Recent Advances on the Timely Computing Base Model*

António Casimiro Miguel Correia

Faculdade de Ciências da Universidade de Lisboa
Bloco C5, Campo Grande, 1749-016 Lisboa, Portugal
{*casim,mpc*}*@di.fc.ul.pt*

1. Introduction

The development of applications in large-scale real-time systems is known to be a complex task. One of the main difficulties consists in reconciling timeliness expectations with the uncertainty of the environment. The problem has been addressed by several authors, each in its own way [3, 5, 6], but all share the observation that synchronism or asynchronism are not homogeneous properties of systems.

The lack of a generic model able to deal with this partial synchrony problem in a systematic way was one of the reasons that motivated our work around the definition of a new model, which we called the **Timely Computing Base (TCB)** model [7]. It assumes that systems, however asynchronous they may be, and whatever their scale, can rely on services provided by a special module, the TCB, which is timely, that is, synchronous.

Since we devised the Timely Computing Base model, we have taken systematic steps to validate it. In [8] we have shown how to solve a fundamental problem: to interface a payload system of any degree of asynchrony, to a synchronous subsystem as the TCB. We have also discussed the implementation of one of the application classes we propose (fail-safe) on the TCB. In a recent work [2] we show how to implement time-elastic applications on the TCB. Implementing a TCB is a subject of its own. In [1], we describe one of the possible implementations of the TCB, based on RT-Linux, using Linux as the payload support system.

Recently, we have extended the TCB model with a set of security properties to make it trusted not only in the time domain but also in the value domain. The resulting **Trusted Timely Computing Base (TTCB)** is presented in [4]. In the present paper we present a brief report on all these advances.

*This work was partially supported by the EC, through projects IST-1999-11583 (MAFTIA) and IST-2000-26031 (CORTEX), and by the FCT, though the Large-Scale Informatic Systems Laboratoty (LASIGE) and projects Praxis/P/EEI/12160/1998 (MICRA) and Praxis/P/EEI/14187/1998 (DEAR-COTS).

2. Building a TCB

One of the main characteristics of the TCB model is that it assumes the system to be composed of two parts: a *payload* part, where applications execute, and a *control* part, made of local TCB modules. The set of all TCB modules, interconnected by a control channel, constitutes a distributed TCB. While the payload part can have any degree of synchronism, possibly being completely asynchronous, the control part is assumed to by synchronous. Therefore, it is clear that in any implementation of a system with a TCB, a small part of the system must have synchronous properties. The Real-Time Linux (RT-Linux) system is an extension to Linux, designed to allow the execution of real-time tasks in parallel with the normal payload applications. Therefore, RT-Linux was an obvious choice to implement a TCB. For the infrastructure we used standard PCs, a Fast-Ethernet switched network and standard Ethernet cards [1].

We have studied the practical implications of implementing a RT-Linux version of the TCB. We took a pragmatic approach which consisted in investigating whether it would be possible: a) to accurately predict the execution time of TCB activities, which is needed for a schedulability analysis; b) to allow the TCB to handle multiple service requests, arriving at unpredictable instants, and still behave timely and provide timely services; c) to implement a RT-Linux TCB, following the basic construction principles of *Interposition*, *Shielding* and *Validation*.

Our study showed that RT-Linux is clearly not a perfect real-time operating system, at least not for generic PC hardware, but that it is possible to use special safety mechanisms to reduce the problem of occasional timing failures. We then conducted some experiments to obtain practical results for the timeliness of the RT-Linux system and of the Fast-Ethernet switched network. They have shown that RT-Linux is able to schedule real-time tasks with bounded scheduling delays and that Fast-Ethernet switched networks can provide fairly constant message delivery delays.

3. Timely Computing with the TCB

Our approach to the problem of dependable computing with a TCB has always been based on the definition of

generic classes of applications, which would exploit particular abilities of the TCB to enjoy certain "good" properties. We have defined three of these application classes: the *fail-safe*, the *time-elastic* and the *time-safe* class. How practical are these application classes? Can one build real-life applications based on the TCB? Since we first devised the Timely Computing Base model [7], we have methodically addressed these issues. In [8] we have shown how to interface a payload system of any degree of asynchrony, to the TCB. We have also shown how a fail-safe application would look like, when implemented on the TCB. In a recent work [2] we have shown how to implement time-elastic applications on the TCB.

3.1. Supporting Fail-Safe applications

Any class of applications with a fail-safe state can be implemented using a TCB. This is because the TCB has the ability of timely detecting timing failures. In fact, the TCB provides a set of services that may be used by applications to behave timely. We have proposed an interface with a set of basic services, including a *duration measurement*, a *timely execution* and a *timing failure detection* service. Fail-safe applications can use the TCB as follows. When a timing failure occurs it is detected by the TCB in a bounded amount of time, which possibly (timely) executes some fail-safe procedure to bring the system into the fail-safe state. In [8] we show how to use the interface and how these applications look like.

3.2. Supporting Adaptive applications

Another class of applications that may benefit from the TCB services is the time-elastic class. Time-elastic applications are those whose bounds can be increased or decreased dynamically, such as QoS-driven applications. These applications, unlike the fail-safe ones, are typically immune to sporadic timing failures. However, since they cannot handle unbounded failure rates, when a bound is assumed for a given timing variable it is expected to hold with a certain probability. For the application to work correctly it is necessary that the coverage of the assumed bounds keeps stable. We have expressed this fact by introducing the *coverage stability* property. In a recent paper [2] we show that in open and unpredictable environments, where coverage levels tend to vary during the execution, it is possible to construct a *QoS coverage service* using the TCB, to allow time-elastic applications to achieve the coverage stability property.

The basic idea consists in using the TCB duration measurement service to collect timing information during an interval of mission, which is used to build a probability distribution function pdf of a timing variable. Then, with this pdf it is possible to determine $\langle$*bound*,*coverage*$\rangle$ pairs and chose the one that best fits the requirements. In particular, it is possible to know the bound that must be used to keep the coverage constant.

In contrast with other solutions, we propose a rigorous approach to the problem of (timing) QoS monitoring. Several methods and assumptions can be used to obtain a pdf, each with its own associated errors. In [2] we describe the QoS coverage service, its interface, and a concrete method to build pdfs and implement the service.

4. Making the TCB a Trusted Component

The TCB model is well suited to environments with poor baseline timeliness guarantees, such as the internet. It is designed to provide timely services under benign fault assumptions. However, these highly open environments are prone to malicious attacks, which can eventually compromise the robustness of the TCB. We have observed that similarly to synchronism, which is not an homogeneous property of systems, also security can vary in time (e.g. before and after an intrusion) and space (there are components more secure than others). This observation suggests that it may be possible to add simple, but fundamental security services, to make the TCB a timely and highly secure component. We call it the **Trusted Timely Computing Base (TTCB)**. In [4] we describe the TTCB model, services and interface. At this stage, we pay a special attention to the problem of securing the interactions between processes living in the payload part of the system and the TTCB.

References

[1] A. Casimiro, P. Martins, and P. Veríssimo. How to build a timely computing base using real-time linux. In *Proceedings of the 2000 IEEE Intl. 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. Submitted.

[3] T. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, Mar. 1996.

[4] M. Correia, P. Veríssimo, and N. Neves. Supporting the execution of resilient non-byzantine intrusion-tolerant protocols. Submitted.

[5] F. Cristian and C. Fetzer. The timed asynchronous distributed system model. *IEEE Transactions on Parallel and Distributed Systems*, pages 642–657, June 1999.

[6] P. Veríssimo and C. Almeida. Quasi-synchronism: a step away from the traditional fault-tolerant real-time system models. *Bulletin of the TCOS*, 7(4):35–39, Winter 1995.

[7] P. Veríssimo and A. Casimiro. The timely computing base. DI/FCUL TR 99–2, Department of Computer Science, University of Lisboa, Apr. 1999. Short version appeared in the Digest of Fast Abstracts, The 29th IEEE Intl. Symposium on Fault-Tolerant Computing, Madison, USA, June 1999.

[8] P. Veríssimo, A. Casimiro, and C. Fetzer. The timely computing base: Timely actions in the presence of uncertain timeliness. In *Proceedings of the International Conference on Dependable Systems and Networks*, pages 533–542, New York City, USA, June 2000. IEEE Computer Society Press.