Benefits and Implications of the DECOS Encapsulation Approach

Martin Schlager, Erwin Erkinger
TTTech Computertechnik AG
Schoenbrunner Strasse 7, 1040 Vienna, Austria
Telephone: +43 1 5853434
Fax: +43 1 5853434 90
{martin.schlager,erwin.erkinger}@tttech.com

Wilfried Elmenreich, Thomas Losert
Vienna University of Technology, Austria
Institute of Computer Engineering
Treitlstrasse 3, 1040 Vienna, Austria
{wil,tl}@vmars.tuwien.ac.at

Research Report 80/2005

Abstract – *In contrast to federated architectures, an integrated architecture provides means to support mixed-criticality systems, i.e., systems that consist of distributed application parts (subsystems) with different criticality levels, on top of the same physical hardware. A major prerequisite for the integration of subsystems with different criticality levels, is given by a strong and reliable protection of the subsystems against each other - both in space and time. Within DECOS, an Encapsulated Execution Environment is set-up in order to establish the required level of protection by providing a mixture of hardware (e.g., memory protection) and software mechanisms (e.g., real-time operating system).*

The development of an Encapsulated Execution Environment is driven by the enormous advances in the domain of dependable real-time control systems in the past decade and the resulting increase of system size in terms of required hardware components (ECUs). The paper shall give a survey of the resulting benefits of the chosen approach and will investigate on its implications. Thereby, it will examine the need for proper development methods that assist the application developer. For instance, the emulation of a subsystem or its parts within an integrated architecture through a simulation requires new dedicated approaches resulting from the inherently more complex structure of the integrated architecture.

1 Introduction

Trends in the transportation sector entail the need for higher integration of embedded applications. In particular, companies in the automotive domain can no longer afford to increase the number of electronic control units (ECUs) within a car. Today, more then 90 ECUs can be found in German upscale cars [1]. According to [1], a reduction to 10–15 ECUs is expected within the next decade. Furthermore, the improvement of reliability of electronics is an important challenge. Sustainable innovations in the automotive industry are increasingly dependent on more complex electronic and software systems. New safety-relevant control applications such as driver assistance systems or advanced chassis control systems require sophisticated fault tolerance mechanisms in order to reach the required level of dependability. However, according to figures of the German automobile association ADAC, 52% of all breakdowns are directly related to defects of car electronics. Furthermore, decreased geometries, lower power voltages, and higher frequencies have a negative impact on reliability [2].

In order to alleviate current problems, a shift from federated architecture solutions to an integrated architecture is foreseeable in the area of dependable embedded systems [3, 4]. An integrated architecture establishes a framework for the execution of mixed-criticality applications on the same hardware by offering methods for resource protection and diagnosis on an architectural level.

Traditionally, integration of mixed-criticality applications to be executed on the same hardware required all application parts to inherit the highest criticality level that exists within the overall system. In order to reduce development effort, application parts with different criticality levels have usually been implemented as self-contained units with their own processing and input/output systems (federated approach). In contrast, current work on integrated solutions aims at combining the complexity management advantages of the federated approach, but also realizes the functional integration and hardware efficiency benefits of an integrated system [5].

The intention of the European Union Framework Programme 6 integrated project DECOS [6] is to develop the basic enabling technology for such a move from a federated distributed architecture to an integrated distributed architecture. A reduction of hardware, development, validation and maintenance cost, and an increase of dependability of embedded applications in various application domains is expected by the results of DECOS.

2 Aspects of Integration

Integration of mixed-criticality applications that possibly originate from different vendors is beneficial for a number of reasons. Nevertheless, there are several requirements for the set up of an integrated architecture. This section gives a short overview over benefits and implications of integration, state of the art, and key requirements.

2.1 Benefits and Implications of Integration

The integration of several federated systems to be executed on the same hardware platform leads to a *reduction of hardware resources.* It follows that a significant reduction of cost, space, and power consumption can be achieved. Furthermore, the *overall dependability* can be improved because of fewer cables and connectors [7].

Nevertheless, the integrated architecture approach imposes several difficulties. Integrating several federated systems (i. e., systems with different criticality levels and from different vendors) into an integrated architecture system adds complexity with respect to design comprehension and requires proper *design methods.* Moreover, the functional units of the previously federated application parts have to be encapsulated in order to avoid malicious mutual interference and to trace back responsibility to a particular vendor in case of an error.

Property	Pre-crash system	Engine control system	Door lock system
Dependability	Ultra-high	High	Low
Architecture	Distributed redundant	Local	Distributed complementary
Timing	Guaranteed with fast response time ($<$ 1ms)	Periodical (ca. 1ms)	Sporadically with response time ≈ 0.5 sec

Table 1: Application requirements for the example scenario

Figure 1 depicts an example scenario from the automotive domain with three federated systems: door lock system, pre-crash system, and engine control. Table 1 gives the application requirements for the subsystems of the example scenario.

We assume that the pre-crash system has the highest criticality, because a malfunction of this subsystem can cause accidents with major damage and costs. Therefore, the pre-crash system consists of a set of distributed redundant nodes where the system stays operational even in case of an arbitrary failure of one node. Such systems are required to provide ultra-high dependability, with mean-time-to-failure (MTTF) rates better than 10^9 hours [8, 9].

The engine control and the door lock system are of lesser criticality. The engine control function is executed on one ECU locally while the door lock system is distributed over a small non-fault-tolerant ECU, which is located at each door.

The door lock system has the lowest criticality since we assume that there is a mechanical backup system that allows to lock each door with the car keys separately.

The engine control subsystem has to manage the engine cycle which has a duration depending on the revolutions per minute of the engine. At 6000 RPM, the engine cycle is 10 ms, which means that the engine control application requires periodic activation in the order of several ms.

The door lock system is activated sporadically and requires a comparably long response time. We further assume that the pre-crash system is activated in case of an upcoming accident and must be able to perform its activities within a very small time frame with highest priority. In our example, we assume that the pre-crash system is designed as a distributed fault-tolerant system with redundant subsystems that are assigned to different ECUs such that a single fault, e. g., breakdown of an ECU, can be tolerated.

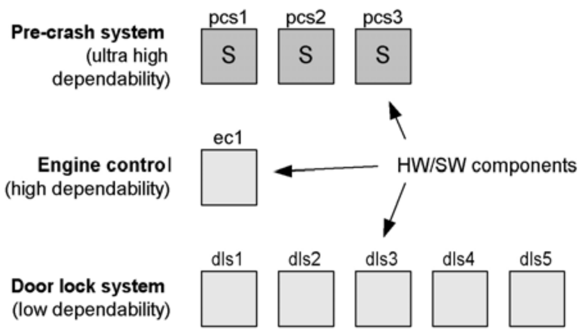

Figure 1: Example of three federated systems

In case of three autonomous (federated) systems, each module (pcs1, pcs2, pcs3, . . .) will be realized as a stand alone HW/SW component (node) as depicted in figure 2 (A). Such a node consists of at least one processing unit (CPU), storage space (memory) and I/O. It may interact with other nodes (communication interface) and with its environment (interface to environment).

We assume that in an integrated architecture the three systems of the example can be integrated to be executed on the same hardware. For such integration, nodes are required that can host more than one federated module (refer to figure 2 (B)).

The SW modules that are executed on this integrated node share resources such as processing unit, storage space, I/O, communication interface and interface to environment.

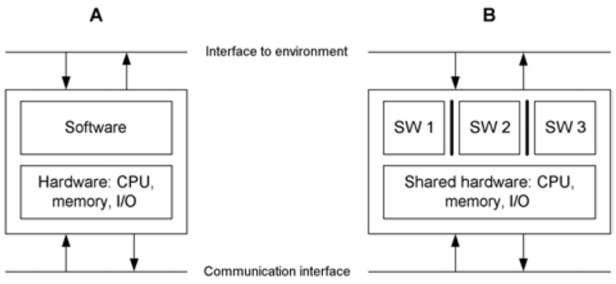

Figure 2: HW/SW component

Integrating the three federated systems of our example requires that the dependability properties that are established for each single system are not invalidated by the integration. Thus, it must be guaranteed that a non-safety-relevant application, e. g., the door lock system, can not interfere with a safety-relevant application, e. g., the pre-crash system. Two levels of interference must be avoided:

- First, interference in *time* must not occur, i. e., the engine control application must not block CPU usage or access to (shared) I/O of the pre-crash system.
- Second, interference in *space* must be avoided, i. e., the engine control application must not overwrite memory regions of the pre-crash system.

Furthermore, the integrated architecture must meet all existing communication requirements. In our example, an integrated architecture has to provide communication mechanisms that are fast enough for the engine control while at the same time guaranteeing reliable communication for the pre-crash system. In the future, it could be possible that sensor and actuator networks, body electronics, safety-relevant functions, and even multimedia applications shall be executed on the same base architecture. In such a case, very high bandwidth in the gigabit range would have to be offered while at the same time guaranteeing reliable and predictable communication with a priori known latencies and jitter.

2.2 State of the art

The integration of functional units into one physical system demands a strong and reliable protection of the units against each other. In order to fulfil this requirement, a mixture of hardware (e. g., memory protection) and software (e. g., RTOS [10]) is necessary. In the past, a series of standards (or de facto standards) have been evolved, covering the needs of the industry (or application) area in which they are used. For instance, in December 1985 the Department of Defence of the United States of America released the TCSEC Orange Book [11], which is an important basis of most of the avionics standards, like the ARINC653 [12]. The ARINC653 specification contains a very strict interface for C, C++ and ADA interfaces, with the focus *to define a general-purpose APEX (APplication/EXecutive) interface between the Operating System (O/S) of an avionics computer resource and the application software.* ARINC (Aeronautical Radio Inc.) itself is a company of which the United States scheduled airlines are the principal stockholders. Several companies have been providing products that at least partially fulfil the ARINC 653 criteria (e. g., Greenhills [13], Wind River Systems, Lynux Works [14], BAE Systems [15]). The automotive industry has also developed standards that deal with multiple functional units within a physical system (e. g., OSEKtime [16]), but these standards are not as mature as the standards in the aerospace industry. All standards/specifications have in common that they formulate mechanisms to partition and protect the available resources, i. e., processing time, memory space and I/O that are shared by the functional units.

Approaches for higher integration can already be found in series production within the aerospace domain. For instance, Airbus developed an integrated modular avionics (IMA) architecture for the A380 in order to process numerous aircraft functions on top of a modularized processing hardware platform. However, according to [17], the IMA architecture does not extend to all functions. Several flight-critical systems are still developed as separate (federated) systems.

2.3 Integrated Architecture Requirements

An integrated architecture is aimed at hosting large and complex real-time computer systems. In order to properly set up an integrated architecture system, the following prerequisites have to be established:

- Hardware base: Hardware components must be provided that offer sufficient processing power, memory, required I/O and mechanisms to support physical encapsulation, e. g., by a memory protection unit, I/O blocks with different access rights.

- Architectural services: On an architectural level, dedicated services must be available that prevent unintended mutual influence of application parts. Furthermore, from the viewpoint of an application, the communication interface and the interface to the environment shall resemble the interfaces within the federated system. Thus, it must be possible that different communication protocols coexist within an integrated architecture.

- Application development support: An integrated architecture system that may consist of thousands of cooperating sub-units requires proper design methods and tool support. The mental complexity must be reduced by the introduction of structure and hierarchical relationships (refer to [18]). Furthermore, the generation of scheduling information, i. e., schedules that control communication and schedules that control the execution of applications must incorporate many dependencies that are hard to dissolve manually.

3 The DECOS Approach

The goal of DECOS is to develop the basic enabling technology to move from a federated architecture towards an integrated architecture [6]. Integration of mixed-criticality systems, i. e., systems that guarantee high dependability for mission-critical system parts while supporting non-critical system parts on the same hardware will be supported. The components and tools developed within the project will consist of hardware components, software services and tools for the design of an integrated architecture. The intersectoral approach of DECOS is driven by the participation of project partners from aerospace, automotive, and industry domains.

3.1 Terminology

This section shall give a short definition of important terms that are used in the scope of the DECOS project (refer to [6, 19, 7]):

- Node: A node forms a fault containment region (FCR) and can contain one or more partitions. In accordance to [20], a node shall be defined as a self-contained computer with its own hardware (processor, memory, communication interface, interface to the process environment) and software (application programs, operating system), which performs a set of well-defined functions within the distributed computer system.

- Partition: A partition forms an encapsulated execution space within a node with a priori assigned static resources (CPU time, memory) that can host one job.

- Distributed Application Subsystem (DAS): A distributed application subsystem (DAS) is a nearly autonomous application system that performs a specified service (e. g., power train management, body electronics, multimedia subsystem in a car).

- Job: A job is defined as a software subsystem of a DAS that is a unit of distribution.
- Port: A port is an access point of a job for the sequential sending and receiving of messages from/to a job.
- Interface: An interface is a set of one or more related ports of a job.
- Gateway Job: A gateway job is a job that comprises two or more interfaces to different DASs and can act as a gateway between these DASs.
- Physical TT Channel: A physical TT channel is a time-triggered physical channel between nodes.
- Virtual Channel: The communication system that transports messages between the ports of the different jobs of a DAS is called virtual channel. Many encapsulated virtual channels (event-triggered and time-triggered) can be implemented on a single physical channel (e. g., CAN on top of TTP).

3.2 Node Design

In order to allow integration of mixed-criticality applications, a DECOS node consists of a mixture of hardware and software components that enable encapsulation of jobs. As mentioned above, encapsulation must be guaranteed in time (guaranteed CPU usage without unintended interference) and space (guaranteed memory usage without unintended interference). Furthermore, virtual communication links are required in order to allow the integration of jobs with different communication behavior on the same hardware. Different communication protocols (event-triggered or time-triggered) may be used. For instance, a safety-relevant job could require the time-triggered protocol TTP, while the non-safety-relevant jobs depend on the event-triggered protocols CAN and LIN or FlexRay.

Setting up protection in space requires hardware support, i. e., a protection hardware of the employed microcontroller that assigns memory access rights to a certain partition. A pure software solution would not be able to guarantee the same level of trustworthiness as a hardware-enabled solution. In particular, a software solution, can only *detect* errors in case of malicious behavior of partitions. In contrast, hardware-enabled partitioning *avoids* errors by blocking illegal requests of malicious partitions.

Within DECOS, a *software supported hardware solution* will be set up that combines hardware enabled partitioning and software error checking. In a hardware/software enabled partitioning scenario, a dedicated hardware protection unit provides basic partitioning. In addition to basic partitioning, a software mechanism (part of the operating system) is responsible for splitting up the memory regions into smaller execution spaces by constantly reconfiguring the memory protection hardware during runtime.

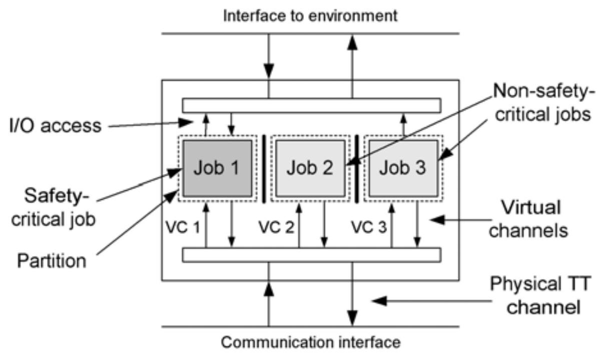

Figure 3: A DECOS node (example configuration)

The operating system of a DECOS node will consist of a Core OS part (COS) and a number of partitions that may include their own Partition OS (POS). The correct function of the COS is critical to the correct operation of the whole node. Thus, the criticality level of the COS is determined by the highest existing criticality level of all jobs that are executed within the system.

The COS provides partitioning of the available CPU time by assigning a certain time slot to each partition in which the job of the partition is granted access to the CPU. The schedule of the partitions time slot is statically known, i. e., it is a priori generated and is not changed during runtime.

Furthermore, the COS provides protection of I/O. In case of memory mapped devices, the particular memory region of the device is protected by the hardware protection unit, i. e., the particular memory region is assigned to exactly one partition. In case of I/O blocks that shall be shared between different jobs, a dedicated I/O gateway job has to be set up that controls I/O access.

A mixed-criticality node requires the execution of several low-level functions – for instance, the time-triggered exchange of messages or fault tolerance tasks. In order to maximize the available CPU time for the jobs, several low-level functions are implemented as function block units, i. e., with their own hardware.

Within DECOS, a newly developed communication controller, and a supporting layer for virtual communication and fault tolerance is realized on an FPGA basis. The communication controller is responsible for time-triggered communication and decouples the jobs from the statically scheduled exchange of messages. A supporting layer for virtual communication and fault tolerance will provide low-level communication services like frame packing and unpacking, byte-order conversion and channel-redundancy handling.

3.3 Architecture Design

A distributed application subsystem consists of a number of jobs that are assigned to certain partitions of several nodes. Thus, an integrated architecture must consist of multiple nodes that are interlinked by a dedicated communication channel. As depicted in figure 4, each node has its own local interface to the environment and to the physical communication channel that is used for interconnection of the nodes.

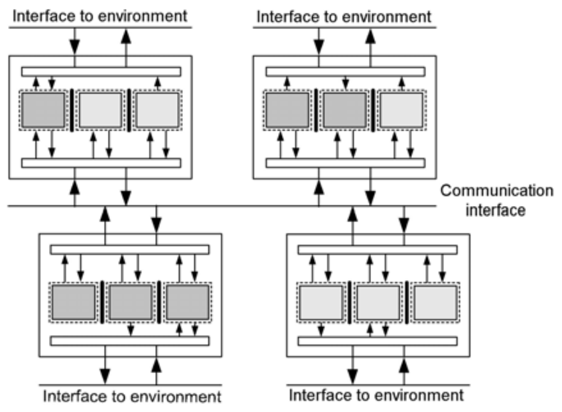

Figure 4: DECOS integrated architecture

An integrated architecture for mixed-criticality applications [7, 6] must be based on a target architecture that supports the safety requirements of the highest considered criticality class. This is of particular importance in systems with safety-critical subsystems, where the physical structure of the integrated system is determined to a significant extent by the independence requirement of fault containment regions [21] of the safety-critical subsystem. Thus, an integrated architecture must be built upon a safety-critical target platform like the Time-Triggered Architecture [22] that provides the following *basic services*:

- Communication infrastructure: Predictable transport of messages.
- Co-ordination of resources: Fault-tolerant clock synchronization of global time (sparse time base [23]).
- Fault/error partitioning: Fault/error isolation and tolerance.
- Resource awareness and management: Communication fault diagnosis and deterministic recovery.

In addition to the services of the target architecture, several additional services are required in order to set up an integrated architecture:

- An *encapsulated execution environment*, that provides protection and scheduling of the distributed application subsystems.
- *Virtual communication links* that allow the emulation of different time-triggered and event-triggered communication protocols.
- A *diagnostic service* in order to improve the identification of faulty components.

Within DECOS, an *encapsulated execution environment* will be provided on architecture level that is the basis for the integration of different distributed application subsystems. The encapsulated execution environment includes the operating systems of all nodes together with scheduling information. The scheduling information must take into account all constraints that have been identified for the jobs of the DASs (e. g., periodicity and validity span of messages, dependencies between jobs).

Virtual communication links and gateways are realized within DECOS in order to support different communication protocols of the federated applications. On node level, a job shall suppose the communication link to be real. Thus, virtual communication links that are emulated on top of a physical communication link have to be provided on an architectural level. Furthermore, the interconnection of virtual communication links and physical communication links shall be provided by the realization of gateways.

A *diagnostic service* will be set up that supports the diagnosis of transient and intermittent component failures and identifies erroneous states on job level (as opposed to node level). For the diagnostic service, information from more than one DAS is required in order to identify frequent transient faults on a particular node. This is because transient faults would probably be noticed by jobs from different

DASs that are executed together on the same node. Frequent transient faults could be an indication for an upcoming permanent hardware fault.

3.4 Application Development

The integration of different distributed application subsystems to be executed on the same hardware raises the need for proper design methods. The application developer must be assisted by appropriate tools, i.e., tools that automatically generate schedule and configuration files and determine the allocation of executables to particular nodes. DECOS tool support extends to middleware resource requirements calculation, resource allocation for message transmissions and task execution and generation of configuration files for the target hardware. Furthermore, proper concepts will be elaborated in order to fulfil resource allocation requirements within an integrated architecture. Thereby, it has to be guaranteed that properties established for a particular Distributed Application Subsystem (DAS) are preserved (at least to a large extend) in case the DAS is employed in a different environment.

Further research and development activities could improve on-line monitoring and simulation capabilities of the DECOS integrated architecture. Real-time monitoring of the transmitted data on the TT physical channel assists in determining the system behavior and helps debugging. Real-time simulation, i.e., simulation of the behavior of a particular job within the integrated architecture, is a desirable approach when it is not permissible to operate on the real system. The real system may be not available, either because it has not yet been built, or because it is located at a different physical site. During early test phases, simulation is a specially preferred approach for safety reasons, since in these phases the confidence in the correctness of the system is still rather low. Furthermore, simulation sometimes provides the only possibility to test the system behavior in "rare event scenarios", because in the real world it would either be very difficult, unsafe or costly to guide the system into these situations. E.g., the behavior of a control system in case of critical temperatures in a nuclear power plant or crashes of airplanes can not be tested by establishing the respective situation.

4 Summary and Outlook

The current trend of architecture design for distributed embedded real-time systems calls for integration of mixed-criticality subsystems. In the past, each distributed application subsystem had to be executed on its own hardware base in order to guarantee elimination of mutual interference or had to be developed in accordance to the highest existing criticality level. In the near future, it shall be possible to execute mixed-criticality distributed application subsystems on the same distributed architecture. Furthermore, it may be a future goal to consider different levels of security requirements of the software modules.

Goal of the DECOS project is to develop an integrated architecture that offers methods for encapsulation as well as diagnostic services. The deployment of an integrated architecture solution will decrease development, production, and maintenance efforts and increase the dependability of embedded real-time applications. In the aerospace domain, effort has already been invested in order to integrate several subsystems within the newest generation of airplanes. However, within DECOS we expect further development of integrated solutions and the exploitation of the benefits of an integrated architecture in the automotive and other branches of industry.

Acknowledgments

We warmly acknowledge many discussions with members of the DECOS team. In particular, we would like to thank our colleagues from Vienna University of Technology for establishing a precise and sound definition of concepts and terminology. Besides Prof. Kopetz, head of department at the Institute of Computer Engineering, several more persons deserve our gratitude: Roman Obermaisser, Philipp Peti, Bernhard Huber, and Christian El Salloum. Furthermore, we would like to thank our colleagues at TT-Tech, in particular Stefan Poledna, Georg Stöger, and Bernhard Wenzl and our colleagues from Austrian Research Centers Seibersdorf, in particular Manfred Gruber and Georg Weissenbacher for their comments on previous versions of this paper.

This work was supported in part by the European IST project DECOS under contract No. IST-511764. All necessary clearances for the publication of this paper have been obtained. If accepted, the author will prepare the final manuscript in time for inclusion in the conference proceedings and will present the paper at the conference.

References

[1] Verband der Automobilindustrie (VDA). HAWK2015 – Herausforderung Automobile Wertschöpfungskette. Henrich Druck + Medien GmbH, Schwanheimer Strasse 110, D-60528 Frankfurt am Main, 2003.

[2] C. Constantinescu. Trends and Challenges in VLSI Circuit Reliability. IEEE Computer Society, 2003.

[3] H. Kopetz. A roadmap of the TTA. Research Report 4/2003, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 1-3/182-1, 1040 Vienna, Austria, 2003.

[4] N. Suri. Dependable embedded systems and services: A personal crystalball outlook. Available at http://www.ece.cmu.edu/~koopman/des03/des03_suri.pdf, October 2003.

[5] R. Hammett. Flight-critical distributed systems: design considerations. *IEEE Aerospace and Electronic Systems Magazine*, pages 18(6):30–36, 2003.

[6] M. Gruber. DECOS project proposal, 2004. Austrian Research Center, Vienna, Austria, http://www.decos.at.

[7] H. Kopetz, R. Obermaisser, P. Peti, and N. Suri. From a federated to an integrated architecture for dependable embedded real-time systems (draft), 2004. Vienna University of Technology, Austria and Darmstadt University of Technology, Germany.

[8] N. Suri and C.J. Walter. Advances in ultra-dependable systems. *IEEE Press*, 1995.

[9] H. Kopetz. On the fault hypothesis for a safety-critical real-time system. In *Proceedings of the Automotiove Workshop San Diego*, CA, USA, January 2004.

[10] MNIS. RTOS. http://www.rtai.org, 2005.

[11] United States Department of Defense. Department of Defense Trusted Computer System Evaluation Criteria, DoD 5200.28-std. http://www.radium.ncsc.mil/tpep/library/rainbow/5200.28-STD.html, December 1985.

[12] Airlines Electronic Engineering Committee. Avionics Application Software Standard Interface, ARINC Specification 653. http://www.arinc.com, 1997.

[13] Green Hills Software Inc. Integrity. http://www.ghs.com, 2005.

[14] Arun Subbarao. The technology behind LynxOS v4.0's linux ABI compatibility. http://www.linuxdevices.com/articles/AT8943314364.html, June 2002.

[15] BAE Systems. CsLEOS. http://platformsolutions.na.baesystems.com, 2005.

[16] OSEK VDX. TTOS. http://www.osek-vdx.org, 2005.

[17] C. Adams. A380 innovations: A balancing act. *Aviation Today*, march 2003.

[18] H.A. Simon. *The Sciences of the Artificial*. MIT Press, 1996.

[19] H. Kopetz. Distributed real-time systems engineering. Lecture slides, Presentation of the Time-Triggered Architecture, May 2004.

[20] H. Kopetz. *Real-Time Systems, Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, Boston, Dordrecht, London, 1997.

[21] H. Kopetz. Fault containment and error detection in the Time-Triggered Architecture. Research Report 39/2002, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 1-3/182-1, 1040 Vienna, Austria, 2002.

[22] H. Kopetz and G. Bauer. The Time-Triggered Architecture. *Proceedings of the IEEE*, 91(1):112 – 126, January 2003.

[23] H. Kopetz. Sparse time versus dense time in distributed real-time systems. In *Proceedings of the 12th International Conference on Distributed Computing Systems*, Yokohama, Japan, June 1992.