AN ARCHITECTURE FOR RELIABLE DISTRIBUTED COMPUTER-CONTROLLED SYSTEMS

Luís Miguel Pinho, Francisco Vasques

In Distributed Computer-Controlled Systems (DCCS), both real-time and reliability requirements are of major concern. Architectures for DCCS must be designed considering the integration of processing nodes and the underlying communication infrastructure. Such integration must be provided by appropriate software support services.
In this paper, an architecture for DCCS is presented, its structure is outlined, and the services provided by the support software are presented. These are considered in order to guarantee the real-time and reliability requirements placed by current and future systems.

1. Introduction

Distributed Computer-Controlled Systems (DCCS) are increasingly used in the industrial environment, where computer systems are expected to perform correctly, even in the presence of faults. The traditional approach to guarantee the dependability requirements of DCCS is to replicate some of its components, in order to tolerate individual faults. However, when replicated components are used, there is the need for reliable and time-bounded communication services. Messages must be correctly and orderly delivered according to their timing requirements. Therefore, the full integration of the communication infrastructure with the processing nodes is required in order to obtain the desired level of confidence in the system.

Using COTS as the systems' building blocks provides a cost-effective solution, and at the same time allows for an easy upgrade and maintenance of the system. However, as COTS hardware and software does not usually provide the confidence level required by reliable real-time applications, reliability requirements must be guaranteed by a software-based fault-tolerance approach.

The use of COTS components usually implies the use of fail-uncontrolled components. It is not possible to guarantee fail-silent properties for off-the-shelf

B. Kleinjohann (ed.), *Architecture and Design of Distributed Embedded Systems*

hardware and/or software, as these components usually do not have the required self-checking mechanisms for detecting faults. Fail-uncontrolled components require the use of active replication (Powell, 1994), since masking faults in one component requires the replication of such component in other nodes. Consequently, a COTS-based system must be able to manage by its own such component replication.

The proposed architecture is targeted to provide a guaranteed (timely and reliable) execution environment to hard real-time applications. In addition, it is also targeted to provide the adequate quality of service to soft real-time applications, which must not interfere with the behaviour of the hard real-time applications. It is not targeted to safety-critical systems, as these systems require a greater level of dependability and a more restricted set of failure assumptions (Laprie, 1992).

2. System Architecture

The system architecture (Figure 1) is based on the use of a set of processing nodes, where distributed hard real-time applications may execute. To ensure the desired level of reliability to hard real-time applications, specific components of these applications may be replicated.

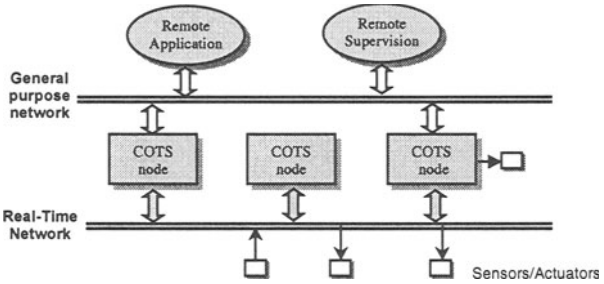

Figure 1. System architecture.

Nodes are interconnected by a real-time network, which provides the communication infrastructure for the hard real-time applications (interconnecting controllers, sensors and actuators). This real-time network is also intended to support the replica management mechanisms. At the above level, as there is the need of interconnection with the upper levels of the DCCS (e.g. for remote access, remote supervision and/or remote management), there is a general-purpose network interconnecting some of the DCCS nodes.

2.1. Node Architecture

Each node (Figure 2) integrates both a hard real-time subsystem (HRTS) and a soft real-time subsystem (SRTS). The goal of the HRTS is to provide a framework to support reliable hard real-time applications, which are at the core of the system. The SRTS provides the interface for the remote supervision management of the DCCS.

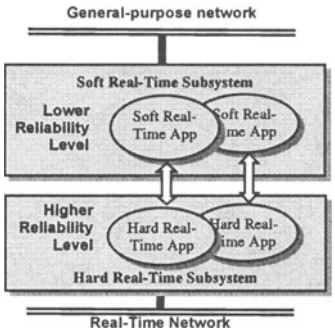

Figure 2. Node structure.

The communication mechanisms between both subsystems must guarantee that failures in the SRTS (less reliable) do not interfere with the HRTS (concerning its timing and reliability requirements). Therefore, mechanisms for memory partitioning must be provided, and the communication mechanisms must guarantee the integrity of data transferred from the SRTS to the HRTS, by upgrading its confidence level.

The HRTS is responsible for providing a framework for reliable execution of hard real-time applications. Hence, applications have guaranteed execution resources, including processing power, memory and communication support. This claims for a separated real-time communication network for the HRTS, where messages sent from one node to another are received and processed in a bounded time interval. The HRTS Support Services are responsible for the real-time communication management and also provide a transparent framework for the replication of application components.

The SRTS provides a set of services to support the supervision and management level of the DCCS. It may provide CORBA/HTTP servers, which can be accessed using supervision and management tools. At this system level, flexibility is a major goal, since new services can be created as the system is upgraded.

2.2. Communication Infrastructure

Current work is being performed in order to assess the suitability of the Controller Area Network (CAN) (ISO, 1993) to act as the real-time network. Although being originally designed for use within road vehicles, CAN is also being considered for the automated manufacturing and distributed process control environments (Zuberi and Shin, 1997). Several studies on how to guarantee the real-time requirements of messages in CAN networks are available (e.g. (Tindell et al., 1995)). Nevertheless, the continuity of service is not fully guaranteed, since it may be disturbed by temporary periods of network inaccessibility (periods during which stations cannot communicate with each other, due to the existence of on-going error detection and recovery mechanisms). A study of the inaccessibility characteristics of CAN networks has been presented at (Rufino and Veríssimo, 1995), identifying the duration of its error detection and recovery periods. The integration of the inaccessibility studies with the timing analysis (Pinho et al., 2000a) indicate that

CAN presents some problems, as it is not able to provide different integrity levels to the supported applications. However, it is also perceived that, under an appropriate set of fault assumptions, it can be used to support reliable real-time DCCS (Pinho et al., 2000a).

3. Hard Real-Time Subsystem

The HRTS allows real-time applications to be distributed over the nodes of the system (Figure 3). It is based on the software integration of COTS components, that is, "replication handled entirely by software using off-the-shelf hardware" (Guerraoui and Schiper, 1997), rather than building software on top of specialised hardware.

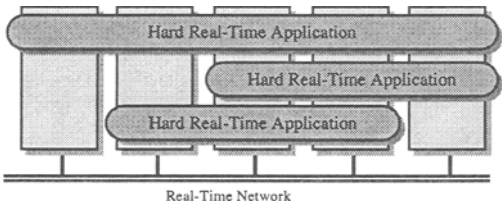

Figure 3. HRTS structure.

The HRTS provides a framework to support hard real-time applications, where timing requirements are guaranteed through the use of current off-line schedulability analysis techniques (Response-Time Analysis (Audsley et al., 1993)). A multitasking environment is provided to support real-time applications, with services for task communication and synchronisation (including distribution).

One hard real-time application is constituted by several tasks (processing units), which combined together perform the desired service. In Figure 4, a hard real-time application is divided in four tasks, which execute in different nodes of the HRTS. Each node has its own (non-distributed) COTS kernel and hardware, which provides the desired real-time multitasking support. An additional advantage of using both a COTS kernel and hardware is that it provides means for the easy upgradability and portability of the system.

The goal of the HRTS support software (Figure 4) is to provide the distribution support (including both the application distribution and the replication management) to hard real-time applications. This module manages the communication between different nodes, resulting from the replica management, the application distribution and the interface with the controlled environment.

The HRTS supports the active replication of software with dissimilar task sets in each node. The reason for allowing dissimilar task sets is twofold. By providing different execution environments in each node, the tolerance to design faults is increased, as the probability of the same fault occurring in more than one node decreases. At the same time, the architecture flexibility is increased, since nodes are not just duplicates, allowing for a more flexible design of real-time applications.

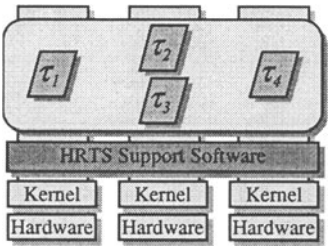

Figure 4. Distributed Hard Real-Time Application.

However, multitasking applications with differentiated execution environments are likely to result in replicated components with non-deterministic executions. Hence, the HRTS support software provides mechanisms to guarantee deterministic execution. As these mechanisms need to be time-efficient, they are not based in replica co-ordination but in the concept of timed messages (Poledna et al., 2000).

A layered approach is provided to the HRTS, in order to simplify the system development. The HRTS support software (Figure 5) EINBETTENcomprises two layers:

1. The *Communication Manager* layer, which is responsible for the reliable and timely transfer of real-time data;
2. The *Replica Manager* layer, which is responsible for the transparent management of the replicated components, in order to not burden the programmer with explicitly programming of replicate managing mechanisms.

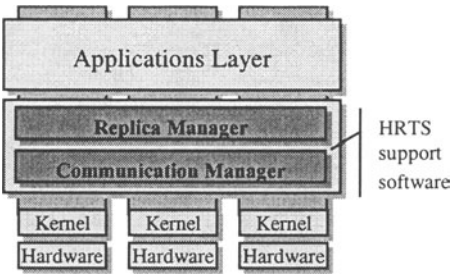

Figure 5. Hard Real-Time Subsystem layers.

3.1. Scheduling model

The HRTS is intended to support one or more hard real-time applications. Each application consists of a set of related tasks (τ_1 ... τ_n), being each task a single processing unit. Tasks from the same application can be allocated to different nodes, (distributed environment). In order to use the well-known Response Time Analysis (Audsley et al, 1993), each task is released only by one invocation event, but can be released an unbounded number of times. A periodic task is released by the runtime (temporal invocation), while a sporadic task can be released either by another task or

by the environment. After being released, a task cannot suspend itself or be blocked while accessing remote data (external blocking).

Tasks are allowed to communicate with each other either through *shared data* or by *release event* objects. Shared data objects are used for asynchronous data communication between tasks, while release event objects are used for the release of sporadic tasks. Tasks are designed as small processing units, which, in each invocation, read inputs; carry out the processing; and output the results. The goal is to minimise task interaction, in order to improve the schedulability analysis and increase the system's efficiency.

As there is no synchronous interaction between tasks, the release of a task cannot be directly made by other tasks. Thus, sporadic tasks are suspended waiting in a release event object, which is triggered by waking tasks, whereas the runtime executive triggers periodic tasks. Internal blocking due to task communication can be bounded and off-line analysed using Priority Ceiling Protocols (Sha et al., 1990).

3.2. Replication Model

As there is the target of reliability through replication, it is important to devise which is the replication unit (that is, the smaller replication entity). Therefore, the notion of *component* is introduced. Applications are divided in components, each one being a set of tasks and resources that interact to perform a common job. The component can include tasks and resources from several nodes, or it can be located in just one node. In each node, several components may coexist. As an example, Figure 6 shows a real-time application with 4 tasks (τ_1, τ_2, τ_3 and τ_4) divided in two different components. Component C_1 encompasses tasks τ_1 (node 1) and τ_2 (node 2). Its replica encompasses tasks τ_1' (node 3) and τ_2' (node 5). Component C_2 encompasses tasks τ_3 (node 2) and τ_4 (node 3), while its replica encompasses tasks τ_3' (node 4) and τ_4' (node 5).

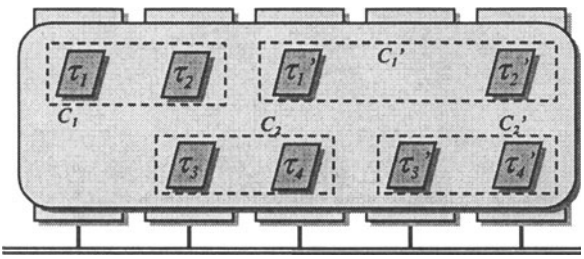

Figure 6. Replicated Hard real-time application.

A similar concept to the *component* can be found in the notion of "capsules" of the Delta-4 architecture (Powell, 1991). As the component, a Delta-4 "capsule" is the unit of replication, embodying a set of tasks (referred as threads) and objects. However, a "capsule" has its own thread scheduling and separated memory space, and is also the unit of distribution. Thus, the Delta-4 concept of "capsule" is more related to Unix processes, whilst the presented component is a more lightweight concept, which is used to structure replication units.

By creating components, it is possible to define the replication degree of specific parts of the application, according to its desired reliability level and the reliability of its components. The degree of replication of a component is referred as *n-replicated component*. In Figure 6, both components C_1 and C_2 are 2-replicated components.

By replicating components, efficiency decreases as the number of tasks and messages increases and there is the need for agreement on the output of computations. Hence, it is possible to trade reliability for efficiency and vice-versa. Although efficiency should not be regarded as *the* goal of a reliable system, it can be increased by means of decreasing the degree of redundancy of more reliable components (if this assumption can be guaranteed).

The component is the fault-containment unit. Faults in one task may produce the failure of the component. However, if a replica of the component fails, the application will not fail, since the output consolidation will mask the failed replica. Therefore, in the model of replication, the outputs of internal tasks (within a component) do not need to be agreed. The output consolidation is only needed when results are made available to other components or to the controlled system. As can be seen in Figure 7, several possibilities exist for the configuration of an application. The first part of the Figure shows the same configuration presented in Figure 6, while in the second part there is a solution where the application is divided in three components and only component C_2 is replicated. The double arrows indicate communication between different components, thus communication needing consolidated data.

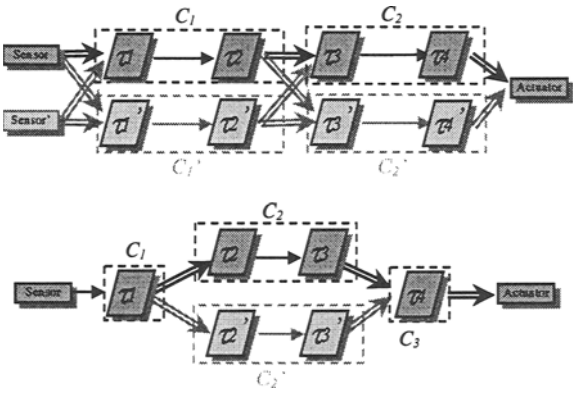

Figure 7. Examples of application configuration.

Note that the second solution is more efficient, as there are only two more tasks than the strictly needed by the application. However, the reliability assumption of both the sensor and components C_1 and C_3 (and the nodes where they execute) must be higher than in the previous solution, as they are not replicated.

There is the need to guarantee that replicas execute deterministically, that is, replicated tasks execute with the same data and timing-related decisions are the same in each replica. This determinism can be achieved restricting the application from using timing non-deterministic mechanisms. However, the use of multitasking would

not be possible, since task synchronisation and communication mechanisms inherently lead to timing non-determinism. The use of timed messages (Poledna et al., 2000) allows a restricted model of multitasking to be used and at the same time eliminates the need for agreement between the internal tasks of each component. With timed messages, agreement is only needed to guarantee that all replicated components work with the same input values and that they all vote on the final output. The use of timed messages implies the use of appropriate clock synchronisation algorithms, since there is the need of clocks with a bounded difference.

3.3. HRTS Replica Manager

The goal of the Replica Manager layer is to provide hard real-time applications with the set of resources required for communication between distributed tasks and between replicated components. In the HRTS, tasks communicate with each other by using shared data and the release of event objects. However, these mechanisms must be different when they are used for intra-component communication or for inter-component communication. In addition, there is also the difference when communication is due to distribution or it is due to the replication mechanisms.

If precedence relations exist between tasks, the communication mechanisms can be simplified, since these precedence relations guarantee deterministic execution (Wellings et al., 1998). If the receiving task is sporadic and is released by a sending task, it is guaranteed that, in all replicated components, the replicas of the task will execute with the same data. The same reasoning can be applied when the receiving task is periodic with a period related to the period of the sender task.

Although the goal of the replica manager is to transparently manage distribution and replication, it is considered that a completely transparent use of these mechanisms may introduce unnecessary overheads, since there are some special cases that must be considered. Therefore, the application programmer (transparent approach) does not consider the use of components at the design phase. Later, in a configuration phase, the system engineer configures the components and its replication level and allocates the different tasks in the distributed system. In this phase, the communication streams that need timed messages are identified. Guidelines for splitting the application in components are to be developed to ease the job of engineers.

3.4. HRTS Communication Manager

The Communication Manager layer is responsible for providing a reliable and timely transfer of real-time data. The group communication abstraction is used as the framework for reliable communication and to support the replica management (Powell, 1994). In the replication model, a set of replicas from the same component is referred as a group. The Communication Manager must provide the following set of mechanisms:

1. *1-to-many communication*, when a task of a non-replicated component wishes to disseminate its result to the *n* input tasks of a *n-replicated* component (reliable multicast protocol).

2. *Many-to-1 communication*, when an input task of a non-replicated component receives inputs from a *n-replicated* component (consensus algorithm).
3. *Many-to-many communication*, when a group of *n* output tasks of *a n-replicated* component disseminates its results to the *n* input tasks of a *n-replicated* component (interactive consistency (Pease et al., 1980) algorithm).
4. *1-to-1 communication*, for communication between tasks of the same component (intra-component communication) or between the output task of a non-replicated component and the input task of a non-replicated component (no need for specific algorithms).

The suitability of the CAN protocol for the communication infrastructure is being studied (Pinho et al., 2000a) (Pinho et al., 2000b). Although current results indicate that CAN presents some problems as it is not resilient to station errors, it is perceived that, with the appropriate set of fault assumptions, it can be used as the communication infrastructure.

3.5. Interconnection with the outside world

The interconnection of the HRTS with the SRTS must provide mechanisms for transfer of information between both subsystems. Communication from the HRTS to the SRTS does not present any major problem, since it is assumed that this information has a higher reliability level. However, if the output to the SRTS comes from replicated components, appropriate agreement must be performed. Conversely, the reliability of the data arriving from the SRTS must be increased, in order to prevent the introduction of erroneous values. Also, if the data is to be provided to replicated components, reliable communication algorithms must be used to disseminate this data.

Interconnection with the controlled system is performed through the use of sensors and actuators. Sensor values can be treated as the output of non-replicated components and its dissemination must be performed accordingly to the desired reliability. The time at which the value is valid must be agreed upon. Output to actuators must also be agreed upon between different replicas. Such agreement may be made either in the computational system or the actuators may perform themselves this agreement, by mechanical or electronic voting on the result.

4. Conclusions

In this paper, an architecture for Distributed Computer-Controlled Systems (DCCS) is presented. It is targeted to provide a guaranteed (timely and reliable) execution environment to current and future systems.

The structure of the architecture is presented, together with the guidelines used in its design, and its scheduling and replication models. The support software, which provides distribution support (including both the application distribution itself and the replication management) to hard real-time applications, is also discussed.

Acknowledgements

The authors would like to thank the anonymous referees for their helpful comments. This work was partially supported by FLAD (project SISTER 471/97), FCT (project DEAR-COTS 14187/98) and IDMEC.

References

Audsley, N., Burns, A., Richardson, M., Tindell, K and Wellings, A. (1993). "Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling". In *Software Engineering Journal*, Vol. 8, No. 5, pp. 285-292.

Guerraoui, R. and Schiper, A. (1997). "Software-Based replication for Fault Tolerance". In *IEEE Computer*, April 1997, pp. 68-74.

ISO 11898 (1993). *Road Vehicle - Interchange of Digital Information - Controller Area Network (CAN) for High-Speed Communication.* ISO.

Laprie, J. L. (1992). *Dependability: Basic Concepts and Terminology.* Dependable Computing and Fault-Tolerant Systems, Vol. 5, Springer-Verlag.

Pease, M., Shostak, R. and Lamport, L. (1980). "Reaching Agreement in the presence of Faults". *Journal of the ACM*, Vol. 27, N. 2, pp 228-234.

Pinho, L. M., Vasques, F. and Tovar, E. (2000a) "Integrating Inaccessibility in Response Time Analysis of CAN Networks". In *Proceedings of the 3rd IEEE International Workshop on Factory Communication Systems*, Porto, Portugal, pp. 77-84.

Pinho, L., Vasques, F. and Ferreira, L. (2000b) "Programming Atomic Multicasts in CAN", In *Proc. of the 10th International Real-Time Ada Workshop*, ACM, Ada Letters, To Appear

Poledna, S., Burns, A., Wellings, A. and Barrett, P. (2000). "Replica Determinism and Flexible Scheduling in Hard Real-Time Dependable Systems". In *IEEE Transactions on Computers*, Vol. 49, N. 2, pp 100-111.

Powell, D. (Ed.) (1991). *Delta-4: A Generic Architecture for Dependable Distributed Computing.* ESPRIT Research Reports, Springer Verlag.

Powell, D. (1994). "Distributed Fault Tolerance – Lessons Learnt from Delta-4". In *Hardware and Software Architectures for Fault Tolerance. Experiences and Perspectives.* Banatre, M. and Lee P. A. (eds.). Lecture Notes in Computer Science 774, Springer-Verlag, 199-217.

Rufino, J. and Veríssimo, P. (1995). "A Study on the Inaccessibility Characteristics of the Controller Area Network". In *Proc. of the 2nd CAN Conference*, London, United Kingdom.

Sha, L., Rajkumar, R. and Lehoczky, J. (1990). "Priority Inheritance Protocols: An Approach to Real-Time Synchronization". In *IEEE Tr. on Computers*, Vol. 39, N. 9, pp. 1175-1185.

Tindell, K., Burns, A. and Wellings, A. (1995). "Calculating Controller Area Network (CAN) Message Response Time". In *Control Engineering Practice*, Vol. 3, No. 8, pp. 1163-1169.

Wellings, A., Beus-Dukic, Lj. and Powell, D. (1998). "Real-Time Scheduling in a Generic Fault-Tolerant Architecture". In *Proc. of the Real-Time Systems Symposium*, Madrid, Spain.

Zuberi, K. and Shin, K. (1997). "Scheduling messages on Controller Area Network for Real-Time CIM Applications". In *IEEE Transactions on Robotics and Automation*, Vol. 13, No. 2, pp 310-314.