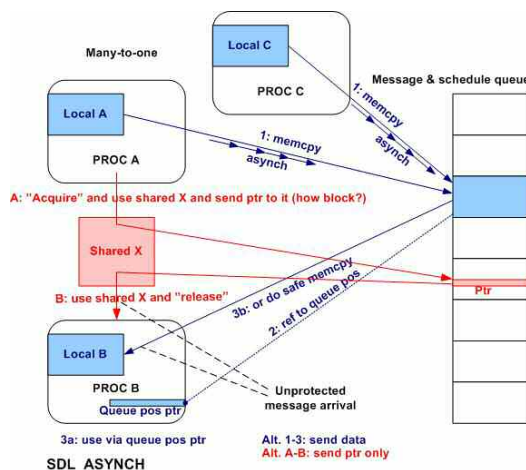
There are channel initialisation, inputs (for use in ALT constructs and not) and outputs (outputs do not know whether they are part of any ALT). An input may also have a timeout attached. And the local communicating state machines (processes) may need to want themselves to be rescheduled in some cases.

All these macros define state changes that are visible from the outside of a process.

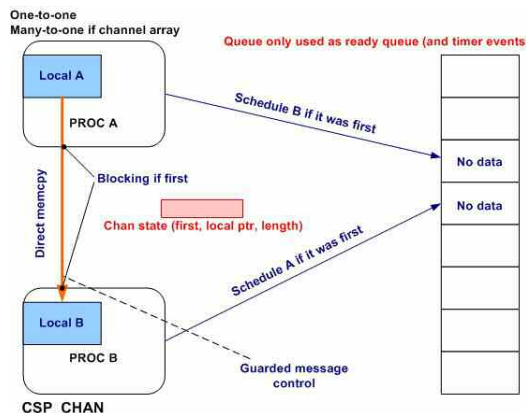
7. Semantics of asynchronous messages

A message is sent (*memcopy*'ed) into a message queue of individual elements, all with the size of the largest element. Proc A and B may proceed to send other messages immediately, before they are descheduled, i.e. they do not block. A receiver must handle the incoming message queue (and *memcopy* the data again if it wants to keep it) on a first come first

serve basis. Should any of the messages not confirm with some process inner state, it is discarded or set aside for later processing. The process becomes another scheduler above the run-time scheduler. If one needs to send larger chunks of data than one could afford to put into the message queue, a pointer to the message is sent instead. This requires access mechanisms to be built into the data segment, and some kind of feedback to the sender to inform that data has been read and is free (i.e. some synchronisation mechanism). In effect, one invents part of the channel concept anew every time. Also, with asynchronous messages, a (fast) producer and a (slow) consumer may be out of phase, and there is no mechanism to avoid queue overflow (..than to insert synchronosity). In most systems an overflowed interprocess message queue is detected by the run-time system, which then often have no other way out than to restart the system.



8. Semantics of synchronous channels



A *channel* is a named entity. You send or receive “on a channel”, not with a named process. It acts as a handle to a “protected” region of the code (or rather to a state-controlled protected phase), where the two interacting processes meet and do a *memcpy* from sender’s internal to receiver’s internal data structures. The *memcpy* is invisible in process code. In our case the channel is one-way. This meeting and *memcpy*’ing phase is often called a *rendezvous*, an Ada term. The part arriving first (sender or receiver) blocks and is descheduled until the second part (receiver or sender) arrives. Then *memcpy* and continuation of last part, and rescheduling of first.

Observe that an input *timeout* or simply a *wait* (for polling) is implemented as a channel with a timer on the sender side. The run-time system inserts the re-scheduling message when a timer has timed out. It also hinders timeouts to enter a process incorrectly if the optional channel(s) attached have already been taken.

Observe that we have not implemented timeout on sending. To implement this on on-chip interprocess communication with non-preemptive scheduling would have been straightforward, had we seen any need for it.

The *asynchronous* channel implemented here does not cause the message buffer to overflow if one rule is obeyed: That no new asynchronous signal is sent before contact with the receiver has been achieved.

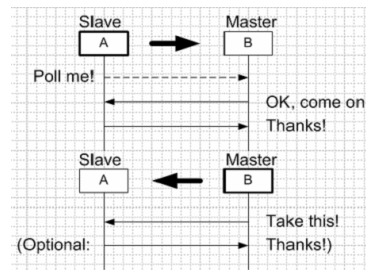
9. “From message queue to ready queue”

This paper’s title implies that the message queue is now *not* used for anything else than for sending dummy signals to a process, with the only purpose of scheduling it. So, we could have dropped the message queue for a process ready queue. But the asynchronous run-time system is well tested and we wanted to keep it.

Such a ready queue would also need to contain the cause of the rescheduling. The place this is needed is in an ALT, where we must know which clause that caused the rescheduling (i.e. channel number or timeout).

10. Deadlock avoidance

Where two processes spontaneously need to send data to each other, a blocking communication scheme might cause deadlock. This is a state where processes try to communicate – in a circular pattern – *blocking* for each other to become ready, and therefore unable to proceed. This is pathological and *must not* happen. All processes run at the same priority, so any priority inversion problem is ruled out in the system.



The pattern we chose to avoid this was to give the processes clear roles: slave and master. Master may send on the blocking channel any time (solid arrows) when it has something. Slave would never block on any spontaneous message, since the asynchronous “poll me” message (stippled arrow) lets the slave go on and not block, and then instead hang in an INPUT or ALT on the input channel. When the “OK, come on” message arrives from the master, the agreed upon protocol assures no deadly embrace.

11. Coding examples

(The text folds starting with “... “ in the code excerpt here contain code, or other folds.)

```
P_BS100_Sin (void)
{
  switch (ContextPtr->State)
  {
    ... ST_INITIALIZING_A (Initialize)
    ... ST_STATE_IN_ALT_030_032_104_A
    ... --> ST_STATE_IN_ALT_030_032_104_END_A
    ... ST_STATE_OUT_031_A
    ... ST_STATE_OUT_110_A
    ... ST_STATE_OUT_END_A
    ... ST_STATE_STOPPED_A, (crash)
    ... default, (crash)
  }
  ... Common action: get more to do from workpool
}
```

All process data is kept in a local “Context” in each process. It is allocated on the heap in the initialising phase. The scheduler schedules a process when it *calls* it. A process keeps track of its own state and does a switch/case on this state. It must run to completion on every scheduling, since there is non-preemptive scheduling. Therefore a process function return goes back to the scheduler.

This gives the nice side effect of one common stack for all processes.

Asynchronous i/o is pulled in/out by interrupts that use queues, which are polled by driver type processes.

```

case ST_STATE_IN_ALT_033_093_A:
{
  bool_a_alt_taken = FALSE;
  g_ALT_AL_Comp_Driver = CHAN_ALT_ENABLED_ON_A;

  ALT_CHAN_IN_F (ContextPtr->Guard_033_093, g_chan_033,
                ContextPtr->ALCP_0331_0931,
                S_EVENT_ALT_033_093_A, &alt_taken);
  ALT_CHAN_IN_F (ContextPtr->Guard_033_093, g_chan_093,
                ContextPtr->ALCP_0331_0931,
                S_EVENT_ALT_033_093_A, &alt_taken);
  ALT_TIMER_IN_F (ContextPtr->Guard_Timer_EMTH, MS_TIMEOUT_A,
                TU_MS_A, S_EVENT_TIMEOUT_A,
                &g_ALT_AL_Comp_Driver, &alt_taken);

  ContextPtr->State = ST_STATE_IN_ALT_033_093_120_END_A;
  break;
}

```

Above, is an example of an input ALT construct, where we see two channels with a timeout. A component of the ALT will be skipped if its "Guard_" becomes FALSE. This is the way to control client's access.

Not shown is the hand-coded test to verify that all guards are not FALSE (equal to the occam STOP, where a process cannot proceed). If so, it is a programming or design error, suited to "crash" – for further investigation and required program update.

Observe that there is no busy waiting by the process to facilitate waiting on a channel. This is because the second contender "pulls" the rescheduling of the first. This is the usual *monitor & condition variable* solution (also [7]).

12. Formal basis of the architecture

CSP, on which this scheme is based, has not been much discussed here, but it is possible to model and verify any system with this process algebra [9]. This would be out of reach (expensive), and not very interesting for us (small system and use of known software patterns). (Admittedly, this author knows CSP through occam and has had no hands-on experience with it.) However, other process algebras, like FSP, analysed with the free LTSA tool, may also be used [10] (it has been tried). Modelling asynchronous systems (albeit with finite size buffers, which makes them synchronous when buffers are empty/full) is also possible with Promela and the free SPIN tool [11].

The channel layer API discussed here was modelled on macros used by the code generator of the *SPoC* occam-to-C compiler [12]. SPoC also was a non-preemptive run to completion system, with appropriate scheduling queues and a timer queue (but no message queue). All the communication states and process start & stop that we would have to code by hand in the project was done automatically by SPoC, since occam supports channel input, output, ALT,

input timeout and wait, as well as compositional prioritised processes.

13. Discussion

The system adds approximately 2 KB of program memory. Execution time overhead (as compared to the asynchronous system) depends on whether the asynchronous system uses any synchronization to make it "safe" (this would make the systems about equal), and how the asynchronous system uses the received data. If it needs to keep the data past next descheduling, the asynchronous system needs *two* memory's.

Setting up a communication or an ALT obviously takes more cycles than just returning from the process, which is the asynchronous behaviour. But these cycles are only a small percentage of burnt cycles in our processes anyhow, so the overhead has been insignificant for our applications.

We would not have used this system had we had, say, 8 KB of code space (the SDL runtime system is about 20 KB, and we need that anyhow). But with 128 KB of code space (or, soon 256 KB – nice for an 8 bit machine) and 16 MHz clock, the added well-being of knowing that the system never overflows the message queue or sends unwanted messages into a processes, outweighs the overhead.

Using this methodology (with occam, SPoC and a C CSP library) has proven valuable for about 15 years, where (provided functional requirements are stable), "ship & forget" was more the rule than the exception. This was in embedded systems (then with Transputers and later DSPs and Intel 386ex machines) and on host Windows machines.

When the communication states have been set up, using them is straightforward: fill local structs in the context and set a state variable to send, or just set a state variable to input or wait.

But, there is more complexity to this system than I like to admit, also when it comes to personal engineers' preferences and background. If OO has had its way, the CSP kind of thinking certainly also has [13].

Grasping the communicating state machines, which are not in the application domain, but constitute the skeleton of the process/data-flow architecture, is individual. A channel most probably seems as belonging to OSI *network* (3) or *transport* (4) layer, and certainly not the application layer (7). Some programmers learn this methodology easily; let them handle it. Some resist or do not bother about these technicalities, let them concentrate on the product proper and *application* communication layer. The infrastructure

person(s) should then set up the necessary construct for the application people to just use.

However, when the communication infrastructure code once has been set up, it tends to stay stable and work.

Setting the size of the present ready queue is a matter of finding the maximum scheduling incidence volume. When this is found, even the producer-consumer problem will not cause further queue usage. To find this value we let the scheduler catch any overflow and then increase, with a margin. This is the same procedure as with a pure asynchronous system. However, when maximum has been found, there is no room for further surprises, since the value is a function of the number of channels and processes, not the communication pattern.

A future dream is to have (a subset of?) Ada available for microcontrollers of this type, or Java (where CSP libraries [14][15] are available). Or hope that result of ongoing occam research will hit industry some day [16]. In the meantime, we could use solutions as the one discussed here, which really is quite dependable, even if it is based on hand-written C.

14. References

Ref. [17] has been added since it is a good starting point for both theory and practice of this field of computer science. Use this list as hands-on and academic starting points, not especially for direct referencing of origins.

[1] *Wikipedia* at <http://en.wikipedia.org/wiki/SDL> at #Specification_and_Design_Language
CSP at #Communicating_sequential_processes

[2] C.A.R. Hoare, *Communicating Sequential Processes*, Prentice Hall, 1985

[3] *Wikipedia* [1] at #occam_programming_language

[4] *Wikipedia* [1] at #Ada_programming_language

[5] J. Rumbaugh, I. Jacobson and G. Booch, *The Unified Modeling Language Reference Manual, 2.ed.*, Addison Wesley, 2004

[6] P.H. Welch, University of Kent at Canterbury, UK in mail list, at <http://www.wotug.org/lists/occam/1142.txt>

[7] P.B. Hansen and C.A.R. Hoare, monitors, see *Wikipedia* [1] at #Monitor_%28synchronization%29

[8] Integrity from Green Hills Software Inc.

[9] The tool FDR2 by Formal Systems

[10] J. Magee and J. Kramer, *Concurrency, State models and Java programs*, Wiley, 1999.

Free tool at <http://www-dse.doc.ic.ac.uk/concurrency/>

[11] G.J. Holzmann, *The Spin Model Checker, Primer and reference manual*, Addison-Wesley, 2003

Free tool at <http://spinroot.com/spin/whatispin.html>

[12] M. Debbage, M. Hill, S. Wykes, D. Nicole, *Southampton's Portable Occam Compiler (SPOC)*, In: Miles, Chalmers (ed.), "Progress in Transputer and occam Research", IOS Press, Amsterdam, 1994 (WoTUG 17 proceedings), pp. 40-55.

Free tool at: <http://gales.ecs.soton.ac.uk/software/spoc/>

[13] Ø. Teig, *CSP: arriving at the CHANnel island - Industrial practitioner's diary: In search of a new fairway*, in "Communicating Process Architectures", P.H. Welch and A.W.P. Bakkers (Eds.), IOS Press, NL, 2000, Pages 251-262, at <http://home.no.net/oyvteig/pub>

[14] P.H. Welch and P.D.Austin. *Communicating Sequential Processes for Java (JCSP)*, 1999-

At <http://www.cs.kent.ac.uk/projects/ofa/jcsp/>

[15] Communicating Threads for Java (CTJ), G. Hilderink, J. Broenink, W. Vervoort, A. Bakkers *Communicating Java Threads*, in the Proceedings of the 20th World Occam and Transputer User Group Technical Meeting, pp. 48-76, ISBN 90 5199 336 6, IOS Press, The Netherlands. At <http://www.ce.utwente.nl/javapp/>

[16] P.H. Welch, F.R.M. Barnes (University of Kent at Canterbury), *occam-pi: blending the best of CSP and the pi-calculus*,

at <http://www.cs.kent.ac.uk/projects/ofa/kroc/>

[17] WoTUG home page: <http://www.wotug.org/>

Transcript from the discussion

Q(Shi):What do you think about future of ADA/Java in embedded systems?

A: Java used in e.g. mobile phones, ADA used in other areas, such as military.

But occam is pretty much a dead language..

In some newer architectures, with many processors communicating, there may arise some of the same problems as with occam and transputers.

Q(AS): This is used in e.g. smoke detectors, i.e. smoke detectors, i.e. systems with very high dependability requirements?

A: Yes, this will be a part of our next generation of smoke detectors, that we sell.

An Embedded Future for Distributed System Architectures

Trygve Lunheim, Amund Skavhaug
Department of Engineering Cybernetics, NTNU
trygvelu@itk.ntnu.no, skavhaug@itk.ntnu.no

Abstract

Recent advances in distributed system architectures may provide the solution to a number of challenges in future embedded systems. Middleware and grid technologies make it possible to utilize the processing power in subsystems, so that greater reliability through fault-tolerance can be achieved, as well as efficient use of resources through load-sharing.

In this paper we try to revisit and reassess some of the established truths and beliefs regarding the use of distributed operating systems and middleware in embedded systems.

1. Introduction

Embedded systems are found everywhere, and are becoming increasingly connected. For industrial systems, such as in a chemical plant or an oil rig, there may be from tens to tens of thousands of different processing units in operation, and these are becoming more powerful and multi-purpose. In homes, DVD players, gaming consoles and multimedia components are becoming more powerful than our personal computers, in terms of processing capability.

There is a trend towards connecting and networking these systems, and integration of services such as multimedia and entertainment/games along with data from embedded applications and internet services. These services mostly have different real-time and Quality of Service (QoS) requirements.

Although the applications for industrial systems and consumer products are very different, many of the requirements will often be similar. A video stream of bad quality because of lost frames might be as unacceptable for the consumer as it is to lose real-time data from a sensor in process control. Video and multimedia are also becoming commonplace in industry networks, e.g. for surveillance purposes. *Security* is obviously essential for the factory plant, and also in a home, especially when considering

wireless networks, which are becoming increasingly widespread.

Traditionally, the domain of distributed systems has been dominated by supercomputers and parallel computing. However, supercomputing ability is already becoming commonplace in homes and everywhere around us. Examples include the next generation in game consoles, Microsoft's XBox 360 and Sony's PlayStation 3, which are both equipped with multi-core processors, and support for multiple operating systems [1].

By using principles from distributed computing it is possible to achieve fault-tolerance, and thus increase the *reliability* of applications. Load-sharing is also possible, in order to make more efficient use of the available resources.

However, there is a need for new standardized interfaces for enabling this kind of distributed computing in embedded systems. The distributed architectures that are chosen need to be *scaleable* and *reliable* as well as extendable across different hardware and network protocols. There should also be very little manual effort involved to configure and maintain the system, thus reducing the operating cost. While in industry it is possible to rely on support personnel, this is not an option in most homes.

In part 2 and 3 of this paper we review some of the different architectures for distributed computing, and try to give some background for the development of these. In part 4 we revisit some established beliefs regarding the use of distributed architectures in embedded systems. Finally, in part 5 we will address some possibilities for the future of distributed real-time and embedded (DRE) systems.

2. Distributed operating systems

Distributed operating systems have been developed over the years to address different needs, such as parallel execution of processes, reliable transactions or real-time behavior.

2.1 General distributed operating systems

Traditionally, distributed operating systems have been developed for massively parallel computing, usually for large, homogenous systems, relying on some common, specialized hardware or software mechanism. The development has gone from distributed systems based on transactions and point-to-point message passing, e.g. using MPI [2], to support for distributed shared memory, which may be either uniform or non-uniform. SGI's NUMAflex architecture [3] is an example where special hardware is being used to support the latter.

Some initiatives in the development of large distributed operating systems in later years have been built with open source software, e.g. Beowulf Linux clusters [4].

One of the main characteristics of a distributed operating system is that it should just appear as one big system from the "outside", although the system consists of a number of elements. In other words, the distributed nature of the system should be *transparent* for the applications. In order to achieve this there is usually a large degree of homogeneity in these systems.

2.2. Distributed real-time operating systems

Real-time operating systems are typically niche products, with an emphasis on predictability, timeliness and reliability. Although a large variety of different real-time operating systems exist, the authors are not aware of many that are specifically designed to work as distributed operating systems.

One example, however, is QNX, which is a commercial real-time microkernel operating system. Support for message passing between distributed processes is transparently built into the QNX system core. This is achieved through the proprietary QNet [5] protocol, which runs on top of standard Ethernet, a serial line, or a TCP/IP connection.

However, this approach requires that all nodes run the QNX operating system, and this is often not possible, or even desirable. The trend is to use established and open standards for communication when this is needed, e.g. TCP/IP sockets, which are normally available, even if these are at a lower abstraction layer.

3. Middleware

Distributed applications may run on systems that are not explicitly distributed on the level of a

distributed operating system. The infrastructure that is used for "stitching together" such systems is referred to as *middleware*. Middleware can be seen as an abstraction layer between the application and the operating systems, network protocols and hardware that the distributed application(s) run on.

An overview of the different requirements and solutions for providing middleware can be found in [6]. The requirements for middleware include

- Network communication
- Coordination
- Reliability
- Scalability
- Heterogeneity

The solutions for middleware have gone from providing basic services, such as transactions and message passing, to more advanced models of distributed computing, with *object-oriented* or *component* middleware.

3.1 Middleware in use

In some niche areas the use of middleware standards, such as CORBA [7] and D/COM [8], has been successful. There are, however, some shortcomings to these standards.

CORBA is supported on many platforms, but for smaller, embedded systems it is probably too large and cumbersome. In the earlier versions of CORBA it was a problem that different vendors had their own implementations which were not standardized to be portable, but these problems seem to be resolved.

The CORBA 3.0 standard introduced the CORBA Component Model (CCM), which uses *containers* and *ports* to make it easier for application designers to create distributed objects that interact with each other within this framework. Some initiatives to support real-time and QoS services for CORBA are TAO and CIAO [TAO].

COM/OPC [9] has found some acceptance within industrial process control, where MS Windows has gained support in recent years. However, there is a lack of support for new services, such as handling Quality of Service for prioritized traffic. There is also a need to increase the reliability of the OPC services.

IndustrialIT [10] is an architecture specifically created for easing the integration and reuse of software components within distributed industrial systems. This framework, which is an ABB product, is object-oriented, client-server based, and built using established standards from Microsoft, i.e. ActiveX and COM/OPC.

3.2 GRID middleware

GRID [12] networks are commonly referred to as the future in distributed computing, as can be seen from the number of EU funded GRID research projects. Grid technologies are service-oriented, and provide loose coupling between distributed applications. The Open Grid Services Architecture (OGSA) builds on Web Services.

The Globus Toolkit [13] is the first initiative for building GRID infrastructure that can be used for general applications. The key components of Globus includes a Grid Security Infrastructure (GSI), providing security, a Monitoring and Directory Service (MDS), providing meta-information and broker functionality, and a Grid Resource Allocation Manager (GRAM), providing resource management in the Grid.

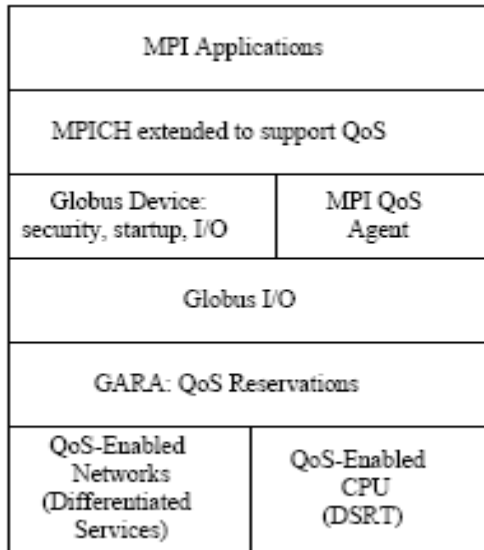


Fig 1. MPICH-GQ architecture

The MPICH-GQ API provided in GRAM is an extension of MPI which supports QoS reservations in networks and other resources.

GRID technology is still in its infancy, and there are many areas where further development must be made before it can be used in real-time and embedded systems. Support for end-to-end real-time performance is perhaps the most important. Energy and memory footprint requirements are other important aspects that need to be considered.

[This part will be expanded in a future version]

4. Use of distributed systems

Within the field of distributed computing there are some established truths and beliefs about the use of

distributed system technologies in embedded systems. In light of recent advances it may be time to revisit some of these.

4.1 Processing requirements

Embedded systems are often implemented with as little resources as possible, thus the application(s) need to have a small memory footprint as well as low processing requirements. Microcontrollers or computers with very limited resources are typically used, thus reducing cost of components as well as energy consumption during operation.

On the other hand, middleware technologies and distributed operating systems require extra processing and communication overhead, due to their general nature. In general, it has not been feasible, or at least considered prohibitively difficult, to use distributed architectures such as CORBA in most embedded systems.

However, in recent years the microcontrollers and industrial computers that are being used have become more powerful and versatile. Today a microcontroller may be equipped with a TCP/IP stack, have multi-threading capabilities, and fulfill all the requirements that are necessary for distributed computing.

4.2 Real-time requirements

There are often real-time requirements in embedded applications, especially within the industry, where the failure of a process to reach its deadline could lead to possibly catastrophic failures. Whereas the focus in distributed and parallel computing is on providing best effort service and maximum throughput, the focus in real-time systems is on determinism and predictability.

Middleware architectures have had a lack of focus on real-time requirements, and although there are exceptions, e.g. TAO [11], widespread use of these has not yet taken place in embedded systems.

There is an ongoing effort to improve the real-time properties and QoS support in middleware systems. This research is especially driven forward by the need to support multimedia streaming, e.g. used in video conferences and VoIP applications. Multimedia services are also important in industrial networks, and advances in this field will probably automatically be used in other applications as well.

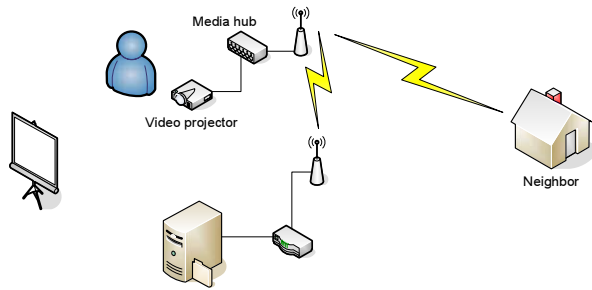


Fig 2. Distributed multimedia in the home

4.3 Security requirements

The importance of network security has become more evident in recent years. The proliferation of wireless network technologies has contributed to this, along with outside threats such as hackers and potential terrorists.

It is unacceptable for the average consumer to have the networked video player controlled by the neighbor, just as it is unacceptable to have unauthorized access to the industrial network of a plant. Thus, security should play a fundamental part in the distributed system, from the early design stages and throughout the development lifecycle.

In earlier middleware solutions security was often treated with less consideration, but in modern middleware platforms, e.g. CORBA and J2EE, this has changed.

The security *meta-model* specified in CORBA comprises several security models and techniques, while other platforms, such as J2EE or .Net can be seen as providing a subset of the CORBA security model, providing security through their environments and APIs.

Code-based access control gives permissions at the code level to access of resources, whereas role-based access control (RBAC) gives permission to a user to access resources based on the user's *role*.

There is a need for general and universal methods to provide authorization and authentication in all levels of the distributed architecture. Relying on different methods for authentication and security within the same system will most often lead to problems. To ensure security it is common to rely on a central Public Key Infrastructure (PKI).

Secure communication is often implemented using Secure Sockets Layer (SSL) and Transport Level Security (TSL).

The Grid Security Infrastructure in the Globus Toolkit supports message-based and transport-based (TLS) security. In the future it is likely that RBAC will be supported. RBAC is especially important for

enforcing security in large distributed systems, whereas for smaller systems it might not be necessary to enforce security from a user perspective.

5. The future for DRE systems

A problem with distributed real-time and embedded systems (DRE) is that these systems typically have a long life-span, and are built with a number of different, possibly proprietary technologies. As the systems evolve and new services are introduced, it becomes increasingly difficult to adapt and maintain these systems, using traditional software design.

For solving this problem a new software paradigm, model driven middleware [MDM], is being developed to help develop and integrate DRE systems. There is an ongoing effort to apply this way of thinking, which seems promising.

Furthermore, we would like to ask the question: Is there really a need to have a distinction between what we refer to as middleware and the internals of a distributed operating system? The distinction may be more related to perception than to the technical nature of the solutions. However, any such solution will need information about the distributed system to be present, i.e. meta information.

5.1 Meta information

This information is an important property of the distributed system. The information about the system could be centralized, and catalogue services used to look up who's where, doing what, and assign resources for applications.

Earlier, such services have been rather "heavy", both in terms of processing cost and labor in order to get them up and running. Today it is possible to implement these services, since we have enough computing power and resources available. There is need for a standard to establish these services universally.

In the Globus Toolkit this information is handled through the Monitoring and Directory Service (MDS). The implementation of the MDS in GT2 was based on the Lightweight Directory Access Protocol (LDAP), while XML is used in version 3 of the toolkit.

XML descriptors are also being used to describe QoS requirements within newer initiatives in middleware, e.g. the CoSMIC MDM toolsuite [14].

In general, XML provides a portable and extensible data format that is now widely accepted, and is commonly used for data representation, e.g. in MS Office. Therefore it seems like a natural choice for

expressing meta information in a distributed heterogeneous environment.

6. Conclusion

Embedded and real-time systems are becoming powerful enough, in terms of processing capacity, to support technologies for distributed architectures, which may enable fault-tolerance and load-sharing for applications.

However, for some of these technologies there are still shortcomings, especially with regard to real-time and QoS support. Security also needs to be handled consistently on all levels of the system, and a common platform should ideally be agreed upon.

The system needs all the necessary information to be available. Further research should work towards widely adopted standards, in order to fulfill the vision of cooperative distributed systems. Middleware or distributed operating systems lack the necessary capabilities for this to happen.

[This part will be expanded in future:
necessary system capabilities are not there (yet)
need for widespread standards for meta information
also: service location
]

7. References

- [1] "IBM, Sony, Sony Computer Entertainment Inc. and Toshiba unveil Cell processor", http://www-03.ibm.com/chips/news/2004/1129_cell1.html
- [2] <http://www.mpi-forum.org/>
- [3] <http://www.sgi.com/products/servers/altix/memory.html>
- [4] <http://www.beowulf.org/>
- [5] <http://www.qnx.com/products/rtos/distributed.html>
- [6] W. Emmerich, "Software Engineering and Middleware: A Roadmap", *Proc. of the Conference on the Future of Software Engineering*, 2000 pp. 117-129.
- [7] Object Management Group, *The Common Object Request Broker: Architecture and Specification*, 3.0.2 ed, 2002
- [8] <http://www.microsoft.com/com/>
- [9] <http://www.opcfoundation.org/>
- [10] L.G. Bratthall, R. van der Geest, H. Hofmann, E. Jellum, Z. Korendo, R. Martinez, M. Orkisz, C. Zeidler, J.S. Andersson, "Integrating Hundred's of Products through One Architecture - The Industrial IT architecture", *Proc. of ICSE'02*, 2002
- [11] A.S. Krishna, D.C. Schmidt, Ray Klefstad, and Angelo Corsaro, "Real-time CORBA Middleware", in *Middleware for Communications*, edited by Qusay Mahmoud, Wiley and Sons, New York, 2003.
- [12] I. Foster, C. Kesselman, J.M. Nick, and S. Tuecke, "*The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration*", Technical report, Open Grid Services Architecture WG, Global Grid Forum, 2002.
- [13] <http://www.globus.org/>
- [14] A. Gokhale, K. Balasubramanian, A.S. Krishna, J. Balasubramanian, G. Edwards, G. Deng, E. Turkay, J. Parsons, D.C. Schmidt, "Model Driven Middleware: A New Paradigm for Developing Distributed Real-Time and Embedded Systems", submitted to *Journal of Science of Computer Programming*, 2005

Transcript from the discussion

Q (AS to everyone.): What do we want? What do we believe we will see in the future? And how does this influence YOUR particular field of work?

Shi: In embedded system, there will be more chip space?
Limit on computer resources will not be a problem. I think we basically agree,

Kone: If I look at my field, it is interesting to consider a human specific language?

Ries: Some non-distributed issues are not solved.

The ideas put forward implies tremendous overhead. It makes sense when not using CORBA,, but there is (will always be) need for several protocols. Power consumption is a limitation!

Reliability issues?

Risks associated. Security is often not considered in industrial applications, which can be a problem. Features not considered safety critical becomes safety critical because people rely on them

There was some confusion about proper frameworks to model/store the meta information in the system. This is an important issue: Which information should every device have, and what should be shared?

Allocation of Dependable Software Modules under Consideration of Replicas¹

Georg Weissenbacher

Wolfgang Herzner

Egbert Althammer

ARC Seibersdorf research

[georg.weissenbacher, wolfgang.herzner, egbert.althammer]@arcs.ac.at

Abstract

In dependable embedded systems, it is current practice to assign each application subsystem to a dedicated processor. However, several activities aim at an integrated approach, allowing the deployment of multiple application subsystems on a single distributed computer system [1]. The resulting large number of feasible allocations of tasks to processors makes it hard for the developer to determine an optimal solution, and therefore automatic allocation is desirable. This paper presents an allocation algorithm that takes advantage of the existence of replicated software and hardware components in dependable embedded systems and the resultant symmetric solutions in order to minimize the number of allocations taken into consideration.

1. Introduction

More and more functions in today's cars are realized by embedded computer systems. Eventually, even highly safety critical mechanical and hydraulic control systems will be replaced by electronic components. The DECOS project [2] aims at making a significant contribution to the safety of dependable embedded systems by transitioning from federated to integrated systems [1]. In federated systems, each application subsystem is located on a dedicated processor. The federated approach provides natural separation of application functions, but causes increased weight and cost due to resource duplication and the large number of wires and connectors. Integrated systems not only help to alleviate this problem, they also permit communication among application functions. A remarkable feature of the integrated DECOS architecture is that hardware *nodes* are capable of executing several *tasks* of application subsystems of different criticality.

Throughout this paper, we will use the notion of a *node* instead of processor.

An integrated architecture provides a fixed number of nodes, each of which has certain properties (e.g., size of memory, computational power, I/O resources). All tasks have to be allocated such that given functional and dependability constraints are satisfied. The allocation of tasks to nodes can have a strong impact on the result of the assessment of the concrete allocation. Which allocation is deemed optimal depends on an optimization function which assigns a cost value to every solution.

Such allocation problems are commonly denoted as Constraint Satisfaction Optimization Problems (CSOPs) [3]. This class of problems has been subject of research for several decades, and a great number of solutions has been established [3, 4]. While such highly general algorithms are usually a good first approach, the exploitation of problem-specific features can result in a significant reduction of the complexity of the problem. In this paper we explain why replicated components, which are indispensable in safety critical systems, make the allocation problem easier.

2. Related Work

The number of contributions related to CSOP is overwhelming, and it is impossible to give a complete overview of this field. Several efficient heuristics for the allocation problem are presented in [5], but the main focus of this work is on the assessment of the goodness of a mapping. Symmetry resulting from equivalent processors has already been considered for scheduling of embedded systems in [6]. A method for handling symmetry during search is described in [7]. Furthermore, it should be noted that exploitation of symmetry is a standard technique in the related area of Model Checking [8]. Our approach aims at reducing the search space by taking advantage of the fact that equivalent redundant elements (i.e., replicas) exist. We

¹ Research supported in part by EC IST FP6 IP DECOS No. 511764

are not aware of any publication that relates symmetry and replicated components.

3. Formal Definition of the Problem

A Constraint Satisfaction Problem (CSP) consists of a finite set of variables Z , each of which can be assigned a value from a corresponding finite domain D . The set of *valid* assignments is restricted by a number of *constraints* C . Constraints are relations ranging over a set of variables and their values. Assignments under which at least one of the constraints is violated (i.e., evaluates to false) are *invalid*.

A CSOP is composed of a CSP augmented by an optimization function f that associates each assignment with a numerical value. The goal of a CSOP is to determine the assignment with the optimal value of f . The optimization function f indicates the “goodness” of a solution by assigning higher values to better solutions.

A CSP (or CSOP) is *satisfiable* if (and only if) a valid assignment exists. An algorithm for a CSP (or CSOP) is *complete* if (and only if) it finds every valid assignment. An algorithm is *sound* if every result that it returns is indeed a valid assignment.

Resource allocation problems (as the one presented in Chapter 1) are CSOPs. Such problems are in general NP-complete [9], i.e. no polynomial time algorithms are known. Remember that we want to assign tasks to nodes. Each node can host more than one task, therefore, tasks will take the role of variables Z in the CSOP defined above, and the set of available nodes will define the domain D for these variables. Assume that n denotes the size of Z , and m is the cardinality of D . In the worst case, one will have to consider m^n assignments to find the optimal solution, which is defined by the optimization function f .

The number of valid assignments is restricted by various constraints. Examples for constraints would be dependability constraints (location of replicas, fault containment), resource constraints (memory, computation resources, availability of sensors or actuators on a certain node, bandwidth), and performance constraints (deadline, precedence). To prevent loss of generality, we do not make any assumptions about the nature or representation of constraints in this paper. Depending on the allocation algorithm and the constraints, it is often necessary to consider a large number of partial assignments before the search tree can be pruned.

4. Optimization using Equivalence Classes

This chapter outlines the optimization which we propose. In a safety critical system, replicas are intro-

duced to increase reliability. Replicas are redundant elements with equivalent properties, e.g. an exact copy of a job running on a different node. Since such redundant elements share the same attributes, they are interchangeable with respect to the constraints and the optimization function f . Consider a system with two nodes a_1 and a_2 , and two replicated tasks j_1 and j_2 . The mappings $m_1: \{j_1 \leftarrow a_1, j_2 \leftarrow a_2\}$ and $m_2: \{j_1 \leftarrow a_2, j_2 \leftarrow a_1\}$ are equivalent with respect to the evaluation of the constraints and the optimization function f . m_1 and m_2 are redundant, i.e. only one of them has to be considered.

Furthermore, nodes may be considered as equivalent if they are of the same type and provide the same resources. The impact of equivalent elements in the domain D on the performance of the search algorithm is not as significant as the impact of task replicas, but experiments have shown that the gain is still worth the additional effort.

In order to take advantage of the existence of equivalent tasks and nodes the sets of equivalent elements have to be computed. The complexity of this problem depends on the representation of the constraints and the optimization function f . It is impossible to propose a generic approach for solving this problem. The deduction of equivalence sets is trivial if the constraints and the optimization function depend solely on the attributes (e.g., memory demands or size, computational power) of the elements. In that case, equivalence sets can be inferred by simply comparing these attributes. An even more trivial case would be if tasks are actually tagged as replicas. If constraints are given in First Order Logic [10], for instance, it might be necessary to use more complex methods like term rewriting [10] to identify the equivalence sets. However, it should be mentioned that completeness of the algorithm used to compute these sets is not a necessary precondition: If the equivalence of an element remains undetected, the search algorithm will perform less efficient, but it is still sound and complete.

5. Commonly Used Search Algorithms

We present two well known search algorithms, one of which we will later on optimize by applying the concept outlined in the previous chapter. We assume that the search space is bounded and represented as a rooted tree graph (i.e., a tree graph with a distinct root vertex). This assumption is hardly restrictive, but eases argumentation. Each inner vertex represents a partial assignment, and leaves represent complete allocations. Each vertex except the root vertex has one incoming edge. Each vertex except the leaves has $m > 0$ outgoing

edges. An edge represents an allocation of a task to a node. We denote a sequence of vertices such that there is an edge from each of its vertices to the successor vertex, with each vertex occurring only once in the sequence, as path. The maximum length of a path is n .

“Depth First Search” (DFS) algorithms explore one individual path after another. Starting with the root element, each path is successively extended until either one of the constraints is violated or a valid assignment has been found. In case of a constraint violation, the algorithm backtracks to the vertex visited last and follows an edge that is different from the one that led to the constraint violation.

“Breadth First Search” (BFS) algorithms start at the root element and successively explore all neighbors of the vertices that have already been visited. Whenever a constraint is violated at a vertex, the corresponding subtree is pruned.

BFS as well as DFS algorithms are uninformed search algorithms and have equal time complexity if all valid solutions have to be generated. More sophisticated approaches use heuristics, aiming at considering more promising paths first. The optimization we propose is orthogonal to the above mentioned methods, i.e. it can be applied to improve DFS and BFS as well as other search algorithms.

6. Applying the Optimization to DFS

Let us demonstrate, based on DFS, how a single allocation step is performed. In DFS, the task variables j_1, \dots, j_n are considered one after another, and we assume that the variable currently processed is j_i . j_i is assigned a_l , and if no constraints are violated, the algorithm proceeds to j_{i+1} . Otherwise, a_2 will be considered as node running j_i . If all assignments to j_i yield a constraint violation, the algorithm will have to backtrack and consider different nodes for j_{i-1} . If the allocation problem is satisfiable, the algorithm will sooner or later yield a valid allocation.

Now, consider that the n tasks j_1, \dots, j_n have been partitioned into k equivalence sets J_1, \dots, J_k . The nodes a_1, \dots, a_m have been partitioned into l equivalence sets A_1, \dots, A_l . The equivalence sets J_1, \dots, J_k are considered one after another. Assume (without loss of generality) that J_i contains r tasks j_1, \dots, j_r . The first element $j_1 \in J_i$ is allocated on $a_l \in A_l$. If no constraints are violated, the algorithm has to perform following two actions:

1. Remove j_1 from J_i , since j_1 need not be considered again.
2. Check which attributes of a_l have changed due to the assignment. For instance, the amount of available memory and computation resources

will have changed, since j_1 consumes at least some of these resources. Therefore, it is necessary to place a_l in a different equivalence set, or, if a_l fits into none of the existing equivalence sets, to create a new equivalence set A_l^+ that contains only a_l . In the latter case, A_l^+ is added to the other node equivalence sets, thus increasing their number by one.

If the assignment $j_1 \leftarrow a_l$ results in a violation of at least one constraint, the algorithm tags A_l as “considered for J_i ” and proceeds with considering the next equivalence set of nodes. If all equivalence sets have been considered, the algorithm has to backtrack similar to the simple DFS approach.

The pseudo-code for the optimized allocation algorithm is presented in Listing 1. The code shows a DFS algorithm that has been optimized using equivalence sets. We will further on refer to this algorithm as “Equivalence-Set based DFS” (EDFS). The proposed optimization can be applied to other search algorithms like BFS or algorithms that use heuristics in an analogous way.

7. Complexity Analysis

As mentioned above, the worst case execution time for a resource allocation problem is of exponential order. In our case, this means that m^n assignments would have to be considered. It is obvious that EDFS behaves like DFS if each equivalence set contains exactly one element (at least if one does not consider the overhead for handling equivalence sets). This is not surprising, since resource allocation remains an NP-complete problem.

Obviously, the larger the equivalence sets become, the better is the performance of EDFS (in other words, EDFS performs better for a small number of equivalence sets). As suggested above, we assume that we have n tasks which are partitioned into k equivalence sets, and m nodes partitioned into l equivalence sets.

Let $W(i, r_i, s_i)$ denote the worst case number of assignments that EDFS needs to process for one task equivalence set. r_i is the cardinality of the current task equivalence set, and s_i denotes the number of current equivalence sets of nodes. Then, in the worst case, EDFS has to consider at least

$$O \prod_{i=1}^k W(i, r_i, s_i) \quad \text{with } r_i = |J_i|$$

partial allocations, i.e. the runtime of the algorithm is at least exponential in k . In Listing 1, the task equivalence sets J_1 to J_k are considered one after another. Since the order of tasks within each equivalence set J_i does not

matter, the term $W(i, r_i, s_i)$ can be approximated by applying the formula for combinations with repetition. Repetitions are allowed, since even if one task of the current task equivalence set has been assigned to a certain node, this very node is in general still available for the assignment of an additional task from the same set. This fact is reflected by the variable r_i .

Obviously, r_i is the number of elements to be chosen. Estimating the number of objects from which one can choose is slightly more complex: The number of equivalence sets for nodes at step i is hard to predict, since it depends on whether the algorithm was able to reintegrate nodes into the existing equivalence sets after allocating tasks to them. The worst case is that the number of equivalence sets for nodes reaches m after $(m-1)$ assignments, since we have to assume that each assignment generates a new node equivalence set A_i^+ . This implies that node equivalence sets make no significant contribution to the improvement of the complexity – in the worst case. The effort for reintegrating nodes into the existing equivalence sets is linear in the number of nodes, since the node has to be compared to at most $(m-1)$ other nodes.

The number of ways to choose k elements from a set of n if repetitions are allowed is

$$\binom{n+k-1}{k} = \frac{(n+k-1)!}{k!((n-1+k)-k)!}$$

We can therefore conclude that

$$O(W(i, r_i, s_i)) = O\left(\frac{(s_i + r_i - 1)!}{r_i!(s_i - 1)!}\right)$$

with $r_i = |J_i|$ and $s_i = m$.

```

    ∅ h
      ∅ h
        h
; //
x k k;
// w h x
x 1 ;
\ x ;
∪ ;
{ ∈ |
  h h
  x } ;
\ ;

```

```

x ;
∅ h
x k k;
T A 1 ;
x 1 ;
x y x ;
+ { T A ← x } ;
\ T A ;
x x \ x ;
x < ∅ h x
y x ;
x h x
h w h
x h
x ;
;
;
;
// y x
y
h k k
+ x
\ x ;
y
∅ ∅
h k k ∅ x ;
;

```

Listing 1: Pseudo-code for EDFS

The worst case execution time of EDFs depends on a multitude of parameters, namely the number of task equivalence sets, the cardinalities of these sets, and the number of processors. Most of these parameters are irrelevant for the runtime of the simple DFS algorithm. Therefore, it is impossible to provide a general comparison the efficiency of DFS and EDFs. As mentioned above, DFS and EDFs perform equally well if there are no equivalent elements (aside from the overhead of EDFs, which will be discussed in Chapter 9).

The example and the experimental evaluation presented below will underpin the superiority of EDFs over DFS.

8. Example

In this section we present a small but still realistic example of an embedded system and explain how EDFs is more appropriate for the task allocation than a brute force DFS approach.

We consider a Steer-By-Wire system consisting of 7 tasks. The system provides a “Driving Assistant” that recognizes critical driving situations, prevents oversteering and takes countermeasures if the driver steers too sharp. The system comprises following modules:

1. **Driver_Assistant:** Receives its input from sensory data sources and adjusts these values (if necessary) in order to keep the car safely on course.
2. **Steering_Algorithm:** Determines the analogue value that is used to control the mechanical steering system (under consideration of the angle of the steering wheel and the actual turning angle of the wheels).
3. **Steering_Rack_Sensor:** Provides information about the forces that affect the mechanical steering system.
4. **Steering_Rack_Control:** Controls the turning angle of the wheels using the values provided by the steering algorithm. If the task that realizes the steering algorithm is defunct, the steering rack control uses the raw data provided by the sensory tasks.
5. **Force_Feedback:** Provides haptic feed-back to the driver.
6. **Turning_Angle_Sensor:** Measures and provides the current angle of the steering wheel.
7. **Speed_Sensor:** Measures and provides the current speed of the vehicle.

For reasons of simplicity, we assume that each module is realized by a single task. Tasks have to be replicated if their breakdown could potentially jeopardize the safety of the driver. In our example, this is the

case for the `Steering_Rack_Control` task, the `Turning_Angle_Sensor` task, and the `Speed_Sensor` task (see Table 1). Due to replication we have to consider an overall number of 10 tasks. In order to prevent common mode failures, task replicas must not be located on the same node. This requirement can be expressed as constraint that limits the number of valid allocations.

Task	Time Budget	Number
Turning_Angle_Sensor	270	2
Speed_Sensor	270	2
Steering_Rack_Control	130	2
Steering_Rack_Sensor	110	1
Steering_Algorithm	110	1
Driver_Assistant	270	1
Force_Feedback	110	1

Table 1: Time budgets of Steer-By-Wire tasks

We aim at distributing the tasks over 4 nodes. In our simple example we assume that all nodes can be equipped with the required sensors. In practice, it would be possible to further restrict the valid allocations by means of constraints. Two of the 4 nodes have sufficient memory resources such that each of them can run 3 tasks. The memory resources of the remaining two nodes are chosen such that only two tasks can be hosted. Each of the nodes is assigned a time budget of 600 “units”. Table 1 associates each task to a time budget that is required to execute the corresponding code.

According to the results of the complexity analysis, the worst case of a non-optimized search algorithm would be that 4^{10} (=1048576) allocations have to be considered. The experimental evaluation below will show that even the brute force DFS algorithm considers only a fraction of these solutions.

The computation of the worst case for EDFs is slightly more complicated. We have to consider 7 task equivalence sets, 3 of which contain two tasks, while the remaining sets contain only one single task. Furthermore, we start with two node equivalence sets.

In the first step, we have to consider that the number of node equivalence sets is potentially increased by one whenever an assignment of a task to a node is performed. We will approximate this fact by assuming that there are 3 node equivalence sets for the first task equivalence set.

$$W(1, r_1, s_1) = \frac{(3 + 2 - 1)!}{2!(3 - 1)!} = 6$$

under the assumption that $r_1=2$ and $s_1=3$. For the second and third task equivalence set we have to presume the existence of 4 node equivalence sets:

$$W(2, r_2, s_2) = \frac{(4+2-1)!}{2!(4-1)!} = 10$$

with $r_2=2$ and $s_2=4$. $W(3, r_3, s_3)=10$ follows from the same considerations. EDFS performs equally to DFS for the remaining 4 task equivalence sets, which contain only a single element, since

$$W(i, r_i, s_i) = \frac{(4+1-1)!}{1!(4-1)!} = 4 \text{ for } i = 4..7$$

Therefore, EDFS considers $6 \cdot 10^2 \cdot 4^4 = 153600$ allocations at most, i.e., in the worst case, the algorithm performs approximately 7 times better than search without optimization.

9. Experimental Evaluation

We have implemented a DFS as well as the EDFS algorithm presented in Listing 1 in the OCaml [11] programming language. The DFS algorithm yields all valid allocations, while EDFS generates only one representative for each set of symmetric solutions. A complete list of valid allocations could be gained by generating all permutations under consideration of the equivalence sets. In practice, this will not be desirable since the optimization function f is indifferent to symmetric solutions, and considering all valid allocations would be a pointless effort.

We measured the efficiency of the algorithms by counting the number of partial assignments that are considered. The constraints have to be evaluated for each partial assignment, and if we assume that this evaluation takes a significant amount of time, this value is a good indicator for the improvement. According to our measurements, the overhead for handling the equivalence sets amounts for about 15 to 20% of the runtime. We believe that this number can be improved significantly, since we used a purely functional programming style and we did not apply any optimizations.

Our implementation of the DFS algorithm considered 15516 partial assignments before it yielded the valid allocations for the Steer-By-Wire system presented above. The EDFS algorithm terminated after considering only 797 partial assignments, exceeding the expectations from the previous chapter. (The reason for this improvement is that the worst case did not occur, presumably due to early truncation of several branches of the search tree and due to the fact that our worst case considerations explicitly excluded reintegration of nodes into equivalence sets.)

Apart from the Steer-By-Wire example, we have made several experiments that underpin our assumption that EDFS performs better than DFS in many cases. In

the worst case ($n=k$ and $m=l$, i.e., all equivalence sets contain exactly one element) EDFS degenerates to DFS. Table 2 presents the results of some of these experiments. The 3rd line presents the worst case where EDFS and DFS perform equally due to the absence of equivalence sets.

n	m	k	l	DFS	EDFS
10	4	7	2	15516	797
10	10	5	5	14308950	77911
10	10	10	10	223742	223742
8	4	4	2	768	33

Table 2: Experimental evaluation of EDFS

The actual number of partial allocations considered by EDFS and DFS depend on many more parameters than shown in Table 2. For example, strong constraints can result in an early truncation of branches of the search tree. Furthermore, the ordering in which the variables are considered and the ordering in which the values are assigned has a tremendous impact on the efficiency of the search algorithm [3]. For the Steer-By-Wire example, we were able to find variable orderings make DFS and EDFS perform significantly better. In general, finding an optimal ordering for the variables is infeasible, since even checking that a particular ordering is optimal is NP-complete [8].

Note that the usage of the equivalence set based optimization does not rule out other optimizations like variable ordering heuristics. A great number of suggestions for generic optimizations can be found in [3].

10. Conclusion

Whether the effort to consider equivalence sets is worthwhile depends on the concrete application of the algorithm. The potential symmetry caused by the existence of replicas makes dependable embedded systems an ideal field of application, and our experiments show that EDFS performs much better than non-optimized search algorithms. As mentioned in the previous section, the proposed optimization can easily be combined with other optimization approaches. The approach will be considered for software/hardware integration [12] within the DECOS project [2].

11. References

- [1] H. Kopetz, R. Obermaisser, P.Peti and N. Suri, "From a Federated to an Integrated Architecture for Dependable Embedded Real-Time Systems (draft)", Vienna University of Technology, Austria, and Darmstadt University of Technology, Germany, 2004.
- [2] Dependable Embedded Components and Systems, Integrated Project within the EU Framework Programme 6, <http://www.decos.at>
- [3] E.P.K. Tsang, *Foundations of Constraint Satisfaction*, Academic Press, London and San Diego, 1993.
- [4] V. Kumar, "Algorithms for Constraints Satisfaction Problems: A Survey", *The AI Magazine*, v. 13 n. 1, AAAI, 1992, pp. 32-44.
- [5] N. Suri, A. Jhumka, M. Hiller, T. Marlowe, "A Software Integration Approach for Designing Dependable Distributed Systems" submitted to Elsevier Science, 2004.
- [6] C. Ekelin, "An Optimization Framework for Scheduling of Embedded Real-Time Systems", Doctoral thesis at Chalmers University of Technology, Sweden, 2004
- [7] B. M. Smith, I. P. Gent, "Symmetry Breaking in Constraint Programming", *Proceedings of ECAI*, 2000, pp. 599-603
- [8] E.M. Clarke, O. Grumberg, D.A. Peled, *Model Checking*, The MIT Press, Cambridge, Massachusetts and London, England, 1999.
- [9] M.R. Garey, D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Company, New York, 1979.
- [10] F. Baader, T. Nipkow, *Term Rewriting and all that*, Cambridge University Press, Cambridge, 1998.
- [11] OCaml, <http://caml.inria.fr/ocaml>
- [12] S. Islam, G. Csertan, W. Herzner, "A Multi Variable Optimization Approach for SW-HW Integration", Fast Abstract to appear at HASE 05: High Assurance Systems Engineering Conference, Heidelberg, Germany, 2005

Transcript from the discussion

Q(lan): it's a space search for optimal solutions. Can you
A: It's a matter of comparing with the best solution so far. Heuristic approach.
Evaluation of the optimal solutions

Q(A): actually it is a problem with several electronic subsystems in cars, because they fail
A: several problems with several electronic systems. Cost, heat dissipation.
Scaling is important, as can be seen in case of Airbus A380.
Problem with integrated architecture could be with locating the faulty part(s), when something fails.

Q(Shi): does approach apply to safety critical only, or?
There's no limitation on the method for this.

Q: is there need for a MMU for the OS, applications on the platform?
specification

Q: can be
Used also for industrial control, avionics, automotive and lots of applications

Assessing Reliability of Real-Time Distributed Systems

Åsmund Tjora
Norwegian University of Technology and Science
Department of engineering cybernetics
O.S. Bragstads plass 2d
7491 Trondheim
Norway
{tjora, skavhaug}@itk.ntnu.no

Abstract

Serial replication structures are used in fault tolerance mechanisms in systems. Analysing the timing behaviour of systems using these mechanisms can be difficult, however, it is necessary to do this kind of analysis if the systems are to be used in real-time applications.

In this paper, two ways of analysing the timing behaviour in such systems are studied, a simulator and an analytical model. Some of the strengths and weaknesses of these methods are discussed, and it is demonstrated with an example how they can be used.

1. Introduction

Many of today's real-time applications will, in addition to the timing requirements, also have reliability requirements. It is therefore important to study the combination of real-time and fault tolerant systems. A fault tolerance mechanism may be able to detect and correct a number of faults, however, in a real-time system, it is possible that the same mechanism introduces new faults if the time it uses for detection and correction causes the system to miss a deadline. Because of this, a fault tolerant mechanism may give very little improvement to the reliability of a system, depending on the system's characteristics.

For many real-time systems, it has been common to use a parallel fault tolerance structure, that is, several replicas of one object are run simultaneously. This will give a deterministic temporal behaviour even if a fault is detected in one of the replicas, and comparison of the results from the different replicas can be used as an extra fault detection mechanism. On the negative side, parallel fault tolerance mechanisms needs extra resources.

In a system using a serial fault tolerance structure, only

one instance of an object is active at one time, while the other replicas are passive. If a fault is detected, the task running on the object is stopped, and one of the passive replicas is made active. The task is then rerun on this new active object. The mechanisms based on serial structures use fewer resources than those based on parallel structures, making them quite popular in general purpose systems. In real-time systems, the time used for detecting a fault, updating the states of the replica, and rerunning the task may cause deadline misses. When considering a fault tolerance mechanism based on a serial structure for a real-time system, it is therefore important to analyse the time used by such a system.

In this paper, two main methods of analysing the timing behaviour of a fault tolerant system are presented: Analytical models of the behaviour, and simulation.

Analytical models give precise results, and once a model is developed for a system, the same model can often be used with little or no modification in similar systems. However, the mathematical expressions can become quite complicated, even for a simple system, and such models are therefore not always easy to understand or use.

Simulation may give good results, and is often easier to understand and use than analytical models. A problem with simulations is that they have to be run a large number of times if events with a low probability of occurring are to be studied.

2. The model of the fault-tolerant system

The system that we are modelling is a simple Client-Server system where the server is using a fault tolerant mechanism based on a serial structure. The model we are using for the fault tolerance mechanism is based on the cold and warm passive replication methods described in Fault Tolerant CORBA [3].

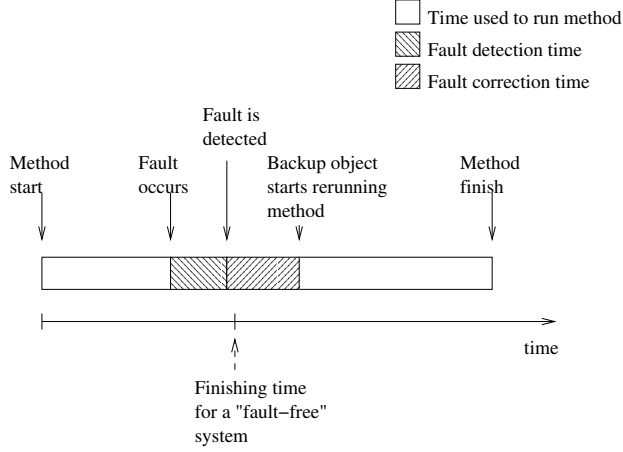


Figure 1. Timing behaviour of system during a fault

In the system, there is one active primary object and several passive secondary objects. The state of the primary object is logged, and if the primary object fails, the state of one of the secondary objects is updated from the log. This object will now be the new primary, and the task that was running on the failed object will be rerun.

In the system, fault detection is based on some kind of watchdog mechanism. A fault detector will either ask the primary object if it is “alive”, or the primary object will generate “heartbeats” that the fault detector will listen to. In both cases, if the fault detector doesn’t get a response in what is considered a reasonable time, it is assumed that the primary object has failed, and the fault correction mechanisms will start to work.

When a fault is detected, the fault correction mechanism will prepare one of the secondary objects by updating its state from the log. When the object is up to date, the task that was running will be rerun on it. The timing behaviour of the system in a case where a fault occurs is shown in figure 1.

We also have to consider cases where several faults occur during the service of one task.

If a new fault occurs after the correction of the previous fault has started, the whole detection-correction-rerun cycle has to be restarted. However, if the fault occurs before the previous fault is detected, it will have no extra effect on the system, since it is still the already faulty object that is active.

3. The Analytical Model

The analytical model we use is the one presented in [4]. The expressions are derived using a method similar to the one used to find the distribution of the busy period in queuing systems, as presented by Kleinrock in [2].

ing systems, as presented by Kleinrock in [2].

In the expressions, we use the moment generating functions (mgf) of the different distributions. A distribution’s moment generating function is the laplace transform of its probability density function (pdf), i.e. $\mathcal{F}(s) = \int_0^\infty e^{-st} f(t) dt$.

In this paper, we use calligraphic letters (i.e. $\mathcal{F}(s)$) to represent a distribution’s moment generating function and normal lowercase letters (i.e. $f(t)$) to represent the distribution’s probability density function.

The model gives us the moment generating function, $\mathcal{G}(s)$, of the run-time distribution for a fault-tolerant system based on a serial structure as a function of the moment generating functions of the fault-free run-time, the fault detection and correction times and the fault intensity. Mathematical tools may then be used to transform the mgf back to the time domain.

We assume that the time used to run a fault-free method is given by the pdf $m(t)$, the time used to detect faults is given by the pdf $i(t)$, and the time used to correct a fault (i.e. updating the state of a passive object and restarting method on this object) is given by the pdf $c(t)$, and that these distributions have the moment generating functions $\mathcal{M}(s)$, $\mathcal{I}(s)$, and $\mathcal{C}(s)$. We also assume that the faults are independent and that fault arrivals can be modelled by a poisson process with intensity λ .

Given these parameters, the mgf of the run-time distribution in a system where faults may occur is given by

$$\begin{aligned} \mathcal{G}(s) &= \mathcal{M}(\lambda + s) + \left(\sum_{k=1}^{\infty} \left(\frac{\lambda}{\lambda + s} \right)^k \mathcal{I}(ks) \right. \\ &\quad \left(\sum_{i=0}^{k-1} (-1)^i \binom{k-1}{i} (\mathcal{M}((i+1)(\lambda + s)) \right. \\ &\quad \left. \left. - \mathcal{M}((i+2)(\lambda + s))) \mathcal{C}((i+1)(\lambda + s)) \right) \right) \end{aligned} \quad (1)$$

There is an infinite sum in the expression. This is explained by the fact that there always will be a possibility that the system will fail again while a task that has failed before is still running. This also makes it possible that an infinite number of faults may occur during the running of the same task. The infinite sum can be approximated to a finite sum in two ways:

- Assuming that only a finite number N faults will happen during the run-time of one task, so that no new faults will occur after this.
- Assuming that the task will fail if the number of faults occurring during the running of one task exceeds N .

The mgf for the first approximation is given by

$$\begin{aligned}
& \mathcal{G}_1(s) \\
&= \mathcal{M}(\lambda + s) + \left(\sum_{k=1}^{N-1} \left(\frac{\lambda}{\lambda + s} \right)^k \mathcal{I}(ks) \right. \\
&\quad \left. \left(\sum_{i=0}^{k-1} (-1)^i \binom{k-1}{i} (\mathcal{M}((i+1)(\lambda + s)) \right. \right. \\
&\quad \left. \left. - \mathcal{M}((i+2)(\lambda + s))) \mathcal{C}((i+1)(\lambda + s)) \right) \right) \\
&\quad + \left(\frac{\lambda}{\lambda + s} \right)^N \mathcal{I}(Ns) \left(\sum_{i=0}^{N-1} (-1)^i \binom{N-1}{i} \right. \\
&\quad \left. (\mathcal{M}(i\lambda + (i+1)s) - \mathcal{M}((i+1)\lambda + (i+2)s)) \right. \\
&\quad \left. \mathcal{C}(i\lambda + (i+1)s) \right)
\end{aligned} \tag{2}$$

while the mgf for the second is given by

$$\begin{aligned}
& \mathcal{G}_2(s) \\
&= \mathcal{M}(\lambda + s) + \left(\sum_{k=1}^N \left(\frac{\lambda}{\lambda + s} \right)^k \mathcal{I}(ks) \right. \\
&\quad \left. \left(\sum_{i=0}^{k-1} (-1)^i \binom{k-1}{i} (\mathcal{M}((i+1)(\lambda + s)) \right. \right. \\
&\quad \left. \left. - \mathcal{M}((i+2)(\lambda + s))) \mathcal{C}((i+1)(\lambda + s)) \right) \right)
\end{aligned} \tag{3}$$

If there is a low probability that a fault occurs during the running of a task, the probability of several independent faults occurring during the same task becomes so small that these approximations are reasonable. Also, since the systems we are analysing have deadlines, and each fault will take some time to detect and correct, there can only be a finite number of faults before we are guaranteed to *miss* the deadline.

A difference between the two approximations is that the first approximation models a system that will never fail completely, while the second will always have a small probability that the running of a task is never finished.

4. The Simulator

The simulator is designed as a simple server-client system, as shown in figure 2. For the system presented in this paper, the network part is omitted.

4.1 The Client

In the system presented here, the client part of the simulator functions mostly as a generator of an object type called *process*. At regular intervals, the client generates a

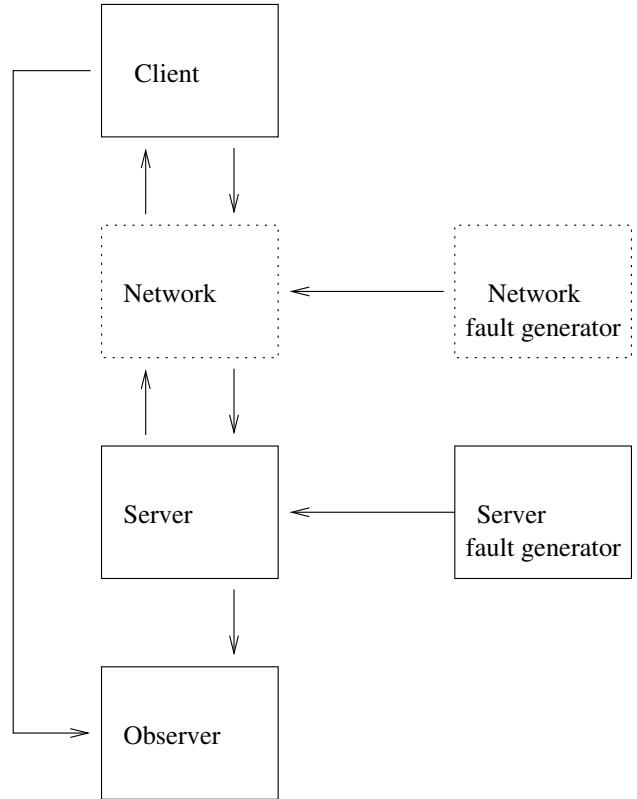


Figure 2. Structure of a simple client-server simulator.

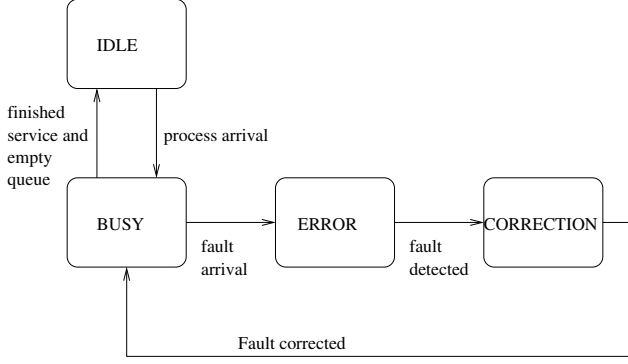


Figure 3. Simplified state machine diagram for the server model.

new *process* object, which is time-stamped and sent to the server. When the network part of the model is used, it also receives *process* objects that are returned from the server, time-stamps them, and sends them to the observer.

4.2 The Fault Generator

The fault generator is a “pure” generator that creates *fault* objects at intervals drawn from a negative exponential distribution. These are sent directly to the server.

4.3 The Server

During normal operation, the server function is quite straightforward. When a *process* object arrives, it is placed in a queue. If the server is *idle* when this happens, or if it has just finished a service while the queue is not empty, it will take the next object in the queue. It will then set its status to *busy*, draw a service time from the service time distribution, and set the next event to happen when this time is up.

When a *fault* object arrives, the server will set its status to *error*, draw a time from the detection time distribution, and set the next event (i.e. the fault detection event) to happen when this time is up. When the fault is detected, the server sets its status to *correction* and draws the time to the next event from the correction time distribution. After the fault is corrected, the server returns to *busy*, and restarts service on the *process* object by setting the next event to happen after the service time.

A simplified state machine diagram for the server is shown in figure 3

4.4 The Observer

The observer receives *process* objects that has finished running. The data from these objects are extracted and pre-

sented in a way that a program used for data analysis can read.

4.5 Implementation

The simulator is implemented in C++, using the ADEVS discrete event simulator framework [1] as a base. For data analysis, Matlab is used.

5. Example

We will now look at the results from the mathematical model and the simulator for a simple system.

5.1 System parameters

For the fault free running time, we have chosen a triangular distribution with minimum time 6, maximum time 10 and the mode at 8.

$$m(t) = \begin{cases} 0 & , 0 \leq t < 6 \\ \frac{t-6}{4} & , 6 \leq t < 8 \\ \frac{10-t}{4} & , 8 \leq t < 10 \\ 0 & , t \geq 10 \end{cases} \quad (4)$$

The fault detection time is uniformly distributed with a minimum time 1 and a maximum time 3.

$$i(t) = \begin{cases} 0 & , 0 \leq t < 1 \\ \frac{1}{2} & , 1 \leq t \leq 3 \\ 0 & , t > 3 \end{cases} \quad (5)$$

The fault correction time is uniformly distributed with a minimum time 0 and a maximum time 5.

$$c(t) = \begin{cases} \frac{1}{5} & , 0 \leq t \leq 5 \\ 0 & , t > 5 \end{cases} \quad (6)$$

These distributions have the following moment generating functions:

$$\mathcal{M}(s) = \frac{e^{-6s} - 2e^{-8s} + e^{-10s}}{4s^2} \quad (7)$$

$$\mathcal{I}(s) = \frac{e^{-s} - e^{-3s}}{2s} \quad (8)$$

$$\mathcal{C}(s) = \frac{1 - e^{-5s}}{5s} \quad (9)$$

The mean time between faults is set to 10000.

The maximum number of faults during the run of one method for the analytical model is set to 2. While this may seem like a very small number, the effect of modelling more

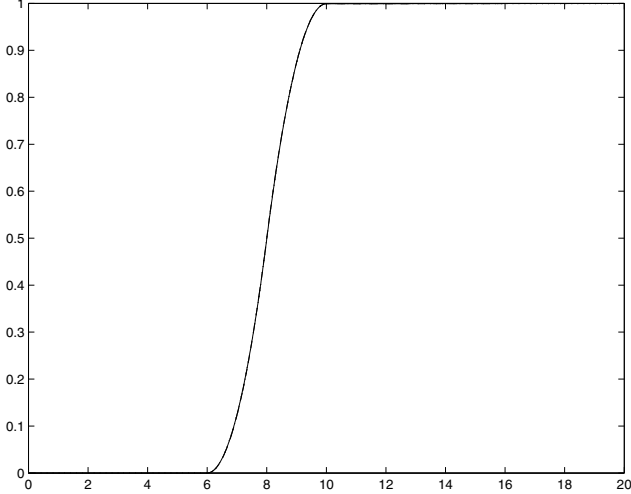


Figure 4. Results from the simulation and the analytical model

faults in this system is very small, while it makes the expression much larger. Using equation 3, the mgf for the run-time distribution for the system is found to be

$$\begin{aligned} \mathcal{G}_1(s) = & \mathcal{M}(\lambda + s) \\ & + \left(\frac{\lambda}{\lambda+s}\right)\mathcal{I}(s) \\ & (\mathcal{M}(\lambda + s) - \mathcal{M}(2\lambda + 2s))\mathcal{C}(\lambda + s) \\ & + \left(\frac{\lambda}{\lambda+s}\right)^2\mathcal{I}(2s) \\ & ((\mathcal{M}(\lambda + s) - \mathcal{M}(2\lambda + 2s))\mathcal{C}(\lambda + s) \\ & - (\mathcal{M}(2\lambda + 2s) - \mathcal{M}(3\lambda + 3s))\mathcal{C}(2\lambda + 2s)) \end{aligned} \quad (10)$$

Since the inverse transform of the mgf will be quite complicated, Matlab and Simulink is used to numerically transform the results to the time domain.

The simulator is set to run 100000 tasks. Matlab is used to evaluate and present the results from the simulator.

5.2 Results

We plot the cumulative density functions (cdf) of the finishing times, using the results from the simulator and the analytical model together, as shown in figure 4. If we zoom in on the plot as in figure 5, we can see that the analytical results (solid line) and the simulator results (dashed line) follow each other quite closely, with an exception in the area around $t = 20$, where the simulator results show a somewhat lower probability of finishing at this time than the results from the analytical model. The curves also show the cdfs of a non-faulty system (top dotted line) and a non-fault-tolerant system (bottom dotted line).

The graphs clearly shows that the improvement caused by the fault tolerant mechanism is almost negligible if the

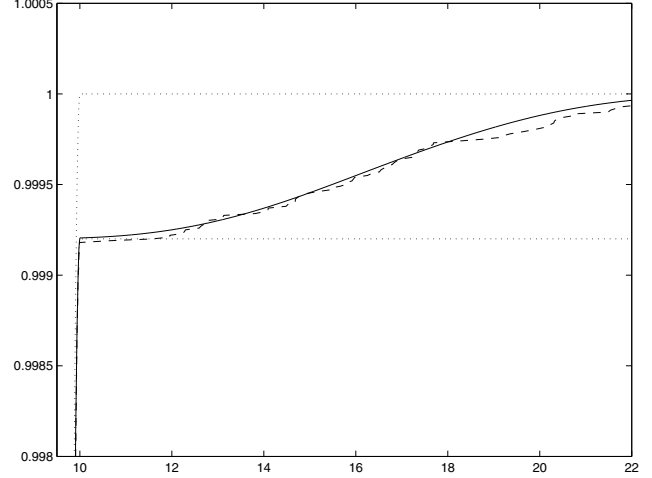


Figure 5. Details of results from the simulation and the analytical model together with the fault-free and non-fault-tolerant distributions

deadline is close to 10, while the improvement is quite good if the deadline is around 20.

The results can be used to determine the probability that a task with a deadline will succeed or fail. If, for instance, we set the deadline to 20, the probability of a failure is $1.2 \cdot 10^{-4}$. Had there been no fault tolerance mechanism, this probability would have been $8.0 \cdot 10^{-4}$.

By changing parameters, we can see how the fault detection and correction times affect the system. As an example, we add 5 to the fault correction time, so it is uniformly distributed between 5 and 10:

$$c(t) = \begin{cases} 0 & , 0 \leq t < 5 \\ \frac{1}{5} & , 5 \leq t \leq 10 \\ 0 & , t > 10 \end{cases} \quad (11)$$

$$\mathcal{C}(s) = \frac{e^{-5s} - e^{-10s}}{5s} \quad (12)$$

The results are plotted with the earlier results in figure 6, showing that the delay in fault correction times causes the fault tolerant mechanism to be almost ineffective if the deadline is 20 or less.

6. Discussion and conclusion

In the models presented here, we assume that all faults are independent, and we also only model faults that can be detected and corrected by the simple fault tolerance mechanism described in section 2. In many cases this is good enough for the analysis of a system.

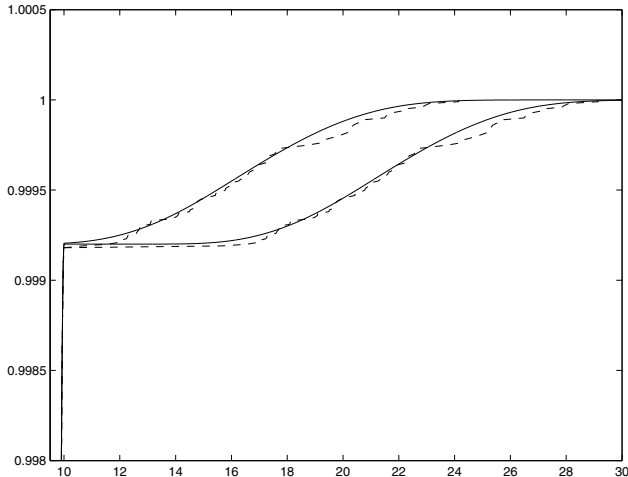


Figure 6. Results showing the effect of adding a delay of 5 to the fault correction

If the behaviour of other fault classes, detection and correction mechanisms are known, expanding the simulator so it is able to handle these, is, in most cases, not very difficult. The difficulty of expanding the analytical model varies greatly, depending on how the faults affect the system and how the detection and correction mechanisms work.

In this paper, two different ways of creating models for the study of the timing behaviour of a fault tolerant real-time system, the simulator and the analytical model, and how the results from these can be used to find the failure probability of methods running on this kind of system are shown.

Some of the strengths and weaknesses of these models are discussed.

Both the simulator and the analytical models have the potential to be expanded, so they can be used to study a wider range of fault types, detection methods, correction methods and fault tolerant structures.

References

- [1] Adevs: A discrete event system simulator. <http://www.ece.arizona.edu/~nutaro/index.php>.
- [2] L. Kleinrock. *Queuing Systems Vol 1: Theory*. John Wiley and Sons, 1975.
- [3] Object Management Group. *The Common Object Request Broker: Architecture and Specification, ver 2.6*, December 2001.
- [4] A. Tjora and A. Skavhaug. A general mathematical model for run-time distributions in a passively replicated fault tolerant system. In *Proceedings. 15th Euromicro Conference on Real-Time Systems*, 2003.

Transcript from the discussion

Q(Ian): You try looking at probability with discrete markov models. Couldn't you assign probabilities to transitions and use derive analytical formulas directly from the markov model instead of Laplace functions?

A: I am more interested in the time: the time between when a task arrives and when it finished, more so than what particular state the system is in at any time.

Q (Ian): You mention that the system lends itself to doing other kinds of fault-tolerant strategies. Have you considered using something with checkpointing, what would the analytical model look like?

A: One of the things that I've looked at is what will happen if you have specific times where you can check if there is a fault, because you won't get the same results. It will be somewhat easier because in this model you can detect a fault at any time while it's running (more complex).

But we need to consider other fault detection mechanisms, because there will be other fault types and behaviours, and I've begun looking into some of these. But I do not have results here.

Q(ES): If I was a user I would be interested in what is the worst-case behaviour?

A: agrees

ES: An idea is to show more information about the system, maybe in 3 dimensions, to make comparisons, with respect to failure rate (λ , repair rates)? More instructive that way.

Another idea: perhaps introducing a coarse algorithm to get atleast some results, to improve dependability.

ÅT: plan to use short, imprecise algorithm for results.

e.g. use a different algorithm/distribution for the backup routine

ASSESSMENT OF SAFETY CRITICAL SYSTEMS WITH COTS SOFTWARE AND SOFTWARE OF UNCERTAIN PEDIGREE (SOUP)

Torbjørn Skramstad^{1,2}

¹ Norwegian University of Science and Technology, NTNU, NO-7491 Trondheim, Norway, torbjorn.skramstad@idi.ntnu.no

² Det Norske Veritas Research, Veritasveien 1, NO-1322 Høvik, Norway, torbjorn.skramstad@dnv.com

Abstract – Mission- and safety critical system designers are more and more forced to use a Commercial-Off-The-Shelf (COTS) approach due to more focus on cost and development times, even if COTS components normally are not specifically designed and developed for robust operation. Many safety critical systems have to be assessed or certified by independent organisations. This paper addresses the challenges assessors and certification bodies meet when facing the assessment of such systems.

Keywords – safety, risk analysis, COTS, Operating Systems, Safety Integrity Level, assessment.

I. INTRODUCTION AND BACKGROUND

Software-based systems replace older technologies in safety- and mission-critical applications. Software has found its way into aircraft engine control, railroad interlocking etc. New critical applications are developed, such as automating aspects of surgery, or steering and piloting of automobiles. Software has traditionally been regarded as non safety-critical, because humans using manual backup could take over if it failed. However, increased requirements to speed and volume mean that this fall-back capability is being eroded. Control systems are also becoming more and more complex.

At the same time, due to economical reasons and pressure from the market to reduce development times the use of COTS, and sometimes older proprietary software of uncertain pedigree (SOUP), is increasing. These systems are taking over an increasing amount of safety-critical operational functions in many types of systems. This includes propulsion and steering systems on-board ships, engine control and braking systems on cars, and vital functions in medical devices.

Imagine a large passenger ferry loosing steering control in the heavy populated British channel, or a Liquid Natural Gas carrier loosing propulsion in the narrow Bosphorus sound. Ideally, the control system and the safety system should be independent systems. However, it is becoming more common to integrate these into the same system.

II. WHAT IS A SAFETY RELATED FUNCTION

A safety related function is a designated function that 1) implements the required safety functions necessary to

achieve or maintain a safe state for the (EUC) Equipment Under Control (e.g. a ship or a car); and 2) is intended to achieve, on its own or with other E/E/PE safety-related systems, other technology safety-related systems or external risk reduction facilities, the necessary safety integrity for the required safety functions.

The requirements to safety related functions will typically be: 1) the required reliability of the safety function (how often is it acceptable that the function fails); 2) critical required response time (what is the maximum time from the demand of a function to the successful actuation of the safety related function); 3) manual emergency handling is possible and procedures are available for handling functional failures of the safety related function within the necessary time frame.

III. FUNCTIONAL SAFETY AND ASSESSMENT OF SAFETY CRITICAL SYSTEMS

Safety standards ^[1] ^[2] ^[3] have requirements to the development as well as to the assessment and certification of systems that may endanger its users and the environment when failing. The objective of Functional Safety Assessment is to review whether the combination of safety-related systems and external risk reduction facilities can achieve the overall safety requirements. Safety assessment is normally carried out by personnel who are independent from the design and test team. The level of independence depends on the phase of the life-cycle being assessed, the SIL of the system (see section IV for a definition of SIL), and the system's complexity and/or its novelty. The SIL, or safety integrity level, is a number (1 lowest, 4 highest) indicating the required degree of confidence that the system will meet its specified safety features.

IV. WHAT IS COTS AND SOUP?

In a safety related context typical definitions of COTS/SOUP are: 1) 'Software, the pedigree of which is unknown or uncertain, which could in any way affect the correct operation of a safety-related system' ^[10], 2) 'Commercially available application sold by vendors through public catalogue listings. COTS is not intended to be customised or enhanced' ^[2].

Typical examples of COTS are: Operating Systems, Database Systems, Expert system shells, CASE Tools (Editors, compilers, GUI builders, Test tools), Libraries or other re-usable software, Application oriented packages, e.g. SCADA, Networking and communication software, Utilities (e.g. spreadsheets, word processors) etc. A hardware component may also be a COTS – however, the discussion of assessment of such COTS are outside the scope of this paper.

V. THE DILEMMA OF THE ASSESSOR

The most common safety standards, such as IEC61508^[1], DO-178B^[2], and EN50128^[3], have strict requirements to how software should be developed and tested (for IEC61508 also requirements to hardware). None of these standards explicitly allow the use of COTS (or SOUP) in safety related applications. The reason for this is that one can not demonstrate that the safety requirements to development, testing and maintenance have been met. In addition the configuration status of a COTS is often uncertain. In order to use COTS in subsystems the supplier has to enforce a system architecture that is robust and reliable enough to fulfill requirements, and this has to be demonstrated by risk analyses or similar, such as FMEA, Fault Tree Analysis, Event Tree Analysis, Common Cause Analysis and software HAZOP. Such a robust architecture could be obtained by introducing redundancy by e.g. having diverse COTS, that is, COTS subsystems with the same functionality, but developed by independent teams. For an operating system e.g. this would mean that the supplier would have to supply sufficient evidence for diversity between to diverse OSs and the reliability of each the OSs such that the required SIL level is obtained at system level.

Assessment of COTS software in safety related systems is treated in several papers^{[10][11][14][15]}.

VI. HOW TO DEAL WITH COTS SOFTWARE?

COTS may fail in many different ways depending on the type of COTS. Typical examples are, a) the software component delivers wrong output; b) the COTS function is not carried out sufficiently fast, e.g. the COTS may be “hanging”; and, c) the COTS software writes into some other application’s instruction or data area causing other components and functions to fail. For certain applications and architectures it may be possible to supervise or protect memory through MMS or by calculating checksums regularly. Another alternative can be to use a certified COTS. For some real time operating systems certified versions are available, e.g. VxWorks and OSE. However, these are usually very expensive and it is important to check the context the COTS is certified in,^{[14][15]} (is it on the relevant hardware and software platforms, and is it exactly the same version?).

A few studies have been done in order to compare the robustness of operating systems and other COTS. One of the most known methods for doing such tests is the Ballista testing system^{[5][6]}. The results from these studies show that some commercial operating systems have a significant robustness failure rate. A study for HSE in UK performed by CSE international, UK^[12], concludes under certain circumstances and limitations; *“On the basis of evidence from widespread use, some numeric reliability data, observed reliability growth, the existence of test projects and the limited analysis carried out by this study, it is concluded that “vanilla” Linux would be broadly acceptable for use in safety related applications of SIL 1 and SIL 2 integrity”*. They even state that it might be feasible to certify Linux to SIL3.

Another alternative to assess the COTS is to perform exhaustive testing, but this might be infeasible for large and complex COTS such as an operating system. Sometimes, it is possible to ask the vendor to get sufficient documentation from the development and testing process to fulfill safety standard requirements. Due to the nature of COTS this will normally not be possible, and where this has been tried, we have often been met with silence.

In a recent EU research project, Safe-PC^[4], supported by the IST program under the 5th Framework, the purpose was to develop an architecture based on commercially available PCs and operating systems, and demonstrate through safety assessment its conformance with IEC 61508 SIL1 and SIL2. In this project DNV acted in the role as assessor and certification body in the project.

It will clearly be unacceptable to use a single, non-certified COTS if the consequences of COTS failure is unacceptable. The architecture in Safe-PC is therefore based on a diversity approach with diverse architecture (hardware and software)^[7], the two main components are named PCU and PCX. Studies have been made in order to compare the robustness and diversity of operating systems^{[5][6]}. The results from these studies show that some commercial operating systems have a significant robustness failure rate. In Safe-PC we used the following solution: At Java Virtual machine level both Microsoft OS and Sun OS supports Java. For Safe-PC the simplest hardware configuration that guarantees sufficient diversity was the following: a) PCU HW: Sun Blade 150, b) PCX: Any x86 PC. And OS selection a) PCU OS: Sun Microsystems Solaris 8/9, b) PCX OS: Microsoft Windows 2000. Java VM/Compiler selection: a) PCU Java: Jamaica VM (compiled with Java Compiler from Jamaica), b) PCX Java: Standard Java VM with standard Java compiler^[7].

The final assessment of the Safe-PC was done in the spring of 2004^[8], and it concluded that the concept used could be assessed to fulfill the SIL2 requirements of IEC 61508 for the limited use of the system.

An alternative to the requirements in the relevant standard, adequate statistical data from field use may be used in the assessment. This is the so-called ‘proven-in-use’ argument. For COTS this is hardly possible, since in order to certify to SIL2 a fault-free operation of 120 years is necessary^[12]. Even for widely distributed COTS this will be difficult, mainly due to difficulties related to lack of sufficient configuration status data^{[14] [15]}.

A different approach for handling of COTS in safety-critical applications is based on ‘wrapping’ of the COTS (encapsulating the software component), and is discussed in a Galileo certification study for the European Space Agency^[8], performed by a consortium headed by DNV. In this solution the COTS can be protected from erroneous data from other components, and the rest of the system may be protected from erroneous output from the COTS.

In 2001, the HSE, UK commissioned research into how pre-existing software components may be safely used in safety related systems in a way that complies with the IEC 61508 standard. Two reports resulted from this research^{[10] [11]}. The first report summarizes the evidence that is likely to be available in practice relating to a COTS, while the second report considers how the available evidence can best be used within the framework of the IEC 61508 to support an argument for the SIL achieved by a safety function. In^[10] five different types of assessment that can be carried out on COTS / SOUP; a) process assessment, b) assessment of previous use, c) black-box assessments, d) white-box assessments, and e) third party assessments. The report concludes with proposing that the justification for SOUP / COTS should be based on a safety case approach with a number of additions to an IEC 61508 approach to a) take into account the need for a safety case, b) cope with the presence of COTS throughout the safety life-cycle, including architecture development, risk analysis, software engineering, operation. This is quite similar to the approach used in the Safe-PC assessment^{[7] [8]} where also several assessment methods were applied throughout the safety life-cycle.

VII. SHIP CLASSIFICATION AND CLASS SOCIETIES

Ships and offshore floating installations are approved or certified by classification societies. DNV is one of the leading class societies in the world together with Lloyds Register, ABS, GL, and BV. Such a classification is guaranteeing that the ship or offshore installation is safe to operate. This approval is based on a set of rules, normally proprietary to each classification society. DNV’s rules have requirements to strength, stability etc., and also to control systems. These rules are based on experience systematically gathered over a long period of time. The rules are more prescriptive than the requirements laid down in the most used safety standards^{[1] [2] [3]}. Instead of SIL they use ‘essential functions’ (‘...a system which supports equipment which needs to be in continuous operation for maintaining

the vessel’s manoeuvrability.....’), and ‘important systems’ (‘...a system supporting equipment which need not necessarily be in continuous operation for maintaining the vessel’s manoeuvrability, but which is necessary to maintain the vessel’s main functions.....’) to distinguish between the seriousness of malfunction.

In a recent report^[15] by DNV proposes to work towards applying a risk based methodology for assessing systems with COTS / SOUP. This requires that detailed *functional requirements* to the overall system are provided. The overall functional requirements must be broken down to detailed functional requirements to the sub-modules. Based on the functional requirements it is possible to compare the actual performance specifications of the software and the hardware modules and then decide on the applicability of the modules. The following functional parameters are at least required both on the overall system level and at the COTS module level: a) identification of safe state, b) functional reliability of safety function to go to safe state, c) response time for entering the safe state.

VIII. CONCLUSION

It is obvious that certification companies and classification societies such as DNV will have to develop and impose new and clearer guidelines and rules for how to deal with COTS in safety critical systems. Some of these rules may be based on current best practice, such as found in IEC61508 or EN50128. However, rules for how to deal with systems built mainly of COTS (many suppliers of safety-critical systems have asked to use Window XP as operating system) have to be developed based on current and future research. Or, alternatively: Could non-reliable COTS be allowed if combined with the traditional: Manual override for critical functions? In total, the work anticipated to increase the dependability of software intensive systems in several domains is considered daunting – but it is a risk society must manage.

REFERENCES

- [1] IEC61508 *Functional safety of electrical /electronic/ programmable electronic safety related systems* – Part 1: General requirements, Part 2: Requirements for electrical /electronic/ programmable electronic safety related systems, part 3: Software requirements, IEC 1998.
- [2] DO-178B *Software considerations in airborne systems and equipment certification*, RTCA/EUROCAE, 1987.
- [3] EN50128 *Railway applications – communications, signaling and processing systems – software for railway control and protection systems*, CENELEC March 2001.
- [4] <http://dbs.cordis.lu/fep/cgi/srchidadb?ACTION=D&SESSION=169392005-8->

- [5] N. P. Kropp, P. J. Koopman, D. P. Siewiorek, *Automated Robustness testing of Off-the-Shelf Software Components*, 28th Fault tolerant Computing Symp., June 1998.
- [6] P. Koopman, J. DeVale, *Comparing the robustness of POSIX Operating Systems*, Proceedings of FTCS'99, June 1999.
- [7] T. Skramstad, M. Sjästad, *Assessment Plan for evaluation of Safe-PC to satisfy IEC 61508 for SIL2*, Report No. D6.1, Safe PC: Dependability Enhanced Distributed System and Software Architecture in Safety-Related Applications, Sept. 2003. IST-33387.
- [8] T. Skramstad, M. Sjästad, *Results of assessment of the Safe-PC Concept*, Report No. D6.5, Safe PC: Dependability Enhanced Distributed System and Software Architecture in Safety-Related Applications, July. 2004. IST-33387.
- [9] J-P. Blanquart, G. Bulsa, P. Robert, T. Skramstad, J-L. Terrailon, *Software aspects in the certification of Galileo*, Proceedings V GNSS International Symposium, Sevilla, May 2001.
- [10] P.G. Bishop, R.E Bloomfield, P.K.D. Frome, *Justifying the use of Software of Uncertain Pedigree (SOUP) in safety-related applications*, HSE Report 336, 2001, ISBN: 0 7176 2010 7.
- [11] C. Jones, P.K.D. Frome, P.G. Bishop, *Methods for assessing the safety of safety-related Software of Uncertain Pedigree (SOUP)*, HSE Report 337, 2001, ISBN: 0 7176 2011 5.
- [12] H.R. Pierce, *Preliminary Assessment of Linux for safety related systems*, Research Report 011, 2002, HSE, ISBN: 0 7176 2538 9.
- [13] D. J. Smith, K.G.L. Simpson, *Functional Safety, A Straightforward Guide to IEC 61508 and Related Standards*, Butterworth-Heinemann, 2001, ISBN: 0 7506 5270 5.
- [14] A. Lawrence, *A personal view of using COTS equipment in safety-related applications*, IEE Workshop "Can COTS and SOUP be justified in Safety Applications?", 21 October 2004, London. http://www.iee.org/oncomms/pn/functionalsafety/cots_soup.cfm
- [15] S. Hutchesson, *The Challenges of Providing Assurance Evidence for COTS and Reused Components*, IEE Workshop "Can COTS and SOUP be justified in Safety Applications?", 21 October 2004, London. http://www.iee.org/oncomms/pn/functionalsafety/cots_soup.cfm
- [16] S. Ruud, K. Hovden, P. Martinsen, O.M. Nesvåg, T. Skramstad, K. Fotland, *Guidelines for the application of COTS in Control Systems on ships*, DNV Technical Report 2004-1518, January 2005, (limited availability outside DNV).

(Due to illness Skramstad has to stay at his hotel-room for the whole event in Porto, and could not make the presentation then. Therefore we arranged a video-conference later where he made his presentation. Below is some of the questions asked.)

Ian Peake: I would be interested to see one of the last few slides---the "wrapping" slide---it seems reminiscent of the DFSSMs work from DSSE

Ian Peake: yes, I'm not sure how to quickly explain but just to say that Monash has looked at ways of characterising proper interactions with components

Ian Peake: there is a well known reference to "USS Yorktown" (on RISKS digest) where a Windows NT machine failure resulted in the disablement of a US Navy vessel

Ian Peake: is it possible to explain the distinction between white box/black box and "third-party" assessments in the "five different types of assessment that can be carried out on COTS/SOUP"

- a) process assessment
- b) assessment of previous use
- c) black-box assessment
- d) white-box assessment
- e) third party assessment

Component-Based Context-Dependent Hybrid Property Prediction

Anders Möller^{†*} Ian Peake[‡] Mikael Nolin^{†*} Johan Fredriksson[†] Heinz Schmidt[‡]

[†]MRTC, Mälardalen University, Västerås, Sweden

[‡]Monash University, Melbourne, Australia

*CC Systems, Uppsala, Sweden

E-mail: Anders.Moller@mdh.se

Abstract

Many embedded systems for vehicles and consumer electronics critically depend on efficient, reliable control software, and practical methods for their production. Component-based software engineering for embedded systems is currently gaining ground since variability, reusability, and maintainability are efficiently supported. However, existing tools and methods do not guarantee efficient resource usage in these systems.

We present a method that enables resource-efficient component-based control software by extending hybrid property prediction methods (i.e. combining static and dynamic techniques) to be context-dependent, enabling less pessimistic extra-functional component property predictions and, hence, improved resource utilisation.

1 Introduction

Increasing reliability and efficiency of software intensive dependable embedded control systems is critical [1]. Hence, industry demands practical and accurate engineering approaches to model, predict, and verify both core software functionality and extra-functional aspects of the software.

Component-Based Development (CBD) is successfully practised to achieve enhanced software reuse and maintainability in office/Internet applications. However, in order to be equally successful in the area of embedded control – component technologies have to be resource-constrained and equipped with methods to model and predict extra-functional aspects of the software (e.g. timing and memory consumption).

The key to achieve efficient resource utilisation, on a system level, is to have access to tight and accurate models of the resource needs of the components in the system. One key resource is the CPU, where the resource requirement of a component is Worst-Case Execution Time (WCET). Recent *hybrid* methods for WCET prediction [2] have been proposed which promise a practical approach to gaining tight WCET estimates for traditional, monolithic, programs. However, existing methods for WCET estimation are overly pessimistic in a CBD setting since they are *context-oblivious*, i.e. not explicitly taking into account the current usage-context of each component.

We focus on the problem of efficient resource usage with preserved analysis accuracy of embedded Product Line Architectures (PLAs) [3], like, e.g., control software in vehicles and consumer electronics. In order to facilitate PLAs – software components must be used (and reused) across different hardware platforms and products.

To maximise reuse in these systems, components need to be flexible. Hence, reusable components often include behaviours that are only used in a few configurations. These behaviours cannot easily be removed (e.g., by dead-code elimination), because they are offered by interfaces and cannot be removed by methods based on analysis of components in isolation. Hence, most existing property prediction approaches are overly pessimistic, and, thus, design-for-reuse tends to work against accurate WCET predictions (and, therefore efficient resource utilisation) in existing models and approaches.

In previous work [4] we showed how a component model, custom-made for embedded control-systems [5, 6], can be combined with novel methods for architecture-based, compositional reasoning, modelling, and prediction [7, 8, 9].

In this paper, we propose the use of *context-dependent hybrid property prediction* methods to make efficient use of system resources. We extend the existing hybrid prediction methods by considering the component *usage-context*. We use Dependable Finite State Machines (DFSMs) to facilitate compositional, architecture-based, reasoning about system properties based on context-dependent component properties and the structure of the component assembly [9].

We illustrate our approach using the SaveComp Component Model [5], and an adaptive cruise controller implementation [6]. In this paper, we limit our context-dependent predictions to component WCET in order to reach efficient processor utilisation. Nevertheless, our approach is generally applicable to other extra-functional properties, such as memory usage, assuming that properties are compositional.

2 Background

The systems considered in this paper are categorised as dependable complex distributed computational-intense embedded real-time systems running in, e.g., vehicles or customer electronics. In these business segments methods to reuse software cross different hardware platforms and product families/versions are inquired [1]. Hence, component-based software engineering is gaining more and more interest.

2.1 The SaveComp Component Model

The SaveComp Component Model (SaveCCM) [5, 6] is a component model for control software development. SaveCCM provides three main architectural elements: *components*, *switches*, and *component assemblies*. A component is not allowed to have any dependencies to other components, or other external software (e.g. the operating system), except the visible dependencies through its input- and output-ports. A switch provides means for conditional transfer of data and/or triggering between components. Component assemblies allow composite objects to be defined, and make it possible to form aggregate components from groups of components, switches, and assemblies enabling different levels of abstraction.

The interface of an architectural element is defined by a set of ports, i.e. points of interaction between the element and its environment. SaveCCM distinguish between input- and output ports, and there are two complementary aspects of ports: the data that can be transferred via the port and the triggering of component executions. The graphical syntax of SaveCCM (see Figure 1), is similar to UML 2.0 component diagrams, but with additions to distinguish between the different types of ports. Principally, the SaveCCM syntax uses \triangleright to represent triggering ports (i.e. the control flow) and \square to represent data ports (see Figure 1).

Example: An Adaptive Cruise Controller

In this section we present an Adaptive Cruise Controller (ACC) prototype, implemented in SaveCCM [6] (see Figure 1).

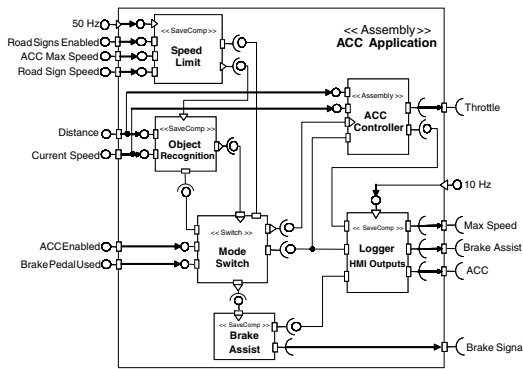


Figure 1. An Adaptive Cruise Controller described in the SaveCCM graphical modelling language

The ACC extends the regular cruise controller (used in most cars) in that it helps the driver keep a safe distance to a preceding vehicle, autonomously changes the speed depending on the speed limit regulations, and helps the driver to slam the brake in extreme situations.

The application is based on four components, one switch, and one component assembly. The assembly (Figure 2 (a)) is

in turn implemented using two assemblies (Figure 2 (b)). Furthermore, the application has two different trigger frequencies, 10 Hz and 50 Hz. Logging and HMI output activities execute with the lower rate, and control related functionality at the higher rate.

For a detailed presentation of the ACC application functionality, we refer to [6].

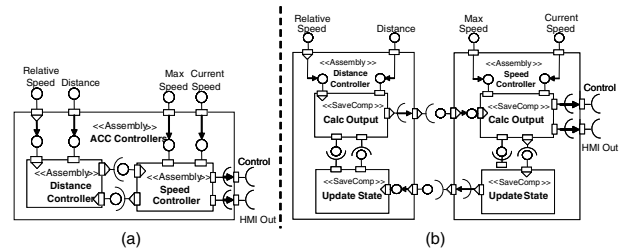


Figure 2. The ACC control assembly (a), and the implementation of the feedback controller (b)

2.2 WCET Prediction Methods

WCET bounds may be obtained via *static* (model-based), *dynamic* (measurement-based) or more recently proposed *hybrid* methods.

Static methods promise safe WCET estimates but critically depend on time-intensive construction and evaluation of models of underlying platforms [10]. Moreover, as hardware becomes increasingly complex (processors with pipelines and caches) the variance between typical and worst-case performance is growing significantly. As a result, static WCET analysis tends to produce increasing pessimism in the calculated WCET bound [11].

Dynamic methods are often cheaper to construct, but with little guarantee that acquired measurements can be used directly to derive WCET upper bounds (see Figure 3). Realising run-time measurements by trying all possible input data combinations (i.e. the complete value space) is typically not feasible.

Hybrid methods overcome some deficiencies by combining static and dynamic methods. Static analysis is used to limit the input value-space, and run-time measurements are used to calculate upper bound WCET predictions.

However, existing methods for WCET predictions are not suitable for component-based development. Existing methods take a whole-of-system approach and thus produce overly pessimistic predictions for components. Effectively these methods are only capable of producing a single portable WCET estimate for a component, whereas, in fact, the true WCET of the component may be dependent on the context in which it is later deployed.

In, e.g., [2], static (model checking) approaches are used to generate test-cases which are in turn used to generate WCET observations for functions (strictly program segments), or even basic blocks within functions. These include the use of

heuristics and model checkers to generate the necessary test harnesses which exercise all possible paths leading to instrumented points within the code around “primitive” elements, then incorporating the results into conventional predictions. However, since such techniques do not make use of mechanisms to qualify the usage context of a given function f , it is not possible to reuse a predicted WCET for f in a different context. Indeed it is not clear from the content of several published papers whether it is possible to discriminate between different implicit contexts for f within the original program, or whether the same WCET for f is used regardless of context.

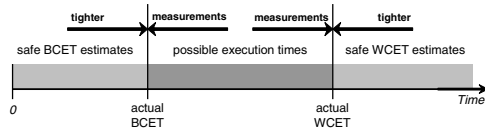


Figure 3. Best-Case Execution Time (BCET) and Worst-Case Execution Time (WCET) predictions

2.3 Context-Independent Analysis

Consider the ACC controller assembly Distance Controller from Figure 2 (a), which for the purposes of this illustration we treat as a component (see Figure 4). The Distance Controller asynchronously interoperates with its environment via four ports: three input ports and one output port.

Two input ports are for Relative Speed and Distance to the car in front, and a pair of input/output ports are for communicating with the Speed Controller. The Distance input port contains information about the distance to the vehicle in front together with an enabled/disabled boolean value. The Relative Speed input port has in the same way an integer representing the relative speed compared to the vehicle in front and a boolean enabled/disabled port. The ports related to Speed Controller communication are not considered in this example. Hence, in Figure 4 only the Distance and Relative Speed inputs are visualised.

When used in a product line of vehicles, some vehicles will be sold with the full functionality of the ACC, whereas others will be sold with a simpler, traditional cruise control function. However, this component is able to provide both functions, and will hence be deployed in both vehicle types (i.e. in two different contexts). By *disabling* both the Relative Speed and Distance ports, the Adaptive Cruise Controller (ACC) becomes a more traditional Cruise Controller (CC), by not taking into account the distance to the vehicle in front. Figure 4 visualises these two component modes by separating the internal control flow of the component.

For many components, a WCET bound, even if tight, occurs rarely, and only in certain situations. In many contexts the execution time may be much less. For example, assume that the WCET for the Distance Controller is 4ms. This might only occur rarely, in ACC mode. In the less demanding

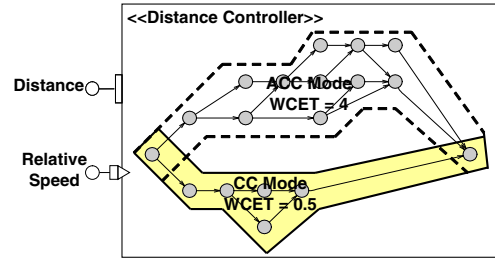


Figure 4. Context-dependent control-flow for the Distance Controller

CC-mode, the execution might never be more than say 0.5ms. Ideally, then, the WCET of a given component should always be qualified by stating the context in which it is valid (see Figure 4).

3 Dependent Finite State Machines

To model context, we make use of the formal notion of a *protocol type* from Dependent Finite State Machines (DFSMs).

DFSMs are parameterised dynamic formal models for components. They extend communicating finite state machines and model components’ abstract implementation (abstraction) and deployment context as rigorous parameters of a components interface specification. The abstraction parameters cater for dynamic specialisations and variations in product lines, the requires parameters capture properties and variation in different deployment contexts. Network of DFSMs represent parameterised product line architectures. Actualisation of parameters are compositional as well as the incremented assembly of components into such networks. DFSMs are particularly suitable for architecture-based reasoning about extra-functional properties.

An abstract example of what can be modelled with DFSMs is the context dependent behaviour of CompC in Figure 5. The behaviour of CompC is limited to the requested services from CompA and CompB, i.e. the values on CompC input ports are limited to the valid output from CompA and CompB. Hence, the execution behaviour of CompC can be described as a function of the critical services required by CompA and CompB.

DFSMs have their basis in trace languages, which can be regarded as an extension of regular languages.

Regular languages promises a useful trade-off between precision and computational feasibility suitable for solving the problem identified above. DFSMs describe the allowed interactions between a given component and its environment (i.e. protocol types) as well as how the component itself is implemented. DFSMs also provide ways of talking about the structure of, and relationships between, those protocols by modelling a network of interface-protocol dependencies (see Figure 5).

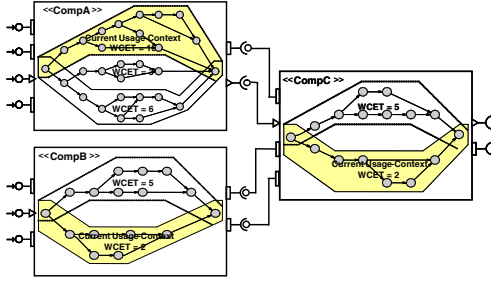


Figure 5. An abstract example of a context-dependent control-flow

3.1 Protocol Types

A *protocol type* is a formalisation of the protocol acceptable to a component, defined as a regular language.

For a concrete example of DFSMs, consider the protocol type of *Distance Controller* (see Section 2.3), taking into account parameters (and their respective abbreviations) *Distance.Value* (*Dist*), *Distance.Enabled* (*DistE*), *RelativeSpeed.Value* (*Speed*), and *Speed.Enabled* (*SpeedE*).

When *Distance Controller* is triggered, it reads values from single element buffers corresponding to its input ports. The allowed values of the relevant ports are determined by their types.

As is standard in behavioural contract specifications, valid calls to *Distance Controller* can be represented by a regular language. Consider representing a call to a component by a string of symbols consisting of the component’s name and its actual parameters – simple binary encodings of the values present in port buffers as actual parameters.

For example, triggering the *Distance* with port assignments (*Dist*=01100100, *DistE*=1, *Speed*=00001010, *SpeedE*=1), would be represented by the string `invocdistance 01100100 1 00001010 1`.

A *regular language* L giving the set of all valid calls based on the above scheme can be defined using regular expressions as follows:

$$L ::= \text{invocdistance } \text{Dist } \text{DistE } \text{Speed } \text{SpeedE}$$

where (assuming a simple unsigned 8-bit representation for integer values):

Dist = integer (8-bit)
 DistE = bit
 Speed = integer (8-bit)
 SpeedE = bit
 integer = bitstring
 bitstring = bit⁸
 bit = 0|1

Most importantly for our purposes, a protocol type can be used not only to describe the protocol acceptable to a component, but also the way a component is used in a given context.

Consider a component C whose protocol type is defined as the language L . Then it is possible to consider, for a given deployment context of the component, another protocol L' which describes the way C is used in that context. The sub-

protocol L' must conform to L , that is, L' must be a *sublanguage* of L (contain only strings of L).

For example, if *Distance* were deployed into an environment where *Distance* and *Relative Speed* inputs were permanently disabled (i.e. CC-mode), the context could be described by the language L_{CC} (a sublanguage of L):

$$L_{CC} ::= \text{invocdistance integer 0 integer 0}$$

For the remainder of this paper, when we use the “context”, it should be understood to include the notion of protocol subtype.

Finally, the notion of subprotocols leads to the following critical observation. For an upper-bound property such as WCET, the WCET cannot go up when context is restricted. Formally, consider a component C whose protocol type is L and two usages of C where one is strictly narrower than the other: formally L and L' where $L' \subseteq L$, with corresponding WCETs W_L and $W_{L'}$. Then the inequality $W_{L'} \leq W_L$ must be satisfied. This property forms the basis of context-dependent property predictions.

4 Context-Dependent Property Prediction

Accurate architectural-based reasoning about system performance is enabled by exploiting *context-dependent component property models*. Such models make it possible to formally and accurately capture the variation in properties that may occur depending on the way the components are used.

The above properties of protocol types can be exploited to derive context-dependent property predictions. Our approach conceptually separates engineer-defined static design-time configurations of the components (e.g. components differently configured to suit different product lines) from the deployment-context (i.e. component relations in the current assembly).

The usage condition, or context, of a specific component usage, instantiation or deployment, is formalised in terms of a protocol type.

Context-dependent property models are collections of *guarded* component properties: a value representing a property of some component is always qualified by pairing it with the context (i.e. a protocol type) in which it is valid.

4.1 Static Design-Time Contexts

So, for example, the fact that the WCET for *Distance Controller* is 4ms is expressed by the pair $(4ms, L)$, where L is as given above.

For accuracy, additional pairs may be added, refining the property model in other useful contexts. For maximum accuracy, search the pairs for the narrowest matching context with the lowest WCET. For maximum efficiency, it is possible to impose a lattice ordering over pairs based on the subprotocol relationship between guards so that not all pairs need to be considered for a given context.

For example, to express that, when neither the distance nor relative speed inputs are enabled, the WCET for the *Distance Controller* is much lower, e.g., 0.5ms, we simply add the pair $(0.5ms, L_{CC})$, where L_{CC} is as given above.

Practically speaking, implementing a subprotocol test amounts to a test for regular language inclusion, which can be implemented relatively cheaply using finite automata.

Each component in an assembly can in the same way be equipped with context-dependent information about WCET (see Figure 5) that can be reused for accurate design-time property predictions. For any component with a suitable set of guarded property pairs, in a given deployment context, a property value can be predicted at design time. A deployment context includes information about how a given component is used, particularly the deployment environment model (i.e. static design-time configurations of the components).

In this way system-level properties of the application can be derived from the context-dependent component properties. In the same way, component assemblies (i.e. hierarchical groupings of components) can be accurately predicted by propagating the usage conditions down through the assemblies.

Analyses are performed component-wise, deriving for each component a set of guarded WCETs, as needed to provide the desired level of accuracy. Where components contain no branches, a single, general, usage context may suffice. In other cases, more detailed characterisations of usage context may be required, but only up to the required level of accuracy.

4.2 Deployment Contexts

To further tighten the property predictions, one is not limited to engineer-defined static design-time configurations of the components. Additionally, to facilitate more fine-grained architectural-based reasoning, properties may be further constrained by considering the effect of connected components on the deployment context (see Figure 5).

Usage conditions can be fed into the network as constraints that propagate through the network and eliminate execution alternatives. The process can be likened to dead-code elimination, except that it is performed at the level of the property model – component code itself is not affected (see Figure 4).

For adequate accuracy and performance, this approach relies on two assumptions: (i), that correct (but not perfectly tight) upper bounds are acceptable; and, (ii), that component types with widely varying WCETs (see Figure 6) can be considered to be the union of a small set of sub-behaviours induced by non-overlapping contexts (see Figure 4), where each sub-behaviour can be accurately characterised by a single correct (and accurate) property bound.

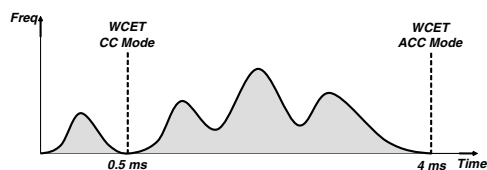


Figure 6. A conceptual context-dependent WCET graph for the Distance Controller

4.3 Context-Dependency in SaveCCM

To fully achieve the above in SaveCCM, the component protocol semantics, presented in [4] must to be extended in

order to sufficiently detailed characterise the conditions leading to the WCET variation. While in general such problems are equivalent to the halting problem, existing WCET techniques already take into account context by identifying e.g., simplified domain models for variables including sub-ranges.

Modelling SaveCCM sufficiently to propagate interesting contexts from higher architectural levels is additionally complex because it includes component scheduling and asynchronous [12], buffered data flow between tasks of different frequency. Nevertheless, it should be feasible to derive such a semantics, since there are well understood extensions to automata to model relevant primitives, such as concurrency and variables.

5 Context-Dependent Hybrid Prediction

As stated in section 2, existing hybrid analysis techniques are based on a whole-of-system approach which is unsuitable for component-based analysis, making it difficult to reuse parts of an WCET analysis for individual components with sufficient accuracy.

In this section we present a hybrid property prediction method that uses DFSMs to generate context-dependent instrumentation data. We then employ the framework-based software component monitoring approach described in [13] to measure the execution behaviour using the obtained instrumentation data.

This approach is especially beneficial in embedded product line architectures, since it facilitates early predictions of system-level properties (that can be used to guide the developer choosing inexpensive hardware) and, also, a pragmatic way to obtain extra-functional performance properties.

The resulting monitoring information, as well as the instrumentation data, is stored together with the component in the repository. Since, in a product line architecture the same application might execute on different target hardware. In this case, the engineers will simply reuse the instrumentation data and just measure the component performance on the new target hardware.

5.1 Development Process

Using the context-dependent hybrid property prediction approach together with hybrid schedulability analysis [14, 15], an attractive development process can be obtained (see Figure 7). The architectural model of the component assembly is used for architectural-based analysis, but also for test and instrumentation:

Test and Instrumentation: Design-time context-dependent analysis (as described in Section 4) is used in order to achieve early assessments about extra-functional component properties [9]. This approach yields formal context models which may be used to derive run-time measurements of the component properties as for existing hybrid approaches. Critically, however, both observed and predicted properties are reusable with confidence since they are combined with context models which state in which contexts the properties are valid. Hence, we can obtain sufficiently accurate estimations of the components extra-functional properties.

System Deployment: These properties are then used to perform tight schedulability analysis using hybrid techniques [4, 15]. Also, the system generator can be augmented to automatically insert instrumentation code to perform run-time monitoring [13] of the deployed system. And, during run-time, the unused processing time can be reclaimed using variable component performance modes as described in [14].

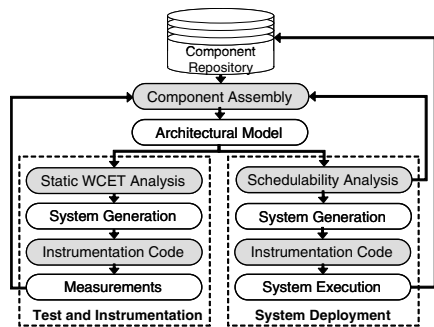


Figure 7. Component-based development using context-dependent predictions

6 Conclusions and Future Work

We present a compositional method to increase resource-efficiency in component-based control systems for product line architectures by extending hybrid property prediction methods to be context-dependent.

We introduce component usage-profiles, where extra-functional properties depend on the current usage-context, in order to obtain tighter property predictions and, hence, more efficient resource-utilisation.

As for future work, we plan to extend our theories to work with other performance properties, such as, e.g., memory or reliability [7]. We also plan to extend hybrid resource reclamation methods (such as [14, 15]) using probability distributions of the components execution time in order to predict the quality-of-service level that can be performed in the background alongside the hard real-time schedule.

Acknowledgements

We would like to thank Prof. Hans Hansson and Dr. Jukka Mäki-Turja for valuable feedback on early versions of this article.

The Monash authors acknowledge the assistance of Mr Yauheni Veryha and Dr Peter Bort for their collaboration on the Extra-Functional Consistency and Prediction (eCAP) project, seeking to apply DFSM-based techniques to a commercial WCET tool.

References

[1] A. Möller, J. Fröberg, and M. Nolin. Industrial Requirements on Component Technologies for Embedded Systems. In *Proceedings of the 7th International Symposium on Component-Based Software Engineering*, May 2004. Edinburgh, Scotland.

[2] I. Wenzel, B. Rieder, R. Kirner, and P. Puschner. Automatic timing model generation by cfg partitioning and model checking. In *Proc. Conference on Design, Automation, and Test in Europe*, March 2005.

[3] P. Clements and L. Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley, 2001. ISBN 0-201-70332-7.

[4] I. Peake, A. Möller, and H. W. Schmidt. Modelling and Verification of Dependable Component-Based Vehicular Control-System Architectures. Submitted for publication, available as MRTC report ISSN 1404-3041 ISRN MDH-MRTC-180/2005-1-SE, Mälardalen University, May 2005.

[5] H. Hansson, M. Åkerholm, I. Crnkovic, and M. Törngren. SaveCCM - a Component Model for Safety-Critical Real-Time Systems. In *Proceedings of 30th Euromicro Conference, Special Session Component Models for Dependable Systems*, September 2004.

[6] M. Åkerholm, A. Möller, H. Hansson, and M. Nolin. Towards a Dependable Component Technology for Embedded System Applications. In *Proceedings of the 10th IEEE International Workshop on Object-oriented Real-Time Dependable Systems (WORDS05)*, February 2005. Sedona, Arizona, USA.

[7] R. Reussner, H. Schmidt, and I. Poernomo. Reliability Prediction for Component-Based Software Architectures. *Journal of Systems and Software*, 66(3):241–252, 2003.

[8] H. W. Schmidt, I. Peake, J. Xie, I. Thomas, B. Krämer, A. Fay, and P. Bort. Modelling Predictable Component-Based Distributed Control Architectures. In *Proceedings of the Ninth IEEE International Workshop on Object-Oriented Real-Time Dependable Systems*, January 2004. Anacapri, Italy.

[9] H W Schmidt, B J Krämer, I Poernomo, and R Reussner. Predictable Component Architectures Using Dependent Finite State Machines. *Lecture Notes in Computer Science*, 2941:310–324, 2004.

[10] J. Engblom, A. Ermedahl, M. Sjödin, J. Gustafsson, and H. Hansson. Worst-case execution-time analysis for embedded real-time systems. *Software Tools for Technology Transfer*, 14, 2001.

[11] R. Kirner and P. Puschner. Discussion of Misconceptions about Worst-Case Execution-Time Analysis. In *Proceedings 3rd Euromicro International Workshop on WCET Analysis*, July 2003. Porto, Portugal.

[12] A. Wall, K. Sandström, J. Mäki-Turja, and C. Norström. Verifying Temporal Constraints on Data in Multi-Rate Transactions. In *Proceedings of the 7th International Workshop on Real-Time Computing and Applications Symposium*, December 2000. Cheju Island, South Korea.

[13] D. Sundmark, A. Möller, and M. Nolin. Monitored Software Components – A Novel Software Engineering Approach –. In *Proceedings of the 11th Asia-Pacific Software Engineering Conference, Workshop on Software Architectures and Component Technologies*, November 2004. Pusan, Korea.

[14] J. Fredriksson, M. Åkerholm, K. Sandström, and R. Dobrin. Attaining flexible real-time systems by bringing together component technologies and real-time systems theory. In *Proceedings of the 29th Euromicro Conference, CBSE Track*, September 2003. Belek, Turkey.

[15] J. Mäki-Turja, K. Hänninen, and M. Nolin. Efficient Development of Real-Time Systems Using Hybrid Scheduling. In *Proceedings of the International conference on Embedded Systems and Applications*, June 2005. Las Vegas, Nevada, USA.

Transcript from the discussion

Q(ES): What if you don't know what the structure of the aggregate is, if the aggregation patterns are restricted this gives us some chance to say something about behaviour.

A: Yes, this is interesting..

Q(Shi): you mention the reuse of components, what about reuse of test case

A: You probably have to be very careful. There could be minor differences

Q: modelling of concurrency is a problem. Your new approach to deal with this?

A: In order to be able to cope with component-based systems you need to know that your system is extendable, and can be used to model concurrency. Effectively, if you add an independence relation to a set of symbols the independence relation will tell you something more about the structure, you get a partial ordering of symbols. You can end up with a state machine, Petri net model, with further analysis possible. You're not ruling out modelling these kind of things.

Important to model the semantics of communication between tasks, and times you have to wait, what guarantee do we have that a message gets there.

Amund's provoking opinion: "guaranteeing anything is useless.. it's all a matter of probabilities. Bad things happen" .. This caused some laughter and agitation in the audience.

Q(AS): How about the external context, interrupts and other tasks/events interfering. The probability of being in cache would be affected etc..

Ian: This is another line of investigation, which needs to be looked at.

Ian: Isn't it a bit scary that we're moving towards CPU architectures not being fully documented in terms of processing capabilities, this makes it hard to model and predict behaviour.

Maybe most of all a problem when dealing with consumer products, Windows OS etc.

Presentation Slides

PORTO 2005

Workshop on Software intensive dependable
embedded systems

Organized by
Amund Skavhaug and Erwin Schoitsh



www.ntnu.no

What the workshop was all about

- For our Ercim WG: to increase visibility
- For participants:
 - Meet new people
 - Visit a "new" conference at the same time
 - Have their ideas discussed
 - Discuss the ideas of others
- To link with EUROMICRO



www.ntnu.no

Something was different from others...

Draft proceedings was made available before the event.

A brief agenda with titles was given out to all participants at the event.

Final proceedings will include edited transcript of discussion from the presentations.



www.ntnu.no

Brief facts

- 10 papers reviewed and accepted
- 2 long half-days were used for discussions and presentations of these
- A web-site for the workshop was made
- The papers were also presented as posters, quite visibly and well-placed for all to see during the whole event.
- A number of positive remarks were given for the format of the event, by participants and organizers.



www.ntnu.no

Some more things done

- The final publication was to be edited to include the main-points from the discussions. (This have been very difficult.)
- Questions asked, and the answers given, were given to the participants for review before inclusion. (They have been put on a WIKI-type web site we created)
- The final publication will exist both as bound paper and as pdf on the web.
- A video-meeting was arranged with participation of 7. 5 in Trondheim, one in Australia and one in England.



www.ntnu.no

Suggestions for next event

- We were welcomed back !
- Dubrovnik in Croatia will attract people
- We will have some of the similar things done for practical reasons, i.e. payment to Euromicro conference(s) for the participants.
- We need a more clearly stated goal of what shall be achieved in the workshop !
- Call for papers will begin in Desember 2005 ?



www.ntnu.no

ERCIM **DECOS**

ARC (Austrian Research Centers) Seibersdorf Research Information Technologies
ERCIM Working Group on Dependable Embedded Software-Intensive Systems

EU Framework Program 6: PRIORITY [2], Proj.nr. 511764
 [Information Society Technologies, Embedded Systems Unit, FP6/Call 2]

Integrated Project: DECOS
 Dependable Embedded Components and Systems

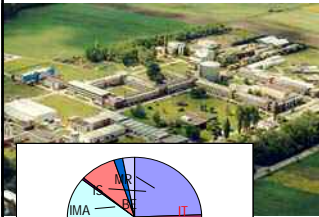
Erwin Schoitsch
 Euromicro, Porto, Portugal
 August 31st, 2005

Euromicro, Porto, 31 August 2005 **seibersdorf research** Slide 1

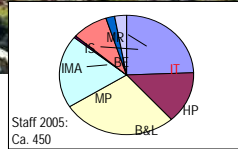
ERCIM **DECOS**

Seibersdorf research – Divisions

Largest enterprise of ARC – Austrian Research Centers: Austria's largest independent, application-oriented research organisation (10 sites, 750 staff)



• Information Technologies
 • Health Physics
 • Biogenetics, Natural Resources
 • Life Sciences
 • Materials & Production Engineering
 • Integrated Microsystems Austria
 • Biomedical Engineering
 • Intelligent Infrastructures and Space Applications
 • Media Research Studios Salzburg

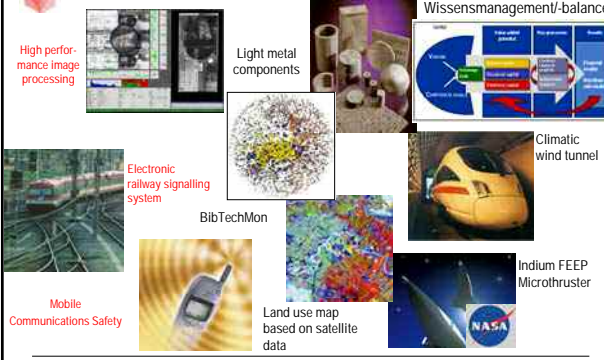


Staff 2005: Ca. 450

Euromicro, Porto, 31 August 2005 **seibersdorf research**

ERCIM **DECOS**

ARC Products (examples)



High performance image processing
 Light metal components
 Electronic railway signalling system
 BibTechMon
 Mobile Communications Safety
 Land use map based on satellite data
 Indium FEFP Microthruster
 Climatic wind tunnel
 Wissensmanagement/-balance

Euromicro, Porto, 31 August 2005 **seibersdorf research** Slide 3

ERCIM **DECOS**

Smart Systems for

Computer Vision
 • Optical high-performance inspection, e.g. bank notes
 • Video surveillance for public safety and road traffic
 • Development of vision-based automotive systems for safety and comfort

Telecommunications & EMC Technology
 • EMC-compliant design
 • Mobile communications safety
 • Mobile radio channel simulators
 • Quantum cryptography

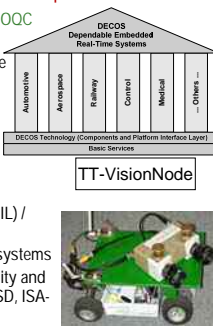
Ambient Intelligence
 • SW/HW Development of dependable real-time systems
 • System safety analyses (RAMSS) and software testing (V&V)
 • Software Engineering

Euromicro, Porto, 31 August 2005 **seibersdorf research** Slide 4

ERCIM **DECOS**

IT - Dependable Embedded Systems Group

- Co-ordinator of EU Integrated Projects **DECOS**, **SECOOC**
- TT-VisionNode (SensorNode)
 - Integration of Image Processing and Dependable Controls, Smart Cameras and Sensors
- Accredited V&V Lab (EN ISO/IEC 17025)
- Research Topics
 - Methodology & tools for dependable embedded components and systems
 - Model based V & V of components & systems
 - Host-target testing with Hardware-in-the-loop (HIL) / Software-in-the-loop (SIL)
 - RAMSS/Hazard analyses for component based systems
 - European Projects and Networks on Dependability and Software Process Management (ENCRESS, AMSD, ISA-EuNet, SPIRE, OLOS, ACRuDA, ESPITI, ...)

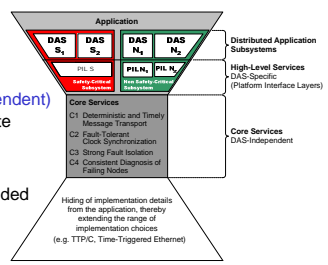


Euromicro, Porto, 31 August 2005 **seibersdorf research** Slide 5

ERCIM **DECOS**

DECOS Basics

Objective:
 Development of fundamental (domain and technology independent) enabling technologies to facilitate paradigm shift from federated to integrated design of dependable real-time embedded systems



Application: DAS S₁, DAS S₂, DAS N₁, DAS N₂
 Distributed Application Subsystems
 High-Level Services: DAS-Specific (Platform Interface Layers)
 Core Services: DAS-Independent
 Core Services: C1: Deterministic and Timely Message Transport; C2: Fault-Tolerant Clock Synchronization; C3: Strong Fault Isolation; C4: Consistent Diagnosis of Failing Nodes
 Hiding of implementation details from the application, thereby extending the range of implementation choices (e.g. TTP/C, Time-Triggered Ethernet)

Project Facts
 Start: July 1st, 2004, Duration: 3 Years, Budget: 14.3 Mio €, EU Funding: 9 Mio €

Euromicro, Porto, 31 August 2005 **seibersdorf research** Slide 6

ERCIM **DECOS**

Why Integrated Architectures – Automotive Example :

- From mechanics to electronics and software – 80 ECUs, one per function and per supplier?
- Increase of electronics from 25% to >40%
- Innovations 80 – 90% based on electronics and software
- 55% of car failures caused by electronics, cabling/connectors and software (GI conference Oct. 2003 – DC postponed “X-by-wire”) – Liability !!!

seibersdorf research
EU Collaborator for Autotech Research Center

Slide 7

ERCIM **DECOS**

DECOS Motivation

Facilitate the systematic design & deployment of “integrated” electronic subsystems in embedded systems through:

- **Electronic Hardware Cost Reduction** (fewer ECU's, cables, connectors)
- **Enhanced Dependability by Design** (clear partitioning of safety-critical and non safety-critical subsystems by design)
- **Reduced Development Costs** (modular certification, reuse of software components, structured integration for communication & computational elements)
- **Diagnosis and Maintenance** (diagnosis of transient and intermittent component failures)
- **Intellectual Property (IP) Protection**

seibersdorf research
EU Collaborator for Autotech Research Center

Slide 8

ERCIM **DECOS**

Applications are composed of subsystems of different criticality

e.g. crash-prevention system, engine control system, door lock system

Federation: advantage=encapsulation, IPR protection
disadvantage=cost, space, power consumption, complexity in cabling, connectors, buses

Integration: advantage=cost, space, power consumption, SW-only implementation (IPR), dependability benefit fewer connectors etc,
disadvantage: more design complexity, highest SIL level for all components?

Requirements for Integrated Architectures: No interference between “jobs”(SW-components) in

- time (predictable allocation of time slots to jobs in DAS's of nodes)
- space (encapsulated execution environment, etc.)

seibersdorf research
EU Collaborator for Autotech Research Center

Slide 9

ERCIM **DECOS**

DECOS provides

- predictable allocation of time slots to jobs in DAS's of nodes (TT core)
- space (configurability, composability of mixed-criticality applications)
 - ♦ encapsulated execution environment,
 - ♦ virtual communication links and gateways (jobs to behave as if physically separated),
 - ♦ fault tolerance layer (functional, value and time domain),
 - ♦ partitioned OS in HW, software controlled (diagnosis, fault tolerance)
 - ♦ diagnostic service (identifying faulty components, intermittent and transient faults and Heisenbugs by job statistics) (supports SW-IPR protection by transparent “black box” integration)

Allocation: conforms to IEC 61508 (allocation of safety functions to system components, component-based modular safety case): many feasible assignments may be possible – validation (see G. Weissenbacher)

seibersdorf research
EU Collaborator for Autotech Research Center

Slide 10

ERCIM **DECOS**

IEC 61508

seibersdorf research
EU Collaborator for Autotech Research Center

Slide 11

ERCIM **DECOS**

DECOS will provide

- A hardware-software base (a software-supported hardware solution for hardware-enabled partitioning and software error checking)
- Software services (“High level Services” to build appl. on top of core services)
- Prototype tools for design, development and validation/verification and simplification of certification of applications based on integrated DECOS architecture, especially for deployment (assignment of jobs of DAS's to nodes (=hardware unit). Nodes are the basic Fault Containment Region (FCR) and basic safety critical HW target platform based on (any) **basic core services** (TTP/C, TT-Ethernet, Layered FlexRay,...) providing:
 - predictable transport messages
 - fault-tolerant clock synchronization of global time
 - Fault/error isolation and tolerance
 - communication fault diagnosis and deterministic recovery


seibersdorf research
EU Collaborator for Autotech Research Center

Slide 12

ERCIM **DECOS**

DECOS Partners (19)

- Industrial Partners:**
Audi Electronics Venture, Airbus, EADS, Infineon, TTTech, Fiat, Profactor, Hella, Liebherr, Thales, Esterel
- Research Centers:**
ARC Seibersdorf (Co-ordinator), SP Swedish Test. & Res. Inst.
- Universities:**
TU Vienna, TU Darmstadt, TU Hamburg-Harburg, Uni Kassel, Uni Kiel, Uni Budapest



Euromicro, Porto, 31 August 2005 **seibersdorf research** Slide 13

ERCIM **DECOS**

DECOS Subprojects

- ♦ SP 1: Architecture Design (TU Darmstadt + TU Vienna)
- ♦ SP 2: Component Design and Implementation (TTTech)
- ♦ SP 3: Silicon Infrastructure (Infineon)
- ♦ SP 4: Validation and Certification (ARCS)
- ♦ SP 5: Application Automotive (Audi)
- ♦ SP 6: Application Aerospace (Airbus)
- ♦ SP 7: Application Control (Profactor)
- ♦ SP 8: Training, Dissemination and Standardization (ARCS), Policy and Gender Issues
- ♦ SP 9: IP Management and Assessment (ARCS)

Euromicro, Porto, 31 August 2005 **seibersdorf research** Slide 14

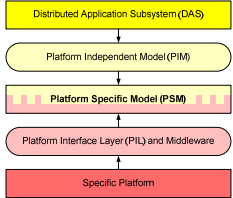
ERCIM **DECOS**

Subproject 1 – Architecture Design (Design Methods)

To reduce mental complexity: (nearly) independent subsystems with precisely defined LIFs (Linkage Interfaces), internals hidden

DECOS: Technology invariant software interfaces, application design and development

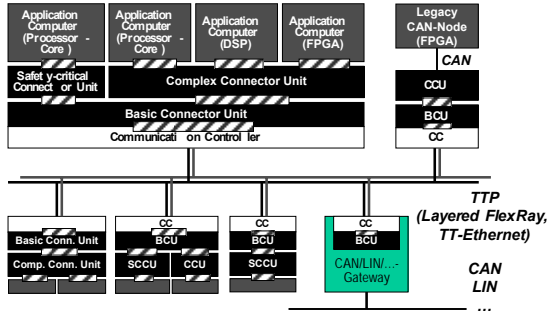
- ♦ Distributed Application Subsystem (DAS) modelling
- ♦ Platform Independent Model (PIM)
- ♦ Platform Specific Model (PSM)
- ♦ Hardware-Software Integration
- ♦ Platform Interface Layer (PIL)
- ♦ Middleware Services



Euromicro, Porto, 31 August 2005 **seibersdorf research** Slide 15

ERCIM **DECOS**

DECOS System Architektur




Euromicro, Porto, 31 August 2005 **seibersdorf research** Slide 16

ERCIM **DECOS**

Subproject 2 – Component Design and Implementation

Encapsulation / Diagnosis / FT-Layer

- ♦ WP 2.1 Encapsulated Execution Environment (partition OS)
- ♦ WP 2.2 Virtual Communication Links incl. Gateways
- ♦ WP 2.3 Diagnostic Services
- ♦ WP 2.4 Optimized FT-Layer

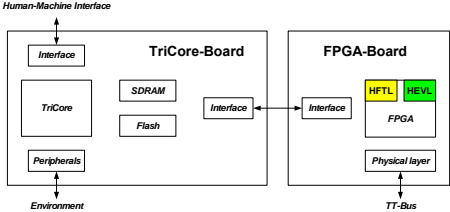


Euromicro, Porto, 31 August 2005 **seibersdorf research** Slide 17

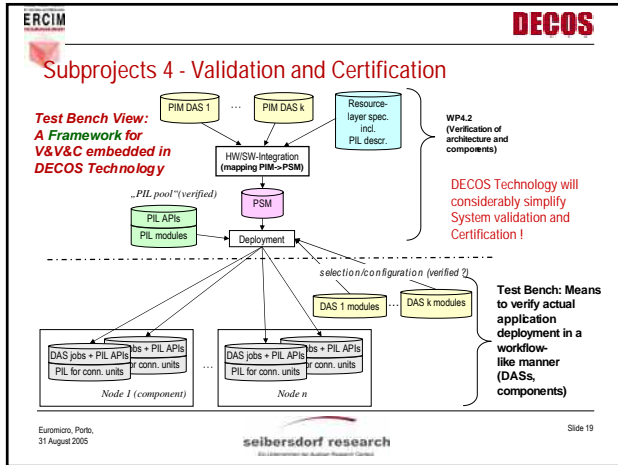
ERCIM **DECOS**

Subproject 3 – Silicon Infrastructure (Middleware)

Fault-Tolerance Layer (HFTL) / Event Layer (HEVL)



Euromicro, Porto, 31 August 2005 **seibersdorf research** Slide 18



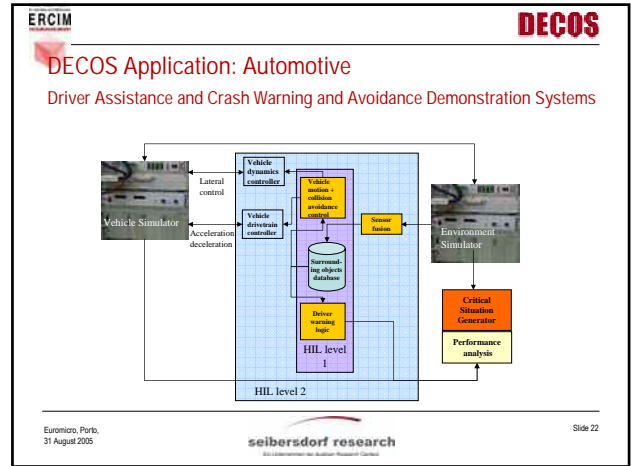
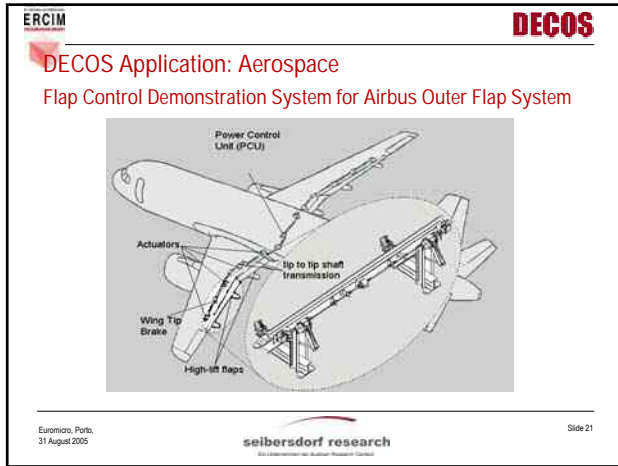
ERCIM **DECOS**

DECOS Application Areas

- Automotive
- Aerospæ
- Railways
- Industrial Control
- Medical Systems
- Autonomous Systems

DECOS will develop structured guidelines for domain-independent and technology independent integration.

EuroMicro, Porto, 31 August 2005 **seibersdorf research** Slide 20



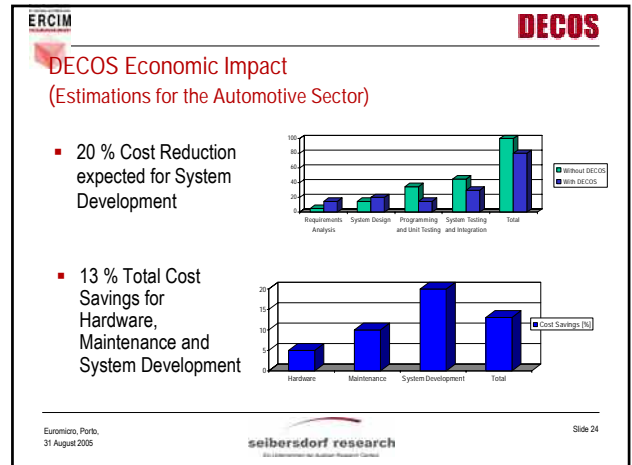
ERCIM **DECOS**

DECOS Application: Industrial Control

Vibration Control Demonstration System for Nano Imprinting Machines

Objectives:
 Suppression of critical vibrations in high-end nano-imprinting machines for next-generation Sensors, Microoptics, Bio- and Nanotechnology
Long Term Vision: Structural Control.

EuroMicro, Porto, 31 August 2005 **seibersdorf research** Slide 23



ERCIM **DECOS**

Economic Impact of Dependable Embedded Systems and DECOS Technology (examples)

- Electronics in Cars: 170 billion € HW, 100 b € SW (2010), Europe in leading position
- Driver assistance systems: 2-3 b € 2007, increasing by 50% within a few years
- Aerospace industry revenues: 265 b € total, 70 b € civil, Europe in leading position
- European mechanical Engineering Industry: 353 b € turnover, 32% of innovations DES-based (HW, SW) rising to 40%
- SMEs in active safety systems electronics consultation and know-how transfer services in the validation and certification market: 5 b € (2010), annual growth about 30%
- Tool and component manufacturer: annual increase expected in DES market by 20% - 40 %, depending on sector.
- **Environmental, Quality and Safety of Life and Employment pact expected to be considerable large by means of DES -mass deployment**

Euromicro, Porto, 31 August 2005 Slide 25

seibersdorf research
an IBM Business Partner

ERCIM **DECOS**


Thank You For Your Kind Attention

Roadmaps available at <https://rami.jrc.it/roadmaps/amsd>
 DECOS project: <http://www.decos.at>
 ARC-sr, IT: www.smart-systems.at

Euromicro, Porto, 31 August 2005 Slide 26

seibersdorf research
an IBM Business Partner

ERCIM **DECOS**



a

a

Euromicro, Porto, 31 August 2005 Slide 27

seibersdorf research
an IBM Business Partner

Improving the Performance of Embedded Superscalar Microprocessors by Adding Partial Pipelines Huibin Shi

Improving the Performance of Embedded Superscalar Microprocessors by Adding Partial Pipelines

Huibin Shi¹, Chris Bailey¹, Glenn Farrall², Neil Hastie² and Sam Jenkins²

¹ Department of Computer Science, University of York, York, YO10 5DD, U.K.
² TriCore Architecture, Infineon Technologies UK Ltd (Bristol), Hunts Ground Road, Stoke Gifford, Bristol, BS34 8HP, UK

¹ {shi, chris}@cs.york.ac.uk ² {Glenn.Farrall, Neil.Hastie, Sam.Jenkins}@infineon.com

This research was supported by Infineon Technologies UK Ltd (Bristol)

ERCIM Workshop 2005 Porto, Portugal 31st Aug/1st Sep 2005 THE UNIVERSITY OF YORK

Improving the Performance of Embedded Superscalar Microprocessors by Adding Partial Pipelines Huibin Shi

Main Topics

- Motivations and the Partial Pipeline
- TriCore™ 2.0 Microarchitecture and Models
- TriCore 2.0 Instruction Set Simulation and Profiling
- Block Code Simulation and the Performance Comparison
- Future Work

ERCIM Workshop 2005 Porto, Portugal 31st Aug/1st Sep 2005 THE UNIVERSITY OF YORK

Improving the Performance of Embedded Superscalar Microprocessors by Adding Partial Pipelines Huibin Shi

Motivation

- Save the chip space and improve the performance
- Reduce the hardware implementation complexity of a pipeline.
- A partial pipeline can execute a subset of instructions executable by a standard pipeline of its type.
- A partial pipeline is simpler than a standard pipeline
- Reduce the design space

ERCIM Workshop 2005 Porto, Portugal 31st Aug/1st Sep 2005 THE UNIVERSITY OF YORK

Improving the Performance of Embedded Superscalar Microprocessors by Adding Partial Pipelines Huibin Shi

Research Stages at High Level

ERCIM Workshop 2005 Porto, Portugal 31st Aug/1st Sep 2005 THE UNIVERSITY OF YORK

Improving the Performance of Embedded Superscalar Microprocessors by Adding Partial Pipelines Huibin Shi

TriCore 2.0 Microarchitecture

- Three pipelines and one instruction for one pipeline only :

- Some instructions have both 16- and 32-bit versions, e.g. add and add16
- Varied instruction execution latencies

ERCIM Workshop 2005 Porto, Portugal 31st Aug/1st Sep 2005 THE UNIVERSITY OF YORK

Improving the Performance of Embedded Superscalar Microprocessors by Adding Partial Pipelines Huibin Shi

Standard 3-Pipeline Base Microarchitecture Model

- Three pipelines:

- One cycle latency for all instructions

ERCIM Workshop 2005 Porto, Portugal 31st Aug/1st Sep 2005 THE UNIVERSITY OF YORK

Improving the Performance of Embedded Superscalar Microprocessors by Adding Partial Pipelines Hulbin Shi

Standard 4-Pipeline Microarchitecture Model by Duplicating the IP Pipeline

- Four pipelines:

Duplicated IP pipeline →

ERCIM workshop 2005 Porto, Portugal 31st Aug/1st Sep 2005 THE UNIVERSITY OF YORK

Improving the Performance of Embedded Superscalar Microprocessors by Adding Partial Pipelines Hulbin Shi

Special 4-Pipeline Microarchitecture Models by Adding the Partial IP Pipeline

- Special four pipelines:

- Adding Partial IP pipelines for a subset of IP instructions.
- Two groups of partial IP pipelines, one with and one without mul & div operations.

ERCIM workshop 2005 Porto, Portugal 31st Aug/1st Sep 2005 THE UNIVERSITY OF YORK

Improving the Performance of Embedded Superscalar Microprocessors by Adding Partial Pipelines Hulbin Shi

EEMBC Benchmark Suite

- EEMBC: embedded microprocessor benchmark consortium
- EEMBC automotive/industrial embedded microprocessor benchmark suite
- It consists of 16 benchmarks, regarded as a whole program
- Source benchmark programs in C programming language
- Benchmarks are compiled into the TriCore 2.0 assembly code

ERCIM workshop 2005 Porto, Portugal 31st Aug/1st Sep 2005 THE UNIVERSITY OF YORK

Improving the Performance of Embedded Superscalar Microprocessors by Adding Partial Pipelines Hulbin Shi

TriCore 2.0 Instruction Set Simulation

ERCIM workshop 2005 Porto, Portugal 31st Aug/1st Sep 2005 THE UNIVERSITY OF YORK

Improving the Performance of Embedded Superscalar Microprocessors by Adding Partial Pipelines Hulbin Shi

Instruction Execution Trace Profiling

- Analyse the instruction execution trace of each benchmark
- For each execution trace file, main simulation part is considered
- The simulation initialisation and house keeping parts are ignored.
- Look at results of all 16 benchmarks all together.

ERCIM workshop 2005 Porto, Portugal 31st Aug/1st Sep 2005 THE UNIVERSITY OF YORK

Improving the Performance of Embedded Superscalar Microprocessors by Adding Partial Pipelines Hulbin Shi

Computing the Weighted Execution Probability of Each Instruction

$$OverallP_i = \frac{\sum_{j=1}^{16} T_{ij}}{\sum_{i=1}^m \sum_{j=1}^{16} T_{ij}} \quad \dots (1)$$

Where: OverallP_i: the overall execution probability of the i-th instruction
T_{ij}: the total number of execution times of the i-th instruction in the j-th benchmark, i.e. total appearance times in the execution trace of the j-th benchmark.
m: total instructions in the TriCore 2.0 instruction set.

ERCIM workshop 2005 Porto, Portugal 31st Aug/1st Sep 2005 THE UNIVERSITY OF YORK

Improving the Performance of Embedded Superscalar Microprocessors by Adding Partial Pipelines Huibin Shi

Rules of Selecting IP Instructions for Partial IP Pipelines

- Select IP instructions with high execution probabilities
- Create two categories of IP instructions, one with and the other without MUL/DIV instructions, each with 4 groups.
- Find the group with the optimum performance improvement.
- Excluding jump instructions

ERCIM workshop 2005 Porto, Portugal 31st Aug/1st Sep 2005 THE UNIVERSITY of YORK

Improving the Performance of Embedded Superscalar Microprocessors by Adding Partial Pipelines Huibin Shi

Groups of IP Instructions with High Execution Probabilities

Opcode	Operation	Freq. (probability)	Group (Without mul/div operations)	Group (With mul/div operations)
add (16)	addition	7.70%	A	E
sha (16)	arithmetic shift	6.50%	A	E
jlt	cond. Jump	2.37%		
mul (16)	multiplication	2.12%		E
sub (16)	subtraction	2.05%	A	E
madd	multiply add	1.34%		E
msub	multiply sub	0.99%		E
mov (16)	move	0.95%	B	F
jz (16)	jump if zero	0.84%		
jne (16)	cond. Jump	0.51%		
dextr	extract data	0.43%	B	F
sh (16)	shift	0.39%	B	F
...
mov.u	move unsigned	0.02%	D	H
lt.u	less than unsigned	0.02%	D	H

ERCIM workshop 2005 Porto, Portugal 31st Aug/1st Sep 2005 THE UNIVERSITY of YORK

Improving the Performance of Embedded Superscalar Microprocessors by Adding Partial Pipelines Huibin Shi

Microarchitectures in Our Study

Architecture names	Number of pipelines			Partial IP Pipeline	Group of instructions for partial IP
	IP	LS(Load/Store)	LP(LOOP)		
4-pipeline (A)	1	1	1	1	A
4-pipeline (B)	1	1	1	1	A,B
4-pipeline (C)	1	1	1	1	A,B,C
4-pipeline (D)	1	1	1	1	A,B,C,D
4-pipeline (E)	1	1	1	1	E
4-pipeline (F)	1	1	1	1	E,F
4-pipeline (G)	1	1	1	1	E,F,G
4-pipeline (H)	1	1	1	1	E,F,G,H
3-pipeline (base)	1	1	1		
4-pipeline	2	1	1		

ERCIM workshop 2005 Porto, Portugal 31st Aug/1st Sep 2005 THE UNIVERSITY of YORK

Improving the Performance of Embedded Superscalar Microprocessors by Adding Partial Pipelines Huibin Shi

Sampled Block of Code and Their Execution Frequencies

- Instruction execution trace analysis generates the execution frequency of each instruction and each block
- Select blocks of instructions executed at least once in the execution trace for the new simulation, i.e. the block code simulation

ERCIM workshop 2005 Porto, Portugal 31st Aug/1st Sep 2005 THE UNIVERSITY of YORK

Improving the Performance of Embedded Superscalar Microprocessors by Adding Partial Pipelines Huibin Shi

Reconfigurable Block Code Simulation

- Simulate and graphically represent the execution of each basic block, and can report total number of instructions, I_b and cycles, C_b executed of the block.
- Performance contribution results of each block in a benchmark: total instructions executed, $I_b * F_b$ and cycles executed, $C_b * F_b$.
- Results of each benchmark: summation of results of each basic block, total instructions and cycles, $I_j = \sum I_b * F_b$ and $C_j = \sum C_b * F_b$

ERCIM workshop 2005 Porto, Portugal 31st Aug/1st Sep 2005 THE UNIVERSITY of YORK

Improving the Performance of Embedded Superscalar Microprocessors by Adding Partial Pipelines Huibin Shi

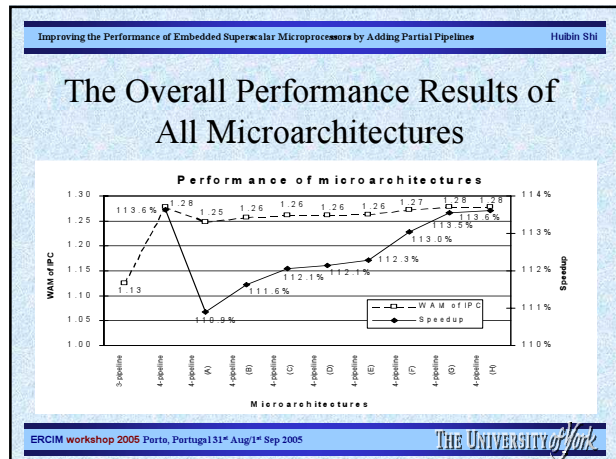
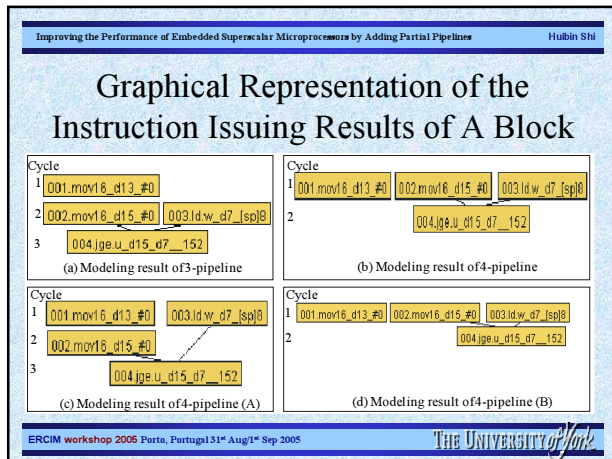
Computing the Performance of Each Microarchitecture

$$WAMofIPC = \frac{\sum_{j=1}^{16} I_j}{\sum_{j=1}^{16} C_j} \dots\dots (2)$$

Where: I_j = total instructions of j-th benchmark,
 C_j = total cycles of j-th benchmark.

$$Speedup = \frac{WAMofIPC (enhanced)}{WAMofIPC (base)} \dots\dots (3)$$

ERCIM workshop 2005 Porto, Portugal 31st Aug/1st Sep 2005 THE UNIVERSITY of YORK



- Improving the Performance of Embedded Superscalar Microprocessors by Adding Partial Pipelines Huibin Shi
- ### Factors Affecting the Performance Results
- One set of EEMBC benchmark assembly code is used for all simulations
 - It was compiled with the optimisation towards the base 3-pipeline microarchitecture
 - Results of the base 3-pipeline microarchitecture are relatively optimum. Results of others are not.
- ERCIM workshop 2005 Porto, Portugal 31st Aug/1st Sep 2005 THE UNIVERSITY of York

- Improving the Performance of Embedded Superscalar Microprocessors by Adding Partial Pipelines Huibin Shi
- ### Future Work
- Simulate microarchitectures with real instruction latencies.
 - Use a different compiler without optimisations towards the base 3-pipeline microarchitecture.
 - Use different the benchmark suite.
 - Study why three instructions, add, sha and sub, get high execution probabilities.
- ERCIM workshop 2005 Porto, Portugal 31st Aug/1st Sep 2005 THE UNIVERSITY of York

- Improving the Performance of Embedded Superscalar Microprocessors by Adding Partial Pipelines Huibin Shi
- ### Summary
- Introduced motivations and the concept of a partial pipeline
 - Presented TriCore 2.0 microarchitecture and models
 - Covered TriCore 2.0 instruction set simulation and profiling
 - Discussed block code simulation and the performance comparison
 - Briefed future work
- ERCIM workshop 2005 Porto, Portugal 31st Aug/1st Sep 2005 THE UNIVERSITY of York

Improving the Performance of Embedded Superscalar Microprocessors by Adding Partial Pipelines Huibin Shi

Thank You!

ERCIM workshop 2005 Porto, Portugal 31st Aug/1st Sep 2005 THE UNIVERSITY of York

The PISA Architecture: A Viable Platform for the Superscalar Execution of Statically Scheduled Stack Code

Soyeb Alli, Chris Bailey
University of York
York, UK
Soyeb.alli@cs.york.ac.uk

The PISA Architecture

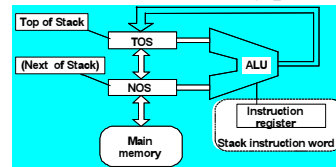
- Partially Implicit Stack Architecture

Static Scheduling in the GPR Architecture

- Example: Loop Unrolling

LD R1, R2, R3	LD R1, R2, R3
ADD R4, R1, R5	LD R7, R8, R9
ST R4, R6(0)	ADD R4, R1, R5
LD R7, R8, R9	ADD R10, R7, R11
ADD R10, R7, R11	ST R4, R6(0)
ST R10, R12(0)	ST R10, R12(0)
UNSCHEDULED	SCHEDULED

Stack Architecture Type Model



- Does not contain visible general purpose registers
- Operands in the stack with implied addressing mode
- ALU can take two operands simultaneously and write the result back to TOS, e.g. 'add'

Stack Processor

- Not possible to schedule code in same way due to implicit addressing
- Example: $C = A + B$ in the stack architecture



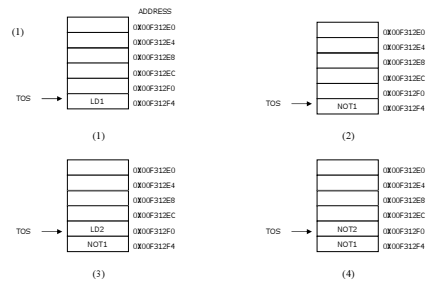
About Stack Addressing

- Instruction consumes TOS and NOS items and writes back to NOS
- NOS becomes new TOS
- IF NOS address (i.e. destination address) specified then can determine source operand addresses
- Stack code can now be scheduled statically

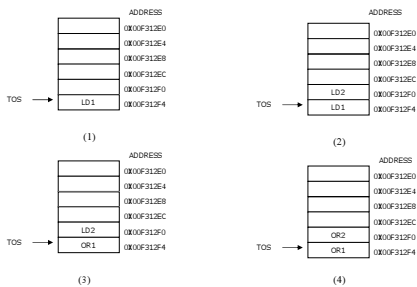
Example

LD1 0	LD1 TOS
NOT1	LD2 ((TOS) -4)
LD2	NOT1 TOS
NOT2	NOT2 ((TOS) -4)
UNSCHEDULED	SCHEDULED with re-naming instructions

Unscheduled



Scheduled



Advantages

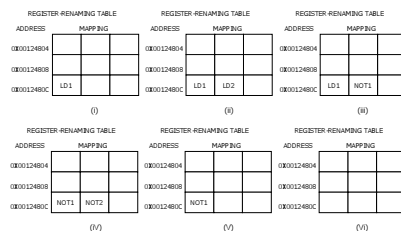
- Minimise stalls by scheduling instructions in delay slots
- Can also queue multiple writes to same stack address by exploiting destroy-on-read properties of stack

Example: Unrolled loop

```


LD1 1
LD2 2
NOT1
NOT2
ST1 1
ST2 2
    
```

Example: Unrolled loop With Register Renaming



Conclusions

- Advantages:
 - Can schedule stack code statically to minimise pipeline stalls
 - Multiple writes can be queued to same stack address
- Disadvantages:
 - Need to specify destination address in instruction
 - Must calculate destination addresses during compilation
 - time consuming





SCTL: a StateChart Transformation Language for Test Sets Reduction

Nicolas GUEIFI, Benoît RIES
University of Luxembourg

ERCIM Workshop, Porto, Portugal
Aug. 31 – Sept. 1, 2005

Context and Problematic


- Work Context
 - SE2C
 - SESAME Project
 - Industrial Partner
 - Kind of systems : Embedded Systems (car, plane, PDA, etc.)
 - Activity: Specification-based Testing
- Problematic
 - Exhaustive test often impossible
 - **Difficult selection** of test cases to perform

2

Objectives


- Aim
 - Improve the test phase
- Approach
 - Create an abstract specification that allows generating reduced test sets.
- Formalisms
 - Behaviors specifications : statecharts
 - Abstraction: model transformation language



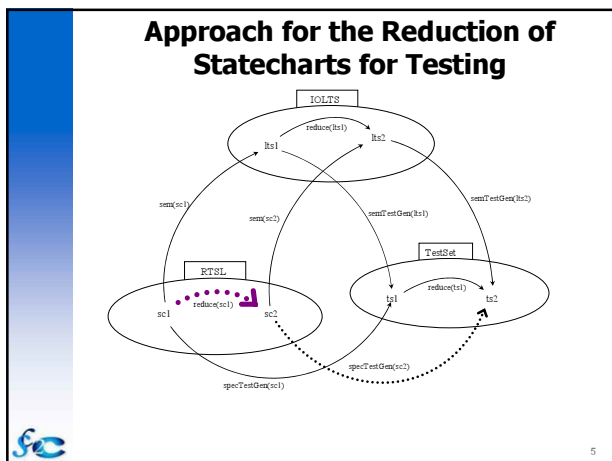
3

Plan

- Approach: Statecharts Reductions for Test
- RTSL: a Statechart Model
- SCTL: a Statechart Transformation Language
- SCTT: Prototype and Case Study
- Conclusion




4



Statecharts Reduction for Testing

"Abstraction is the identification of important aspects of a phenomenon while ignoring its details." [ghezzi02]

- Different Kinds of Abstractions
 - Structural Abstraction
 - Functional Abstraction
 - Communication Abstraction
 - Data Abstraction
 - Temporal Abstraction



6

Plan

- Approach: Statecharts Reductions for Test
- RTSL: a Statechart Model
- SCTL: a Statechart Transformation Language
- SCTT: Prototype and Case Study
- Conclusion



7

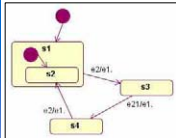
RTSL – a Model of Statecharts

- Objectives
 - Select an abstract syntax
 - Defined with statecharts concepts
 - Having formally defined properties
 - Having a relation of states accessibility
 - Select a semantics
 - Defined in a formal framework of specification-based testing



8

RTSL – Abstract Syntax



- $S_i = \{\text{root}, s1, s2, s3, s4\}$
- $E_i = \{e2, e21\}$
- $A_i = \{e1\}$
- $T_i = \{(s1, e2, e1, s3), (s3, e21, e1, s4), (s4, e2, e1, s2)\}$

A statechart sc_i is a 7-uplet
 $sc_i = (S_i, E_i, A_i, T_i, \text{sub}_i, \text{default}_i, \text{type}_i)$

Where

- S_i : states
- E_i : events
- A_i : actions
- $T_i \subseteq S_i \times (E_i \cup \epsilon) \times (A_i \cup \epsilon) \times S_i$: transitions
- $\text{sub}_i \subseteq S_i \rightarrow P(S_i)$: substates function
- $\text{type}_i \subseteq S_i \rightarrow \{\text{AND}, \text{OR}, \text{BASIC}\}$: type of states
- $\text{default}_i \subseteq S_i \rightarrow S_i$: default state of OR-states
- sub_i
 - $\text{sub}_i(\text{root}) = \{s1, s3, s4\}$
 - $\text{sub}_i(s1) = \{s2\}$
- type_i
 - $\text{type}_i(\text{root}) = \text{OR}$
 - $\text{type}_i(s1) = \text{OR}$
 - $\text{type}_i(s2) = \text{BASIC}$
 - $\text{type}_i(s3) = \text{BASIC}$
 - $\text{type}_i(s4) = \text{BASIC}$
- default_i
 - $\text{default}_i(\text{root}) = s1$
 - $\text{default}_i(s1) = s2$



9

RTSL – Syntax: Accessibility Relation

- Some states possibly inaccessible
- Test Context: useless to keep inaccessible states
- Accessibility relation at the level of the statecharts syntax

$$\rightarrow_i: S_i \times (E_i \times A_i)^* \times S_i$$

$s_0 \xrightarrow{\gamma} s_1$ means that state s_1 may be reached from state s_0 following the transitions which sequence of events/actions is γ



10

RTSL - Semantics

- Semantics given in input/output labeled transition systems (IOLTS)
- Existing Transformation [Gnesi01]
 - Hierarchical Automates (HA) \Rightarrow IOLTS
- Syntax of HA close to RTSL
- Transformation Algorithm
 - Statechart RTSL \Rightarrow HA



11

Plan

- Approach: Statecharts Reductions for Test
- RTSL: a Statechart Model
- SCTL: a Statechart Transformation Language
- SCTT: Prototype and Case Study
- Conclusion



12

SCTL – a statecharts transformation language

- 8 statecharts transformations belonging to 2 categories of abstraction
 - Structural
 - Delete inaccessible states
 - Abstract a state
 - Extract a state
 - Delete a state
 - Delete a transition
 - Modify a default state
 - Functional
 - Modify initial states of a statechart
 - Ignore an event

SCTL - Semantics: *delTrans*

- *delTrans* transformation deletes a transition
- $delTrans \subseteq SC \times SC \times T$ is a relation defined by the assertion:

$$\forall (sc_i, sc_j, t). (sc_i \in SC \wedge sc_j \in SC \wedge t \in T_i \Rightarrow ((sc_i, sc_j, t) \in delTrans_i \Leftrightarrow$$

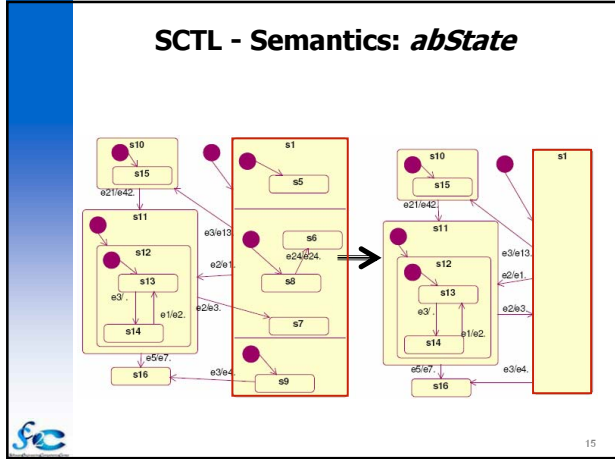
$$T_j = \{t' \mid t' \in T_i \wedge t' \neq t\}$$

$$\wedge S_j = \{s' \mid s' \in S_i\}$$

$$\wedge sub_j = \{(s', stSet') \mid sub_i(s') = stSet'\}$$

$$\wedge type_j = \{(s', ty') \mid type_i(s') = ty'\}$$

$$\wedge default_j = \{(s', s'') \mid default_i(s') = s''\}))$$



SCTL - Semantics: *abState* (cntd)

- *abState* transformation abstract the internal behavior of a state.

$$\forall (sc_i, sc_j, s). (sc_i \in SC \wedge sc_j \in SC \wedge s \in S_i \wedge s \neq root_i \Rightarrow ((sc_i, sc_j, s) \in abState \Leftrightarrow$$

$$S_j = \{s' \mid s' \in S_i \wedge s' \notin sub_i^+(s)\}$$

$$\wedge T_j = \{t' \mid t' \in T_i \wedge src_i(t') \notin sub_i^+(s) \wedge tgt_i(t') \notin sub_i^+(s)\}$$

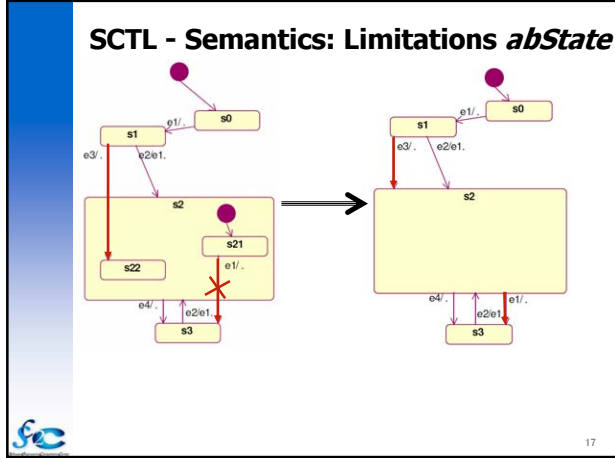
$$\cup \{(s', e, a, s) \mid \exists s''. ((s', e, a, s'') \in T_i \wedge s' \in S_j \wedge s'' \in sub_i^+(s))\}$$

$$\cup \{(s, e, a, s'') \mid \exists s'. ((s', e, a, s'') \in T_i \wedge s' \in sub_i^+(s) \wedge s'' \in S_j)\}$$

$$\wedge sub_j = \{(s', stSet') \mid s' \in S_j \wedge sub_i(s') = stSet' \wedge s' \neq s\} \cup \{(s, \emptyset)\}$$

$$\wedge type_j = \{(s', ty') \mid s' \in S_j \wedge type_i(s') = ty' \wedge s' \neq s\} \cup \{(s, BASE)\}$$

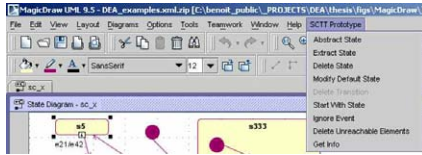
$$\wedge default_j = \{(s', s'') \mid s' \in S_j \wedge s' \neq s \wedge default_i(s') = s''\}))$$



Plan

- Approach: Statecharts Reductions for Test
- RTSL: a Statechart Model
- SCTL: a Statechart Transformation Language
- SCTT: Prototype and Case Study
- Conclusion

SCTT - Prototype



- Prototype integrated to the UML MagicDraw tool
- ≈ 2 000 lines of codes
- Implement the 8 SCTL transformations
- Transformations are performed on the selected element and the active opened diagram
- Case study using this prototype

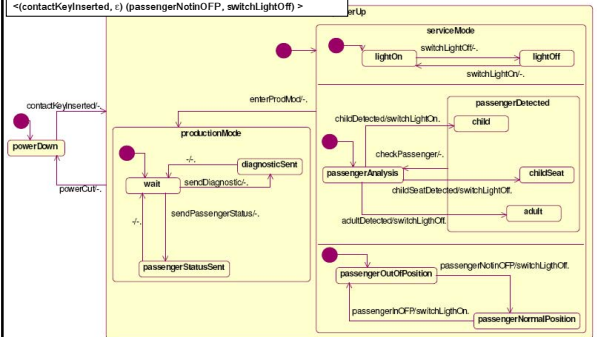


19

Case Study 1/3

Subset of traces :

```
<(contactKeyInserted, c) (enterProdMod, c) (sendPassengerStatus, c)>
<(contactKeyInserted, c) (enterProdMod, c) (sendDiagnostic, c)>
<(contactKeyInserted, c) (childSeatDetected, switchLightOn) >
<(contactKeyInserted, c) (passengerNotInOFF, switchLightOff) >
```



Case Study 2/3

Subset of traces before transformation:

```
<(contactKeyInserted, c) (enterProdMod, c) (sendPassengerStatus, c)>
<(contactKeyInserted, c) (enterProdMod, c) (sendDiagnostic, c)>
<(contactKeyInserted, c) (childSeatDetected, switchLightOn) >
<(contactKeyInserted, c) (passengerNotInOFF, switchLightOff) >
```

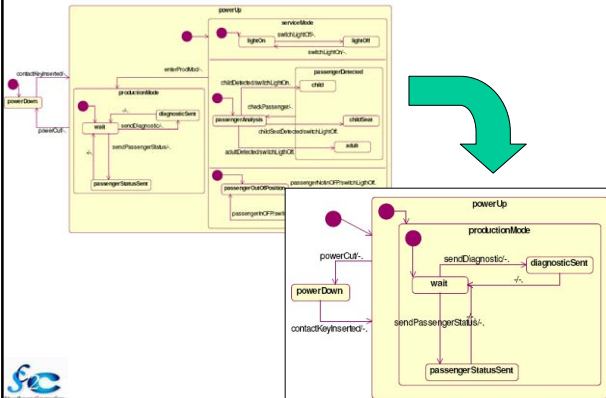


After transformation *exState* de l'état *serviceMode*:
 <(childSeatDetected, switchLightOn) >
 <(passengerNotInOFF, switchLightOff) >



21

Case Study 3/3



Conclusion

- Adaptation of a model of statecharts (RTSL)
 - Abstract syntax
 - Semantics (RTSL \Rightarrow HA \Rightarrow IOLTS)
- Definition of a transformation language (SCTL) for RTSL statecharts
 - SCTL transformations defined with the elements of RTSL abstract syntax
 - 8 transformations based on structural and functional abstractions
- Perspectives
 - Validation of SCTL transformations regarding the semantics of RTSL statecharts
 - Take into account real-time concepts for safety-critical systems
 - Extension of the RTSL syntax
 - Definition of additional transformations to SCTL



23

SCTL - Syntax

- SCTL Grammar in BNF:

```

<transfo> ::= <struct_transfo>
           | <funct_transfo>
           | <transfo> . <transfo>

<struct_transfo> ::= delUnreach( <statechart> , <statechart> )
                  | exState( <statechart> , <statechart> , <state> )
                  | abState( <statechart> , <statechart> , <state> )
                  | delTrans( <statechart> , <statechart> , <trans> )
                  | ...

<funct_transfo> ::= startWith( <statechart> , <statechart> , <state> )
                  | ignoreEvt( <statechart> , <statechart> , <event> )
    
```

- Examples:

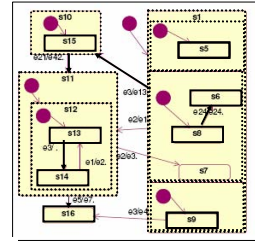
- delTrans(SC0, SC1, T5).delUnreach(SC1, SC2)
- ignoreEvent(SC7, SC8, E2).delUnreach(SC8, SC2)
- exState(SC3, SC6, S21)
- ...



25

Statecharts – Basic Concepts

- States Hierarchy (substates, superstates)
- Basic state, AND-state, OR-state
- Default states
- Initial states
- Transition
→ event/action
- Configuration
- Basic Configuration



26

Test Method For Real-Time Embedded systems

O.Koné, J.P.Bodeveix, R.Bouaziz
CNRS/IRIT, Univ. Paul Sabatier - Toulouse

Porto, 30/08-03/09 2005



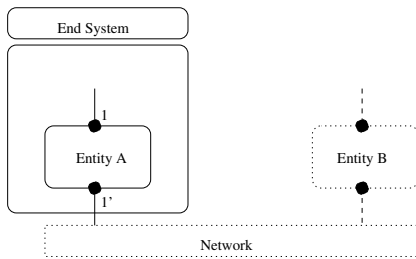
Outline

- 1 Testing vs Dependability
 - Testing Embedded Systems
 - Testing Real-Time
 - Complexity due to Real-Time
- 2 Our approach
 - Main features
 - Symbolic Reachability
 - Local Exploration vs Dependability requirement
- 3 Illustration
 - The specification model
 - The dependability requirement
 - The computed tests patterns



Testing embedded systems

Check system dependability before deployment



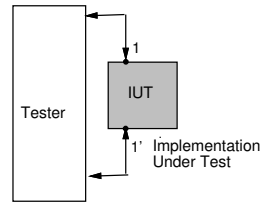
- The system interacts with the environment through gates or **ports** (1 and 1')
- We consider **black box** systems
Implementation: hardware software component chip



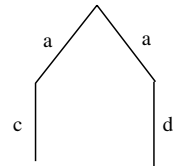
Methodology: Black box / Functional testing

We model and test only the functional behaviour

Test architecture



Spec. model

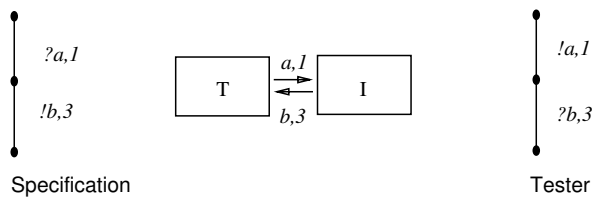


- We model (and test) the *observable functional* behaviour
No structural testing, ...
- *Objective*: to automate tests computation from the system specification model



Testing real-time

How to compute tests from real-time specification ?

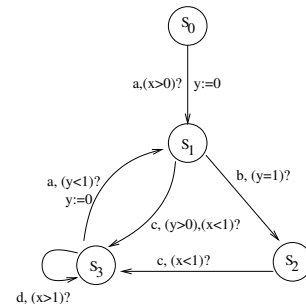


- Manage time with clocks:
 $x:=0$ (reset); $x=2, x<3$ (constraints)
- Test case (test pattern / test scenario):
 $(x:=0) \rightarrow (x=1, !a; y:=0) \rightarrow (y=3, ?b) \rightarrow \text{PASS.}$
- How to *compute test patterns* from a real-time model / specification ?



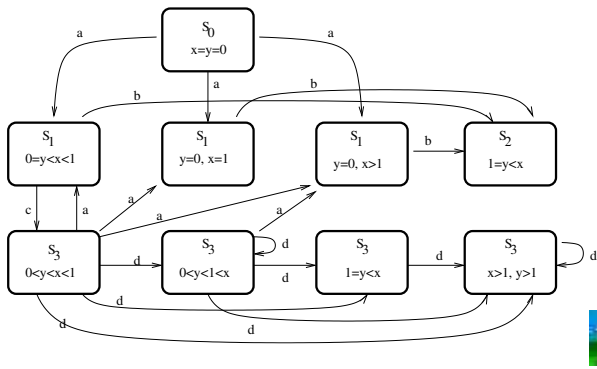
Complexity involved with Real-Time

A sample timed automaton



Behaviour computation involves combinatory explosion

The behaviour (region) graph of the timed automaton



Our approach

Main features: We Combine Symbolic and Local Analysis

Objectives :

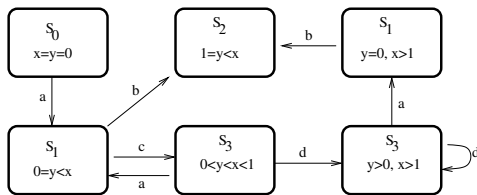
- Compute / explore a behaviour graph
- Select some test path
- Handle combinatory explosion

For that:

We Combine Symbolic Reachability with Local Exploration

Symbolic reachability

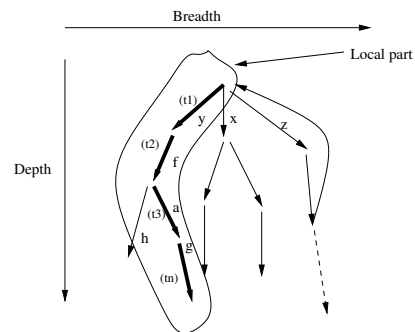
A behaviour graph with reduced size



We have saved state space ... and memory !

Local or partial exploration

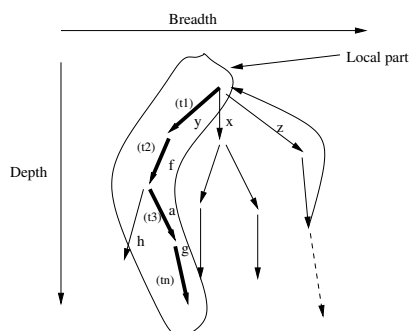
Dependability requirement



Dependability requirement: $f \rightarrow g$

Local or partial exploration

Dependability requirement



Dependability requirement: $f \rightarrow g$

Design tests from dependability requirements

Test search is guided by D_R

A Dependability Requirement (D_R) is

- Such requirement considered as **critical** in the system behaviour.
- The example below:
After input **a** is received, output **b** must be computed within **3 time units!**
- Of **high** importance in the system design and test.

Tests Design is based on

- Synchronisation of *Spec* and D_R
- Searching the local part of *Spec* that meet D_R .

Design tests from dependability requirements

Test search is guided by D_R

A *Dependability Requirement* (D_R) is

- Such requirement considered as **critical** in the system behaviour.
- The example below:
After input a is received, output b must be computed within 3 time units!
- Of **high** importance in the system design and test.

Tests Design is based on

- Synchronisation of *Spec* and D_R
- Searching the local part of *Spec* that meet D_R .



Illustration of the approach

A generic example of embedded system

- Interactions $L = \{a, b, c, d, e\}$.
 - ? a (input) System initialised in the environment
 - ? b (input) Signal detected OK from the environment
 - ! c (output) Environment is working OK
 - ! d (output) Warning signal
 - ! e (output) Emergency signal
- Clocks
 $C = \{x, y\}$
- States
 $S = \{S1, S2, S3, S4, S5, S6, S3S4, S8\}$
S1 Starting state



Illustration of the approach

The system specification model

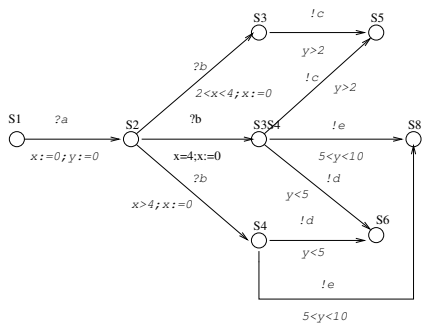


Illustration of the approach

The model of dependability requirement

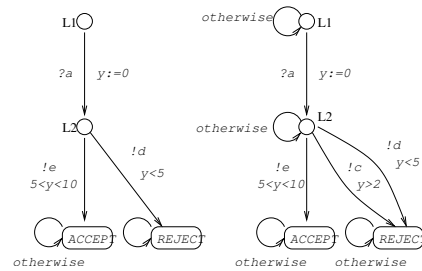


Illustration of the approach

The model of the test system

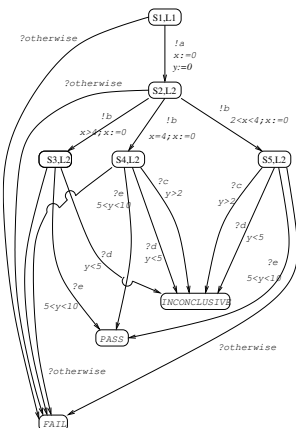
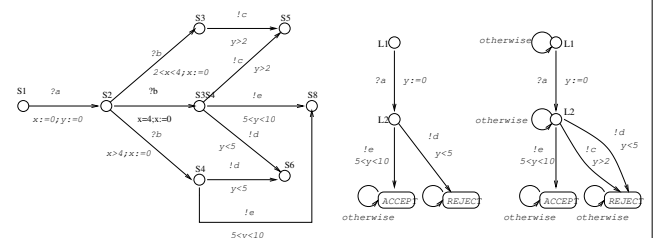


Illustration of the approach

Input and output of the test computation process



An example of **successful test** is :

(!a; x:=0; y:=0) → (!b, x=4; x:=0) → (?e, 5<y<10) → PASS.

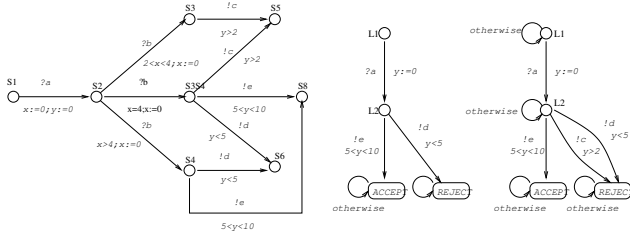
An example of **failure execution** is :

(!a; x:=0; y:=0) → (!b, x>4; x:=0) → (?e, y=10) → FAIL.



Illustration of the approach

Input and output of the test computation process



An example of **successful test** is :

$(!a; x:=0; y:=0) \rightarrow (!b, x=4; x:=0) \rightarrow (?e, 5 < y < 10)$
 \rightarrow PASS.

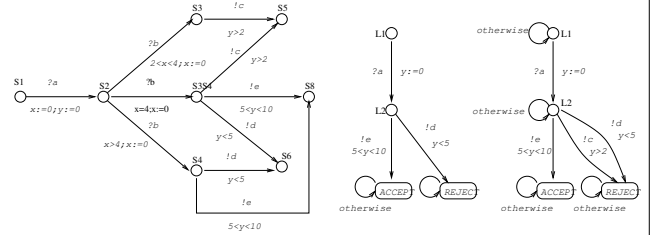
An example of **failure execution** is :

$(!a; x:=0; y:=0) \rightarrow (!b, x > 4; x:=0) \rightarrow (?e, y=10)$
 \rightarrow FAIL.



Illustration of the approach

Input and output of the test computation process



An example of **successful test** is :

$(!a; x:=0; y:=0) \rightarrow (!b, x=4; x:=0) \rightarrow (?e, 5 < y < 10)$
 \rightarrow PASS.

An example of **failure execution** is :

$(!a; x:=0; y:=0) \rightarrow (!b, x > 4; x:=0) \rightarrow (?e, y=10)$
 \rightarrow FAIL.



Summary

- Towards **full automation of tests** computation and deal with industrial embedded systems (currently concerned with aerospace industry)
- Our approach handles **the model size/complexity** by combining *symbolic* and *local* analysis.
- Outlook
 - Implement a prototype tool
 - Investigate security issues, not covered by spec. model.



From message queue to ready queue

Case study of a small, dependable synchronous blocking channels API
"Ship & forget rather than send & forget"

Øyvind Teig
Autronica Fire and Security, Trondheim
(A UTC Fire and Security company)
<http://home.no.net/oyvteig>

18 pages

From message queue to ready queue, Teig, ERCIM05, Porto

Abstract

- CSP style synchronous interprocess communication
- on top of a run-time system supporting SDL asynchronous messaging
- Unidirectional, blocking channels are supplied
- Benefits are
 - no runtime system message buffer overflow
 - "Access control" architectural design
- A pattern to avoid deadlocks is provided
- The message buffer is obsolete, and a ready-queue-only could be asked for.
- May be formally verified with the CSP process algebra.

Case study of a small, dependable synchronous blocking channels API "Ship & forget rather than send & forget".
Øyvind Teig, Autronica Fire and Security, Trondheim (A UTC Fire and Security company), <http://home.no.net/oyvteig>
Page 1

From message queue to ready queue, Teig, ERCIM05, Porto

1. Introduction
2. SDL and CSP
3. Blocking
4. Access control of other processes

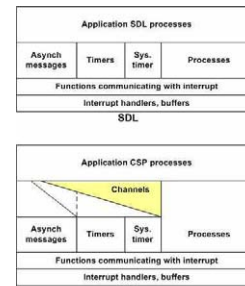
....

All these points will be implicitly covered in the next pages

Case study of a small, dependable synchronous blocking channels API "Ship & forget rather than send & forget".
Øyvind Teig, Autronica Fire and Security, Trondheim (A UTC Fire and Security company), <http://home.no.net/oyvteig>
Page 3

From message queue to ready queue, Teig, ERCIM05, Porto

5. The layered architecture



Case study of a small, dependable synchronous blocking channels API "Ship & forget rather than send & forget".
Øyvind Teig, Autronica Fire and Security, Trondheim (A UTC Fire and Security company), <http://home.no.net/oyvteig>
Page 4

From message queue to ready queue, Teig, ERCIM05, Porto

6. The channels C API abstraction

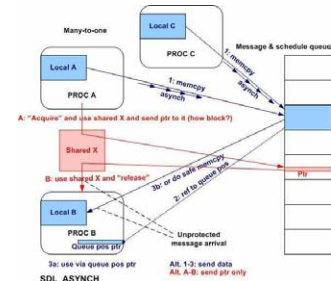
```

#define CHAN_INIT_F (CHAN_SENDER, RECEIVER, ALTTAKEN)
#define CHAN_IN_F (CHAN_DATA_EVENT)
#define CHAN_IN_VARLEN_F (CHAN_LEN_DATA_EVENT)
#define ALT_CHAN_IN_F (GUARD_CHAN_DATA_EVENT, ALTTAKEN)
#define ALT_CHAN_IN_VARLEN_F (GUARD_CHAN_LEN_DATA_EVENT, ALTTAKEN)
#define ALT_CHAN_IN_ASYNC_SIGNAL_F (GUARD_CHAN_EVENT, ALTTAKEN)
#define CHAN_OUT_F (CHAN_DATA_EVENT)
#define CHAN_OUT_VARLEN_F (CHAN_LEN_DATA_EVENT)
#define CHAN_OUT_ASYNC_SIGNAL_F (CHAN_RESCHEDULE_F)
#define CHAN_IN_ASYNC_SIGNAL_F (CHAN_EVENT)
#define ALT_TIMER_IN_F (GUARD_TTIME_INIT_EVENT, ALTTAKEN)
#define FSM_RESCHEDULE_F (EVENT)
    
```

Case study of a small, dependable synchronous blocking channels API "Ship & forget rather than send & forget".
Øyvind Teig, Autronica Fire and Security, Trondheim (A UTC Fire and Security company), <http://home.no.net/oyvteig>
Page 5

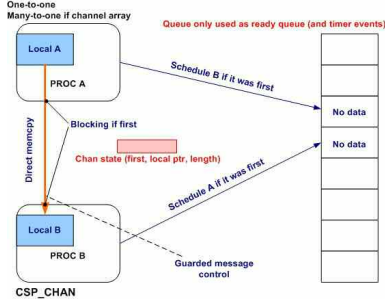
From message queue to ready queue, Teig, ERCIM05, Porto

7. Semantics of asynchronous messages



Case study of a small, dependable synchronous blocking channels API "Ship & forget rather than send & forget".
Øyvind Teig, Autronica Fire and Security, Trondheim (A UTC Fire and Security company), <http://home.no.net/oyvteig>
Page 6

8. Semantics of synchronous channels

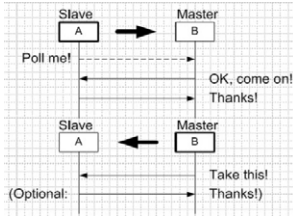


9. "From message queue to ready queue"

- Explained with figures above, i.e.:
- Message queue(s) not needed any more
- Ready/scheduling queue(s) all we need
- But we stuck to what we had!
- Of course, timer queue(s) needed

10. Deadlock avoidance

- Observe that we cannot deadlock on erroneous use of semaphores, we have none
- Only on mutual waiting for each other in a cycle
- The pattern above requires Master to treat Slave-only after "Poll me!", no other process
- (An asynch channel equals an overwritebuffer composite process with size 1 communicating over synchronous channels)



11. Coding examples (1)

```
P_BS100_Sim (void)
{
    switch (ContextPtr->State)
    {
        ... ST_INITIALIZING_A (Initialize)
        ... ST_STATE_IN_ALT_030_032_104_A
        ... -> ST_STATE_IN_ALT_030_032_104_END_A
        ... ST_STATE_OUT_031_A
        ... ST_STATE_OUT_110_A
        ... ST_STATE_OUT_END_A
        ... ST_STATE_STOPPED_A, (crash)
        ... default, (crash)
    }
    ... Common action: get more to do from workpool
}
```

- Non-preemptive "return" to scheduler gives one common stack

4. Access control of other processes (more)

- In addition to the "channel switch" mentioned we have:
- A process needs to obey the protocol semantics, it does not need to know the semantics of the other processes' internal behaviour
- Only have to look at *this* process to understand what service it offers its environment. That service is independent of its environment - its behaviour is the same anywhere
- This is WYSIWYG semantics
- Processes become dependable software components: no unpublished interactions (side-effects) between CSP parallel processes
- It also shows the compositional semantics of CSP
- Erroneous use of semaphores - not WYSIWYG!

11. Coding examples (2)

```
case ST_STATE_IN_ALT_033_093_A:
{
    bool a_alt_taken = FALSE;
    g_ALT_AL_Comp_Driver = CHAN_ALT_ENABLED_ON_A;

    ALT_CHAN_IN_F (ContextPtr->Guard_033_093, g_chan_033,
                  ContextPtr->ALCP_033i_093i,
                  S_EVENT_ALT_033_093_A, &alt_taken);
    ALT_CHAN_IN_F (ContextPtr->Guard_033_093, g_chan_093,
                  ContextPtr->ALCP_033i_093i,
                  S_EVENT_ALT_033_093_A, &alt_taken);
    ALT_TIMER_IN_F (ContextPtr->Guard_Timer_ENTX, MS_TIMEOUT_A,
                  TU_MS_A, S_EVENT_TIMEOUT_A,
                  &g_ALT_AL_Comp_Driver, &alt_taken);

    ContextPtr->State = ST_STATE_IN_ALT_033_093_120_END_A;
    break;
}
```

12. Formal basis of the architecture

- The channel layer API discussed here was modelled on macros used by the code generator of the SPoC occam-to-C compiler
- Based on occam, which is a running subset of CSP
- The CSP process algebra "discovered" by this used through occam
- However, the CSP may be used to model and verify any system
- This would be out of reach (expensive), and not very interesting for us (small system and use of known software patterns).
- However, other process algebras, like FSP, analysed with the free LTSA tool, may also be used.
- Modelling asynchronous systems (albeit with finite size buffers, which makes them synchronous when buffers are empty/full) is also possible with Promela and the free SPIN tool

13. Discussion (1)

- "CHAN_CSP" adds about 2 KB of program memory with some IN, ALT and OUT macros used.
- Execution time overhead and *memcpy*, discussion
- Processor cycles for ALT, discussion
- SDL runtime system is about 20 KB
- But even with 128 KB of code space (or, soon 256 KB – nice for an 8 bit machine) and 46(11) MHz clock, the added well-being of knowing that the system never overflows the message queue or sends unwanted messages into a processes, outweighs the overhead.

13. Discussion (2)

- This author used occam, SPoC and a C CSP library for 15 years,
- "Ship & forget" rather than "send and forget"
- Communication states need to be learned
- Complexity and engineers' preferences and background.
- If OO has had its way, the CSP (or the like) also has a way to go.
- Grasping the communicating state machines is individual.
- A channel most probably seems as belonging to OSI *network* (3) or *transport* (4) layer, and certainly not the application layer (7).
- Some programmers learn this methodology easily.
- However, when the communication infrastructure code once has been set up, it tends to stay stable and work.

13. Discussion (3)

- Size of the present "ready queue" is a matter of finding the maximum scheduling incidence volume.
- When maximum has been found, there is no room for further surprises, since the value is a function of the number of channels and processes, not the communication pattern.
- A subset of?) Ada available for microcontrollers of this type
- Java (where CSP libraries are available).
- Or hope that result of ongoing occam research will hit industry some day.
- In the meantime, we could use solutions as the one discussed here, which really is quite dependable, even if it is based on hand-written C.

14. References (1)

- The edit-by-anyone internet based *Wikipedia* dictionary has been used for some essential computer science terms.
- Wikipedia articles often point to more academic sources.
- The last reference (to www.wotug.org) has been added since it is a good starting point for both theory and practice of this field of computer science.
- The reference list (of this "industrial" paper - as opposed to "academic") should be used as hands-on *and* academic starting points, not especially for direct referencing of origins.

14. References (2)

- Added references, not in paper:
- <http://rmo.net/prelude> - Raw Metal occam - an OS for embedded applications
 - <http://www.transpreter.org> - occam on anything

An Embedded Future for Distributed System Architectures?



Trygve Lunheim, Amund Skavhaug
Department of Engineering Cybernetics
Norwegian University of Science and Technology

Workshop focus on..

Dependable Software Intensive Embedded Systems

Software Intensive implies demanding applications

- Multimedia-rich content/capabilities
- Large CPU and memory requirements

These embedded systems are becoming commonplace, such as

- Industrial applications/networks with multimedia capabilities
- Entertainment centres, multimedia hubs in homes
- DVD-recorders and -players
- Flatscreen TVs
- Gaming consoles

Embedded system capabilities

Some of these "embedded systems" are becoming more powerful than our personal computers!

Next generation PlayStation processor, Cell architecture, has

- PowerPC like core that does thread scheduling, runs OS (e.g. Linux)
- Another 8 parallel processor cores for signal processing/multimedia

And this is only the initial processor of this architecture..

While it's true that most embedded systems are designed with a minimum of computing resources in order to reduce cost, there is still a huge potential in utilizing the processing power of these new systems, if they were to become connected.

Outline

- Background
- Distributed operating systems
- Middleware
- CORBA vs GRID technologies
- Conclusion

Outline

- Background
- Distributed operating systems
- Middleware
- CORBA vs GRID technologies
- Conclusion

Background

A few years ago there was a lot of talk about how appliances would soon become connected and "intelligent". The refrigerator could keep track of when the milk went out of date, and put in an order for fresh milk and other groceries through the Internet.

It seems we're not quite there yet (at least not everywhere!), but maybe we're getting there in the future.

Our question: Why haven't we reached our goal yet?

(If letting your refrigerator handle your credit card is our goal...)

Industrial networks

In manufacturing plants, oil platforms and other industry, we have seen

- Use of new, media-rich technologies, with high bandwidth requirements
- Increased use of COTS technologies (Ethernet, TCP/IP..)
- A need for horizontal and vertical integration of systems:
Subsystems that are used within the organization often don't play well together, due to their heterogeneous nature

Although the applications for industrial systems and consumer products are very different, many of the requirements will often be similar. I.e. security, reliability, real-time requirements..

Predicting the future

Many different standards and architectures exist, for:

- Parallel computing
- Distributed objects, software components
- Fault tolerance, load sharing

Problem:

no ubiquitous standard for distributed real-time and embedded systems

Purpose of the paper:

to review some trends in distributed and embedded computing, and make some predictions about the future of distributed architectures in embedded systems

Outline

- Background
- Distributed operating systems
- Middleware
- CORBA vs GRID technologies
- Conclusion

Distributed operating systems

Traditionally, research on distributed support for general operating systems has been concerned with achieving increased performance.

The support for distributed applications has evolved through stages:

- Support for transactions
- Message passing (MPI)
- Shared memory (uniform / non-uniform memory access)

However: General operating systems are not designed for the real-time or embedded systems. Although they may well be used for some such applications, it is often not possible or even desirable to do so, due to pricing, memory requirements etc..

Real-time operating systems

Real-time requirements => low overhead on communication and CPU

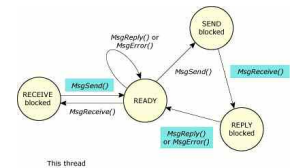
For example: VxWorks, LynxOS, QNX, RT-Linux...

Often lacking in support for distributed applications directly in the OS
But support for communications, e.g. TCP/IP, Ethernet.

Distributed applications are often built using TCP/IP socket communication, even if this is at a lower abstraction layer than desirable.

QNX Neutrino

Real-time microkernel OS,
IPC based on message-passing
MsgSend()/MsgReceive()/MsgReply()



Offers efficient message passing in a distributed system transparently:
QNet protocol works between tightly coupled machines (trusted)
Sharing of namespace allows for sharing of resources and MP:

Supports QoS over multiple networks, load balancing over links

QNX MP example, local

Server process:

```
attach = name_attach(NULL, "mydev", 0);
while (1) {
    rcvid = MsgReceive(attach->chid, &msg, sizeof(msg), NULL);
    if(rcvid == our_msg) { printf("Server received %d\n", msg.data);
        MsgReply(rcvid, EOK, 0, 0);
    }
}
```

Client process:

```
fd = name_open("mydev", 0);
for(i=0; i<5; i++) {
    MsgSend(fd, &msg, sizeof(msg), NULL, 0);
}
```

13

ERCIM Workshop on Dependable Software Intensive
Embedded systems

11.12.2005

QNX MP example, global namespace

Server process:

```
attach = name_attach(NULL, "mydev", NAME_FLAG_ATTACH_GLOBAL);
while (1) {
    rcvid = MsgReceive(attach->chid, &msg, sizeof(msg), NULL);
    if(rcvid == our_msg) { printf("Server received %d\n", msg.data);
        MsgReply(rcvid, EOK, 0, 0);
    }
}
```

Client process:

```
fd = name_open("/dev/name/global/mydev", 0);
for(i=0; i<5; i++) {
    MsgSend(fd, &msg, sizeof(msg), NULL, 0);
}
```

14

ERCIM Workshop on Dependable Software Intensive
Embedded systems

11.12.2005

Native MP vs Open standards

QNX native message passing works well, when you are developing a distributed application and you want to run it on QNX nodes.

This is often not the case.

It's proprietary, so you need to use a particular product (QNX).

You may want to use other operating systems, or retrofit an older application, not using QNX MP.

Using a particular OS mechanism or even specialized hardware to implement distributed applications implies homogeneity, while a large number of systems are inherently heterogeneous.

Tight coupling of processes, but what if we want/need loose coupling?

15

ERCIM Workshop on Dependable Software Intensive
Embedded systems

11.12.2005

Outline

- Background
- Distributed operating systems
- Middleware
- CORBA vs GRID technologies
- Conclusion

16

ERCIM Workshop on Dependable Software Intensive
Embedded systems

11.12.2005

Middleware

Middleware enables distributed processing across heterogeneous architectures and/or networks

Examples of middleware platforms include

- D/COM, OPC
- J2EE
- .NET
- CORBA
- GRID/Web Services

17

ERCIM Workshop on Dependable Software Intensive
Embedded systems

11.12.2005

Middleware cont'd

There are many types of middleware, with different application areas, qualities and shortcomings, such as

- Transactional middleware
- Message-oriented middleware
- Component-based middleware
- Model-driven middleware
- Adaptive/reactive middleware

Component middleware enables reusable services to be composed, configured and installed, to create applications rapidly and robustly. Interfaces are defined between components, and infrastructure services are built into the middleware. Examples include COM, J2EE and CORBA Component Model (CCM).

18

ERCIM Workshop on Dependable Software Intensive
Embedded systems

11.12.2005

Middleware properties

We'll consider some important properties of middleware architectures, with respect to embedded systems

- Meta-information handling
- Support for Real-time services
- Security support

19

ERCIM Workshop on Dependable Software Intensive Embedded systems

11.12.2005

Outline

- Background
- Distributed operating systems
- Middleware
- CORBA vs GRID technologies
- Conclusion

20

ERCIM Workshop on Dependable Software Intensive Embedded systems

11.12.2005

Common Object Request Broker Architecture

CORBA Component Model introduces components, with interfaces (facets), hooks for dependencies and attributes. Interfaces are defined using CORBA's Interface Description Language (IDL).

The component model facilitates software development through reuse of software, and fault tolerance through replication of components (FT CORBA).

We have not seen widespread use of CORBA in embedded systems yet..

- Has gone through several revisions
- Implementations are still considered large/unwieldy
- Difficult to set up and use?

21

ERCIM Workshop on Dependable Software Intensive Embedded systems

11.12.2005

GRID, a service-oriented architecture

Built on Web Services, for supporting collaboration between research institutions, and sharing of computing resources. Typically used to solve large computational tasks, using a dynamic collection of clusters that are distributed. Applications can be built using the Globus Toolkit

Service-oriented architecture:

"standard interfaces and protocols that allow developers to encapsulate information tools as services that clients can access without knowledge of, or control over, their inner workings"

GRID technologies enable a separation of concerns between discipline-specific content and domain-independent software and hardware infrastructure. The general nature of these services are of interest to us.

22

ERCIM Workshop on Dependable Software Intensive Embedded systems

11.12.2005

Meta information in CORBA

- Part of OMG Model Driven Architecture
- Meta-Object Facility (MOF) provides a framework for managing any type of metadata in CORBA
- Layered architecture with a meta-modeling layer and an object modeling layer to tie together metamodels and models
- OMG defined XML Metadata Interchange (XMI) for representing and exchanging metamodels

23

ERCIM Workshop on Dependable Software Intensive Embedded systems

11.12.2005

Meta information in GRID

Monitoring and Discovery System (MDS)

Useful information

- Characteristics of computing devices
- Characteristics of infrastructure
- Policy

Grid Laboratory Uniform Element (GLUE)

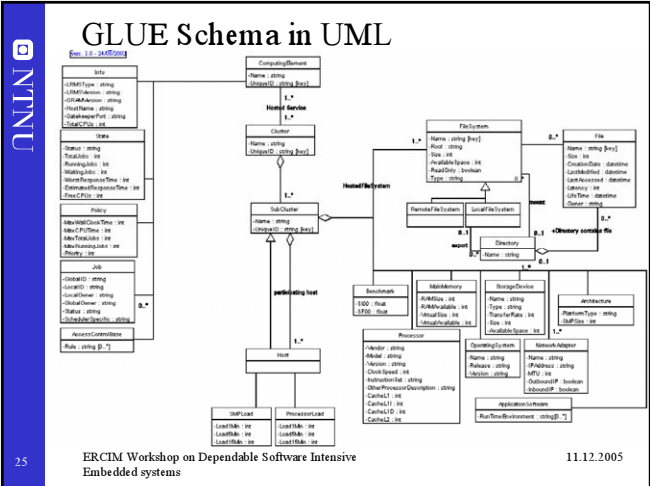
Implemented in XML for MDS

LDAP and SQL versions also, used in older versions of MDS

24

ERCIM Workshop on Dependable Software Intensive Embedded systems

11.12.2005



Real-time capabilities in middleware

CORBA Real-Time implementations exist, e.g. TAO, CIAO. These include support for timers and real-time scheduling, as well as support for Quality of Service in networks.

In the Globus Toolkit there is extended support for QoS at the application level, using the GARA architecture. So it is possible to do reservation for high priority applications. Better support for real-time applications should be worked on in the future.

However: The underlying support for guaranteed end-to-end QoS in IP networks is still missing, i.e. there is no standard that is generally agreed upon.

ERCIM Workshop on Dependable Software Intensive Embedded systems 11.12.2005

Security

Security has often been overlooked in embedded and real-time systems, and many times treated as an afterthought.

Fortunately it has become more common to consider security in all levels of the system, and from the earliest stages of development. Threats and means to security must be defined, and the correct methods to reach an assurance level must be found.

This may be a long and cumbersome process, leading to increased cost.

However:
The consequences of not doing things right when it comes to security may be worse.

ERCIM Workshop on Dependable Software Intensive Embedded systems 11.12.2005

Security in CORBA and GRID

- Security was not originally a part of the CORBA specification, but added afterwards
- SecSIG, a Special Interest Group within OMG defines security in CORBA, as a core service
- For example, the TAO implementation of CORBA implements some of the security functionality in the CORBA specification, providing secure transport where needed, using Secure Socket Layer (SSL)

The Grid Security Infrastructure supports security at message and transport layers, using Transport Level Security (TLS).

In the future, Role Based Access Control (RBAC) should play an important role in distributed applications.

ERCIM Workshop on Dependable Software Intensive Embedded systems 11.12.2005

CORBA vs GRID middleware

Some critics have claimed that Web Services/SOAP brings nothing new, when it comes to middleware. CORBA, which has been used for years, is more efficient and better defined.

Both will probably play a role in the future. CORBA has been around for longer, is more well-defined/standardized, and is probably more efficient, both in terms of processing/memory as well as communication.

However, GRID technology has a lot of momentum and support from big players in the industry. It also has more potential to become ubiquitous.

For some applications it is probably better to use something like CORBA, but for providing a general interface to the outside world it makes sense to use something like SOAP or WSDL. This also holds for embedded systems.

ERCIM Workshop on Dependable Software Intensive Embedded systems 11.12.2005

Outline

- Background
- Distributed operating systems
- Middleware
- CORBA vs GRID technologies
- Conclusion

ERCIM Workshop on Dependable Software Intensive Embedded systems 11.12.2005

The future?



Looking into the crystal ball, there are some strong trends towards:

- Increased integration of systems and services
- More computing power, higher bandwidth etc.
- Increased use of COTS technologies

Future work

- Finding exact requirements of distributed architectures for embedded systems
- Specifying the minimum set of functionality in order to implement these on embedded systems, in terms of size and processing requirements

31

Conclusion

- Embedded systems are soon becoming powerful enough to perform in complex distributed environments
- There is a need for standards to achieve proper discovery, configuration, monitoring and scheduling within such an environment
- Advanced distributed architectures and middleware are large and require resources and effort. Probably because *distributed computing is complex*

32

Questions?

33

References

CORBA <http://www.omg.org/>
Globus Toolkit <http://www.globus.org/toolkit/>

34



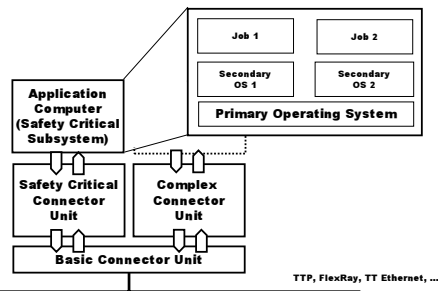
Allocation of Dependable Software Modules under Consideration of Replicas

ERCIM Workshop on Dependable Software
Intensive Embedded Systems

Motivation

- Federated Systems increase cost and weight
- Therefore, there is a tendency towards Integrated Systems
 - ◆ Dependable Embedded Components & Systems (DECOS)
 - ◆ Integrated Modular Avionics (IMA) in Airbus A380
- Advantage: Sharing of Resources
 - ◆ Communication (Shared bandwidth, less wiring)
 - ◆ Computation (Several subsystems on one node)
 - ◆ I/O Resources (e.g., sensors - better redundancy)
- New Challenge: Allocation of distributed subsystems

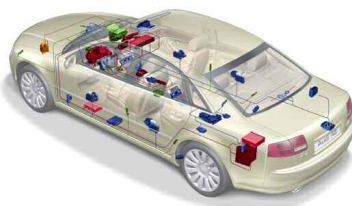
Example: Node of DECOS Architecture



Role of Allocation within DECOS project

- Allocation/Scheduling is part of development methodology
 - ◆ MDA: Platform Independent Model vs. Platform Specific Model
- Tasks can be allocated to any node as long as **constraints** are fulfilled:
 - ◆ Resource Constraints
 - ◆ Dependability constraints
 - ◆ ...
- Optimization function is used to find optimal result

Allocation of Distributed Subsystems (1)



Allocation of Distributed Subsystems (2)

- Currently, task seems trivial:
 - ◆ Software is developed for manufacturer-specific components
 - ◆ No choice to locate software "elsewhere"
- Integrated Systems increase flexibility:
 - ◆ Uniform hardware nodes (often from consumer domain)
 - ◆ Software may run on any chip (provided that **constraints** are fulfilled)

ARC Seibersdorf research GmbH INFORMATION TECHNOLOGIES

Allocation is NP complete

1. 2. 3. 4. 5. 6. 7. ...

2^3 possibilities

Porto, September 1, 2005
Georg Weissenbacher

seibersdorf research
An enterprise of the Austrian Research Centers

ARC Seibersdorf research GmbH INFORMATION TECHNOLOGIES

Symmetric Solutions (1)

1. 2. 3. 4. 5. 6. 7. ...

2 replicated (equivalent) nodes

3 & 4 symmetric because nodes are equivalent

2 & 6 symmetric because tasks are equivalent

Porto, September 1, 2005
Georg Weissenbacher

seibersdorf research
An enterprise of the Austrian Research Centers

ARC Seibersdorf research GmbH INFORMATION TECHNOLOGIES

Symmetric Solutions (2)

- Significant reduction of number of considered solutions!

- How can we implement this efficiently?

Porto, September 1, 2005
Georg Weissenbacher

seibersdorf research
An enterprise of the Austrian Research Centers

ARC Seibersdorf research GmbH INFORMATION TECHNOLOGIES

Outline of Algorithm with Equivalence Sets (1)

- Find equivalence sets for Tasks and Nodes

- Process one Task Equivalence Set (TES) after another

Porto, September 1, 2005
Georg Weissenbacher

seibersdorf research
An enterprise of the Austrian Research Centers

ARC Seibersdorf research GmbH INFORMATION TECHNOLOGIES

Outline of Algorithm with Equivalence Sets (2)

- Allocate the first element Task 1 on Node 1

- Split Node Equivalence Set

Porto, September 1, 2005
Georg Weissenbacher

seibersdorf research
An enterprise of the Austrian Research Centers

ARC Seibersdorf research GmbH INFORMATION TECHNOLOGIES

Outline of Algorithm with Equivalence Sets (3)

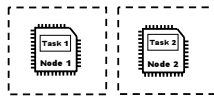
- If assignment fails (e.g., node has not enough resources to host this task), *backtrack*.
- Backtracking means:
 - consider a different NES
 - If no NESs are left to consider, reconsider the TES that was processed before the current TES
 - If this fails, too, there is no valid assignment
- But: Elements of an Equivalence Set are never considered separately, therefore not all allocations are considered

Porto, September 1, 2005
Georg Weissenbacher

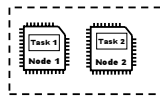
seibersdorf research
An enterprise of the Austrian Research Centers

Outline of Algorithm with Equivalence Sets (4)

- After each assignment, try to **recombine** NESs:



If Task 1 and Task 2 are equivalent and Node 1 and Node 2 are equivalent



Complexity of the Allocation algorithm (1)

- Processing of each single TES is basically a combination (order of tasks is ignored) with repetition (because nodes are "put back" into the pool).
- Choose k elements from a set of n :

$$\binom{n+k-1}{k} = \frac{(n+k-1)!}{k!((n-1+k)-k)!}$$

Complexity of the Allocation algorithm (2)

- We can never be sure if **recombination** is successful. Therefore, consider worst case that there are no node equivalence sets
- In step i , s_i denotes the size of the current TES, and r_i denotes the number of NESs:

$$O(W(i, r_i, s_i)) = O\left(\frac{(s_i + r_i - 1)!}{r_i!(s_i - 1)!}\right)$$

Complexity of the Allocation algorithm (3)

- Overall complexity (k TESs have to be considered):

$$O\left(\prod_{i=1}^k W(i, r_i, s_i)\right) \text{ with } r_i = |J_i|$$

- Example:
 - Steer-by-wire system

Example: Steer-by-Wire (1)

- Driver_Assistant: Receives input from sensors and adjusts it
- Steering_Algorithm: Determines analogue value used to control the mechanical steering system
- Steering_Rack_Sensor: Provides information about forces
- Steering_Rack_Control: Controls turning angle of wheels using the values provided by Driver_Assistent
- Force_Feedback: Provides haptic feed-back to driver
- Turning_Angle_Sensor: Measures and provides current angle of steering wheel
- Speed_Sensor: Measures and provide current speed of vehicle

Example: Steer-by-Wire (2)

- 7 tasks, 3 of them replicated: Overall number of 10 tasks
- Allocate to 4 processors (2 equivalence groups)

Task	Time Budget	Number
Turning_Angle_Sensor	270	2
Speed_Sensor	270	2
Steering_Rack_Control	130	2
Steering_Rack_Sensor	110	1
Steering_Algorithm	110	1
Driver_Assistant	270	1
Force_Feedback	110	1

- Worst case of unoptimized algorithm: 4^{10} (=1048576)

Example: Steer-by-Wire (3)

- Optimized: 7 Task Equivalence Set
- Step 1: 2 tasks in equivalence set, assume we have 3 nodes:

$$W(1, r_1, s_1) = \frac{(3+2-1)!}{2(3-1)!} = 6$$

- Step 2: 2 tasks in equivalence set, by now 4 different nodes

$$W(2, r_2, s_2) = \frac{(4+2-1)!}{2(4-1)!} = 10$$

Example: Steer-by-Wire (4)

- Step 3: Similar to step 2, therefore $W(3, r_3, s_3) = 10$
- Remaining 4 steps: No equivalence classes, therefore worst case is equivalent to DFS (4^4)
- Overall worst case: $6 \cdot 10^2 \cdot 4^4 = 153600$

Example: Experimental Evaluation

- For given example:
 - DFS: 15516 partial assignments
 - Optimized: 797 partial assignments

n	m	k	l	DFS	EDFS
10	4	7	2	15516	797
10	10	5	5	14308 ⁵⁰	77911
10	10	10	10	22374 ²	22374
8	4	4	2	768	33

Conclusion

- Consideration of equivalent elements reduces search space
- Can be combined with other optimizations (e.g., variable ordering, heuristics)
- But: No magic: Worst case is still exponential

Assessing Reliability of Real-Time Distributed Systems

by Åsmund Tjora and Amund Skavhaug

Department of engineering cybernetics
Norwegian University of Technology and Science

1

Introduction

- Many real-time applications have reliability requirements in addition to timing requirements.
- A fault tolerance mechanism must be chosen so that deadlines are met even when a fault occurs
- Many real-time systems uses fault tolerance mechanisms based on parallel structures, that is, tasks are run on several instances of an object. Even if one of the instances should fail, the others will still be able to complete the task before the deadline.
- In addition, comparison between the results can be used as an extra fault detection mechanism
- However, mechanisms based on parallel structures can be resource consuming.

2

Introduction

- Fault tolerance mechanisms based on serial structures are less resource consuming than those based on parallel structures
- When using these mechanisms, if a fault occurs, the fault is corrected and the task is rerun. This will, of course, take extra time
- Because of this, it is possible that the extra time used to tolerate a fault will cause a deadline miss.
- For these systems, it is important to analyze the timing behavior. This analysis can be done in different ways.

- Two methods for this analysis are shown in this presentation:
- A mathematical model
- Simulation

3

Introduction

Analytical model

- Precise results
- Once derived, they can often be used on similar systems with little modifications
- Can be quite complicated
- Not always easy to understand or use

Simulation

- Not precise, but often good enough
- Usually easy to create and understand
- Rare events does not always appear in the results

4

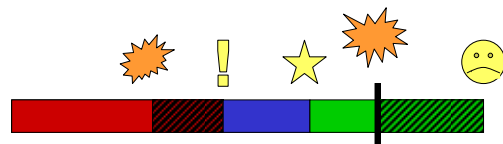
Model of a fault-tolerant system

- Client-Server system
- The server is replicated
- One server replica is running, the others are passive
- If the active replica fails, the state of one of the passive replicas are brought up to date. This is now the new active replica
- The task that was currently running is restarted on the new active replica.

5

Model of a fault tolerant system, timing

- Normal operation
- Fault occurs
- Fault is detected
- Correction
- Normal operation
- Deadline?



6

Analytical Model

- Uses moment generating functions for the distributions
- Method used to derive expressions is similar to the one used to derive the expression for the busy period in a queuing system
- The moment generating functions used in the expression are:
 - $M(s)$ The fault free run-time distribution
 - $I(s)$ The fault detection time distribution
 - $C(s)$ The fault correction time distribution
- Faults are modeled as if generated by a poisson process with an intensity of λ
- The expression derived is

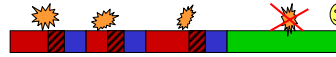
$$G(s) = M(\lambda + s) + \left(\sum_{k=1}^{\infty} \left(\frac{\lambda}{\lambda + s} \right)^k I(ks) \left(\sum_{n=0}^{k-1} (-1)^n \binom{k-1}{n} (M((i+1)(\lambda + s)) - M((i+2)(\lambda + s))) C((i+1)(\lambda + s)) \right) \right)$$

Analytical model

- There is a possibility that there will be an infinite number of faults while running the same task:



- We can, however, get a good approximation by assuming there will be a maximum of N faults:



- Or that the task will fail if there are more than N faults:



Analytical model

- For the first approximation, the mgf will be

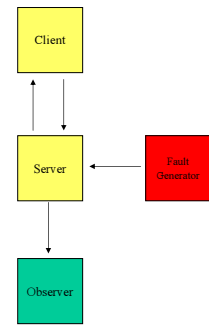
$$G(s) = M(\lambda + s) + \left(\sum_{k=1}^{\infty} \left(\frac{\lambda}{\lambda + s} \right)^k I(ks) \left(\sum_{n=0}^{k-1} (-1)^n \binom{k-1}{n} (M((i+1)(\lambda + s)) - M((i+2)(\lambda + s))) C((i+1)(\lambda + s)) \right) \right) + \left(\frac{\lambda}{\lambda + s} \right)^N I(Ns) \left(\sum_{n=0}^{N-1} (-1)^n \binom{N-1}{n} (M((i+1)(\lambda + s)) - M((i+2)(\lambda + s))) C((i+1)(\lambda + s)) \right)$$

- For the second approximation, the mgf will be

$$G(s) = M(\lambda + s) + \left(\sum_{k=1}^{\infty} \left(\frac{\lambda}{\lambda + s} \right)^k I(ks) \left(\sum_{n=0}^{k-1} (-1)^n \binom{k-1}{n} (M((i+1)(\lambda + s)) - M((i+2)(\lambda + s))) C((i+1)(\lambda + s)) \right) \right)$$

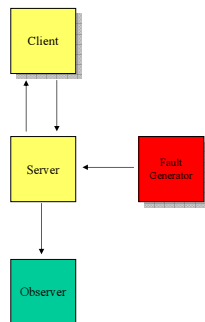
Simulator, structure

- Uses a simple client-server model as a base.
- The server is made so it simulates the fault tolerant mechanism described earlier
- A fault generator is generating faults at random intervals
- The run times and other useful data is sent to an observer, which present the data in a way that is useful for later analysis



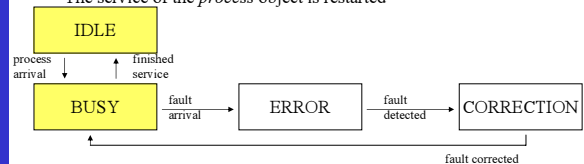
Simulator, Client and Fault Generator

- In this system, both the **Client** and the **Fault Generator** are pure generators.
- The **Client** generates *process* objects that are sent to the **Server** at regular intervals
- The **Fault Generator** is a generator of fault objects. The intervals between faults are drawn randomly from an exponential distribution.



Simulator, Server

- When the **Server** receives a *process* object from the **Client**, a service time is drawn from the service time distribution. During normal operation, the *process* object finishes after this time
- When a *fault* object arrives, normal operation is stopped. The **Server** enters the ERROR state, and a fault detection time is drawn
- After the fault detection time, the Server enters the CORRECTION state, and a fault correction time is drawn
- When the fault is corrected, the **Server** returns to normal operation. The service of the *process* object is restarted



Simulator

- Simulator is implemented in C++, using the ADEVS discrete event system simulator framework
- Matlab is used for data analysis

13

Example

- Simple system parameters
- Fault free run-time: Triangular distribution

$$m(t) = \begin{cases} 0 & , 0 \leq t < 6 \\ \frac{t-6}{2} & , 6 \leq t < 8 \\ \frac{10-t}{2} & , 8 \leq t < 10 \\ 0 & , t \geq 10 \end{cases} \quad M(s) = \frac{e^{-6s} - 2e^{-8s} + e^{-10s}}{4s^2}$$

- Fault detection time: Uniform distribution

$$f(t) = \begin{cases} 0 & , 0 \leq t < 1 \\ \frac{1}{2} & , 1 \leq t \leq 3 \\ 0 & , t > 3 \end{cases} \quad F(s) = \frac{e^{-s} - e^{-3s}}{2s}$$

- Fault correction time: Uniform distribution

$$c(t) = \begin{cases} \frac{1}{2} & , 0 \leq t \leq 5 \\ 0 & , t > 5 \end{cases} \quad C(s) = \frac{1 - e^{-5s}}{5s}$$

14

Example

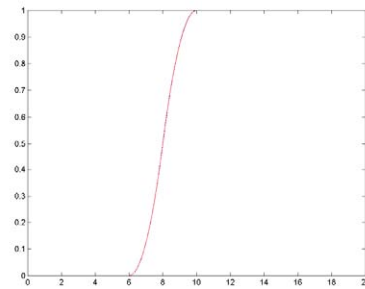
- Mean time between faults: 10000
- Maximum number of faults that can occur before a task fails: 2
- The mgf for the run-time distribution:

$$G(s) = M(\lambda + s) + \left(\frac{\lambda}{\lambda + s}\right) (F(s)M(\lambda + s) - M(2\lambda + 2s))C(\lambda + s) + \left(\frac{\lambda}{\lambda + s}\right)^2 I(2s) (M(\lambda + s) - M(2\lambda + 2s))C(\lambda + s) - (M(2\lambda + 2s) - M(3\lambda + 3s))C(2\lambda + 2s)$$

- Number of tasks run by simulator: 100000

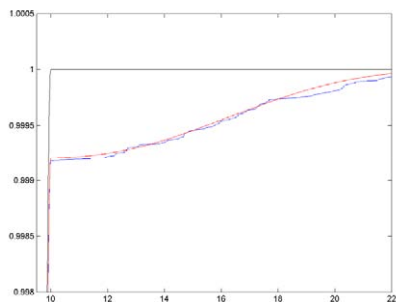
15

Results



16

Results



17

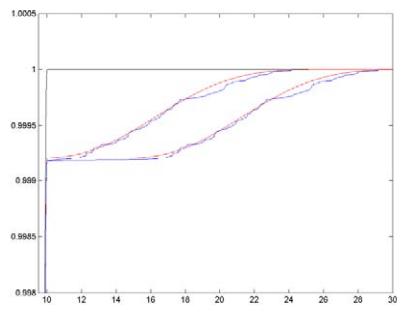
Changing the correction time

- A delay of 5 is added to the time used to correct a fault
- The new pdf and mgf:

$$c(t) = \begin{cases} 0 & , 0 \leq t < 5 \\ \frac{1}{5} & , 5 \leq t \leq 10 \\ 0 & , t > 10 \end{cases} \quad C(s) = \frac{e^{-5s} - e^{-10s}}{5s}$$

18

Changing the correction time



Discussions

Very simple fault mechanism is assumed:

- Independent faults
- All faults can be detected and corrected with the fault tolerance mechanism

Other fault classes and behaviors can be modeled

- For most fault behaviors, expanding the simulator is not very complicated.
- The difficulty of expanding the analytical model varies depending on the modeled behavior

Assessment of Safety Critical Systems with COTS and software of uncertain pedigree (SOUP)

Torbjørn Skramstad
 Norwegian University of Science and Technology (NTNU)
 Det Norske Veritas Research (DNVR)

Det Norske Veritas (DNV)

- ◆ “To safeguard life, property and the environment”
- ◆ Foundation established 1864. Self owned.



- ◆ Certification
- ◆ Classification
- ◆ Consultancy
- ◆ Technical services

NTNU facts

FACTS

- ◆ 7 faculties and 53 departments
- ◆ NTNU Library
- ◆ Museum of Natural Science and Archaeology
- ◆ 40 000 applicants each year, 5 500 of these admitted
- ◆ 20 000 registered students
- ◆ 2 500 degrees awarded each year
- ◆ 200 PhD degrees awarded each year
- ◆ 3 500 staff (faculty: approx. : 900)
- ◆ Budget NOK 2.8 billion (EUR 340 million)



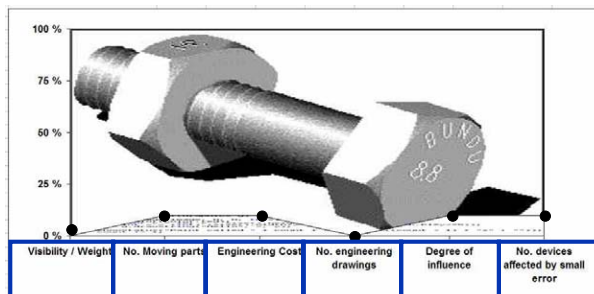
Background

- ◆ Software-based systems replace older technologies also in safety and mission-critical applications
 - Aircraft engine control and steering
 - Railway signals and train control
 - Medical devices
 - Steering and piloting of automobiles
 - Control systems on ships
- ◆ Earlier: Manual backup could take over.
- ◆ Increased requirements to speed and volume erodes this fall-back capability
- ◆ Control systems in ships are becoming more and more complex

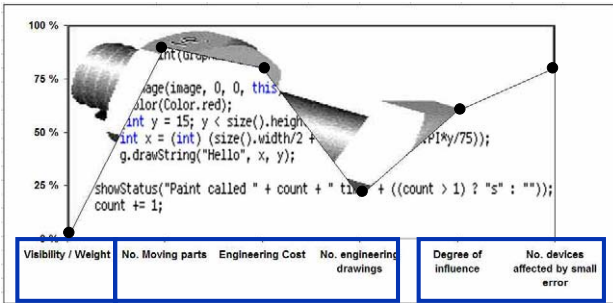
ICT is essential in modern maritime systems



Onboard ICT – some time ago



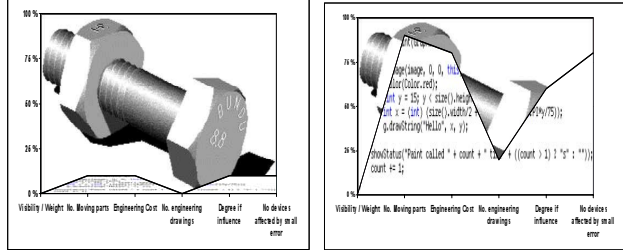
Onboard ICT – these days



ERCIM WS on Dependable Software Intensive Embedded Systems Porto 01.09.2005

Slide 7

Change



- Software cannot be avoided, even during manual operation
- The cost, complexity, and reach of software has escalated widely, and will continue to do so.
- Has industry refocused according to changed risk picture?

ERCIM WS on Dependable Software Intensive Embedded Systems Porto 01.09.2005

Slide 8

Examples of Incidents & losses

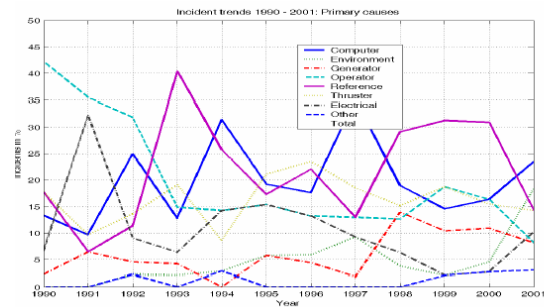


ERCIM WS on Dependable Software Intensive Embedded Systems Porto 01.09.2005

Slide 9

Trends for the primary causes of incidents in the 12 year period 1990-2001

- The numbers are in percentage out of total reported incidents (**Loss of position 1 & 2, and Lost time incidents**) the respective years.



ERCIM WS on Dependable Software Intensive Embedded Systems Porto 01.09.2005

Slide 10

Ship accidents

- Several recent ship accidents, catastrophes and near-accidents are most probably due to software failures or failures in interfaces between software and hardware
 - Tricolor collision in the British channel – economical loss > 1 bill. USD
 - Color line passenger ferry got on ground in Oslo fjord – ship had to be repaired (loss of income over several months)
 - Several DP (Dynamic positioning) vessels in North Sea loss of position (near accidents)

ERCIM WS on Dependable Software Intensive Embedded Systems Porto 01.09.2005

Slide 11

Challenges

- Economic reasons and pressure from the market to reduce development times lead to a steadily increasing use of COTS or older proprietary software of uncertain pedigree (SOUP)
- Control and safety systems onboard ships have become more dependent of computers and software
- Industry request approval of essential applications based on non-safety COTS software and hardware
- PCs with standard software (e.g. Microsoft software) is currently used in several of critical ship applications such as steering and propulsion control
- DNV as a classification society must have a clear position regarding how COTS should be used and assessed on ships

ERCIM WS on Dependable Software Intensive Embedded Systems Porto 01.09.2005

Slide 12

What is COTS?

- “Commercially available application sold by vendors through public catalogue listings. COTS is not intended to be customized or enhanced” (*DO-178B*)
 - “Software, the pedigree of which is unknown or uncertain, which could in any way affect the correct operation of a safety-related system” (*HSE / UK*)
- Typical examples of COTS:
 - Operating systems
 - Database systems
 - SCADA (*supervisory control and data acquisition*)
 - Network and communication software
 - A hardware component (e.g. a PC) can also be a COTS

Requirements to safety-related systems

International safety standards

- IEC 6108
- DO-178B
- EN50126-29
- UK Min. of Defence 00-55, 00-56
- etc.
- Safety Integrity Level (SIL 1-4)
- Requirements to design and development for each SIL level
- Independent assessment

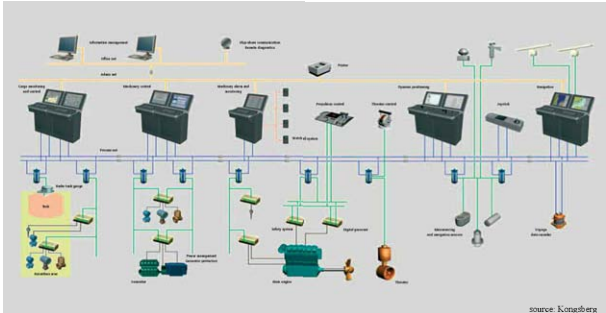
DNV classification rules:

- Essential functions
- Important function
- No specific requirements to software
- Requires that an independent system can take over in case of failure in an essential system (e.g. manual operation)
- No single failure in a control system should lead to loss of an essential function and an accident
- Requirements to functional testing of all functions

Comparison of safety standards

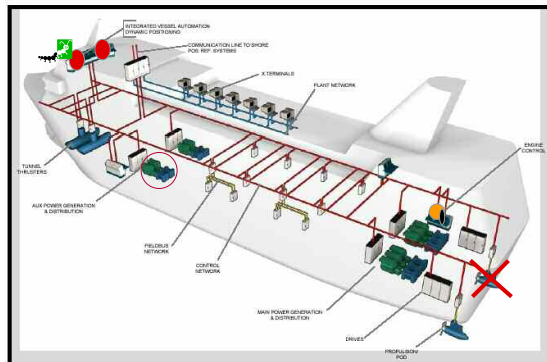
	Def Amd 505	MIL-STD-883C	SU-225/STANAG 4004 & 4052	UK DEF Stan 00-56	UK DEF Stan 00-55 & 00-54	ARP 4754 & 4751	ETC/DO-178B	IEC 6108
Level of Protection and Evidence	Requirements to hold from guidance or record file	Requirements only, but these are usually open to interpretation	Requirements to hold from document 4452 separate requirements and guidance	Separate requirements and guidance documents	Separate requirements and guidance documents	Guidance only	Guidance only	Requirements in parts 1-4, guidance in small-foot print and parts 5-7
Tasking and Confirmation	No tasking	Tasking to application	Tasking to application	Tasking to application	No tasking	Tasking by selecting guidance	Tasking by selecting guidance	No tasking; derived access methods recommended
Approval Responsibilities	Analysis, Evidence, Safety Management Group	Tasking for Analysis, System Safety Working Group	IECIC (comprehensive Control Board, 4452 Task for Systems Safety Working Group)	Independent Safety Analysis, Project Safety Committee	As for DEF Stan 00-55, plus V&V Team	Certification Authority	Certification Authority	Functional safety assessment
Deliverables	Safety Case, safety review reports, evidence and audit reports	Progress reports	4454 technical tasks only	Safety Case, audit reports	00-55 Software quality development and V&V plans 00-54 design plan	Certification Plan, Certification Summary	Software Accomplishment Summary, quality and verification plans	Plans for safety assessment and validation
Safety Planning and Control	audits and evaluations	audits	4454 as management plans, peer reviews and non-repeatable	Audits, DRACAS	As for 00-50	Certification Plan	Lifecycle phase transition criteria	-
Prevent Lifecycle	No lifecycle assumed	No lifecycle assumed	4454 one lifecycle that includes evidence 4452 on lifecycle assumed	One lifecycle, includes verification	-	-	-	No development lifecycle assumed; derived safety lifecycle
Prod Development Process	Insulation, maintenance commissioning	Failure analysis	4454 maintenance 4452 modification	Modification	Maintenance	Insulation, maintenance modification	Modification	Insulation, commissioning maintenance modification
Development Constraints	Design methods, programming languages	Design for maintain programming languages	4454 very detailed requirements for architecture and coding	design methods, coding tools	ISO 9001, ISO 9000-3, insurance, tools, coding	None	None	Detailed design methods and architecture
Hazard Analysis	-	Integration, human factors, health	4454 no hazard analysis 4452 Integration, change	Change, operations & support, health	No hazard analysis	Integration	No hazard analysis	Continuous, to identify spurious hazards
Risk and Impact Assessment	Component SIL derived from accident scenario & external probability, use of fault tolerant design	Risk from derived from hazard severity & probability, level of fault tolerant design	4452: based on Software Control Catalogues, but not used independently	Target SIL or probability derived from accident scenario and protective measures	SILs derived from DEF Stan 00-50	Radar Assessment Level & failure probability derived from failure severity and fault tolerant design	Software Level depends on failure severity and fault tolerant design	Target SIL and probability derived from accident scenario and protective measures
Design Assurance	Require depends on SIL, use of formal methods	None specific requirements	None specific requirements	Require depends on SIL, use of formal methods	Require depends on SIL, use of formal methods	Equivalent to design assurance from Assurance Level	General requirements, depend on Software Level	Require depends on SIL, use of formal methods
Software Assurance	Require depends on SIL, use of formal methods	None	None & dynamic analysis	Require depends on SIL, use of formal methods	Require depends on SIL, use of formal methods	None	None	Require depends on SIL, use of formal methods
Hardware Assurance	Testing, Use of formal methods depend on SIL	Testing, commercial	4454 none 4452 required	Use of static & dynamic analysis depends on SIL	Static & dynamic analysis	None	None	Use of static & dynamic analysis depends on SIL
Human Factors	SILs achieved by operator training procedures	Training, procedures, operator hazard analysis	4454 detailed design requirements 4452 procedures, hazard analysis	Estimation of operator failure rates, training, procedures	00-55 procedures	None	None	Procedures, training
Non Development Items	Transfer assurance or build safety case	Depend on case, training	4452 analysis & testing	Safety case	Verification, validation, use case service history	New safety assessment, use case service history	New safety assessment, use case service history	Service history or verification & validation

Control system onboard ship



- Dynamic Positioning
- Navigation
- Power management
- Trustee and Propulsion control
- Alarm Management
- Cargo & Ballast control
- Stabilizers
- Deck machinery
- Future?

Example of passenger ship subsystems



Problems with COTS

- Important quality factors such as reliability and robustness is unknown
- Configuration status is unknown
- May fail in many different ways
 - Deliver wrong output
 - Function not carried out fast enough
 - Does not deliver output (COTS “hanging”)
 - COTS “writes” into some other application’s data or code area causing this component to fail
 - No safe state defined

Some ways to get around the problem

- ◆ If the COTS is an operating system (OS)
 - Some real-time OSs are certified (VxWorks, OSE) – but expensive and not necessarily certified to current configuration
 - A study by HSE in UK they states that “vanilla” Linux in safety related applications at SIL 1 and SIL 2
- ◆ “Proven-in-use” arguments. Probably only feasible for SIL 1
 - Reliability / Robustness may depend on environment
 - Version control?
- ◆ Exhaustive testing of COTS based on operational profiles
 - Operational profile identifies the criticality of the component and their duration and frequency of use
- ◆ “Wrapping” of the COTS
- ◆ Use diverse COTS (such as in the Safe-PC project – two different operating systems)

Assessment of COTS proposed to HSE / UK

- ◆ Adelar proposes five different types of assessment of COTS
 - Process assessment
 - Assessment of previous use
 - Black-box assessment
 - White-box assessment
 - Third party assessment
- ◆ Adelar proposes to base the assessment on a safety case approach and includes risk analysis e.g. HazOp and/or FMEA

Current computer and software relevant DNV rules

- ◆ Non-rotating bulk data store
- ◆ Required response times
- ◆ Detection of execution failures of individual modules are required
- ◆ Discrimination of faulty modes to ensure maintained operation at least of modules with same or higher priority
- ◆ Software development process requirements (ISO 9001)
- ◆ Redundancy
- ◆ Separate hardwired emergency systems
- ◆ Monitoring of functions, watchdogs

Example of evaluation of COTS software and hardware

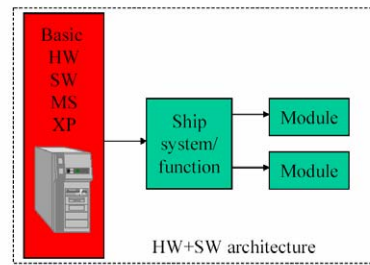


Fig.1. Maximum required response time for MS-XP computer to transfer signals to safe state is 0.5 second. Computer hardware and software are not specified to handle this requirement. This solution is assumed to be **not acceptable** as there is no provisions for ensuring the required time response.

Another example

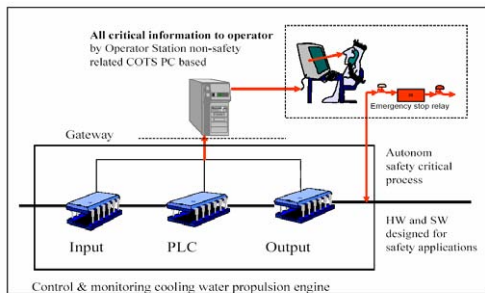


Fig. 3. Operator is assumed to get time critical information through PC non-safety specified COTS HW&SW. This solution is **not acceptable** as a PC based system can not support time critical information and has no self-monitoring or watch dogs.

Still another example

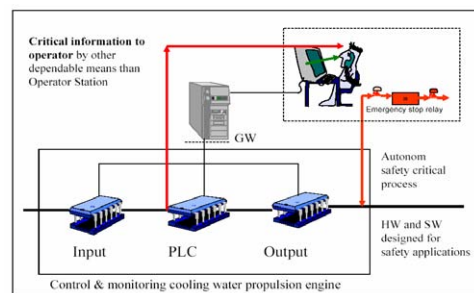
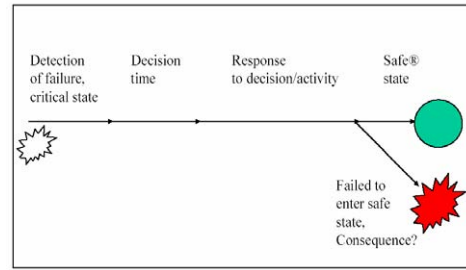


Fig. 2. Operator obtains critical information about e.g. cooling water for propulsion engine by horn or light signals not connected to non-safety specified Operator Station (OS) COTS module. This solution is assumed to be **acceptable** as the safety functionality is assumed to be handled by horn or light signals and the independent emergency stop relay.

Main recommendations - requirements

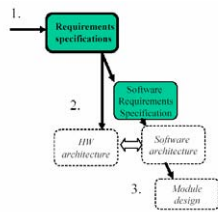
- ◆ Normal operational states for normal operation
- ◆ Possible critical failures in normal operation and possible safe states
- ◆ How to detect critical failures?
- ◆ What should be activated when a critical situation is detected
- ◆ What is the maximum response time for bringing the system to safe state?
- ◆ What is the reliability for bringing the system to safe state?
- ◆ What is the possible consequence of not bringing the ship / system to safe state?
- ◆ COTS should be considered as black boxes
- ◆ Non-safety COTS may be embedded in a separate external safety layer

Response time to transfer to safe state



Criticality assignment to COTS is done top down

1. Overall requirements should be stated for the function/system to the supplier
2. Overall requirements are then divided into requirements to the HW+SW architecture and modules by the supplier
3. A COTS SW module/submodules may be assigned more or less critical requirements to functionality



COTS wrapped in safety layer

No standard requirement can be given to a non-safety specified COTS which will ensure that the component itself will satisfy the given requirements to the safety function

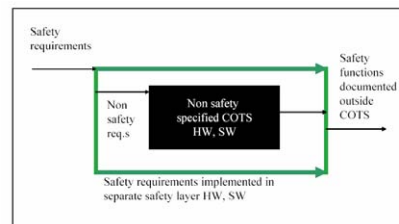


Fig. 9. Safety requirements may be implemented in a separate external safety layer independent of the non-safety specified COTS.

Finally - a paradox

Sarbanes-Oxley and Software Projects

"If we managed finances in companies the way we manage software...somebody would go to prison."

Component-based context dependent hybrid property prediction

- Anders Möller^{1,3}, Ian Peake², Mikael Nolin^{1,3}, Johan Fredriksson¹, Heinz Schmidt²
 - Mälardalen Uni, MRTC, Västerås, Sweden (1)
 - Monash University, DSSE, Melbourne, Australia (2)
 - CC Systems, Vasteras, Sweden (3)

Overview / motivation

- component-based software engineering for embedded systems: reusability, maintainability
- variability: product line architectures; function vs properties
- existing component-based techniques do not guarantee efficient resource usage
- proposal: modify existing methods to be *context-dependent*
- use Dependent Finite State Machines (DFSMs)
- focus on WCET for this presentation (WCET is prerequisite for scheduling)

Hybrid WCET prediction

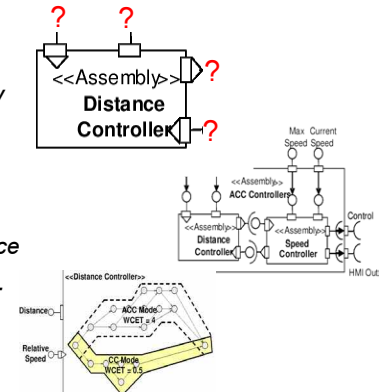
- Prediction alone *inaccurate, pessimistic, expensive*
- Measurement alone rarely finds true WCET bound (not dependable)
 - too many paths to try
 - low-level effects (e.g. cache)
- Combine prediction and measurement to eliminate weaknesses

```

DistanceController(
  int speed, bool speedE,
  int distance, bool distanceE
)
{
  // check both rel. speed and distance
  if (distE && speedE) {
    // accelerations
    difference (...) // or differential (...)
    // time-to-impact
    add (...) // or integral (...)
  }
  // speed only
  if (speedE && !distE) {
    ...
  }
  // distance only
  if (distE && !speedE) {
    ...
  }
}
    
```

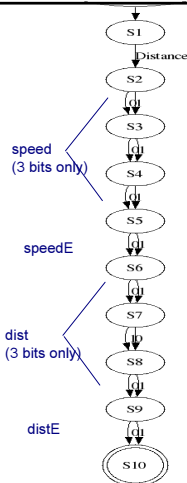
Hybrid WCET prediction: problem

- whole-of-system approach
 - test results overly pessimistic for some contexts / results not easily reused
 - requires full source
- need for "context-dependent" approach



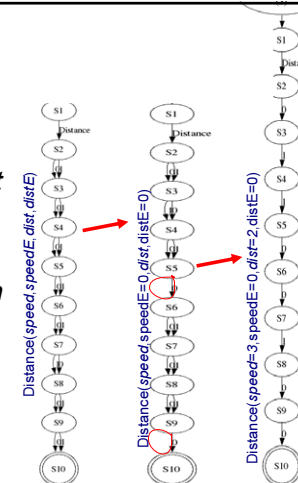
DFSMs and protocol types

- DFSMs (from "RADL") model all a component's externally visible characteristics
- protocol types model individual component interfaces (ports), enabling e.g. behavioural contracts
- protocol types defined as regular languages (with straightforward extension to concurrency/trace languages)



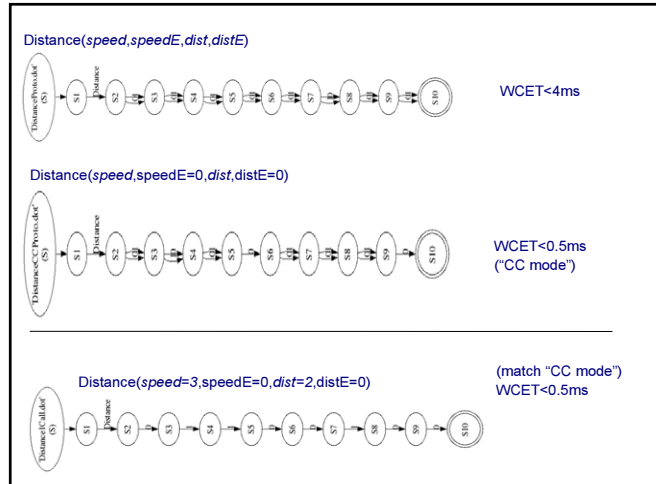
Protocol restriction

- protocol restrictions model reduced use of interface in a component deployment context--- fewer behaviours
- protocol (type) restriction is regular language restriction (subset of accepted strings)
- narrower contexts reduce WCET



Context-dependent property analysis

- *context* of component type T == restriction of T's protocols
- e.g.: the universal context of T is simply its DFSA's original gate protocols (trivial restriction)
- qualify ("*guard*") property V of some T with a context C: $\langle V, C \rangle$
- $\langle V, C1 \rangle$ only valid in a given context C2 if C2 equals or (further) restricts C1
- analysis techniques adapted to handle context
- properties reusable only if guards satisfied



Context-dependent hybrid prediction

- *per component*
- determine candidate worst-case paths and preconditions (using DFSA ops), per "mode"
- test only "worst-case" paths
- where context is obscured by difficult-to-analyse patterns, "degrade gracefully" to context-independent worst-case scenario
- "worst case" test cases may be reusable on similar architectures

Context-dependent hybrid prediction

- protocol types function as contracts: global analyses may be adapted, by enforcing consistency of needed assumptions
- e.g.: if variable range analysis determines (and depends on) param $V2\{1,2,3,5\}$, this is included in the context
- both static and dynamic aspects of contexts may be modelled using DFSMs

Related work (e.g.)

- "Measurement-based Worst-Case Execution Time analysis using Automatic Test Data Generation"
 - Kirner, Puschner, Wenzel", in Proc WCET04, 2004
 - Hybrid WCET:
 - measurement of WCET of program segments + static WCET
 - model checker constructs test cases
 - reduce path explosion by condensing low level code
 - reduce state space to improve model checking performance
 - Full prototype, feasibility case study
 - Monolithic / whole-of-system approach:
 - for accuracy, each new component/system of interest must be analysed as a single unit

Evaluation

(Context-dependent prediction)

- "Extra-functional Consistency and Prediction of Component-Based Control Systems" (eCAP) project
- research collaboration Monash-ABB (German research centre, Ladenburg)
- prototype of context-dependent WCET
 - prediction of IEC 6 1131-3 code WCET on ABB Advant Controllers
 - context-dependent prediction architecture in place

Evaluation (Context-dep. prediction)

- requires adaptation of existing techniques, e.g. range analysis
 - does not address pessimistic WCET bounds due to cache/pipelining/branch prediction
- risk of complexity overwhelming analysis cost:
 - finding “worst-case paths” (plural), i.e., modes might be expensive
 - risk of context explosion? (too many choices or combinations of context per component type)
 - need to model concurrency semantics of inter-component communication

Evaluation (hybrid context-dependent WCET)

- Not yet:
 - simple measurement tool based on random inputs was prototyped by ABB; may be extended

Conclusions / future work

- Initial evaluation promising
- Potential to increase accuracy and efficiency of properties
- Applicable to other behaviourally-derivable, compositional properties (space, reliability, with probabilistic extensions);
- Hybrid scheduling: c.f. hybrid resource reclamation methods using WCET probability distributions (MRTC)

Discussion..

All participants of the workshop and the co-located Euromicro conferences

Prof. Einar J. **Aas**, NTNU, Electronics and Telecommunications, Norway

Ms. Takoua **Abdellatif**, BULL SA / INRIA, France

Mr. Nabil **Abdelli**, Thales Communications, France

Mr. Pekka **Abrahamsson**, Technical Research Centre of Finland, VTT Electronics, Finland

Prof. Luciano **Agostini**, Federal University of Rio Grande do Sul, Federal University of Pelotas - UFPel, Brazil

Mr. Marcus **Alanen**, Åbo Akademi University, Finland

Mr. Soyeb **Alli**, University of York, Department of Computer Science, United Kingdom

Mr. Alexandre **Alvaro**, VALDIR ALVARO, Maria da Graca Pandur Alvaro, Brazil

Mr. Per **Andersson**, Lund University, Dept. of Computer Science, Sweden

Prof Mark **Arnold**, Lehigh University, USA

Mr. Dejan **Baca**, , Sweden

Mr. Olivier **Barais**, Uni. des Sciences et Technologies de Lille, Jacquard INRIA Project, LIFL, France

Mr. Krzysztof **Berezowski**, Wroclaw Univ. of Technology, Inst. of Engineering Cybernetics, Poland

Mr. Rani **Bhutada**, Albert-Ludwigs-University of Freiburg, Chair of Microelectronics, IMTEK, Germany

Prof. Stefan **Biffi**, TU Vienna, Austria

Mrs Ana Paula **Blois**, Federal University of Rio de Janeiro, COPPE- System Engineering and Computer Science Pro, Brazil

Mr. Maik **Boden**, Fraunhofer IIS EAS, Germany

Msc Egor **Bondarev**, Eindhoven University of Technology, WSKI HG 5.04, Netherlands

Mr. Yannick **Bonhomme**, CEA / LCSD, Laboratoire Conception des Systèmes Durcis, France

Prof. Melvin **Breuer**, University of Southern California, EE-Systems Dept., USA

Dr. Fernando **Brito e Abreu**, QUASAR Research Group, Departamento de Informática, FCT/UNL, Portugal

Mr. Oswaldo **Cadenas**, University of Reading, School of Systems Engineering, United Kingdom

Mr. Carlos **Calafate**, Polytechnic University of Valencia, Dept. of Computer Engineering (DISCA), Spain

Mr. Bernard **Candaele**, THALES, France

Prof. Gregorio **Cappuccino**, University of Calabria, DEIS-UNICAL, Italy

Mr. Bengt **Carlsson**, Blekinge Institute of Technology, Sweden

Mr. Antonio **Castro**, AJCASTRO.COM, Cons. Inf., Lda., Portugal

Mr. Herve **Chang**, University of Nice Sophia Antipolis, I3S Laboratory, France

Mrs Maria **Charalambous**, University of Cyprus, Department of Computer Science, Cyprus

Dr. Michel **Chaudron**, Technische Universiteit Eindhoven, Mathematics and Computing Science, The Netherlands

Prof. Gerhard **Chroust**, J.Kepler University Linz, System Engineering and Automation, Austria

Mr. Vladimir **Ciric**, University of Nis, Faculty of Electronic Engineering, Serbia and Montenegro

Mrs Lucía **Costas Pérez**, University of Vigo, Department of Electronic Technology, Spain

Dr. Ivica **Crnkovic**, Mälardalen University, Department of Computer Science and Engineering, Sweden

Mr. Danielly **Cruz**, Federal University of Pernambuco, Centro de Informática, Brazil

Mr. Robert **Czerwinski**, Silesian University of Technology Gliwice, Institute of Electronics, Poland

Mr. Guglielmo **De Angelis**, ISTI - CNR, Italy

Mr. Francesca **Di Pillo**, University of Rome Tor Vergata, Italy

Mr. Glen **Dobson**, Lancaster University, Department Computing, United Kingdom

Dipl.Inform. Thomas **Dreibholz**, University of Duisburg-Essen, Institute for Experimental Mathematics, Germany

DI Walter **Dürr**, Austrian Aerospace GmbH, Austria

Mr. Colin **Egan**, University of Hertfordshire, Dept. of Computer Science, United Kingdom

Ms Golnaz **Elahi**, Amirkabir University of Technology, Computer Engineering and Information Technology Dept., Iran

Mr. Jad **El-khoury**, KTH, Department of machine design, Sweden

Mr. Peeter **Ellervee**, Tallinn University of Technology, Estonia

Prof. Luca **Fanucci**, Pisa University, Dept. Information Engineering, Italy

Mr. Edil S T **Fernandes**, Federal Univ.of Rio de Janeiro, Dept. of Comp.Science/COPPE, Brazil

Mr. Jorge **Fernandez**, Instituto Nacional de Tecnología Industrial, Programa de Software, Argentina

Eng Mónica **Figueiredo**, University of Aveiro, Instituto de Telecomunicações, Portugal

Mrs. Maria **Fischerova**, Slovak Academy of Sciences, Institute of Informatics, Slovakia

Mr. Petr **Fiser**, Czech Technical University in Prague, Computer Science and Engineering, Czech Republic

Mr. Oana **Florescu**, Eindhoven University of Technology, Electrical Engineering Department, The Netherlands

Mr. Anibal **Foti**, Instituto Nacional de Tecnología Industrial, Argentina

Dr. Michael **Freeman**, University of York, Dept. of Computer Science, United Kingdom

Mr. Fakhreddine **Ghaffari**, CNRS, I3S les algorithmes, France

Mr. Yashar **Ghiassi**, Sharif University of Technology, Iran

Mr. Øystein **Gjermundnes**, NTNU, Department of Electronics and Telecommunications, Norway

Mr. Jan **Gottschick**, Fraunhofer ISST, Germany

Mr. Miguel **Goulão**, QUASAR Research Group, Departamento de Informática, Portugal

Prof. Dr. Tom **Gross**, Bauhaus-University Weimar, Faculty of Media, Germany

Mr. Karl-Erwin **Grosspietsch**, Fraunhofer Gesellschaft, AIS, Germany

Mr. Paul **Grünbacher**, Kepler University Linz, System Eng. and Automation, Austria

Prof. Vladimir **Hahanov**, Kharkov National University of Radio Electronics, DA Department, Ukraine

Mr. Panu **Hämäläinen**, Tampere University of Technology, Institute of Digital and Computer Systems, Finland

Mr. Matthias **Handy**, University of Rostock, Institute of Applied Microelectronics and CE, Germany

Dr. Marko **Hännikäinen**, Tampere University of Technology, Institute of Digital and Computer Systems, Finland

Mr. Zhiyuan **He**, Linköping University, Dept. of Computer and Information Science, Sweden

Mr. Michael **Hicks**, University of Hertfordshire, Computer Science, Uk

Mr. Markus **Hillenbrand**, University of Kaiserslautern, Faculty of Computer Science, Germany

Mr. Didier **Hoareau**, Valoria Lab, University of South Brittany, France

Mr. Chung-Ming **Huang**, Nat. Cheng Kung University, Dept. of Computer Science and Information Engineering, Taiwan, Roc

Mr. John **Hutchinson**, Lancaster University, Computing Department, United Kingdom

Prof. Yukihiko **Iguchi**, Meiji University, Department of Computer Science, Japan

Dr. Sylvia **Ilieva**, Sofia University, Faculty of Mathematics and Informatics, Department of IT, Bulgaria

Mr. Pooya **Jaferian**, Amirkabir University of Technology, Computer Engineering and Information Technology Dept., Iran

Ms Diane **Jamieson**, Curtin University of Technology, Curtin Business School, Australia

Mr. Gert **Jervan**, Tallinn University of Technology, Estonia

Prof. Lech **Jozwiak**, Eindhoven University of Technology, Faculty of Electrical Engineering, The Netherlands
Prof. Guy **Juanole**, University of Toulouse III LAAS-CNRS, France
Dr. Artur **Jutman**, Tallinn University of Technology, Department of Computer Engineering, Estonia
Dr. Mira **Kajko-Mattsson**, Stockholm University and Royal Inst. of Technology, Dept. of Computer and Systems Sciences, Sweden
Dr. Zoran **Kalafatic**, University of Zagreb, Faculty of Electrical Engineering and Computing, Croatia
Mr. Dariusz **Kania**, Silesian University of Technology Gliwice, Institute of Electronics, Poland
Mr. Daniel **Karlsson**, Linköping University, Sweden
Mr. Nikolay **Kavaldjiev**, University of Twente, Computer Science and Applied Mathematics, DIES, Netherlands
Mr. Kenneth **Kent**, University of New Brunswick, Faculty of Computer Science, Canada
Mr. Heikki **Keränen**, VTT Technical Research Centre of Finland, VTT Electronics, Finland
Mr. Osnat **Keren**, Bar-Ilan University, Israel
Mr. Konrad **Klößner**, Fraunhofer Gesellschaft, Inst. for Applied Information Technology FIT, Germany
Mr. Mikko **Kohvakka**, Tampere University of Technology, Institute of Digital and Computer Systems, Finland
Mrs Sabine **Kolvenbach**, Fraunhofer Gesellschaft FIT, Germany
Mr. Zdenek **Kotasek**, Brno University of Technology, Faculty of Information Technology, Department of C, Czech Republic
Mr. Milos **Krstic**, IHP, Germany
Ing. CSc. Hana **Kubátová**, Czech Technical University in Prague, Computer Science and Engineering, Czech Republic
Mr. Krzysztof **Kuchcinski**, Lund University, Dept. of Computer Science, Sweden
Mr. Rikard **Land**, Mälardalen University, Dept. of Computer Science and Electronics, Sweden
Mr. Anders **Larsson**, Linköpings Universitet, Sweden
Dr. Kung-Kiu **Lau**, The University of Manchester, Dept. of Computer Science, U. K.
Mr. Weng Fook **Lee**, Senior MTS, Emerald Systems Design Center, Penang
Prof. Francesco **Leporati**, University of Pavia, Dipartimento Informatica e Sistemistica, Italy
Mr. Andreas **Lindahl**, Chalmers University of Technology, Dept. of Computer Science and Engineering, Sweden
Mr. Tadeusz **Luba**, Warsaw Univ. of Technology, Inst. of Telecommunications Fundamentals, Poland
Dr. Ketil **Lund**, Simula Research Laboratory, Norway
Mr. Trygve **Lunheim**, Norwegian University of Technology and Science NTNU, Engineering Cybernetics, Norway
Mr. Hugh **Maaskant**, Philips Research, The Netherlands
Dr. José **Machado da Silva**, FEUP, INESC Porto, Portugal
Dr. K.L. **Man**, Eindhoven University of Technology, The Netherlands
Ing. Tomas **Marek**, Czech Technical University in Prague, Faculty of Electrical Engineering, Czech Republic
Mr. Sérgio **Martins**, Universidade do Porto, Faculdade de Engenharia, Portugal
Mr. Farhad **Mehdipour**, Qazvin Islamic Azad University, Electrical and Computer Engineering, Iran
Mr. Andreas **Meissner**, Fraunhofer IPSI - Integrated Publication and Info, Germany
Mr. David **Melendi**, Uni. of Oviedo, Computer Science Department, Spain
Mr. Ivan **Milentijevic**, University of Nis, Faculty of Electronic Engineering Computer Science Dept., Serbia And Montenegro
Mr. Nils Brede **Moe**, SINTEF ICT, Norway
Mr. Anders **Möller**, Mälardalen University, Sweden

Ms Sarah **Mount**, Coventry University, Uk

Dr. Luiza de Macedo **Mourelle**, State University of Rio de Janeiro, System Engineering and Computation, Brazil

Prof. Dr. Paul **Müller**, University of Kaiserslautern, Germany

Prof. Kazuaki **Murakami**, Kyushu University, Department of Informatics, Japan

Mr. Leonardo **Murta**, COPPE/UFRJ, Brazil

Mr. Fredy **Navarrete**, Politècnica de Catalunya, Spain

Mr. Tien **Nguyen**, University of Wisconsin, Usa

Mr. Claus W.L. **Nielsen**, Scandinavian Airlines Systems, Airline IT, Denmark

Mr. Antti U. **Niskanen**, VTT Technical Research Centre of Finland, Telecommunication Systems, Finland

Prof Antonio **Nunez**, Universidad Las Palmas de GC, IUMA Institute for Applied Microelectronics, Spain

Mr. Tero **Nurmi**, University of Turku, Department of Information Technology, Finland

Mr. Valentine **Ogier-Galland**, NexWave Solutions, France

Eng. Arnaldo **Oliveira**, Universidade de Aveiro, Electrónica e Telecom, Portugal

Dr. Hannes **Omasreiter**, DaimlerChrysler AG, Germany

Mr. Roberto R. **Osoario**, University of Santiago de Compostela, Dept. of Electronics and Computer Science, Spain

Prof Kone **Ousmane**, CNRS Toulouse - France, France

Mr. Danilo **Pani**, DIEE - University of Cagliari, Dept. Electrical and Electronic Engineering, Italy

Mr. Piotr **Patronik**, Wroclaw University of Technology, Poland

Dr. Ian **Peake**, Monash University, Center for Distributed System and Software Engineering, Australia

Dr. Witold **Pleskacz**, Warsaw University of Technology, Institute of Microelectronics and Optoelectronics, Poland

Mr. Andrea **Polini**, Istituto di Scienza e Tecnologie dell'Informazione, (ISTI/CNR), Italy

Mr. Andreas **Putzinger**, FIM, Johannes Kepler University Linz, Inst. for Information Processing and Microprocessor Techn., Austria

Mr. Yang **Qu**, VTT Technical Research Center of Finland, VTT Electronics, Finland

Mr. Ricardo Gonçalves **Quintão**, Centro Universitário da Cidade — UNIVERCIDADE, Federal University of Rio de Janeiro, Brazil

Mr. Oddny **Ra**, Buskerud University College, Norway

Dr. Oystein **Ra**, Buskerud University College, Norway

Mr. Rudolf **Ramler**, Software Competence Center Hagenberg, Austria

Mr. Tomi **Räty**, VTT Technical Research Centre of Finland, Electronics, Finland

Mr. Benoît **Ries**, University of Luxembourg, Luxembourg

Dr. Ana **Ripoll**, University Autònoma of Barcelona, Computer Science Dept., Spain

Mr. Sami **Rosendahl**, SysOpen Digia Plc., Smartphone Business Division, Finland

Mr. Steffen **Ruelke**, Fraunhofer IIS EAS, Germany

Dr. Richard **Ruzicka**, Brno University of Technology, Faculty of Information Technology, Czech Republic

Mr. Hamid **Safizadeh**, Amirkabir University of Technology, Dept. of Computer Engineering and Information, Iran

Mr. Alberto **Sampaio**, Instituto Superior de Engenharia do Porto, Portugal

Mr. Marcos **Sanchez-Elez Martin**, Universidad Complutense de Madrid (UCM), Arquitectura de Computadores y Automatica, Spain

Dr. Titos **Saridakis**, NOKIA, Finland

Mr. Soujanya **Sarkar**, Texas Instruments (India) Ltd, Digital Signal Processing Systems, India

Prof. Tsutomu **Sasao**, Kyushu Institute of Technology, Dept. of Computer Science and Electronics, Japan

Mr. Erwin **Schoitsch**, ARC Seibersdorf Research, Austria
Prof. Rimantas **Seinauskas**, Kaunas University of Technology, Lithuania
Mr. Md. Sumon **Shahriar**, Univ. of Dhaka, Dept. of Computer Science and Engineering, Bangladesh
Mr. Huibin **Shi**, University of York, Department of Computer Science, United Kingdom
Dr. José Luis **Sierra**, Universidad Complutense de Madrid, Dpto. Sistemas Informáticos y Programación, Spain
Mr. Miguel **Silva**, FEUP/DEEC, Portugal
Dr. Amund **Skavhaug**, Norwegian Univ.of Techn. and Science, Engineering Cybernetics, Norway
Prof. Torbjorn **Skramstad**, Norwegian University of Science and Technology, Det Norske Veritas Research, Norway
Prof. Janusz **Sosnowski**, Warsaw Univ. of Technology, Inst. of Computer Science, Poland
Mr. Enrique **Soto Campos**, Universidade de Vigo, Dept. Tecnoloxía Electrónica, Spain
Dr. Christoph **Steindl**, IBM Austria, Austria
Mr. Sander **Stuijk**, TU Eindhoven, The Netherlands
Mr. Øyvind **Teig**, Autronica Fire and Security, Norway
Mr. Egon **Teiniker**, Salomon Automation G.m.b.H, Institute for Technical Informatics, Austria
Prof. François **Terrier**, CEA-List, France
Mr. Bart **Theelen**, Eindhoven Univ. of Technology, Electrical Engineering, The Netherlands
Prof. Dr. Dirk **Timmermann**, University of Rostock, Institute of Applied Microelectronics and CE, Germany
Mr. Massimo **Tivoli**, University of L'Aquila, Italy
Mr. Åsmund **Tjora**, NTNU, Norway
Mr. Martin **Törngren**, KTH, Division of Mechatronics, Sweden
Mr. Pedro **Trancoso**, University of Cyprus, Department of Computer Science, Cyprus
Dr. Li **Tun**, National University of Defense Technology, China
Mr. Raimund **Ubar**, Tallinn Technical University, Estonia
Mr. Manuel **Uruena-Pascual**, Universidad Carlos III de Madrid, Departament of Telematic Engineering, Spain
Mr. Tero **Vallius**, University of Oulu, Information and Electrical engineering, Finland
Dr. Rob **van Ommering**, Philips NatLab, The Netherlands
Mrs Paula **Ventura Martins**, INESC-ID, Universidade do Algarve, Portugal
Mr. Heikki **Verta**, Nokia Research Center, Finland
Mr. Kashif **Virk**, Technical University of Denmark, Denmark
Dr. Salvatore **Vitabile**, ICAR - Istituto di CALcolo e Reti ad alte prestazioni, Italian National Research Council, Italy
PDEng Lucian **Voinea**, Technische Universiteit Eindhoven, WSKI HG 5.04, The Netherlands
Mr. Klaus **Waldschmidt**, J.W. Goethe Universität, Technische Informatik, Germany
Mr. Zheng **Wang**, The University of Manchester, School of Computer Science, UK
DI. Georg **Weissenbacher**, ARC Seibersdorf Research, Austria
Mrs. Claudia **Werner**, COPPE/UFRJ - Federal University of Rio de Janeiro, Brazil
Mr. Hans **Westerheim**, SINTEF ICT, Norway
Zhaojun **Wo**, University of Massachusetts, Amherst, Usa
Mr. Christophe **Wolinski**, IRISA/IFSIC, France
Dr. Dong **Wu**, School of Electronics and Computer Science, University of Southampton, UK
Mr. Ge **Zhang**, University of Kaiserslautern, Computer Science AG ICSY, Germany

About ERCIM

ERCIM — the European Research Consortium for Informatics and Mathematics — fosters collaborative work within the European research community and aims at increasing cooperation with European industry. The members of ERCIM include leading research establishments from seventeen European countries. Encompassing over 12,000 researchers and engineers, ERCIM is able to undertake consultancy, development and educational projects on any subject related to its field of activity. ERCIM was founded in 1989 and is a European Economic Interest Grouping (EEIG). ERCIM is the European host of the World Wide Web Consortium (W3C).

ERCIM office

BP 93, 2004 route des Lucioles, F-06902 Sophia Antipolis

Tel: +33 4 92 38 50 10; Fax +33 4 92 38 50 11; E-mail: office@ercim.org

<http://www.ercim.org>

ISBN: 2-912335-15-9
