Dependable Adaptive Real-Time Applications in Wormhole-based Systems

Pedro Martins
pmartins@di.fc.ul.pt
Univ. of Lisboa*

Paulo Sousa
pjsousa@di.fc.ul.pt
Univ. of Lisboa

António Casimiro
casim@di.fc.ul.pt
Univ. of Lisboa

Paulo Veríssimo
pjv@di.fc.ul.pt
Univ. of Lisboa

Abstract

This paper describes and discusses the work carried on in the context of the CORTEX project, for the development of adaptive real-time applications in wormhole based systems. The architecture of CORTEX relies on the existence of a timeliness wormhole, called Timely Computing Base (TCB), which we have described in previous papers. Here we focus on the practical demonstration of the wormhole concept, through a demo with two complementary facets. The objective is to illustrate the effectiveness of the concept from a practical, yet rigorous, perspective, which is done with the help of an emulation framework that we present in the paper. Furthermore, the paper also describes two different ways of implementing timeliness wormholes on top of both wired and wireless infrastructures.

1. Introduction

The convergence of several factors, including advances in wireless and sensory technologies, is creating the conditions for the construction of highly autonomous and proactive applications, such as those considered in the context of the IST CORTEX project [9], under which we have developed, applied and experimented our work.

CORTEX assumes that a new class of (*sentient*) applications can be envisaged, which is composed of a (possibly large) number of smart components that are able to sense their surrounding environment and interact with it. These components, referred to as *sentient objects*, interact with each other and with the environment by accepting and producing events, and these interactions may occur in ways that demand predictable and sometimes guaranteed Quality of Service (QoS). The problem is that achieving predictability is made difficult by the characteristics of the changing environment in which these objects typically operate, dictated by unstable and mobile object population, unpredictable network load, varying connectivity, and the presence of failed system components.

In order to deal with this problem, the architecture of a CORTEX system explores the wormhole metaphor [7], considering the existence of a specific timeliness wormhole, called a *Timely Computing Base* (TCB) [10].

The design and development of CORTEX applications is done in accordance with the TCB model, where TCB components (which can be seen as local representations of the wormhole) provide a set of very simple (but crucial to timeliness) services to sentient objects.

In previous papers we have already addressed the fundamental concepts underlying the development of dependable applications in wormhole enabled systems [2, 3]. In this paper, the fundamental objective is to illustrate these concepts from a more practical point of view, which is done by describing and discussing a demonstration that was conceived in the context of the CORTEX project. The demonstration has two facets that illustrate not only the practical relevance of the concepts, but also the effectiveness of the solutions under a rigorously controlled scenario.

In addition, this paper also addresses an issue that due to its importance has already received some attention in previous papers: the feasibility of TCB wormholes. Since the demonstration makes use of two distinct TCB instantiations, which rely on both wired and wireless infrastructures, we discuss the most relevant aspects of these two implementations.

The paper is organized as follows. In Section 2 we review key issues for programming with TCB wormholes. Then, in Section 3 we describe the two TCB implementations used in this demo. After that, Section 4 is totally devoted to the presentation and discussion of the demonstration. Finally, Section 5 concludes the paper.

2. The TCB timeliness wormhole

The idea of wormhole-based systems has been introduced in [8]. Essentially, the wormhole metaphor helps illustrating a possible way of constructing distributed systems, by following a few guiding principles that address the problems posed by uncertainty. First, assume that uncertainty is not ubiquitous and is not everlasting— this means

* Faculdade de Ciências da Universidade de Lisboa. Bloco C6, Campo Grande, 1749-016 Lisboa, Portugal. Navigators Home Page: http://www.navigators.di.fc.ul.pt. This work was partially supported by the EC, through project IST-FET-2000-26031 (CORTEX), by the FCT, through the Large-Scale Informatic Systems Laboratory (LaSIGE), and by Microsoft Research Ltd, UK.

that the system has some parts which are more predictable than others and tends to stabilize. Then, be proactive in achieving predictability— in concrete, make predictability happen at the right time, right place. These more predictable parts can be seen as wormholes, through which it is possible to do things much faster or reliably than apparently possible in the other parts of the system.

The wormhole concept can in fact be instantiated in different ways. For example, when applied in the security domain, a wormhole takes the form of a security kernel, a trusted component, as described in [4]. On the other hand, when timeliness is the relevant non-functional property to secure, the wormhole should essentially be timely. Note again that these "better" properties (timeliness, trustworthiness) must be obtained *by construction* (making some parts be more predictable). In this work we are concerned with timeliness and, therefore, we focus on the TCB timeliness wormhole (thoroughly discussed in [10]), to which we will refer in the remainder of this paper simply as the TCB.

2.1. Architecture, properties and services

The architecture of a system with a TCB has a generic or *payload* part, in which protocols and application processes are executed. Communication takes place over a global network or *payload* channel. The system also has a *control* part, made of local TCB modules, interconnected by some form of medium, the *control* channel. Processes execute on the several sites, in the payload part, making use of the TCB whenever appropriate.

In a system with a TCB the payload part *can have any degree of synchronism*. On the other hand, the TCB subsystem (the control part) enjoys, by construction, a few synchronism properties: known upper bounds on processing delays, on the rate of drift of local clocks and on message delivery delays.

Given that the TCB is a comparably small part of the system (recall the wormhole concept), it provides just the following essential services: *timely execution*, *duration measurement* and *timing failure detection*. These services are provided to processes via the local TCB components, even when they have a distributed scope.

2.2. Dependable applications in TCB-based systems

The availability of a TCB may be exploited to handle the negative effects of the lack of synchrony and reliability of the payload system. When *timing failures* occur, there are essentially three kinds of problems that can arise: *unexpected delays* (the most immediate effect), *decreased coverage* and *contamination*.

Unexpected delays correspond to the violation of timeliness properties. Decreased coverage can be explained as follows. When we make assumptions about the absence of timing failures, we have in mind a certain coverage, which is the correspondence between system timeliness assumptions and what the environment can guarantee. If the environment conditions start degrading to states worse than assumed, the probability of timing failure increases, that is, the coverage incrementally decreases. Contamination occurs when a safety property of the system is violated on account of the occurrence of timing failures.

In order to avoid these problems, it is sufficient to ensure that coverage stays close to the assumed value, a condition expressed by the *Coverage Stability property*, or that the effect of timing failures is confined to the violation of timeliness properties alone, as specified by the *No-Contamination property*.

Unfortunately, not all applications can enjoy these properties and escape the consequences of uncertain synchrony. However, when assisted by a TCB, some applications classes can deal with the effects of timing failures and achieve varying degrees of dependability. For example, the *fail-safe* class, which exhibits correct behavior or else stops in fail-safe state, the *time-elastic* class, which exhibits coverage stability, and the *time-safe* class, which exhibits no-contamination.

In the demonstration described in Section 4 we will observe applications of the fail-safe and time-elastic classes and the way in which they use the TCB to behave in a dependable way. In particular, we will see how the TCB-based QoS coverage service introduced in [2] can be used to provide support for a *dependable adaptation* framework.

3. Construction of TCB wormholes

The feasibility of the TCB synchrony assumptions is an important issue, which has already received our attention in [1]. Now we review the implementation discussed in that paper, designed for communication over a wired infrastructure. Additionally, we describe a TCB implementation designed for wireless/mobile systems, which is used in the demonstration.

In order to secure the required synchrony properties, the TCB subsystem has to be constructed is such a way that its timeliness is not impaired by the rest of the system (the payload part). Furthermore, it is necessary to ensure the predictability of execution times within the TCB and communication delays over the control channel.

Concerning the predictability of execution times, it is fundamental to use operating systems and hardware platforms with the necessary support for real-time operation. So far, we have implemented TCB prototypes for the most popular real-time versions of the Linux operating system (RTAI and RT-Linux), executing over a PC architecture, and also for the Windows CE operating system, executing over iPAQ Pocket PCs. These operating systems provide the necessary support in terms of priorities, scheduling and other features. Potential predictability problems caused by PC hardware (e.g. loose control over shared buses) can be addressed

with specific solutions, as exemplified in [6] for the case of the PCI bus.

In the currently available TCBs, the synchronous communication channel is based on a physically different network from the one supporting the *payload* channel. Therefore, the problem of achieving predictability is much alleviated, since this allows to establish an upper bound on the network load.

We developed TCBs for both wired and wireless ad hoc networks. In the former case, we exploit some specific characteristics of switched Fast-Ethernet networks (absence of collisions in micro-segmented topologies when operating in full-duplex mode), to enforce predictability at the MAC level. Message scheduling problems are automatically solved centrally at the switch.

Ensuring the timely operation of the TCB over wireless networks is a more complex task. In fact, the most commonly available technology for wireless LANs in existence today (IEEE 802.11b) does not support real-time communication when operating in ad hoc modes.

There are some proposals to endow ad hoc networks with real-time capabilities, such as the *Time-Bounded Medium Access Control* protocol (TBMAC) [5] or the upcoming IEEE 802.11e [11] standard, which introduces the concept of traffic categories. However, since no implementation of TBMAC nor 802.11e have yet been made available, we developed a *mock-up* of a real-time MAC protocol for ad hoc wireless networks. Under certain conditions, this *mock-up* provides the desired properties and thus fully serves the purposes of the demo presented in Section 4.3. More specifically, we use a simple *token-based* algorithm on top of a regular 802.11b channel to enforce a predictable medium access latency and we assume that it is possible to isolate the control channel from the payload channel using a dual network architecture, with two non-overlapping 802.11b ad hoc wireless networks.

4. Demonstration setup

4.1. Emulation system

Several classes of sentient applications have strict dependability requirements that emerge from safety rules imposed by the environment and that must be preserved by the system under any circumstance — avoiding collisions between moving objects (e.g. planes, cars). In some of these applications, the computations must comply with timeliness constraints so that safety rules can be secured.

Testing such kind of sentient applications in real scenarios using real hardware may not be always possible, because of cost reasons and/or safety constraints. Therefore, we propose a software platform for the emulation of real environments, which can be a very useful and inexpensive tool to test subtle coordination and synchronization phenomena, difficult to reproduce and/or follow in real-life systems.

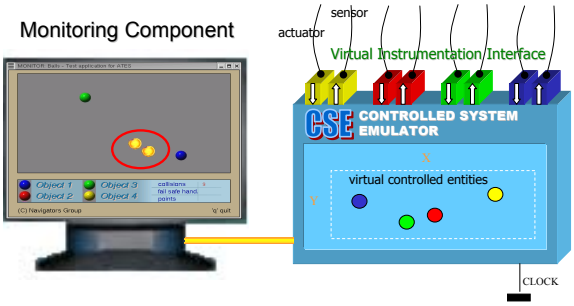

Figure 1. Emulation system.

The software emulator is illustrated in Figure 1. The central component, or module, that emulates a real environment is called *Controlled System Emulator* (CSE). To explain how the CSE can be used, we will focus on the concrete example of our demo. The environment emulated for the first part of the demonstration (presented in Section 4.2) consists of a bi-dimensional space delimited by walls in which four virtual controlled entities move at a certain speed and in several directions. Each entity is defined by four physical attributes: a position $\langle x,y \rangle$ in the plane, a shape, a speed and a direction. For simplicity, but without loss of generality, physical laws such as friction or kinetic energy transfers are not modelled in this environment. Therefore, in the absence of external control, entities tend to keep a constant speed and direction, unless they collide with a wall or with another moving entity, in which case they will change their direction.

The entities emulated inside the CSE are shaped as four colored balls: red, green, blue and yellow. They can represent real objects with similar dynamics, such as cars, robots or even sub-atomic particles. Initially, the balls start their movements with random speeds and directions.

The progress of the emulator is triggered by a periodic clock signal with a certain pre-defined frequency (which just affects our perception of the emulation speed, but not the emulation states). At each clock tick, the positions of all the controlled entities are updated accordingly to their evolution rules, dictated by their speed and direction.

The emulation system has a monitoring component (see Figure 1) that provides a representation of the emulated environment. Periodically, the monitor gathers the state of the emulated environment from the CSE and updates its graphical representation. Note that this graphical representation does not have to be updated with the same frequency as the frequency of emulator updates. In any case, it is guaranteed that all relevant events occurred in the emulated environment (e.g. ball collisions) will eventually be notified.

The CSE provides a virtual instrumentation interface composed of sensors and actuators to interact with the emulated environment. In our example, the CSE provides a sensor/actuator pair for each emulated ball. Through the sensor a control application can know the position, speed and di-

rection of a ball and through the actuator it can change the movement (speed and/or direction) of the emulated ball.

In distributed control applications (e.g. if each ball is controlled from a different node), each sensor/actuator pair can be remotely accessed through a remote interface made available to each controller entity (each ball controller). In order to ensure bounded errors when sensing and actuating on the emulated environment from a remote interface, it is necessary to guarantee a timely connection between the remote instrumentation interface and the CSE central unit.

4.2. Sentient computing using emulated balls

In what follows we describe an application composed of four sentient objects that control the movement of the emulated balls, trying to avoid collisions among them. This is the safety rule that must be ensured in this application.

The infrastructure that was used to build the emulation system and this application is illustrated in Figure 2. The emulator runs in a dedicated machine that executes a real-time version of the Linux operating system (RTAI-Linux). The emulator CSE component runs as a hard real-time task of RTAI kernel and is executed periodically (this establishes the clock signal of the CSE). The monitoring application is an *X11* application (user-level process) that periodically acquires the state of the CSE and displays a graphical representation of the emulated space and balls.

Each CORTEX machine runs a sentient object that controls one of the emulated balls. As explained above, remote access is provided through a remote instrumentation interface. Dedicated real-time communication channels have been setup to allow a timely remote sensing/actuation.

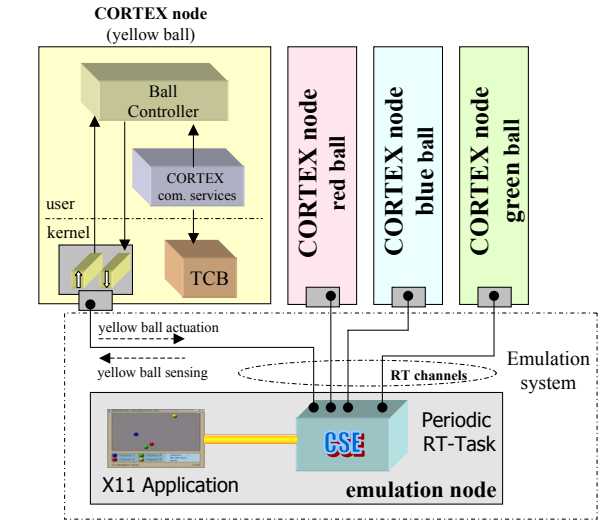

Figure 2. Balls demo setup.

Sentient objects communicate with each other through the payload channel. We use CORTEX communication services to publish and subscribe information concerned with the control algorithm. A sentient object publishes the position, speed and direction of the ball it controls, and subscribes the same information from the other sentient objects. This allows every sentient object to build a local image of the overall system. This image will be used to decide when and how to actuate on the controlled ball.

In this application scenario, it is fundamental to ensure that every ball controller has a consistent view (in time and space) of the environment, so that it takes the correct control decisions in order to avoid collisions. Therefore, the communication and processing needed to construct the local (real-time) image of the environment has to fulfill some timeliness requirements. For the control algorithm it is sufficient to have a representation of each ball with a known and bounded positioning error. To achieve that, a maximum ball speed must be defined and each controller must perform the following real-time computations: 1) publish the information of its controlled ball with a given period (to refresh other controllers' real-time images); 2) timely receive the information published by the other controllers (to update on time its real-time image). This also implies: 3) timely actuating in response to received information (e.g. change the ball direction to avoid a possible collision).

The objective of the demo is to show that despite the lack of timeliness of the payload channel, we can use the TCB to secure the required safety property and avoid collisions.

Given that the application is of the fail-safe class (the fail-safe state corresponds to stopping a ball), every controller (sentient object) can use the *Timing Failure Detection* (TFD) service provided by the local TCB component in order to force the system to switch to a fail-safe state in a timely manner when a timing failure occurs (i.e. when 1), 2) or 3) are not met). The demonstration shows that a controller will timely stop its ball whenever timing failures are detected by the TCB, and that it will recover as soon as its real-time image becomes consistent again.

Although the safety of the system is ensured by using TFD primitives of the TCB, it is desirable to minimize the activation of fail-safety procedures. If possible, it is better to ensure that some progress is nevertheless made, even if in a degraded operational mode (e.g. slowing down the speed of a ball). In order to adapt the essential variables while ensuring coverage stability (thus, dependably adapting), the controllers use the QoS services provided by the TCB.

To observe the behavior of the application under several timeliness conditions, we have developed an additional demonstrator module that provides a better control of the demo. It allows to: 1) inject artificial timing failures (possibly omissions) on the payload communication channel; 2) enable or disable the use of TFD and/or QoS adaptation services; and 3) specify the desired level of coverage to be used in the QoS specification. The demo explores the several combinations, as summarized in Table 1.

TFD	QoS adapt.	artificial delays	description
ON	ON	no	The system behaves as expected, balls do not collide.
OFF	OFF	yes	The balls start colliding because timing failures are being generated and TFD is OFF.
ON	OFF	yes	The balls stop on fail-safe states because timing failures are being generated and TFD is ON. However, they keep blocked on fail-safe because QoS is OFF.
ON	ON	yes	The speed of the balls is adjusted accordingly to the injected delays.

Table 1. Balls demo guideline.

4.3. A cooperating cars scenario

So far we have demonstrated the fundamental mechanisms underlying the construction of adaptable real-time applications, with the help of a TCB. However, we believe that it is interesting to enrich the demonstration with a more realistic setup, still showing how fail-safety and time-elasticity characteristics of an application can be used to overcome the uncertainty of the environment in TCB-based systems.

This part of the demo is inspired in the vision of autonomous cars communicating and cooperating with other cars or entities in its proximity. The demo takes as background scenario a city area where only perpendicular streets exist. There are three cars with different colors (red, green or blue), advancing in a different street and always in the same direction (up, down, left or right). This virtual world is modelled as a small sphere, so that cars are continuously meeting with each other in the same street crossings.

All cars have the same control logic, advancing if the two following conditions are satisfied: 1) there is no car in front; and 2) there is no car approaching from the right. This control logic is only applied to a safety area around the car, to avoid having a car stopping in a crossing due to another one approaching from the right (or standing in front) but still very far. The car safety area depends on its speed— a higher speed results in a higher safety area.

In this scenario, each car periodically disseminates an event (with its position, speed and direction), which can be received and used by every car to construct a real-time image of the reality (an RT-image). These events must be timely delivered, in order to achieve this RT-image and avoid car crashes. Intuitively, time bounds for the dissemination of events are related to the sender speed: a higher speed requires a smaller deadline. Because we are using a TCB-based system, a car can detect deadline violations of the events it is supposed to receive (by using the TFD service). Then, it will (timely) switch to a fail-safe state (e.g. by stopping) because its RT-image may no longer be consistent with the reality. Moreover, each car can (and should) also adapt its speed to the environment conditions, following the same reasoning of the first part of the demonstration.

In addition, this demo application also illustrates a *coverage awareness* mode of operation, privileging the knowledge of current coverage level, without requiring car speed changes.

In the demo, each car is represented by an IPAQ. The actual position of the device is simulated in this demo, just because the IPAQs are not equipped with a location mechanism such as the one provided by a GPS receiver. The speed is also simulated, and changeable through the IPAQ's car interface (see Figure 4a). We have used the Windows CE TCB implementation to provide each IPAQ with a local TCB component. Therefore, IPAQs communicate using the 802.11b dual network architecture described in Section 3. Beyond the IPAQs, there is a monitor application running in a laptop which is described below. The overall demonstrator architecture is depicted in Figure 3 (the TCB control channel network is omitted, for clarity of presentation).

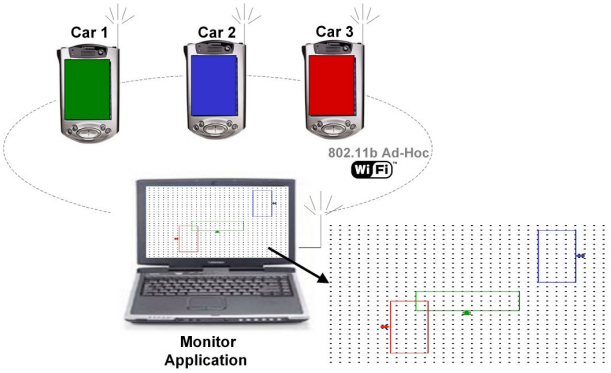

Figure 3. Cooperating Cars demo architecture.

The monitor application interface running on the laptop is divided in three types of panels. A **Reality View panel** shows the actual reality (*the view of God*), allowing to observe the behavior of each car and detect car crashes (laptop screen in Figure 3). For each of the three cars, there exists a **Car Control panel** (Figure 4b) that allows to (centrally) control the speed of the car and the parameters (e.g. level of coverage) of the QoS-Adaptation feature. This panel also shows some car statistics, namely the average delay of received events and the number of lost events.

An interesting aspect of this demo is that when the QoS-Adaptation feature is enabled it is possible to continuously observe the *pdf* generated by the TCB QoS coverage service in the car control panels. This allows to observe the evolution of the environment conditions and to better understand the variations on speed (coverage stability) or on the level of coverage (coverage awareness).

To observe the effects of Fail-Safety/QoS-Adaptation, we can also control on the fly the configuration of the cars, that is, whether they are using or not the fail-safety and

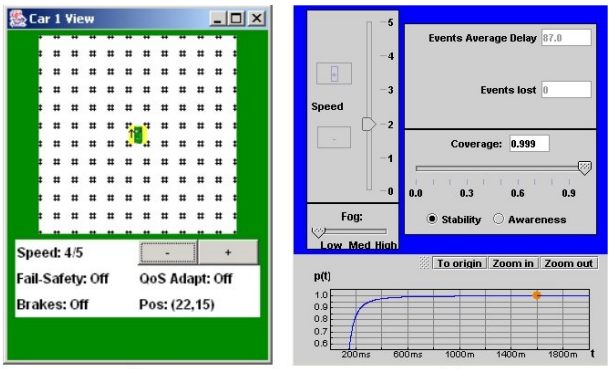

Figure 4. a) Car interface (IPAQ); b) Control panel (monitor).

QoS-adaptation mechanisms. This can be done in the **Master Control panel** of the monitor application.

The IPAQ car interface is presented in Figure 4a. The top part of the screen displays the proximity view of the car— the car is always in the center because this is a relative view. In the bottom part of the screen, it is possible to observe/change the car speed, observe if Fail-Safety and QoS-Adaptation features are ON or OFF, the state of the breaking system (activated when some other car is in the proximity) and check the current position of the car.

The demo presentation is divided in two major parts:

Fail-Safety demo: one extra feature of the monitor application is the possibility of injecting artificial delays in the events received by a car. We call this artificial delays ”fog”. The intensity of the injected fog can be selected among three predefined values (see Figure 4b): low, medium and high. The intensity directly determines the artificial delay to be used. If we inject fog (with a sufficient intensity) while the Fail-Safety feature is turned OFF, car crashes are more likely to happen. The demonstration shows a car crash happening in these conditions. Then we turn ON the Fail-Safety feature and we see that the car in which the fog is being injected stops before any car crash occurs (switching to a fail-safe state in response to a timing failure). The fail-safe state is indicated with a siren that appears in the respective car control panel. A car recovers from a fail-safe state as soon as a timely event is received from all cars from which a late event was received.

QoS-Adaptation demo: even without fog, when the Fail-Safety feature is ON, cars obviously stop if they detect a timing failure. The probability of an event to suffer a timing failure is directly proportional to the sender speed. As mentioned before, a speedy car in a degraded environment could cause all other cars in its proximity (i.e. that have to receive its events) to constantly stop because of timing failures.

To avoid this, we turn ON QoS-Adaptation in coverage stability mode. If we select a sufficiently high level of coverage (through the coverage slider presented in Figure 4b), cars adjust their speed to reflect the environment conditions and in this way the probability of a timing failure to occur decreases.

5. Conclusions

In this paper we have focused on a practical demonstration of the fundamental mechanisms and concepts involved in the construction of dependable real-time adaptive applications in TCB-based systems. Because a TCB is just a particular instantiation of a more general concept, that of wormhole-based systems, this demonstration also serves to illustrate the validity of the wormhole metaphor.

The demonstration shows, in particular, that the correctness of real-time applications can be jeopardized because of the uncertainty of the environment. Then it shows the importance of being able to detect timing failures in a timely manner and of the ability to dependably adapt. The scenario consider applications that exhibit fail-safe and time-elastic properties, which are fundamental properties to handle uncertainty with the help of a TCB.

References

[1] A. Casimiro, P. Martins, and P. Veríssimo. How to Build a Timely Computing Base using Real-Time Linux. In *Proc. of the 2000 IEEE Workshop on Factory Communication Systems*, pages 127–134, Porto, Portugal, Sept. 2000.

[2] A. Casimiro and P. Veríssimo. Using the Timely Computing Base for Dependable QoS Adaptation. In *Proc. of the 20th IEEE Symposium on Reliable Distributed Systems*, pages 208–217, New Orleans, USA, Oct. 2001.

[3] A. Casimiro and P. Veríssimo. Generic timing fault tolerance using a timely computing base. In *Proc. of the 2nd Int. Conference on Dependable Systems and Networks*, Washington DC, USA, June 2002.

[4] M. Correia, P. Veríssimo, and N. F. Neves. The design of a COTS real-time distributed security kernel. In *Fourth European Dependable Computing Conference*, Oct. 2002.

[5] R. Cunningham and V. Cahill. Time bounded medium access control for ad hoc networks. In *Proc. of the Workshop on Principles of Mobile Computing*, Toulouse, France, Oct. 2002.

[6] S. Schönberg. Impact of PCI-Bus load on applications in a PC architecture. In *Proc. of the 24th IEEE International Real-Time Systems Symposium*, pages 430–439, Cancun, Mexico, Dec. 2003.

[7] P. Veríssimo. Traveling through wormholes: Meeting the grand challenge of distributed systems. In *Proc. Int. Workshop on Future Directions in Distributed Computing*, pages 144–151, Bertinoro, Italy, June 2002.

[8] P. Veríssimo. Uncertainty and predictability: Can they be reconciled? In *Future Directions in Distributed Computing*, pages 108–113. Springer-Verlag LNCS 2584, 2003.

[9] P. Veríssimo, V. Cahill, A. Casimiro, K. Cheverst, A. Friday, and J. Kaiser. Cortex: Towards supporting autonomous and cooperating sentient entities. In *Proceedings of European Wireless 2002*, pages 595–601, Florence, Italy, Feb. 2002.

[10] P. Veríssimo and A. Casimiro. The timely computing base model and architecture. *Transactions on Computers - Special Issue on Asynchronous Real-Time Systems*, 51(8), Aug. 2002.

[11] I. . WG. Draft Supplement to STANDARD FOR Telecommunications and Information Exchange Between Systems - LAN/MAN Specific Requirements - Part 11: Wireless Medium Access Control (MAC) and physical layer (PHY) specifications: Medium Access Control (MAC) Enhancements for Quality of Service (QoS), IEEE 802.11e/D3.0, May 2002.