BASEMENT: An Architecture and Methodology for Distributed Automotive Real-Time Systems

Hans Hansson, *Member*, *IEEE*, Harold Lawson, *Fellow*, *IEEE*, Olof Bridal, Christer Eriksson, Sven Larsson, Henrik Lön, *Student Member*, *IEEE*, and Mikael Strömberg

Abstract—BASEMENT™ is a distributed real-time architecture developed for vehicle internal use in the automotive industry. BASEMENT covers application development, as well as the hardware and software that provide execution and communication support. This paper gives an overview of the BASEMENT concept, as well as presenting two system realizations. The first realization is based on the commercial real-time kernel Rubus, while the second is an ultra-dependable architecture (DACAPO) with provisions for fault tolerance at various system levels.

BASEMENT is designed for the automotive systems of the future. These systems will be required to simultaneously handle multiple safety critical functions and a large number of less critical functions. All of these features are to be provided at a production cost substantially lower than that of current systems, and, at the same time, with a reliability allowing vehicles to be built without mechanical backup systems, even for safety critical subsystems such as braking and steering.

The key constituents of the concept are: 1) resource sharing (multiplexing) of processing and communication resources, 2) a guaranteed real-time service for safety critical applications, 3) a best-effort service for nonsafety critical applications, 4) a communication infrastructure providing efficient communication between distributed devices, 5) a program development methodology allowing resource independent and application oriented development of application software, and 6) a straightforward and well-defined operation principle enabling efficient fault tolerance mechanisms to be employed.

Index Terms—Distributed real-time system, holistic approach, automotive application, software development, real-time kernel, scheduling, fault-tolerance.

1 Introduction and Requirements

The development of new automotive functions based upon the use of modern electronic, computer, and communication technologies has been accelerated in recent years. Several products have been developed; for example, ABS (Anti-lock Brake Systems), Cruise Control, and Engine Management Systems. In addition, experimental projects, e.g., within the European PROMETHEUS and DRIVE programs, have resulted in the identification of a number of interesting functions. Several of these have been realized in prototype implementations. It now appears that a variety of new functions based on multiplexed distributed processing will be successively introduced during the 1990s and into the next century. It is expected that this shift to a new technology in automobiles will alter the basic approach to automotive design as indicated by Rivard [1]:

> "Designers will escape from the mechanical function add-on approach. They will seek to optimize total vehicle performance through the use of electronics. … The design of the vehicle will be approached from the perspective of the customer and will be committed to improving ride quality, handling characteristics, steering effectiveness, brake feel, information display format, and to improving the power-train in terms of economy, emissions and driveability."

With this scenario in mind, the Vehicle Internal Architecture (VIA) research project was launched within the Swedish Road Transport Informatics (RTI) program (1991-1994), which was supported by the Swedish National Board for Industrial and Technical Development (NUTEK), as well as the Swedish National Road Administration, the Swedish Transport Research Board, Saab-Scania AB, AB Volvo, and Swedish Telecom. The RTI program was coordinated with the European programs DRIVE and PROMETHEUS.

In VIA, we developed BASEMENT, a vehicle internal real-time architecture. The objective was to design a platform that meets the stringent demands of the automotive industry. BASEMENT is a pilot for future vehicle internal distributed real-time systems. As such, it is required to provide

- A communication infrastructure, allowing cost-effective communication between physically distributed units.
- An execution platform for application software, pro-

- *H. Hansson is with the Department of Computer Systems, Uppsala University, Uppsala, Sweden. E-mail: Hans.Hansson@docs.uu.se.*
- *H. Lawson is with Lawson Konsult AB, Stockholm, Sweden. E-mail: bud@lawson.se.*
- *O. Bridal and H. Lönn are with the Department of Computer Engineering, Chalmers University of Technology, , Sweden. E-mail: {olle, hlonn}@ce.chalmers.se.*
- *C. Eriksson is with the Department of Computer Engineering, Mälardalen University, , Sweden. E-mail: cen.mdh.se.*
- *S. Larsson and M. Strömberg are with Mercel AB, Göteborg, Sweden. E-mail: {Sven.Larsson, Mikael.Stromberg}@mecel.se.*

For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 105217.

viding *guaranteed* services for safety critical applications, while giving *acceptable* response times for non-safety critical applications.
- Resource sharing, i.e., permitting multiple vehicle internal applications to efficiently share communication infrastructure as well as computing resources (processors).
- A priori predictability for safety critical applications, i.e., it should be possible to off-line (before runtime) determine if sufficient resources are available to guarantee required behavior.
- Reliability, i.e., the probability that system failures are avoided for a specified length of time should be very high. The maximum allowed failure probability for a one-hour time span is in the order of 10^{-8} for safety-critical automotive applications.
- Facilities for communication with vehicle external equipment.
- Open interfaces, i.e., the interfaces, connectors, and communication protocols should be precisely defined. This is to allow different vendors to develop compatible equipment, and to facilitate the integration of components from different vendors in one system.
- An application development environment and methodology, providing engineers with an application oriented interface, as well as tools for efficient development and integration of applications.
- An architecture which allows large product series to be implemented at a very low cost.
- Simplicity, both in terms of minimal run-time overhead (i.e., minimal amount of *nonproductive code*), and in terms of a simple and intuitive method for application development. This simplicity facilitates validation and formal proof of correctness, as well as the development and introduction of fault tolerance mechanisms with high coverage and short recovery latencies.

In this paper, we present the main results of the VIA project. Its main contribution lies in the holistic and application oriented nature of our approach to designing distributed automotive real-time systems and applications. Our focus has been on how to design a working system satisfying the above listed requirements, rather than solving a very specific technical problem. The work is, of course, based on state-of-the-art results, but to satisfy the, in many cases, contradictory requirements, we were, as most engineers, forced to compromise. For instance, to facilitate fault detection and recovery, we use more traditional static-cyclic scheduling, rather than more flexible (and less traceable) dynamic scheduling methods such as Fixed-Priority / Rate-Monotonic or Earliest-Deadline First.

This paper is based on the collective efforts of the BASEMENT design team, with members from the following organizations: Mecel AB, Arcticus Systems AB (in cooperation with Mälardalen University), Swedish Institute of Computer Science, Lawson Konsult AB, Chalmers University of Technology (Department of Computer Engineering), and Uppsala University (Department of Computer Systems).

1.1 Outline

Section 2 provides an overview of the BASEMENT concept. Section 3 introduces the principles of operation. In Section 4, we present two prototype BASEMENT systems: one based on the commercially available real-time kernel Rubus, and the other being the ultra-dependable Basement architecture DACAPO. In Section 5, we present a BASEMENT system realization and its application: an Autonomous Intelligent Cruise Controller. Section 6 discusses some design decisions and reviews background and related work. Finally, in Section 7, we summarize and present some new activities that the VIA project has led to.

2 The BASEMENT Concept

The BASEMENT concept provides a holistic view of developing automotive applications in the sense that, not only the execution, but also the development, of application software is considered. The structure and behavior of application software can be described at a rather high level of abstraction. Such descriptions are independent of the actual hardware on which it will be executed. Tools are provided for mapping abstract descriptions to a particular BASEMENT system.

The design of applications is based on a hardware metaphor in that software is built from a set of predefined (or user-defined) software components which, in analogy with hardware circuits, are termed *Software Circuits*. The main motivation for using such a metaphor is that it allows a structuring of the software which is conceptually close to hardware design, and, thus, will be familiar to engineers in the automotive industry. Also, the simple structure increases provability and improves human to human communication concerning designs. This reduces the risk of specification and implementation errors.

Fig. 1 presents an idealized view of the design process. The development of application software starts by defining its *abstract behavior*, which essentially amounts to building a network of software circuits. No information about timing and location is included in the abstract behavior. Such information is provided in the subsequent phases: Adding timing and other constraints yields the *concrete behavior*. Resource information provided to the mapping tool describes the *target system* and indicates location constraints (e.g., that a particular software circuit must execute on a particular node). Based on the concrete behavior and this information, the mapping tool generates code to be executed on the various nodes.

2.1 The Hardware Architecture

A BASEMENT system consists of a set of nodes interconnected with a communication network, as depicted in Fig. 2. A *node* can be viewed as a computer (processor + main memory) with a network interface and a set of input / output devices (sensors and actuators) allowing interactions with the "physical process" (the vehicle). The *communication network* is required to be *deterministic*, i.e., it should (in principle) provide error free transmission of data with bounded and predictable delays. The communication network also provides facilities for communication with vehicle external equipment and networks.

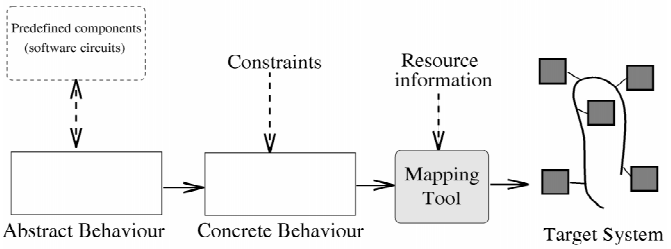


Fig. 1. Behavior–Mapping–Resources.

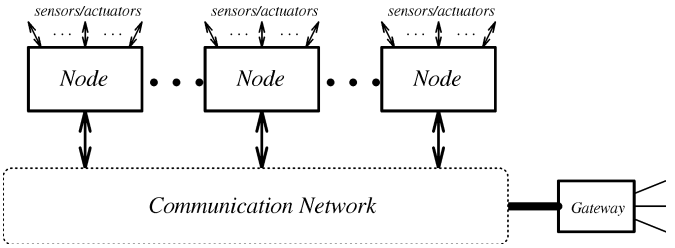


Fig. 2. The hardware structure.

It should be noted that Fig. 2 illustrates an abstract architecture, in the sense that an actual system realization might be more complex. For instance, due to reliability requirements, each node might contain several redundant processors and there might be redundant networks.

2.2 The Software Architecture

Automotive applications are either *safety-critical* or *non-safety-critical*, e.g., braking is a safety-critical application, whereas climate control is considered to be non-safety-critical. Safety-critical real-time applications have stringent timing constraints (deadlines) that must be fulfilled under all circumstances. Also, for non-safety-critical applications, there are usually timing constraints, but these constraints are less strict, and a failure to meet such a constraint will not result in a hazardous situation (potentially leading to an accident). The terms *hard* and *soft* real-time applications are often used to denote safety-critical and non-safety-critical applications, respectively. Applications are implemented by *processes* (tasks) which contain program logic in the form of software circuits. In analogy with applications, a process is characterized as either being hard or soft, depending on whether its timing constraints are stringent or not. A soft real-time application is implemented by one or more soft real-time processes, whereas a hard-real-time application is implemented by at least one hard real-time processes, possibly together with some additional soft and / or hard processes.

The basis for the software architecture is the fundamental difference between hard and soft processes. The color Red is associated to hard processes, and Blue to soft processes. A Red and a Blue service is provided. A single process, as well as the set of processes handling a particular application, may be distributed over several nodes.

There is a strict separation between Red and Blue processes (see Fig. 3). Since both types of processes use the same network, there is a shared *communication service*. To prevent Blue processes from interfering with Red network accesses, and thus violating the strict requirements on Red processes,

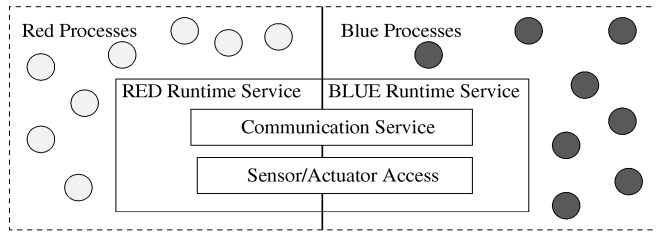


Fig. 3. Software structure in one node.

the *communication service* reserves a certain amount of network accesses for Red processes.

The *Sensor/Actuator access* module provides functions for accessing the physical sensors and actuators attached to the node. Processes may share sensors, but actuators cannot be shared, i.e., several processes may read the value of a sensor, but only one process has the exclusive right to write a value to a physical actuator.

The *Red Runtime Service* provides Red processes with sufficient execution support to guarantee that their deadlines are always met. The *Blue Runtime Service* allows Blue processes to efficiently share the remaining resources. That is, the Blue subsystem only has at its disposal the resources (e.g., processing power and network accesses) which are not needed by Red processes.

2.3 Software Development

The software development methodology is an important aspect of the concept, since it prescribes a way of developing application software for BASEMENT systems. The methodology is based on developing sets of interconnected *Software Circuits* (SCs). Each SC has a set of input connectors where data is received, and a set of output connectors where data is produced (see Fig. 4a). Communication between two or more SCs is achieved via connectors, as illustrated in Fig. 4b.

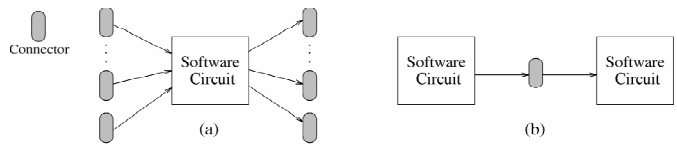


Fig. 4. (a) A Software circuit and its connectors. (b) Communication via connectors.

The execution of a software circuit is enabled when appropriate data is available at all input connectors, at which time the circuit can perform its processing and produces data at the output connectors. Conceptually, the operation of a SC is partitioned into the three phases:

1) Read data from input connectors; this is an atomic operation in the sense that exactly the data present in the input connectors when the reading starts will be read.
2) Perform processing. During this phase the SC cannot interact with its environment, i.e., the results can only be based on data read during phase 1, and, possibly, some local data contained in the SC.
3) Write data to output connectors.

Software circuits can be combined to form larger software circuits. Acyclic networks of interconnected SCs are used to program the behavior of Red and Blue processes, as illustrated in Fig. 5.

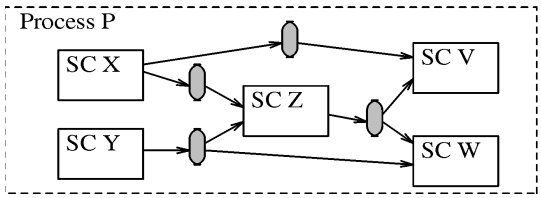


Fig. 5. A process.

Processes are either periodic or aperiodic. A periodic process is invoked regularly at fixed points in time, whereas aperiodic processes are event driven and invoked only when a particular event (or set of events) occur. Red processes must be periodic, whereas Blue processes may be either periodic or aperiodic.

More detailed descriptions of the Software Methodology are provided in [2], [3], and [4]. Central properties of the Methodology include:

1) *The treatment of Software Circuits as transforms between Connectors which hold Sensor and Actuator signals.*

As a result, the abstract structure and behavior is viewed as a signal flow graph, where the execution of transforms is either time driven (Red) or event driven (Blue). Due to the simplicity of the time and event driven approaches that are employed, complex communication and synchronization mechanisms are not required. The same abstract descriptions suffice for both Red and Blue.

For Red software circuits, a static schedule guarantees the execution order of the software circuits where produced signals are known to be available prior to their consumption.

Blue software circuits are typically initiated on an event driven basis, including the receipt of new external signals, as well as the internal production of a new Connector signals that trigger the execution of software circuit component logic.

Software circuits are built from concrete behaviors developed via a restricted sequential programming style, in which simple (pre-)Programmed OPerations (POPs) are utilized. POPs are the simplest form of software circuits. More complex SCs can be built from POPs.

2) *Processes composed of software circuits can be organized according to a hierarchy of levels.*

Higher level processes consider lower level processes as machines that they can observe and regulate (via sensor and actuators).

A useful level structuring is composed of the following:

STRATEGIC CONTROL
PRIMARY CONTROL
INTERFACE AND CONDITIONING
TERMINATOR

Terminator Level—The physical level composed of hardware sensors and actuators.

Interface and Conditioning—For sensors, the programmed behaviors at this level provide processing logic to transform signals into processable quantities including A to D conversions. For actuators, the programmed behaviors transform digital values into actuation signals including D to A conversions.

Primary Control—Programmed behaviors for fundamental control loops. The control loops provide both active (continuous sampled) and passive (on demand event-based) regulation. Primary control functions (SCs) always process logical sensor and actuator values.

Strategic Control—Utilizes connector information (logical sensors and actuators) to observe and regulate lower level processes. Provides strategic higher level functions including monitoring, fault detection, fault isolation, fault tolerance measures, adaptive control (including fuzzy and artificial neural net based). A strategic level control elicits changes in the lower levels via access to logical sensors and/or actuators.

All applications involve the utilization of the layers up to and including Primary Control. Some applications will also employ strategic control levels.

3) *Component structure is attained via the use of standard "Connectors" which are used as holding places ("latches") for signals.*

Various partitionings of Software Circuits reflecting the various levels of process (machine) control and various granularities (coarse grain down to fine grain) can be employed in order to accommodate definition and utilization of standard components and/or the development of or delivery of components from suppliers.

The software methodology is a key constituent of BASEMENT. It provides a straight-forward means of viewing the functions to be provided as an advanced form of signal processing. Consequently, the complexities introduced by more general purpose software methodologies are avoided.

The hypothesis is that this minimal, but sufficient, methodology approach will yield increased understanding, provide a basis for verifiable solutions (even formally), and lead to efficient implementations which minimize the usage of nonproductive code (for the application and system software). Further, the implementation of fault tolerant solutions can be accommodated.

3 Principles of Operation

There is, as illustrated in Section 2.2, a clear separation between Red and Blue subsystems, e.g., the handling of processes in the two subsystems is independent, as will be further explained in this section.

3.1 The Red Subsystem

The Red subsystem is based on the cyclic, off-line, scheduling paradigm. In this paradigm, a fixed schedule (statically defining the sharing of resources at run-time) is calculated off-line. That is, information about requirements and demands (e.g., the CPU time needed for performing processing) are fed into a tool which generates static schedules; one for each node. These schedules define the execution order of processes at run-time. The generation of static

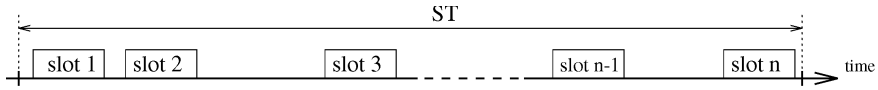


Fig. 6. Network scheduling.

schedules requires the scheduled processes to be periodic. That is, the Red subsystem is tailored for handling periodic safety critical processes. The length of the generated schedules is denoted *the systolic base time* (ST), since it defines a base frequency of a system.

It is the task of the Red *kernel* (one in each node) to guarantee that the off-line generated schedules are followed. Since the run-time kernels are time driven by different clocks, the precision of the clocks must be taken into account in the off-line scheduling, and a clock synchronization algorithm must be used at run-time. The BASEMENT concept does not prescribe or preclude any particular clock synchronization algorithm, but it requires a known upper bound for the maximal difference between local clocks. In [5], we further discuss clock synchronization.

3.1.1 Modes and Mode-Shifts

It is assumed above that all Red processes are continuously operating at fixed frequencies during the execution of the system. This might be suitable for some applications, but many applications are characterized by a set of distinct operational modes with different requirements on the processes to execute, as well as their frequencies. For instance, in the automotive environment, modes can include Cold, Startup, Idling, Moving, Emergency, Fault, and Diagnosis. For each mode, the set of active processes and their frequencies must be defined. A separate static schedule is then generated for each mode, and facilities for dynamically moving from one mode to another (a mode-shift) are provided.

3.2 The Network

The communication service is Time Division Multiplexed. Fig. 6 illustrates that, in the scheduling of the network, a set of (n) communication slots are statically allocated in each ST, a slot being the unit of network scheduling in which one node is given the exclusive right to send one Red message.[1] Note that the slots may be placed anywhere in the ST, as long as they are nonoverlapping. The off-line scheduler statically assigns slots to particular nodes. Communication time not allocated by the off-line scheduler is available for Blue communication.

Some of the reasons for using a statically allocated communication schedule are:

- to obtain predictable durations for the effects of transient message errors, i.e., until the next prescheduled update,
- simplified analysis and testing of the communication system,
- possibility for short latency between sensor readings and message transmissions, as well as between message reception and actuator output,
- rapid detection of faults which lead to a deviation from the predetermined communication schedule,
- simplified implementation of a distributed clock, and
- no protocol related limitations of bit rate (as in collision arbitration protocols, such as CAN [6]).

A discussion of the information that should be contained in the messages in order to support node synchronization and fault tolerance can be found in [7].

3.3 The Blue Subsystem

The Blue subsystem is intended for processes that are "less safety critical" than Red processes. Consequently, the resources available to the Blue subsystem are those remaining after (the static) allocation of resources to the Red subsystem. In contrast with the time driven operation of the Red subsystem, the Blue subsystem is *event driven*, meaning that Blue processes are activated in response to the occurrence of events (including time events).

On each node handling Blue processes, a preemptive priority driven scheduler is used for scheduling the Blue processes. The BASEMENT concept does not prescribe or preclude any particular method for assigning priorities to Blue processes, i.e., both dynamic priority assignments (e.g., Earliest Deadline First (EDF)) and static assignments (e.g., Rate or Deadline Monotonic assignments) are possible.

To employ the hardware metaphor as a programming style, Blue software circuits also use connectors as holding places for sensor/actuator values. However, to allow reuse of existing software, the use of semaphores and queued message passing is allowed, though not recommended.

Blue and Red processes can interact via shared connectors, as illustrated in Fig. 7. The process Red_P in Fig. 7 writes values in the connector A. The *event* of entering a new value in A will trigger the release of the Blue process Blue_P, which calculates a new value for the connector B. Note that there is no precedence relation between B and *SC_A* (indicated by [ ]), since it otherwise would be possible for Blue_P to prevent Red_P from executing SC_A.

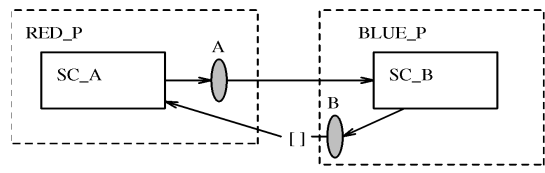


Fig. 7. Blue-Red processes interaction.

A Blue process is either periodic or aperiodic, whereas Red processes are all periodic. There is no guarantee that deadlines of Blue processes are met, though it might, in some cases, be possible to evaluate the schedulability of Blue processes (using, e.g., fixed priority scheduling theory [8]).

1. In an actual system implementation, a slot might—for efficiency reasons—carry more than one message, i.e., each prescheduled slot might contain a finite number of Red messages transmitted from one or several nodes.

4 Prototype Systems

In developing and evaluating BASEMENT, we have designed and built several prototype systems focusing on different aspects of the concept. In this section, we will present two such prototypes. The main component of the first prototype is the commercial real-time kernel Rubus, while the second is the ultra-dependable DACAPO architecture with provisions for fault tolerance at various system levels.

In addition to these prototypes, we have developed some design support tools in the VIA project, including a system simulator and an associated off-line scheduling tool [9], [2]. The Intelligent Cruise Controller presented in Section 5 is yet another example of prototype development within VIA.

4.1 The Rubus System

In this section, we will describe a BASEMENT prototype system developed by the Department of Computer Engineering at Mälardalen University in cooperation with Arcticus Systems AB. The system consists of Intel 386 nodes connected with a Controller Area Network (CAN) bus. CAN [6] is a real-time communication bus developed by Bosch for use in automotive systems. A configuration compiler is used to map application software (given as networks of interconnected software circuits, as outlined in Section 2.3) to the resources in a particular target environment. We will first describe the used real-time kernel (Rubus), thereafter, the communication system, and, finally, the configuration compiler.

4.1.1 The Rubus Real-Time Kernel

Within the VIA project, Arcticus Systems AB has developed the real-time kernel Rubus[2] [10], [11]. Rubus combines time-driven execution, as required by the Red subsystem, with event-driven execution, as required by the Blue subsystem. The objectives when developing the kernel were:

- Modular design, i.e., the kernel should be conveniently configurable for small single CPU (microcontroller) configurations, as well as distributed systems where support for clock synchronization and network communication is required.
- To support application development in C.
- To support the execution of off-line scheduled (Red) processes.
- To support the execution of event-driven (Blue) processes.
- To guarantee the behavior of Red processes, while allowing efficient execution of Blue processes.
- To allow, Red only, Blue only, and mixed configurations.
- To allow interrupt handling (while guaranteeing the behavior of Red processes).
- To allow configuration of different degrees of error detection. Currently, two levels are supported: one checking validity of parameter values only, and the other additionally performing extended checking of data-structures and messages using, e.g., invariants and checksums.
- To support communication between Blue and Red processes.

2. *Rubus Arcticus* is Latin for cloudberries.

Rubus supports execution of Red threads, Blue threads, and interrupt service routines. Conceptually, a thread corresponds (in terms of the application software) to the execution of one of the software circuits of a Red or Blue process. Red threads are invoked according to a preruntime generated schedule. A Red thread returns control to the Red Executive (see Fig. 8) when terminating. A Blue thread is invoked by the Blue executive as the result of some event, and it returns control to the Blue executive by some system service, e.g., by executing a delay statement.

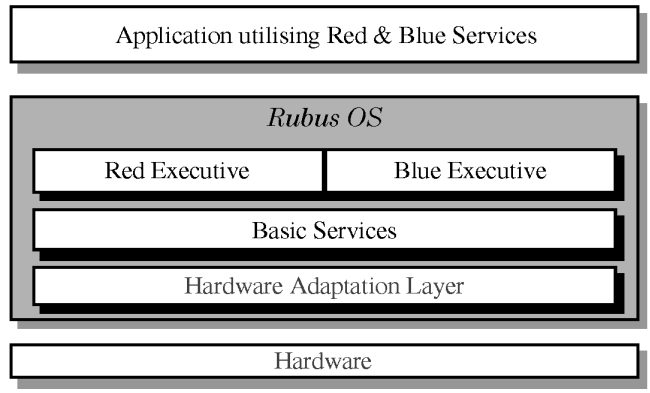


Fig. 8. Overview of Rubus.

Rubus provides services supporting execution of applications programmed as sets of Blue or Red processes, or a combination thereof. These services are provided by the Rubus OS, shown in Fig. 8. The main components of Rubus are:

- The Red Executive, which provides services for Red threads. The temporal behavior of the Red Executive is predictable due to its simplicity and minimal amount of code. If an application uses both Red and Blue services, the Red services will always be given priority.
- The Blue Executive, which basically includes the services provided by POSIX [12] compliant real-time executives, including message handling, semaphores, monitoring routines, and I/O handling.
- The Basic Services, which include common features used by both the Blue and Red Executives, including treatment of time and interrupt handling.
- The Hardware Adaptation Layer (HAL), which includes all processor dependent code. This is to facilitate the porting of Rubus to different hardware platforms.

The main task of the Red Executive is to dispatch threads according to the current schedule and time. The schedule is defined by its period, the threads, the deadlines of the threads, and thread activation times relative to the start of schedule. The Red executive supports preemption between threads. This is to enable handling of processes with large differences in their temporal requirements. Such processes are, as pointed out by Poledna [13], common in automotive systems. As an illustration of the problem, consider the two single thread processes *A* and *B*, having periods, deadlines, and execution requirements of 10ms, 10ms, 2ms, and

1000ms, 500ms, 30ms, respectively. A nonpreemptive execution of task *B*, would require the allocation of a 30ms time slot, thereby making it impossible for *A* to perform its required three 2ms executions during that time. With preemption, on the other hand, *A* can preempt *B* and thereby satisfy its timing requirements. The preemption is accounted for in the schedule.

To minimize stack usage and increase performance, the Red executive supports threads sharing the same stack. This, together with the minimal size of the Red executive, will make it possible to use a Red kernel on simple microcontrollers which have only memory on-chip.

The Red Kernel has the following responsibilities:

- To execute the threads in the order specified in the active schedule (this is handled by the Dispatcher).
- Change schedule upon request.
- Monitor the schedule to detect deadline violations. An error handler is invoked if this occurs.

The Red Kernel is similar to the Mars Operating system presented by Reisinger [14]. The main difference is that the Red Executive in Rubus is integrated with a Blue executive.

The Blue Executive is a traditional event triggered kernel, using a priority based preemptive scheduling algorithm, which guarantees that the thread with highest priority, among the threads ready to run, will always be executed first. The synchronization between threads is supported by message queues, signals, and semaphores, according to the POSIX standard [12].

Rubus allows Red and Blue threads to interact in a controlled manner, such that the communication mechanisms do not influence the timing of Red threads. A Red thread can make information available to a Blue thread either by letting the Blue thread read a shared variable or by sending an event to the Blue executive. The transfer of data from Blue to Red threads is (following [15]) handled by special attributed objects with two copies of the data: one active and one passive. The Blue thread writes the data to be transferred to the passive copy, and then it performs an atomic commit operation to change the passive copy to active. The Red thread always reads the active copy.

4.1.2 Communication System

The communication system is implemented as a protocol in a layer above the CAN protocol. The protocol makes a distinction between hard and soft real-time messages. Hard real-time messages are preruntime scheduled, whereas the soft messages are scheduled on-line.

The protocol uses the TDMA scheduling outlined in Section 3.2. Several frames can be sent in each communication slot. A frame is directly mapped to a CAN message.

To make it possible for the nodes to have a consistent view of the communication slots, an approximate global time base must be established. For details on the clock synchronization used, see [16].

TDMA based protocols usually allocate each communication slot to a particular node. In its communication slot, the node can either send a message or leave the communication slot unused. This might lead to an inefficient use of channel bandwidth. We use a different approach, which allows several nodes to send frames in a communication slot. This could be implemented, since each CAN message has a priority and CAN provides a suitable hardware arbitration mechanism. The protocol also handles membership agreement and mode changes. For a detailed description of the protocol, we refer to [17].

4.1.3 The Configuration Compiler

When designing Red applications, it is very important to have tools that automatically map application software to resource structures (specific target environments). This section briefly describes a Configuration Compiler performing this mapping.

The Configuration Compiler requires both an architecture specification and a configuration specification. The former lists the number and types of nodes and buses, the latter describes the involved Red and Blue processes.

The configuration compiler consists of two cooperating tools: a communication handling tool and a pre-run-time scheduler.

The *communication handling tool* inserts special communication handling system threads in the application software. This is to achieve communication transparency from the senders point of view, e.g., if a receiver executing on the same node as the corresponding sender is reallocated to a different node, then only the inserted communication thread needs to be altered, such that the data will be sent over the bus instead of being copied from one local variable to another. For a detailed presentation of the communication handling tool, we refer to [18].

The *preruntime scheduler* generates, for each mode and node, a schedule and some additional information needed by the Red Executive, including clock synchronization information and information about the bus communication. The scheduler uses a heuristic search to find a feasible schedule for a set of Red processes. The scheduler supports precedence and mutual exclusion relationships, but requires threads to be preallocated to nodes. The scheduler can be configured to support either preemptive or nonpreemptive execution. A detailed description of the scheduler can be found in [19].

The Configuration Compiler differs from similar work, e.g., by Fohler [20] and Ramamritham [21], in that BASEMENT communications are not a priori included in the precedence graphs. This allows synchronizations and communications to be treated as orthogonal concepts. The Configuration Compiler also supports mutual exclusion relationships to handle shared resources. This is not handled by the scheduler developed by Fohler. In addition, the Configuration Compiler takes communication delays into account, something which, e.g., the scheduler proposed by Xu [22] does not handle. Also, the Configuration Compiler gives the user a possibility to choose between generating a preemptive or nonpreemptive static schedule.

4.2 DACAPO—A Fault-Tolerant Architecture

DACAPO (Dependable Architecture for Control of Applications with Periodic Operation) [23] is a version of the BASEMENT architecture that has been developed with particular attention paid to dependability aspects, such as

reliability and safety. The architecture is designed to have the Fault-Operational[3] (FO) property with respect to permanent and transient hardware faults. This property ensures that the system is fully operational when any permanent fault is present, and that the system can recover from any transient fault.

The two most important concerns in the development of the DACAPO architecture have been to enforce the FO property, such that there are virtually no single points of failure, and to achieve a very high detection coverage for any nonrecoverable fault. Assuming that any detected nonrecoverable fault is manually repaired shortly after its occurrence, this strategy results in an ultra-dependable system.

4.2.1 Node Architecture

A DACAPO system consists of a number of fault-tolerant nodes interconnected by two buses. Every module inside the node is duplicated as shown in Fig. 9.

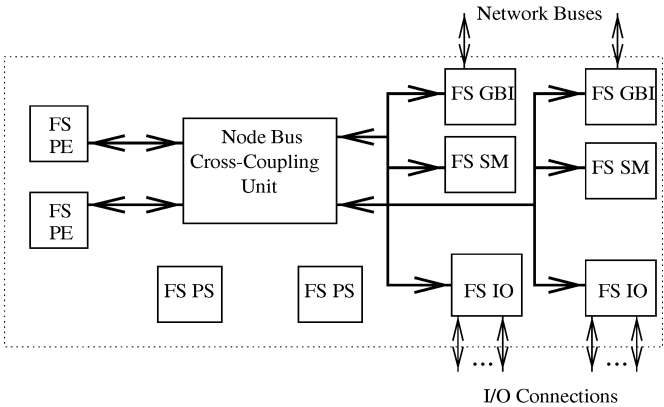


Fig. 9. The DACAPO node architecture.

The abbreviations used in the figure are:

- PE: Processing Element (with local memory)
- PS: Power Supply
- GBI: Global Bus Interface
- SM: (State) Memory
- IO: Input/Output Interface

As indicated in the figure, all modules are fail-silent (FS), with the exception of the cross-coupling unit that provides the connection between any processing element and any node bus. This unit is designed to be fault-operational (FO) in the sense that at least one processing element can access at least one node bus when a fault is present in the unit. Furthermore, all modules are powered such that a failure in either power supply does not cause two equivalent modules to fail simultaneously. Thus, the node as a whole is a fault-operational (FO) entity. In fact, the modules are designed such that the node is fault-operational/fail-silent (FO/FS) for most fault combinations. This means that the node does not fail when a first fault occurs and that a second fault either causes the node to become silent toward the network and the actuators or to remain operational.

3. Traditionally, fail-operational has been used in place of fault-operational, but the latter corresponds better to present-day accepted taxonomy.

4.2.2 Communication

Every message transmitted on the buses includes an identification of the message such that the receivers can verify that they are synchronized with the transmitter with respect to the prescheduled stream of messages. This schedule is determined off-line and known by all nodes, thus facilitating the rapid reintroduction of a temporarily unsynchronized node. In addition to the message identifier and the actual data transmitted, the message frame includes a number of fields supporting the membership agreement procedure and the detection of transmission errors and stale data.

At any point in time, the operational nodes should have a consistent view of the system status with respect to which nodes that are operational and which ones that are not. This membership agreement is supported by the inclusion of a membership field in each message. This field indicates whether or not a certain node, uniquely identified by the message number, is considered operational by the transmitting node.

Each bus interface (GBI in Fig. 9) contains a dual-port RAM, which is accessed by the node processors as well as by the communication interface. Thus, data pass via this memory in one direction for transmission and in the other for reception. The dual-port RAM is used as a wrap-around buffer and has the capacity to store all data transmitted in an entire ST (systolic base time, see Section 3.2). Whenever a processor writes to this RAM, an "update" bit in the corresponding memory cell is set. This bit accompanies the data as it is subsequently read by the communication interface, transmitted onto the bus, and stored in the dual-port RAMs of the other nodes. Since the update bit is reset in the dual-port RAM directly after the corresponding data is read from memory, it indicates whether the data is stale or fresh.

While the interpretation of the messages associated with Red processes is established a priori, Blue messages have to include an identification of the message data.

4.2.3 Clock Synchronization

Communication in a BASEMENT system is time deterministic in the sense that the start time of each message slot is fixed. As first proposed by Babaglou and Drummond [24], predefined events that are known in all nodes can be used as implicit clock readings. The deviation from the expected time of an event indicate the lead or lag of the clock in the originating computer node. In DACAPO, the communication protocol has been tailored to support this synchronization method. Nodes broadcast a message at the start of each message slot, thus providing the implicit clock reading. Clocks are then adjusted using the Daisy-Chain method [5]. With this method, all nodes adjust their clocks to comply with the start of the last message slot. To exclude readings from faulty clocks, the adjustments are omitted if the deviation exceed the tolerated clock precision. If, e.g., an averaging clock synchronization algorithm is used, all nodes have to deliver their clock readings before an adjustment can be made. With the Daisy-Chain method, clocks are synchronized on every message, and time-deterministic communication is possible even with low-cost, low-accuracy clocks. The simplicity of the algorithm also reduces the complexity of the hardware.

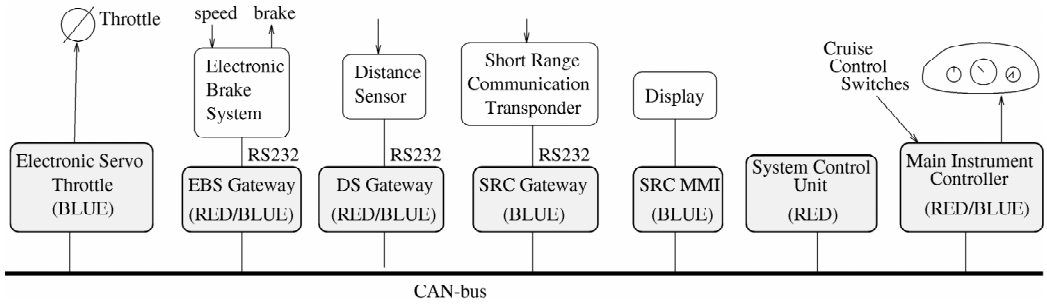


Fig. 10. The architecture of the AICC system.

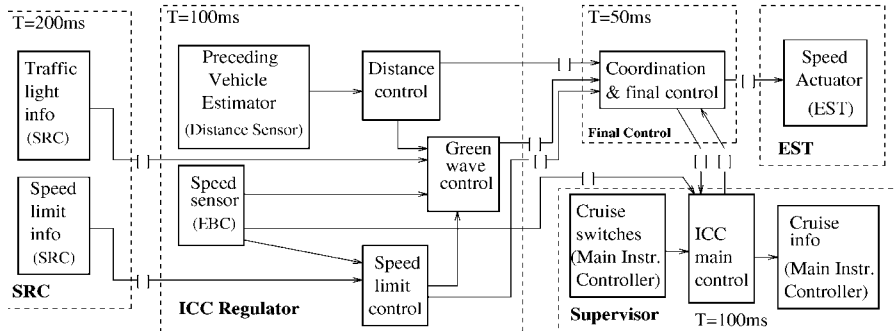


Fig. 11. Software Circuit model of the AICC system.

5 An Intelligent Cruise Controller

In this section, we provide an example of a BASEMENT system, used for implementing an Autonomous Intelligent Cruise Controller (AICC). The AICC system can receive information from road signs and adapt the speed of the vehicle to automatically follow speed limits. Also, with a vehicle in front cruising at lower speed, the AICC adapts the speed and maintains safe distance. The AICC can also receive information from the roadside (e.g., from traffic lights) to calculate a speed profile which will reduce emission by avoiding stop and go at traffic lights (a "green wave" function).

The AICC system described in this section is installed in a Saab automobile, which was one of several AICC cars demonstrated at the PROMETHEUS Board Member Meeting in Paris, October 18-20, 1994. The system is implemented on a distributed microcomputer platform following the BASEMENT concept. Fig. 10 presents the system architecture. The microcontroller nodes are connected through a CAN-bus. The communication protocols used in our system incorporate a synchronization algorithm to allow the nodes marked Red and Red/Blue to synchronously execute their off-line generated schedules. Some of the used actuator/sensor nodes are not equipped with CAN interfaces, and are therefore connected to the CAN-bus through gateways. This was a convenient way to obtain a flexible system that enabled rapid prototyping of the AICC system.

All nodes in Fig. 10, except the System Control Unit node, are mainly sensor/actuator nodes responsible for primary control of the actuators and filtering of sensor values. The Electronic Servo Throttle node maintains the desired speed and acceleration provided by the System Control Unit. The Electronic Brake node supplies the System Control Unit with speed and acceleration information (in a future implementation it will also be responsible for desired deceleration). The Distance Sensor detects and measures the distance and relative speed of vehicles in front. The SRC transponder communicates with roadside beacons to obtain information about speed limits and traffic lights. The Main Instrument Controller supplies the System Control Unit with commands from the driver, as well as presenting selected information on the main instrument. The System Control Unit handles the strategic control of the AICC system. It receives information from the other nodes, and calculates (at predetermined times in the schedule) the Electronic Servo Throttle acceleration set-point and the information to be presented to the driver. It is also responsible for supervision and failure detection.

Fig. 11 shows a simplified model of the software circuits used in the design of the AICC system (note that, connectors are omitted to simplify the drawing and that [ ] denotes communication without strict precedence). The Blue SRC process polls the SRC transponder every 200ms. This is fast enough considering the response times that are required for the type of information received. The Red ICC Regulator process calculates the desired acceleration set-points based on current and future speed limits, distance to vehicle in front, and green wave driving (if traffic light information has been received). The calculated set-points are presented to the Final control process, which decides with which value the EST should be actuated. Since the EST is a Blue process executing on a Blue only node not synchronized with the rest of the system, it requires a higher rate of parameter updates. The Supervisor process decides the overall ICC strategy based on driver requirements. It also monitors the AICC system and sends relevant information

to the Main instrument controller for presentation to the driver.

The AICC system described above follows the BASEMENT concept. It is based on a previous implementation not following the new concept. From our experiences of the two AICC system implementations, we conclude that there are definite advantages of using BASEMENT. Some of the reasons for this are

- that the synchronization provided by BASEMENT decreased the communication bandwidth requirements, since the delays in the regulator loops are under precise control and since the synchronization makes over sampling unnecessary, and
- that the understandability of the system behavior has increased, due to the clear and well defined precedence relations between different processes and software circuits.

6 Discussion

One of the major overriding considerations in the development of BASEMENT has been the question of safety. Vehicle manufacturers must place this high on the requirements list.

When safety critical computer-based systems are to be produced, specific properties of the system and surrounding engineering process activities must be scrutinized in detail. There are a large number of known and buried risks in all aspects of the underlying engineering activities and in their interrelationships. Consequently, risk analysis, reduction, and minimization of all aspects are vital. As an important step toward risk minimization, a holistic view has been taken. The need for a holistic philosophy, as well as the impact of not having such a philosophy, has been described by Lawson [25], [26].

With respect to the holistic view, risk reduction and minimization in computer-based systems is accomplished by a combination of various means: Via robust architectures and design, via verifiable specifications, via thorough hazard analysis and hazard elimination, via the incorporation of safety mechanisms in software and hardware, via a well controlled engineering processes, via quality control measures, via qualified personnel selection, via control of methods and tools, to name the more prominent.

It is important to observe that not only the product, but also the means of developing and supporting the product, must be minimized in respect to risk. Concerning Methods and Tools for safety critical systems, Lawson [27] has indicated the following:

> "Ideally, a safety critical system should be developed with a small set of well proven and well supported methods and tools that only provide the analysis and design facilities and construction mechanisms required for the safety critical Embedded Control System product (no more, no less). The risks associated with varying from this ideal position must be critically analyzed."

It is our contention that the approach taken in BASEMENT leads to a requirement for a small set of well-focused methods and tools, while satisfying the safety requirements, as well as the various other requirements, such as cost-effectiveness and short lead times.

We also claim that a supporting real-time kernel (such as Rubus described in Section 4.1), due to the straightforward application development mechanisms employed, can be minimized thus leading to a more robust and manageable solution to the dynamic run-time aspects.

The approach selected for the Red and Blue Services is based upon relevant developments in the fields of real-time systems and distributed processing.

As a starting point for the Red Service, the project exploited the concepts of Cy-Clone (see [28], [29]). Cy-Clone addresses, in a holistic-resource-adequate manner, behavior, mapping, and resource structure issues. The notion of using a hardware paradigm (software circuits) for software development has its origin in this work.

The concepts upon which Cy-Clone are based have been successfully applied in other "motion control" applications; for example, in the ATC (Automatic Train Control) system developed by Standard Radio and Telefon AB in the late 1970s for the Swedish National Railways (SJ) (see [25]). Further, we find somewhat related solutions in many time and safety critical aircraft and space applications. An early example is the SIFT (Software Implemented Fault Tolerance) computer developed at SRI International (see [30]).

Through distributed processing, the goals of providing resource adequacy (sufficient parallel processing and communication capacity) is accomplished. Further, a firm basis is provided for the implementation of fault tolerant solutions. In this regard, the project gained significant insight by examining the MARS (Maintainable Real-Time System) developed at the University of Vienna [31].

The advantages of time-driven over event-driven systems, in respect to predictability and many of the other essential abilities, have been indicated by Kopetz [32]. Time-driven processing, coupled with resource adequacy, provides for a reduction of mapping complexity (allocation and scheduling of processing and communication) and results in deterministic predictable solutions.

For addressing the Blue Service, the project had the advantage of the participation of Arcticus, who has had many years of experience in delivering event driven real-time kernels (see [33]). Further, ideas concerning the distributed nature of event-driven systems were derived from several sources, including [34].

The notion of using a leveled approach to the development of control systems has been inspired by the robotics work of Brooks [35] at MIT. This notion has been proven in practice in the development of the Saab Trionic Motor Management System as reported in [3].

The development of software circuits by defining POPs (preprogrammed domain relevant application operations) has been described by Lawson [36].

The ideas concerning fault tolerant distributed computing have their origin in the work at Chalmers Technical University. Torin [37] has characterized essential properties for automotive electronics.

Reliability analysis is an essential part of the design of any safety-critical system. A methodology for reliability analysis of complex repairable fault-tolerant systems has also been developed within the VIA project. This technique allows analytic approximate expressions for the reliability

of an architecture to be obtained. The methodology is described in [38].

During the project, several other works in the area of distributed real-time systems have been examined and have influenced the thinking of the Team Basic group, for example, the work on the Spring Architecture at the University of Massachusetts [39].

The subject of distributed real-time systems is currently the object of research and development in many academic research projects, as well as in advanced industrial projects. There are several points of view in regard to achieving a suitable solution. There has been a tendency to focus on the communication aspects (hardware, protocols, and schedules) for distributed real-time environments. While these aspects are important and interesting results have been attained (see, for example, Tindell and Clark [40]), they represent one aspect of the goals of producing safe, reliable, cost-effective real-time solutions. Attention must be given to many other issues, as brought to the forefront in the BASEMENT concept.

7 Conclusions

We have presented the BASEMENT distributed real-time architecture and methodology. The architecture provides a framework for building distributed multiplexed vehicle control systems capable of satisfying the many demanding requirements of future vehicular systems. The design methodology and tools provide engineers with a simple application oriented interface and support for generation of efficient and reliable implementations. The main strength of BASEMENT is, however, its holistic view: Both architecture and methodology are encompassed.

The concept has been explored via a number of prototype system developments, including the Rubus, DACAPO, and AICC systems presented in this paper. BASEMENT has also had a catalytic effect on Swedish Real-Time Systems research and development. Follow-up projects include:

- Participation in the European X-by-wire project (Chalmers and Mecel). X-by-wire aims at developing techniques and tools that make it possible to build vehicles without mechanical backup systems, even for safety critical functions such as braking and steering.
- ASTEC-RT, a project within the national Advanced Software Technology center of excellence program (Uppsala and Mecel) aiming at further development of methods and tools for automotive software development.
- design of a coprocessor implementation of a kernel supporting mixed execution of Red and Blue BASEMENT processes [41] (Mälarden).

References

[1] J.G. Rivard, "The Self Driving Car," *The Sab-Scania Technical J.*, 1987.

[2] H. Hansson, H. Lawson, M. Strömberg, and S. Larsson, "BASEMENT a Distributed Real-Time Architecture for Vehicle Applications," *Real-Time Systems*, vol. 11, no. 3, pp. 223-244, Nov. 1996.

[3] H. Lawson, B. Nilsson-Almstedt, and M. Strömberg, "Application Function Development for Multiplexed Automotive Control Systems," *Proc. Vehicular Technology Conf. '94*, pp. 1,093-1,097, Stockholm, June 1994.

[4] H.W. Lawson, "Application Software Development Methodology for Basement Platforms," Technical Report ProVIA-93602, Lawson Konsult AB, 1994.

[5] H. Lönn and R. Snedsbøl, "Synchronisation in Safety-Critical Distributed Control Systems," *Proc. IEEE Int'l Conf. Architectures and Algorithms for Parallel Processing*, Brisbane, Australia, 1995.

[6] "Road Vehicles—Interchange of Digital Information—Controller Area Network (CAN) for High Speed Communication," ISO/DIS 11898, Feb. 1992.

[7] O. Bridal, L.-Å. Johansson, and R. Snedsbøl, "On the Design of Communication Protocols for Safety-Critical Automotive Applications," Technical Report ProVIA-93406, Dept. of Computer Eng., Chalmers Univ. of Technology, Göteborg, Sweden, 1993.

[8] N.C. Audsley, A. Burns, and A.J. Wellings, "Deadline Monotonic Scheduling Theory and Application," *Control Eng. Practice*, vol. 1, pp. 71-78, 1983.

[9] H. Hansson and M. Sjödin, "An Off-Line Scheduler and System Simulator for the BASEMENT Distributed Real-Time Systems," *Proc. 20th IFAC/IFIP Workshop Real-Time Programming (WRTP '95)*, P. Laplante and W. Halang, eds., Nov. 1995.

[10] C. Eriksson, K.-L. Lundbäck, and H. Lawson, "An RTOS Integrated with an Off-Line Scheduler," *Proc. IFAC Workshop Algorithms and Architectures for Real-Time Control*, Ostende, Belgium, May 1995.

[11] C. Eriksson and K.-L. Lundbäck, "Rubus OS Real-Time Operating Systems, Tutorial," technical report, Arcticus Systems AB, 1996.

[12] "IEEE STD 1003.1b-1993," IEEE, ISBN 1-55937-375-X, July 1994.

[13] S. Poledna, "Replica Determination in Fault Tolerant Real-Time Systems," PhD thesis, Technishe Universität Wien, Institut für Technische Informatik, 1994.

[14] J. Reisinger, "Time Driven Operating Systems—A Case Study on the MARS Kernel," technical report, Institut für Technische Informatik, Technischen Universität Wien, 1992.

[15] P.D.V. van der Stok and A. Engel, "Shared Data Concepts for Dedos," *Proc. 10th IFAC Workshop Distributed Computer Control Systems*, H. Kopetz and M.G. Rodd, eds., Semmering, Austria, Sept. 1992, vol. 3, *IFAC Workshop Series*, Pergamon Press.

[16] H. Thane, "Distributed Real-Time Clock Synchronisation on the Can Bus," master's thesis, Uppsala Univ., Mar. 1995.

[17] C. Eriksson, M. Gustafsson, and H. Thane, "A Communication Protocol for Soft and Hard Real-Time Systems," *Proc. Eighth Euromicro Workshop Real-Time Systems*, pp. 187-192, 1996.

[18] C. Eriksson and K. Sandström, "The Translation of an Application Configuration to a Runable Application by Utilising a Pre Run-Time Scheduler," Technical Report CUS95RR04, Dept. of Real-Time Computer Systems, Mälarden Univ., Västerås, Sweden, 1995.

[19] C. Eriksson, R. Hassel, and K. Sandström, "The RRT Off-Line Scheduler," Technical Report CUS94RR04, Dept. of Real-Time Computer Systems, Mälarden Univ., Västerås, Sweden, 1994.

[20] G. Fohler, "Flexibility in Statically Scheduled Hard Real-Time Systems," PhD thesis, Technishe Universität Wien, 1994.

[21] K. Ramamritham, "Allocation and Scheduling of Complex Periodic Tasks," *Proc. 10th Int'l Conf. Distributed Computing Systems*, pp. 108-115, 1990.

[22] J. Xu, "Mulitprocessor Scheduling of Processes with Release Times, Deadlines, Precedence, and Exclusion Relations," *IEEE Trans. Software Eng.*, vol. 19, no. 2, pp. 139-154, Feb. 1993.

[23] O. Bridal, L.Å. Johansson, J. Ohlsson, M. Rimén, B. Rostamzadeh, R. Snedsbøl;, and J. Torin, "DACAPO: A Dependable Distributed Computer Architecture for Control of Applications with Periodic Operation," Technical Report no. 165, Dept. of Computer Eng., Chalmers Univ. of Technology, Göteborg, Sweden, 1993.

[24] Y. Babaglou and R. Drummond, "(Almost) No Cost Clock Synchronisation," *Proc. 17th Ann. IEEE Int'l Symp. Fault-Tolerant Computing (FTCS-17)*, Pittsburgh, Pa., pp. 42-47, June 1987.

[25] H.W. Lawson, "Philosophies for Engineering Computer-Based Systems," *Computer*, vol. 23, no. 12, pp. 1,859-1,874, Dec. 1990.

[26] H.W. Lawson, *Parallel Processing in Industrial Real-Time Applications*. Prentice Hall, 1992.

[27] H.W. Lawson, "Assessment of Safety Critical Embedded Control Systems ('A Safety Case Approach')," *Proc. Software Technology*

Conf. (STC '95), U.S. Dept. of Army, Navy, and Air Force, Salt Lake City, Ut., Apr. 1995.
[28] H.W. Lawson, "Cy-Clone—An Approach to the Engineering of Resource Adequate Cyclic Real-Time Systems, Real Time Systems," *Real-Time Systems—The Int'l J. Time-Critical Computing Systems*, vol. 4, no. 1, 1992.
[29] H.W. Lawson, "Engineering Predictable Real-Time Systems: Lecture Notes for the NATO Advanced Study Inst. on Real-Time Computing," *Real Time Computing*, W.A. Halang and A.D. Stoyenko, eds. Springer-Verlag, 1992.
[30] J.H. Wensley, L. Lamport, J. Goldberg, M.W. Green, K.N. Levitt, P.M. Meliar-Smith, R.E. Shostak, and C.B. Weinstock, "SIFT: Design and Analysis of a Fault-Tolerant Computer for Aircraft Control," *Proc. IEEE*, vol. 66, no. 10, pp. 1,240-1,255, Oct. 1978.
[31] H. Kopetz, A. Damm, C. Koza, M. Mulazzani, W. Schwabi, C. Senft, and R. Zainlinger, "Distributed Fault-Tolerant Real-Time Systems: The MARS Approach," *IEEE Micro*, pp. 25-58, Feb. 1989.
[32] H. Kopetz, "Event Triggered versus Time Triggered," *Proc. Int'l Workshop Operating Systems of the 90s and Beyond*, vol. 563, *Lecture Notes in Computer Science*, pp. 87-101. Springer-Verlag, 1992.
[33] K.-L. Lundbäck, "The Real Time Executive for Embedded Systems O'Tool (third edition)," Arcticus Systems AB, Järfälla, Sweden, 1991.
[34] A. Goscinski, *Distributed Operating Systems*. Reading, Mass.: Addison-Wesley, 1991.
[35] R.A. Brooks, "A Robust Layered Control System for a Mobile Robot," *Artificial Intelligence at MIT—Expanding Frontiers*, P.H. Winston and S.A. Shellard, eds. MIT Press, 1990.
[36] H.W. Lawson, "Application Machines—An Approach to Realizing Understandable Systems," *The Euromicro J.*, vol. 35, nos. 1-5, pp. 5-10, 1992.
[37] J. Torin, "Dependability in Automotive Electronics Requirements, Directions and Drivers," Technical Report 112, Dept. of Computer Eng., Chalmers Technical Univ., Gothenburg, 1991.
[38] O. Bridal, "A Methodology for Reliability Analysis of Fault-Tolerant Systems with Repairable Subsystems," *Proc. IMA Conf. Mathematics of Dependable Systems (MDS 95)*, Sept. 1995.
[39] J.A. Stankovic, "The Spring Architecture," *Proc. Second Euromicro '90 Workshop Real-Time Systems*, pp. 104-113, 1990.
[40] K. Tindell and J. Clark, "Holistic Schedulability Analysis for Distributed Real-Time Systems," *Microprocessing and Microprogramming*, vol. 40, pp. 117-134, 1994.
[41] J. Stärner, L. Lindh, J. Adomat, and J. Furunäs, "Scheduling Coprocessor in Hardware for Single and Multiprocessor Real-Time Systems," Dept. of Real-Time Computer Systems, Mälarden Univ., Västerås, Sweden, submitted for publication, 1996.

Hans A. Hansson received an MSc degree in engineering physics, a licentiate degree in computer science, a BA degree in business administration, a doctor of technology degree in computer science from Uppsala University, Sweden, in 1981, 1984, 1984, and 1992, respectively. He is currently department chairman and senior lecturer in the Department of Computer Systems, Uppsala University, but was previously a researcher at the Swedish Institute of Computer Science in Stockholm, Sweden. His research interests include timed and probabilistic modeling of distributed systems, real-time system design, scheduling theory, distributed real-time systems, and real-time communications networks. He is president of the Swedish National Association for Real-Time and coordinator for ARTES, a national Swedish strategic real-time initiative.

Sven Larsson received an MSc degree in electrical engineering from Chalmers University of Technology, Göteborg, Sweden, in 1988. He has been employed by Mecel AB, Göteborg, Sweden, as a systems engineer since 1988 and is working with distributed embedded real-time systems for the automotive industry.

Harold W. Lawson received the bachelor of science degree from Temple University (Philadelphia, Pennsylvania) and the PhD degree from the Royal Technical University, Stockholm. He has been active in the field of computing since 1958, with broad international experience in industrial and academic environments. He is experienced in many facets of computing and computer-based systems, including software engineering, computer architecture, real-time, programming languages and compilers, operating systems, various application domains, as well as computer related education and training.

During his industrial career, he has contributed to several pioneering efforts in hardware and software technologies at Univac, IBM, Standard Computer Corporation, and Datasaab. He has held permanent and visiting professorial appointments at several universities, including Polytechnic Institute of Brooklyn, University of California, Irvine, Universidad Politecnica de Barcelona, Linköping University, Royal Technical University, University of Malaya, and Keio University.

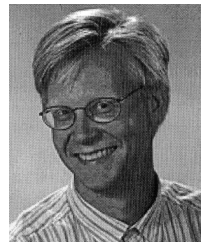
He has performed consulting and/or presented seminars for more than 50 corporations and seminars at more than 60 universities and colleges in North America, Europe, and the Far East. His publications include several books, contributed chapters, and more than 80 technical contributions.

Dr. Lawson is a fellow of the IEEE, fellow of the ACM, ACM National Lecturer, and IEEE European Distinguished Visitor. He was a founding member of SIGMICRO, EUROMICRO, and the IEEE Computer Society Technical Committee on the Engineering of Computer Based Systems.

Olof Bridal received the MSc degree in electrical engineering in 1985 and the licentiate in computer engineering in 1989, both from Chalmers University of Technology in Göteborg, Sweden. He is now a PhD candidate in the Department of Computer Engineering at Chalmers. Bridal's primary research interests are the design and analysis of ultra-dependable distributed real-time systems, primarily for future automotive applications.

Christer Eriksson received a BSc in mathematics from Uppsala University in 1988 and a licentiate degree from the Royal Institute of Technology, Stockholm, Sweden, in 1994 and 1997, respectively. He worked for ABB Automation from 1984 until 1988, primarily on the run-time system for ABB Master. He is currently a lecturer at Mälardalen University. His research interests are design of real-time systems, object-oriented programming, distributed architectures for real-time systems, and real-time operating systems.

Henrik Lönn is a PhD student in computer engineering at Chalmers University of Technology. He received the diploma of Imperial College in Robotics in 1993 and an MSc in automation engineering and lic. eng in computer engineering from Chalmers in 1993 and 1995, respectively. Recently, he spent six months at Volvo Technical Development working with real-time systems. He is a student member of the IEEE.



Mikael Strömberg received an MSc degree in electrical engineering from Chalmers University of Technology, Göteborg, Sweden, in 1985. He is a program manager at Mecel AB, Göteborg, Sweden, and has been employed by Mecel AB since 1986, where he is working with distributed embedded real-time systems for the automotive industry.