A Position on Design, Methods, and Tools for Object-Oriented Real-time Computing

A. Bondavalli [•] and F. Di Giandomenico[*]

[•]CNUCE Istituto del CNR, Via S. Maria, 36, 56126 Pisa, Italy,
[*]IEI Istituto del CNR, Via S. Maria, 46, 56126 Pisa, Italy

Real-time is a major characteristic of many systems, increasingly employed today in disparate sectors of our society. To specify and program systems, exhibiting real-time properties, a number of computing paradigms have been adopted, both explicitly defined to specify real-time behaviors and imported from other application areas with the addition of mechanisms to deal with real-time. The object-oriented paradigm, well affirmed since a few decades, belongs to the latter category. In the recent years, the OO languages have been (and are being) modified to be successfully used in an increasing variety of real-time applications.

However, it has to be reminded that the issues of real-time systems are orthogonal to those of the object oriented paradigm: real-time properties can be achieved through the object-oriented technology, as well as using other technologies or programming paradigms if proper mechanisms for dealing with timing issues are incorporated. So OO shares with other paradigms the need of proper features for coping with real-time, what may change are the specific mechanisms and the obvious restrictions that must be defined.

Another important aspect to be highlighted is that real-time systems are usually employed in application fields with demanding dependability requirements too. Actually, all systems that must fulfill requirements of a correct behavior in the time domain, require correctness also in the value domain. To be useful, a service has to be provided within its temporal envelope, but has to be also functionally correct (possibly, a degraded service is acceptable, in adverse system conditions). This implies that design guidelines and methods have to consider both aspects. Traditionally, real-time and dependability issues have evolved almost independently, and only recently effort is being put in considering both aspects in an integrated way. Apart from the necessity for the application to satisfy requirements on both sides, the integration approach allows a better coordination of the means for real-time and fault tolerance, exploiting the application of consolidated techniques in one side to solve similar problems on the other side.

For those systems supporting highly critical applications (and having timing requirements), a crucial aspect remains the validation. This means that in such cases the timeliness properties of the system must be correctly specified, implemented and, most important, guaranteed and verified. However, also in cases when real-time constraints are not so strict and vital, the ability to analyse the real-time behavior of the system remains very important at least because it heavily contributes to an effective design and implementation phases of the system lifecycle.

The easiness of managing the analyzability of a real-time system heavily depends on the degree of variability of the environment in which the system is called to operate, and the information about its run-time evolution. Two extreme classes of real-time systems can be distinguished: i) “closed” control systems, which operate in very constrained environments, and ii) “open” systems, characterized by high run-time variability and poor statically known information.

Systems in category i) are ultradependable real-time systems. They are called to operate in very critical environments, such as nuclear or avionics applications, very strong guarantees on the timing behaviour are required, and a limited margin of flexibility to the system itself is accepted. Fundamental in these systems is to be able to predict and verify the timely-correct behaviour of tasks.

Typical solutions for this class of systems are design of static systems, usually adopting a time-driven approach, and where the run-time behaviour is completely statically known. These systems have very nice properties and allow strict guarantees on tasks executions to be met and verified (high predictability), since they rely on assumptions about workload, available resources, external system conditions etc. Moreover they are usually equipped with design environments which support the specification and verification of their real-time properties.

The time-triggered paradigm (TTP) [4] fully incorporates the concepts of “closed” control system. In a time-triggered architecture, all information about the behaviour of the system, e.g., which node has to send what type of message at a particular point in time, is known a priori (at design time) to all nodes of the ensemble. All computation and communication actions are derived from the progression of a globally synchronized time base, with benefits on predictability, composability, and transparent implementation of fault-tolerance.

The GUARDS (Generic Upgradable Architecture for Real-time Dependable Systems) [5] generic architecture, recently developed in the framework of an Esprit European project, is another representative of architectures for such highly dependable real-time applications (with the additional aim of genericity, to decrease the life cycle costs). In this generic architecture, which employs physical redundancy among the measures towards dependability, a convenient interchannel communication network has been defined, which keeps in a relatively strict synchronization the multiple channels composing the architecture. Because of the ultradependable nature of the intended applications, a design method which allows (and force) the development of systems that can be submitted to schedulability analysis has been preferred to alternative methods which leave completely to the designer the responsibility to limit the tasks/threads behaviour to the appropriate semantics. The HRT-HOOD design method, which has such requisite, has been adopted; a Temporal Properties Analysis toolset (including a Schedulability Analyzer and a Scheduler Simulator) has been defined in addition, based on those available in HRT-HoodNICE. However, extensions have been also required, to cover weaknesses of the method, in particular related to the design of distributed systems; the concept of Virtual Nodes has been introduced, which allows to take into account the multi-channel structure of GUARDS (objects can be allocating to different processors within a channel) and to define spatial firewalls around objects of a given criticality. The overall structure of the software application is then extracted from the HRT-HOOD design and the related code is automatically generated. To this end, a set of mapping rules has been defined to translate the HRT-HOOD design in terms of threads implemented in a sequential programming language (which could be C or the sequential subset of ADA) and executed by a POSIX compliant microkernel. These mapping rules also include that a subset of the target languages will be considered to allow the analysis of the WCET of the tasks. For example, recursive subprogram calls, unbounded loop constructs, dynamic storage allocation and variable length array have not been allowed.

In addition to timing issues, limitations to a full object oriented approach have been also dictated by the fact that a) GUARDS typical applications are dedicated to control complicated and reactive devices, stressing the hard real-time and safety aspects rather than the management of sophisticated data; b) mechanisms such as dynamic binding and dynamic thread creation are potentially dangerous in dependable software.

Category ii) is much wider. Nowadays there is the willingness to design and operate more and more complex applications, working in dynamic distributed (also on a large scale) environments and subject to varying operating conditions.

In the design of such systems and applications much higher flexibility and dynamicity is required, obviously sacrificing predictability. A range of intermediate solutions have been and are currently sought, aiming at a tradeoff of these contrasting aspects, trying to retain as much as possible the analysability of the timing properties while admitting (considerable) dynamicity of the operative environment. Among these studies those in which we have been involved are the definition of a framework for software-implemented, adaptive fault tolerance in a real-time context [2], the work on applying tolerance to timing faults in hard real-time (dependable) systems [3], and the research on reflective use of tasks admission policies [1].

[2] proposes a programming notation, named FERT, which extends previous work in two main ways: by including features that explicitly address the real-time constraints; and by a flexible and adaptable control strategy for managing redundancy within application software modules. This redundancy-management design is introduced as an intermediate level between the system design (which may itself consist of multiple levels of design) and the low-level, non-redundant application code. Application designers can specify fault tolerance strategies independently for the individual application modules, including adaptive strategies that take into account available resources, deadlines and observed faults. They can use appropriate design notations to notify the scheduling mechanisms about the relative importance of tasks, their timing requirements and both their worst-case and actual usage of resources. Run-time efficiency can thus be improved while preserving a high degree of predictability of execution.

The work in [3] aims at preserving predictability in presence of hard real-time tasks, without requiring the knowledge of tasks WCET. It has to be reminded that the estimation of the WCET is often a difficult task; moreover, even when such an estimate is known, it can result in an excessively pessimistic value, which is only encountered by the task in rare executions, with the consequent inefficient use of the system. The novel idea in this approach, called TAFT, is that tolerance to timing faults is applied, instead of the traditional avoidance of timing violations, strictly dependent on the knowledge of the WCET. The adherence to real-time requirements is pursued by properly structuring the task in a couple (TP): a nominal part, TN, and an exceptional part, TX. T_N represents the normal computation of the task, and can admit fault tolerant execution for coping with value faults too, while T_X is an exception handling block which does not execute any application code but actions to terminate the TP in a controlled way, leaving data in a defined state and without violating the deadline. Execution time can be reserved for both tasks, but only the WCET of TX has to be known and guaranteed. The scheduler, then, accepts a TP if it can guarantee the requested time for TX, thus assuring the predictability of the system behaviour, despite the possible failure of TN because of timing failure. The TAFT scheduling approach has been effectively employed in conjunction with the object-oriented paradigm, having been inserted in an architecture for an adaptive object oriented system with integrated monitoring, dynamic execution time prediction and scheduling.

In [1], reflective systems are considered, a class of systems able to follow variations of the external environment and of their internal state and adapt their behaviour on how system resources must be best used. In this framework, an admission algorithm has been defined, to be adopted at run-time by a “reflective planner” to make the optimal choice on the workload to accept for execution. Resorting to this kind of planner, greatly simplifies the resource management; in particular, scheduling activities are markedly simplified by the overload avoidance. This algorithm, LDR, is based on the concept of “task utility”, a well known concept in real-time systems to allow a good usage of resources under a variety of adverse circumstances, as transient overloads and degrading availability of computing resources. Given a system workload, a set of admitted fault-tolerant execution strategies for the system tasks (each one with its own utility for the system), and a number of homogeneous computing resources, LDR selects the optimal “admission vector”, that is that part of the workload which optimise the expected cumulated reward achieved by the system. At run-time, the planner dynamically applies the LDR algorithm, to derive a new admission vector whenever it is necessary due to changes in the environment. The proposed planning approach shows an interesting attempt to introduce flexibility in real-time systems, still basically maintaining a static organization, with the consequent benefits on analysability and control of the timing behaviour. Although it has not been directly related with the object oriented technology, the planning approach is well support by OO: the planner can be designed as a (meta)object, with the LDR algorithm implemented as one of its methods.

Besides [1] which is more related to a supporting system level, both the proposals in [2] and [3] are concerned with the user level. They both provide means for the user to specify his fault tolerance and timeliness measures in a flexible and adaptive way. These mechanisms can easily bee incorporated in OO languages. This should be done by using the language extensions mechanisms which are often provided (e.g. UML).

References

[1] A. Bondavalli, F. Di Giandomenico and I. Mura, “An Optimal Value-Based Admission Policy and its Reflective Use in Real-Time Systems,” *The International Journal of Time-Critical Computing Systems, Kluwer Academic Publisher*, vol. 16, pp. 5-30, 1999.

[2] A. Bondavalli, J. Stankovic and L. Strigini, “Adaptable Fault Tolerance for Real-Time Systems,” in *Responsive Computer Systems: Toward Integration of Fault-tolerance and Real-time*, D. F. Ed. (Eds.), Kluwer Academic Publishers, 1995, pp. 187-208.

[3] M. Gergeleit, M. Mock and E. Nett, “T-CORBA: making object-oriented systems time-aware,” *Computer Systems Science & Engineering*, vol. 13, pp. 151-160, 1998.

[4] H. Kopetz, “The Time-Triggered Approach to Real-Time System Design,” in *Predictably Dependable Computing Systems*, J.-C. L. B. Randell, H. Kopetz, B. Littlewood (Eds.) (Eds.), Springer Verlag, 1995, pp. 53 - 66.

[5] D. Powell, J. Arlat, L. Beus-Dukic, A. Bondavalli, P. Coppola, A. Fantechi, E. Jenn, C. Rabejac and A. Wellings, “GUARDS: A Generic Upgradable Architecture for Real-Time Dependable Systems,” *IEEE TPDS*, vol. 10, pp. 580-599, 1999.