6. Real-Time and Safety-Critical Systems

A system is one in which the timing of the output is significant [195]. Such a system accepts inputs from the 'real world' and must respond with outputs in a timely manner (typically within milliseconds – a response time of the same order of magnitude as the time of computation – otherwise, for example, a payroll system could be considered 'real-time' since employees expect to be paid at the end of each month). Many real-time systems are *embedded* systems, where the fact that a computer is involved may not be immediately obvious (e.g., a washing machine). Real-time software often needs to be of high integrity [10].

The term *safety-critical system* has been coined more recently as a result of the increase in concern and awareness about the use of computers in situations where human lives could be at risk if an error occurs. Such systems are normally real-time embedded systems. The use of software in safety-critical systems has increased by around an order of magnitude in the last decade and the trend sees no sign of abating, despite continuing worries about the reliability of software [172, 92].

The software used in computers has become progressively more complex as the size and performance of computers has increased and their price has decreased [216]. Unfortunately software development techniques have not kept pace with the rate of software production and improvements in hardware. Errors in software are renowned and software manufacturers have in general issued their products with outrageous disclaimers that would not be acceptable in any other more established industrial engineering sector. Some have attempted to use a 'safe' subset of languages known to have problematic features [8, 113]. In any case, when developing safety-critical systems, a *safety case* should be made to help ensure the avoidance of dangerous failures [254].

6.1 Real-Time Systems

Real-time systems may be classified into two broad types. *Hard real-time* systems are required to meet explicit timing constraints, such as responding to an input within a certain number of milliseconds. The temporal requirements are an essential part of the required behavior, not just a desirable property. *Soft real-time* systems relax this requirement somewhat in that, while they have to run and respond in

J. P. Bowen et al., *High-Integrity System Specification and Design*

real-time, missing a real-time deadline occasionally only causes degradation of the system, not catastrophic failure.

In the first paper in this Part [200], Jonathan Ostroff considers the specification, design and verification of real-time systems, particularly with regard to *formal methods*, with mathematical foundation. He addresses the application of various formal methods that are suitable for use in the development of such systems, including the difficulties encountered in adopting a formal approach.

As well as the formal correctness of a real-time system [120, 147], there are other issues to consider. Predicting the behavior of a real-time system can be problematic, especially if interrupts are involved. The best approach is to ensure predictability by constructing such systems in a disciplined manner [104]. Often real-time systems incorporate more than one computer and thus all the difficulties of a parallel system must also be considered as well, while still trying to ensure system safety, etc. [238].

Hybrid systems [102] generalize on the concept of real-time systems. In the latter, real-time is a special continuous variable that must be considered by the controlling computer. In a hybrid system, other continuous variables are modeled in a similar manner, introducing the possibility of differential equations, etc. Control engineers may need to interact effectively with software engineers to produce a satisfactory system design. The software engineer involved with such systems will need a much broader education than that of many others in computing.

6.2 Safety-Critical Systems

The distinguishing feature of safety-critical software is its ability to put human lives at risk. Neumann has cataloged a large number of accidents caused by systems controlled by computers, many as a result of software problems [192] and software failures are an issue of continuing debate [114]. One of the most infamous accidents where software was involved is the Therac-25 radiotherapy machine which killed several people [169]. There was no hardware interlock to prevent overdosing of patients and in certain rare circumstances, the software allowed such a situation to occur.

The approaches used in safety-critical system development depends on the level of risk involved, which may be categorized depending on what is acceptable (both politically and financially) [12]. The analysis, perception and management of risk is an important topic in its own right, as one of the issues to be addressed when considering safety-critical systems [221].

The techniques that are suitable for application in the development of safety-critical systems are a subject of much debate. The following extract from the BBC television program *Arena* broadcast in the UK during October 1990 (quoted in [45]) illustrates the publicly demonstrated gap between various parts of the computing industry, in the context of the application of formal methods to safety-critical systems: