Real-time software systems development: a framework for effective learning
G. Halaka, S. MacDonald

Department of Mechanical, Manufacturing and Software Engineering, Napier University, 10 Colinton Road, Edinburgh, EH10 5DT, UK

Abstract

Real-time systems with a substantial software component are being developed for increasingly diverse and complex applications. Explanation of the issues surrounding the development of real-time software is an important part of the curriculum of a software engineering undergraduate course. This paper describes the characteristics of real-time software development and how the Department of Mechanical, Manufacturing and Software Engineering (MMSE) has developed a laboratory to support student learning.

1 Introduction

The Bachelor of Engineering Honours degree in Software Engineering offered by Napier University aims to provide students with both an understanding of the principles of software development and with experience of developing a range of software engineering applications.

Software engineering subjects are heavily dependant on suitably equipped laboratories which enable the students to apply the theory covered by lectures and directed study material. Real-time systems, in particular, have timing and synchronisation constraints which add to their complexity at all stages of the development process. To support the real-time software syllabus a specialist laboratory has been developed by MMSE. The laboratory is equipped with networked computer systems, hosting a variety
of CASE tools and software development environments, which interface to a number of manufacturing applications ranging from control engineering to distributed data acquisition systems. These applications offer challenging target systems for real-time software control. This paper describes the systems available for the development of real-time software, the facilities offered by the laboratory, and shows how these facilities can be used to support the effective teaching of the real-time software development process.

2 Real-time software characteristics

Real-time systems are a category of software intensive applications in which the correctness of the system depends not only on the logical results of computations, but also on the time at which the results are produced. A real-time system must respond to external asynchronous events within a predictable and deterministic time. Response time requirements depend largely on the specific problem domain and can vary from as much as 1 second for a security system, to microseconds for defence applications[1].

Real-time systems can be categorised as hard or soft depending on the criticality of their response. In a hard real-time system the response time at which results must be produced is pre-defined and tasks must be prioritised and scheduled to meet strict deadlines. A hard real-time system is deemed to have failed if a deadline is missed [2]. Applications which fall into this area include process control, avionics, robots, vision systems and command and control. In a soft real-time system the response time is bounded rather than critical. Soft real-time systems include time sharing and client/server applications.

Course subject aims

The aim of the Real-Time Software Systems module is to apply software engineering techniques to the analysis, design and implementation of hard real-time systems. To achieve this aim the syllabus contains three sections: underlying hardware enabling technology, underlying software enabling technology and software engineering methods and techniques.

3 Underlying hardware technology

A real-time system consists essentially of a controlling system and a controlled system (Figure 3.1). The controlled system could represent a machining centre, a robot assembly centre or an automated warehouse system. The controlling system is typically a microprocessor based controller, or micro controller, that detects and affects physical changes in
the environment through strategically located sensors and actuators. The two main configurations for micro controllers are embedded single board controllers[3] and multi-board controllers[4].

![Real-Time System](image1)

Figure 3.1 Real-Time System

### 3.1 Embedded single board controllers

This type of controller is used in applications where space and cost is at a premium, for example, in car engine management systems, laser printers and telecommunication devices. The application software is blown into PROM and the single board is self-contained with respect to the CPU, PROM, RAM, ROM and I/O interfaces (Figure 3.2). Special microprocessors have been tailored to the needs of such applications, for example, the Intel 80186 and 80196[5][6]. This technology can be used for selective student projects, however it is less useful in classroom teaching since testing and debugging requires the use of expensive in-circuit emulators. More importantly, the student is required to have knowledge of the physical configuration of the board as well as the interfacing harnesses.

![Single Board Controller](image2)

Figure 3.2 Single Board Controller

### 3.2 Board based controllers

This type of controller makes use of an industrial bus such as VME, Multibus II, STD, or STE to integrate a number of boards into a customised controlling system (Figure 3.3). An important advantage of this approach is the availability of integrated development environments that allow remote
access to networked target systems, making it ideal for classroom teaching. There are a large number of vendors providing a wide selection of boards that suit many different applications and it is feasible to upgrade to more advanced technology.

![Board Based Controller Diagram]

**Figure 3.3 Board Based Controller**

### 3.3 Subject syllabus

A primary aim of the course is to enable the student to identify and specify an appropriate architecture for a specific application. For example, the student might specify a single processor controller to control an automated warehouse, or a multi-processor controller to control a multi-axis robot. To achieve this aim the following topics are covered:

- microprocessor architecture
- programming models
- memory structure
- I/O modes of operations
- interrupt handling mechanisms
- architecture support for multitasking programming
- passive bus architecture.

### 4 Underlying software enabling technology

Historically, the programming of real-time systems was undertaken with a minimum of analysis and design, and implemented in assembly language. Systems have increased in complexity and the issues of safety, reliability and productivity must be addressed. As a result, appropriate development methodologies, high level programming languages and general purpose real-time operating systems are now used. C has become the defacto procedural programming language in the real-time arena, while object-oriented languages increasingly popular.
Most real-time systems are event driven systems with a tightly controlled response time and the multi-tasking model of programming with pre-emptive priority scheduling makes efficient use of the CPU. Unfortunately, this approach brings with it numerous programming problems such as process synchronisation and inter-process communication. In addition, numerous timing problems will creep in as a direct result of inconsistent task switching time and interrupt latency. These challenging problems clearly create a huge overhead on the programmer directly and affect productivity and reliability. Consequently many vendors provide Real-Time Operating Systems (RTOS)[2][7][8] bundled with an integrated development environment. RTOS's are characterised by a short and persistent task switching time and interrupt latency, which means they are able to respond to internal and external events within a deterministic time frame. An RTOS is structured around two essential parts: a scheduling mechanism and a real-time kernel.

4.1 Real-time scheduling

An RTOS usually employs a pre-emptive priority scheduling algorithm[9][10], for which the task priority is defined by the programmer. The programmer may fine tune the scheduling algorithm in order to guarantee some CPU time for low priority tasks. Some applications, like continuous control and data collection applications, require a high resolution scheduler, while others, like data processing applications may utilise a low resolution scheduler. An RTOS provides the programmer with the capability to adjust the rate at which the scheduler can interrupt task execution to check the status of the task queues.

4.2 Real-time kernel

The main feature of a real-time kernel is that it can be stripped down or augmented in functionality to suit the specific application. A properly designed kernel will occupy anything between 15kbyte to 40kbyte in run time memory and may reside in PROM. This property means that kernels can be bundled with many memory space conscious embedded applications. Moreover, kernels provide the essential set of primitives for handling inter-process synchronisation, inter-process communication, memory management, disk management and networking. Real-time kernels are usually target oriented and only serve a certain range of microprocessors.

4.3 Subject syllabus

The main aim of this part of the syllabus is to enable the students to understand the basic concepts of operating systems within a hard real-time context and to show how they can be integrated into a specific application domain. The syllabus covers the following topics:
5 Software engineering methods and tools

The approach advocated by methods such as SA/SD and object oriented techniques is to generate a technology free model then progressively refine it by superimposing the impact of the underlying technology. In industry, however, the controlled system and its control technology is defined, if not developed, in advance. This approach nullifies the analysis stage and the development process becomes merely a design and implementation stage which integrates the system requirements with the underlying technology.

5.1 Subject syllabus

The system requirements represent the set of operations performed by the system while the underlying hardware architecture represents the controller configuration which may consist of a single processor, multi-processor or distributed processors. The system software architecture represents the services made available by the real-time kernel. To select an appropriate integration methodology, we experimented with SA/SD[12] and OMT[13]. SA/SD provided a good model of the system requirements, but when the underlying technology was integrated into the model the outcome was a complex and confusing view of the system. OMT provided an easier model to understand mainly due to the direct analogy it provided between objects in the model and objects in the environment, and the integration phase was a natural progression. Since the students have already covered OMT in a previous course, we focus on the system design side of the methodology. The topics covered include:

- Breaking the system into subsystems
- Identifying concurrency
- Allocating subsystems to processors and tasks
- Handling global resources
- Handling boundary conditions
- Implementation
6 Applications

The Systems Integration Laboratory has been established to provide a customised development environment for hard real-time systems to satisfy the following aims:
- students should be able to carry out software development with the aid of integrated software development tools
- students should be able to develop software on multiple target systems remotely
- each target system should drive an experiment rig which exhibits a different control strategy
- the operational details, control strategy and safety issues should be described using on-line multimedia packages

6.1 Laboratory structure

The laboratory is structured around a network of 15 PCs and three target micro controllers; one VME bus based and two Multibus II based. Each target controller can be accessed over the network by a peer-to-peer TCP/IP protocol. All machines are connected to a file server which provides a secure repository for student files and controlled access to integrated development tools (see Figure 6.1).

![Figure 6.1 The Laboratory Layout](image)

The configuration of the target controllers has been inflated both to allow for multiple access and to drive multiple experiment rigs. The experiment rigs are as follows:

- a three car lift system
- a closed loop refrigeration systems
- an automated assembly cell
- a fan heat exchanger
The experiment rigs are wall-mounted or desktop based and have extensive safety features. They offer challenging control problems. For example, the three car lift system requires a scheduling algorithm which operates the lift in the most efficient way and process synchronisation between the three cars. The software must monitor pressure sensors and actuate stepping motors to ensure smooth operation. Interrupt driven processes and process prioritisation are an important feature of the controlling software.

The automated assembly cell is a full scale case study which consists of a flexible manufacturing cell, a flexible assembly cell, an automated warehouse and an AGV which shuttles palletised parts between the work cells and the warehouse. The control software must schedule concurrent processes and perform data acquisition and analysis.

### 6.2 Software development tools

The development environment supporting the work undertaken by the students is based on the iRMX for Windows Operating System. This is a real-time kernel featuring multitasking on multiprocessor systems for the Intel x86 family. The kernel supports Multibus II and the PC AT bus and can be installed on a standard PC for development and a Multibus II controller for production. This capability enables the students to develop the software using standard editors and compilers on the PC and transfer the executable image to the target for run-time testing and debugging. iRMX offers library functions for multitasking, scheduling and low level I/O operations and a symbolic debugger. These facilities give the students the opportunity for close examination of their software in relation to equipment which will move in response to their commands.

The PC network hosts a CASE tool supporting the OMT method. The students can capture system requirements and establish the design and test scenarios for their application.

### 7 Conclusions

The laboratory will be used to support the development of hard real-time software on both undergraduate and postgraduate courses. It will also support the development of distributed applications and can be used to demonstrate distributed system paradigms.

The software development environment and a range of support software including CASE tools, operational information and on-line tutorials are available and students are encouraged to undertake unsupervised laboratory work.
The overall aim of the laboratory is to enable students to develop real-time software in an environment which closely reflects the industrial situation. Use of the systems and technologies described above will give the students a deeper understanding of the techniques employed to develop real-time software and gain experience of real-time software development.

8 References

5. Triebel, W. The 80386DX Microprocessor: Hardware, Software and Interfacing, Prentice Hall, 1992