Introduction to
Real-time software systems
Draft Edition

Jan van Katwijk
Janusz Zalewski

D R A F T   V E R S I O N
of

November 2, 1998
Chapter 1

Introduction

1.1 General introduction

Information technology is of ever growing importance and it is most likely that it will remain so for the forthcoming decades. The role of computers and computer systems in our society is growing with a dramatal speed, not only in business applications, but also in technical and embedded applications.

In these lectures our interest is the application of computer systems in technical applications. As the price of computer power will continue to decrease, computerization of applications varying from automobiles to households, will increase. Even nowadays, the amount of processors used in embedded application far exceeds the amount of processors used in ‘regular’ computer systems, furthermore, the total effort spent in developing embedded software also exceeds the effort spent in developing business administration and scientific applications.

The kind of systems we are concentrating on in these lectures are real-time (control) systems, i.e. systems that interact with some physical external world and systems that are subject to temporal constraints from that external world. Usually, these systems are embedded, i.e. logically or physically built into the application.

It is certainly worthwhile to realize that the kind of requirements put on such applications is different than the requirement on many other applications:

- installing new versions of software is often complicated. Consider for example an embedded computer system within an automobile. Updating the software with a ‘new’ version (e.g. one with a bug removed) usually involves having a large numbers of automobiles to return to the garage.

- extreme costs of a fault, the occurrence of an error. Apart from the costs involved in repairing the error in the embedded system of the products of a product line, the occurrence of an error itself might inflict severe costs. A computer controlled dishwasher may stop functioning, which is probably annoying and might cause water damage. However, improper operation of the computer system in e.g. a car, a missile, an airplane or a life-support system in an intensive care department might cause the loss of lives. It is b.t.w. beyond any doubt that several accidents with airplanes were caused by improper functioning of software.

- restricted operational environments. Although current development environments grow, both in size and capacity, the growth of execution environments does not necessarily keeps up with this. Consider as an example an arbitrary commodity with an embedded computer system. Depending on the price of the commodity, the costs of the embedded equipment plays a more or less
important role. Being able to minimize these costs may make the difference between gain or loss on the market. Even nowadays, a large fraction of the embedded processors are 4-bit and 8-bit based systems.

- notion of time. Essential for most embedded systems is that they are ‘real-time’, meaning that (i) the internally maintained time must synchronize with time in the external world, and (ii) the system must have guaranteed response times.

1.2 Definition and classification

The notion of real-time is in itself not well-defined. Many systems have to obey performance constraints and are called real-time. Even when we stick to a definition such as a real-time system is a system in which the conceptual notion of time equates the external notion of time, we are not fully satisfied.

In general, we assume that the degree to which the results of computations of a real-time system are valid, is a function of the time the result is available. Depending on the degree of useability of results on moments later than the required time, we call systems hard or soft real-time systems.

In a typical soft real-time system, the result will be of less value when delivered late. Consider e.g. a weather forecast system that needs more than 24 hours computing time for a 24-hour prediction. The results are typically late and therefore of less value. In a typical hard real-time system, the system as a whole will malfunction when reactions are late. As an example, a controller that causes an airbag to inflate to late on the occurrence of a crash simply malfunctions.

The class of problems we are interested in here consists of simple real-time feedback control systems. They can be depicted diagrammatically in fig. 1.1. In this figure, we identify:

- a controlled process, i.e. a plant or a physical process;
- a controller, i.e. a computersystem interfaced to the physical process using sensors and actuators, and somehow connected to the user interface.
- an environment, i.e. a user interface, a network, a storage.

The variables from figure 1.1, separated into respective categories, are described briefly below:

1. The external reference value (or set of values) for the process is set by an external user. It is the task of the controller to ensure that the controlled process behaves in a way satisfying this value or these values.

2. The controller issues commands to the controlled process, e.g. switch on, increase power, in order to achieve the desired behaviour of the controlled process;
(3) The output of the controlled process is observed (3) and fed back as controlled variables into the controller;

(4) Other measured variables; auxiliary signals are received from the controlled process (inputs to the Controller) which are not controlled but used in the determination of the best values of Controller commands

(5) Environment interface: user interface, disk I/O, computer network, etc.

To illustrate that a relatively large class of problems is indeed covered by this simple diagram, we make the following observations:

- if we remove connection (2) from the figure 1.1, we get a the typical structure of a data acquisition system. The system observes the environment, takes values from this environment, most likely does some processing, and communicates its observation to the external world (through connection (3)). It does not control the physical process. Typical examples are weather registration systems or traffic registration systems.

- when feedbacks (3) and (4) are broken, we get the typical structure of a programmed control system. In this case the controller controls the physical process by sending signals, however, no observations are made on the actual state of the physical process. Typical examples are a washing machine controller or a traffic light controller, and

- when all connections (2-4) are present, we have the typical structure of a feedback control system. In this kind of systems the state of the external world is observed, compared to some set point definition and signals are issued to actually influence the physical process. Typical examples are an elevator controller or a flight control system.

Of course, the same structure of having an environment interact with a controlling system can be observed at many places. Most economic systems operate in a similar way, the flow of economics is observed and whenever required the government takes action (e.g. by changing the interest rates).

1.3 Outline of these lecture notes

In these notes we address some topics of real-time software. In chapter 2 we discuss the notion of concurrency. We discuss its appearance as tasks, i.e. in Ada 95, as UNIX processes and as threads. Throughout the chapter, we use a single example, the problem of the dining philosophers.

In chapter 3 we address the notion of time, we discuss clocks and we discuss timers. We measure time and build safe periodical processes.

In chapter 4 we discuss the development of real time software by showing a more or less standard architecture for a wide class of problems. We exemplify the use of the architecture in development by two examples, a simple artificial data acquisition problem and a more complex controller for a pendulum.

In chapter 5 we discuss the notion of scheduling. We show scheduling issues and we discuss a variety of scheduling algorithms and the effects of their use. Furthermore, we discuss deadline monotonic scheduling and its analysis in hard real time systems.

In chapter 6, we discuss some formal techniques for further analysing safety and liveness properties of real time software systems.

Notice that chapter 6 is not yet available in this document