CHAPTER 1

Introduction: The World of Real-Time Systems

1.1 WHAT ARE REAL-TIME SYSTEMS?

Real-time systems are computer systems that monitor, respond to, or control an external environment. This environment is connected to the computer system through sensors, actuators, and other input-output interfaces. It may consist of physical or biological objects of any form and structure. Often, humans are part of the connected external world, but a wide range of other natural and artificial objects, as well as animals, are also possible.

The computer system must meet various timing and other constraints that are imposed on it by the real-time behavior of the external world to which it is interfaced. Hence comes the name *real time*. Another name for many of these systems is *reactive systems*, because their primary purpose is to respond to or react to signals from their environment. A real-time computer system may be a component of a larger system in which it is embedded; reasonably, such a computer component is called an *embedded system*.

Applications and examples of real-time systems are ubiquitous and proliferating, appearing as part of our commercial, government, military, medical, educational, and cultural infrastructures. Included are:

- vehicle systems for automobiles, subways, aircraft, railways, and ships
- traffic control for highways, airspace, railway tracks, and shipping lanes
- process control for power plants, chemical plants, and consumer products such as soft drinks and beer
- medical systems for radiation therapy, patient monitoring, and defibrillation
- military uses such as firing weapons, tracking, and command and control
- manufacturing systems with robots

- telephone, radio, and satellite communications
- computer games
- multimedia systems that provide text, graphic, audio, and video interfaces
- household systems for monitoring and controlling appliances
- building managers that control such entities as heat, lights, doors, and elevators

A simple example of a reactive system is a digital watch that responds to button presses for displaying and setting various times, stopwatches, and alarms. This is also obviously a real-time system because it must respond to clock signals predictably and rapidly enough to display the time accurately. An automobile cruise control system is a good example of an embedded system. It regulates car speed by sensing the drive axle rotations, speedometer, cruise control switch, and throttle to control the brake and throttle; it clearly has timing and other constraints related to performance and safety.

1.1.1 Real-Time versus Conventional Software

Computer hardware for real-time applications typically consists of a number of fairly standard components such as processors, memory units, buses, and peripherals, connected to some real-time input and output devices such as sensors and actuators. Our concern is mainly with the software or programs running on this hardware. Real-time software differs significantly from conventional software in a number of ways.

First is the dominant role of *timing constraints*. A program must not only produce the correct answer or output, but it must also compute the answer "on time." In other words, a program must be both logically and *temporally* correct. More generally, real-time software must also satisfy timing assertions that involve relations over relative and absolute times. The most common and simplest such assertion is a *deadline*—a limit on the absolute or relative time when a computation must complete. In a robot control system, for example, there may be a deadline or limit between the time that a moving robot senses an obstruction in its path and the time that an actuator, such as a wheel controller, is activated to move the robot in a safer direction.

A qualitative distinction can be made between *hard* and *soft* real-time systems. Hard real-time systems are those that, without exception, must meet their timing constaints—if a constraint is violated, the system fails. At the other extreme are soft real-time systems which can still be considered successful, that is, perform their mission, despite missing some constraints. There is a continuum between the extremes of "hardness" and "softness," and most systems fit somewhere in between.

For example, a very hard system might be one that controls the vertical motion of an elevator; missing a particular timing constraint could mean that the elevator stops between two floors, requiring emergency procedures to rescue the occupants. A very soft subsystem might be one that counts the number of vehicles per unit of time entering a highway, and does this when the system has nothing else to do; by definition, it can fail to complete a count. A communication system that occasionally loses messages (but informs the sender) fits in the middle of the hard/soft spectrum. Another example of a softer application is a telephone system that sometimes fails to make a connection; the sender just dials again.

Most timing constraints are *deterministic*, that is, nonstatistical, in nature. Deadlines and other assertions involving time are expressed in terms of exact or fixed values, rather than aggregate measures such as averages. The reason, of course, is that failures to meet deterministic guarantees often mean mission failures, especially for harder real-time systems. For example, a railway crossing gate on a road must always be closed by the time a train reaches the crossing, not closed "most of the time" or on the average. These kinds of deterministic constraints can be contrasted with standard software performance and other timing measures which are usually treated as governed by some stochastic process.

A second distinguishing feature of real-time systems is *concurrency*. Computer systems use concurrency to improve performance, for example, by employing several processors running in parallel. Many systems also use concurrency as a model to represent logically parallel activities, even though they may be implemented by interleaving the activities on a single processor. In addition to these reasons, real-time systems must also deal with the inherent physical concurrency that is part of the external world to which they are connected. Signals from the environment can arrive simultaneously; physically disjoint and parallel activities may be monitored and controlled by a single computer system; it is often necessary to know the real time at which signals are received; and output signals may need to be emitted at approximately the same time due to timing constraints.

Systems design becomes especially difficult when one combines the problems of concurrency with those related to time. In an illuminating article on real-time programming, [Wirth77] defined an informal hierarchy of program complexity: The first or lowest level of software complexity—the least complex—are sequential programs; next in complexity are multiprograms which create an illusion of parallelism by factoring out time (eliminating interrupts at the higher level), implementing logical parallelism typically via coroutines, and providing for synchronization and resource management; the highest and most complex level are real-time programs which include the parallel features of multiprograms but are also execution-time dependent.

A third major characteristic of real-time systems is the emphasis on and significance of *reliability* and *fault tolerance*. Reliability is a measure of how often a system will fail. Alternatively, to paraphrase [Leveson95, p.172], it is the probability that a system will perform correctly over a given period of time. However, virtually no system is perfectly reliable and failures must be expected. Fault tolerance is concerned with the recognition and handling of failures. Errors and failures can be very costly, causing, for example, money losses, property damage, mission failures, or loss of human life. Consequently, it is very important to avoid failures if possible, through techniques for reliability, and to respond appropriately, gracefully, and with as little cost as possible to failures that do occur (fault tolerance).

Systems or parts of systems can be classified according to their *criticality*, which is a measure of the failure cost—the higher the cost of failure, the more critical the system. A very critical system with high failure cost might control an aircraft or a nuclear power plant. Note that criticality is a different dimension than hardness/softness, even though hardness and criticality often go together, as in the last examples. A computer game might still be a hard system, even though it is not a critical one; if a constraint is not met, the game might fail but the failure may be benign to the participant(s).

Most conventional computer systems are general purpose. They run several applications at the same time, and the applications are often unknown to the system designers. Real-time systems, on the other hand, are often *application-specific* and *stand-alone*. That is, all of the software, including the operating system, is tailor-made or adapted for the particular application.

While there are examples of completely automatic real-time systems, the more common configurations and applications have one or more humans who interactively control and monitor the system behavior. The *human–machine interfaces* need to be designed especially carefully in order to prevent human errors and confusion.

A final difference between real-time and conventional programs relates to *testing* and *certification*. Because of the high costs associated with failures, it is usually not feasible to test and debug systems with their actual complete environments. Instead, one must rely upon simulations, testing of subsystems, careful specifications, comprehensive analysis of designs, and extensive run-time procedures for fault detection and handling.

1.1.2 A Comprehensive Example: Air Traffic Control

Air traffic control systems (ATC)[1] provide particularly good examples for a number of reasons. First, they are certainly an important, widely known, and difficult application [Perry97], and there are many of them throughout the world. They also contain instances of all of the distinguishing features of real-time systems presented in the previous section. Finally, parts of them can be abstracted to illustrate various technical ideas.

We consider the ATC network that monitors aircraft flying in the United States airspace. Figure 1.1 shows the basic control points in the U.S. air traffic control environment. The airspace is divided into volumes called *sectors*. If a plane is flying in a controlled airspace, there is an air traffic controller, a person, responsible for that aircraft. As aircraft pass from one sector to another, control of that aircraft passes from one controller to another. The sectors near airports are called *Terminal Radar Approach Control* (TRACON) sectors; here, the air traffic controller directs traffic to the tower for landing, and away from the tower to *En Route* sectors after takeoff.

The main goals of these systems are safety, efficiency, and performance. We want to prevent collisions and the occurrence of other hazards. For this purpose, air traffic controllers need to assure that adequate aircraft separation distances are maintained; and that weather hazards, natural obstacles, and restricted airspaces are avoided. Efficiency and performance goals include maximizing airspace capacity and airport throughput, minimizing aircraft delays, and minimizing fuel consumption.

At each individual center, there may be a variety of computer workstations and servers. There are also many forms of communications among components and between centers. Proposed changes in ATC include the ability for controllers to send flight plans directly from ground ATC to flight management systems on-board an aircraft. Radar will be augmented by aircraft that can report their position—determined through communication with global positioning systems (GPS)—directly to ground control.

[1]We use the abbreviation ATC to mean air traffic control, air traffic control system, and air traffic control systems, relying on the context to distinguish among the three possibilities.

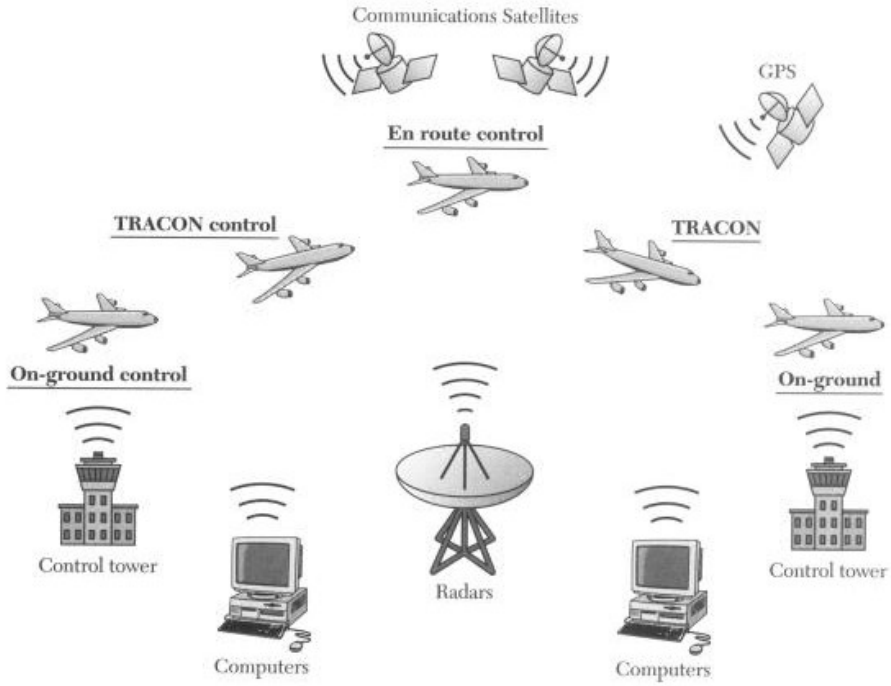

Figure 1.1 Air traffic control systems: Components and control points.

A Simplified ATC

Consider the following simplified version of an ATC [Scallon&Nast87] as a more detailed example. The purpose of the system is to keep track of all aircraft in a given three-dimensional airspace and to ensure that the aircraft maintain a minimum separation distance at all times. Figure 1.2 shows the computer system and its external environment.

A radar tracks the position of each aircraft in the space. When an aircraft enters the space, its presence is announced through a digital communications subsystem; on leaving the space, the computer system broadcasts a notice of its departure to neighboring sites. The communications subsystem can also be used to communicate directly with an aircraft. A console with keyboard and graphics display interfaces to a human operator; it displays the track of each aircraft and responds to operator commands. Operator commands include ones to interrogate the status and descriptive data for an aircraft and to transmit messages to aircraft, for example, to change direction in order to avoid a collision.

The radar can scan any portion of the airspace, given space coordinates. If the radar detects an object at the commanded position, it returns a "hit." An aircraft is considered lost if the radar fails to produce a hit; in this case, the operator must be notified and corrective action taken. The radar is also used to scan the space for unknown objects and lost aircraft.

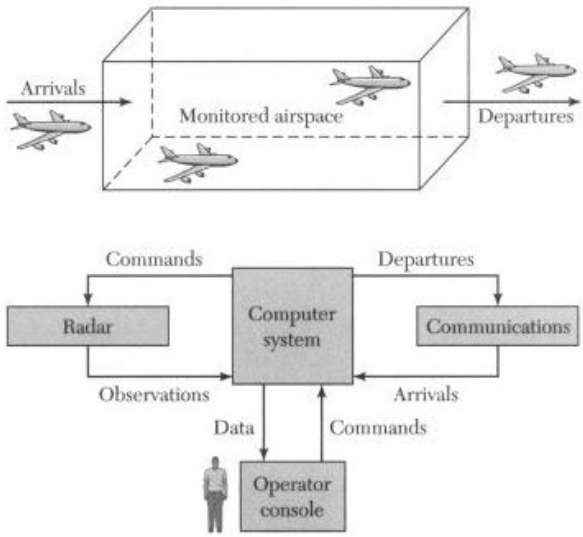

Figure 1.2 Simplified ATC example.

Some reasonable timing constraints for handling the various devices and the environment are:

1. The radar should track each plane in the space at a rate of at least one observation every 10 seconds per plane.
2. The current position and track of each plane should be updated and displayed at least once every three seconds.
3. A response must be provided to an operator command within two seconds.

Where do these constraints come from? The objectives are to give the operators sufficient time to understand the state of the airspace, to control the position of aircraft by sending messages to them, and to respond quickly to unusual or emergency situations, involving, for example, potential collisions or overloads. The informally defined global objectives are translated into the above more detailed and precise constraints.

Note also the inherent parallelism in the environment monitored by the computer. Radar observations, new arrivals, and operator commands can arrive independently and simultaneously. Similarly, the output signals, radar, display, and departures, are logically concurrent.

► EXERCISES 1.1

1. Which of the following computer systems are real-time systems? Justify your answer in terms of the characteristics described in the text.

- An automatic teller machine that dispenses cash to credit card and bank customers.
- An elevator system that responds to patron requests inside and outside an elevator, controlling the elevator and door operations.