Are COTS suitable for building distributed fault-tolerant hard real-time systems*?

Pascal Chevochot, Antoine Colin, David Decotigny, and Isabelle Puaut

IRISA, Campus de beaulieu, 35042 Rennes, France

Abstract For economic reasons, a new trend in the development of distributed hard real-time systems is to rely on the use of Commercial-Off-The-Shelf (COTS) hardware and operating systems. As such systems often support critical applications, they must comply with stringent real-time and fault-tolerance requirements. The use of COTS components in distributed critical systems is subject to two fundamental questions: are COTS components compatible with *hard* real-time constraints? are they compatible with fault-tolerance constraints? This paper gives the current status of the HADES project, aiming at building a distributed run-time support for hard real-time fault-tolerant applications on top of COTS components. Thanks to our experience in the design of HADES, we can give some information on the compatibility between COTS components and hard real-time and fault-tolerance constraints.

1 Introduction

Real-time systems differ from other systems by a stricter criterion of correctness of their applications. The correctness of a real-time application not only depends on the delivered result but also on the time when it is produced. Critical applications (*e.g.* flight control systems, automotive applications, industrial automation systems) often have *hard* real-time constraints: missing a task deadline may cause catastrophic consequences on the environment under control. It is thus crucial for such systems to use *schedulability analysis* in order to prove, before the system execution, that all deadlines will be met. Moreover, critical applications exhibit fault-tolerance requirements: the failure of a system component should not cause a global system failure.

Mainly due to economic reasons, a new trend in the development of distributed real-time systems is to rely on the use of Commercial-Off-The-Shelf (COTS) hardware and operating systems. As such systems often support critical applications (*e.g.* aircraft control systems), they must comply with stringent real-time and fault-tolerance requirements. The use of COTS components in distributed critical systems is subject to two fundamental questions: are COTS components compatible with *hard* real-time constraints? are they compatible with fault-tolerance constraints?

---

* To appear in the 2000 Workshop on parallel and distributed real-time systems, Caucun, Mexico, may 2000. This work is partially supported by the French Department of Defense (DGA/DSP), #98.34.375.00.470.75.65. Extra information concerning this work can be found in the Web page *http://www.irisa.fr/solidor/work/hades.html.*

The objective of our work is to build a run-time support for distributed fault-tolerant hard real-time applications from COTS components (processor and operating system). The run-time support is built as a *middleware* layer running on top of COTS components, and implements scheduling and fault-tolerance mechanisms. This work is achieved in cooperation with the French department of defense (DGA) and Dassault-Aviation. Our work is named hereafter HADES, for *Highly Available Distributed Embedded System*.

This paper gives the current status of the HADES project. Thanks to the experience gained in designing HADES, we can give some information on the compatibility between COTS components and hard real-time and fault-tolerance constraints. The remainder of this paper is organized as follows. Section 2 gives our preliminary results concerning the compatibility between COTS components and hard real-time constraints. Section 3 then deals with the issue of providing fault-tolerance facilities on COTS components. An overview of the prototype of the run-time support of HADES is given in Section 4.

2 COTS and hard real-time constraints

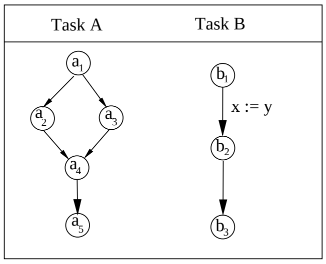
2.1 Methodology

Schedulability analysis is used in hard real-time systems to prove, before the system execution, that all deadlines will be met. Schedulability analysis must have static knowledge about the tasks (*e.g.* worst-case arrival pattern, worst-case execution time, deadline, synchronizations and communications between tasks). *Worst-case execution time analysis (WCET analysis)*, through the analysis of a piece of code, returns an upper bound for the time required to execute it on a given hardware. Our approach to be able to apply system-wide (application and run-time support) schedulability analysis to is to combine the use of a *static task model* and WCET *analysis*.

Static task model. Applications must be structured according to a task model that defines off-line:

- The internal structure of tasks: every task is described by a directed acyclic graph whose nodes model synchronization-free computations and edges model precedence constraints and data transfers between nodes.
- A set of attributes related to task execution. *Synchronization attributes* serve at expressing exclusion constraints between nodes. *Timing attributes* express temporal properties for tasks and nodes (task arrival law, deadline, earliest and latest start time). *Distribution attributes* define the site where each node of a task graph executes. *Fault-tolerance attributes* specify for each task or portion of task which replication strategy must be applied, and the requested replication degree.

Figure 1 illustrates the task model by depicting two tasks A and B, distributed on two sites named $Site_1$ and $Site_2$. Distribution attributes indicate that nodes a_1 to a_4, as well as nodes b_1 and b_2 execute on $Site_1$; nodes a_5 and b_3 execute on $Site_2$. Synchronization attributes state that a shared resource R is modified by nodes a_5 and b_3, and thus that there is an exclusion constraint



Attributes		
Task name	A	B
Synchronization	$Use(a_5, R, modified)$	$Use(b_3, R, modified)$
Distribution	$Location(a_1..a_4, Site_1)$ $Location(a_5, Site_2)$	$Location(b_1..b_2, Site_1)$ $Location(b_3, Site_2)$
Timing	$Periodic(A, 50)$ $Deadline(A, 40)$	$Periodic(B, 40)$ $Deadline(B, 40)$
Fault tolerance	$Active(a_2, 2, Site_1, Site_2)$	$Active(b_1, 2, Site_1, Site_2)$

Figure1. Illustration of the task model

between these two nodes. Timing attributes specify that both A and B are periodic with respective periods 40 and 50, and give the tasks deadlines. Finally, fault-tolerance attributes specify that nodes a_2 and b_1 must be made reliable by using active replication of degree 2 and that the node replicas must be located on $Site_1$ and $Site_2$ (see § 3).

The interest of using the above static task model is that it forces the application developer to provide all informations needed to achieve schedulability analysis.

Use of WCET analysis. To achieve schedulability analysis, the WCETs of tasks have to be known. They can be obtained through the actual execution of tasks, or through the analysis of their code (WCET analysis). We propose to use WCET analysis because it produces a *safe* estimation of the tasks WCETs. A WCET analysis tool named HEPTANE (*Hades Embedded Processor Timing ANalyzEr*) has been developed. It analyses C code and produce timing information for the Pentium processor. HEPTANE operates on two program representations: the program syntax tree, obtained through the analysis of the program source code, and the program control flow graph, obtained through the analysis of the program assembly code generated by the C compiler. User-provided annotations are used to identify the worst-case execution path for loops. HEPTANE is used to obtain the WCET of nodes of application graphs, which, by construction are blocking-free, and to obtain the WCET of the run-time support itself (operating system, and the middleware layer we have developed to provide services for fault-tolerance). In order to be suited to WCET analysis, the middleware layer has been structured into two layers: (i) a low layer, written directly in C and designed so that it never blocks; (ii) a high layer, made of tasks structured using the static task model described above, so that blocking points are statically identified.

Applying WCET analysis to systems that use COTS components raises a number of issues, which are described in the two following paragraphs.

2.2 WCET analysis and COTS hardware

COTS processor include architectural features, such as instruction caches, pipelines and branch prediction. These mechanisms, while permitting performance improvements, are sources of complexity in terms of timing analysis. A trivial approach to deal with them is to act if they were not present (*i.e.* to suppose all memory accesses lead to cache misses and assume there is no parallelism between the execution of successive instructions). However, this approach leads to largely overestimated WCETs.

In order to reduce the pessimism of WCET analysis caused by the processor microarchitecture, HEPTANE takes into account the effect of instruction cache, pipeline and branch prediction when computing programs WCETs.

- *Pipeline.* The presence of pipelines is considered by simulating the flow of instructions in the pipelines, in a method similar to the one proposed in [1].
- *Instruction cache.* Consideration of instruction cache uses static cache simulation [2]: every instruction is classified according to its worst-case behavior with respect to the instruction cache. The instruction classification process uses both the program syntax tree and control flow graph.
- *Branch prediction.* An approach similar to the one used for the instruction cache was used to integrate the effect of branch prediction (see [3] for details). Experimental results show that the timing penalty due to wrong branch predictions estimated by the proposed technique is close to the real one, which demonstrates the practical applicability of our method: from 98% to 100% of results of predictions can be known statically on a set of small benchmark programs. To our knowledge, this work is the first attempt to incorporate the effect of branch prediction on WCET analysis.

2.3 WCET analysis and COTS real-time operating systems

The first obstacle to the use WCET analysis on COTS real-time operating systems is to obtain their source code. However, at least for small kernels targeted for embedded applications, more and more operating systems come with their source code at reasonable cost.

We have undertaken a study aiming at using WCET analysis to obtain the WCETs of the system calls of the RTEMS real-time kernel, restricted for the sake of the experimentation to act on a monoprocessor architecture. We summarize below the results of the study, showing that, to some extent, the structure of the source code of RTEMS is suited to WCET analysis (for details, see [4]).

The first conclusion of this study is that using WCET analysis to obtain the WCET of the system calls of the RTEMS real-time kernel is feasible. The central part of RTEMS was analyzed in less than two months by a student having no a priori knowledge of the internals of RTEMS and WCET analysis. During the study, we discovered interesting properties of the code of RTEMS, that hopefully exist in other real-time operating systems, and that can have an influence of the construction of WCET analyzers suited to the analysis of operating systems: small number of loops, absence of recursion, small number of function calls performed using function pointers.

Most dynamic function calls (*i.e.* function calls through function pointers) could have been replaced by static ones.

We observed that finding the maximum number of iterations for 75% of the loops required an in-depth study of the source code of RTEMS. We found rather pessimistic bounds for a number of loops (dynamic memory allocation routine, scheduler). Bounds for these loops depend on the operational conditions, such as the arrival law of interrupts or the actual number of active tasks at a given priority.

3 COTS and fault tolerance constraints

3.1 Methodology

Several issues have to be dealt with in order to use COTS components (processor and operating systems) to support applications with fault-tolerance requirements. First, COTS hardware generally do not include any fault-tolerance mechanism: messages may get lost on the network and processors may crash or produce incorrect results. Thus, some form of redundancy (spatial and/or temporal) must be used to support a faulty hardware component. Second, most COTS operating systems themselves are not designed to mask the failure of hardware components (*e.g.* machine crashes, network omissions). Consequently, any error-masking mechanism must be provided as a software layer outside the operating system.

Our approach to support applications with fault tolerance constraints is to combine the use of *off-line task replication* and *basic fault-tolerance mechanisms.*

- Off-line task replication (see § 3.2) transforms the task graphs (see § 2.1) to make portions of tasks fault-tolerant through the use of replication. Off-line replication relies on a number of properties that are verified by the run-time support (fail-silence assumption, bounded and reliable communications).
- Basic fault-tolerance mechanisms are provided by the run-time support (see § 3.3) in order to verify these assumptions. Error detection mechanisms have been designed to provide the highest coverage as possible of the fail-silence assumption. A set of mechanisms (group membership, clock synchronization, multicast) altogether guarantee bounded and reliable communications.

3.2 Off-line task replication

Application tasks are made fault-tolerant through the replication of parts of their code (transformation of their structure, defined as a graph, see § 2.1). Task replication is achieved *off-line.* An extensible set of error treatment strategies based on task replication (currently active, passive and semi-active replication) is provided. Task replication is achieved thanks to a tool we have developed, which is named HYDRA [5]. It takes as input parameters the portions of code to be replicated, the replication strategy to be applied and the replication degree.

Figure 2 illustrates the application of active replication on task A taken as example in Figure 1, where node a_2 is made reliable through the use of active replication of degree 2 on $Site_1$ and $Site2$. The replication tool transforms the graph of task A through the addition of: *(i)* two replicas for node a_2 that must be made fault-tolerant (nodes a_2^a and a_2^b); *(ii)* a node that computes a consensus value for the outputs of a_2^a and a_2^b; *(iii)* new edges in the graph.

The main interest of off-line replication is that replicas can be taken into account easily in schedulability analysis, even if only portions of tasks are replicated (see [6]).

Off-line replication is correct as far as the underlying run-time support in charge of executing tasks ensures a set of properties (see [5]) like for instance the

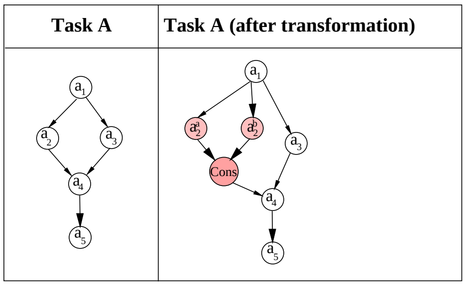


Figure2. Example of application of active replication

exe-fail-silence property, stating that the run-time support of a given site either sends correct results in the time and value domains, or remains silent forever.

3.3 Basic fault-tolerance mechanisms and COTS components

Most COTS operating systems are not designed to mask faulty hardware components. Thus, we have designed a set of fault-tolerance mechanisms, embedded in a middleware layer running on top of a COTS real-time kernel (in our prototype Chorus and RTEMS). Mechanisms fall into two categories:

- *error detection mechanisms:* these mechanisms detect (as far as possible) value and temporal errors and transform them into machine stops in order to reach the fail-silence assumption (property *exe-fail-silence:* given above).
- *fault masking mechanisms:* these mechanisms allow to mask communications faults and site crashes through the provision of fault-tolerant services: group membership service, reliable time-bounded multicast service and clock synchronization service (see [7] for details on these services). The properties of these services are of great importance to ensure *exe-timeliness* and *exe-agreement* properties. These services have been designed using the static task model of § 2.1 to ease their integration in schedulability analysis.

The design of services such as group membership, multicast and clock synchronization on COTS components did not cause any intractable difficulty. However, the problem is that all these services are entirely implemented in software on top of an existing operating system, which leads to modest worst-case performances (approximately two orders of magnitude worse than hardware-implemented solutions like the TTP/C communication chips). An estimation of the influence of these worst-case performance on the range of deadlines that can be supported has still to be achieved.

A set of error detection mechanisms has been integrated into the run-time support. To detect timing errors, the run-time support includes monitoring code to *(i)* detect deadline and WCET exceeding; *(ii)* check that tasks arrival laws conform to their expected arrival laws. These checks are possible because the run-time support is aware of semantic information about the tasks it executes, thanks to the use of a static task model. To detect value errors, the run-time support *(i)* catches every hardware exception (*e.g.* parity error, alignment error, protection violation, division by zero); *(ii)* checks if a set of implementation-dependent invariants hold. We are currently estimating the coverage of the fail-silence assumption provided by these mechanisms.

4 Experimental platform

A prototype of the run-time support of HADES has been developed (see [7] for details). It runs on a network of Pentium PCs, and has been ported on Chorus and RTEMS. It is divided into two layers: (i) a kernel, developed in C, in charge of executing tasks that are developed according to the task model of § 2.1, as well as detecting errors; (ii) a set of services, most of them implementing fault-masking mechanisms. Services are developed according to the task model of § 2.1 to be able to apply system-wide schedulability analysis.

We are currently porting an avionic application provided by our industrial partner Dassault-Aviation on top of our prototype.

5 Concluding remarks

This paper has given current status of the HADES project, aiming at building a distributed run-time support for applications with hard real-time and fault-tolerance constraints. Concerning the support of hard real-time constraints, we highly rely on the use of WCET analysis to provide information for schedulability analysis. We have shown that using this class of techniques on COTS hardware and operating system is feasible. We are currently studying the pessimism induced by the analysis, and are studying the use of WCET analysis on larger (and therefore more realistic) operating systems. Concerning fault-tolerance constraints, we have built on top of a COTS kernel predictable fault tolerance mechanisms (fault masking mechanisms such as reliable multicast and clock synchronization, error detection mechanisms to enforce the fail-silence assumption). Designing fault masking mechanisms on top of a real-time kernel did not cause any particular problem, but is not efficient due to the fact that all these mechanisms are implemented entirely in software. The impact of software-implemented mechanisms on the range of deadlines that can be supported is currently under study. We are currently evaluating the efficiency of the error detection mechanisms that have been integrated into the run-time support.

References

[1] N. Zhang, A. Burns, and M. Nicholson. Pipelined processors and worst case execution times. *Real-Time Systems*, 5(4):319–343, October 1993.

[2] F. Mueller. *Static Cache Simulation and its Application.* PhD thesis, Departement of Computer Sciences, Florida State University, July 1994.

[3] A. Colin and I. Puaut. Worst case execution time analysis for a processor with branch prediction. *Real-Time Systems*, 2000. To appear.

[4] A. Colin and I. Puaut. Worst-case timing analysis of the RTEMS real-time operating system. Technical Report 1277, IRISA, November 1999.

[5] P. Chevochot and I. Puaut. An approach for fault-tolerance in hard real-time distributed systems. Technical Report 1257, IRISA, July 1999. A short version of this paper can be found in the WIP session of SRDS'18 p. 292–293.

[6] P. Chevochot and I. Puaut. Scheduling fault-tolerant distributed hard real-time tasks independently of the replication strategies. In *Proc. of RTCSA'99*, Hong-Kong, China, December 1999.

[7] E. Anceaume, G. Cabillic, P. Chevochot, and I. Puaut. A flexible run-time support for distributed dependable hard real-time applications. In *Proc. of ISORC'99*, pages 310–319, St Malo, France, May 1999.