Genericity and Upgradability in Ultra-Dependable Real-Time Architectures

A. Wellings, Lj. Beus-Dukic, A. Burns
Department of Computer Science
University of York
York, Y01 5DD, UNITED KINGDOM

D. Powell
LAAS-CNRS
7 Avenue du Colonel Roche
31077 Toulouse 4, FRANCE

Abstract

We report on the ideas currently being developed within the European GUARDS project to develop a generic upgradable architecture for real-time dependable systems. After a brief introduction and overview of the architecture, we outline the GUARDS approach for scheduling real-time replicated computation.

1 Introduction

Most ultra-dependable real-time computing architectures developed in the past have been specialised to meet the particular requirements of the application domain for which they were targeted. This specialisation has led to very costly, inflexible, and often hardware-intensive solutions that, by the time they are developed, validated and certified for use in the field, can already be out-of-date in terms of their underlying hardware and software technology. This problem is exacerbated in some application domains since the systems in which the real-time architecture is embedded may be deployed for several decades, i.e., almost an order of magnitude longer than the typical lifespan of a generation of computing technology.

A consortium of European companies and academic partners has recently been set up to design and develop a Generic Upgradable Architecture for Real-Time Dependable Systems (GUARDS), together with an associated development and validation environment. The end-user companies in the consortium all currently deploy ultra-dependable real-time embedded computers in their systems, but with very different requirements and constraints resulting from the diversity of their application domains: nuclear submarine, railway and space systems. The overall aim of the GUARDS project is to significantly decrease the lifecycle costs of such embedded systems. The intent is to be able to configure instances of the GUARDS generic architecture that can be shown to meet the very diverse requirements of these (and other) critical real-time application domains.

2 Architecture Overview

To minimise cost and to maximise flexibility, the architecture favours the use of commercial off-the-shelf (COTS) hardware and software components, with application-transparent fault-tolerance implemented primarily by software. The architecture aims to be tolerant of permanent and temporary, internal and external, physical faults and should provide confinement or tolerance of software design faults. A three-pronged approach is being followed to reduce the cost of validation and certification of instances of the architecture: design for validation so as to focus validation obligations on a minimum set of critical components; re-use of already-validated components in different instances; and the support of system and application components of different criticalities.

Drawing on experience from systems such as SIFT [6], MAFT [5], FTPP [4] and Delta-4 [7], the generic architecture is currently defined along three axes (Figure 1): [8]:

- the channel axis: channels provide the primary hardware fault containment regions; it should be possible to configure instances of the architecture with 1 to 4 channels;
- the intra-channel or multiplicity axis: multiple resources can be provided in each channel either for increased performance and/or for use as secondary fault containment regions;

---

1GUARDS is partially financed by the European Commission as ESPRIT project no 20766. The consortium consists of three end-user companies: Technicatome (France), Ansaldo Trasporti (Italy) and Matra Marconi Space France; two technology-provider companies: Intecs Sistemi (Italy), Siemens AG Österreich PSA (Austria); and three academic partners: LAAS-CNRS (France), Pisa Dependable Computing Centre (Italy) and the University of York (United Kingdom). See http://www.cs.york.ac.uk/rts/guards/ for further details of the project.
• the integrity axis: spatial and temporal firewalls will be implemented to enforce a Biba-like integrity policy [3] to protect critical components from residual design faults in less-critical components.

As stated previously, the GUARDS architecture favours the use of commercial off-the-shelf components. However, some parts of the architecture must necessarily be purpose-designed. These are identified on Figure 1 as shaded components:

• the inter-channel communication network, needed to ensure inter-channel synchronisation and interactive consistency;

• the output data consolidation system, needed to combine redundant logical outputs into error-free physical effects in the controlled process;

• the basic operating system services for fault-tolerance, firewalling and real-time scheduling of replicated computation.

To comply with the basic GUARDS requirements of genericity and upgradability, the latter operating system services will be developed using a server-based operating system based on micro-kernel technology. The remainder of this short paper is devoted to our current ideas about what this operating system should provide in terms of generic services for scheduling real-time replicated computation.

3 Real-Time Scheduling

The approach taken to scheduling real-time applications depends on the computational model required by the application. The computational model defines the form of concurrency supported by GUARDS (for example, processes, threads, asynchronous communication, etc.) and any restriction that must be placed on application programs to facilitate their timing analysis (for example, bounded recursion). In general, GUARDS applications may conform to a time triggered, event triggered or mixed computational model.

In keeping with the GUARDS requirement of genericity, the architecture and its operating system must be capable of supporting a range of computational and scheduling models. Furthermore, the operating system must be capable of being tailored so that it only supports those primitives required by a particular application.

Three scheduling models are defined for GUARDS: cyclic, cooperative and pre-emptive. An application’s choice of computational model will depend on a variety of factors such as:

• the computational model of the application

• the performance requirements of the application,

• constraints placed on the process by certification authorities (for example, the particular method of testing or the form of safety case required),

• the ease with which the proposed application can be maintained with the chosen computational model.

Here we focus on the performance requirements.
Cyclic

If the application consists of a fixed set of purely periodic functions, it may be possible to layout a complete schedule such that the repeated execution of this schedule (i.e., it conforms to the time-triggered computational model) will cause all functions to run at their correct rate. The cyclic executive is, essentially, a table of addresses where each address points to a procedure encapsulating all or part of the code for a function. The complete table is known as the major cycle; it typically consists of a number of minor cycles each of fixed duration.

Cooperative

If an application’s requirements cannot be met by the cyclic scheduling model, the next simplest model is one based on cooperative scheduling. Here, each application function is represented as a notional thread. The threads are scheduled at run-time by an application-defined scheduler thread. Once scheduled, each application thread executes until it voluntarily gives control of the processor back to the scheduler thread. Hence, the threads are cooperating in the scheduling of the system.

The main advantages of the cooperative model are that it:

- retains the non-pre-emptive style of execution,
- allows periodic functions, which do not fit into convenient major and minor cycles, to be catered for,
- allows sporadic functions to be executed.

Its disadvantage is that there are inevitably some overheads introduced by the scheduling scheme and it is necessary to undertake some form of timing analysis to determine if all functions meet their deadlines. The actual overheads incurred will depend on the implementation technique used which, in turn, will depend on the difficulty of meeting the real-time requirements of the application.

Pre-emptive

Although the cooperative model has many advantages over the cyclic executive model, there may be some systems whose timing requirements cannot be met because of the potentially large time between a high priority thread becoming runnable and it actually running. Pre-emptive systems have the property that if a high priority thread becomes runnable and a lower priority thread is currently executing, the lower priority thread is stopped from executing and the high priority thread is executed. The lower priority thread executes again when it is the highest priority runnable thread. The main disadvantages are that pre-emptive systems require more kernel support and their behaviour is less deterministic (but fully predictable).

4 Replication

The current Preliminary Definition of the GUARDS Architecture [8] does not commit to a particular inter-channel communication (ICC) network. Here we assume that each channel has an independent inter-channel communication processor. The ICC processor of each channel can transmit signed messages over a private bus to the other ICC processors. Each channel, with its associated ICC processor and ICC bus, constitutes an independent fault containment region. Therefore, a three channel system is adequate for tolerance of one arbitrary failure in one of the fault containment regions.

We model the ICC network as an object which supports a number of links, where a link is a logical connection between the three channels. Each link has a unique identifier and can be opened for an interactive consistency agreement protocol or for voting. In the former, one channel writes a value to the link and all channels can read back from the link the value agreed between them. In the latter, all channels write a value to the link and all read back a voted result.

For an application function to be replicated, it must behave deterministically and each replica process the same inputs in the same order. At any point where there is potential for replica divergence, the channels must perform an interactive consistency agreement protocol to ensure that they take the same decisions and act on the same inputs. Then, in the absence of errors, all replicas produce identical results.

The cost of executing interactive consistency agreement and voting protocols can be significant. There is, therefore, a need to keep their use to a minimum. If we assume replica determinism, then it is only necessary to meet the following requirements:

- R1 — perform interactive consistency (IC) agreement on any replicated sensor values or a Byzantine agreement on single-sourced data.
- R2 — ensure that all replicas receive the same inputs from other replicated functions (replicated inputs).
- R3 — perform voting on any vital output.

There is a trade-off between the frequency of performing comparisons between channels and the early detection of errors and the overheads (and scheduling constraints) imposed by such comparisons.

Replication schemes for the cyclic, cooperative and pre-emptive computational models have been proposed. In summary, the cyclic schedule approach supports the three requirements as follows:

- R1: Sensor IC agreement is synchronised by the schedule itself.
• R2: Identical replicated inputs are guaranteed by the schedule itself (in the absence of errors).
• R3: Voting on vital output is synchronised by the schedule itself.

With the cooperative and pre-emptive models, there is much more uncertainty on when threads associated with the periodic and sporadic functions get scheduled (particularly, if each channel has a different number of threads). Unlike the cyclic executive scheduling, the schedule does not automatically guarantee the required synchronisation properties. The approach we are taking supports the three requirements as follows:

• R1: Sensor IC agreement is synchronised by splitting the thread which reads the sensor into two; the second part starting at an offset in time from the first.
• R2: Identical replicated inputs are guaranteed by the use of timestamps in the absence of errors following the approach taken by Barrett [2].
• R3: Voting on vital output is synchronised by splitting the thread which writes the data into two; the second part having an offset from the first.

The guarantees of hard real-time properties of applications will be undertaken using response time analysis[1].

5 Conclusion
In this short paper we have reported on the ideas being developed in the GUARDS project. The project is still in its early stages and there are many issues still to be resolved. These include: the detailed design of the inter-channel communication network and its impact on the scheduling models to be supported, the integration of a COTS micro-kernel into the software architecture, and the management of intra-channel redundancy techniques.

References