IPP-HURRAY! Research Group

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

Distributed Computer-Controlled Systems: the DEAR-COTS Approach

Paulo VERÍSSIMO (FCUL)
António CASIMIRO (FCUL)
Luís Miguel PINHO
Francisco VASQUES (FEUP)
Luís RODRIGUES (FCUL)
Eduardo TOVAR

HURRAY-TR-0021

October 2000

relatório técnico

technical report

THIS WORK IS PARTIALLY SUPPORTED BY FCT UNDER PROJECT DEAR-COTS (Praxis/P/EEI/14187/1998)

Distributed Computer-Controlled Sytems: the DEAR-COTS Approach

Paulo VERÍSSIMO, António CASIMIRO, Luís RODRIGUES

University of Lisboa, FCUL
Bloco C5, Campo Grande,
1749-016 Lisboa
Portugal
Tel.: +351.21.7500087
E-mail: {pjv,casim,ler}@di.fc.ul.pt
http://www.hurray.isep.ipp.pt

Luís Miguel PINHO, Eduardo TOVAR

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, llf}@dei.isep.ipp.pt
http://www.hurray.isep.ipp.pt

Francisco VASQUES

University of Porto (FEUP)
Rua Dr. Roberto Frias
4050-123 Porto
Portugal
Tel.: +351.22.5081702, Fax:
E-mail: vasques@fe.up.pt
http://www.fe.up.pt/~vasques

Abstract:
This paper proposes a new architecture targeting real-time and reliable Distributed Computer-Controlled Systems (DCCS). This architecture provides a structured approach for the integration of soft and/or hard real-time applications with Commercial Off-The-Shelf (COTS) components. The Timely Computing Base model is used as the reference model to deal with the heterogeneity of system components with respect to guaranteeing the timeliness of applications. The reliability and availability requirements of hard real-time applications are guaranteed by a software-based fault-tolerance approach.

DISTRIBUTED COMPUTER-CONTROLLED SYSTEMS: THE DEAR-COTS APPROACH [1]

P. Veríssimo*, A. Casimiro*, L. M. Pinho, F. Vasques***, L. Rodrigues*, E. Tovar****

** University of Lisboa, FCUL, Bloco C5, Campo Grande, 1749-016 Lisboa, Portugal, E-mail: pjv,casim,ler@di.fc.ul.pt*
*** Polytechnic Institute of Porto, ISEP, Rua de São Tomé, 4200-072 Porto, Portugal, E-mail: lpinho,emt@dei.isep.ipp.pt*
**** University of Porto, FEUP, Rua Dr. Roberto Frias, 4200-465 Porto, Portugal, E-mail: vasques@fe.up.pt*

Abstract: This paper proposes a new architecture targeting real-time and reliable Distributed Computer-Controlled Systems (DCCS). This architecture provides a structured approach for the integration of soft and/or hard real-time applications with Commercial Off-The-Shelf (COTS) components. The Timely Computing Base model is used as the reference model to deal with the heterogeneity of system components with respect to guaranteeing the timeliness of applications. The reliability and availability requirements of hard real-time applications are guaranteed by a software-based fault-tolerance approach.

Keywords: Real-Time, Fault Tolerance, Distributed Embedded Systems, COTS

1. INTRODUCTION

Currently, there is a trend to incorporate Commercial Off-The-Shelf (COTS) components in Distributed Computer-Controlled Systems (DCCS), in order to minimise costs and development time. Therefore, there is the need for architectures allowing the integration of COTS components, hard real-time reliable applications and non-reliable and/or soft real-time applications.

This paper proposes the DEAR-COTS (Distributed Embedded ARchitecture using Commercial Off-The-Shelf components) architecture for supporting real-time and reliable DCCS. This architecture provides DCCS with a generic framework, in which hard real-time applications can execute, guaranteeing their timeliness and reliability requirements, whilst, at the same time, embodying soft real-time or non real-time applications. This framework is put together by means of a Timely Computing Base (TCB), which is used as the reference model to deal with the heterogeneity of system components with respect to guaranteeing the timeliness of applications.

The remainder of this paper is organised as follows. Section 2 presents some of the relevant work concerning dependable distributed real-time systems. Afterwards, Section 3 presents the proposed architecture. Section 4 and 5 present the Hard Real-Time and Soft Real-Time Subsystems of the DEAR-COTS architecture. Finally, Section 6 draws some conclusions and points out possible future directions.

2. RELATED WORK

The problem of reliable real-time DCCS is reasonably well understood when considering synchronous and predictable environments. However, the use of COTS components and the integration,

[1] This work was partially supported by the FCT, through project Praxis/P/EEI/14187/1998 (DEAR-COTS).

in the same environment, of hard real-time and soft or non real-time applications introduces new problems.

The guarantee of these properties cannot be achieved by an "ad-hoc" solution, as it has been shown by structured approaches to the problem of designing dependable real-time systems (e.g. DELTA-4 (Powell, 1991), MARS (Kopetz *et al.*, 1989) or GUARDS (Powell *et al.*, 1999)). However, specialised architectures are costly, thus, there is the need for a fault-tolerant real-time distributed architecture for DCCS, based on highly available and low cost COTS technologies.

A few works have addressed the implementation of timeliness requirements in environments where no real-time guarantees can be given (Cristian and Fetzer, 1999; Veríssimo and Almeida, 1995). But the Timely Computing Base model presented in (Veríssimo *et al.*, 2000) provides a generic framework to deal with this problem. It is used as a reference model in the design of the DEAR-COTS architecture. The appropriateness of the TCB model to COTS-based environments is discussed in (Casimiro *et al.*, 2000) for a distributed system composed exclusively by COTS components It concludes that the system must be able to cope with the occurrence of residual timing failures.

Fault tolerance is the preferred means to achieve reliability. A fault-tolerant service can be implemented by co-ordinating a group of processes replicated on different nodes. There are three main replication approaches: active replication, primary-backup (passive) replication and semi-active replication (Powell, 1991). The use of COTS induces fail-uncontrolled replicas, so it becomes necessary the use of active replication techniques. In active replication, all the replicas process the same inputs, keeping their internal state synchronised and voting all on the same outputs.

It is also clear that the feasibility of the applications' requirements must be ensured by sound schedulability analysis techniques. The use of these techniques (e.g., the well-known Response Time Analysis (Audsley *et al.*, 1993)) allows checking if the task set will meet its deadlines, a required condition for hard real-time applications.

3. THE DEAR-COTS ARCHITECTURE

The DEAR-COTS architecture provides an execution environment for real-time applications, with reliability and availability requirements, through the use of COTS hardware and software. Its main purpose is to provide continuous and adequate service to the controlled system, in order to increase the confidence level put in the controlling system. The DEAR-COTS architecture 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).

Besides the reliability and availability requirements to guarantee the correct behaviour of the supported real-time applications, DCCS also needs 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 DEAR-COTS architecture.

A common characteristic among all these applications is their real-time behaviour. This real-time behaviour is specified in compliance with timeliness requirements, which in essence call for a synchronous system model. However, the demand for an environment with flexibility and interconnectivity capabilities and based on possibly heterogeneous COTS components, makes the enforcement of timeliness assumptions very difficult. The Timely Computing Base (TCB) model provides a generic solution to this problem.

In the DEAR-COTS architecture the TCB model is used as a reference model to deal with the heterogeneity of system components and of the environment, with respect to timeliness. From a system model perspective we devise a generic DEAR-COTS architecture to address the fundamental problems in a global and integrated way. From an engineering point of view, we devise specific mechanisms to deal with the reliability and availability requirements of hard real-time applications, and to deal with the requirements of soft real-time applications.

3.1 *The TCB model*

The TCB model provides a generic framework to deal with the problem of implementing applications with timeliness requirements in environments that are unpredictable or unreliable. The fundamental idea, which is applied to the DEAR-COTS architecture, is that systems are assumed to have two distinct parts with respect to synchrony. In a system with a TCB there is a *control* part, with synchronous properties, made of local TCB modules interconnected by a *control* channel. There is also a *payload* part, over a global network or *payload* channel, that may have any degree of synchronism. In particular, the payload part can either enjoy synchronous properties but can also be completely asynchronous.

The synchronous part is supposed to be a very small and simple part of the overall system. Therefore, its synchronous properties can be en-

forced with much higher coverage than if the synchronous properties would have to be assumed for the overall system. Applications execute in the payload part of the system and use a set of basic services provided by the control part, the TCB: *Timely Execution*, *Duration Measurement* and *Timing Failure Detection*. The definition of these services and the respective API can be found in (Veríssimo *et al.*, 2000).

Applications can benefit from the TCB by construction, using TCB services to "be aware" of their timeliness and, therefore, to behave always in a manner that preserves the safety of the system. This is the case of many soft real-time applications, namely all those that can live with sporadic failures, that can adapt to the available quality of service, or those that can switch to a fail-safe state when some timing failure is detected.

In fact, timing faults affecting components or applications with soft real-time requirements, can be addressed with the help of a timing failure detection service and by applying adequate tolerance or safety measures, as explained in section 5.

3.2 *A generic DEAR-COTS node*

Given the above description of the TCB model, the architecture of a generic DEAR-COTS node, capable of simultaneously handle the requirements of hard and soft real-time applications, can be understood in a very intuitive manner. In fact, the basic idea of casting into the architecture the heterogeneity in system synchrony, has been applied to the generic DEAR-COTS node, as represented by Figure 1.

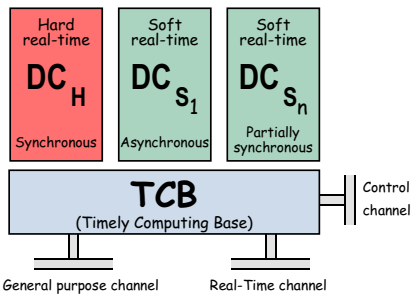

Fig. 1. DEAR-COTS node structure.

The several modules depicted in the figure are intended to fully characterise a node in terms of the synchrony assumptions. The TCB module acts as a gluing element, being always the most synchronous part of the system. It provides timely services to less synchronous modules through an interface that bridges the synchrony gap. The DC_H, DC_{S_1} and DC_{S_n} modules constitute the payload part of the system, and represent the several possible environments with respect to synchrony that may exist in a DEAR-COTS node. Each of these modules can be seen as a particular subsystem on which applications with different requirements will be executed. DC_H represents a hard real-time subsystem (HRTS), with synchronous properties, which is supposed to exist in every node running distributed and/or replicated hard real-time applications. Different DC_S subsystems can be defined, accordingly to the synchrony properties that they enjoy, namely the asynchronous one, DC_{S_1}, and several partial synchrony models, DC_{S_n} (Cristian and Fetzer, 1999; Veríssimo and Almeida, 1995). With the help of a TCB, all these subsystems (including the asynchronous one) can support the execution of certain applications with timeliness requirements. Therefore, we will refer to all of them as soft real-time subsystems (SRTS).

Each node may have access to different communication channels with respect to synchrony. As in the TCB model, we assume that TCB modules communicate through a control channel, which can be implemented as a virtual channel over the physical (real-time) network or can be a separate network by itself. There is also a real-time channel serving DC_H subsystems, implemented over a real-time network, and a general purpose channel serving DC_S subsystems, where no real-time guarantees are provided.

This generic node architecture can indeed be instantiated into different node configurations, depending on the modules used in a node. A DEAR-COTS system is built by interconnecting several DEAR-COTS nodes, choosing suitable configurations for each node.

3.3 *Implementing a DEAR-COTS system*

A DEAR-COTS system is built using distributed processing nodes, where distributed hard real-time and soft real-time applications may coexist. 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. To ensure the timeliness properties of soft real-time applications and allow system resources to be shared by every (soft and hard real-time) application, processing nodes are modelled as DEAR-COTS nodes.

From a generic DEAR-COTS node it is possible to define particular node instances, with different characteristics and purposes. A DEAR-COTS node is characterised by the synchrony subsystems it is composed of. There are essentially three basic node types: Hard real-time nodes (H), soft real-time nodes (S) and gateway nodes (H/S).

Hard real-time nodes are those where only a hard real-time subsystem (HRTS) exists. Therefore, they will exclusively be used to provide a framework to support reliable and available hard real-time applications, which are the core of the

DCCS application. The existence of a TCB in these nodes is not essential to allow the implementation of hard real-time applications. However, as in any system assumed to be synchronous, and even more in a COTS based one, there is always a small probability that timing failures occur. The need for a TCB, as an instrument to amplify the coverage of synchronous assumptions, has to be equated in terms of the dependability required by the supported DCCS application.

Soft real-time nodes only include a soft real-time subsystem (SRTS). The soft real-time subsystem provides the execution environment for the remote supervision and remote management of the Distributed Computer Control System. The existence of a TCB in these nodes is crucial to allow the implementation of applications with timeliness requirements.

A gateway node integrates both a hard real-time subsystem (HRTS) and a soft real-time subsystem (SRTS). In these nodes there are two distinct and well-defined execution environments. The idea is to allow hard real-time components, executing in the HRTS, to interact in a controlled manner with soft real-time components, executing in the SRTS. 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.

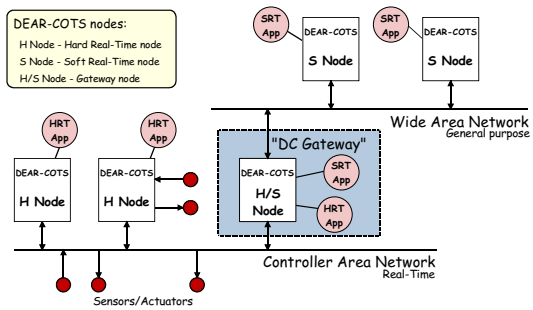

Fig. 2. Generic DEAR-COTS system.

Figure 2 represents a system containing all the above node configurations. H and H/S 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 H/S and S nodes. The control channel required for the TCB is not necessarily an independent network. The assumption of a restricted channel with predictable timing characteristics (control) coexisting with possibly asynchronous channels is feasible in some of the current networks (Prycker, 1995).

At the hard real-time level, the HRTS is responsible for providing 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 SRTS provides a set of services to support the supervision and management level of the DCCS. At this system level, flexibility is a major goal, since new services can be provided as the system evolves. However, since there are no hard real-time guarantees, either the TCB services are used to allow applications to be aware of the available quality of service (and adapt themselves, if possible), or techniques such as best-effort scheduling or value-based scheduling must be used.

4. THE HARD REAL-TIME SUBSYSTEM

The set of H and H/S nodes provides a framework to support distributed hard real-time applications (Figure 3). The reliability and availability requirements are guaranteed by the software replication of COTS components, rather than the usual solution, which is to build software on top of specialised hardware.

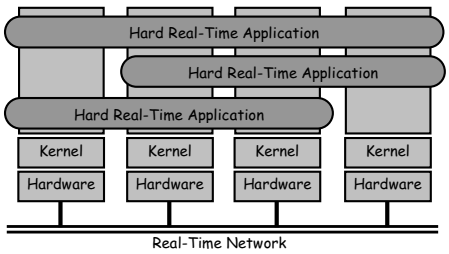

Fig. 3. HRTS structure.

One hard real-time application is constituted by several tasks (processing units). These tasks are distributed over the H and H/S nodes. Each node has its own (non-distributed) COTS real-time kernel and hardware, which provides the desired multitasking support. An advantage of using both a COTS kernel and hardware is that it allows for the easy upgradability and portability of the system.

The set of H and H/S nodes also supports the active replication of software (Figure 4) with dissimilar replicated task sets in each node. By providing

different execution environments in each node, the tolerance to design faults is increased, since nodes are considered as independent from the point of view of failures. At the same time flexibility is increased, since nodes are not just copies of each other, allowing for a more flexible design of real-time applications.

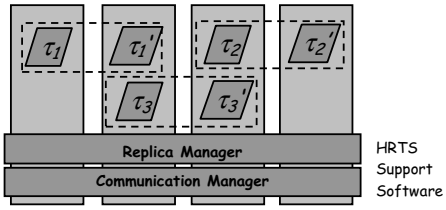

Fig. 4. Replicated Hard Real-Time Application.

The goal of the HRTS support software (Figure 4) is to provide the distribution support (including both the application distribution itself and the replication management) to hard real-time applications. A layered approach to the HRTS is provided to simplify the development of the HRTS support software:

- The Communication Manager layer is responsible for the adequate communication services;
- The Replica Manager layer is responsible for transparently manage the replicated components.

4.1 *Scheduling model*

In the HRTS, 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. In order to allow the use of current off-line schedulability analysis techniques (e.g., 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 (remote blocking).

Tasks are allowed to communicate with each other either through shared 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 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. Internal blocking can be bounded and off-line analysed through the use of Priority Ceiling Protocols (Sha *et al.*, 1990).

4.2 *Replication Model*

As there is the target of reliability through replication, it is important to define the replication unit. From the above definitions, two different entities could be considered: a single task or the complete hard real-time application. However, both 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 *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 5 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 (C_1 and C_2), which ensure the desired level of reliability.

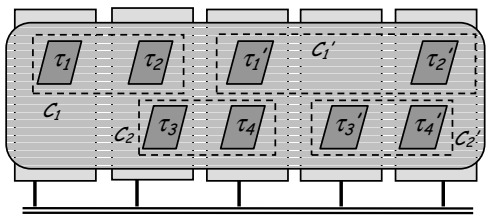

Fig. 5. Hard real-time application.

A similar concept is the "capsule" defined in the Delta-4 architecture (Powell, 1991). As the DEAR-COTS "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 real-time application, accordingly to the reliability of its components and the desired level of reliability for the application. However, by replicating components, efficiency decreases as the number of tasks and messages increases. 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.

Nevertheless, as active replication is used, there is the need to guarantee determinism, *i.e.*, that replicated tasks execute with the same data and timing-related decisions are the same in each replica. The use of timed messages (Poledna *et al.*, 2000) allows a restricted model of multitasking to be used and 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 clock synchronisation algorithms, since clock deviations must be bounded.

4.3 *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.

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.

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, communication streams that need timed messages are identified.

The hindrance of this approach is that, because the programmer is not aware of the possible distribution and replication, complex applications could be built relying heavily in task interaction. However, the model for tasks (presented in Section 4.1), where task interaction is minimised, precludes such applications.

4.4 *HRTS Communication Manager*

The Communication Manager layer is responsible for the adequate communication services, providing a reliable and timely transfer of real-time data. The group communication abstraction can be used as the framework for reliable communication, and also for replica management (Powell, 1994). 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 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 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. However, as this computation is very application-specific, the application programmer defines 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. Thus, an interactive consistency 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.

The suitability of the Controller Area Network (CAN) (ISO, 1993) for the communication infrastructure of the architecture is being studied. Although current results indicate that CAN presents some problems, as it is not resilient to station errors, it is perceived that, with the appropriate fault assumptions, it can be used as the communication infrastructure for the architecture (Pinho *et al.*, 2000).

4.5 *Interconnection with the outside world*

The interconnection of the HRTS with the SRTS in an H/S node 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 is an erroneous data is very application-specific, 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. 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.

4.6 *Composition of real-time protocols*

From the point of view of protocol design the run-time must provide a framework to support the clean composition of micro-protocols. This encourages the re-use of protocol components and allows the applications to configure protocol stacks exactly tailored to their needs. This aspect is particularly relevant in the context of real-time applications where, due to memory and power consumption constraints, it is interesting to execute in each component just the protocol layers required to support the intended functionality.

The x-Kernel (Hutchinson and Peterson, 1991) is an early and influential work on protocol composition. A version of x-Kernel adapted to real-time operation has been developed in the scope of the CORDS project (Travostino *et al.*, 1996). Following a similar approach, we are developing *RT-Appia* (Rodrigues *et al.*, 2000), a modern architecture that attempts to balance the flexibility and efficiency of micro-protocols with the predictability requirements of real-time applications.

In *RT-Appia*, a communication channel is an ordered sequence of *sessions*, instances of *protocol layers*. Sessions communicate through the exchange of events. Each channel is executed in the context of a single thread with a fixed and pre-allocated set of memory resources. Layers declare which events need to be subscribed. This ensured that the events are only processed by the relevant layers. Additionally, this knowledge is used to derive the worst case execution time of each channel activation, based on the processing time of each event and on the worst case chain of events that may be produced in response to a stimuli (from the application, the network or the timer).

5. THE SOFT REAL-TIME SUBSYSTEM

The main goal of the SRTS subsystem is to support the execution of soft real-time applications. The challenging issue is to ensure that applications can have a real-time behaviour despite the occurrence of timing failures.

Timing failures, in a fault-tolerance sense, can be handled either by masking, detection and/or recovery techniques. A generic approach to *timing fault tolerance*, that is, one that can use any or all of the above techniques, requires attributes such as *timeliness*, to act upon failures within a bounded delay, *completeness*, to ensure that failure detection is seen by all participants and *accuracy*, not to detect failures wrongly. In DEAR-COTS, these attributes are mostly ensured by the TCB module and its services.

Separating the mechanisms of timing failure into delay, uncoverage and contamination, allows the introduction of classes of applications that deal with combinations of the former, achieving varying degrees of dependability, when assisted by a TCB: **fail-safe**, which exhibits correct behavior or else stops in fail-safe state; **time-elastic**, which exhibits coverage stability; and **time-safe**, which exhibits no-contamination. In the DEAR-COTS architecture, the idea consists in mapping these application classes (or combinations thereof) into the above-mentioned fault-tolerance techniques. This can be done independently of the synchronism of the SRTS, and allows several degrees of fault tolerance to be achieved, namely:

- Fail-safe operation: by switching to a fail-safe state after the first failure. Requires the timing failure detection service and applications to be of the fail-safe class;
- Reconfiguration and adaptation: by enforcing coverage stability, adapting essential timing variables to environment conditions. Requires applications to be of the time-elastic and time-safe classes.
- Timing error masking: by using replication to mask transient timing errors. Requires accurate timing failure detection and time-safety.

6. CONCLUSIONS

This paper has presented the DEAR-COTS architecture, targeted to the development of reliable distributed computer-controlled systems. It is based in the use of COTS components, and provides a generic framework, in which hard real-time applications can execute, whilst, at the same time,

allowing soft or non real-time applications in the system, without interfering with the guarantees provided to hard real-time applications.

DEAR-COTS systems are built using distributed processing nodes, where a mixture of hard real-time and soft real-time applications may execute. The Timely Computing Base model is used as a reference model to deal with the heterogeneity of system components and of the environment with respect to guaranteeing the timeliness of applications.

The paper also describes the main components of the architecture, the Hard Real-Time and Soft Real-Time Subsystems, and its integration in actual implementations of the architecture. The use of the DEAR-COTS architecture allows to fully integrate COTS components and non hard real-time applications with the timeliness and reliability requirements of DCCS.

7. REFERENCES

Audsley, A. N., A. Burns, M. Richardson, K. Tindell and A. Wellings (1993). Applying new scheduling theory to static priority preemptive scheduling. *Software Engineering Journal* **8**(5), 285–292.

Casimiro, A., P. Martins and P. Veríssimo (2000). How to build a Timely Computing Base using Real-Time Linux. In: *Proceedings of the 3rd IEEE International Workshop on Factory Communication Systems*. Porto, Portugal. pp. 127–134.

Cristian, F. and C. Fetzer (1999). The timed asynchronous distributed system model. *IEEE Transactions on Parallel and Distributed Systems* **10**(6), 642–657. Special Issue on Dependable Real-Time Systems.

Hutchinson, N. and L. Peterson (1991). The x-kernel: An architecture for implementing network protocols. *IEEE Transactions on Software Engeneering* **17**(1), 64–76.

ISO (1993). International standard 11898 - road vehicles - interchange of digital information - controller area network (can) for high-speed communication. Technical report.

Kopetz, H., A. Damm, C. Koza, M. Mulazzani, W. Schwabl, C. Senft and R. Zainlinger (1989). Distributed Fault-Tolerant Real-Time Systems: The Mars Approach. *IEEE Micro* **9**(1), 25–41.

Laprie, J. C., Ed.) (1992). *Dependability: Basic Concepts and Terminology*. Dependable Computing and Fault Tolerance. Springer-Verlag, Berlin Germany.

Pinho, L. M., F. Vasques and E. Tovar (2000). 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.

Poledna, S., A. Burns, Wellings A. and Barrett P. (2000). Replica determinism and flexible scheduling in hard real-time dependable systems. *IEEE Transactions on Computers* **49**(2), 100–111.

Powell, D. (1994). Distributed fault tolerance — lessons learnt from delta-4. *Lecture Notes in Computer Science* **774**, 199–217.

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

Powell, D., J. Arlat, L. Beus-Dukic, A. Bondavalli, P. Coppola, A. Fantechi, E. Jenn, C. Rabéjac and A. Wellings (1999). GUARDS: A Generic Upgradable Architecture for Real-Time Dependable Systems. *IEEE Transactions on Parallel and Distributed Systems* **10**(6), 580–599.

Prycker, M. de (1995). *Asynchronous Transfer Mode: Solution For Broadband ISDN*. third ed.. Prentice-Hall.

Rodrigues, J., H. Miranda, J. Ventura and L. Rodrigues (2000). The design of rt-appia. Technical report. Universidade de Lisboa, Faculdade de Ciências.

Sha, L., R. Rajkumar and J. P. Lehoczky (1990). Priority inheritance protocols: An approach to real-time synchronization. *IEEE Transactions on Computers* **39**(9), 1175–1185.

Travostino, F., E. Menze and F. Reynolds (1996). Paths: Programming with system resources in support of real-time distributed applications. In: *Proceedings of the 2nd IEEE Workshop on Object-Oriented Real-Time Dependable Systems*. Laguna Beach, CA.

Veríssimo, P., A. Casimiro and C. Fetzer (2000). The Timely Computing Base: Timely actions in the presence of uncertain timeliness. In: *Proceedings of the International Conference on Dependable Systems and Networks*. IEEE Computer Society Press. New York City, USA. pp. 533–542.

Veríssimo, P. and C. Almeida (1995). Quasi-synchronism: a step away from the traditional fault-tolerant real-time system models. *Bulletin of the Technical Committee on Operating Systems and Application Environments (TCOS)* **7**(4), 35–39.

Wellings, A., Lj. Beus-Dukic and D. Powell (1998). Real-time scheduling in a generic fault-tolerant architecture. In: *Proceedings of the 19th IEEE Real-Time Systems Symposium*. Madrid, Spain.