1
INTRODUCTION

Over the last 10 years there has been an explosion of real-time scheduling results. Many of these results are based on either the rate monotonic (RM) or earliest deadline first (EDF) scheduling algorithms. The RM results have been collected and presented in an excellent book [3]. To date, a similar effort has been lacking for EDF. Since EDF has many valuable properties, it is important to address EDF in a comprehensive manner. Consequently, this book presents a compendium of results on earliest deadline first (EDF) scheduling for real-time systems. The simplest results presented utilize pure EDF scheduling. As system situations become more complicated, EDF is used as a key ingredient, but is combined with other solutions, e.g., to deal with shared resources.

This Chapter defines and discusses real-time systems, discusses several common misconceptions, presents an example of a real-time application, and gives a detailed purpose and outline for the book.

1.1 REAL-TIME SYSTEMS

Real-time systems are defined as those systems in which the correctness of the system depends not only on the logical result of the computation, but also on the time at which the results are produced [6]. Many real–time systems are characterized by the fact that severe consequences will result if timing as well as logical correctness properties of the system are not satisfied. Typically, a real–time system consists of a *controlling system* and a *controlled system*. In an automated factory, the controlled system is the factory floor with its robots, assembling stations, and the assembled parts, while the controlling system is the computer and human interfaces that manage and coordinate the activities on the factory floor. Thus, the controlled system can be viewed as the *environment* with which the computer interacts.

The controlling system interacts with its environment based on the information available about the environment from various sensors attached to it. It is imperative that the state of the environment, as perceived by the controlling system, be consistent with the actual state of the environment. Otherwise, the effects of the controlling systems' activities may be disastrous. Hence, periodic monitoring of the environment as well as timely processing of the sensed information is necessary.

Timing correctness requirements in a real–time system arise because of the *physical impact* of the controlling systems' activities upon its environment. For example, if the computer controlling a robot does not command it to stop or turn on time, the robot might collide with another object on the factory floor. Needless to say, such a mishap can result in a major catastrophe.

Real–time systems span many application areas. In addition to automated factories, applications can be found in control of automobile engines, avionics, undersea exploration, process control, robot and vision systems, military command and control, and space stations. In other words, the complexity of real–time systems spans the gamut from very simple control of laboratory experiments, to process control applications, to very complicated projects such as a space station. Recently, the need to process continuous streams of audio and video data has given rise to exciting new possibilities for real-time applications.

Timing constraints for tasks can be arbitrarily complicated, but the most common timing constraints for tasks are either *periodic* or *aperiodic*. An aperiodic task has a deadline by which it must finish or start, or it may have a constraint on both start and finish times. In the case of a periodic task, a period might mean 'once per time interval T' or 'exactly T units apart'.

Low-level application tasks, such as those that process information obtained from sensors or those that activate elements in the environment, typically have stringent timing constraints dictated by the physical characteristics of the environment. A majority of sensory processing is periodic in nature. A radar that tracks flights and produces data at a fixed rate is one example. A temperature monitor of a nuclear reactor core should be read periodically to detect any changes promptly. Some of these periodic tasks may exist from the point of system initialization and remain permanent, while others may come into existence dynamically. The temperature monitor is an instance of a permanent task. An example of a dynamically created task is a (periodic) task that monitors a particular flight; this comes into existence when the aircraft enters an air traffic control region and ceases to exist when the aircraft leaves the region.

More complex types of timing constraints also occur. For example, spray painting a car on a moving conveyor must be started after time t_1 and completed before time t_2. Aperiodic requirements can arise from dynamic events such as an object falling in front of a moving robot, or a human operator pushing a button on a console.

In addition, time related requirements may also be specified in indirect terms. For example, a value may be attached to the completion of each task where the value may increase or decrease with time; or a value may be placed on the *quality* of an answer whereby an inexact but fast answer might be considered more valuable than a slow but accurate answer. In other situations, missing X deadlines might be tolerated, but missing $X + 1$ deadlines can't be tolerated.

This raises the question of what happens when timing constraints are not met. The answer depends, for the most part, on the type of application. Needless to say, a real–time system that controls a nuclear power plant or one that controls a missile, cannot afford to miss timing constraints of the critical tasks. These systems must be predictable [7] in their logical and timing performance. Resources needed for critical tasks in such systems have to be preallocated so that the tasks can execute without delay. In many situations, however, some leeway does exist. For example, on an automated factory floor, if it is estimated that the correct command to a robot cannot be generated on time, it may be appropriate to command the robot to stop (provided it will not cause other moving objects to collide with it and result in a different type of disaster). This is an instance of a real–time task producing a result of lower quality, but on time. In the case of a periodic task monitoring an aircraft, depending on the aircraft's trajectory, missing the processing of one or two radar readings may not cause any problems.

In summary, real-time systems differ from traditional systems in that deadlines or other explicit timing constraints are attached to tasks, the systems are in a position to make compromises, and faults including timing faults may cause catastrophic consequences. This implies that, unlike many systems where there is a separation between correctness and performance, in real–time systems correctness and performance are very tightly interrelated. Thus, real–time systems solve the problem of missing deadlines in ways specific to the requirements of the target application. However, it should be said that the sooner a system determines that a deadline is going to be missed, the more flexibility it will have in dealing with the (timing) *exception*. For more detailed descriptions of real-time systems see [1, 2, 5, 7, 8].

1.2 COMMON MISCONCEPTIONS

Real-time systems have unique sets of requirements usually requiring novel solutions. This fact is not always understood or appreciated. This has given rise to a number of misconceptions including: a sufficiently fast computer can satisfy the requirements, hence real-time computing is equal to fast computing. This is wrong. The point is that speed helps a real-time system in achieving the required responsiveness, but in general does not support predictability [7], which is one main objective of real-time systems.

Another key point in which a real-time system differs from a conventional one is *fairness*. In conventional systems, resource allocation is usually done in a way that avoids starvation of any possible task. Sooner or later, a task that needs a resource gets it. In a real-time system this policy is not adequate. In case of resource contention, more important tasks should precede tasks with lower importance and fairness is not important. If a deadline must be missed, it is better to miss a deadline of a less important task, or to increase its response time, than missing a deadline of an important task.

For a full discussion of these and other misconceptions about real-time computing see [6].

1.3 A TYPICAL EXAMPLE OF A REAL-TIME APPLICATION

One typical real-time application[1] is the flight control program for aircraft, such as the EFA (European Fighter Aircraft) that incorporates a Fly-By-Wire system. This application is presented to provide the reader with a brief overview of the major aspects of a real-time system, and to indicate what aspect of a real-time system that this book addresses. See [5, 8] for additional discussions of real-time systems and applications.

[1]This example is taken from [1]. Also see [4].

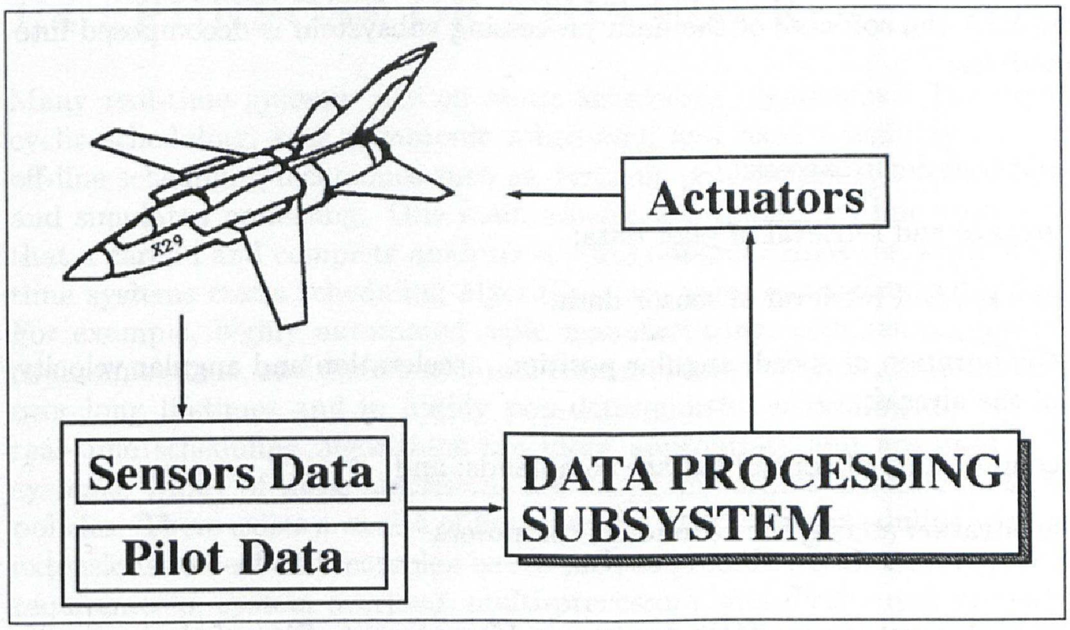

Figure 1.1 A Fly-By-Wire Flight Control System.

In the EFA high performance aircraft the traditional mechanical links with which the pilot usually interacts have been replaced by actuators controlled by a computer system. This is due to the new design approach that is no longer based on the principle of flight mechanical stability, but on a dynamically unstable behavior [4].

The advantage of such design is high maneuverability, but, on the other hand, the aircraft is so unstable that it cannot be flown at all without its computer systems. Instead, an Active Control Technology is needed. In Figure 1.1 the typical main control loop is depicted. This system includes a set of sensors, pilot inputs from control sticks, pedals and consoles, a data processing subsystem, a set of actuators to control the aircraft, and a display for the pilot.

Note that these system components are the typical main ingredients of a real-time system with suitable substitutions depending on the application. For example, a chemical process control plant has sensors measuring acidity, pressure, volume, etc., chemical engineers dynamically adjusting various controls and system settings, a possibly large and distributed data processing subsystem, a set of actuators to control temperature, oxygen levels, etc. that in turn control the chemical reaction, and monitoring displays to allow humans to follow the progress of the production of the plant.

In the EFA the software of the data processing subsystem is decomposed into six modules:

- physical device access;
- storage and retrieval of pilot data;
- storage and retrieval of sensor data;
- computation of speed, angular position, acceleration and angular velocity of the aircraft;
- computation of control surface commands; and
- arbitration among the redundant computers.

In the EFA these six modules execute as 15 processes. Five of them are periodic with deadlines equal to their periods. They are used to probe sensors, whose data are then filtered, averaged and stored. In this application, it is necessary to probe the temperature sensor at least once every 25 milliseconds. All other processes are aperiodic and have deadlines. They are activated by pilot generated interrupts such as when he moves the control stick or pedal, or are activated conditionally on the availability of data.

Almost all the modules have serialized access to their data in order to ensure data consistency. Moreover, the communication module, which provides interprocessor communication (in order to have high reliability the application runs on at least three computers), can lead to communication contention. Note that understandability, predictability, and analyzability are all complicated when there are aperiodic processes contending over device, data and communication sharing. Of course, many real-time applications are much more complicated (e.g., air traffic control and nuclear power plants) and larger (hundreds or thousands of tasks) than this example aircraft application.

While many issues must be addressed to build a real-time system, how timing and resource contention requirements are satisfied is mainly the responsibility of the scheduling algorithm of the system [9, 10]. This book concentrates on the scheduling algorithms and analysis for such real-time systems.

4

1.4 PURPOSE OF THIS BOOK

Many real-time systems rely on static scheduling algorithms. This includes cyclic scheduling, rate monotonic scheduling and fixed schedules created by off-line scheduling techniques such as dynamic programming, heuristic search, and simulated annealing. One main advantage of static off-line scheduling is that a careful and complete analysis is often possible. However, for many real-time systems static scheduling algorithms are quite restrictive and inflexible. For example, highly automated agile manufacturing, command, control and communications, and distributed real-time multimedia applications all operate over long lifetimes and in highly non-deterministic environments. Dynamic real-time scheduling algorithms are more appropriate and are used in such systems. Many of these algorithms are based on earliest deadline first (EDF) policies. There exists a wealth of literature on EDF based scheduling with many extensions to deal with complex issues such as precedence constraints, resource requirements, system overload, multi-processors, and distributed systems. In many cases, formal analysis (as shown in this book) is possible.

This book aims at collecting the significant amount of knowledge that has been developed on EDF scheduling. Rather than just presenting the algorithms, the book also provides proofs, analysis, and sometimes guidelines, rules, and implementation considerations.

Besides learning what these important EDF-based results are, the reader should be able to answer, at least, the following questions:

- what is known about uni-processor EDF scheduling problems,
- what is known about multi-processing EDF scheduling problems,
- what is known about distributed EDF scheduling,
- what anomalous behavior can occur and can it be avoided,
- where is the boundary between polynomial and NP-hard scheduling in EDF problems, and
- what is the influence of overloads on the schedulability of tasks?

It is known that the Rate Monotonic algorithm (RMA) [3] is among the most effective uni-processor real-time scheduling algorithms. This algorithm is one of the best representatives of fixed priority algorithms. To date, a major effort has been devoted to the study of RMA. Less attention has been paid to EDF, even though EDF theoretically allows higher utilization. Another purpose of this book is to expand the comprehension and use of the EDF algorithm.

1.5 FORMAT OF THE BOOK

In general, it is very difficult to codify scheduling knowledge because there are many performance metrics, task characteristics, and system configurations, and a variety of algorithms have been designed for different combinations of these considerations. In spite of the recent advances there are still gaps in the solution space and there is a need to integrate the available solutions. A list of issues includes:

- preemptive versus non-preemptive tasks,
- uni-processors versus multi-processors,
- using EDF at dispatch time versus EDF-based planning,
- precedence constraints among tasks,
- resource constraints,
- periodic versus aperiodic versus sporadic tasks,
- scheduling during overload,
- fault tolerance requirements, and
- providing guarantees and supporting levels of guarantees (meeting quality of service requirements).

Chapter 1 defines real-time systems and gives a brief example of a real-time system. Chapter 2 contains the terminology and assumptions used throughout the book. The fundamental results of EDF scheduling for independent tasks are first presented in Chapters 3 and 4. These Chapters include results on preemption, non-preemption, uni-processors and multi-processors. The overall approach taken in this book is to consider preemption and non-preemption and uni-processors and multi-processors throughout the book rather as separate Chapters. Using EDF in planning mode is discussed in Chapter 5. How to handle system overload is discussed in this Chapter. Chapter 6 discusses results and algorithms relating to general resource requirements. Chapter 7 presents results on scheduling tasks with precedence constraints and resource requirements. Chapter 8 considers problems where periodic and aperiodic tasks must both be scheduled. Scheduling for distributed systems is presented in Chapters 9 and 10. Chapter 11 summarizes the book and discusses open issues.

REFERENCES

[1] W. Halang and A. Stoyenko, *Constructing Predictable Real Time Systems*, Kluwer Academic Publishers, Boston, 1991.

[2] K. Kavi (Ed.), *Real-Time Systems: Abstractions, Languages, and Design Methodologies*, IEEE Computer Society Press, Los Alamitos, 1992.

[3] M. Klein, et. al., *A Practitioner's Handbook for Real-Time Analysis*, Kluwer Academic Publishers, Boston, 1993.

[4] D. Langer, J. Rauch, M. Roaler, "Fly-by-Wire Systems for Military High Performance Aircraft," in *Real-Time Systems Engineering and Applications*, edited by M/ Schiebe and S. Pferrer, Kluwer Academic Publishers, Boston, 1992.

[5] J. Stankovic and K. Ramamritham, *Hard Real-Time Systems*, IEEE Computer Society Press, Los Alamitos, 1988.

[6] J. Stankovic, "Misconceptions About Real-Time Computing," *IEEE Computer* **21**(10), October 1988.

[7] J. Stankovic and K. Ramamritham, "What is Predictability for Real-Time Systems?," *Real-Time Systems* **2**, 1990.

[8] J. Stankovic and K. Ramamritham, *Advances in Real-Time Systems*, IEEE Computer Society Press, Los Alamitos, 1993.

[9] J. Stankovic, M. Spuri, M. Di Natale and G. Buttazzo, "Implications of Classical Scheduling Results for Real-Time Systems," *IEEE Computer*, Vol. 28, No. 6, pp. 16-25, June 1995.

[10] A. Tilborg and G. Koob (Eds.), *Foundations of Real-Time Computing - Scheduling and Resource Management*, Kluwer Academic Publishers, Boston, 1991.