

Adding Timing Analysis to Functional Design to Predict Implementation Errors

Paolo Gai

Evidence Srl, Italy, pj@evidence.eu.com

Giuseppe Lipari, Marco Di Natale, Nicola Serreli

Scuola Superiore S. Anna, Pisa, Italy, [[@sssup.it">lipari,marco,serreli](mailto:lipari,marco,serreli)][@sssup.it](mailto:lipari,marco,serreli)

Luigi Palopoli

University of Trento, Trento, Italy, palopoli@dit.unitn.it

Alberto Ferrari

PARADES GEIE, Rome, Italy, aferrari@parades.rm.cnr.it

Copyright © 2007 SAE International

ABSTRACT

The classical V-cycle methodology for the design of embedded automotive systems is typically implemented by a sequence of steps, from a functional specification down to the implementation at the programming level with the support of an RTOS. The validation of the design is a complex task that consists of analyzing and verifying by testing both functional and non-functional requirements. An important subset of non-functional requirements consists of timing constraints. Implementation must be checked against any violation of the latency and schedulability constraints; otherwise the functionality of the entire system could be severely compromised. Unfortunately, even in state-of-the-art processes, this step is not supported by adequate methods and tools. Subsequently, the process is error-prone and subject to implementation errors, and it is very difficult to generate derivative designs.

In this paper, we propose the use of real-time scheduling theory as a formal underpinning for a design process in which the timing behavior is considered from the earliest phases of the development.

To support the designer in this phase, we have built the RT-Druid design environment, which extends the scope of traditional (and existing) schedulability and timing analysis tools in many directions. First, RT-Druid captures the mapping of the functional components of the system to the concurrent threads implementing them, and to the hardware/software platform, allowing one to

precisely trace such design decisions as the allocation of functions to a thread, and to identify potential problems in thread scheduling, communication and synchronization. Second, RT-Druid provides schedulability analysis together with sensitivity analysis, showing how variations of the thread parameters affect the response times and the schedulability of the system. Finally, RT-Druid provides support for application modes to analyze systems with dynamic workloads.

The RT-Druid design environment is integrated into the Eclipse open development framework and allows easy integration with third party tools. It supports the OSEK/VDX standard with import/export of OIL specifications and the generation of configuration code for Evidence's ERIKA Enterprise RTOS.

INTRODUCTION

The design and implementation of embedded systems (like automotive controllers) typically follows the classical V-Cycle design methodology (shown in Figure 1). In the top-down design process, provided with the initial set of requirements, control engineers design the control algorithms using such functional specification tools as Matlab/Simulink or ASCET and validate them, usually by simulation. Then, software engineers refine the functional specification into an implementation to be executed on the target platform under the control of an RTOS. At this stage, the functional components developed in the previous stage are mapped onto real-time threads. The validation of this step requires

checking the correctness of the timing requirements by extensively testing the implementation on the target.

Often, this software implementation phase is driven by the experience of the developer, rather than by a well-founded and codified methodology.

Therefore, such decisions as the number of threads, or as which functional block must be assigned to which thread, are carried out by trial and error on target implementations and/or prototypes, resulting in very expensive and tedious design cycles. In addition, such a stage of the design process is not well supported by automated tools and, in most cases, very little of it remains in the process history and documentation. Thus, the knowledge and experience remains with the developers rather than becoming part of the company knowledge base.

If timing errors or unsatisfactory performance are detected during the testing period, a chain of engineering changes is initiated that entails many iterations between the control and architectural design. In particular, one can search for the best trade off between implementation complexity and schedulability (or time performance) and can tune design parameters, including system resources and (when possible) activation rates.

Moreover, in case a derivative of the current design must be evaluated, the project manager and the designers may face difficulties in predicting whether the new algorithmic solutions fit in the current hardware and software (task) architecture. Feedback is provided and corrective actions are taken only very late in the design flow. As an extreme situation, the derivative of the current design can be impossible to build.

We propose the use of schedulability analysis theory, supported by a design tool, to (partially) fill in this gap. Schedulability analysis enables formal validation of the timing behavior of a system of concurrent threads. In this context, the word “formal” is not referred to the use of model checkers or theorem provers based on the specification of the system as a collection of timed automata (or of similar abstractions) and of its desired properties as a set of logical predicates [23]. On the contrary, we refer to the use of analytical results from the real-time theory to evaluate the schedulability of a system (i.e., its ability to execute meeting its timing constraints) in the worst case. Indeed, given the activation rates of the thread, their execution time profiles, their synchronization properties, and their timing constraints, real-time scheduling is a powerful tool to predict the timing behavior of the system under the worst-case scenario. The price to be paid with respect to formal methods like the ones used in the UPPAAL tool [23], is the introduction of conservative assumptions. On the other hand, the method we advocate is much more scalable (since it is based on closed form computations rather than on exhaustive exploration of the state-space).

Moreover, recent advances on sensitivity analysis allow to predict the variation of the system response times against changes of the task attributes, resulting from uncertainty in their values. In later stages, during the iteration cycles in the software implementation stage, sensitivity analysis can also provide valuable hints on how to modify the system parameters to improve the design.

A fundamental part of this approach is the derivation of timing models of the controller. To integrate the analysis into the design flow, it is necessary to provide information on the system timing properties and constraints, including the specification of the application software and of the hardware/software platform (i.e. the RTOS, the architecture of the hardware controller board, etc.).

In this paper, we present the RT-Druid toolset, a design environment based on the concept of virtual platforms and specifically developed to support the validation of the timing properties of the system. RT-Druid works at different levels. First, it allows to import the functional specification from several tools (currently from Matlab/Simulink, but we are planning support for other models, like ASCET SD) and allows the designer to annotate the model with the timing properties and constraints.

Then, the tool supports the designer in the mapping phase, when functional components are allocated to concurrent threads, and the RTOS parameters are defined (scheduling policy, threads priorities, etc.).

After mapping a functional design onto a set of threads, the designer can run the schedulability analysis tool to obtain information on the timing properties of the system under the worst-case scenario. Unlike existing tools for schedulability analysis, RT-Druid performs a sensitivity analysis on the thread parameters, giving feedback on how to modify the design to correct timing errors, such as violations of deadline constraints or simply to improve the performance. To address the complexity of current embedded systems, RT-Druid supports the definition and the analysis of different application modes (i.e. working conditions and configurations). To help the designer, RT-Druid has been integrated with tracing tools (such as the Lauterbach tool suite) to automatically import the timing profiles of the application threads. Finally, RT-Druid supports the generation of the threads and RTOS configuration for the ERIKA Enterprise RTOS (an OSEK compatible RTOS), as well as support for the compilation and linking process. Design parameters, such as ceilings for semaphores protecting shared resources and task priorities, may be automatically synthesized (according to the OSEK/VDX specification), to optimize design goals including schedulability, memory footprint, and the response times of selected threads).

All the previously described development stages are supported and automated by the tool. In particular, the information obtained from the timing analysis can be used to explore the space of the possible architecture

solutions and to find the best trade-offs among the complexity of the control algorithms, performance and cost.

The rest of the paper is organized as follows: we briefly present the typical V-Cycle methodology and the related design flow with the automatic generation of code. Then, we propose a methodology which integrates schedulability analysis into the workflow to predict timing errors and we introduce the RT-Druid tool, which implements the proposed methodology. Finally, we present a case study done in collaboration with Magneti Marelli Powertrain and we briefly describe the upcoming work on tool integration, being performed in the context of the European IST INTEREST Project.

V-CYCLE METHODOLOGY

In this section we describe the typical V-Cycle design flow and highlight the problems that software engineers are facing today.

Embedded control systems development is about producing systems or sub-systems according to a set of specifications that dictate their required functional and non-functional properties. The output of the design stage is a set of models that describes the system at different levels of abstraction. Properties and constraints are captured by mathematical formalisms.

A typical design flow of embedded software (a V-shape process) is represented in Figure 1. It can be roughly divided into two stages of fundamental importance. The first stage, at the logical level, is the domain of application developers, such as control engineers or other domain experts. The output of this phase is the model of the software application, which provides a solution to the functional requirements in terms of one or more block diagrams. Architectural solutions and timing properties are typically not part of the model.

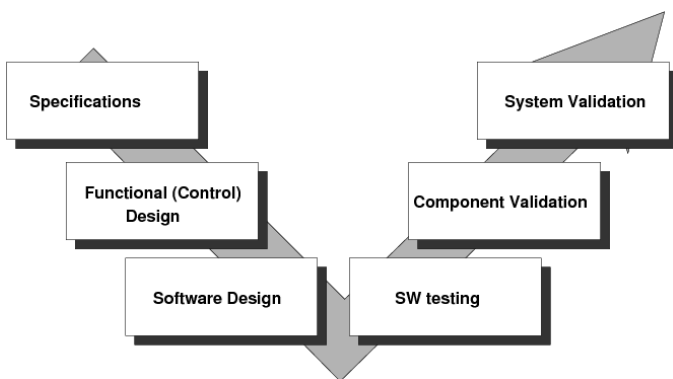


Figure 1: V-Shaped design methodology

In fact, the functional model describes the functionality and the logical organization of the components at a high level of abstraction, without including any implementation detail.

The second stage of the design process (software design) is at the architecture level, where software engineers (real-time systems experts) map the functions/components developed in the previous stage into real-time threads. Moreover, they select the RTOS configuration parameters, including the scheduling policy, the resource managers, the synchronization primitives and the threads priorities. For example, in the case of automotive systems, the RTOS will most probably be compliant with OSEK/VDX [1], a standard widely used in the automotive domain. The OSEK OS standard defines 4 classes of services (BCC1, BCC2, ECC1, ECC2) at increasing levels of complexity, for use by the designers. Another option available to developers is the use of time triggered systems, for example using the OSEK-Time API. In this paper we restrict our analysis to priority driven Operating Systems, which are the target of the case study presented at the end of this paper. After coding the application, the implementation must be validated by means of extensive software testing, both at the component level and at the system level.

Automated design tools support the first stage of the design. In the control system domain, the Matlab/Simulink tool suite is a de-facto standard in the automotive industry. This tool allows the developer to specify and simulate the behavior of the controller, validating its functional correctness. The developer can also directly generate C code for the implementation of the controller. However, the tool does not take into account the architectural properties of the system. Some proposals have been made to specify the model of the architecture in Matlab/Simulink, namely the TrueTime tool [2]. However, the tool is probably not mature yet for industrial use, and suffers from some shortcomings of the simulation engine in Matlab/Simulink. Other tools for design and code generation are ASCET-SD [3] and TargetLink by dSpace [4].

This software design stage is transparent to functionality (only provides its implementation), but it is crucial to achieve performance/cost tradeoffs with the objective of providing the best possible quality and the best possible accuracy to system functions within the timing constraints and/or the performance target. Unfortunately, this stage is only partially supported by tools. In particular, the “connection” between the functional model and the architectural model is not formalized and not supported by any tool. For example, once the functional blocks have been allocated to the threads, some logical relationships between functional blocks, such as precedence constraints or synchronization on shared data buffers, may be lost. Therefore, design changes such as swapping the order of execution of two functions, or moving the execution of one functional component into a different thread, cannot be done without the risk of introducing severe errors. As a

consequence, system developers and software designers tend to freeze the architecture configuration and implementation decisions on the basis of previous experience, and adjust the performance of the system by incremental and localized changes, like changing the priority of a thread, or possibly fine tuning the generated code by hand to improve the response time or to minimize the amount of required memory.

The process cannot be easily reversed and very often it is not clear how the initial design decisions influence the performance of the system. Since important information is lost when moving from the functional model to the architectural model, such information cannot be retrieved if we need to go back to the functional design stage. Therefore, the designers tend not to perform any loop between the two stages: once the functional design is completed and committed, it is typically unaffected by issues arising during the implementation.

The problem is that the performance and the timing properties of the system cannot be estimated until the code is executed on the target platform. Therefore, it is very difficult to foresee whether the controller will perform according to the requirements. Checking the correctness against the timing requirements requires extensive testing of the target (hardware) implementation.

As anticipated in the introduction, this phase often ends up in one or more iterations, during which the mapping between functionality and the software architecture is refined. If the performance is not satisfactory, or if some timing constraint is violated, the designer tries to trade off implementation complexity for schedulability. However, as discussed before, without a clear understanding of which are the most appropriate actions to take, there can be many iterations, which do not necessarily lead to an optimal implementation. For example, if a thread misses its deadline, it is not clear which thread should reduce its computation time (i.e. which functional block needs to be simplified) or which execution rates should be reduced to make the system schedulable.

DESIGN FLOW USING AUTOMATIC CODE GENERATION

The transition from the first stage of the V-cycle (related to the control algorithm definition and simulation), to the second stage (related to the implementation of a control algorithm as a set of instructions in a programming language), is generally complex and subject to coding errors, because it often includes the translation by hand of a control algorithm from a mathematical specification to an implementation in a programming language such as C. For this reason, most commercial design flows, such as those based on Ascet-SD [3] or Matlab/Simulink [5], provide extensive support for code generation.

These tools support the design process from the specification of a set of control algorithms to the simulation of the specified system at the logical level. Such simulation does not take into account the platform

on which the system will be implemented: no thread is specified, and all computations are performed in zero logical time (the so called synchronous model of computation). Once the simulation of the control algorithms fulfills the requirements, the engineer would like to ideally “press a button” to generate the code to be executed on the target platform. Commercial tools like Real-Time Workshop and Embedded Coder [6] or TargetLink [4] are able to perform an automated translation of the model into source code and drastically reduce the amount of functional errors due to the manual coding phase.

Although this approach works well for very simple single-rate controllers, when dealing with complex multi-rate systems with hundreds of functional blocks, the generated code is not satisfactory. Among the many problems that arise, the most important is to preserve the semantics of the model, to ensure that the behavior of the simulations (where a zero-time execution of the functions is assumed) match with the behavior of the real-time implementation (where functions take time to complete). In many cases, between simulation and real-time execution the order of access to shared memory buffers changes, with many subtle errors arising. For a more detailed description of these problems and possible solutions, the interested reader is referred to [7,8].

For these reasons, when the system is very complex (as in many automotive applications), the code generation features of such tools are only partially used. Automatic code generation is used to obtain a set of C functions corresponding to each functional item (for example, a subsystem in Matlab/Simulink). Each function is then manually “mapped” onto a real-time thread, which is periodically activated at the right frequency.

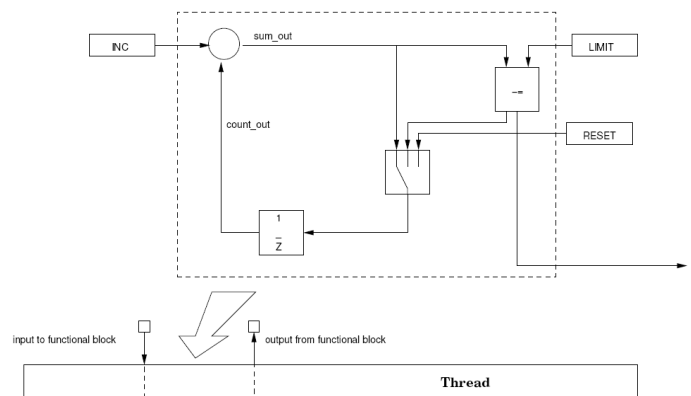


Figure 2: Mapping the code generated from the functional model to the execution tasks.

The specification of the most appropriate mapping is often a non-trivial task, which in general involves a lot of trade-offs and rules-of-thumb. In addition to deciding the number of threads and allocating functions to threads, the designer must carefully check for shared memory objects, and create appropriate synchronization mechanisms like mutex semaphores to protect critical sections of code. An error in such a critical stage of

development can introduce subtle race condition faults that may not be captured by the testing phase, because they only appear with a particular interleaving of the tasks at run time and are difficult to reproduce in a systematic way. The designer also has to select the RTOS configuration and set the priority ordering of the threads.

Unfortunately, existing design tools do not cover this delicate step of the design. One of the main advantages of using supporting tools like the one we propose is to enable designers to document the various mapping choices together with other performance results coming out from schedulability analysis and simulation.

SCHEDULABILITY ANALYSIS TO PREDICT TIMING ERRORS

The output of the mapping phase is a set of threads, each with its own scheduling parameters and timing requirements. Each thread is basically a sequence of function calls. Threads are then scheduled by an RTOS (for example, in the automotive domain, an OSEK/VDX compliant RTOS).

The developer must then evaluate and test the design. A simple, brute force testing is not appropriate. In general, even for single thread systems, a test cannot be performed exhaustively, since the number of states and combinations of inputs is unmanageable. The situation is much worse for concurrent real-time systems consisting of many threads, for the difficulty of discovering race conditions, and for the difficulty in identifying and testing the worst-case scenario (which usually occurs with a low probability).

Therefore, there is a need for performing some temporal analysis on the system along with the test. Given the worst-case execution time (WCET) of all the threads in the system, it is possible to perform a schedulability analysis, to identify the worst-case scenario, and to compute the worst-case response time of all the threads.

Unfortunately, the computation of the WCET of a thread is a difficult task. Tools exist that can derive upper bounds for the WCET of the threads from a static program analysis [9,10]. However, they usually give conservative results, especially on modern CPUs with cache and pipelines. The usual approach is to run a large number of tests, profiling the execution time of all the threads, and then use such measures to estimate a bound on the WCET. From these bounds, schedulability analysis can be performed.

Traditional schedulability analysis tools provide limited information, typically consisting of the worst-case response time of the threads. However, the dependency of the response times from the system attributes is non-linear, and it is difficult to evaluate the impact of a change in one or more parameters on the end-to-end latencies. For example, if the WCET bound of a thread has not been correctly estimated, and the actual value is

10% higher than the estimate, does the system remain schedulable?

The idea is that schedulability analysis must enable formal validation of the timing behavior of a task-based system at the highest possible level in the design flow. A fundamental part of this approach is the derivation of timing models of the controller. For this purpose, timing properties and constraints should be explicitly stated (by appropriate design annotations) and the (hardware and software) resources available for the execution of the application tasks and the resource allocation and scheduling policies must be clearly specified.

Given the complexity of the mapping decisions and of the analysis, one of our objectives is to empower automotive developers with a set of tools to integrate the specification of the mapping decisions together with a schedulability analysis framework. This integration has been performed in the RT-Druid tool, which is described in the following sections.

THE RT-DRUID TOOL

To support the designer in the mapping and schedulability analysis phases, we built the RT-Druid design environment, which extends the scope of traditional (and existing) schedulability and timing analysis tools, adding to the thread architecture design the support and capture of the mapping of the functional components (imported from ASCET-SD and Simulink) to the implementing threads. As described later, the tool allows to capture the functionality of the system (the functional blocks as specified in the functional analysis tools), the architecture of the system (the set of entities and RTOS abstractions which are instantiated in the system, such as CPUs, threads, resources), and the mapping between the functional blocks and the RTOS tasks, and between tasks and CPUs.

The functional, architectural and mapping models let the user capture the order constraints required for the correct implementation of the desired functionality, and the timing constraints independently from the thread architecture, allowing the exploration of the solution for finding the best trade-offs between functional accuracy, performance and cost.

THE DEVELOPMENT FLOW USING RT-DRUID

To separate the development flow in the different phases the RT-Druid tool can be used at different levels.

Developers can use RT-Druid as a code generator for OSEK kernels. In particular, given an OIL specification, the tool is capable of generating the configuration code for an OSEK RTOS and for ERIKA Enterprise. It also produces makefiles and debugging information compliant with the OSEK ORTI standard, as well as integrated debugging scripts for commercial debuggers like Lauterbach Trace32 [11].

During the development, users can be interested not only in generating application configurations but also in specifying the software architecture, which will be specified in terms of tasks, resources, and mapping of tasks to CPUs. In this scenario, the tool can be used to specify the task and resource properties in the system, together with the execution time profile of each task (typically the Worst Case Execution Time, WCET, is specified, see Section “WCET PARAMETER SPECIFICATION”). As a result, the RT-Druid tool can perform schedulability analysis to evaluate the timing behavior of the system. In particular, it is possible to know which of the tasks are schedulable and which are not and also by how much time should each task be reduced to make the system schedulable (which is the result of the “sensitivity analysis”, see below).

Finally, the tool not only allows the specification of the application architecture, but also of the functional blocks that are mapped into the tasks. When performing the timing analysis, the tool not only checks the schedulability, but it also provides information about the contribution of each functional block to the response time of each task. When it is necessary to reduce the response time of a task to make the system schedulable, this information can be used to decide which of the functional blocks is the likely cause of the problem. Hence, the developer can decide corrective actions such as optimizing the control algorithms to reduce their execution times, or modifying the mapping of the control algorithms to the tasks. On the contrary, if the system is schedulable, the developer is offered information about how much he/she can increase the computation time without compromising schedulability. This information can be used to take decision on derivative designs that may include new functionalities into the system or to adopt more aggressive requirements (which lead to more demanding function implementations).

THE RT-DRUID METAMODEL

The objective of the RT-Druid toolset is to help the system designer in all phases of the design and development of a complex embedded application. For this reason, the RT-Druid metamodel is divided into sections, each one containing information on one phase of the design and development process.

The metamodel is specified as an XML schema, providing a detailed view of several aspects of an application related its timing behavior.

The XML schema is designed to allow cooperation among designers working in large teams, where each team is responsible for a different part of the design process. In this case, the XML file containing the system specification can be split into different sections to allow proper processing and rights management.

The metamodel consists of the following sections:

- **Application modes.** In this section, a set of “running modes” can be specified. The basic idea of the running modes is to model and analyze the behavior of the application in different situations. For example, consider a car engine which has to be analyzed at different engine rotation speeds, typically generating different CPU loads. Most of the information which can be entered in the following phases can be specified mode-by-mode to model the fact that the behavior and configuration of an application change in response to different execution conditions. For example, a power train controller could use different control algorithms depending on the rotation speed to avoid CPU overloads.
- **Modeling the functional behavior of the application.** In this section, the designer models the behavior of the system and the application without being concerned about its implementation on a specific platform. The description of the functional model of the system is done in the FUNCTIONAL section of the XML Schema.
- **Modeling the software and hardware architecture of the system.** The designer models the characteristics of the platform on which the application will be implemented. This requires the definition of the number of computational nodes, the number and types of processors for each node, the RTOS used on each node, the number of real-time tasks, etc. The architectural model is specified in the ARCHITECTURAL part of the XML Schema.
- **Mapping the functional model to the architectural model.** In this section, the functional elements defined in the FUNCTIONAL section are mapped onto the appropriate architectural elements. For example, it is important to decide which task will perform a specific operation, and to which node/RTOS such a task is allocated. Of course, many different mapping choices are possible, and the choice of the mapping can influence the performance of the system. In the RT-Druid XML schema, the mapping is specified in the MAPPING section.
- **Back-annotation of performance measurement.** The functional models are augmented with timing execution information derived by worst (best) case execution time analysis (not provided directly by the environment) or back-annotations extracted from physical or virtual implementation. This back annotation information will be useful for the next phase, the schedulability analysis. The back-annotation information is contained in the ANNOTATION section of the RT-Druid XML schema.
- **Schedulability analysis.** This section contains the results of the off-line analysis of the system. The analysis tells the designer if the system is schedulable (i.e. if all tasks will complete in time)

under all conditions. Moreover, the analysis gives sensitivity information about the system. For example, if the system is not schedulable, the analysis tells which task will miss its deadline and what are the design changes that can fix the problem. In case the system is schedulable, it tells how much computation time is available for each task before a deadline is missed. The results of the analysis are provided by the tool in the SCHEDULABILITY section of the XML schema.

The following sections describe in detail the semantics of the models defined by the XML schema.

Application functionality

The model of the functionality is a hierarchical network of components, communicating via ports or shared variables, with precedence constraints.

The generic model of the functionality must capture an abstraction of the functional view that allows the specification of the timing behavior, and it must be compatible with the metamodels in use by the commercial tools for functional modeling and simulation, such as Simulink, ASCET, and UML 2.0.

The application functionality is defined using a set of entities divided into two categories: *basic* and *composed* objects. The first comprises the following:

- **PROC**
Used to model computation objects, such as UML 2.0 objects and components, or SIMULINK subsystems. They define a set of methods (named functions operating on internal attributes, including the output and state computation functions typical of SIMULINK blocks and operations of UML objects) as a provided interface and references to methods of used objects (other PROCs or operations on VARs, such as reads and writes) as a required interface. PROC objects typically interact through VARs (typical SIMULINK assumption), but the metamodel leaves the option of PROC processes calling each other's methods (as in UML object diagrams).
- **VAR**
Used to model protected objects, as in UML, or state variables and communication variables (port variables) in SIMULINK. They define a set of methods operating on the internal attributes of the object. In the simplest case a VAR object is a simple variable that can be read or written.
- **TRIGGER**
External objects (indeterminate, not of interest during the analysis) triggering the execution of methods in PROCs and VARs. As an example, they can be external sources of interrupts which trigger the execution of a set of PROCs.

- **EVENT**
Named entities associated with time instants (for example: *activation*, *begin* and *end*) related to the execution of the object methods.
- **PARTIALORDER**
This section may be defined to collect all the precedence constraints during the execution of the various methods of PROCs and VARs. Each precedence constraint is represented by an ORDER element, which is a pair defining a precedence relationship between two events. For example, it is possible to specify that PROC A must be completed before PROC C can start, both activated by trigger T.
- **TIMECONST**
This section contains the expression of a timing constraint, such as, for example, periodicity or the minimum time between two consecutive events, or even deadline, jitter or offset constraints between any two events.

The second category of objects is used to control complexity by implementing a hierarchy of objects. The hierarchy is defined using the abstraction of a SUBSYSTEM. A SUBSYSTEM is a structured (named) component with a provided and required interface, with an internal structure or implementation (see UML 2.0 components or SIMULINK subsystems).

Basically, the implementation of a SUBSYSTEM consists in a set of PROC, VAR, and SUBSYSTEM objects (nesting of SUBSYSTEMs is supported).

The subsystem specifies a set of methods which are required by and provided to the external environment. Therefore, complexity is managed by allowing the specification of a set of black boxes which can be connected to the rest of the system and refined in a later stage.

As shown in the previous paragraphs, RT-Druid allows the specification of the functionality of complex systems as a set of connected objects. The tool does not manage the specification of the functional behavior, but only models the properties affecting the timing behavior. RT-Druid currently supports an import feature from Simulink which is able to import a structure of PROCs, VARs, and SUBSYSTEMs from an MDL file specification.

System architecture

The model of the architecture is hierarchical and captures the topology of the car network, the number of processors for ECUs and the RTOS.

As for the functional specification, the main idea behind the specification of the application architecture in RT-Druid is to capture only the aspects of the architecture that are related to high-level timing properties, and not those required to perform a cycle- or even gate-level

accurate simulation of the hardware architecture, such as the one performed by Cadence Cierto VCC.

In particular, the hardware architecture consists of a set of ECUs connected by a bus; each ECU contains a number of CPUs (in general there can be more than one in the case of multi-core systems), and each CPU contains an RTOS.

Each RTOS provides a set of services and resources, and has a set of parameters related to the scheduling policy implemented by the operating system.

In parallel to the ECU structure, the architectural specification contains the definition of TASKs (which are the executable entities handled by the RTOS). A task basically models an OSEK Task, which can be activated in a periodic, sporadic or aperiodic way, and includes some additional properties which in general depend on the operating system (such as priorities, ceilings, number of pending activations, stacks, and so on).

The TASK object can also be used to model ISR1 and ISR2 interrupts, which are modeled as periodic or sporadic interrupts. Interrupts arriving at variable rates (e.g., interrupts related to the engine rotation speed), can be modeled by defining a separate application mode for each possible range of activation rates.

The architectural specification also contains MUTEXes, which are RTOS objects used to implement mutual exclusion, such as the OSEK Resources. The basic idea behind MUTEXes is that a VAR implementing a protected object needs a mutual exclusion service on the access to the data structure. This service is provided by the architecture-level MUTEX (see Figure 3; The dotted lines represent the mapping between the functional and the architectural specification of the system). The mapping link between the functional part (the VAR) and its implementation in the architectural part (the MUTEX) is described in the next section.

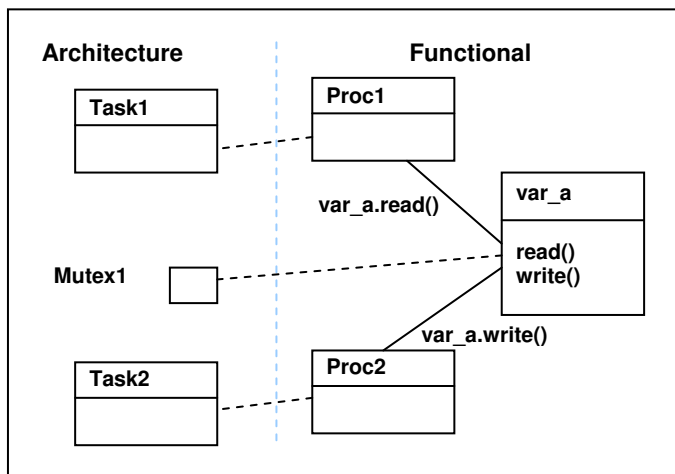


Figure 3: PROCs, VARs and MUTEXes are used to model protected objects.

The use of mutexes requires the specification of the functional part of the application. For this reason, and for the only purpose of enabling schedulability analysis on the task architecture, without the need to specify the model of the functional part, we introduced the RESOURCE object, which models a resource to be accessed in a mutually exclusive way and implemented by the operating system (see Figure 4). In this way, it is possible to specify a system architecture without being forced to specify the functional model implemented by each task.

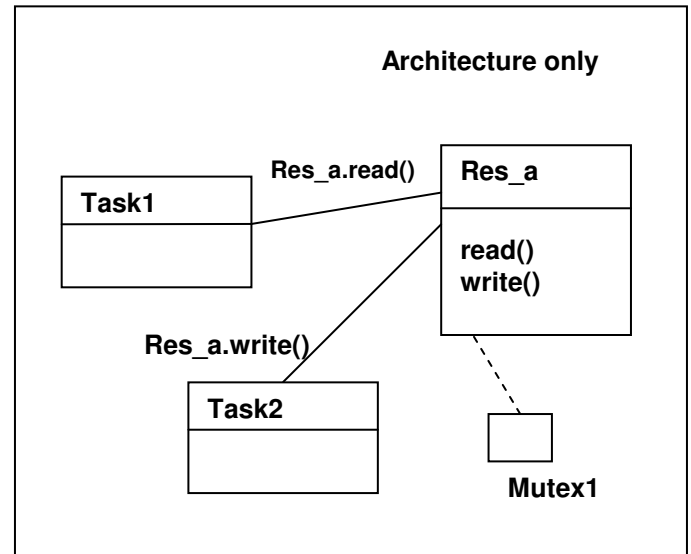


Figure 4: Usage of RESOURCES.

Mapping functionality and architecture

The mapping phase consists of allocating each functional block to a software thread and each communication variable to a communication resource of the implementation (e.g. a task's private variable, a protected shared variable, etc.). The task activation rates must be entered as parameters of the architectural models, and compliance checks are performed with the functional blocks' activation rates. If more than one functional block is mapped to a task, the order of execution must be provided during the mapping phase.

The tool supports three possible mappings:

- PROCMAP
Used to specify an object mapping the execution of the functions (methods) of a PROC in the context of a task. All the mappings have an order which specifies the sequence of the function call inside the task.
- TASKMAP
Used to specify which RTOS controls the execution of a particular task. This is especially useful in multicore or distributed systems where the developer must specify the processor to which a given task has been allocated.

- **VARMAP**
Used to specify the MUTEX which will handle the protected access to a (shared) VAR object.

The three mappings are shown in Figure 5.

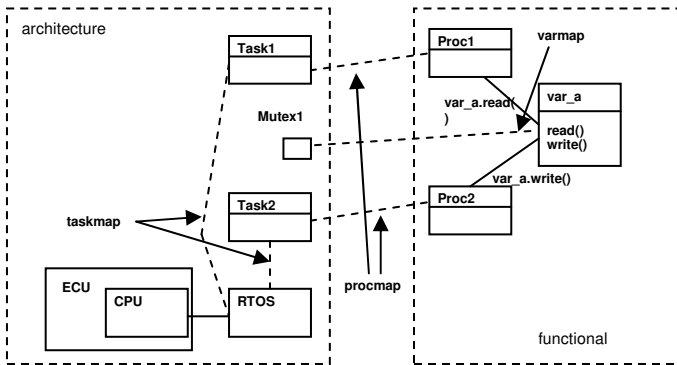


Figure 5: The three mappings needed to properly map the system functionality into the system architecture.

The RT-Druid tool currently implements a manual mapping procedure, but in future versions a graphical editor will help the designer specify the mapping, and there will be support for an automatic specification to guide the mapping process (a preliminary work in this direction, which we plan to use in a future extension of the tool, is described in [12]).

Back annotation

The back annotation phase is used to assign a value to the timing attributes of the system objects. The Annotation section contains all the timing attributes of the mapped model entities as a list of EXECTIME elements. Each EXECTIME may refer to a TASK, a PROC or a METHOD, and represents a worst case time, a best case time, or a distribution of values to allow possible use in simulations and stochastic analysis (currently not implemented).

Depending on the application development stage, the timing annotations can be derived from different sources:

- During the early exploration stages, when the application is not implemented yet, the WCET information is typically obtained from estimates made by the developers.
- At a later stage, the execution time estimates can be derived from actual measurements on the target. The system is run under specific testing conditions, and the performance of each object is measured and back-annotated. For example, each task is run with different input data, providing an extensive coverage of the branch and iteration statements, and the execution time is measured and recorded. Recording can be done by adding specific instrumentation code which reads hardware timers, as well as using hardware tracing mechanisms such as those provided by the Lauterbach Trace32 tool [11].

- Additionally, the developer could use WCET estimation tools like the ones provided by [9,10], which provide estimates of the code WCET by using abstract interpretation of the source code or real target measurements.

We are currently working towards the integration of third party tools into RT-Druid to ease the importing of the timing information into the model.

Schedulability Analysis

The schedulability analysis phase, described in detail in the following section, produces information that is stored into the XML Schema and can be saved in the project repository “as is” or it can be interpreted to produce appropriate reports such as graphics and HTML output that can be read by a standard web browser without additional tools.

In particular, the information stored in the schedulability section includes the results of the schedulability analysis for each execution mode and for each CPU. Each mode is analyzed separately, producing a “scheduling scenario”, containing a general report section in text form, a CPUSCHED section providing the results of the analysis on the particular CPU, and a TASKSCHED section providing the results of the analysis for each task.

The CPUSCHED section is a container for general CPU information like the computed CPU utilization, the speed factor (ideal factor by which the CPU speed should be increased or decreased to make the system schedulable), the utilization bound for the selected scheduling algorithm, and the results (yes/no) of the schedulability check for all the tasks on the CPU.

The TASKSCHED section contains, for each task, the computed task utilization, the CDelta factor, which is the amount of computation time that should be subtracted (if negative) or added (if positive) to the task to make it schedulable, the TDelta factor, which is the amount of time that should be subtracted (if negative) or added (if positive) to the task period to make it schedulable, the worst case response time for the task, and the results (yes/no) of the schedulability check for the task.

USING SCHEDULABILITY ANALYSIS TO PREDICT TIMING ERRORS

The objective of the RT-Druid design environment is to support the designers in exploring the degrees of freedom of the current implementation and the possible performance/cost trade-offs. This capability requires support for the selection of the schedulability analysis techniques that are available at the concurrent programming level, and the corresponding restrictions that they impose onto the logical architecture design and the mapping itself.

In order to validate this stage the designer performs schedulability analysis on his/her mapping hypothesis. Design parameters, such as ceilings for semaphores protecting shared resources and task priorities, can be specified by the designer or may be automatically synthesized according to the OSEK/VDX specification. Once the designer has specified the needed parameters, the tool gives useful information to understand the system timing behavior, such as the response time of each task including offsets analysis [13,14,18], and the stack usage optimization using preemption thresholds [15]. One of the most important elements of information is the feedback provided by the schedulability analysis [16], coupled with the possibility of defining different running modes to analyze the system in different situations, as described in the following subsections.

Using sensitivity analysis in the early stages of the design process

The schedulability analysis does not simply provide a boolean answer on the feasibility of the system with respect to the timing constraints, but also provides performance information (i.e. response times) and sensitivity analysis.

The sensitivity analysis provides a clear understanding of the critical parts of the mapped system (e.g. execution times and/or rates jeopardizing the system schedulability or possible bottlenecks).

The typical information provided by the sensitivity analysis is an estimate of the amount of time by which the execution time of each task should be reduced (or increased) to make the system schedulable (not schedulable). Moreover, we can provide an estimate of the computational speed of a CPU that would make the system under analysis schedulable (a scaling factor for the speed of the CPU being considered for the system). This type of feedback is of the utmost importance when making architectural exploration (i.e., when evaluating different hardware options).

Sensitivity analysis is based on the definition of a feasibility region in the domain of the design variables of interest, and on the capability of measuring the distance of a given system configuration, expressed in the n-dimensional space of the design attributes, from the boundary defining the schedulability condition.

For example, Figure 6 represents a very simple two task system and the feasibility region assuming the variables subject to sensitivity analysis are the computation times (the periods and deadlines are given and used to compute the boundary of the region). If the current system configuration is S' , then the system is schedulable and the distances along the axis, C_1 and C_2 give a measure of the additional computation times that are available at each task. Furthermore, the ratio $\delta_2/(\delta_1+\delta_2)$ gives the proportional factor by which the speed of the CPU can be reduced while retaining the schedulability of the task set.

Due to lack of space, we can not describe in detail all the analytical details about sensitivity analysis; the interested reader is referred to [20,22].

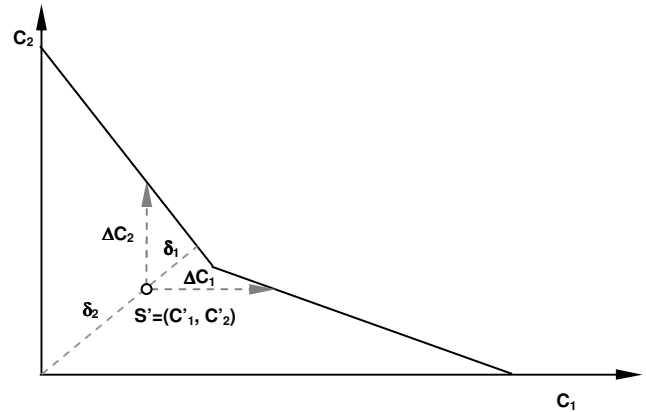


Figure 6: Sensitivity analysis in the domain of the computation times.

The sensitivity analysis is extremely important, since several iterations between mapping and schedulability analysis may be required. These iterations are not performed at the implementation level, hence blind-folded, but are rather driven by the time sensitivity information, which provides an estimation of the resource (time) budgets that should be accounted for in the next iteration to best satisfy the requirements.

In our design methodology, the sensitivity information is available for any state of the development process, providing continuous information to designers and even more importantly to project managers. When the decision to accept a derivation of some requirements or the integration of a new functionality must be taken, the managers can estimate if this derivation might have critical timing impacts very early in the design flow, drastically reducing the risk of adopting variations of the design and predicting and planning the design activities carefully.

Using modes to analyze different scheduling scenarios

To obtain better estimates from the schedulability analysis in case of dynamic behavior and/or flexible structure of the embedded controller, RT-Druid allows different configurations to be captured in terms of functional, architectural, mapping and timing information of the system: these different configurations are captured as *system modes*.

Modes are abstractions that can be used to specify a configuration of the system that is used only under particular working conditions. A typical example is a power-train controller in which, when the revolution speed of the crankshaft is faster than a given value, a particular functional element is not executed or its timing behavior is drastically changed. Another example could be an airplane controller which has different control algorithms when the airplane is taking off, flying and

landing, and these algorithms have requirements expressed by a different set of tasks and resources. For this reason, the specification of modes in the system is useful for describing the behavior and the composition of the system in all its aspects and in a unified way. Execution modes are typically implemented using OSEK OS *Application modes*.

THE RT-DRUID DEVELOPMENT ENVIRONMENT

The RT-Druid design environment has been implemented in Java, and it is integrated (as a set of additional plug-in modules) into the Eclipse open development framework. The entire design model is formally defined and represented by an XML schema derived from an Eclipse EMF specification, which controls the format of all files exchanged by the toolset, defines the elements of the functional and architecture level design, the mapping relationships, the annotations, adding timing attributes to the design objects, and the schedulability-related information.

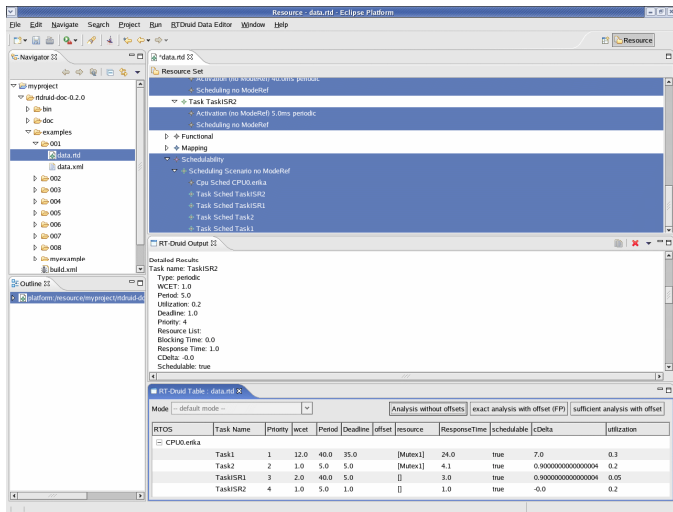


Figure 7: The schedulability analysis plug-in.

The tool also has a graphical interface which helps the designer in the schedulability analysis process (see Figure 7, which in the bottom part contains the scheduling report and a table view showing the schedulability analysis results), and in specifying the various characteristics of the OIL file specification. Future versions of the tool will also include a graphical interface to specify the mapping between functional and architectural models.

In addition to the graphical user interface, RT-Druid supports a scripting interface based on Eclipse ANT. This is particularly useful if the tool needs to be integrated with third party tools and custom text-based design flows and regression tests.

CODE GENERATION USING RT-DRUID

As an additional feature, RT-Druid handles code generation from OSEK OIL configuration files. The OSEK OIL standard is a specification of a configuration language to statically configure the RTOS entities to be used inside an application. With respect to the metamodel presented in the previous sections, an OIL file is basically an architectural specification which specifies tasks and resources, with an implicit mapping of tasks to the unique CPU defined in the OIL file.

The OIL specification is integrated in our proposed metamodel, and all the OIL declarations and definitions are mapped to specific parts of the RT-Druid data structures. The RT-Druid plug-ins allow the user to import, export, and generate code for the Evidence ERIKA Enterprise RTOS based on the OIL specifications.

Another feature of the RT-Druid code generator is the possibility to automatically generate an ORTI file specification for OSEK RTOSes. The ORTI File is a specification used to communicate the internal data structures of the kernel to debuggers, to allow a proper visualization of the information in a human readable format.

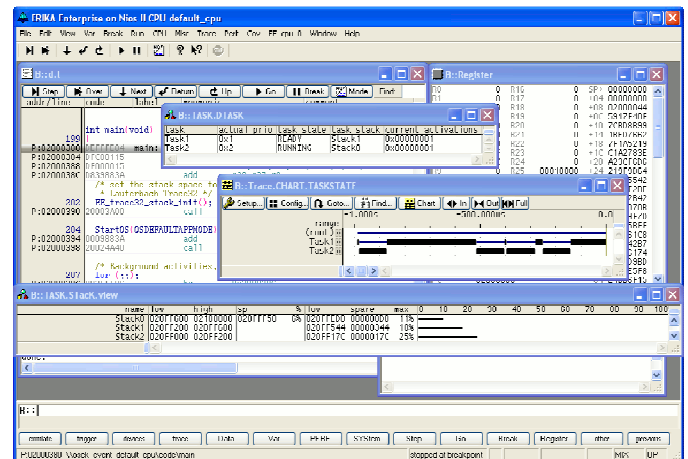


Figure 8: Lauterbach Trace32 with ORTI integration generated by RT-Druid.

For example, Figure 8 shows a Lauterbach Trace32 debugger tracing the execution of an ERIKA Enterprise application. The figure shows a set of windows containing the parameters of the software system (task priorities, resources, stacks, ...), as well as context change graphics obtained from the microcontroller trace information. Future extensions include an interpreter of the Lauterbach traces to measure the execution times of the tasks, as required in the back annotation stage (see the Back Annotation Section).

CASE STUDY: SCHEDULABILITY ANALYSIS OF FUNCTIONAL SPECIFICATIONS OF AN ENGINE MANAGEMENT UNIT

The RT-Druid toolset has been initially funded by Magneti Marelli Powertrain Spa to perform the analysis of the timing performance of code generated from Matlab/Simulink [20,21]. The tool has been integrated in the design process of an embedded power train unit following the steps described in the next paragraphs.

First, the control algorithms developed in Matlab/Simulink are imported inside the RT-Druid metamodel as a set of PROC, VAR, and TASK items. Source of external events with variable rates are modeled defining different execution modes for different rate values. This partitioning of modes allows a more accurate evaluation of the tradeoffs between computational load and control algorithm complexity at different engine rotational speeds.

At every major build of the application software, the application source code is generated from the functional specification using dSpace TargetLink, and it is run on the microcontroller connected to an engine HW simulator. Estimates of the software execution times are obtained from the Lauterbach Trace32 [11], and the maximum execution time for each task in the case study is measured from real execution traces. The execution time results are then inserted into the RT-Druid model for the corresponding execution mode.

Finally, the schedulability and sensitivity analysis is performed to get information related to the schedulability of the system at different frequencies. An HTML report is automatically produced from the XML metamodel, and it is attached to each build to provide feedback to the project lead about the schedulability of each task at different engine rotation speeds. Furthermore, the information provided by sensitivity analysis has been used to plan the allocation of new additional functionality in product extensions or derivatives.

RELATED WORK: INTEGRATION WITH THIRD PARTY TOOLS, AND THE IST INTEREST PROJECT

Integrating schedulability analysis into a complex design flow such as an automotive or aerospace design flow involves the interaction and synchronization between different tools from different vendors, which range from functional modeling frameworks to compilers, WCET analyzers, and automatic build systems.

In this complex scenario, RT-Druid is not a replacement for existing modeling and code generation tools, but a framework integrating contributions from existing tools to provide a comprehensive timing analysis and to help the designers in the development iterations. In this scenario, it is fundamental that the various tools taking part in the

design process be integrated and able to exchange and share design information.

In this context, the European IST INTEREST Project [17] is striving to overcome the current lack of integration and interoperability of tools for the development of embedded systems. The project partners (Evidence is one of them) comprise a unique group of European tool vendors that jointly address the problem of provide support for system-level design. The project will target aspects such as mapping the functional specification to the system architecture, the analysis of timing-related objectives for system and nodes, and tool integration along the V-cycle, fully covering all of the analysis, system design, module design, implementation (including certified code generation), functional testing, module testing, system testing and requirements validation.

CONCLUSIONS

Implementing modern embedded systems requires skills that range from control algorithms to design, implementation and testing with limited hardware resources. Several commercial tools, focusing on specific aspects of the V-cycle are available to the designer, but they are often not integrated. For these reasons (and others) the design of complex embedded systems requires uncommon skills, and often the results of the design process can be evaluated only at the end of the design process, especially the evaluation of the temporal performance of the system.

In this paper we proposed a methodology which addresses the timing validation process of complex embedded designs. Using the methodology, the designer can perform schedulability analysis and timing validation during the early stages of the development process, being able to drive the modification of the design and the adoption of new features by using sensitivity analysis. The methodology is supported by a framework named RT-Druid, implemented as a set of plug-ins. RT-Druid extends traditional schedulability analysis by capturing the mapping of the functional components of the system to the concurrent threads, providing schedulability analysis and sensitivity analysis with the support of application modes. Moreover, RT-Druid provides code generation for OSEK-based systems using the OIL language and producing the ORTI kernel awareness information.

As future work, we will work to extend the framework integrating third party tools in the context of the INTEREST IST Project, we will work towards improving the graphical interface to simplify the specification of the mapping between the functional and architectural specification, and we will extend the methodology to distributed systems to evaluate end-to-end timing issues.

ACKNOWLEDGMENTS

The authors would like to acknowledge G. Gaviani, W. Nesci, G. Reggiani, G. Gentile and C. Gabellini from Magneti Marelli PowerTrain Spa for the support and the feedback during in the implementation and the testing of the RT-Druid toolset.

This work has been funded by the Commission of the European Communities under contract 033661 (INTEREST Project).

REFERENCES

1. OSEK/VDX Standard, <http://www.osek-vdx.org>.
2. Dan Henriksson, Anton Cervin, Karl-Erik Årzén: "TrueTime: Real-time Control System Simulation with MATLAB/Simulink". In Proceedings of the Nordic MATLAB Conference, Copenhagen, Denmark, October 2003. Also on <http://www.control.lth.se/truetime/>
3. ETAS GmbH. Ascet-SD. <http://www.etas.de>.
4. dSPACE Inc. Targetlink. <http://www.dspace.de/>.
5. The Mathworks. The mathworks simulink and stateflow. <http://www.mathworks.com>.
6. Matworks Embedded Coder. <http://www.mathworks.com/products/rtwembedded/>
7. Scaife, N., and Caspi, P. Integrating model-based design and preemptive scheduling in mixed time- and event-triggered systems. In Euromicro conference on Real-Time Systems (ECRTS'04) (2004).
8. Semantics-preserving and memory-efficient implementation of inter-task communication under static-priority or EDF schedulers. S. Tripakis, C. Sofronis, N. Scaife and P. Caspi, 5th ACM Intl. Conf. on Embedded Software (EMSOFT'05).
9. Absint aiT. <http://www.absint.com/ait/>
10. Rapita Systems ltd, Rapitime, <http://www.rapitasystems.com>
11. Lauterbach Trace32, <http://www.lauterbach.com>
12. Cesare Bartolini, Giuseppe Lipari, Marco Di Natale, "From functional blocks to the synthesis of the architectural model in embedded real-time applications", Proceedings of the Real-Time and Embedded Technology and Applications Symposium (RTAS 05), March 7-10 2005, San Francisco, California.
13. J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In Proceedings of the IEEE Real-Time Systems Symposium, December 1989.
14. C.L. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in a hard-realtime environment. Journal of the Association for Computing Machinery, 20(1), 1973.
15. Paolo Gai, Giuseppe Lipari, Marco Di Natale, "Stack size minimization for embedded real-time system on-a-chip", Design Automation for Embedded Systems, volume 7, nos. 1/2 sept 2002, Kluwer Publisher.
16. Enrico Bini and Giorgio C. Buttazzo. The space of rate monotonic schedulability. In Proceedings of the 23rd IEEE Real-Time Systems Symposium, pages 169–178, Austin, Texas, December 2002.
17. INTEREST IST Project, <http://www.interest-strep.eu/>
18. Rodolfo Pellizzoni and Giuseppe Lipari, "Feasibility Analysis of Real-Time Periodic Tasks with Offsets", Real-Time Systems, vol. 30, No. 1-2, May 2005
19. A. Ferrari, G. Gaviani, G. Gentile, L. Romagnoli, S. Monti, and M. Beine. Automatic Code Generation and Platform Based Design Methodology: An Engine Management System Design Case Study. SAE 2005 Transaction Journal of Passenger Cars - Electronic and Electrical Systems, (2005-01-1360), march 2006.
20. Enrico Bini, Marco Di Natale, Giorgio C. Buttazzo, Sensitivity Analysis for Fixed-Priority Real-Time Systems, Proceedings of the 18th Euromicro Conference on Real-Time Systems, Dresden, Germany, July 2006.
21. Paolo Gai, Marco Di Natale, Giuseppe Lipari, Alberto Ferrari, Claudio Gabellini and Paolo Marceca, A comparison of MPCP and MSRP when Sharing Resources in the Janus Multiple Processor-on-a-Chip Platform, Proceedings of the RTAS '03.
22. Enrico Bini, Marco Di Natale, Giorgio C. Buttazzo, Sensitivity Analysis for Fixed-Priority Real-Time Systems Tests, to appear on the Journal of Real-Time Systems, 2007.
23. Johan Bengtsson, Kim G. Larsen, Fredrik Larsson, Paul Pettersson and Wang Yi., Uppaal - a Tool Suite for Automatic Verification of Real-Time Systems. In Proceedings of the 4th DIMACS Workshop on Verification and Control of Hybrid Systems, New Brunswick, New Jersey, 22-24 October, 1995.