IPP-HURRAY! Research Group

http://www.hurray.isep.ipp.pt

Polytechnic Institute of Porto
School of Engineering (ISEP-IPP)

The DEAR-COTS Hard Real-Time Subsystem

Luís Miguel PINHO
Francisco VASQUES (FEUP)

HURRAY-TR-0003

May 2000

THIS WORK IS PARTIALLY SUPPORTED BY FCT UNDER PROJECT DEAR-COTS (Praxis/P/EEI/14187/1998) AND BY FLAD UNDER PROJECT SISTER (Proj.471/97)

The DEAR-COTS Hard Real-Time Subsystem

Luís Miguel PINHO

IPP-HURRAY! Research Group
Polytechnic Institute of Porto (ISEP-IPP)
Rua Dr. António Bernardino de Almeida, 431
4200-072 Porto
Portugal
Tel.: +351.22.8340502, Fax: +351.22.8340529
E-mail: lpinho@dei.isep.ipp.pt
http://www.hurray.isep.ipp.pt

Francisco VASQUES

University of Porto (FEUP)
Rua dos Bragas
4050-123 Porto
Portugal
Tel.: +351.22.2041774, Fax: +351.22.2074241
E-mail: vasques@fe.up.pt
http://www.fe.up.pt/~vasques

Abstract:

In this report, the Hard Real-Time Subsystem of DEAR-COTS is described, and the services it must provide are identified. This report is an input of ISEP/IPP and FEUP for the specification of the DEAR-COTS architecture (deliverable to the FCT).

DEAR-COTS Architecture

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. In an industrial environment, unexpected malfunctioning of a DCCS is unacceptable, as it may cause economic, environmental or life losses. It is obvious that the dependability (reliability and availability) of a DCCS must be considered as an integrated part of any DCCS architecture.

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 dependable 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, the use of COTS implies that specialised hardware can not be used to guarantee reliability requirements. 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 of off-the-shelf hardware and/or software, as these components usually do not have the required self-checking mechanisms for detecting faults and consequently it is not possible to mask those faults. Fail-uncontrolled components require the use of active replication [1], since masking faults in one component requires the replication of such component in other nodes. Consequently, the system must be able to manage this component replication, guaranteeing that whenever a components fails, another component will assume its responsibilities.

Besides these reliability and availability requirements to guarantee the correct timing behaviour of the supported real-time applications, DCCS has also the need to be interconnected with other parts of the overall system. Currently, these kind of systems demand for more flexibility and interconnectivity capabilities, while guaranteeing the availability and reliability requirements of the supported real-time applications. Hence, the integration of hard real-time applications, whose requirements have to be guaranteed, with soft real-time applications, where a more flexible approach can be used, is another goal of the architecture.

The proposed architecture intends to cope with:

- Internal physical faults, which are addressed through replication of systems' components;
- Temporary design faults, which can be tolerated due to the differences in the replicas execution environment [1], as they were temporary internal physical faults;
- Permanent design faults, which must be tolerated by means of design diversity. This means that there must be a way to allow for such design diversity;

- Temporary external physical faults must be avoided with appropriate filtering and shielding of the system. Therefore, such kind of faults will not be considered in the development of the software architecture;

The proposed architecture is targeted to provide a guaranteed (timely, reliable and available) 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 (e.g. supervision and management tasks), 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 [2].

2 The DEAR-COTS architecture

The DEAR-COTS architecture is targeted to reliable Distributed Computer-Controlled Systems (DCCS). It provides an execution environment for real-time applications, with reliability and availability requirements, through the use of COTS hardware and software.

A DEAR-COTS system (Fig. 1) is built using distributed processing nodes, where distributed hard real-time applications may execute. To ensure the desired level of fault tolerance (reliability and availability) to the supported hard real-time applications, specific components of these applications may be replicated.

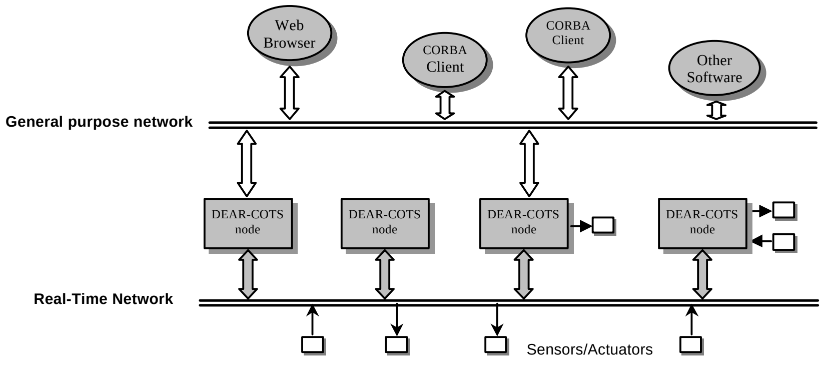

Fig. 1 – DEAR-COTS architecture

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 also provides the DEAR-COTS architecture with a communication infrastructure to support the replica management mechanisms. At the above level, as there is the need to interconnect these nodes 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.

By providing a distributed architecture to the development of real-time computer systems, several advantages are obtained:

- the overall throughput of the system can be increased, as more processing power is available for the applications;

- specialised hardware (e.g. specific sensor/actuator interfaces, human-machine interfaces, etc.) can be used to interconnect the real-time computer system with the environment, general-purpose nodes are used for processing activities;
- the reliability of the system is increased, since application components can be replicated and thus single points of failure are avoided in the system.

The use of broadcast networks, instead of point-to-point links decreases the cost of the system, and also increases the ease of installation, maintenance and expansion (reconfiguration). However, providing only a broadcast medium to interconnect distributed hard real-time applications decreases the overall reliability of the system, as the network becomes a single point of failure. Hence, the confidence placed in this shared medium must be evaluated, and if considered necessary, network redundancy must be provided.

Each DEAR-COTS node (Fig. 2) integrates both a hard real-time subsystem (HRTS) and a soft real-time subsystem (SRTS). The goal of the hard real-time subsystem is to provide a framework to support reliable and available hard real-time applications, which are the core of the DCCS application. The soft real-time subsystem provides the interface for the remote supervision and remote management of the Distributed Computer Control System.

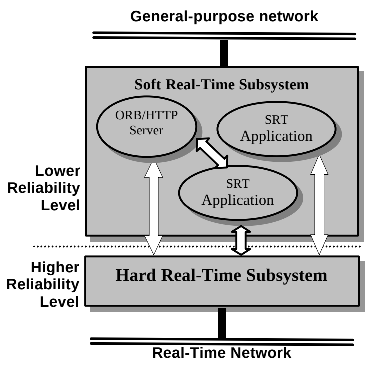

Fig. 2 - DEAR-COTS node structure

The communication mechanisms between both subsystems must be carefully designed, guaranteeing that failures in the soft real-time subsystem (less reliable) do not interfere with the hard real-time subsystem (concerning its timing, availability and reliability requirements). Therefore, mechanisms for memory partitioning must be provided, and also the inter-communication mechanisms must guarantee the integrity of data transferred from the SRTS to the HRTS, by upgrading its confidence level.

At the hard real-time level, the HRTS is responsible for allowing hard real-time applications a framework for reliable execution. Hence, applications have guaranteed execution resources, including processing power, memory and communication infrastructure. This is the main reason for the need of 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 provided as the system evolves. However, there can be no hard real-time guarantees and thus techniques such best-effort scheduling or value-based scheduling must be used.

Although both subsystems are fully integrated in the DEAR-COTS architecture, from the DCCS point of view, the HRTS is the most important one. Therefore, the main focus of this technical report is the HRTS characterisation, including its integration with both the external environment and the SRTS.

3 The Hard Real-Time Subsystem

The HRTS of the DEAR-COTS architecture (Fig. 3) allows real-time applications to be distributed over the nodes of the system. It is based on the software integration of COTS components, that is, "replication handled entirely by software using off-the-shelf hardware" [3] rather than the usual solution, which is to build software on top of specialised hardware.

The HRTS subsystem provides a framework to support hard real-time applications, and their timing requirements guaranteed through the use of current off-line schedulability analysis techniques (e.g., the well-known Response-Time Analysis [4]). A multitasking environment is provided to support the real-time applications, with services for task communication and synchronisation (including distribution).

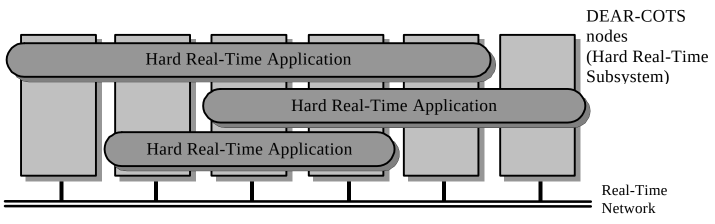
DEAR-COTS nodes (Hard Real-Time Subsystem)
Hard Real-Time Application
Hard Real-Time Application
Hard Real-Time Application
Real-Time Network

Fig. 3 – HRTS structure

Although the possibility of executing several hard real-time applications together in the same nodes, there are no provisions for memory partitioning between them. It is considered that the level of reliability associated to these applications is sufficient enough, in order to guarantee the non-interference between them. Furthermore, as there is the provision for replication of applications, the target of the DEAR-COTS architecture will be to avoid errors to be spread between nodes (error confinement).

One hard real-time application is constituted by several tasks (processing units), which combined together perform the desired service. These processing tasks are distributed over the nodes of DEAR-COTS. In Fig. 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 multitasking support. This means that the kernel must be a real-time one, providing a multitasking environment with a real-time behaviour. An additional advantage of using both a COTS kernel and hardware is that it allows for the easy upgradability and portability of the system.

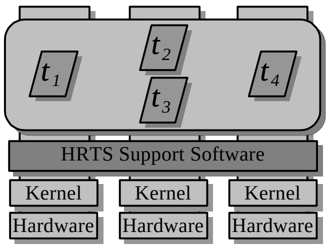

Fig. 4 – Distributed Hard Real-Time Application

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

The HRTS also supports the active replication of software (Fig. 5) with dissimilar replicated 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, since nodes are considered as independent from the point of view of failures. At the same time, the architecture flexibility is increased, since nodes are not just copies of each other, allowing for a more flexible design of real-time applications.

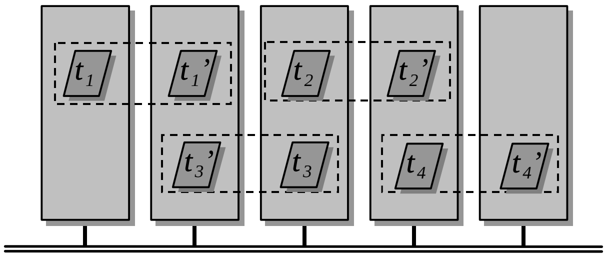

Fig. 5 – Replicated Hard Real-Time Application

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 the replica deterministic execution. As these mechanisms need to be time-efficient (introducing a small execution overhead), they are not based in replica co-ordination, but in the notion of timed messages [5].

A layered approach to the HRTS is provided in order to simplify the development and use of the HRTS support software (Fig. 6).

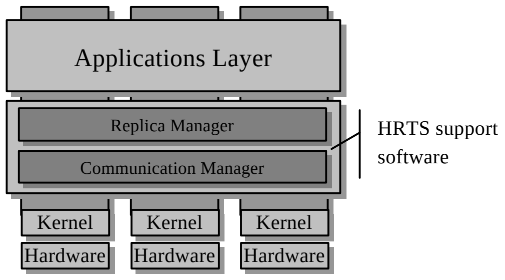

Fig. 6 – Hard Real-Time Subsystem layers

The HRTS support software comprises two layers, to provide the desired services for hard real-time applications:

- The Communication Manager layer is responsible for the adequate communication services, in order to provide a reliable and timely transfer of real-time data;
- The Replica Manager layer is responsible for transparently manage the replicated components, in order not to burden the programmer with explicitly programming of replicate managing mechanisms.

3.1 Scheduling model

The HRTS is intended to support one or more hard real-time applications (Fig. 3). Each one of these applications is distributed over some (or all) of the DEAR-COTS nodes. Each application consists of a set of related tasks ($\tau_1 \ldots \tau_n$), being each task a single processing unit. Tasks from the same application can be allocated to different nodes, which provides a distributed environment.

In order to allow the use of the scheduling model of [4] (later refined in [6]), 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 (Fig. 7) either through shared data (from now on called *data objects*) or by release event objects (which can also carry data). Shared data objects are used for asynchronous data communication between tasks, while release event objects are used for the releasing of sporadic tasks.

As there is no synchronous interaction between tasks, the release of a task cannot be made directly by other tasks or by the runtime executive. Thus, sporadic tasks are suspended waiting in a release event object, which is triggered by waking tasks. Internal blocking due to task communication can be bounded and off-line analysed through the use of Priority Ceiling Protocols [7].

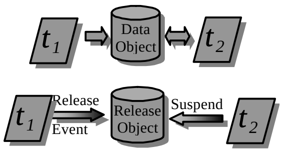

Fig 7.

As real-time applications can also be distributed over the DEAR-COTS nodes, there will be real-time tasks interacting through the real-time network (Fig. 8). As can be seen in figure 8, when τ_1 sends data to τ_2 the response-time of the message depends not only on the scheduling of tasks in its node, but also in the message scheduling. As there could be several messages queued to be transferred, the response time analysis of the task execution must consider the interference of the overall message scheduling.

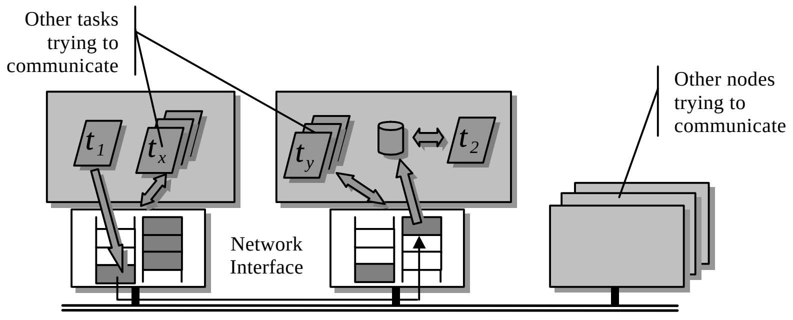
Other tasks trying to communicate
Other nodes trying to communicate
t_1
t_x
t_y
t_2
Network Interface

Fig. 8 - Distributed Communication

In order to allow the schedulability analysis of this distributed model, techniques such as holistic schedulability analysis [6] must be used. In such model, the schedulability analysis of the communication network is integrated with the schedulability analysis of processing nodes. Furthermore, the HRTS Support Software also interferes in the scheduling model. Therefore, this interference must be also modelled and integrated in the schedulability analysis.

To prevent external blocking, remote communication can only be asynchronous communication, e. g., a task cannot suspend waiting for the return of a remote read. However, mechanisms such as Remote Procedure Calls can be used if the client task is spilt in two different tasks (Fig. 9), one for sending the request and another to wait for the response.

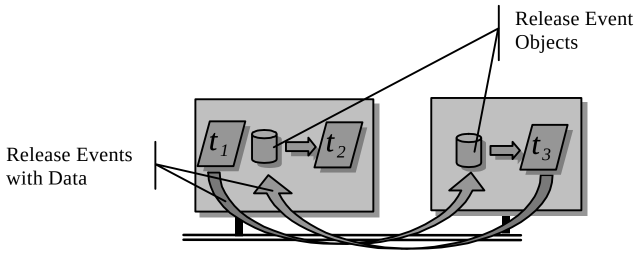

Fig 9. Client task is split in two tasks (τ_1 and τ_2) in order to access remote server task (τ_3)

In order to allow the use of the presented model, each task characteristics must be known off-line. These characteristics are the task deadline, the objects needed for the execution of each task, its computation time and the period of task invocations (or its minimum inter-arrival time for sporadic tasks). The use of offset scheduling theory [8] (which allows decreasing the worst-case response time of tasks) is not precluded. Each application designer has the choice of using or not of such offsets theory.

3.2 Replication Model

As there is the target of reliability through replication, it is important to devise which is the unit of replication (that is, the smaller replication entity). From the presented definitions, two different units could be considered as the replication unit: a single task or the complete hard real-time application. However, as there is the goal of minimising the need for replica co-ordination, any of these two solutions would be very restrictive. In the latter, in order to replicate part of the application all of its tasks would have to be replicated, which would unnecessarily increase the processing load. In the former, each task output would have to be consolidated, which would increase the inter-task communication load.

Therefore, the notion of a DEAR-COTS software 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, Fig. 10 shows a real-time application with 4 tasks (τ_1, τ_2, τ_3 and τ_4). The application is divided in two different components, which to ensure the desired level of reliability are replicated.

Component C_1 encompasses tasks τ_1 (in node 1) and τ_2 (in node 2). Its replica encompasses tasks τ_1' (in node 3) and τ_2' (in node 5). Component C_2 encompasses tasks τ_3 (in node 2) and τ_4 (in node 3), while its replica encompasses tasks τ_3' (in node 4) and τ_4' (in node 5).

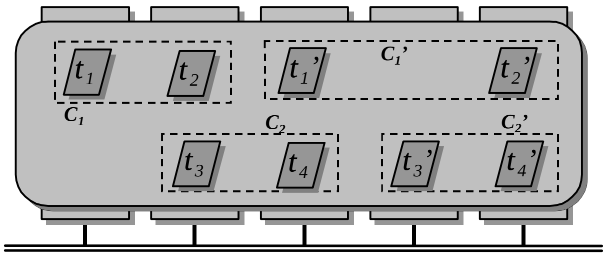

Fig. 10 – Hard real-time application

By creating software components, more flexibility is attained in the design of replicated hard real-time applications. It is possible to define the replication degree of specific parts of the real-time application accordingly to the reliability of its components and the desired level of reliability for the application. The degree of replication is defined as *n-replicated component*. In Fig. 4 both components C_1 and C_2 are 2-replicated components.

By replicating software components, efficiency decreases as the number of tasks and messages increases. Also, as there is the need for agreement on the output of computations, efficiency is also decreased. 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 decreasing the degree of redundancy of more reliable components (if this assumption can be ensured).

The component is the unit of replication, therefore a component is a fault-containment unit. Faults in one task may produce the failure of the component. However, if a replica of a component fails, the application will not fail, since the output consolidation will mask the failed replica.

Therefore, in the DEAR-COTS model of replication, the outputs of internal tasks (within a component) do not need to be agreed. The output consolidation is only needed when the result is made available to other components or to the controlled system. As can be seen in Fig. 11, several possibilities exist for the configuration of an application. The first part of the figure shows the same configuration presented in Fig. 10, while in the second part there is a solution where the application is divided in three components and only component C_2. The gray colour is used for the replication-related additions to the system. The double arrows indicate communication between different components, thus communication with consolidated data.

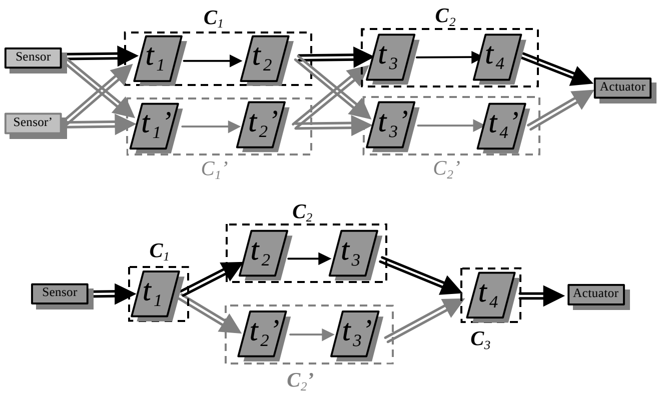

Fig. 11 – Examples of real-time application configuration

Note that the second solution is more efficient, as there are only two more tasks than strictly needed by the application and thus the number of messages is much smaller. However, the reliability assumption of both the sensor and of 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.

Although not shown in the previous figures, a component can have more than one input task (task that receives data or events from outside the component) and more than one output task (task that output results to other components or to the environment). Component tasks that are not input nor output tasks are named internal tasks.

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.

Guaranteeing that replicas take the same scheduling decisions, by performing an agreement in every scheduling decision, allows for the use of non-deterministic mechanisms. However, it imposes the modification of the underlying scheduling mechanisms and leads to a huge overhead on the system, since agreement must be made at every dispatching point.

The use of timed messages [5] 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. Therefore, the DEAR-COTS model of replication uses timed messages for implicit replica co-ordination. 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 they need 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 resources required for communication between distributed tasks and between replicated components. In the HRTS, tasks communicate with each other by using shared data and release 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 the replication mechanisms.

If precedence relations exist between tasks, the communication mechanisms can be simplified, since these precedence relations guarantee deterministic execution [9]. If the receiving task is sporadic and is released by a sending task, it is guaranteed that in all replicated components the replicas of this second 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 a second task.

Although the goal of the replica manager is to transparently manage replication, it is considered that a completely transparent use of these mechanisms may introduce unnecessary overheads, since as noticed in the previous paragraph, 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. In this phase, the application itself identifies which are the communications needing timed messages. Guidelines for the splitting the application in components are to be developed to ease the job of the systems engineer.

On the other hand, there is no provision for transparent distribution of the application. Thus, from the identified mechanisms, programmers are only able to explicitly see the difference between inter- and intra-node communication.

Therefore, from the application programmer perspective, resources must exist for:

- Sharing data between tasks in the same node;
- releasing tasks in the same node;
- sharing data between tasks in different nodes;
- releasing tasks in different nodes.

Note that, in the HRTS scheduling model, remote blocking is avoided by preventing tasks from reading remote data. Hence, when sharing data between tasks in different nodes, there is no risk of a task being blocked when reading from a remote data object.

Although these are the only available mechanisms to the programmer, the Replica Manager layer must also provide resources for replica management activities. Note that, when replication is considered, resources that provide deterministic execution are needed for communication between tasks without schedule-captured precedence relations, and appropriate communication algorithms must be used for data transfer between components (these will be described in the next section).

The identification of which are these resources, and their relationship with the communication algorithms is a work currently being performed in the DEAR-COTS framework.

3.4 HRTS Communication Manager

The Communication Manager layer is responsible for the adequate communication services between nodes, providing a reliable and timely transfer of real-time data. Therefore, it must provide mechanisms for:

- *1-to-1 communication*: communication between tasks of the same component (inter-component communication), or between an output task of a non-replicated component and an input task of a non-replicated component;
- *1-to-many communication*: communication between an output task of a non-replicated component and *n* input tasks of a *n-replicated* component;
- *many-to-1 communication*: communication between *n* output tasks of a *n-replicated* component and an input task of a non-replicated component;
- *many-to-many communication*: communication between *n* output tasks of a *n-replicated* component and *n* input tasks of a *n-replicated* component.

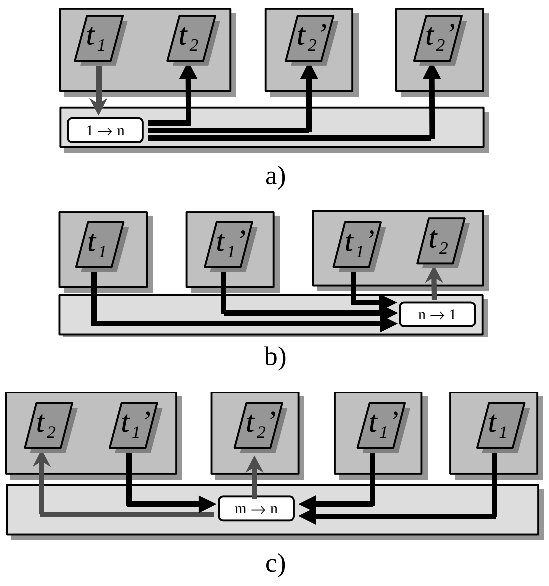

Fig. 12 – Example of replicated tasks' communication: (a) *1-to-many communication*; (b) *many-to-1 communication*; (c) *many-to-many communication*.

The group communication abstraction can be used as the framework for reliable communication, and also for replica management [2]. In the replication model of DEAR-COTS, a set of replicas from the same component is referred as a group. The goal is to use current group communication techniques for simplifying the needed communication mechanisms.

When a task wishes to disseminate its result to more than one task (*1-to-many communication*), reliable multicast algorithms must be used in order to guarantee that replicated receivers get the same information.

When a task receives inputs from more than one task (*many-to-1 communication*), it must use a consensus algorithm, to choose one of the, possibly different, results it receives, or to compute a result based on the received results (e. g. the average value). However, as this computation is very application specific, thus a generic communication mechanism can not be used. For that purpose the HRTS provides the application programmer with the capability to define appropriate functions for each data stream, which are applied to the set of results.

In *many-to-many communication*, a group of outputs disseminates its results to other group. Each value is the *opinion* that each member of the group has on the result of the computation. All these values must be correctly received by each receiving replicated component and each one must agree on the same result. Thus, an interactive consistency [10] algorithm must be used. Once again the programmer must define specific choosing functions.

For *1-to-1 communication* there is no need for specific algorithms besides a reliable transfer of data, as there is no replication. In the case of inter-component communication, it does not need to be agreed with its replicas (due to the use of timed messages).

As the use of the timed messages mechanism for replica management introduces the need for approximately synchronised clocks, the Communication Manager layer must also provide such a mechanism to the HRTS. The selected algorithms must be time-analysable, since they interfere not only with the message scheduling in the network, but also with the schedulability analysis in the DEAR-COTS nodes.

Furthermore, as the communication network is a shared resource between the DEAR-COTS nodes, its timing behaviour in the presence of failures must be analysed in order to guarantee that, under the assumed failure assumptions, the timing requirements of hard real-time applications are guaranteed. Otherwise network redundancy must also be provided. Currently these algorithms are being studied and their relationship with the HRTS upper layers is a work to be performed in the DEAR-COTS framework.

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. These subsystems are in separated memory spaces, in order to prevent errors in the SRTS to interfere with the HRTS behaviour.

Communication from the HRTS to the SRTS does not present major problems, 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. As the definition of what it is an erroneous data is once again very application specific, a generic mechanism can not be used. Appropriate filters must be defined for each data stream, which must be applied to the incoming data. As before, if the data is to be provided to replicated components, reliable communication algorithms must disseminate this data.

Interconnection with the controlled system is performed through the use of sensors and actuators. These devices can be linked to some of the DEAR-COTS nodes, but also directly to the real-time network. However, in the case of direct links to the real-time network, the devices must be sufficiently intelligent in order to allow the programming of the DEAR-COTS reliable communication algorithms.

Sensor values can be treated as the output of components. The dissemination of the values must be performed using the algorithms identified in the previous section. However, the time at which the value is valid must also 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. It is out of the scope of the DEAR-COTS project to study actuator agreement whenever such agreement is made outside the computational system.

4 Conclusions

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

5 References

1. Powell, D. 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, 1994, 199-217.
2. Laprie, J. L. (ed.). Dependability: Basic Concepts and Terminology. Dependable Computing and Fault-Tolerant Systems, Vol. 5, Springer-Verlag, 1992.
3. Guerraoui, R. and Schiper, A. Software-Based replication for Fault Tolerance. IEEE Computer, April 1997, 68-74.
4. Audsley, N., Burns, A., Richardson, M., Tindell, K and Wellings, A., Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling. In Software Engineering Journal, Vol. 8, No. 5, pp. 285-292, 1993.
5. Poledna, S., Burns, A., Wellings, A. and Barrett, P., Replica Determinism and Flexible Scheduling in Hard Real-Time Dependable Systems. IEEE Transactions on Computers, Vol. 49, N. 2, pp 100-111, 2000.
6. Tindell, K. and Clark, J., Holistic Schedulability Analysis for Distributed Hard Real-Time Systems. In Microprocessors and Microprogramming, Vol. 40, pp. 117-134, 1994.
7. Sha, L., Rajkumar, R. and Lehoczky, J. Priority Inheritance Protocols: An Approach to Real-Time Synchronization. IEEE Transactions on Computers, Vol. 39, N. 9, pp. 1175-1185, 1990.
8. Tindell, K., Adding Time-Offsets to Schedulability Analysis. Department of Computer Science, University of York, Technical Report YCS-221, 1994.
9. Wellings, A., Beus-Dukic, Lj. and Powell, D., Real-Time Scheduling in a Generic Fault-Tolerant Architecture. Proceedings of the Real-Time Systems Symposium, Madrid, Spain, 1998
10. Pease, M., Shostak, R. and Lamport, L., Reaching Agreement in the presence of Faults. Journal of the ACM, Vol. 27, N. 2, pp 228-234, 1980.