APRES 2012

Proceedings of the 4th Workshop on Adaptive and Reconfigurable Embedded Systems

Beijing, China
April 16, 2012

In conjunction with CPSWeek 2012
April 16 – 19, 2012

Edited by Paulo Pedreiras and Insik Shin

Workshop Organization

Program Chairs

Paulo Pedreiras	University of Aveiro
Insik Shin	KAIST

Program Committee

Madhukar Anand	Cisco
Neil Audsley	University of York
Antonio Casimiro	University of Lisbon
Jian-Jia Chen	ETH
Petru Eles	Linkoepping University
Joaquim Ferreira	University of Aveiro
Sebastian Fischmeister	University of Waterloo
Nathan Fisher	Wayne State University
Marisol Garcia-Valls	University Carlos III in Madrid
Sathish Gopalakrishnan	University of British Columbia
MoonZoo Kim	KAIST
Christoph Kirsch	University of Salzburg
Karthik Singaram Lakshmanan	CMU
Chang-Gun Lee	Seoul National University
Jinkyu Lee	University of Michigan
Jane Liu	Academia Sinica
Ricardo Marau	University of Porto
Pau Marti	Technical University of Catalonia
Thomas Nolte	Malardalen University
Roman Obermaisser	University of Siegen
Sangsoo Park	Ewha Womans University
Carlos Eduardo Pereira	UFRG
Stefan Petters	ISEP - IPP
Linh Thi Xuan Phan	University of Pennsylvania
Paulo Portugal	University of Porto
Dumitru Potop-Butucaru	INRIA
Lei Rao	General Motors Research Center
Eric Rutten	INRIA Grenoble
Ina Schaefer	Technical University Braunschweig
Mario Sousa	University of Porto
Martin Torngren	KTH
Gera Weiss	Ben Gurion University of the Negev
Shaofa Yang	Shenzhen Institutes of AdvancedTechnology
Fan Zhang	Chinese Academy of Sciences

Steering Committee

Luis Almeida	University of Porto
Karl-Erik Årzén	Lund University
Sebastian Fischmeister	University of Waterloo
Insup Lee	University of Pennsylvania
Julián Proenza	University of the Balearic Islands

Foreword

Welcome to Beijing, and the 4th Workshop on Adaptive and Reconfigurable Embedded Systems (APRES 2012). The purpose of this workshop is to provide an open forum to discuss new and on-going research that is centered on the idea of adaptability as first class citizen and that considers the tradeoffs between flexibility and complexity.

The APRES program includes a keynote presentation on “Network Challenges in Cyber-Physical Systems“, regular paper sessions with 7 peer-reviewed papers and 1 invited paper, and a short paper session with 3 peer-reviewed extended abstracts, covering new and important issues such as adaptive resource managements for GPU-based embedded systems and virtualized real-time systems, configurable fault-tolerance and energy-optimization for automotive systems, and adaptive automata -- just to name a few. In addition, we also have a best paper award for the first time.

The APRES 2012 would not occur without tremendous efforts of many individuals. We would like to take this opportunity to thank all the people involved. Thank you to the authors who submitted their work and the program committee for their work in reviewing the papers and helping to make the workshop a success. Special thanks to Luis Almeida, for accepting our invitation to be keynote speaker for the event, and to the Steering Committee members Luis Almeida, Karl-Erik Årzén, Sebastian Fischmeister, Insup Lee and Julián Proenza for their support and assistance in organizing the workshop.

We hope you enjoy APRES 2012, and find the technical program interesting and stimulating. We wish you a wonderful stay in the amazing city of Beijing, and look forward to seeing you again in 2013.

Paulo Pedreiras and Insik Shin
Co-chairs
4th Workshop on Adaptive and Reconfigurable Embedded Systems (APRES 2012)

Workshop Program

9:00 – 9:10 Welcome

9:10 – 10:10 **Keynote Talk**

- Network Challenges in Cyber-Physical Systems
  Luis Almeida, University of Porto, Portugal

10:10 – 10:30 Coffee Break

10:30 – 12:00 **Session 1: Adaptive resource management**

- Towards Adaptive GPU Resource Management for Embedded Real-Time Systems
  Junsung Kim, Raj Rajkumar and Shinpei Kato
- Towards Adaptive Resource Management for Virtualized Real-Time Systems
  Stefan Groesbrink, Simon Oberthur and Daniel Baldin
- Control Kernel Based Adaptive Control Implementation
  Raúl Simarro, Pedro Albertos and José Simó

12:00 – 1:00 Lunch

1:00 – 2:15 **Session 2: Automotive applications and beyond**

- SAFER: System-level Architecture for Failure Evasion in Real-time Applications
  Junsung Kim, Raj Rajkumar and Markus Jochim
- Network-Wide Energy Optimization for Adaptive Embedded Systems
  Patrick Heinrich and Christian Prehofer
- A Virtualization-based Fault Resilient Secure Real-Time System Architecture
  Daeyoung Hong, Wonsek Ko, Sung-Soo Lim
- Adaptive Trajectory Coordination for Scalable Robot Motion Planning
  Hoon Sung Chwa, Andrii Shyshkalov, Jinkyu Lee, Hyoungbu Back, and Kilho Lee
- A Self-adaptive Unifying Mechanism For Autonomous Energy Management In Wireless Sensor Networks
  Lina Xu, Olga Murdoch, Gregory O'Hare and Rem Collier

2:15 – 2:45 Coffee Break and Poster Session

2:45 – 4:15 **Session 3: Adaptive formal methods**

- Model-driven development of SOA-based Driver Assistance Systems
  Marco Wagner, Dieter Zobel and Ansgar Meroth

- Modeling and Analysis of Adaptive Embedded Systems using Adaptive Task Automata
  Leo Hatvani, Paul Pettersson and Cristina Seceleanu.

- Provisioning within a WSAN Cloud Concept
  Muhammad Sohaib Aslam, Susan Rea and Dirk Pesch

Keynote Talk

Network Challenges in Cyber-Physical Systems

Luis Almeida, University of Porto, Portugal

Talk Abstract

Cyber-Physical Systems (CPS) rely, up to a large extent, on networking infrastructures, frequently large ones. These necessarily play a central role in supporting the needed system-wide properties, being timeliness a particularly important one as dictated by the dynamics of the associated physical processes.

We claim that emerging CPS applications, such as Smart-Grids, Remote Interaction, Collaborative Robotics, etc, require openness together with tighter timeliness guarantees. We postulate that such guarantees can only be achieved with an adequate communication abstraction supported on adequate protocols. To this end, we have been proposing channel reservation-based communication as a means to provide scalable and open latency-constrained communication and thus enable a more efficient design of CPS.

This presentation will provide a brief tour of our recent work on flexible and composable approaches to real-time communication for distributed embedded systems. This work provides a basis to address the network access problem. We developed solutions for two typical scenarios, one in which we can exercise effective control over the network, typically based on wired media, and another one in which such control is loose and more adaptation to dynamic conditions is needed as in wireless cases.

We will end with our recent efforts towards scaling some of the previous approaches to provide end-to-end guarantees, highlighting some open challenges towards adequate networking infrastructures for effective CPS.

Speaker Biography

Luis Almeida is currently an associate professor at the Electrical and Computer Engineering Department of the University of Porto and a member of the Institute of Telecommunications in Porto where he coordinates the Distributed Real-Time and Embedded Systems Lab. He is also a member of the IEEE, Computer Society, and particularly of its Technical Committee on Real-Time Systems, member of the IFIP Tecnical Committee on Embedded Systems, member of the Strategic Management Board of the EU/ICT NoE ArtistDesign, leading the Real-Time Networks activity in that NoE, Vice-President of the RoboCup Federation and President of the Portuguese RoboCup National Committee.

His current interests are real-time communication protocols for Cyber-Physical Systems with an emphasis on mechanisms to support predictable operational flexibility as needed for dynamic QoS management, graceful degradation and open distributed real-time systems in general. He co-authored over 200 refereed publications, 3 patents and 8 book chapters. He regularly participates in the organization and program committees of scientific events in the Real-Time Systems, Industrial Systems and Robotics communities, including RTSS, ECRTS, DATE, SIES, WFCS, ETFA and RoboCup.

Towards Adaptive GPU Resource Management for Embedded Real-Time Systems

Junsung Kim and Ragunathan (Raj) Rajkumar

Department of Electrical and Computer Engineering
Carnegie Mellon University

Shinpei Kato

Department of Computer Science
University of California, Santa Cruz

Abstract

In this paper, we present two conceptual frameworks for GPU applications to adjust their task execution times based on total workload. These frameworks enable smart GPU resource management when many applications share GPU resources while the workloads of those applications vary. Application developers can explicitly adjust the number of GPU cores depending on their needs. An implicit adjustment will be supported by a run-time framwork, which dynamically allocates the number of cores to tasks based on the total workload. The runtime support of the proposed system can be realized using functions which measure the execution times of the tasks on GPU and change the number of GPU cores. We motivate the necessity of this framework in the context of self-driving technologies, and we believe that our frameworks for GPU programming are useful contributions given the increasing emphasis on parallel heterogeneous computing.

1 Introduction

Graphics processing units (GPUs) are becoming more and more commonplace in many application domains widely ranging from high-performance computing to embedded mobile computing. For example, three of the top five supercomputers on the TOP500 list [12], announced as of March 2012, use GPUs to accelerate computations, while recent tablets, such as ASUS Eee Pad Transformer Prime, also leverage embedded GPUs, like Tegra 3 [9], to enhance performance under power constraints. This trend is expected to continue.

One notable application domain of GPUs is automotive engineering. Modern automobiles employ several tens of processing units. Further advances in safe-driving features, such as adaptive cruise control, stop-and-go cruise control, lane keeping, and assisted lane change, would require even larger computing capabilities. For vehicles to become fully or semi-autonomous, a multitude of computer vision, sensor fusion, signal processing, and graphics sub-systems must operate and communicate in real-time. Given their highly data-parallel and compute-intensive workloads, parallel computing is a useful solution. As technology stands today, the GPU is the most well-suited platform. In fact, NVIDIA GPUs will be used for infotainment systems platforms in future product lines of BMW vehicles [10].

Automatic safety features require smart planning and intelligent processing of data obtained from many sensors equiped in the vehicle, including LIDAR (LIght Detection And Ranging), radar, camera, and ultrasonic sensors. One common characteristic in these types of processing is that GPUs can accelerate their processing speeds significantly. For example, autonomous driving should ideally follow the best path among many potential paths, whose calculations can happen in parallel. Calculating as many potential paths to follow as possible will yield better quality of driving. As a matter of fact, CMU's autonomous vehicle team showed that their motion planning algorithm was sped up by 40 times [6], using an NVIDIA GTX 260 GPU that integrates 192 compute cores on a chip.

In addition to motion planning, a perception algorithm as well as sensor data processing can benefit from the GPU. The perception system of a self-driving car should be able to detect, classify and track the obstacles around itself. Various types of sensors will generate voluminous amount of information that must be processed in order to understand the vehicle's surroundings. For example, a self-driving car at CMU manages 1536 objects from LIDAR sensors before they are fused with other types of sensor data. There has been on-going research using GPU to build a perception system [1], and their GPU implementation yielded 30,000-times-faster performance compared to the case of using only one CPU.

As described above, there has been research on applying GPU to different applications on self-driving cars, and it is clear that GPU can provide great benefits on realizing safer or self-driving car technologies. However, not much research has been done when those technologies are deployed together on self-driving cars, where the loads of

each application dynamically vary depending on the environment. The period and the computation time of the planning algorithms for autonomous driving highly depend on the vehicle speed, so the planning algorithms can be heavily loaded when the car is (say) on a highway. The load of the perception algorithms mainly depends on the number of obstacles around the car. Hence, the perception system requires more computing resources when the car is driving in an urban area. Therefore, an intelligent method of sharing many cores on GPU would be essential when we use GPUs on self-driving cars. For example, if a self-driving car has a 96-core GPU, the planning algorithm of the car can use 72 cores on the highway and use 12 cores in the urban environment. A self-driving car [13] requires tens of tasks, and the dynamic core management should be fulfilled across all tasks if those tasks utilize GPU.

In this paper, we present two conceptual frameworks for GPU applications to adjust their task execution times given current workload conditions. These frameworks enable smart GPU resource management when many applications share GPU resources while the workloads of those applications vary. These frameworks support both explicit and implicit adjustment. With support for explicit adjustment, application developers can adjust the number of GPU cores depending on their needs. A run-time framwork will dynamically allocate the number of cores to tasks based on current workloads. The runtime support of our proposed system can be realized using functions which measure the execution times of the tasks on GPU and change the number of cores.

The rest of paper is organized as follows. Section 2 describes how our proposed system is modeled. Section 3 presents the methods for adaptively managing GPU resources, and we conclude our paper in Section 4.

2 System Model

We assume real-time embedded systems that contain CPU and one or more GPUs as compute devices. An application task starts execution on the CPU, and offloads its data-parallel compute-intensive workload onto the GPU when needed. Once offloaded onto the GPU, the task becomes non-preemptive due to many reasons. In fact, it is technically possible to preempt the running task on the GPU by loading and restoring its context, but it requires additional firmware, runtime, and OS support, and the preemption cost would be non-trivial due to a very large set of GPU registers and states. We, hence, restrict our attention to a non-preemptive execution model for GPU computing. GPUs may also pose some constraints in multi-tasking. Even the NVIDIA Fermi architecture [8], one of the most popular GPU product lines, allows only one context to use GPU resources at once, though this context may spawn multiple GPU kernels (jobs) simultaneously. In other words, if task-level parallelism is required, the entire system must run in the same context. We, however, believe that this constraint will not limit the concepts we describe in this paper. The same GPU context can be used to exploit concurrent parallel job executions, to serve at least as a proof of concept. We also expect that future product lines will remove this concern.

We consider real-time applications where each task runs in a periodic or sporadic manner under deadline constraints. Such a task set may include motion planning and vision-based perception in state-of-the-art autonomous driving vehicles, where the periods often correspond to frame-rates, and the deadlines occur at the end of the period. We also presume that the computing demand of each task is highly variable. For example, the performance of planning and perception tasks is usually governed by the number of objects, the size of data, and the desired quality of output. These workloads are also very parallelizable using the GPU. The contributions of this paper are not limited to autonomous driving tasks but are also generally applicable to highly variable workloads running on the GPU.

3 Adaptive GPU Resource Management

In this section, we describe adaptivity support for GPU applications. We particularly focus on solving resource allocation problems. The goal is to support embedded real-time systems that exhibit highly variable workloads. Since GPUs integrate a large number of cores on a chip, we aim to enable the execution of highly variable workloads in a timely manner by adjusting allocated cores at runtime.

Several approaches have been studied for adaptive GPU resource management. Some work [3, 4, 5] took time-driven approach that controls timings and the duration of time allowed to access GPU resources, *i.e.*, scheduling and reservation. In these time-driven approaches, application tasks need not to be aware of what is happening in GPU resource management, because it is handled by the OS or runtime scheduler. However, they can not manage task execution times. They also limit the number of contexts that can access the GPU simultaneously to remove performance interference. Therefore, GPU resources could be wasted if a running context does not fully use compute cores.

We consider a different approach than previous work that enables GPU applications to adjust their task execution times. The number of cores used in the program is a major factor that affects the execution time. Hence, we explore how to adjust the core allocation at runtime. It is important to note that the programmer is typically responsible for allocating the number of cores (or threads mapped to cores) in GPU programming. In order to adjust the number of cores at runtime, it is essential to provide the programmer with an

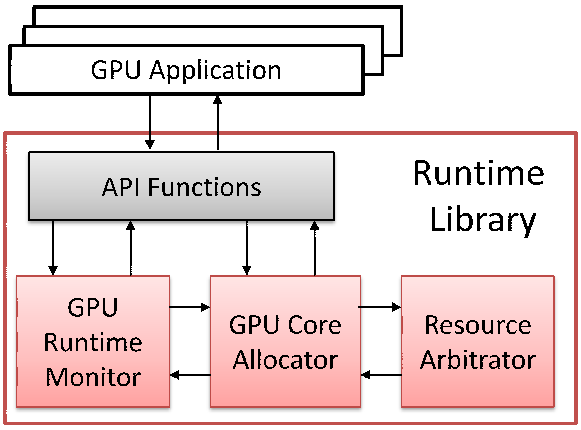


Figure 1. The proposed conceptual architecture for GPU resource management.

interface to obtain the information on the number of cores available or allocated for the program at runtime. The programmer is then responsible for making the program adaptive to the number of cores.

In the following, we present two frameworks that could be used to implement the proposed approach. We plan to implement a real system as a proof-of-concept, leveraging open-source software [2]. The proposed architecture is also illustrated in Figure 1.

3.1 Explicit Adjustment

In our explicit adjustment framework, the programmer is responsible for adjusting the number of cores to relax or tighten the computing demand. There will be no adjustment unless the programmer explicitly takes an action. A typical usage of this framework with periodic real-time tasks is as follows.

At the end of each period, the programmer calls a function provided by our framework that returns the latest task execution time. The programmer next calls either of the following two API functions. One increases the number of cores to be used by the next GPU execution to speed up the program. The other decreases it to slow down the program. This framework is usable in practice because the programmer often knows the desired task execution time to meet the frame-rate or deadline. It is also flexible in that the programmer can determine when to increase or decrease the number of cores.

A downside of this framework is that a task may misbehave and interfere with other contending application tasks, if the programmer fails to call the API functions correctly. We can cap the maximum number of cores available for an individual task to prevent it from abusing GPU resources, but the adaptivity of computing depends on the programmer, and outside system control.

3.2 Implicit Adjustment

Our second approach to adaptive resource management is an implicit adjustment framework. In this framework, the number of cores to be allocated for the program is set by the runtime system. Hence, the adaptivity of computing does not really depend on the programmer. If the program is not aware of this framework, however, it may fail to run, since the number of core allocated for the program may be different from what the program assumes.

The programmer specifies the desired task execution time as a set point before the task starts. If this set point is not specified, the runtime system tries to derive it internally as time goes by. When the task uses the GPU, the runtime system consistently updates the number of cores available for the corresponding task in the next period based on the previous execution time records. It is still the programmer's duty to check the number of available cores before offloading the computation onto the GPU.

This implicit adjustment framework is more preferable to the explicit adjustment framework, as it can enforce adaptive GPU resource management. However, it requires consensus in the programming model that the number of cores allocated for the program could be changed every time it is offloaded onto the GPU, and the programmer must be aware of it to make the program work. We claim that this is a natural trade-off between the generality of programming and needed adaptivity of service.

3.3 Runtime System Support

The runtime system provides the API for real-time GPU programmers. In order to support adaptive GPU resource management, we must provide some additional API functions.

- Our adaptive GPU resource management frameworks require a function to measure the execution time of each job running on the GPU. This function is easy to implement. Since we assume that job execution on the GPU is non-preemptive, the amount of time in run-to-completion can be accounted as job execution time. This accounting method is also known to work from previous studies [5, 11]. For the explicit adjustment framework, this function must be exposed to the programmer, while it is used internally by the implicit adjustment framework.

- We also need several functions to change the number of cores to be allocated for the program. Some existing programming languages for GPGPU, *e.g.*, CUDA [7],

provide the API to allow the programmer to specify the shape of the grid structure and the number of threads mapped to compute cores. We can use this API as it is, or provide a corresponding API if the underlying programming language does not support it.

In addition to these API functions, the runtime system must be able to detect when the program is offloaded onto the GPU and when it is completed on the GPU. Since the programmer calls a specific API function to launch the GPU program in most GPU programming models, it is very easy to record the start time of GPU execution. The detection of the completion time of GPU execution, on the other hand, is not straightforward. We would need to use an interrupt to notify the runtime system of the completion of GPU execution. Polling on a particular register is an alternative, but it would not be suitable for latency-sensitive real-time systems, as previous work demonstrated [5].

Finally, runtime system support must be integrated with the API so that the programmer can make use of our frameworks under a single unified programming model. We plan to extend our CUDA runtime library developed in previous work [2] to support our adaptive frameworks. While this is our planned prototype implementation, and our frameworks can also be integrated with other programming models beyond CUDA.

4 Summary

In this paper, we have discussed adaptivity requirements in embedded real-time systems with GPUs, and presented two frameworks for adaptive GPU resource management. We conjecture that the generality of programming may need to be compromised to achieve adaptivity of resource allocation on the GPU. Nonetheless, adaptive resource management is a key solution in optimizing performance under resource-constrained environments. We believe that our frameworks for GPU programming are useful contributions in this line of work, given the increasing emphasis in highly parallel heterogeneous computing.

References

[1] J. Ferreira, J. Lobo, and J. Dias. Bayesian real-time perception algorithms on gpu. *Journal of Real-Time Image Processing*, 6:171–186, 2011. 10.1007/s11554-010-0156-7.

[2] S. Kato, S. Brandt, Y. Ishikawa, and R. Rajkumar. Operating Systems Challenges for GPU Resource Management. In *Proceedings of the International Workshop on Operating Systems Platforms for Embedded Real-Time Applications*, pages 23–32, 2011.

[3] S. Kato, K. Lakshmanan, Y. Ishikawa, and R. Rajkumar. Resource Sharing in GPU-accelerated Windowing Systems. In *Proceedings of the IEEE Real-Time and Embedded Technology and Aplications Symposium*, pages 191–200, 2011.

[4] S. Kato, K. Lakshmanan, A. Kumar, M. Kelkar, Y. Ishikawa, and R. Rajkumar. RGEM: A Responsive GPGPU Execution Model for Runtime Engines. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 57–66, 2011.

[5] S. Kato, K. Lakshmanan, R. Rajkumar, and Y. Ishikawa. TimeGraph: GPU Scheduling for Real-Time Multi-Tasking Environments. In *Proceedings of the USENIX Annual Technical Conference*, 2011.

[6] M. McNaughton, C. Urmson, J.M. Dolan, and Jin-Woo Lee. Motion planning for autonomous driving with a conformal spatiotemporal lattice. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 4889 –4895, may 2011.

[7] NVIDIA. CUDA C Programming Guide. `http://developer.nvidia.com/nvidia-gpu-computing-documentation`.

[8] NVIDIA. NVIDIA's Next Generation CUDA Compute Architecture: Fermi (Whitepaper). `http://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf`.

[9] NVIDIA. Tegra 2 and Tegra 3 super chip processors. `http://www.nvidia.com/object/tegra-superchip.html` seen on March 7th, 2012.

[10] NVIDIA Press. NVIDIA GPUs to Be the Infotainment Centerpiece Across BMW's Next-Generation of Cars. `http://pressroom.nvidia.com/easyir/customrel.do?easyirid=A0D622CE9F579F09&version=live&prid=704317&releasejsp=release_157&xhtml=true` seen on March 7th, 2012.

[11] C. Rossbach, J. Currey, M. Silberstein, B. Ray, and E. Witchel. PTask: Operating system abstractions to manage GPUs as compute devices. In *Proc. of the ACM Symposium on Operating Systems Principles*, 2011.

[12] Top500 Supercomputing Sites. `http://www.top500.org/`.

[13] C. Urmson, J. Anhalt, H. Bae, D. Bagnell, C. Baker, R. Bittner, T. Brown, M. Clark, M. Darms, D. Demitrish, J. Dolan, D. Duggins, D. Ferguson, T. Galatali, C. Geyer, M. Gittleman, S. Harbaugh, M. Hebert, T. Howard, S. Kolski, M. Likhachev, B. Litkouhi, A. Kelly, M. McNaughton, N. Miller, J. Nickolaou, K. Peterson, B. Pilnick, R. Rajkumar, P. Rybski, V. Sadekar, B. Salesky, Y-W. Seo, S. Singh, J. Snider, J. Struble, A. Stentz, M. Taylor, W. Whittaker, Z. Wolkowicki, W. Zhang, and J. Ziglar. Autonomous Driving in Urban Environments: Boss and the Urban Challenge. *Journal of Field Robotics*, 25(8):425–466, 2008.

Towards Adaptive Resource Management for Virtualized Real-Time Systems

Stefan Groesbrink
Design of Distributed Embedded Systems
University of Paderborn
stefan.groesbrink@hni.upb.de

Simon Oberthür
Design of Distributed Embedded Systems
University of Paderborn
oberthuer@upb.de

Daniel Baldin
Design of Distributed Embedded Systems
University of Paderborn
daniel.baldin@hni.upb.de

***Abstract*—System virtualization is a powerful approach for the creation of integrated systems which meet the high functionality and reliability requirements of complex real-time systems. It is in particular well-suited for mixed-criticality systems, since the often applied pessimistic manner of critical system engineering leads to heavily under-utilized resources. Existing static resource management approaches for virtualized systems are inappropriate for the dynamically varying resource requirements of upcoming adaptive systems. In this paper, we propose a dynamic resource management architecture for system virtualization which incorporates criticality levels and allows the addition of subsystems at runtime. The two-level approach offers flexibility beyond virtual machine borders, that is exploited to improve the resource utilization by an adaptive distribution of the resources over the entire system.**

I. Introduction

For the next generation of advanced embedded and cyber-physical systems, there is a trend towards adaptability and inclusion of self-optimization [1]. Systems that adjust their goals and behavior at runtime according to changes of the environment or defects are characterized by varying resource requirements and demand flexible and dynamic resource management architectures. Integrated environments can often provide a more resource-efficient implementation compared to multiple separated hardware systems. System virtualization is a promising approach for the integration of multiple systems with maintained separation, and is therefore gaining significant interest in the embedded real-time world [2]. The hypervisor allows the sharing of the underlying hardware among multiple operating systems (OS) in isolated virtual machines (VMs).

Virtualization is in particular well-suited for mixed-criticality systems, which consolidate subsystems of differing criticality (e.g. safety-critical subsystems, mission-critical subsystems, and subsystems of minor importance [3][4]). First, virtualization facilitates multi-OS platforms: adequate operating systems can be provided for the very differing demands of the subsystems, for example a highly efficient real-time operating system for safety-critical tasks and a rich general purpose operating system for tasks with a human machine interface. Multiple existing software stacks including operating system can be reused to create a system of systems, in contrast to approaches that add real-time support for applications to a general purpose operating system, such as RTAI [5]. Second, the resource utilization can be increased significantly by the addition of subsystems of low criticality to critical subsystems. The dimensioning of the resources of critical systems has to expect the worst-case demand at all times, which usually results in a poor utilization.

Virtualization solutions for the server and desktop market apply highly dynamic approaches, are however not real-time capable. We use a multi-mode approach. A survey of real-time mode change protocols can be found in [6], virtualized systems are however not covered. In order to guarantee real-time requirements, existing virtualization solutions for embedded systems typically assign the resources statically to the VMs [2]. This is hardly compatible with the dynamics of adaptive systems. In this paper, we present a more flexible resource management protocol with the potential of a considerable resource utilization improvement. Furthermore, it enables open systems, in which the addition of applications or even subsystems at runtime is possible. Openness is a matter of considerable importance for many adaptive real-time systems.

II. Dynamic Resource Management beyond Virtual Machine Borders

A. The Flexible Resource Manager

In previous work, we have developed the *Flexible Resource Manager (FRM)* for self-optimizing real-time systems [7]. It is an OS extension to improve the resource utilization of systems with *EDF Scheduling* [8]. Each application is equipped with a set of profiles and transitions between them. Profiles represent implementation alternatives, representing for example different optimization levels for self-optimizing applications. Each profile contains information about minimum and maximum resource requirements. The FRM is in charge of switching between these profiles at runtime. Profiles are specified by the developer of an application.

The FRM improves the resource utilization by allowing an over-utilization and assigning temporary unused resources at other applications' disposal. Over-utilization is allowed if a plan for solving every possible conflict exists and if these plans are schedulable under hard real-time constrains. The standard approach to achieve safe and predictable behavior for multiple applications on a real-time operating system is to allocate the maximal required resources upfront. When an application does not use the complete amount of resources, the FRM

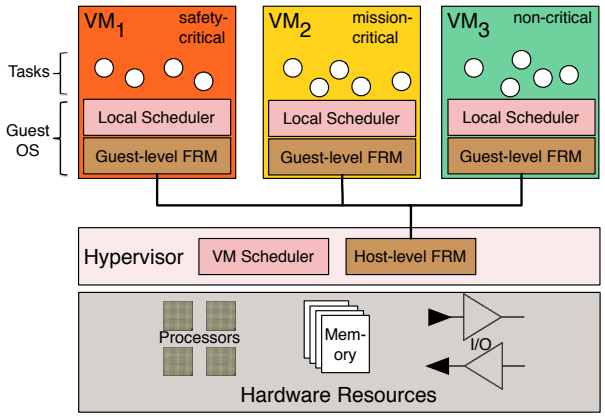


Fig. 1. **Hierarchical FRM: Example**

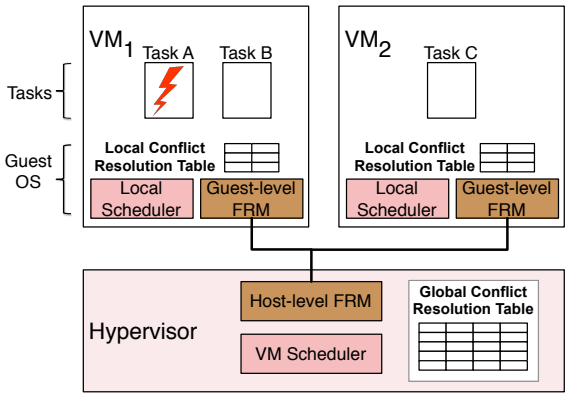


Fig. 2. **Conflict Resolution**

assigns fractions of these resources to other applications. This is realized by profile switches. The FRM selects a profile with a lower resource consumption for the application that currently does not need the complete amount, and activates profiles of other applications with higher resource consumptions. If the resource lending application needs more resources, the resource conflicts are solved according to the accepted plan. Schedulability and real-time capability, even during profile transitions, have been formally proven in [7].

B. Adaption to System Virtualization

Until now, the FRM addressed non-virtualized architectures with a single operating system. In this work, we adapt the concept to system virtualization. Figure 1 depicts an example with three virtual machines of different criticality. The assignment of criticality on VM level is no restriction, since our use case deals with the consolidation of entire systems. The addition of more than one guest of the same criticality is of course possible. Our solution implies a resource management protocol with FRM instances on two levels. A host-level FRM is part of the hypervisor and retains the ultimate control of the hardware resources. The guest operating systems on the second level use their local scheduler and a guest-level FRM to schedule their applications and assign resources to them.

A partitioned approach with decisions on both levels and communication in both directions follows. A guest-level FRM is informed by the host-level FRM about the resources assigned and selects appropriate application profiles to utilize them efficiently, as it does in the non-virtualization case. In particular, it can assign temporary unused resources to other tasks. When the subsystem is over-allocated, a situation in which the addition of the maximum limits of a resource of the active profiles exceeds the assigned quantity, a *resource conflict* may occur. In this case, the resources are given back to the original task by switching to other profiles. This is done according to plans which are stored in a local conflict resolution table (fig. 2). An entry contains a sequence of profile switches that solves the conflict. This table is filled automatically at runtime with the result of the online schedulability analysis. A profile switch is only accepted, if the schedulability analysis found a schedulable conflict resolution.

The guest-level FRMs inform the host-level FRM about the dynamic resource requirements and current resource utilization. For this reason, a second profile class is introduced: next to *application profiles* that specify the resource requirements for applications, there are *subsystem profiles* that specify the minimal and maximal resource limits for subsystems. Subsystem profiles unite the active profiles of the subsystem's applications to a single profile on subsystem level by adding the resources' minimum and maximum limits. In case of an application profile transition, the subsystem profile is updated and communicated to the host-level FRM. The host-level FRM's resource assignment among the virtual machines is based on the subsystem profiles.

Just like the guest-level FRM shifts resources between applications, the host-level FRM can take resources away from subsystems and allow other guests to use them. The required reconfiguration plan for conflict resolution in real-time is stored in the global conflict resolution table. Conflict resolution in this two-level approach is explained using the example of fig. 2. It is assumed that task A, executed in virtual machine VM_1, has a specific worst-case requirement of a resource, and consequently this resource share was assigned. Since the actual resource usage of task A was significantly below the reserved amount, the guest-level FRM switched to another profile and made a fraction of the assigned resource available to task B. In case of a resource conflict, task A requires a larger resource share than left, the guest-level FRM resolves the conflict by switching to profiles with a resource distribution that fulfills task A's requirements.

If the resource conflict was caused by a resource shift to another guest system, the guest-level FRM cannot resolve it. A share of the resource reserved for virtual machine VM_1 could have been assigned to virtual machine VM_2 by the host-level FRM, and further assigned by VM_2's guest-level FRM to task C. The host-level FRM informed VM_1's guest-level FRM about this resource shift and the guest-level FRM noted this in the local conflict resolution table. In case of a resource conflict of task A, the guest-level FRM of VM_1 informs the host-level FRM, which prompts the guest-level FRM of VM_2 to release the supplemental resources. The guest-FRM of VM_2 switches the profile of task C to one of lower resource utilization,

the host-level FRM can accordingly switch the profiles of VM_2 and VM_1 and inform the guest-level FRM of VM_1 to ultimately effect the conflict resolving profile switch for task A. Such a conflict resolving with involvement of the host-level FRM and consequences for more than one subsystem is called *global reconfiguration*, as opposed to a *local reconfiguration* which is accomplished by a single guest-level FRM.

The FRMs are in charge of determining the resource assignments at runtime. The decision-making task is executed as the idle task. If not enough idle time is left for it, the resource distribution is not improved. Whether a reconfiguration has positive effects or not depends on the ratio between overhead of the reconfiguration process to benefit, which again is dependent on the duration after which the reconfiguration has to be undone. The duration depends on various stochastic events which are caused by both the applications itself and the environment. The FRM permanently gathers information about the resource utilization and computes with the help of *Dynamic Bayesian Networks* a likeliness of a change of the resource requirements. The probabilistic analysis produces a mean time of resource changes, which is used as an approximation of the future [7]. The criticality is of great importance. Safety-critical subsystems do not benefit from additional resources, since the worst-case demand is assigned. Therefore, resources are not shifted to safety-critical VMs, but shifted away from them to less-critical VMs.

C. Scheduling

The most important resource is the central processing unit. Our approach targets homogeneous multiprocessor systems. System virtualization requires scheduling decisions on two levels (*hierarchical scheduling* [9]): both the virtual machines and the tasks within the VMs have to be scheduled. There are two main approaches in the multiprocessor scheduling domain. As opposed to *partitioned scheduling*, migration of tasks among processors is possible under *global scheduling* [10].

Partitioned scheduling is for multiple reasons the right approach for system virtualization. It introduces in general a lower overhead, since the task migration of global scheduling results in overhead for synchronization and lost performance due to cache misses [10]. Furthermore, the consolidation by system virtualization runs entire software stacks consisting of guest operating system and tasks within a virtual machine. It is therefore a coarse-grained approach to reuse systems with verified (or even certified) characteristics that should not be split up. This is especially true for mixed-criticality systems, since a mixing of criticality levels within a subsystem should be avoided. Task migration is therefore neither desirable nor in general technically possible across operating system borders.

Our approach allocates each safety-critical guest and mission-critical guest to a dedicated processor. A processor is not shared between multiple critical guests, but an addition of non-critical guests is possible. If critical and non-critical guests share a processor, the non-critical guests are scheduled in background: whenever the critical guest is not running, the idle processor is used to execute the non-critical guest. Consequently, the execution of a system with n safety-critical or mission-critical guests requires a hardware platform with at least n processors. Contrary to task migration, the migration of an entire virtual machine from one processor to another is less problematic and in some situations of significant help, as explicated in the following subsection *Open Systems*. To keep the introduced mapping of critical guests to processors, only non-critical guests are possibly migrated.

D. Open Systems

Open systems require dynamic scheduling algorithms and an online acceptance test which validates the schedulability whenever a new task enters the system [11]. The acceptance test checks whether it is possible to add the arriving task to the set of previously guaranteed tasks or not. Our approach enables the addition of both entire guest systems and tasks at runtime. Based on the fact that the FRM requires the scheduling algorithm EDF, under which the processor may be utilized up to 100% [8], a request to add a new real-time task τ_k with a worst-case execution time of C_k and a period of T_k to the existing task set of the subsystem of n tasks can be accepted if and only if

$$\frac{C_k}{T_k} + \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1. \tag{1}$$

The much more complex acceptance test for the addition of an entire new guest system is depicted in fig. 3 (guest system G_k, processors P_1 to P_m, utilization U, criticality C, number of critical guests n_C). A criticality of one is set if the system is non-critical and a higher value denotes a critical system. More than three criticality levels are possible, even though our example uses three levels.

(1) If there is a processor that is unused up to now, the new guest system is assigned to this processor and the request is accepted.

(2) If a non-critical guest system shall be added, it is assigned to the processor with the lowest utilization. The addition of a non-critical guest system does not endanger the real-time tasks, since it is scheduled in background.

(3) If there is not a critical guest assigned to each processor, the arriving guest system is assigned to the processor with the lowest utilization among the processors without critical guest. The schedulability of the new guest system is guaranteed, since it shares the processor only with non-critical guests and obtains therefore highest priority. It is possible that the addition of the new guest implies an unbalanced distribution of guest systems to processors. Therefore, non-critical guests are possibly migrated.

(4) If a critical guest is assigned to each processor, but there is at least one guest of lower criticality level, the arriving guest system replaces the critical guest of lowest criticality. This could again lead to an unbalanced distribution among the processors, for which reason non-critical guests are migrated, if this improves the distribution.

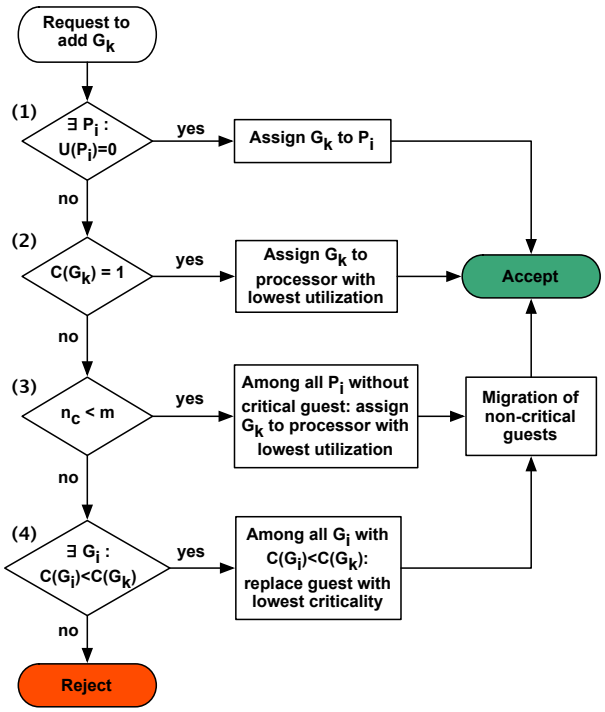


Fig. 3. **Acceptance Test for Guest System (guest system G_k, processors P_1 to P_m, utilization U, criticality C, number of critical guests n_C)**

If none of the checked conditions was valid, the arriving guest system has to be rejected. Concerning step (4), it is debatable whether a higher criticality level legitimates automatically the replacement of an entire guest system. It depends on the application domain and may be unjustified and in this case has to be deactivated to favor system continuity. But there are definitely situations in which this replacement makes sense, for example when a mission-critical subsystem is replaced by a safety-critical subsystem to protect device or environment at the expense of an unsuccessful mission.

The host-level FRM performs the schedulability test for subsystems. The FRM approach requires that both arriving tasks and arriving subsystems provide profiles, if they have to be scheduled under real-time constraints. Non-real-time applications and subsystems can be accepted without profiles.

III. Conclusion

The presented resource management architecture focuses on three basic concepts. *System virtualization* and its ability to reuse subsystems is a powerful technique to meet the functionality and reliability requirements of increasingly complex systems and has potential to support the migration to multiprocessor platforms. *Mixed-criticality systems* and virtualization's integration of both multi-source software and subsystems with very differing characteristics are a good fit, and resource sharing is particularly promising for this combination. *Open systems* increase the flexibility enormously and are of paramount importance for upcoming adaptive embedded and cyber-physical systems.

By combining these three concepts, our approach provides a flexible resource management architecture for the dynamically varying resource requirements of integrated adaptive systems. The two-level solution beyond virtual machine borders has the potential to increase the resource utilization significantly compared to static approaches. Multiple criticality levels are handled appropriately to achieve two goals: the deterministic behavior of critical missions is guaranteed and the service quality of non-critical missions is possibly increased by shifting resources to them. The resource assignment can adapt to chances in application behavior and environment and the addition of tasks and entire subsystems at runtime is possible.

The approach is currently under development. We integrate it into our real-time hypervisor *Proteus* [12]. The implementation of our approach requires paravirtualization, since the guest-level FRMs have to pass information to the hypervisor. According to paravirtualization [13], modified guest operating systems that are able to communicate with the hypervisor are hosted. The requirement to modify the guest operating system is outweighed by the advantages gained in terms of flexibility of an explicit communication and cooperation of host and guest.

Acknowledgment

This work was funded within the project *ARAMiS* by the German Federal Ministry for Education and Research with the funding ID 01IS11035 and within the project *Collaborative Research Center 614 - Self-Optimizing Concepts and Structures in Mechanical Engineering* by the Deutsche Forschungsgemeinschaft. The responsibility for the content remains with the authors.

References

[1] G. Vouros, A. Artikis, K. Stathis, and J. Pitt, "Organized adaption in multi-agent systems," in *OAMAS*, 2008.

[2] G. Heiser, "The role of virtualization in embedded systems," in *1st Workshop on Isolation and Integration in Embedded Systems*, 2008.

[3] D. de Niz, K. Lakshmanan, and R. Rajkumar, "On the scheduling of mixed-criticality real-time task sets," in *30th IEEE Real-Time Systems Symposium*, 2009.

[4] S. Baruah, H. Li, and L. Stougie, "Towards the design of certifiable mixed-criticality systems," in *Proceedings of the IEEE Real-Time Technology and Applications Symposium (RTAS)*, 2010.

[5] P. Mantegazza, E. L. Dozio, and S. Papacharalambous, "Rtai: Real time application interface," in *Linux Journal*, 2000.

[6] J. Real and A. Crespo, "Mode change protocols for real-time systems: A survey and a new proposal," in *Real-Time Systems*, vol. 26(2), 2004.

[7] H. S. Lichte and S. Oberthür, "Schedulability Criteria and Analysis for Dynamic and Flexible Resource Management," *Electron. Notes Theor. Comput. Sci.*, vol. 200, no. 2, pp. 3–19, 2008.

[8] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," in *Journal of the ACM*, 1973.

[9] G. Lipari and E. Bini, "A methodology for designing hierarchical scheduling systems," in *Journal of Embedded Computing*, vol. 1(2), 2005.

[10] R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," in *ACM Computing Surveys*, 2010.

[11] G. Buttazzo, *Predictable Scheduling Algorithms and Applications*. Springer, 2011.

[12] D. Baldin and T. Kerstan, "Proteus, a hybrid virtualization platform for embedded systems," in *3rd IFIP TC 10 International Embedded Systems Symposium*, 2009.

[13] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, 2003.

Control kernel based adaptive control implementation

Raúl Simarro, Pedro Albertos
Department of Systems Eng. and Control
I.U. de Automática e Informática Industrial
Universitat Politècnica de València
Valencia, 46071, Spain
Email: {rausifer,pedro}@isa.upv.es

José Simó
Department of Computer Engineering
I.U. de Automática e Informática Industrial
Universitat Politècnica de València
Valencia, 46071, Spain
Email: jsimo@disca.upv.es

***Abstract*—A control system with distributed computing resources should always guarantee the safe control of the plant. In this contribution, the concept of control kernel is used for that purpose. Two types of nodes with different resources are defined: the powerful server node and the resource-constrained light node. This architecture allows to split the control tasks into two blocks. Those demanding strong computing resources are allocated in the server nodes and those compelling tasks required to ensure the safety of the controlled plant are allocated in the light nodes. Resource limitations lead to control adaptation. Two simple applications illustrate some of the benefits of this architecture with one server node and one light node, even the architecture can be extended to several nodes. In the first case, an adaptive control is implemented in the server node, providing the control algorithm to the light node, which is also able to compute a local safe control action. In the second experiment, two different control tasks requiring different resources are implemented in a mobile robot control. To keep bounded the computing time at the local level, the supervisor decides the time allocated to each activity, providing the resulting controller to the light node.**

I. Introduction

Adaptive control requires, in a direct or indirect way, to carry out a parameter estimation task to compute and update the controller parameters [1]. In many cases, adaptation also implies changes in the controller structure as well as past data retrieval. All these tasks may require a lot of computation time, not being suitable to be implemented in an environment with limitation of resources.

Moreover, safety is a crucial issue in embedded control systems [2], [3]. Independently of the number of variables to be controlled by the same processor, the systems with hard real time requirements must ensure the delivering of control actions to all actuators. The quality of the delivered signal may depend on the processing level: data, computational algorithms and resources availability, among others, but always must ensure the safe system operation [4]. Besides components malfunction, in complex control systems, safety can be affected by either the appearance of high priority aperiodic tasks, the variation of the controlled system dynamics requiring switching controllers, the missing of execution deadlines and messages or the variation of communications delays. In this context, in order to run control applications in a safe mode, if the control action has not been delivered on time by the current controller, a back-up control action should be delivered at the time required by the process. This signal may be the result of a simple calculation (but sufficiently safe), an emergency shutdown or simply a safe response such as: *keep unchanged*. Note that this operation can be also interpreted as a controller switching.

There are many different approaches to design and implement embedded control systems, (see, for instance, [5], [6]). In this work, the concept of *control kernel* [4] is used to compute the control in two stages. This control architecture has been implemented on multiple fully automated mobile vehicles performing activities requiring coordination between them to avoid obstacles, path tracking, scanning, data collection, etc.. To do this one of the robots acting as the supervisor calculates the trajectories to be followed by others, and adapts to control environmental conditions, so the others can use their computation time for data collection, processing products, or other tasks, delegating the calculation of the trajectory to the supervisor. In case of communication problems or excessive computation time they can apply a safe control action.

The control kernel approach presents some novel properties based on the isolation provided by the middleware implementation. Control tasks are moved to nodes where its execution is more efficient, based on the observed availability of resources. As a consequence, the architecture provides a transparent framework to combine different controllers to be applied as decided.

The control kernel architecture is summarized in the next section. Then, the adaptive control algorithm is split into two parts to be implemented in different nodes. The proposed approach is tested on two simple experimental systems to evaluate its possibilities. Some results are reported. Finally, discussion is motivated based on these preliminary results.

II. Control Kernel Architecture

Two node types can be defined, [7]: Light nodes and Service nodes (see Figure 1). Service nodes are powerful embedded computers running a full featured RTOS with complete networking with I/O capabilities. Light nodes are small and low power consumption SoC processors with limited computing and networking capabilities but complete I/O features.

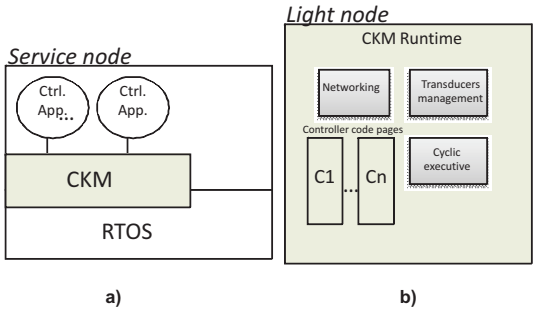


Fig. 1. Control nodes.

A control kernel middleware is implemented in both nodes, [7]. In a distributed embedded control architecture, many of these nodes can be interconnected in a wired or wireless network, see Figure 2.

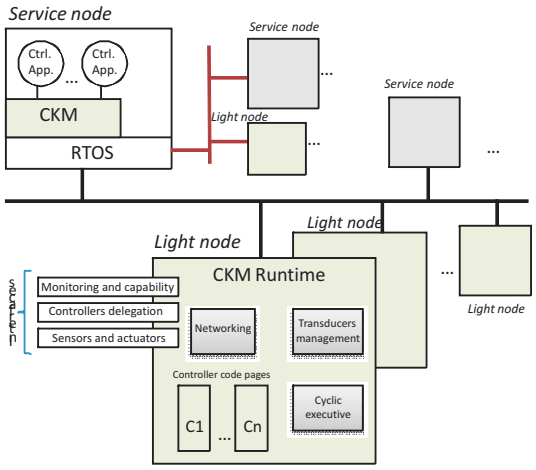


Fig. 2. Control Kernel based architecture.

Control Applications run in service nodes on top of a full featured Control Kernel Middleware (CKM). This middleware offers abstractions and functionalities related to control tasks real time execution, access to sensors and actuators, and communications management. The programming model of CKM follows the concept of code delegation. In this sense, a control application delegates the execution of some control code to the CKM that provides computational resources to execute it. Note that a control task, once inside the CKM, can run on whatever service node of the DCS having access to the communications space of the task.

Light nodes are a cost-effective solution in order to allocate some computer power as close as possible to each actuator. This is mandatory in order to reduce the nondeterminism in the time delivering of the control action to the plant. Light nodes run a retail of the CKM: the CKM Runtime. This Runtime communicates with the CKM offering interfaces for management, sensing and acting as well as code uploading. A light node can be used as simple slave component to interface DCS or it can run locally its own controllers in a cyclic executive environment. Ensuring the delivery of an appropriate control action guarantees the safety of the system, even if this action is to stop the plant operation.

In a Control Application, any control task that has been delegated to the CKM can be transferred to a light node by uploading the native code page and asking for switching. Controller pages can be uploaded through the CKM Runtime without any interference with the controllers currently running in the node. The uploaded pages are activated for running by the switching mechanism provided by the CKM Runtime. Attention should be paid to the system schedulability [8].

In particular, service nodes may include supervising and optimizing control activities and light nodes can run activities to drive the system to a safe position or run a simple algorithm that guarantees a minimum of performance in the system at any time. In this sense, Light node ensures that a control action ($u(k)$) to be sent to the process always exists. This signal may be just a safe action (disconnect, open, close, unchange, etc.) or the result of a simple calculus (computed locally in the node) ($u_l(k)$) or it may be the signal calculated ($u_s(k)$) and received from a service node.

III. Adaptive control Implementation

A typical structure of adaptive control involves two loops. The classical feedback loop is keeping the required performance of the controlled plant, whereas the extra loop is in charge of evaluating the quality of the control, performing a parameter estimation algorithm and determining the actual control to be applied to the plant. The actual control may be also rather complex. The adaptation will update the controller parameters and/or structure. By using the control kernel concept, a slightly different alternative is proposed in this paper. Two different control actions are computed (Figure 3). One is very simple and fast, also requiring very few resources, and it could be just a proportional action. It is a *reactive control*. For that, no data retrieval is necessary and the computing time is very short. Unfortunately, the controlled plant behavior will not be the best achievable with a more sophisticated controller, but it must ensure the controlled plant stability. This action will be provided by the light node in charge of the process.

The most desirable control action, as provided by the adaptive controller which is allocated in the server node, will be delivered if there are enough available resources (communication bandwidth, computing time, memory accessibility and so on). In the end, this control action will be based on a part related to the current measurement (proportional action), one part computed from the past measurements and errors (the integral action) and one (or more) parts due to the error prediction (derivative action). This is clearly illustrated in the case of using a PID adaptive control [9].

In this case the light node will compute the local control action based on a local gain K_l (1)

$$u_l(k) = K_l e(k) \tag{1}$$

where $e(k)$ is the current error. Depending on the local resources, this control law can be also based on a *frozen* PID controller, provided by the service node but being updated

from time to time, not continuously. On the other hand, the server node will evaluate (2)

$$u_s(k) = K_s e_k + K_1 e(k-1) + K_2 e(k-2) + u(k-1) \quad (2)$$

by applying the full control structure and adapting the controller parameters.

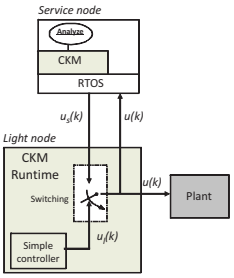


Fig. 3. Control switching.

Under normal operation, the actual control will be $u(k) = u_s(k)$. In the event of a disturbance (lack of communication, lost of measurement, or just not enough time to compute and adapt the controller parameters ($\{K_s, K_1, K_2\}$), the light node will provide the local control action to both the plant and the server, to be used in future computations, Figure 3.

IV. Experimental work

Due to space shortage, two simple applications are presented to illustrate the possibilities of the control kernel approach. As already mentioned, the same approach can be used for more complex applications where there is a general coordination and adaptation at the server level and a local reactive control. First, an adaptive control of a simulated process suffering high priority communication interferences is evaluated. In this case, the reactive local control is used if the adaptation takes too much time. Then, for a mobile robot with two different tasks (tracking a trajectory and avoiding obstacles) two different control algorithms are used. They are implemented in the server node, being combined in a weighted way according to the operating conditions. The time allocated to each one is adapted to keep constant the total computing time available for this control. At the local node, the final control action is computed and delivered.

A. Adaptive control under interferences

A simulated process is controlled to get fast response and adaptation capabilities. The adaptive control algorithm is run in the service node and any time a new controller is decided it is sent to the local controller. In the event of a high priority aperiodic task (acting as an interference to the control process computation), the local node keeps running the last updated controller or a simpler back-up one.

The process transfer function is $G(s) = \frac{66.777}{s^2+9.391s+77.16}$ and it is assumed invariant, the adaptation being motivated by the appearance of an interference. It has been simulated and evaluated in a Matlab/Simulink environment, using the Truetime tool to evaluate the behavior, as shown in Figure 4. A faster and less performing controller is implemented in the local node to be directly applied to the plant if there is no action coming from the server node. Initially, the plant is controlled through the control action sent by the "complex" controller implemented in the server node. At $t = 6s$, an interference forces the control to be transferred to the basic control implemented in the light node. The control is softer and the plant response is degraded, as can be seen in Figure 5, during the time the interference is active. In the first appearance (interval $\{6, 7\}$), the system is in the transient response and the behavior is degraded. During the second interval ($\{8 - 13\}$) a change in the reference happens and the response is again degraded but as soon as the interference disappears, the control is assumed by the service node and the response is stabilized and improved. It is worth to note that the basic controller does not include integral action, as can be seen in the steady-state error appearing in the plant response before $t = 13s$. Finally, the interference appears just at the time of a reference change, $t = 15s$, and the plant response is also degraded.

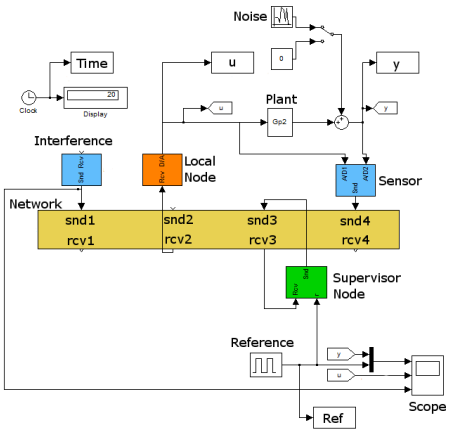


Fig. 4. Control kernel implementation and simulation.

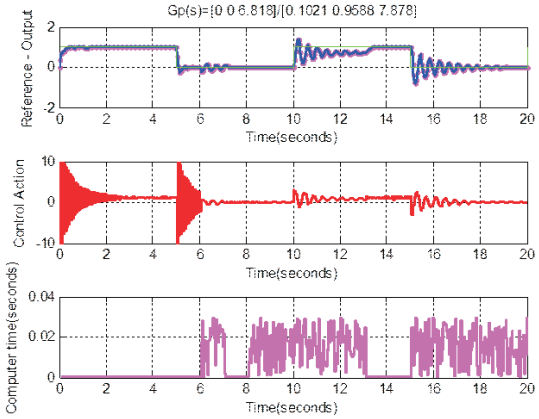


Fig. 5. Switching control due to interferences.

B. Controller adaptation due to goal changes

In the second experiment, a mobile robot is controlled to follow a trajectory avoiding unexpected obstacles [10]. Both control algorithms are implemented in the server node. The obstacle avoidance control algorithm involves heavier computing load, reducing the time used to compute the control action to follow the trajectory. The local controller sends to the server node the sensed information where the robot performance are evaluated allowing the analysis of the operating conditions and the selection of the most suitable controller by the *supervisor*. A decision is taken about how much weight should be allocated to each control algorithm. The combined controller task is transferred to the light node.

The performance of the control is illustrated in Figure 6, where there are two obstacles and a reference trajectory.

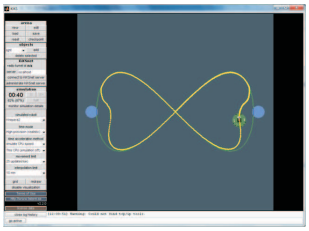


Fig. 6. Mobile robot: trajectory tracking and obstacles avoidance.

The goal is to keep bounded the computing time allocated for the robot guidance. This is shown in the upper graphic in Figure 7, where the computing time is almost constant. In the lower graphic, the percentage of time devoted to evaluate the obstacle avoidance control algorithm is shown. As it can be seen, in the time intervals characterized by the cycles $\{0-70\}, \{100-130\}, \{385-410\}$, no obstacles are detected and all the computing time is allocated to follow the trajectory. During the cycles in between, the obstacle starts to be detected and the server node determines the use of the light node to compute the control action to mainly avoid the obstacle, degrading the trajectory tracking performance, (Figure 6).

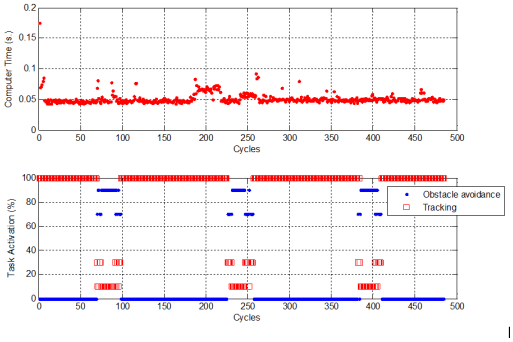


Fig. 7. Time devoted to obstacle avoidance, keeping bounded the total computing time.

V. Conclusion

The use of the Control Kernel structure offers many advantages in allocating different control tasks according to the operating conditions. Two types of nodes are defined: the powerful server node, probably taken care of many control loops and activities, and the resource limited light node, attached to a process, able to handle some related control loops. A number of these nodes can be connected in a network to implement a distributed control system.

In particular, for adaptive control, the more computing power demanding tasks are implemented in the server node where the actual controller structure and parameters are computed. This information is transferred to the local node, where there is also a back-up controller to ensure the delivering of a safety control action.

Two control scenarios have been considered. In the first case, based on a simulated plant, the appearance of aperiodic high priority tasks reduces the availability of computing time and provokes the switching between the adaptive control provided by the server and the basic control computed by the light node (Figure 3), leading to a degrading of the control performance (Figure 5). In the second experiment, where trajectory tracking and obstacle avoidance should be accomplished, the scenario evaluation and the selection of the control algorithm to be used are decided at the server node level, the light node implementing the controller decided by the supervisor.

Many other options are open with this control kernel structure and it is a matter of further research and experimentation.

Acknowledgment

This project has been partially granted by Consellería de Educación Generalitat Valenciana, under PROMETEO project number 2008-088, and the CICYT project COBAMI: DPI 2011-28507-C02-01/02.

References

[1] K. Astrom and B. Wittenmark, *Adaptive Control*, 2nd ed., Addison-Wesley Longman Publishing Co., Boston,MA,USA, 1994.

[2] Q. Li and C. Yao. *Real-Time Concepts for Embedded Systems*. CMP Books, 2003.

[3] P. Albertos, A. Crespo, M. Valls and I. Ripoll. "Embedded Control Systems: Some Issues and Solutions." *16th IFAC World Congress.* Prague, Czech Republic. Elsevier, 2005.

[4] P. Albertos, A. Crespo and J. Simó. "Control Kernel: A key concept in embedded control systems." *4th IFAC Symposium on Mechatronic Systems*. 2006.

[5] P. Martí, M. Velasco, J. M. Fuertes, A. Camacho and G. Buttazzo. "Design of an Embedded Control System Laboratory Experiment." *IEEE Transactions on Industrial Electronics* 57, n 10 (October 2010): 3297-3307.

[6] Z. Peng, M. Longhua and F. Xia. "A Low-Cost Embedded Controller for Complex Control Systems." IEEE/IFIP *International Conference on Embedded and Ubiquitous Computing*. Shanghai, 2008. 23-29.

[7] R. Simarro, J. Coronel, J.E. Simó and J.F. Blanes. "Hierarchical and Distributed Embedded Control Kernel." *XVIIth IFAC World Congress.* Seoul, Korea, 2008.

[8] A. Crespo, P. Albertos, P. Balbastre, M. Vallés, M. Lluesma and J.E. Simó. "Schedulability issues in complex embedded control systems." *IEEE Conference on Computer Arded Control Systems Design.* Munich, Germany, 2006.

[9] F. Radke and R. Isermann, "A parameter-adaptive PID-controller with stepwise parameter optimization". *Automatica* Vol 23, N 4, July 1987, Pages 449457.

[10] A. Valera, M. Valls, P. Albertos, R. Simarro, I. Benítez, y C. Llácer. "Embedded Implementation of Mobile Robots Control." *17th IFAC World Congress*. Seoul, Korea, 2008. 6821-6826.

SAFER: System-level Architecture for Failure Evasion in Real-time Applications

Junsung Kim, Ragunathan (Raj) Rajkumar
Carnegie Mellon University
5000 Forbes Ave., Pittsburgh, PA, USA
{junsungk, raj}@ece.cmu.edu

Markus Jochim
General Motors R&D
30500 Mound Rd., Warren, MI, USA
markus.jochim@gm.com

Abstract

We propose a layer called SAFER (System-level Architecture for Failure Evasion in Real-time applications) to incorporate configurable task-level fault-tolerance features such as Hot Standby and Cold Standby in order to tolerate fail-stop processor and task failures for distributed embedded real-time systems. To detect such failures, SAFER monitors the health status and state information of each task and broadcasts the information. When a failure is detected, SAFER reconfigures the system to recover failed processors and tasks. SAFER has been implemented on Ubuntu 10.04 LTS and deployed on Boss, an award-winning driverless vehicle developed at CMU. We provide preliminary measurements using one of the autonomous driving simulation scenarios used during the 2007 DARPA Urban Challenge.

1. Introduction

Advances in distributed embedded real-time systems have enabled a variety of different applications such as sensor networks, industrial control systems, avionic systems and automotive systems which are tightly coupled with the physical world. Such applications need to satisfy strict timing constraints based on operating characteristics, making timing guarantees an essential requirement. Furthermore, system reliability can be of high importance for some safety-critical applications that interact with the physical world. However, a trend towards increasing complexity in distributed embedded real-time systems poses challenges in designing a reliable system.

The conventional way of improving reliability has been adding redundant hardware. However, this approach becomes less attractive to many applications because the amount of necessary hardware multiplies as the size of the system increases. It is also not consistent with the growing needs of flexible system design. Therefore, we propose a layer called SAFER (System-level Architecture for Failure Evasion in Real-time applications) to incorporate configurable task-level fault-tolerance features such as Hot Standby and Cold Standby in order to tolerate *fail-stop* processor failures and task failures for distributed embedded real-time systems in a timely manner. To detect such failures, SAFER monitors the health status and state information of each task and broadcasts the information. When a failure is detected, SAFER reconfigures the system to recover failed processors and tasks using task-level replication techniques.

SAFER targets multiple goals. Most importantly, *no single point of failure* is permitted. In other words, a task/processor failure should not lead to system failure. *Failure recovery within a guaranteed duration* should also be achieved. Embedded systems are usually tightly connected to the physical world. In such a case, failure recovery without predictable timing behavior could return unpredictable results in the physical world.

Apart from the two goals above, *predictive fault discovery and notification, resource isolation, ease of use of abstraction, ease of application development, and sensor/actuator control* are other factors considered in the design of SAFER.

The rest of this paper is organized as follows. Section 2 describes the architecture of SAFER and its implementation. Section 3 presents preliminary results measured on Boss, an award-winning autonomous vehicle developed at CMU. Section 4 presents the related work, and we conclude in Section 5.

2. The SAFER Architecture

The SAFER layer is composed of SAFER daemons (one on each processor) and a library offering a task execution environment. The library enables any task launched on the SAFER layer to be periodically executed (with reconfigurable parameters). The daemons have a master-slave architecture [2], and the master SAFER daemon controls the slave SAFER daemons responsible for managing tasks on each node[1] and monitoring its health status. The reconfigurable parameters for each task are given to the task library when the task is launched by a SAFER daemon. For the underlying communication layer, an inter-process communication primitive such as IPC [11] and SimpleComms [12] can be used. The overall architecture of the SAFER layer is illustrated in Figure 1.

The SAFER layer utilizes two main features to avoid system failure in the presence of fail-stop processor or task failures. The SAFER layer supports task-level replication techniques such as Hot Standby and Cold Standby, where

[1] In this paper, node is interchangeably used with processor.

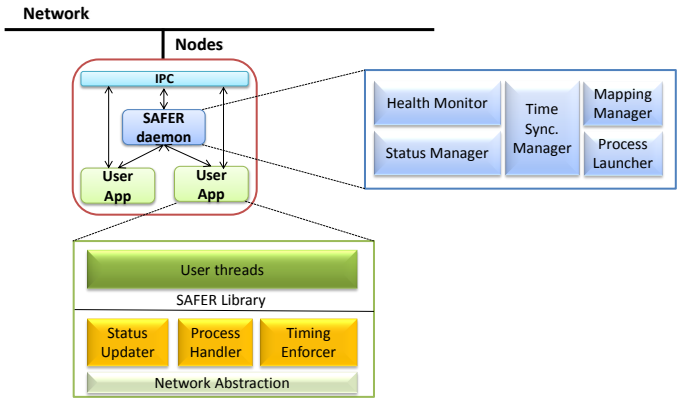


Figure 1. The overall architecture of SAFER

selective tasks on failed processors are recovered on other live processors. Replicas must therefore be placed on independent nodes, a constraint that is referred to as a *placement constraint* [6]. The major benefit of using selective task-level recovery is its flexibility. Since we can selectively recover tasks, we can increase the reliability of more critical tasks by adding more Hot Standbys/Cold Standbys for those critical tasks. We can also efficiently manage the available computing resources by not replicating less-critical tasks, thus enabling an affordable solution.

The second feature is the fail-over of the master SAFER daemon. Since the SAFER daemons have a master-slave relationship and manage tasks on each machine, the master SAFER daemon becomes a single point of failure. Hence, when the master SAFER daemon fails, one of the slave SAFER daemons will be promoted to become the master SAFER daemon. This can be done using a group membership protocol [4, 10]. Assuming a synchronous communication network[2] [4], the membership protocol of SAFER is different from the existing work in the sense that (i) we provide predictable timing behavior and (ii) the recovery duration of the SAFER daemons is deterministic. In our membership protocol, all SAFER daemons including the master and the slaves broadcast messages to each other. The master can detect the failures of a slave by the lack of heartbeat messages. The master SAFER daemon will command as necessary the slave SAFER daemons on live processors to recover any failed tasks. The death of the master can also be detected by the slaves due to the absence of heartbeats, and one of the slave SAFER daemon will be promoted to become the master by following a predetermined sequence of the slave SAFER daemons.

2.1. The SAFER Daemon

As illustrated in Figure 1, a SAFER daemon is composed of a health monitor, status manager, time synchronization manager, mapping manager and process launcher.

[2]The failure model of network is beyond the scope of this paper. We assume any packet eventually arrives at the destination.

The health monitor and status manager are responsible for monitoring the health status of the other processors and for changing the local node's role between the master and the slave. The time synchronization module offers a global time service. The mapping manager and process launcher can automatically deploy tasks on the nodes running a SAFER daemon.

2.1.1. Health Monitor and Status Manager

The health monitor periodically sends heartbeat signals to the other nodes in the system and monitors the health status of the other daemons and their processors. Therefore, the health monitor enables the SAFER daemons to agree upon the availability of each node. The period of heartbeat signals is configurable, and the list of current running tasks is added to the heartbeat signal and is broadcast to the other nodes. The status manager watches the current status of tasks running on its own node and notifies the failure of any task if there is a task failure (say due to a segmentation fault) by capturing the OS signal.

2.1.2. Time Synchronization Manager

The SAFER layer offers a global time service using a service similar to NTP [8] used for time synchronization over the Internet. The master SAFER daemon behaves as a time server, and each slave becomes a client for this service and listens to messages from the time server. This service is essential to synchronize all the daemons so that the failure recovery occurs within the given timing requirement using a proper offset between the primary task and its Hot/Cold Standbys. This also enables the timing enforcer of the SAFER library to have less penalty in resource scheduling.

2.1.3. Process Mapping Manager and Launcher

The process mapping manager and launcher are responsible for automatically deploying tasks on the nodes of the SAFER layer based on a given system configuration file. The system configuration file includes information about where tasks are allocated and how many resources tasks demand. It also contains the location of the primary and Hot/Cold Standbys if the tasks are selected to have backups. The process mapping manager maintains the information from the system configuration file and updates whenever the information changes. Changes to this information can occur due to processor failures, demand changes, task completions, and so forth. Based on the up-to-date information from the process mapping manager, the process launcher loads tasks on the different processors. The process mapping manager and launcher can be connected to a user-interface application such that the application provides a global view of the system with the current health status of each task on each node. As an example, the information from the process mapping manager and launcher are visualized on TROCS [7], the operator interface of Boss [13].

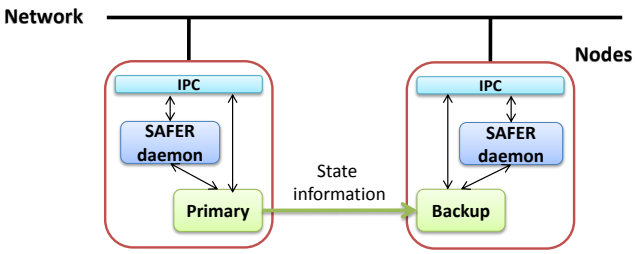


Figure 2. The primary-backup architecture

2.2. The SAFER Library

The SAFER library is a task execution environment composed of a status updater, process handler, timing enforcer and network abstraction. The user threads developed by application developers will run on the SAFER library.

2.2.1. Status Updater

The status updater of the SAFER library supports task-level replication techniques by managing state information between the primary and its Hot/Cold Standbys. The role of status updater changes based on whether a task it monitors is a primary or a backup. The backups subscribe to the primary, and the primary publishes its status. They use the Publish/Subscribe model. The status updater of the primary task periodically transfers its internal state information to its Hot/Cold Standbys, where the update period is configurable. The status updater at Cold Standby updates the state information coming from the primary in case the primary fails. The status updater can also be used as heartbeat signals by Hot/Cold Standbys. This could be useful when the period of the state updater is shorter than the update period of the SAFER daemon. This architecture is also depicted in Figure 2.

2.2.2. Process Handler

The process handler of the SAFER library promotes a backup to be the primary when it receives the corresponding request from the master SAFER daemon. When a backup is promoted, the new primary starts generating outputs for use and confirms its promotion to the master SAFER daemon. It must be noted that a Hot Standby is always running and its outputs are filtered by the network abstraction under the control of the process handler.

2.2.3. Timing Enforcer

The timing enforcer of the SAFER library enables tasks to have guaranteed and protected access to required processing resources in a timely manner based on Linux/RK [9]. In Linux/RK, a shared resource is reserved and enforced by the following parameters: computation time C every T time-units within a deadline D. For the SAFER library, this CPU reservation model in Linux/RK is utilized.

2.3. Failure Detection and Recovery

Heartbeat signals from the health monitor of each SAFER daemon will be used for detecting processor failures. Since we have assumed a synchronous network, we have defined a time delay d to represent the maximum network delay of the heartbeat signal packets. The master SAFER daemon will decide the death of a processor unless it hears a heartbeat signal from a processor within $d + T_{heartbeat}$[3], where $T_{heartbeat}$ is the interval between two consecutive heartbeat signals. We call this failure detection scheme as *time-based detection*. A task failure may be directly detected by the status manager of the SAFER daemon by catching a signal generated by the OS when a task has unexpectedly failed. Then, the mapping manager of the master SAFER daemon will be notified by the status manager, and an appropriate recovery will be initiated. We name this failure detection scheme as *event-driven detection*. It should be noted that event-driven detection cannot be used for processor failure detection.

The recovery from a failure is done by using task-level replication techniques such as Hot Standby and Cold Standby. A Hot Standby of a task is a replicated task running concurrently with its primary. With no failure, a Hot Standby receives the same input as the primary, and the user threads of the Hot Standby do what they are supposed to do except that the outputs from them are filtered by the network abstraction[4]. In the presence of any task failure detected by the master SAFER daemon, the daemon will send a command to the SAFER daemons with the Hot Standbys for the failed tasks. Then, the process handler of each Hot Standby will receive the command to promote itself to be the primary. A similar process is also applicable to the recovery operation for Cold Standbys. One prime difference is that task needs to be launched first.

The Cold Standby of a task is a dormant binary in memory, triggered only by the failure of its primary. Without a failure, a Cold Standby periodically receives and stores the state information of the primary coming from the status updater of the primary. The disadvantage of using a Cold Standby is that the recovery latency could be long when there is a failure detected by the master SAFER daemon. Conversely, since it runs only on demand, it saves computing resources in the absence of failures. We are extending Linux/RK to minimize the latency of recovery.

[3] Increasing the decision boundary can be one way of extending the assumption beyond the synchronous network. For example, many wireless networks try to send a packet n times in order to transfer it reliably, where n is a positive integer greater than 1. Then, the decision boundary can be adjusted to $d + nT_{heartbeat}$.

[4] We do not generate outputs from Hot Standbys because we assume the fail-stop failure model. To relax the failure assumption model so that we can check if the outputs from the primary are valid, the network abstraction should be modified to compare the results of the primary with the results of its Hot Standbys.

Task	Period	Standby	Detection	Recovery
BehaviorTask	$10ms$	Cold	$22ms$	$12ms$
ControllerTask	$10ms$	Hot	$27ms$	$9ms$
LocalPlannerTask	$100ms$	Cold	$23ms$	$66ms$
Planner3DTask_first	$100ms$	Hot	$14ms$	$28ms$
Planner3DTask_second	$100ms$	Hot	$23ms$	$66ms$

Table 1. Evaluation on time-based detection

Task	Period	Standby	Detection	Recovery
BehaviorTask	$10ms$	Cold	$2ms$	$9ms$
ControllerTask	$10ms$	Hot	$4ms$	$2ms$
LocalPlannerTask	$100ms$	Cold	$6ms$	$12ms$
Planner3DTask_first	$100ms$	Hot	$3ms$	$42ms$
Planner3DTask_second	$100ms$	Hot	$4ms$	$92ms$

Table 2. Evaluation on event-driven detection

3. Preliminary Evaluation

SAFER is implemented on Ubuntu 10.04.3 LTS and deployed on Boss which won 2007 DARPA Urban Challenge [13]. To measure the preliminary performance of the SAFER layer with the presence of a failure, we have built a cluster composed of three Intel Quad-Core machines. We ran a scenario used to test Boss during the competition in 2007 without the perception system. The artificial intelligence algorithms for behavior and planning along with vehicle control were run on the cluster. By injecting processor failures through a script, we measured fault detection time and fault recovery time for different tasks with different periods. The fault detection time is the time duration between when a failure happens and when the master SAFER daemon detects the failure. The fault recovery time is the time duration between when the master SAFER daemon detects the failure and when the failed task is completely recovered.

Table 1 captures one-time measurements when time-based detection is used. From the data, it is seen that the failure detection time depends on the period of the master SAFER daemon, which is $20ms$. Most latencies are longer than $20ms$ due to d, the network time delay of $10ms$. The recovery time of a task is related to its task period because the process handler of its Hot/Cold Standby should be able to receive the command from the master SAFER daemon. Table 2 shows the measurements when event-driven detection is used. Since the local SAFER daemon detects local task failure, the failure detection time is reduced.

4. Related Work

Fault-tolerant distributed embedded systems have been extensively studied in [4, 10, 5, 1]. One clear distinction between the existing work and SAFER is that SAFER provides the framework to support timely failure recovery in a generalized setting. Fault-tolerant scheduling in distributed embedded real-time systems has also been widely researched in [3, 6], which can be potentially used for the inputs to SAFER as a configuration file.

5. Conclusion

We have proposed a layer called SAFER (System-level Architecture for Failure Evasion in Real-time applications) to incorporate configurable task-level fault-tolerance features using Hot Standbys and Cold Standbys in order to tolerate fail-stop processor and task failures for distributed embedded real-time systems. SAFER is implemented on Ubuntu 10.04 LTS and integrated into an autonomous vehicle developed at CMU. We have presented initial measurements using one of the driving simulation scenarios used during the DARPA Urban Challenge. Future work to be done includes supporting the graceful degradation based on load and resource changes. A comprehensive scheduling framework of the primary and its backups can also be integrated into SAFER.

References

[1] J. Balasubramanian, et al. Middleware for resource-aware deployment and configuration of fault-tolerant real-time systems. In *Proceedings of the 16th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2010.

[2] N. Budhiraja, K. Marzullo, F. Schneider, and S. Toueg. The primary-backup approach. *Distributed systems*, 2:199–216, 1993.

[3] J. Chen, C. Yang, T. Kuo, and S. Tseng. Real-time task replication for fault tolerance in identical multiprocessor systems. In *Proceedings of the 13th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2007.

[4] F. Cristian. Reaching agreement on processor-group membership in synchronous distributed systems. *Distributed Computing*, 4:175–187, 1991.

[5] P. Felber and P. Narasimhan. Experiences, strategies, and challenges in building fault-tolerant corba systems. *IEEE Transactions on Computers*, pages 467–511, 2004.

[6] J. Kim, K. Lakshmanan, and R. Rajkumar. R-BATCH: Task partitioning for fault-tolerant multiprocessor real-time systems. In *Proceedings of the 10th IEEE International Conference on Computer and Information Technology (CIT)*, 2010.

[7] M. McNaughton, C. Baker , T. Galatali, B. Salesky, C. Urmson, and J. Ziglar. Software infrastructure for an autonomous ground vehicle. *Journal of Aerospace Computing, Information, and Communication*, 5(1):491 – 505, December 2008.

[8] D. L. Mills. Internet time synchronization: the network time protocol. *IEEE Transactions on Communications*, 39(10):1482–1493, October 1991.

[9] S. Oikawa and R. Rajkumar. Portable rk: a portable resource kernel for guaranteed and enforced timing behavior. In *Proceedings of the fifth IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 1999.

[10] R. Rajkumar and M. Gagliardi. High availability in the real-time publisher/subscriber inter-process communication model. In *Proceedings of the 17th IEEE Real-Time Systems Symposium (RTSS)*, 1996.

[11] R. Simmons. Inter Process Communication (IPC). `http://www.cs.cmu.edu/~ipc` as of January 29, 2012.

[12] C. Urmson. SimpleComms. `http://www.cs.cmu.edu/~curmson/SimpleComms.tgz` as of January 29, 2012.

[13] C. Urmson, et al. Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics Special Issue on the 2007 DARPA Urban Challenge, Part I*, 25(1):425–466, June 2008.

Network-Wide Energy Optimization for Adaptive Embedded Systems

Patrick Heinrich, Christian Prehofer
Fraunhofer Institute for Communication Systems ESK,
Munich, Germany
firstname.lastname@esk.fraunhofer.de

Abstract—**This paper discusses network-wide energy optimization of embedded systems which can adapt by switching configurations. We model applications and their task chains in a network of embedded devices, including sleep modes and change of configuration, which provides a basic adaptation mechanism. We present a formal model of the energy consumption for such systems and apply it to automotive embedded systems. In particular, we develop new potential for network-wide energy savings as well as optimizations for adaptive systems. For instance, we show that non-optimal configurations may lead to a globally optimal system setup, if a system adapts regularly.**

Keywords - embedded systems, energy-efficiency, network-wide optimization, adaptive systems, automotive

I. INTRODUCTION

This paper discusses energy optimization of embedded systems which can adapt and switch configurations. While there is considerable work on energy efficiency for embedded systems, we show that there is a need to model and optimize the network-wide energy consumption of such embedded systems.

We focus on automotive systems which consist of a number of electronic control units (ECUs), connected via a communication network. Applications (e.g. collision warning) consist of a chain of tasks, residing on different electronic control units. For each task chain (i.e. application) different configurations are possible. We consider the change between different applications (and their corresponding configuration) at runtime as an essential adaptation mechanism. The change between applications and their active periods are usage dependent.

In our setting, ECUs have to execute real-time tasks. Systems with hard real-time constraints must meet the given deadlines. We assume that the individual tasks are given by worst-case execution time (WCET), their deadline, cycle time and the dependency on other tasks including the necessary communication. The vehicle applications are typically decentralized, i.e. parts of application software (tasks) are executed at different ECUs. This also means that different task allocations are possible for vehicle software.

Most energy saving technologies optimize the energy consumption of one component (i.e. CPU), mostly at runtime. Examples are dynamic hardware resource management and dynamic voltage/frequency scaling. Other methods are optimizing the efficient use of the resources, such as energy-efficient task scheduling [1, 2] or trade-offs, e.g. reducing quality of service to decrease energy consumption [3, 4].

A lot of research focuses on the reduction of system-wide energy consumption, which means ECU-wide in this paper. For instance dynamic power management (DPM), which deactivate unused components of a system to save energy (e.g. [5]). [6] uses dynamic voltage scaling and DPM to reduce energy consumption system-wide and considers the energy consumption of peripherals (e.g. memory) within standby and activation/deactivation time of the processor. A software framework to use the different hardware energy-saving techniques and find a trade-off between those is presented in [7]. This optimization aims to reduce system-wide energy consumption for different applications during runtime, but the energy for reconfiguration is neglected. The possibilities of network-wide energy optimization are not used in these works. A network-wide optimization is considered within wireless sensor networks, but with the focus of energy efficient message routing.

The intention of this paper is to show the potentials and challenges of network-wide energy optimization of adaptive systems by an analytical model. In particular, we show by example that optimizations for single applications (task chains) may not result in network-wide optimal combination of several, alternating applications. We also show cases where additional hardware may reduce the energy consumption. Our model is based on the analysis of the energy consumption of the individual components and their dependencies. In this paper, we use existing data of their energy consumption. Detailed measurements for specific components and activities like networking are non-trivial and go beyond the scope of this paper.

II. NETWORK-WIDE ENERGY OPTIMIZATION

In this section, we discuss factors and challenges to model energy consumption. Besides the CPU power consumption, we consider the energy to communicate via a network and the energy during sleep modes of components. This hardware may also have additional impact on the energy consumption of a system. An example is the calibration time of sensors, which is necessary to assure accurate results. During that time the ECU has to process a neglectable workload, but changing to sleep mode is typically not possible. In this case, an ECU-wide optimization is difficult, because the local tasks need calibrated sensors. Network-wide optimization is able to allocate additional tasks to that hardware to use the unused ECU resources during calibration time.

The challenge of network-wide optimization is to model the energy consumers and their dependencies correctly. Assuming

a specific set of task chains, one for each application, we aim to find the right assignment of tasks to ECUs and the timing of sleep modes to ensure lowest energy consumption. This is a challenge because of the large number of parameters and aspects. A vehicle may have hundreds of functions and even a single hardware has different energy modes. As an example, [8] has five different energy modes.

We consider adaptive systems where the active applications, expressed as task chains, may vary over time. For each set of active task chains, we have several possible configurations and one has to be selected. Alternating task chains may then switch between different active task chains or just between different configurations of the same task chains.

Changing task chains means activating and deactivating tasks and, if necessary, changing the network configuration, which both consume energy.

Furthermore, the energy consumption depends on the usage of the system. If a configuration is used for a very short time, energy-efficiency is less critical w.r.t. the overall system.

III. Modelling Energy Consumption

In this section, we present a model for adaptive, network wide energy consumption of embedded systems. A simple, running example shall show the energy consumption of different configurations and the advantages of network-wide and usage behavior dependent optimization. The illustrated system is part of a vehicle application which changes its set of task chains (i.e. the applications) depending on external conditions. To simplify the example, the system just has one active task chain at any time. Tasks are executable at every node, except they need local resources (i.e. sensors). For simplicity, we assume that the execution time only depends on CPU frequency and it is assumed that a valid schedule can be found.

In our case study, the first application is object identification using radar sensors, which is used for adaptive cruise control (ACC). ACC uses intelligent object detection (e.g. for obstacles). The ACC task chain uses two radar sensor-ECUs and two ECUs which execute the necessary tasks (T) and communicate using a network. Figure 1 shows a configuration (C1a) and its hardware and the tasks with its dependencies. Task T_0 captures the radar data and preprocess these. Task T_1 identifies objects recorded by the radar sensor and task T_2 compares the results of the object identification. Tasks T_1 and T_2 are used in two instances at the two sensors in the example. Task T_3 is part of an application specific function, e.g. ACC, which is out of scope of this case study.

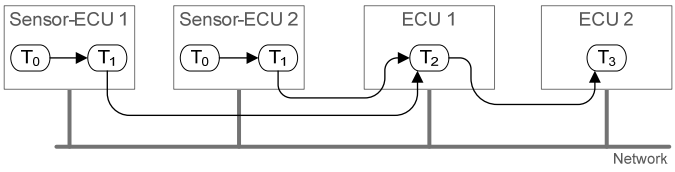


Figure 1: Task chain ACC - Configuration C1a

Assuming ACC is not usable with a vehicle speed below 30 km/h the radar sensor is used for collision warning in this case. Thus we alternate between ACC and the second application, collision warning, in this example. The collision warning application which is less safety critical uses just one radar sensor. This task assembled as configuration (C2a) is shown in Figure 2.

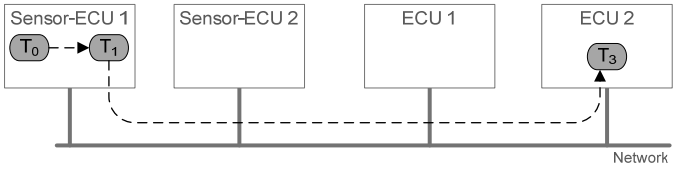


Figure 2: Task chain Collision Warn. - Configuration C2

The tasks and the communication between these have specific parameters. We assume the worst-case execution time at a specific frequency is convertable to the task execution time t_e at frequency f_{te}. The cycle time t_c and the necessary communication per cycle c_t between tasks are also shown in Table 1.

Table 1: Task properties

Task	T_0	T_1	T_2	T_3
Worst Case Execution time	10 ms @ 82 MHz	15 ms @ 82 MHz	10 ms @ 132 MHz	n/a
Cycle time	100 ms	100 ms	100 ms	n/a
Communication per cycle		T_0->T_1: 192 kBit	T_1->T_2: 32 kBit	T_2->T_3: 16 kBit

A. Modeling Energy-Consumers and System Assumptions

In this section, we detail the energy consumers of our case study. Hardware has a processing speed f_{hw} and dependent power consumption P_{hw}. Highly energy efficient processors exist such as [8], which consumes 180 µA/MHz. Other embedded hardware without energy-awareness such as [9] consumes about 5 mA/MHz. Assuming a medium energy-efficient hardware the sensor hardware, which is used here, consumes 800 µA/MHz. The sensor-ECUs work with 82 MHz and a supply voltage of 1.65 V, which result in a power consumption of 108 mW. We assume a more efficient ECU hardware. This needs 500 µA/MHz and consumes 109 mW at 132 MHz and 1.65 V supply voltage. Both components are able to change to a sleep mode, during which the hardware consumes power P_s of 82.5 µW (50 µA). The change to the sleep mode itself and back to normal consumes time t_{cs}, which is supposed to be 5 ms with the power P_{cs} of 108 mW respectively the energy E_{cs} of 0.54 mWs. The number of sleep n_s depends on the tasks' cycle time.

Communication also needs energy for the data transport. Within safety critical systems often time-triggered communication protocols such as FlexRay are used, which need synchronized nodes and a common known communication schedule. This means bandwidth is reserved and a reconfiguration is necessary to reallocate bandwidth. In the specific case of time-triggered communication not only sender and receiver need to be reconfigured, but all nodes within the network need to be reconfigured. We consider here a FlexRay communication controller [10] and the corresponding transceiver [11], for which we can estimate energy consumption as follows. The FlexRay controller has a maximum power consumption of 165 mW and the FlexRay transceiver 175 mW (normal mode). Using the maximum bandwidth of 10 MBit/s a bit consumes 34 nWs/Bit per node, which is a very rough estimation, because of static energy consumers such as for synchronization.

Due to this and for simplicity we estimate a fixed (not per node) energy consumption of 30 nWs/Bit in total within our case study.

In section III.C the energy for changing configuration is considered. We assume an activation/deactivation of one task consumes 1 ms and with that a task activation/deactivation energy E_{tc} of 0.108 mWs, which includes the loading of software into memory and ECU-internal configurations. If the whole ECU changes to sleep mode, just the energy for changing to sleep mode and back is considered. The reconfiguration of the network consumes energy at every node's communication controller as discussed above. [12] measured the switch of a communication schedule including acknowledgement of the nodes which is 80 ms. The energy to change the network configuration E_{nc} within this case study (2 sensor-ECUs and 2 ECUs) is then 108.8 mWs per change. The number of task changes n_{tc}, network changes n_{nc} and ECU mode changes n_{mc} can easily be obtained from the configurations. A further aspect is the different active times of the applications C1 and C2, here C1 is 10% and C2 90% of time activated. This could be the case for a vehicle, which is commonly used within cities, e.g. taxis. The number of configuration changes is assumed to be 60 per hour, e.g. every minute.

The calibration time of the radar sensor (as discussed at section II) is assumed to be 5 ms. This calibration is necessary after every sleep mode and means a restart 5 ms before the first (valid) calculation can be done.

To simplify the calculation, the costs for communication within an ECU are neglected. An ECU just has one sleep mode and a fixed cost for mode switching. The storing and buffering of data is not considered, that means memory costs are neglected. The cost for communication is assumed as fixed energy consumption per transferred bit; no overhead costs (e.g. error checks, sync, keep-alive messages, routing, etc) are analyzed in detail. Sensors are allowed for deactivation, but after activation a specific time period (calibration time) is necessary before a sensor is usable. It is considered that sensor-ECUs and ECUs change to sleep mode after task execution.

B. Finding Optimal Configurations

In this section we present the model to estimate the energy consumption of an application and calculate the energy consumption of different configurations, but without considering the change of configurations. The energy consumption for a specific configuration is modeled with the summation of the energy consumptions of task executions, sleep periods with the associated mode changes and communications. The energy consumption is calculated over a common multiple of all task periods t_{mult}, which also determines the number of sleeps n_s. The equations to calculate the total energy consumption of a configuration are shown at equation (1-4).

$$E_{config} = \sum_{j=0}^{No.\ of\ ECUs} (E_{tasks} + E_{sleeps}) + E_{com} \qquad (1)$$

with

$$E_{\text{tasks}} = t_{task} \cdot P_{hw} = \left(\sum_{i=0}^{Tasks\ at\ hardware} \frac{t_{e,i} \cdot f_{te,i}}{t_{c,i}} \right) \frac{t_{mult}}{f_{hw}} \cdot P_{hw} \qquad (2)$$

and

$$E_{sleeps} = ((t_{mult} - t_{tasks} - n_s \cdot t_{cs}) \cdot P_s) + n_s \cdot E_{cs} \qquad (3)$$

and

$$E_{com} = \sum_{l=0}^{No.of\ network\ communication} (c_t \cdot E_c) \qquad (4)$$

To optimize the energy consumption of configuration C1a (see Figure 1), the task allocation is changed. One possible configuration C1b is shown in Figure 3, which reduces the energy consumption by 2.80%. This is a result of the lower energy consumption of ECU 1 compared to the sensor-ECUs. Figure 4 shows configuration C1c, which reduces the energy consumption further by 2.98%. This energy saving is a result of taking into account the calibration times of the sensors. Comparing configuration C1a and C1c an energy saving of 5.78% is reached. This shows the potential of network-wide optimization.

The application "collision warning" has the configuration C2 (Figure 2). The calculated energy consumptions of all the configurations are shown in detail in Table 2.

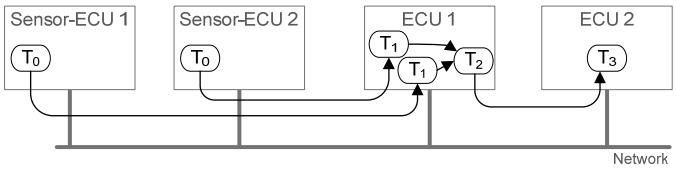


Figure 3: Task chain ACC - Configuration C1b

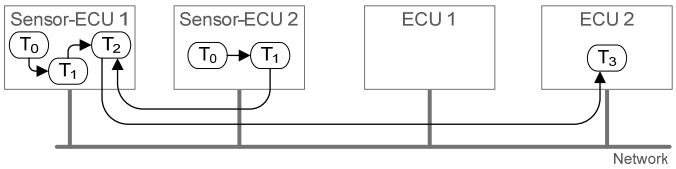


Figure 4: Task chain ACC - Configuration C1c

Table 2: Energy consumption [mWs] of the configurations

Configuration	C1a	C1b	C1c	C2
Task Execution	75.83	63.66	76.96	32.47
Sleep Periods + Mode Change	0.017 + 16.20	0.019 + 16.20	0.018 + 10.80	0.022 + 5.40
Data Transport	2.40	12.00	1.44	0.96
Total	**94.45**	**91.88**	**89.21**	**38.85**

C. Optimal Configurations for Alternating Configurations

In this section we show that the combination of C1a and C2 results in a more energy efficient system instead of the combination of the most energy efficient configurations C1c and C2.

The vehicle may change task chains to execute different functions or to be more energy efficient. Nevertheless changing ECU and network configuration also consumes energy as discussed in Section III.A. Another aspect is the length of stay within a configuration and the number of changes n_{cc} during a time period. Equations (5) and (6) show the calculation of the total energy consumption including energy for configuration changes E_{cc}. Factors α and β represent the percentage within the configurations.

$$E_{total} = \left(E_{config,a} \cdot \alpha + E_{config,b} \cdot \beta\right) + n_{cc} \cdot E_{cc} \qquad (5)$$

with

$$E_{cc} = E_{tc} \cdot n_{tc} + E_{nc} \cdot n_{nc} + E_{cs} \cdot n_{mc} \quad (6)$$

Combining the C1c and C2, which are individually the energy efficient options, results in a set of alternating configurations as shown in Figure 5. (Note that the two alternating task chains are shown in one figure, even though not executed simultaneously.)

However, the set of configurations C1a and C2 as shown at Figure 6 is more energy efficient (2.89%), because of the bigger similarity of the configurations. In detail, the set "C1c+C2" has to (de)activate one task, change mode of sensor-ECU 1 and reconfigure the network. (Note that communication slot of task T_2 is not usable of task T_1, because of different size/bandwidth.) Configuration set "C1a+C2" has to change sensor-ECU 2 and ECU 1 to sleep mode and back, but no task (de)activation and no network reconfiguration. Table 3 shows the energy consumptions of these two sets. In this particular case, changing the network dominates the additional energy consumption. The small number of tasks has no real influence to the energy consumption which is not the case for larger systems.

This shows that a combination of most energy efficient configurations may lead to sub-optimal energy efficiency. The reasons are the energy necessary to change configurations and the usage profile.

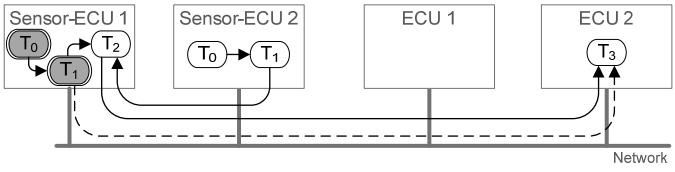


Figure 5: Alternating configurations (C1c+C2) (not active at the same time)

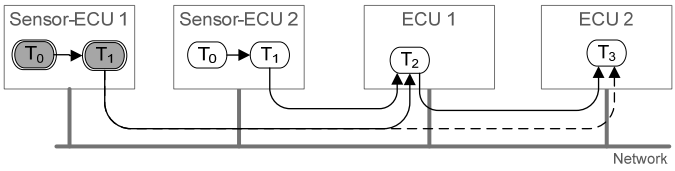


Figure 6: Optimal set of configuration (C1a+C2), which alternate (not active at the same time)

Table 3: Total energy consumption [mWs] of the sets of configurations

Configuration	Without energy for configuration changes	With energy for configuration changes
C1c + C2	43.89	45.71
C1a + C2	44.41	44.43

IV. Conclusion and Further Steps

This paper has discussed and analyzed the energy consumption of decentralized adaptive systems. Based on examples the potentials of network-wide optimization and the effects of configuration to the total energy consumption were shown. The goal was to present the potentials and challenges of network-wide energy optimization of adaptive systems. We have shown that different configurations of a single task chain may have considerable differences in energy consumption of 5.78%. In case of alternating task chains, locally optimal configurations do not result in globally optimal configuration. In our model the energy consumption difference due to this is 2.89%. The reasons are energy consumption for configuration changes and the usage profile of the system. Similarly, adding additional hardware may improve energy efficiency. This results in further research challenges and also new energy saving possibilities.

Our model is based on existing data on energy consumption. Validating our model by measurements would require highly detailed measurements (in terms of time and functional isolation) and is not covered in this paper.

Our approach and first results allow one to design the configuration depending on the use of the car to be more energy efficient. E.g. the configuration for taxis, which are most time in cities and below 60 km/h, may differ from normal cars. Another possibility is the calculation of different valid configurations, which are only energy-efficient for a specific kind of usage. The decision, which set of configuration is used, is done during vehicle runtime, e.g. based on the route of the car, which is known due to the navigation system.

References

[1] Dong-In Kang, S. Crago, and Jinwoo Suh, "A fast resource synthesis technique for energy-efficient real-time systems," in *Proceedings of the 23rd IEEE Real-Time Systems Symposium RTSS 2002*, IEEE, Ed, 2002.

[2] Jingcao Hu and R. Marculescu, "Energy-aware communication and task scheduling for network-on-chip architectures under real-time constraints," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE'04)*, IEEE, Ed, 2004, pp. 234–239.

[3] C. A. Rusu, R. Melhem, and D. Mosse, "Maximizing the system value while satisfying time and energy constraints," *IBM J. Res. & Dev,* vol. 47, no. 5, pp. 689–702, 2003.

[4] M. Baker, *Topics in power and performance optimization of embedded systems*, Dissertation, Arizona State University, 2011, Available: http://hdl.handle.net/2286/jq8f23vj0yh

[5] L. Benini, A. Bogliolo, and G. de Micheli, "A survey of design techniques for system-level dynamic power management," in *Transactions on Very Large Scale Integration (VLSI) Systems*, IEEE, Ed, 2000.

[6] R. Jejurikar and R. Gupta, "Dynamic voltage scaling for systemwide energy minimization in real-time embedded systems," in *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED'04)*, 2004, p. 78.

[7] G. Zeng, H. Tomiyama, H. Takada, and T. Ishihara, "A Generalized Framework for System-Wide Energy Savings in Hard Real-Time Embedded Systems," in *Proceedings of the 5th International Conference on Embedded and Ubiquitous Computing EUC 2008*, 2008, pp. 206–213.

[8] Energy Micro, *EFM32G Reference Manual.* Available: http://cdn.energymicro.com/dl/devices/pdf/d0001_efm32g_reference_manual.pdf.

[9] Freescale Semiconductor, "MPC5554-Microcontroller Data Sheet," 2008.

[10] Freescale Semiconductor, "MFR4310 Reference Manual: FlexRay Communication Controllers," 2008.

[11] NXP Semiconductors, "TJA1080A FlexRay transceiver - Product data sheet," 2011.

[12] P. Heinrich, D. Eilers, R. Knorr, M. Königer, and B. Niehoff, "Autonomous Parameter and Schedule Configuration for TDMA-Based Communication Protocols Such as FlexRay," in *Proceedings of Int. Joint Conference of IEEE TrustCom-11/IEEE ICESS-11/FCST-11,* IEEE, Ed, 20

Model-driven development of SOA-based Driver Assistance Systems

Marco Wagner, Ansgar Meroth
Automotive Systems Engineering
Heilbronn University
Heilbronn, Germany
{marco.wagner, ansgar.meroth}@hs-heilbronn.de

Dieter Zöbel
Institute for Software Technology
University of Koblenz-Landau
Koblenz, Germany
zoebel@uni-koblenz.de

Abstract— This paper describes an approach towards model-driven development of SOA-based Driver Assistance Systems. In the field of assistance systems for truck and trailer combinations Service-oriented Architecture (SOA) is a promising approach to handle the heterogeneity and the high degree of distribution of these systems. Through connecting or disconnecting trailers the system is very likely to change at runtime which sets up the demand of runtime adaption. This paper illustrates a process model to use SoaML for modeling the components and architectures of these systems. Based on these models, model-driven runtime adaption can be carried out.

Keywords-component; Embedded Systems, Driver Assistance Systems, Runtime Adaption, Process Model, Service-oriented Architecture (SOA), SOMA

I. INTRODUCTION

Modern Driver Assistance Systems (DAS) are supporting the driver in many situations. They, for example, assist the driver while changing lanes or influence the brakes of a car in order to keep it on the track. A new category of DAS are systems which support the driver backing up articulated vehicles. From an architectural point of view these systems are quite special. This is mainly because the components needed are distributed over at least two separate units. The connection of these units is not permanent and one pulling vehicle may be hooked-up to several different trailers over time. An example of these systems is the visual assistance as shown in Figure 1. The idea is to calculate the trajectories of the trailer and the overall vehicle and to overlay these on the picture of a rear view camera mounted on the trailer [16]. In order to do so, the steering angle and the angle between truck and trailer, the so-called bending angle, are determined to calculate these trajectories.

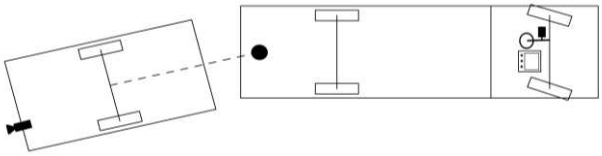


Figure 1 Components of the steering assistance for one-axle trailers.

The previously mentioned system is just one option. Within the real time systems group of the University of Koblenz-Landau several other assistance approaches have been developed. Besides a visual human computer interface the modality used could as well be acoustical or tactile. An approach using modified semantics of the steering wheel has been also investigated [17]. The number of possible variations of the system is highly increased considering the different types of trailers that could be used. This multitude of variations combined with a high degree of distribution of the heterogeneous subsystems and the possibility that the system could change at runtime through disconnecting or connecting one or more trailers cannot be handled by state-of-the-art software architectures in the automotive domain. Therefore, we proposed a novel approach in [1], using service-orientation combined with software agents. In this approach, all functionality is encapsulated in fine-grained services. These services may be located on any device within the articulated vehicle or even on a nomadic device like a Smartphone. In order to set up the assistance one or more software agents discover the currently available services, determine the possible types of assistance and orchestrate the services. This re-configuration is done every time the configuration of the vehicle changes, for example, by connecting a trailer or in case an electronic control unit (ECU) or sensor system fails to work.

In this paper, we focus on identifying a modeling language as well as a process model to specify adaptive embedded systems. As a case study we are using the assistance approaches introduced earlier. With the chosen language and methodology, we aim on collecting and formalizing the assistance approaches being developed so far. We also want to identify the functionalities needed in each type of assistance in order to find similarities. In the next step, these functionalities are converted into service specifications. Using these specifications architectures can be developed for each type of assistance. By merging the modeled architectures a library of SOA-based assistance systems is formed. This library along with the specifications of the services provides the basis for deploying model-driven runtime adaption.

The remainder of this paper is organized as follows: Section II introduces and categorizes several process models for

developing Service-oriented Architectures. In Section III, the customized process model we use is presented and differences to state-of-the-art approaches are pointed out. Section IV concludes the paper and provides information on the future work within this project.

II. Process Models for Service-oriented Architectures

With the paradigm of Service-orientation getting more and more popular, the number of process models to develop such systems went up, too. In 2009 Thomas, Leyking and Scheid identified 21 different approaches in [2]. Most of the currently available models are built to suit for some special purpose, require a particular tool chain or concentrate on one field of application only. However, none of them suits to the domain of automotive SOA solutions. Instead of developing yet another model, we decided to find a process model that can be customized to this special scenario. In order to do so, criteria have been developed and the available approaches have been evaluated based on these. The following criteria have been defined:

1. Completeness of the specification phase
2. Independence of a specific field of application
3. Variability in the scenario of development
4. Tool support
5. Acceptance of the modeling language

Our first criterion is that the modeling approach has to allow a complete system specification which includes the specification of the services as well as the service architectures. This also implies that a detailed technical point of view should be assured rather than focusing on the business domain which is very common using SOA. Finally, concrete methods or techniques on how to carry out the steps within the process model should be proposed.

The second criterion is that the field of application should not be restricted. Specialized models, used for Web Services for example, are not very promising since their focus is too narrow. Converting these to suit embedded automotive systems would change too many of their essential ideas if possible at all.

Another criterion is that the starting position at the very beginning of the process should be variable. This is important because the process model should allow new developments as well as migrating existing systems into SOA.

The fourth criterion is that tool support should be given. Using a tool that for example allows modeling the system graphically reduces development time. In addition, implemented validation functionalities decrease the probability of semantic errors.

Finally, the last criterion is that the modeling language deployed is widely-used and hereby accepted. This demand is set up to ensure the readability of the models in the scientific community.

Using these criteria, eleven process models are analyzed. The first one is a model proposed by Stein and Ivanov in [3]. The model is based on ten phases starting with a business process model ending with the deployment of the developed system. It focuses on business processes and the modeling languages suggested belong to the domain of Web Services. A similar model, the Enterprise SOA Roadmap method is presented in [4]. This model also emphasizes on business modeling since only one of the five steps to be executed is technical. Both of the process models violate criteria two that they shouldn't restrict the area of application.

Other approaches lack concrete modeling techniques. Pingel [5] for example, introduces a technology independent five phase model extending well-known approaches. Another approach in this category is a proposal of Mathas [6] which extends the software lifecycle model by adding some SOA-specific tasks and roles while staying coarse-grained. The Service-oriented Modeling Framework developed by Bell is quite generic, too [7]. The idea of the author is to design a concrete process model for every case of application derived from his abstract methodology. Bell also proposes a special design notation which violates the criterion of using a widely-used modeling language. All these models are rather to be seen as suggestions on how a process model may be set up than being a concrete model itself.

Unlike the previously named ones the models "Service-oriented design and development (SOAD)" [8] by Papazoglou and van den Heuvel and "Creating Service-oriented Architectures (CSOA)" [9] developed by Barry are technical in nature. Both of them are phase-oriented and contain practical techniques to be performed in those phases. Through basing on modeling languages like the business-oriented "Business Process Modeling Language (BPML)" or the "Business Process Execution Language for Web Services (WS-BPEL)" they cannot be used for other fields of application without major changes. This fact violates criterion two.

Another approach is presented by Nadhan in [10]. The author describes a seven step procedure to migrate an existing solution into a SOA-based system focusing on technical issues. Targeting only on the migration scenario this model cannot be used for new developments. In doing so criterion 3 is violated.

Some highly interesting approaches are using the Service-oriented modeling language (SoaML), a notation created to model and design SOA-based systems. This is a promising approach because the language itself satisfies the criteria set up in being not restricted to one field of application and being widely used since it is a profile of the popular Unified Modeling Language (UML). One of these process models is presented in [11]. The authors describe the development of a Service-based monitoring system by identifying and specifying the needed services. Although this is very promising, it does not allow specifying the architecture of the overall system which violates the criterion of enabling the user to carry out a complete system specification. Another methodology using SoaML introduced in [12] closely follows the processes defined in the Model-driven architecture (MDA) approach published by the Object Management Group. Tool support is granted by the modeling tool "Modelio". This process model defines several specification steps within the

computational independent model and the platform independent model of MDA. The approach is very close to "Service-oriented Modeling and Architecture (SOMA)" presented in [13]. This phase-oriented lifecycle model is based on the "Rational Software Architect" and is also free of any restrictions with respect of the area of application. Both of the lastly named methodologies are fitting the criteria set up earlier in this paper. The reasons why SOMA is favored is being more focused on technical issues and offering a more straightforward workflow.
In the next section SOMA is presented in detail and the changes to suit it to embedded automotive systems are explained.

III. Customizing SOMA for automotive embedded Systems

A. Introduction to SOMA

The SOMA methodology has been published by Arsanjani in 2004 [14]. The idea of this approach is to set up a phase-oriented process model that guides through the whole development process. The different phases of the model can be seen in Figure 2. Within the first step named "Business modeling and transformation", the requirements, namely the business processes are modeled and optimized to get a semi-formal description of the workflow. This is normally done using the Business Process Model and Notation (BPMN). Simultaneously, the concomitant project management processes are defined and the computation platform to be used is selected. In SOMA this step is called "Solution management".

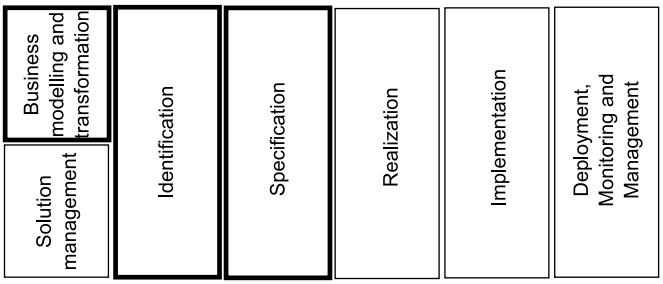


Figure 2 Overview of the SOMA phases [13].

In order to achieve an architecture model, first the service candidates are identified based on the components and flows of the business model. Therefore, SOMA recommends a number of identification techniques that might be used. Next, within the specification phase, the candidates are transformed into services. This is done by modeling the Service Interfaces, Service Contracts and the Participants which realize the functionality of the services. Also, the Service Architecture of the overall system is defined. In the next phase, the so called "Realization", the focus swaps from functional to non-functional requirements. This includes for example the development of the abstraction layers or the communication model. The following step "Implementation" is used to generate or write code that realizes the functionalities and in addition, the code is being tested to fulfill its requirements. In the last phase of SOMA the developed system is put into operation. The functionalities are monitored at runtime and the infrastructure and the network are managed to ensure stability and performance.

In the next subsection, our approach for a Model-driven development of SOA-based Driver Assistance Systems will be described in detail. Focusing only on the functional issues of the system, the business process based SOMA approach is customized towards a methodology suitable of handling embedded automotive applications. Therefore, the phases "Business modeling and transformation", "Identification" and "Specification" are refined.

B. Customized phases of SOMA

In its first phase, Service-Oriented Modeling and Architecture conducts the development of a business process model. This step aims at identifying the tasks and parties within the workflow. Therefore, SOMA recommends the usage of BPMN as a graphical representation to specify business processes. This makes sense for developing SOA solutions in a business context. BPMN is, however, not created to describe technical systems like DAS. To solve this issue, we propose to use an UML 2 Activity Diagram. Similar to BPMN models, Activity Diagrams describe workflows consisting of a number of activities. These activities are important here because they accumulate the functionalities of the system. As they are not restricted to the business domain, Activity Diagrams allow modeling embedded systems without violating the semantics of its components. Figure 3 shows the Activity Diagram of the visual steering assistance system introduced in Section 1. All actions that have to be done to carry out the assistance are modeled as activities. The control flow describes how they cooperate to represent the DAS.

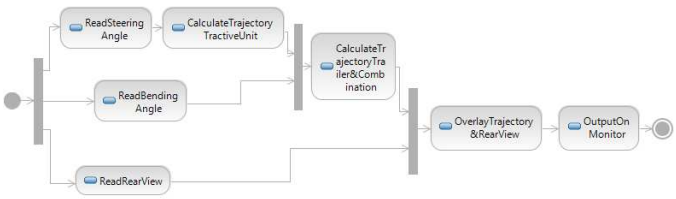


Figure 3 Activity model of the visual steering assistance.

The Activity Diagram itself may be modeled using any kind of description of the system. This includes a specification as well as a systems requirement model or a description in natural language. Within a migration scenario, code may be analyzed to create the diagram.
Having finished the modeling of the workflow using an Activity Diagram, the next step is to identify the service candidates. In SoaML they are called Capabilities [15]. These Capabilities represent entities that offer some distinct functionality and therefore are predestined to become services. Following the recommendations of SOMA, one of the most straightforward ways to identify these service candidates is to

analyze the BPMN model created in the first step. This is done by extracting the lanes of the model which represent some participating party and transfer them into Capabilities. The tasks executed by these parties are modeled as the operations those Capabilities provide. Eventually, this leads to a coarse-grained model with a relatively low number of services.

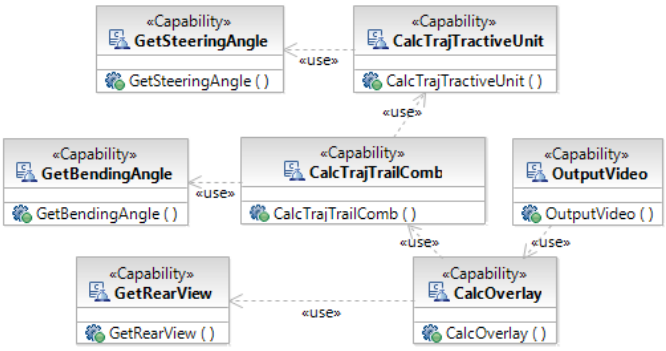


Figure 4 The Service Candidates derived from the Activity Diagram.

In order to design a highly flexible SOA-based Driver Assistance System, we are confident that the services identified with the SOMA method are too coarse-grained. Within this context, the granularity should be enlarged to a certain extend. Considering this demand and the fact that the starting point of this step has changed from a BPMN model to an Activity Diagram, the identification phase has to be changed. In our modified phase, we transfer each activity of the Activity Diagram shown in Figure 3 into a service candidate. The result of this transformation in the case of our example system can be seen in Figure 4. For example, the Capability "GetSteeringAngle", which reads out the current steering angle, may be used in other assistance scenarios as well. A more coarse-grained modeling might prohibit such re-use.

The next phase of the SOMA methodology is the specification. Since this phase is very extensive, it is split up into four sub-phases; specification of the Service Interfaces, Service Contracts, Participants and the Service Architectures.

The specification of the Service Interfaces in SOMA is done by deriving them from the Capabilities. In order to do so, each Capability is represented by a single Service Interface. Furthermore SOMA recommends to specify a number of sub-interfaces of the UML type "Interface" and to assign the operations of the capability to one of these sub-interfaces. Beyond that, no rules or guidelines are given on how many sub-interfaces should be created or how the operations should be distributed onto these.

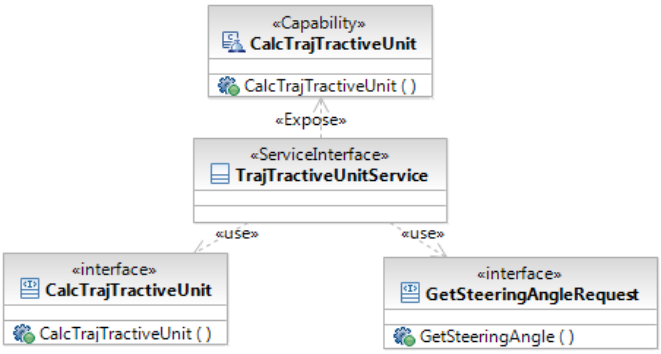


Figure 5 The Service Interface of a service to calculate the trajectory of the drawing vehicle.

Obtaining a common structure is essential to be able to use the model for runtime adaption. Therefore, we have to extend SOMA in this phase, too. This is done by defining two extra rules. At first, any functionality that is provided by the service is mapped into its own sub-interface. These sub-interfaces are so called provided interfaces. The second rule is to create a sub-interface for any functionality that is needed by the service in order to fulfill its tasks. The interfaces modeling the need for a particular service are called requested interfaces. The result of these guidelines can be seen in Figure 5. For example, the interface of a service is shown that offers to calculate the trajectory of the pulling vehicle. This provided service can be seen on the left encapsulated into its own sub-interface. To be able to calculate the trajectory, it needs the current value of the steering angle. This necessity is expressed by the sub-interface on the right.

In a second specification step, Service Contracts are defined. For interacting with a Service Interface the consumer needs to know how to access it. Therefore one or more Service Contracts are defined. Service Contracts formalize the exchange of information between the provider and the consumer of a service [15]. SOMA develops contracts by specifying two attributes: the roles and the protocol of such a service call. The roles can either be "Service Interface", "Interface" or "Class" types according to the SoaML specification. Describing the protocol, any adequate diagram defined in UML may be used such as interaction or state diagram.

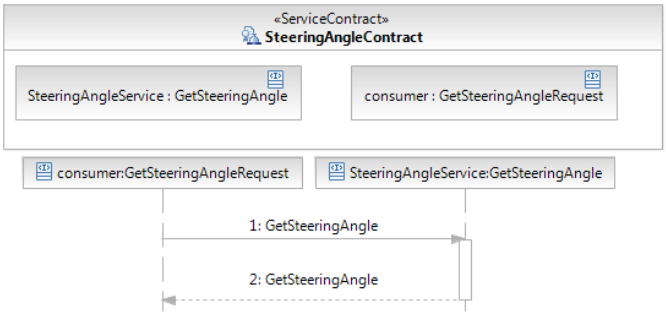


Figure 6 The Service Contract of a Service to determine the steering angle of the drawing vehicle.

We decided to add several constraints to the generic SOMA approach, in order to use the contracts for runtime adaption. One of these constraints is the obligatory use of a Sequence Diagram to model the protocol. This regulation helps to keep a common structure while the Sequence Diagram is able to model further attributes such as time limits. The second constraint is that the messages exchanged are in Remote Procedure Call (RPC) style. Compared to the document style exchange used for Web Services, this method keeps the amount of data transmitted low which is crucial for embedded computing. Figure 6 presents such a contract developed by the changed SOMA methodology using the example of a service developed to determine the steering angle of the vehicle. In this contract, two roles are defined: a provider called "SteeringAngleService" and a consumer. The RPC style protocol is defined in a dedicated Sequence Diagram pictured at the bottom of the figure.

Step three within the specification phase of SOMA is the introduction of Participants. In the systems domain a Participant might be a system, application or component that offers or consumes a service [15]. SOMA uses this type as some kind of particular unit. Therefore, a Participant is created and assigned with one or more ports where each port represents a Service Interface. The idea is to map the functionalities encapsulated within the services to hardware units to use these units for the Service Architecture specified in the last step of the specification phase. The Service Architectures being developed by this approach are rather System Architectures. This is because they do not only illustrate the relations between the software components but also between the hardware components hosting the software.

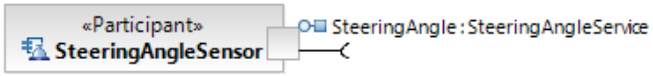


Figure 7 The Participant realizing the steering angle Service.

Since one of the goals of our approach is to allow the services to be distributed on any ECU within the vehicle combination, we do not want to map them onto hardware entities at this point. Therefore, SOMA has to be modified at this step, too. In doing so we are using the broadly framed specification of a Participant in SoaML. Since a Participant is defined to be a unit that provides or consumes services, it is also possible to specify it to be an instantiated process. This process may run on any hardware system of the vehicle. This definition avoids mapping the services to a particular hardware device without violating the specification of SoaML. For example, the Participant realizing the steering angle service is shown in Figure 7.

The last step of the SOMA specification phase is to specify the architecture of the overall system. The Service Architecture illustrates the relationships between the participants involved using ports and contracts. This is done by assigning the ports of the participants to roles within the contracts. This time, we are able to adopt the procedure recommended by SOMA. The idea of our approach is to create a Service Architecture model for every type of DAS for articulated vehicles developed. An example can be seen in Figure 8.

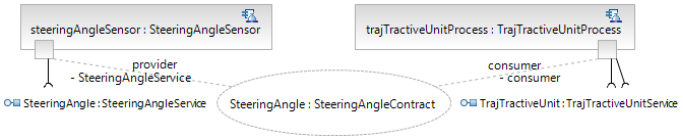


Figure 8 Part of the Service Architecture for the visual steering assistance.

In this figure only a small part of the architecture is shown in order to obtain lucidity. It shows how the Participant realizing the calculation of the trajectory of the drawing vehicle uses the contract of the steering angle service to obtain the data needed. Finalizing the specification phase by modeling the Service Architectures, the phases of SOMA conducting the modeling of functional attributes is finished. Since the development of non-functional components is not in scope of this paper our modified SOMA process model is completed.

C. Model-driven adaption

As a result, this customized process model helps to build up two different databases that can be used for model-driven runtime adaption. First of all, a Service Inventory and hereby a catalog of functionalities is established. It contains a list of the services as well as a description of what they do and how they can be accessed. Second, a library of Service Architectures is created. This library forms a well-defined collection of assistance types. Using this data, the types of the services needed to represent a specific kind of assistance can be determined. These databases form the basis of two different adaption approaches.

The first idea is to pursue an architecture-driven approach. Using a software agent that overlooks the whole system, the currently available services are determined. This is followed by matching them to the catalog of Service Architectures. In doing this the types of assistance realizable can be detected. Additionally, the information about the relationships between the Participants realizing the service can be used to connect them and build up the assistance system.

The information modeled in the Service Interfaces could be used to execute adaption as well, using an interface-driven approach. Since every Service Interface contains not only the services provided but also the services consumed, it is able to explore whether the requested services are currently available within the system. Starting from a data sink, this could be used to determine possible assistance types as well. In the given example, the video out, which is able to offer visual assistance, would start looking for an overlay service which is modeled as its requested interface. The search for this service can be done by invoking the Service Discovery functionality. Having found such an overlay service in the current vehicle configuration, this service itself starts to search for its requested partners. If the chain can be finished and every service requested can be found, the kind of assistance is ready

to be used. If some service needed is missing, the adaption mechanism stops.

IV. Conclusion and future work

By modifying SOMA, we have found an approach to model embedded automotive systems. In order to evaluate the process model, several types of assistance systems have been specified. The systems successfully modeled so far, are the visual assistance for the one-axle and two-axle trailer as well as the acoustical and haptic assistance for the one-axle trailer. The assistance using modified semantics of the steering wheel has also been specified using this approach.

The process model presented fulfills the demands described in Section II. We are able to collect and formalize already existing assistance approaches as well as new developments. By finding service candidates, the functionalities needed within the different DAS are identified. Since theses functionalities are merged into a common database, similarities can be detected. Going through the different steps of the specification phase these functionalities are converted into service specifications as well as architecture specifications. This data is collected within two databases and allows to be used for model-driven runtime adaption as described in Section III.

Having now established the process model and specified the functional attributes of the services and Service Architectures, we are now able to move on with the non-functional components.

The next step is the implementation of a Quality of Service (QoS) parameter for each service. This parameter needs to reflect the performance of each service affected by influences from inside and outside the component. It should be easily computable and allow a comparison between services of the same functionality. The QoS parameter will also be taken into account when a service selection algorithm is set up. We also aim on using the parameterized model for online verification conducted using formal methods. This will be done by transferring the SoaML model into hybrid automata.

Another element of our future work is to define a communication model. State of the art in the modern automobile has the ECUs are connected using a mixture of automotive specific network systems. The communication system to be developed has to be highly flexible and independent from the kind of network used. At the same time the overhead produced should be minimal. Achieving this, the unique characteristics of automotive network systems will be taken into account.

The integration of the approach into AUTOSAR will also be discussed within the project.

The last step of the project will be to validate the architecture using a full scale prototype.

Acknowledgment

Marco Wagner has been supported by a grant of the “Thomas Gessmann-Stiftung”, Essen, Germany.

References

[1] Wagner, M., Zöbel, D. and Meroth, A. Towards an adaptive Software and System Architecture for Driver Assistance Systems. In Proceedings of the 4th IEEE International Conference on Computer Science and Information Technology (ICCSIT 2011) (Chengdu, China, June 10-12, 2011). Wiley-IEEE Press, Piscataway, NY, 2011, Vol. 4, 174-178.

[2] Thomas, O., Leyking, K. and Scheid, M. Serviceorientierte Vorgehensmodelle: Überblick, Klassifikation und Vergleich. Informatik Spektrum, 33 (4). 363-379.

[3] Stein, S., Ivanov, K. Vorgehensmodell zur Entwicklung von Geschäftsservicen. in Fähnrich, K.-P. and Thränert M. Integration Engineering – Motivation, Begriffe, Methoden und Anwendungsfälle, Leipziger Informatik-Verbund, Leipzig, 2007.

[4] Hack, S. and Lindemann, M. Enterprise SOA Roadmap. SAP Press, Bonn, 2007.

[5] Pingel, D. Der SOA-Entwicklungsprozess. in Starke, G. and Tilkov, S. SOA-Expertenwissen, DPunkt Verlag, Heidelberg, 2007, 187-200.

[6] Mathas, C. SOA intern. Hanser Verlag, Munich, 2007.

[7] Bell, M. Service-Oriented Modeling. John Wiley & Sons, Hoboken, NJ, 2008.

[8] Papazoglou, M. and Van Den Heuvel, W.J. Service-oriented design and development methodology. International Journal of Web Engineering and Technology, 2 (4/2006), 412-442.

[9] Barry, D. Web services and service-oriented architecture. Morgan Kaufmann Publishers, San Francisco, 2003.

[10] Nadhan, E.G. Seven Steps to a Service-oriented Evolution. Business Integration Journal, 1 (2004), 41-44.

[11] Gebhart, M., Moßgraber, J., Usländer, T. and Abeck, S., SoaML-basierter Entwurf eines dienstorientierten Überwachungssystems. in 40. Jahrestagung der Gesellschaft für Informatik, (Leipzig, 2010).

[12] Elvesæter, B., Carrez, C., Mohagheghi, P., Berre, A.-J., Johnsen, S. G. and Solberg, A. Model-Driven Service Engineering with SoaML. in Dustdar, S. and Li, F. Service Engineering, Springer-Verlag, Vienna, 2011, 25-54.

[13] Arsanjani, A., Ghosh, S., Allam, A., Abdollah, T., Ganapathy, S. and Holley, K. SOMA: A method for developing service-oriented solutions. IBM Systems Journal, 47 (3), 377-396.

[14] Arsanjani, A., Service-oriented modeling and architecture: How to identify, specify, and realize services for your SOA, 2004. Retrieved August 12, 2011, from IBM developerWorks: http://www.ibm.com/developerworks/library/ws-soa-design1/.

[15] The Object Management Group (OMG) Service oriented architecture Modeling Language (SoaML) - Specification for the UML Profile and Metamodel for Services (UPMS). Specification Beta 2, OMG, Needham, MA, 2009.

[16] Berg, U. and Zöbel, D., Visual Steering Assistance for Backing-Up Vehicles with One-axle Trailer. in Vision in Vehicles 11, (Dublin, 2006), North-Holland Publishing.

[17] Berg, U. and Zöbel, D., Gestaltung der Mensch-Maschine-Interaktion von Lenkassistenzsystemen zur Unterstützung der Rückwärtsfahrt von Fahrzeugen mit Anhänger. in Mechatronik 2007 Innovative Produktentwicklung, (Wiesloch, 2007), VDI, 575-588.

Modeling and Analysis of Adaptive Embedded Systems using Adaptive Task Automata

Leo Hatvani
Mälardalen University
721 23, Västerås, Sweden
Email: leo.hatvani@mdh.se

Cristina Seceleanu
Mälardalen University
721 23, Västerås, Sweden
Email: cristina.seceleanu@mdh.se

Paul Pettersson
Mälardalen University
721 23, Västerås, Sweden
Email: paul.pettersson@mdh.se

***Abstract*—As embedded systems are gaining more significance in many branches of the industry, and as more functionality is moving from hardware to software, industry is pushing at the limits of modeling methods for embedded systems. The importance of this is most obvious in the hard real time systems area where a deadline miss can lead to a catastrophic failure.**

In this paper we present an overview of the current functionality implemented in ATA framework, as well as some of the challenges encountered during the development. We end the paper with some future outlooks for the ATA framework.

I. Introduction

Modern industrial systems are constantly facing the possibility of encountering component failure, or unexpected situations in which the system may be forced to operate under lower capacity. Luckily, many of such systems can be constructed to provide different levels of quality of service. Various systems that regulate quality of service in relation to the available operational capacity already exist (e.g. a voice compression codec can reduce bitrate if not enough bandwidth is present), while additional research is being done to provide safety critical systems with adaptivity features. Consequently, in such cases, formal verification must cater not only for the temporal and functional properties of a system, but also for its ability to dynamically adapt itself, as a response to external and/or internal stimuli.

In this work, we present a high-level overview of the Adaptive Task Automata (ATA) framework (section III) that we have recently proposed [1], in which we have focused on providing the possibility to model and verify systems where the task set can be regulated based on the extra functional system properties. The currently available properties for modeling of the system are related to the schedulabilty of the individual tasks in the system as well as the schedulability of the entire system. By analyzing the schedulability of the system at a runtime, it becomes possible to model systems which will automatically keep themselves schedulable in all possible situations, as demonstrated by the example in the section IV. To fully comprehend the ATA framework, a short overview of the related framework Task Automata is presented in the section II.

II. Overview of Task Automata

The model of *task automata* is a model for real time systems with asynchronous tasks, first introduced with non-preemptive scheduling by Norström et.al [2] and extended to include preemptive scheduling and dynamic priority scheduling by Fersman et.al [3], [4]. By basing our work on this model, we are able to model and verify schedulability of the embedded systems with a task release patterns that can be described in the model of timed automata and executed by scheduling policies with static or dynamic priorities, such as fixed priority scheduling, earliest deadline first, etc. Most of the work on task automata is handling uniprocessor systems, but supports an array of scheduling policies.

Since task automata can be encoded as timed automata [5], they can, in principal, be analyzed using the existing tools created for verification of timed automata such as Uppaal[1] [6], Kronos[2] [7] or others. However, the tool Times[3] [8] has been presented to conveniently support modeling, simulation, schedulability analysis, formal verification and code generation in the model of task automata.

In the next few paragraphs we will give an intuitive description of the task automata model. For a more formal definition, we refer the reader to the paper by Fersman et.al [4].

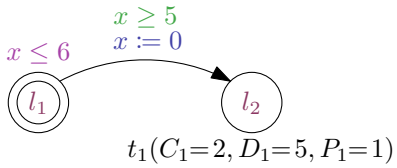


Fig. 1. A task automaton snippet

A simple automaton modeling task release pattern is presented in Figure 1. It consists of: two locations, one edge, and a clock x. Location l_1, denoted by double concentric circle, is the starting location. An invariant ($x \leq 6$) is tied to the location l_1. The edge between l_1 and l_2 is annotated by a guard $x \geq 5$, and a clock reset $x := 0$. The guard controls that the transition cannot be taken if the clock x has value of less than 5, while

[1] http://www.uppaal.org/
[2] http://www-verimag.imag.fr/DIST-TOOLS/TEMPO/kronos/
[3] http://www.timestool.com/

the invariant of the location l_1 defines that the location should be left before the clock x goes over 6.

Once the edge is taken, two things happen: clock x is reset to zero and the task t_1 is released. Task t_1 is denoted by a triple $(C_1 = 2, D_1 = 5, P_1 = 1)$ that defines task's computation time C_1, relative deadline D_1 and priority P_1.

Once the task is released it is added to the task queue. Task queue is formed as $q = [t_i(c_i, d_i), \ldots, t_j(c_j, d_j)]$, where c_i is the remaining computation time and d_i a relative deadline. To be able to verify running of a system now constructed, tasks need to be executed in some way.

Task execution is modeled via a scheduler function which takes two parameters: task queue q and a non-negative integer number δ and returns new task queue q' which represent q after being executed for δ time units. For the previously mentioned q, assuming that the task t_i is currently running on the CPU, result would be $q' = [t_i(c_i - \delta, d_i - \delta), \ldots, t_j(c_j, d_j - \delta)]$. Task has been successfully executed once c_i reaches zero and d_i is greater or equal to zero. If d_i reaches value of zero first, task has broken its deadline and the system is considered unschedulable.

During the verification, the individual automata are connected into an automata network, against which reachability properties are evaluated. In the rest of the paper, automata that are modeling the task release patterns will be referenced as task release automata.

III. Adaptive Task Automata

In the task automata model, the interface between task release automata and the rest of automata network that simulates the execution of tasks is very limited. That is, only task release instructions are propagated from the task release automata to the scheduler and queue. After a task is released, there are no means to follow up on the status of the task as the time is progressing.

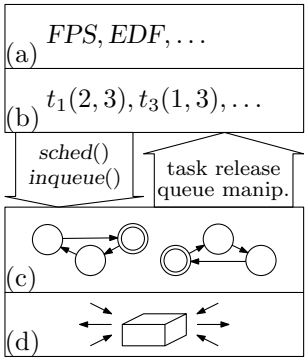


Fig. 2. Essential components of the ATA model: (a) the task scheduling policy, (b) the task queue, (c) task automata network modeling task release patterns, and (d) model of the environment.

Our contribution is the adaptive task automata framework designed for modeling and verification of adaptive embedded systems where it is possible to gather data from the queue and scheduler via a set of predicates, with the potential of enabling alterations of the queue. A visual overview of the structure of ATA is given in Figure 2.

The predicates correspond to schedulability of one or many of the tasks in the queue. In other words, we can find out whether the schedulability of the entire system has been already compromised and even test if it will be compromised if another task would be admitted added to the queue.

The full formal description of the work can be found in our recent paper [1].

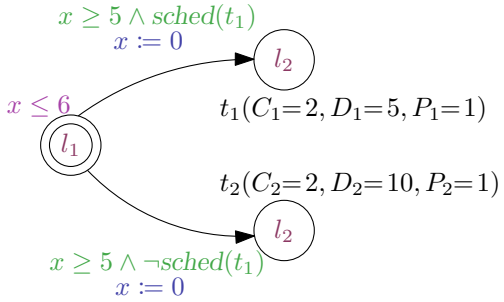


Fig. 3. An adaptive task automaton snippet

In Figure 3, we present an adaptive task automaton similar to the task automaton from Figure 1. Now we have extended it with the schedulability predicate *sched*/1 and introduced another edge and a task that can be released in case that the original task would not have completed before its deadline. The adaptive task automata framework offers few predicates for modifying task release patterns based on the state of the queue:

- *sched*/1 predicate is evaluating whether the task t_1 can be released so that it has a chance of completing in time. We say a chance, since there is still possiblity of releasing another, higher priority task that will preempt this task and render it unschedulable.
  Another functionality of the predicate *sched*/1 is to evaluate whether an already released task can complete in time.
- *inqueue*/1 predicate can be used to find out whether the task is present in the queue or not. This predicate evaluates to true if the task is present in the queue or currently executing on the CPU.
- *sched*/2, a more advanced version of the predicate *sched*/1, is used in conjunction with two parameters: *sched*(t_1, t_2). The predicate evaluates whether the task t_1 can complete in time if the task t_2 would be released now, assuming that the predicate *inqueue*(t_1) holds.

By using the above presented predicates, it is possible to create a temporary inversion of priorities. A task of higher priority can wait before being released to ensure that a task of lower priority completes in time. While modeling a system in our framework, one has to carefully consider whether such behavior is wanted in the system.

Besides the basic predicates: *sched*/1, *sched*/2, and *inqueue*/1, we provide two additional, derived predicates:

- *sched_all*/0 evaluates whether all of the tasks in the current queue are going to meet their respective deadlines;
- *sched_all*/1 evaluates whether the tasks in the queue will meet their deadlines provided that a new task is introduced into the queue.

While implementing the predicates for ATA framework in timed automata, we have encountered many issues related the decidability of the schedulability analysis. One of the most noticeable issues was that schedulability testing predicates rely on testing whether the difference between two clocks is less than a certain constant. This causes the entire system to be categorized as diagonally constrained timed automata, which have been proven to be decidable under the same conditions as diagonal-free timed automata [9].

IV. Example

A. Robot teleoperation

As an example, let us look at a model of a hypothetical system for teleoperation of a robot. This particular robot is equipped with a video camera that sends the image to the user and a microphone that transmits surrounding sound to the human operator. The human operator interfaces with the robot via a user console. We can think of a user console as a self contained, battery powered computer running a single core CPU.

The console provides live feed of what the robot *sees* and *hears* while transmitting operator's commands back to the robot. To keep the real-time requirements, processing loop was created which is executed every 100 time units. The processing loops first scans whether any commands from the operator have been received, acts upon them, then processes incoming video frame and audio packet and displays them to the operator.

One of the design goals of the system would be to extend battery life of the user console while maintaining hard real-time requirements on its functionality. To achieve that, a CPU with three clock frequency scaling modes was built into the console. The full power mode provides maximum performance while sacrificing battery life. The extended life mode provides more battery life with a small, 20%, degradation of the performance. The third mode is most suitable for handling low battery situations since it significantly degrades CPU performance to only half of the full performance.

The designers have chosen to keep audio and input processing at a constant performance and have built the adaptivity feature into the video processing task by providing three different versions of the task that can be released based on the available resources. These versions are shown in the table 4 along with all the other tasks present in the system.

On the other hand, if the audio is not critical to the operator, operator can choose to mute it, thus freeing up the part of the processing cycle otherwise taken up by the audio processing. This enables the system to schedule larger quality task for video.

To model the reduced processing capacity, we are releasing a high priority tasks (t_{int1} or t_{int2}) at the start of each processing cycle. Their computation times correspond to the missing capacity.

A model of such system in the ATA framework consists of:

- a task release automaton that models the release pattern of the input task;
- a task release automaton that models the release pattern of the audio task, while providing audio muting features;
- an automaton modeling user interaction that mutes the audio;
- an automaton modeling the reduction in battery levels that causes reduction of available CPU resources;
- a periodic release automaton modeling different levels of interference corresponding to the reduction of resources;
- and an adaptive automaton modeling the task release pattern of the video task.

The large number of automata is due to wish to make distinction between different tasks and to model external events as separate automata. This can be reduced all the way to a single task release automaton, while sacrificing simplicity of the individual automata.

	P	T	D	C	Description
t_{int1}	7	100	100	20	High interference
t_{int2}	6	100	100	50	Low interference
t_{input}	5	100	100	10	Input processing
t_{audio}	4	100	100	20	Audio processing
t_{video}	3	100	100	70	High quality video
t'_{video}	2	–	100	40	Medium quality video
t''_{video}	1	–	100	20	Low quality video

Fig. 4. Tasks present in the user console for teleoperation of the robot.

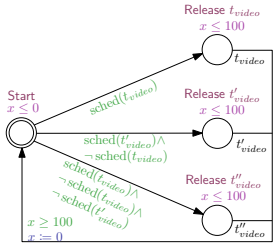


Fig. 5. Adaptive task automaton model for the user console.

In Figure 5 the adaptive automaton for the video task is presented. The automaton tests whether the best priority task can be released first and then downgrades the quality of the video task progressively until one task can be released. In general case, this automaton may deadlock due to the lack of an edge that would be taken if no variant of video task fits into the task set, but by verifying the entirety of the system, it is possible to deduce that the entire system never deadlocks.

A problem of inaccurate measurement of schedulability might present itself in the case when multiple task releases happen in zero time. This is addressed by defining the order in which tasks are admitted to the queue, ensuring that the adaptable task is always admitted last.

We can see that the worst case happens when the operator wishes to use the audio, while the battery is at its lowest setting.

In that case, the video task t''_{video} is released and the video quality is low. In all the other cases, the system automatically detects the possibility to release higher quality video task and it does so.

B. Scalability of the approach

Another example involves the usage of the *sched_all* predicate. We would like to create a scalable scheduling automaton that would enable us to schedule job t_i and its n fallback variants with the fallback sequence $t_i \rightarrow t_{i+1} \rightarrow \ldots \rightarrow t_{i+n}$. This can be accomplished easily in our framework by creating a nondeterministic automaton that can transition into the job release state if the job can be scheduled. Then, by introducing determinism via the addition of the edge priorities [10] we can encode the fallback sequence into the automaton.

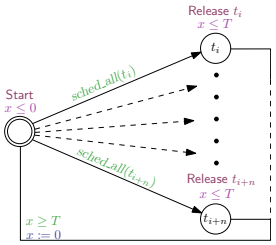


Fig. 6. A universal automaton for scheduling of task with n fallback tasks.

V. Related Work

Schedulability analysis and formal verification of adaptive embedded system models specified in high level languages has recently received increased attention. For instance, several approaches on verifying adaptive embedded systems specified as UML Statecharts are presented by Schaefer [11]. Schneider et al. [12] have proposed a method to describe and analyze adaptation behavior in embedded systems in which the data flow is augmented with quality descriptions that are used by configuration rules to determine potential adaptations. The application of schedulability verification has already targeted multiprocessor systems [13], or satellite systems [14], and results on generalized frameworks for schedulability analysis have also been provided [15]. However, in these studies the non-schedulability of the system cannot be predicted soon enough such that the system does not reach such a state, but only after a task misses its deadline.

VI. Ongoing and Future Work

At the time of writing this paper, we have established decidability of verifying reachability in uniprocessor ATA with fixed priority scheduling and *sched* predicate. We are looking at the boundaries of applicability of our framework and which scheduling policies can be adapted to work within our framework considering that the *sched* predicate requires significant support from the automaton implementing the scheduling policy.

We are also planning to add several new functions that would dynamically alter the queue of the running system, thus simulating forceful termination of the tasks. As well as ways of keeping the system running even in the case when the tasks break their deadlines.

Acknowledgment

This research was supported by the Swedish Research Council, which is gratefully acknowledged.

References

[1] L. Hatvani, P. Pettersson, and C. Seceleanu, "Adaptive task automata: A framework for verifying adaptive embedded systems," in *FASE'12: Proceedings of the 15th International Conference on Fundamental Approaches to Software Engineering*, ser. LNCS, J. de Lara and A. Zisman, Eds., 2012, pp. 115–129. [Online]. Available: http://www.mrtc.mdh.se/index.php?choice=publications&id=2743

[2] C. Norström, A. Wall, and W. Yi, "Timed automata as task models for event-driven systems," in *Real-Time Computing Systems and Applications, 1999. RTCSA '99. Sixth International Conference on*, 1999, pp. 182 –189.

[3] E. Fersman, L. Mokrushin, P. Pettersson, and W. Yi, "Schedulability analysis of fixed-priority systems using timed automata," *Theor. Comput. Sci.*, vol. 354, pp. 301–317, March 2006.

[4] E. Fersman, P. Krcal, P. Pettersson, and W. Yi, "Task automata: Schedulability, decidability and undecidability," *Information and Computation*, vol. 205, no. 8, pp. 1149 – 1172, 2007.

[5] E. Fersman, P. Pettersson, and W. Yi, "Timed automata with asynchronous processes: Schedulability and decidability," in *In Proceedings of TACAS 2002*. Springer-Verlag, 2002, pp. 67–82.

[6] K. G. Larsen, P. Pettersson, and W. Yi, "Uppaal in a Nutshell," *Int. Journal on Software Tools for Technology Transfer*, vol. 1, no. 1–2, pp. 134–152, Oct. 1997.

[7] M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, and S. Yovine, "Kronos: A model-checking tool for real-time systems," in *Computer Aided Verification*, ser. Lecture Notes in Computer Science, A. Hu and M. Vardi, Eds. Springer Berlin / Heidelberg, 1998, vol. 1427, pp. 546–550, 10.1007/BFb0028779. [Online]. Available: http://dx.doi.org/10.1007/BFb0028779

[8] T. Amnell, E. Fersman, L. Mokrushin, P. Pettersson, and W. Yi, "Times: a tool for schedulability analysis and code generation of real-time systems," in *Proc. of International Workshop on Formal Modeling and Analysis of Timed Systems*, ser. Lecture Notes in Computer Science. Springer-Verlag, 2003.

[9] B. Bérard, A. Petit, V. Diekert, and P. Gastin, "Characterization of the expressive power of silent transitions in timed automata," *Fundam. Inf.*, vol. 36, no. 2-3, pp. 145–182, Nov. 1998. [Online]. Available: http://dl.acm.org/citation.cfm?id=305052.305055

[10] A. David, J. Håkansson, K. Larsen, and P. Pettersson, "Model checking timed automata with priorities using dbm subtraction," in *Formal Modeling and Analysis of Timed Systems*, ser. Lecture Notes in Computer Science, E. Asarin and P. Bouyer, Eds. Springer Berlin / Heidelberg, 2006, vol. 4202, pp. 128–142.

[11] I. Schaefer, "Integrating formal verification into the model-based development of adaptive embedded systems," Ph.D. dissertation, TU Kaiserslautern, Kaiserslautern, Germany, Oct. 2008, iSBN 978-3-89963-862-2.

[12] K. Schneider, T. Schuele, and M. Trapp, "Verifying the adaptation behavior of embedded systems," in *Proceedings of the 2006 international workshop on Self-adaptation and self-managing systems*, ser. SEAMS '06. New York, NY, USA: ACM, 2006, pp. 16–22. [Online]. Available: http://doi.acm.org/10.1145/1137677.1137681

[13] F. Yu, G. Li, and N. Xiong, "Schedulability analysis of multi-processor real-time systems using uppaal," in *Information Science and Engineering (ICISE), 2010 2nd International Conference on*, dec. 2010, pp. 1 –6.

[14] M. Mikučionis, K. Larsen, J. Rasmussen, B. Nielsen, A. Skou, S. Palm, J. Pedersen, and P. Hougaard, "Schedulability analysis using uppaal: Herschel-planck case study," in *Leveraging Applications of Formal Methods, Verification, and Validation*, ser. Lecture Notes in Computer Science, T. Margaria and B. Steffen, Eds. Springer Berlin / Heidelberg, 2010, vol. 6416, pp. 175–190.

[15] A. David, J. Illum, K. Larsen, and A. Skou, *Model-Based Framework for Schedulability Analysis Using UPPAAL 4.1*. CRC Press, 2011/12/27 2009.

Provisioning within a WSAN Cloud Concept

Muhammad Sohaib Aslam, Susan Rea and Dirk Pesch
Nimbus Centre For Embedded System Research
Cork Institute of Technology, Ireland
Email: {muhammad.aslam, susan.rea, dirk.pesch}@cit.ie

***Abstract*—Traditionally, Wireless Sensor and Actuator Networks (WSANs) have been used as a standalone technology for a specific application purpose such as heating control. The current growth in embedded ICT infrastructure, driven by visions such as the smart cities idea, is leading to the deployment of a wide range of embedded systems in our environment, which motivates the *System of Systems* [1] and ultimately, with deployment of IP technologies into this space, the *Internet of Things* [2] paradigm. However, to simplify system operation and maintenance as well as to reduce costs, WSANs must become an infrastructure that is capable of providing services to multiple end users concurrently rather than requiring a new infrastructure for a new purpose. Here, we present the concept of a WSAN infrastructure as a *WSAN Cloud*, which provides services to multiple application and data collection systems following to some extend the cloud computing paradigm. Each instance of the WSAN cloud (i.e. a specific set of services configured by a particular end user/system) utilises the WSAN infrastructure as if it was a unique network provisioned for specific requirements. This realisation of the WSAN Cloud as *Network as a Service* or *NaaS* requires the WSAN to support a service orientated software architecture allowing other systems to provision the WSAN infrastructure for their specific needs and allowing multiple systems to use the WSAN uniquely and concurrently. The WSAN-Service Orchestration Architecture "*WSAN-SOrA*" presented here, is a novel approach to service provisioning of embedded networked systems and enables WSANs to act as cloud ready infrastructures that facilitate on-demand provisioning for potentially multiple individual backend systems.**

I. Introduction

Smart Cities [3], [4] is a vision of cities offering novel services based on a digital integration of city infrastructures through computing systems enabling on-demand service delivery. One of the technologies which future smart cites will rely on is Wireless Sensor and Actuator Networks (WSAN). WSANs are a rapidly evolving technology but in their current form will not be able to fully support the *Smart Cities* vision due to the cost associated with equipment, deployment, and operations and maintenance of such an extended embedded systems infrastructure. One way of reducing cost is for WSANs to be able to make its infrastructure available on-demand simultaneously to multiple users. In the context of this paper we view these users as enterprise tier systems based on cloud computing concepts that aim at virtualising the underlying hardware infrastructure down to the WSAN tier.

Cloud computing [5], [6] is based around the concept of delivering services to users. Cloud computing is particularly important for WSANs in terms of broadening their scope. Where Smart Cities are concerned, cloud computing enables a WSAN infrastructure, an essential technology in Smart Cities, to be delivered as a service and in this paper we refer to the virtualization of WSAN infrastructure as a WSAN Cloud. Cloud computing follows SPI (Software, Platform and Infrastructure) model of service delivery. In the context of this paper, we focus on the IaaS (Infrastructure as a Service) concept for delivering the WSAN Infrastructure as a service to the end user and providing a mechanism to provision the infrastructure. IaaS encompasses three domains i.e. "Compute, Storage and Network", and here we focus on the Network as a Service or NaaS aspect. In order to evolve a WSAN into a cloud infrastructure for NaaS, all tiers of a WSAN must support the functions associated with the *Service oriented Architecture (SoA)* [7] principle. We divide the WSAN cloud infrastructure into three tiers.

- **(tier 1)** Node Network: This tier consists of a network of largely wireless embedded devices which are capable of sensing and actuation. These devices are envisioned to be based on IPv6/6LowPAN technology with wireless network interfaces, such as IEEE802.15.4, to communicate with the Gateway tier.
- **(tier 2)** Gateway: This is the middle tier connecting the node network to the backend system. This tier has enhanced computation capabilities and software services in line with backend systems, with interfaces to the Back-end/Enterprise Core tier being RPC, web services or sockets.
- **(tier 3)** Back-End/Enterprise Core: This is the main and computationally most powerful tier, usually running on a server suite. The core provides a platform for implementing components such as management frameworks and end-points for other users/systems that require data or interfacing with the WSAN domain.

The SoA principle is well established at the Enterprise and Gateway tier, however it is a relatively new concept for devices in the embedded Node Network tier. Traditionally, embedded wireless sensor/actuator devices have had low computational power not capable of supporting SoA concepts. However, recently these devices have become powerful enough to support SoA [8] [9] along with the required underlying operating systems such as SOS [10] , Lorien [11] and Squawk [12] to provide necessary platform to implement SoA principles at the embedded Node Network tier.

In this paper we introduce the concept of the WSAN Cloud. This cloud is an organisational domain to which

other organisation/enterpise systems connect and provision the infrastructure to deliver services using the NaaS paradigm. In order for the WSAN infrastructure to support delivery of NaaS and act as a cloud infrastructure, the ability to support SoA is required at all tiers of the infrastructure. Provisioning a SoA infrastructure where there are a large number of devices at the Node Network tier is a complex problem being faced by high-end networks as well. As manual provisioning greatly increases the likelihood of errors, automated processes are required. Provisioning becomes even more complex in cloud infrastructures where a single physical infrastructure is expected to be provisioned a number of times for concurrent usage. In this paper we present an orchestration architecture for automatic provisioning of the proposed WSAN Cloud.

II. Provisioning For NaaS

A. Provisioning

Provisioning is an ambiguous term which is widely used in networking. For example [13], [14] use the term provisioning to define the configuration of a sensor network application. Provisioning in the context of this paper refers to service provisioning in a network to support the concept of NaaS. Although the focus of our research presented here considers the WSAN Cloud within building management, the software architecture proposed is not restricted to buildings but can be used for a wide range of smart infrastructure. As delivery of NaaS requires provisioning, we must support provisioning at the following tiers.

- Node Network Tier: Provisioning device services.
- Gateway Tier: Provisioning service to open end-points and data processing logic.
- Enterprise Core Tier: Provisioning of high-end services to manage segments of the cloud infrastructure for each end user.

B. Delivering NaaS

NaaS is required to provide reusability of the WSAN in order to maximise resource utilisation where resources here are the WSAN hardware i.e. (network devices at the Node Network tier). The user of a NaaS can request a service based on certain requirements e.g. a new Subnet or Humidity readings in a specific WSAN zone. In order to facilitate user requirements the WSAN Cloud needs to provision services, in particular on the embedded devices such as a sensing service or a routing services in a similar fashion as in a local area network where VLAN, OSPF or trunk-port services are provisioned. In a network with hundreds of devices, however, a requirement for the following is needed:

1) Expedite the process of provisioning services on devices.
2) Reduce the need for the human in the loop to mitigate erroneous output.
3) Reusability of provisioning information for other networks
4) A rapid provisioning engine.

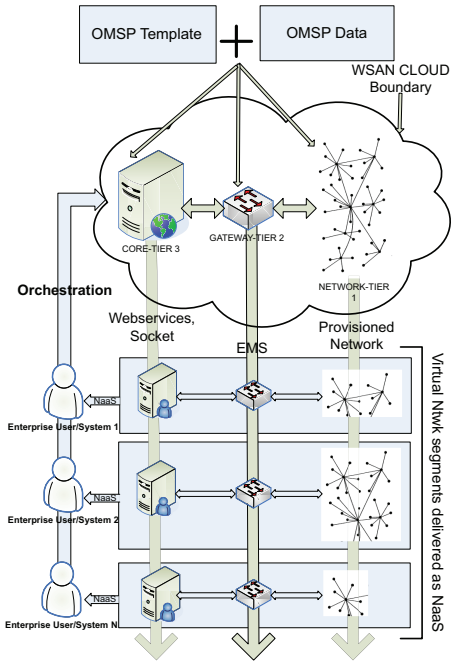
Using automation the provisioning process can be expedited and errors can be reduced. Quantifying the benefits of automated service provisioning can be viewed in terms of time saved, no requirement for expert personnel and reduced labour costs. However, in order to understand how automation expedites the process of provisioning services and reduces the errors associated with the human in the loop let us consider an example of provisioning a WSAN. In order to provision a service in a WSAN we need people with expert knowledge of this type of network. A console is used to access the network and the devices so that they can be configured based on the end-user requirements. Furthermore in addition to this being a tedious and time consuming task, human intervention may also cause error as the experts have to go through complex configuration parameters manually. Such a manual process demands manpower and time, and this can prove costly for large deployments. Rapid provisioning allows an automated provisioning process to remedy all the forementioned disadvantages. The rapid provisioning concept is extensively applied in the Compute and Storage Cloud domain, i.e. VMware is an example of a tool for rapid provisioning. However, the network cloud component is still problematic where custom provisioning is still an issue and is true for both wired and wireless networks. The custom provisioning process is long and error prone mainly due to manual service provisioning and the requirement for network experts. WSANs have been traditionally seen as an isolated, specialised domain, where the application of cloud computing and SoA are relatively novel concepts. In order for a WSAN to be cloud ready the following properties are necessary:

- Need to expose the WSAN as a NaaS providing cloud infrastructure.
- The WSAN can be used by multiple systems with different requirements.
- Automated and rapid provisioning.

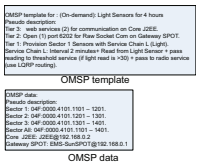
III. Service Orchestration Architecture

The WSAN Service Orchestration Architecture (WSAN-SOrA) as shown in Figure 1a is the architecture we propose and have developed for cloud ready WSANs to deliver NaaS and to automate service provisioning and de-provisioning in the WSAN Cloud. The key attribute of this architecture is to provide an orchestration engine for provisioning services at all tiers of the WSAN Cloud. The ability to support service provisioning in the WSAN by multiple end user systems supports reusability of the WSAN in form of NaaS.

The *WSAN-SOrA* provides a comprehensive automation system for provisioning and de-provisioning, by introducing the concept of model based orchestration. Orchestration is a process which enables the WSAN Cloud to be provisioned or de-provisioned rapidly, driven by *Orchestration Model for Service Provisioning (OMSP)* shown in Figure 1b. Orchestration allows the rapid provisioning of services in network devices, gateway and core to fulfill end-user requirements. In a traditional single tier WSAN network, devices are configured either using a terminal to the device i.e. ”command line”



(a) Provisioning Overview



(b) OMSP Model

Fig. 1. WSAN SOrA

or using some sort of web interface and in most case are hard-coded. The reason for such practice lies in the fact that WSANs were never expected to offer cloud services, that is providing services for multiple end users. A static interface has been sufficient so far. Similarly, in high-end networks there is a similar situation as most of the network devices are provisioned using terminals and static interfaces.

A. Model Driven Orchestration

Models have been used for a number of purposes from simulating a system to using models as a means of understanding how a system functions. Rather than using mathematical model, an OMSP is instead based on simple system descriptive models that are used as drivers for orchestrating the rapid provisioning of a WSAN cloud. OMSP are a generic model for service provisioning, which can be written using XML. The OMSP can be divided into two segments - the OMSP template and the OMSP data. An OMSP template contains a pointer to the network devices and a service description of the services required to be provisioned. The OMSP data contains the device data such as the device address corresponding to the pointer in the OMSP template. An OMSP template provides a reusability mechanism for provisioning as it can be reused for provisioning on number of different networks. The OMSP data is related to the network itself and contains network dependent data. For example Figure 1b shows a pseudo description of an OMSP for reading light from sector 1 devices for a limited time. The OMSP template contains logic which can be applied to a number of different WSAN cloud infrastructures, where the OMSP data contains the data pertaining to the devices in all tiers of a specific WSAN cloud.

B. Orchestration Engine

The Orchestration Engine is a model (OMSP) driven system which uses the OMSP for provisioning the required services on the devices in the network (atomic service provisioning). The Orchestration engine consists of following components:

- Core
- Element Management System (EMS)
- NaaS Endpoint or middleware

1) Core (Back-End): The core of the orchestration engine is the CPU of the engine. It contains the translating components to read the OMSP. The core also configures the service end points for integration between NaaS and the end-users.

2) Element Management System (Gateway): Once the OMSP is translated, the consolidated data is sent to the EMS. Consolidated data refers to the combination of the data from the OMSP and the knowledge base (this is a collection of meaningful data about the network and is described further in section (III-C) in the orchestration engine to enable the EMS to execute the operations necessary to complete the service provisioning tasks. This is the entry point to the WSAN Cloud. Each EMS manages a specific set of devices i.e. an EMS for SunSPOTs, an EMS for TelosB, etc. The EMS is an overlaying system that manages the gateways where gateways are systems running on physical computing devices, e.g. embedded wireless devices, specialised embedded PC boards, etc., that are capable of communicating with tier 1 devices. The EMS calls the gateway device to disseminate the service provisioning data to the network devices in the WSAN Cloud. The data for service provisioning is simply a few bytes as it does not contain system script or code, rather just the service id and the parameters of the service to be instantiated. While a full service upgrade such as updating functionality is possible, the *WSAN-SOrA* does not recommend it due to the fact that devices in the WSAN run with limited energy resources. A Service upgrade can be compared to a full or part firmware upgrade in a high-end network. Similarly, a WSAN service upgrade can be used to modify the implementation of the service, however such an operation is risk prone as this means the devices are unavailable while the update is ongoing and in battery powered wireless networks this consumes a considerable amount of battery power. Furthermore even in high-level wired systems it is not a common practice as it causes disruption to the network service while a service upgrade is in progress.

3) NaaS Endpoint (middleware): This component is an output product of the orchestration engine. Once the orchestration engine executes the OMSP for an end-user, a service endpoint is provided usually in the form of a web-service. The data from

the network provisioned for a specific end-user is provided to the end-user through this service point.

C. Orchestration Flow

The design of an OMSP in the process of provisioning or de-provisioning is the first stage. As the OMSP is passed to the orchestration engine, the rest of the process becomes automated. The OMSP is first decoded and translated using the syntax knowledge base. If new syntax is required this knowledge base acts as a repository for developers to include new syntax. The knowledge base is a comprehensive database for the orchestration engine. It contains information such as data related to the OMSP syntax. With the OMSP being usually written in XML, this format acts as a standardised way of representing a document where the tags and attributes in the documents are meaningless unless there is a reference document. This data relates to the translation that is stored in the knowledge base. The knowledge base also stores the network data coming from devices and contains data pertaining to the available services for each device in the network and the current instance of those services. Service descriptions are written and saved in the knowledge base using XML based SDL (Service Descriptor Language) document. In summary, the knowledge base is a collection of meaningful data about the network and is used to support orchestration and other operations. Once each service is identified and devices are selected, a secondary document is created by the orchestration engine, which contains instructions for the execution part of the orchestration engine. The EMS manager reads this document and calls the appropriate EMS gateway depending on the hardware platform. These gateways contain the base station device capable of communicating with network devices of the same type in the WSAN Cloud. The instruction is then disseminated in the WSAN Cloud and the services in the network are provisioned. Dissemination in the WSAN Cloud is based on the configured wireless communication interfaces and protocols such as IEEE 802.15.4/6LowPAN. The orchestration engine needs to create an end-point (middleware) for the newly provisioned network so that the requesting system may extract its data. This can be done in the form of setting a webservice (passive polling), raw socket (stream), or servlets (push). De-provisioning is similar to provisioning, but services are de-instantiated instead of being created.

IV. Current Development Status

A prototype has been implemented on a testbed of 30 SunSPOTs running the java based Squawk operating system in our research building. Our sample use case assumed that there are 3 users of the WSAN Cloud. User A requires the light and temperature data of the building after every 30 seconds, User B requires the light data from the seminar room every 10 seconds and User C requires the humidity data every 2 minutes. All the user uses the network simultaneously and the provisioned segment of infrastructure they use overlapped with each others. The core (tier 3) was supported by the open source J2EE server Glassfish. The Core consist of technologies such as java enterprise beans, java messaging and webservice management supporting tools. The core connects directly to the gateways. The Gateway uses the java standard edition (J2SE) for supporting the EMS for SunSPOTs, for other type of devices the EMS is written in C, for example we have developed an EMS for the Lorien operating system running on the TelosB mote platform. In our experiment we used three SunSPOTs as gateway devices at different locations to manage different sectors of the embedded wireless network. The Network tier devices are deployed with our SoA based system software. Each service is regarded as a separate process with a dedicated thread provisioned for a specific or unlimited time. We conducted a number of experiments orchestrating NaaS requests to demonstrate the feasibility of the architecture. The average time taken by the orchestrator to provision a device is on average 1200 milliseconds, which is a subjective measure and is dependent on the programming capabilities of the developer. QoS for data reliability is at present focused on device lifetime management.

References

[1] C. Dagli and N. Kilicay, "Understanding behavior of system of systems through computational intelligence techniques," in *Systems Conference, 2007 1st Annual IEEE*, april 2007, pp. 1 –7.

[2] Y. Lu, N. Huansheng, Y. Laurence T., and Y. Zhang, *THE INTERNET OF THINGS: From RFID to the Next-Generation Pervasive Networked Systems*. Auerbach Publications, 2008.

[3] H. Waer and M. Deakin, *From Intelligent to Smart Cities*. Taylor & Francis, 2012.

[4] M. Kehoe *et al.*, *Smarter Cities Series: A Foundation for Understanding IBM Smarter Cities*. An IBM Redguide publication, 2011.

[5] G. Reese, *Cloud Application Architectures: Building Applications and Infrastructure in the Cloud*. OReilly, 2009.

[6] D. S. Linthicum, *Cloud Computing and SOA Convergence in Your Enterprise: A Step-by-Step Guide*. Addison-Wesley Professional, 2009.

[7] M. Bell, *Service-Oriented Modeling (SOA): Service Analysis, Design, and Architecture*. Wiley, 2008.

[8] E. Avils-Lpez and J. A. Garca-Macas, "Tinysoa: a service-oriented architecture for wireless sensor networks," *Service Oriented Computing and Applications*, vol. 3, no. 2, pp. 99–108, 2009.

[9] E. Meshkova, J. Riihijarvi, F. Oldewurtel, C. Jardak, and P. Mahonen, "Service-oriented design methodology for wireless sensor networks: A view through case studies," in *Sensor Networks, Ubiquitous and Trustworthy Computing, 2008. SUTC '08. IEEE International Conference on*, june 2008, pp. 146 –153.

[10] C. chieh Han, R. Kumar, R. Shea, E. Kohler, and M. Srivastava, "Sos: A dynamic operating system for sensor networks," in *Proceedings of the Third International Conference on Mobile Systems, Applications, And Services (Mobisys*. ACM Press, 2005.

[11] B. Porter and G. Coulson, "Lorien: a pure dynamic component-based operating system for wireless sensor networks," in *Proceedings of the 4th International Workshop on Middleware Tools, Services and Run-Time Support for Sensor Networks*, ser. MidSens '09. New York, NY, USA: ACM, 2009, pp. 7–12.

[12] "Squawk for sunspot," Online, Jan. 2012. [Online]. Available: http://labs.oracle.com/projects/squawk/squawk-sunspot.html

[13] S. J. Habib, "Analysis of sensors coverage through application-specific wsn provisioning tool," *IJMCMC*, vol. 3, no. 1, pp. 51–62, 2011.

[14] J. Yick, B. Mukherjee, and D. Ghosal, "Wireless sensor network survey," *Comput. Netw.*, vol. 52, pp. 2292–2330, August 2008.

A Virtualization-based Fault Resilient Secured Real-Time System Architecture

Daeyoung Hong, Wonseok Ko, Sung-Soo Lim
School of Computer Science,
Kookmin University,
Seoul, Korea
{vicente, magicyaba, sslim}@kookmin.ac.kr

Consolidating multiple COTS (Commercial-Off-The-Shelf) real-time control systems into an integrated high performance system guaranteeing the original functional and timing constraints has been and will be a crucial issue in diverse mission critical applications (Figure 1). Though related industry shows significantly increased needs for solutions in COTS integration, there are still a number of problems to be solved necessarily in realizing the solutions. One of the most important issues would be how we could integrate the existing real-time control tasks running on low-end microprocessors into one single system without changing the original tasks, but keeping the required functions and requirements.

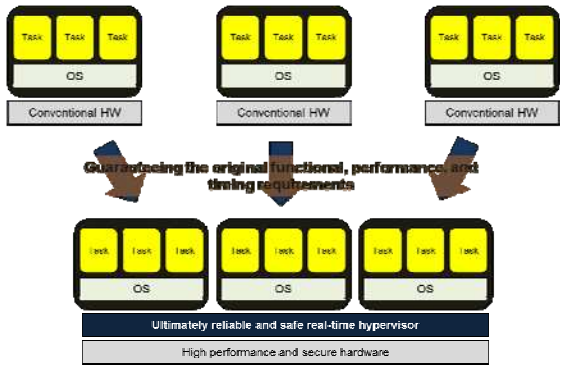


Figure 1 COTS integration in real-time control systems

A number of players including vendors and research groups have been proposing virtualization technique for a realizable solution of COTS integration issues. However, the pure virtualization techniques used for server consolidation would not be appropriate for our purposes since real-time embedded systems should satisfy another category of requirements besides functional consolidation.

We propose a virtualization-based fault resilient real-time control system architecture providing COTS integration, security among VMs, and reliability. The proposed system architecture consists of secured multi-core SoC architecture and virtualization-based software hierarchy. The core features of the proposed virtualization technique are summarized in the following: Minimized TCB (Trusted Computing Base) through microkernel-based light weight hypervisor, Predictable components in hypervisor including IPCs, exception handling, memory management, and housekeeping for VMs, and Runtime monitoring, fault detection, recovery, and upgrading guaranteeing real-time constraints.

Among the key features our hypervisor should provide, the runtime monitoring, fault detection, recovery, and upgrading features necessitate well-defined system components performing control of VMs under privileged mode. We have chosen SIMPLEX model proposed by University of Illinois at Urbana-Champaign and modified the architecture to be realized in our virtualized system architecture. The revised model is named vSIMPLEX.

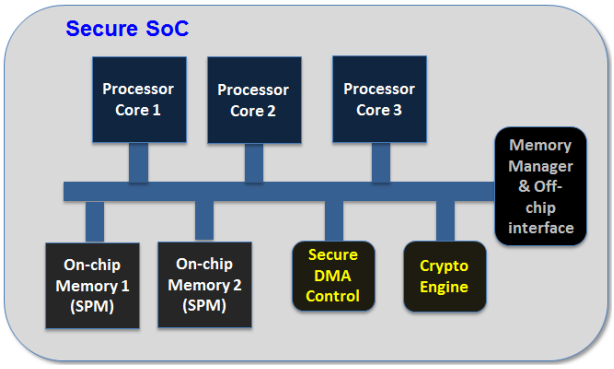


Figure 2 Secured SoC design with on-chip SPMs

Along with the virtualization-based software architecture, our proposed system includes secured multi-core SoC design to support guaranteed reliability and security (Figure 2). Our proposed SoC provides secured memory transfer between on-chip secured memory and off-chip memory, confidentiality through on-chip encryption engine, and predictable memory management by using scratchpad memory. Such features necessitate management policy and solutions in hypervisor layer to provide sufficient transparency and security to upper layer guest VMs and applications. Besides the features mentioned above, our proposed SoC includes second level MMU which supports hypervisor-level memory management which is one of the key features for hardware-level virtualization support in new ARM Cortex A15 or A7 processors.

The proposed architecture has been evaluated using simulation based on SystemC and reference platform based on ARM Cortex cores.

Adaptive Trajectory Coordination for Scalable Robot Motion Planning

Hoon Sung Chwa, Andrii Shyshkalov, Jinkyu Lee, Hyoungbu Back, and Kilho Lee
Dept. of Computer Science, KAIST, Republic of Korea
chwahs@cps.kaist.ac.kr

I. Introduction

This paper aims to address some of the common CPS requirements in the robotics domain. More specifically, it introduces an "adaptive" optimization framework that supports a new, user-centric perspective of "responsiveness" in a "scalable" manner, in the context of trajectory coordination for multi-mobile-robot control. As an example of multi-mobile-robot control, we consider multiple robot display [1], where multiple robots shape a formation to collectively visualize a sequence of images that users give. From the viewpoint of responsiveness, users would perceive a higher quality of service (QoS) it were faster or more stable to complete robots' visualization from the user input. The response time of multiple robot display consists of two parts: the computation time for assigning individual robot tasks (i.e., path planning, trajectory coordination) and the actuation time for performing individual tasks (i.e., robot moving for collective visualization). In order to improve the responsiveness, it is necessary to reduce the response time, that is, both the computation time and the actuation time. However, it is often complicated to reduce the response time since reducing computation time and improving computation results (i.e., improving actuation time in this case) are conflicting objectives in many cases.

In particular, *trajectory coordination*, which determines the trajectory of individual robots without collision for given paths, is a key problem that significantly affects the response time since it aims to minimize the actuation time (i.e., minimizing the maximum robot traveling time). Finding an optimal solution to trajectory coordination is generally NP-hard. Hence, it is necessary to develop approximation algorithms that can explore a tradeoff between complexity (reducing computation time) and performance (reducing traveling time) effectively for sub-optimal solutions.

As a motivational example, Figure 1 represents such a tradeoff produced by an approximation algorithm. This algorithm performs iterative steps to improve performance. It is shown in the figure that a growing number of iteration decreases actuation time but increases computation time. Here, an optimal point, which minimizes the response time, can be found around 60 iterations. Finding such an optimal point in advance is generally difficult because such a tradeoff is quite unpredictable. The maximum robot traveling time greatly depends on various environment parameters (i.e., the number of robots, robot kinematics, robot deployment), and the influences of such parameters on the traveling time are sophisticated. Therefore, it is difficult to estimate the robot traveling time for some input parameters and predict how computation can affect traveling time.

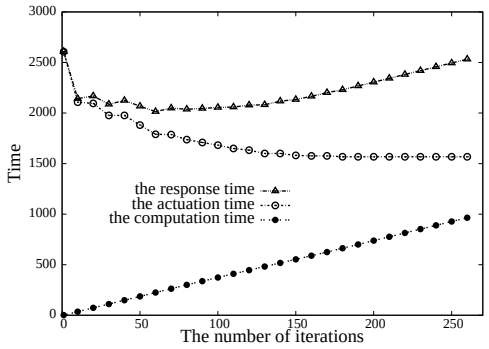


Figure 1. A tradeoff between the actuation time and the computation time according to the number of iterations

This paper introduces an adaptive framework that estimates such a tradeoff dynamically and exploits it adaptively to minimize the response time. The proposed framework performs priority-based trajectory coordination for scalability. Similar to prioritized planning [2], [3], our framework assigns priorities to robots and determines the collision-free trajectories of individual robots in a decreasing order of priority. More specifically, our framework consists of two components: (1) *adaptive sequence generation* for priority assignment, and (2) *sequence-based trajectory coordination* for collision-free trajectory generation. A sequence is defined as an ordered list of all participating robots, and it determines which robots are given higher priorities in collision resolution. Furthermore, the sequence is also used as a unit of computation in trajectory coordination. Different sequences generally produce different actuation times, and many sequences increase a chance to decrease actuation time at the expense of increasing computation time. As such, sequence influences both actuation time and computation time directly, and it provides a mechanism to balance the tradeoff effectively. The proposed framework is then capable of controlling the tradeoff by generating sequences adaptively.

References

[1] *The Flyfire Project at MIT*. - http://senseable.mit.edu/flyfire/.

[2] M. Erdmann and T. Lozano-Perez, "On multiple moving objects," *Algorithmica*, vol. 2, no. 4, pp. 477–521, 1987.

[3] J. P. van den Berg and M. H. Overmars, "Prioritized motion planning for multiple robots," in *IROS*, 2005.

A Self-adaptive Unifying Mechanism For Autonomous Energy Management In Wireless Sensor Networks

Lina Xu*, Olga Murdoch*, Gregory O'Hare*, Rem Collier*
*CLARITY: Centre for Sensor Web Technologies
University and College of Dublin, Ireland
{lina.xu, olga.murdoch}@ucdconnect.ie {rem.collier, gregory.ohare}@ucd.ie

I. ABSTRACT

Wireless Sensor Network (WSN) deployments require the maintenance of hundreds of battery run sensor motes which may be placed in hard to reach locations. As replacing or charging sensor batteries post deployment is both difficult and expensive, maintaining a WSN poses challenging issues. Power saving mechanisms aim to increase the life time of a WSN. This is achieved through the use of power saving techniques that aim to balance the overall usage within a network. A common approach to power saving is to cluster the network and schedule the working duty cycle between the different clusters [1].

Many solutions in this area are specialized techniques that perform well for specific scenarios [1]. While such techniques offer great advantages they do not generalize well, meaning a custom solution is required for each use case. For example, LEACH is a power saving solution that performs well in scenarios without location information but fails to work for time-asynchronous systems. PECAS however, does not rely on time synchronization but is only designed for uniform placement networks. It is also identified that none of the existing solutions can work for mobile networks.

With the emergence of middleware solutions that aim to support diverse scenarios, there is a need to facilitate a wide range of power saving techniques. As a result, there is a need for a generalized power saving mechanism that will support diverse scenarios while meeting the needs of specific use cases. Middleware requires a generalized power saving mechanism that is designed to be both lightweight and adaptive, supporting the specific needs of many sensor driven systems.

We propose a self-adaptive unifying mechanism for autonomous energy management of WSNs with an aim to its incorporation as a power saving component in the SIXTH middleware [2]. This mechanism aims to provide an easy to use, flexible and self-adaptive clustering and sleeping scheduling technique that produces custom power saving algorithms for diverse systems. The mechanisms comprises:

1) A framework: Light-weight, scalable, adaptive and autonomous framework that includes:

- *Voronoi-like clustering:* A Voronoi diagram is a basic geometric data structure often used in WSNs. Traditional Voronoi tessellation requires geographical coordinates for the grouping of discrete points in an Euclidean space into Voronoi cells. The motes that comprise a WSN however, not always have the ability to detect their geographic location. We propose a variation of Voronoi tessellation that uses Round Trip Time (RTT) to measure the relative distance between sensors in an Euclidean space.
- *Round robin scheduling:* During each round only one sensor per cluster is active and monitoring the target. All other sensors remain in different levels of low energy sleep mode. When a new round begins the active nodes switch to sleep mode and a new active node is selected in each cluster.

2) Extensions: Features from clustering algorithms are abstracted and treated as individual extensions. This novel decoupled approach will allow extensions to run independently, each managing problems such as mobility, radio adaptation, probability activation, etc. Extensions can supplement or replace existing extensions in the mechanism to satisfy use case requirements. For example, the mechanism may choose to change the basic scheduling scheme to an optimized version.

While these contributions will form the basis for a middleware based power saving solution, future research should extend these ideas and provide a robust platform for energy management. For example, an evaluation component will monitor the performance of a generated algorithm allowing the system to either justify or reselect extensions in the algorithm, ensuring improved performance in real time. Also, autonomous selection of suitable extensions based on monitoring of current conditions and use case requirements will form the basis for a self adaptive custom power saving solution.

REFERENCES

[1] L. Wang and Y. Xiao, "A survey of energy-efficient scheduling mechanisms in sensor networks," *Mob. Netw. Appl.*, vol. 11, no. 5, pp. 723–740, Oct. 2006.

[2] G. O'Hare, C. Muldoon, M. O'Grady, R. Collier, O. Murdoch, and D. Carr, "Sensor web interaction," in *International Journal on Artificial Intelligence Tools*, 2012.