1

Introduction

This book describes how to design the real-time software for embedded systems. This chapter provides an overview of real-time embedded systems and applications and then describes the major characteristics of real-time embedded systems, both centralized and distributed. This chapter also provides an overview of the emerging field of cyber-physical systems, for which real-time software is a critical component. This chapter then introduces COMET/RTE, the real-time software design method for embedded systems described and applied in this book, which uses the Unified Modeling Language (UML), Systems Modeling Language (SysML), and MARTE (Modeling and Analysis of Real-Time Embedded Systems) visual modeling languages and notations.

1.1 THE CHALLENGE

In the twenty-first century, a growing number of commercial, industrial, military, medical, and consumer products are real-time embedded software intensive systems, which are either software controlled or have a crucial software component to them. These systems range from microwave ovens to Blu-ray™ video recorders, from driverless trains to driverless automobiles to aircraft that "fly by wire," from submarines that explore the depths of the oceans to spacecraft that explore the far reaches of space, from process control systems to factory monitoring and control systems, from robot controllers to elevator controllers, from city traffic control to air traffic control, from "smart" sensors to "smart" phones, from "smart" networks to "smart" grids, an ever-growing volume of mobile and pervasive systems – the list is continually growing. These systems are concurrent, real-time, and embedded. Many of them are also distributed. Real-time software is a critical component of these systems.

1.2 REAL-TIME EMBEDDED SYSTEMS AND APPLICATIONS

A *real-time embedded system* is a real-time computer system (hardware and software) that is part of a larger system (called a *real-time system* or *cyber-physical system*) that typically has mechanical and/or electrical parts, such as an airplane or

https://doi.org/10.1017/CBO9781139644532.002 Published online by Cambridge University Press

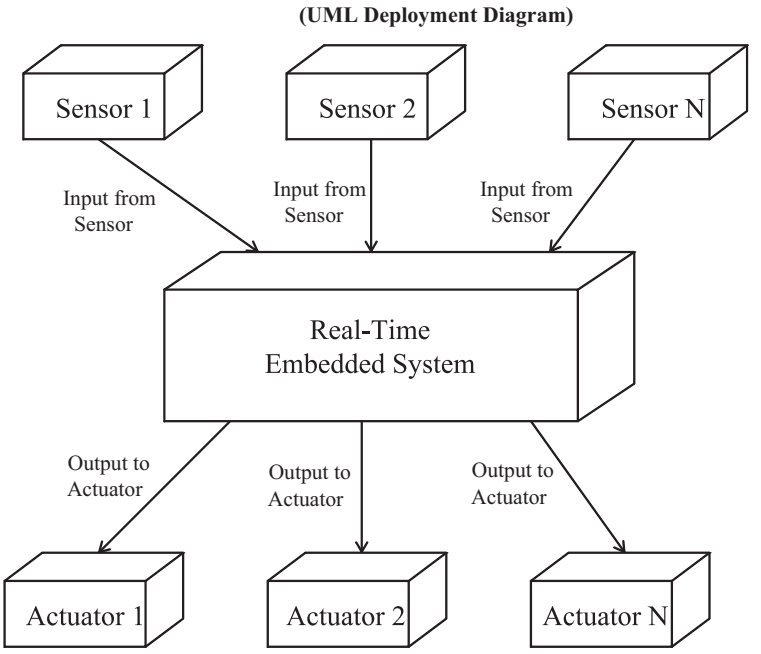


Figure 1.1. Real-time embedded system.

automobile. A real-time embedded system interfaces to the external environment through sensors and actuators, as depicted in Figure 1.1. An example of a real-time embedded system is a robot controller that is a component of a robot system consisting of one or more mechanical arms, servomechanisms controlling axis motion, multiple sensors to provide inputs to the system from external devices, and multiple actuators to control external devices.

Real-time systems are computer systems with timing constraints. The term *real-time system* usually refers to the whole system, including the real-time application, real-time operating system, and the real-time I/O subsystem, with special-purpose device drivers to interface to a variety of sensors and actuators. Although the emphasis in this book is on designing real-time software, in order to develop high-quality real-time software, it is necessary to consider the complete real-time system, since many software quality attributes, such as performance, availability, safety, and scalability, are heavily dependent on the total hardware/software system.

Real-time systems are often complex because they have to deal with multiple independent sequences of input events and produce multiple outputs. Frequently, the order of incoming events is not predictable. In spite of input events having arrival rates and sequences that might vary significantly and unpredictably with time, the real-time system must be capable of responding to these events in a predictable manner within timing constraints specified in the system requirements.

Real-time systems are frequently classified as hard real-time systems or soft real-time systems. A *hard real-time system*, such as a driverless car or train, has time-critical deadlines, such as an emergency stop in front of an obstacle,which must always be met in order to prevent a disastrous system failure. A hard real-time system in which a system failure could be catastrophic is also called a safety-critical system (Kopetz 2011). A *soft real-time system*, such as an interactive Web-based system, is

https://doi.org/10.1017/CBO9781139644532.002 Published online by Cambridge University Press

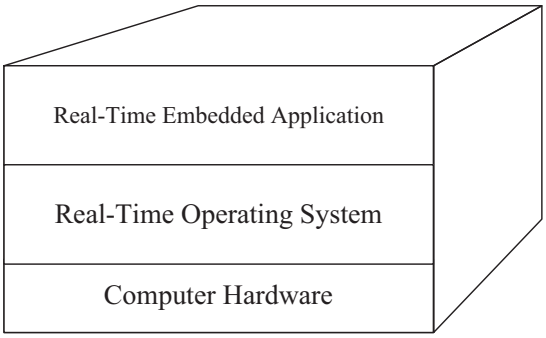


Figure 1.2. Layered architecture of a real-time embedded system.

a real-time system in which missing timing deadlines occasionally, such as response time to a user input, is considered undesirable but not catastrophic.

A real-time embedded system can be designed to have a layered system architecture, as shown in Figure 1.2, consisting of the real-time embedded application, the real-time operating system (with the likely addition of special-purpose device drivers), and the computer hardware.

1.3 CHARACTERISTICS OF REAL-TIME EMBEDDED SYSTEMS

Real-time embedded systems (both centralized and distributed) have several characteristics that distinguish them from other software systems:

a) **Interaction with the external environment**. A real-time embedded system interacts with an external environment that is to a large extent nonhuman. For example, the real-time system might be controlling machines or manufacturing processes, or it might be monitoring chemical processes and reporting alarm conditions.

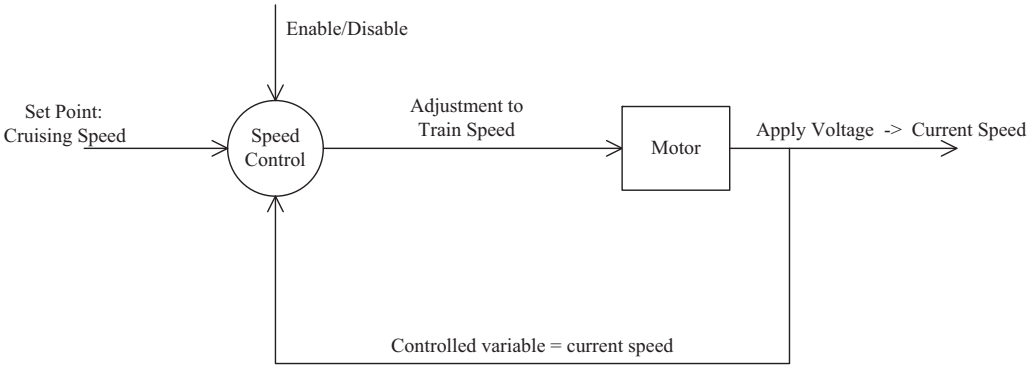
b) **Sensors and actuators**. Interaction with the external environment necessitates sensors for receiving data from the external environment and actuators for outputting data to and controlling the external environment (see Figure 1.1).

A **sensor** is a device that detects events or changes in a physical property (e.g., temperature) or entity (e.g., switch) and converts the measurement (e.g., of temperature) or event (e.g., switch on) into an electrical or optical signal. For example, a thermocouple is a sensor that converts a measurement of temperature into an analog voltage. An analog-to-digital converter then converts the analog voltage into digital inputs to a real-time computer system (Kopetz 2011, Lee and Seshia 2015).

An **actuator** is the means by which a real-time computer system can control an external device or mechanism. Many actuators are devices that convert electrical energy (e.g., in the form of a current) into some kind of motion, for example to open or close a door, or to switch a light on or off.

c) **Measuring time**. A real-time system models the passage of time from the past through the present and into the future. An **event** occurs at an instant of time (conceptually lasting zero time). A **duration** is an interval of time between two

https://doi.org/10.1017/CBO9781139644532.002 Published online by Cambridge University Press



Note: This figure does not conform to the UML notation.

Figure 1.3. Speed control algorithm for automatically controlled train.

events, a starting event and a terminating event. A **period** is a measurement of recurring intervals of the same duration.

There are different units of time in a real-time system. **Execution time** is the CPU time taken to execute a given task on a CPU (or CPUs in a multiprocessor system). **Elapsed time** is the time to execute a task from start to finish, which consists of the task execution time in addition to **blocked time**, which is waiting time when the task is not using the CPU, including waiting for I/O operations to complete, waiting for messages or responses to arrive, waiting to be assigned the CPU, and waiting for entry into critical sections. **Physical time** (or real-world time) is the total time for a real-time command to be completed, for example, to stop a train, which includes the elapsed times of the software tasks involved and then the much longer time required to stop the train physically by applying the brakes and gradually slowing down to a halt.

d) **Timing constraints**. Real-time systems have timing constraints; in particular, they must process events within a given time frame. Whereas in an interactive system, a human might be inconvenienced if the system response is delayed, a delay in a real-time system might be catastrophic. For example, inadequate response in an air traffic control system could result in a midair collision of two aircraft. The required response time will vary by system, ranging from milliseconds in some cases to seconds or even minutes in others.

e) **Real-time control**. A real-time embedded system often involves real-time control. That is, the real-time system makes control decisions based on input data and the current state, without any human intervention. A driverless train has to control the motion of the train automatically, starting from a stationary position, increasing and decreasing speed, cruising at constant speed, slowing down or stopping in the presence of obstacles, and stopping at stations along the route.

In some real-time embedded systems, the control function can be viewed as a process control problem (Kopetz 2011), as shown in Figure 1.3. For example, consider the speed control algorithm in an automatically controlled driverless train. The speed control algorithm has a *set point*, which is the target cruising speed, and a *controlled variable*, which is the current speed of the train.

https://doi.org/10.1017/CBO9781139644532.002 Published online by Cambridge University Press

The speed control algorithm compares the set point with the controlled variable with the goal of increasing or decreasing the current speed of the train as required to make the current speed equal to the cruising speed, plus or minus some small delta value. The positive or negative speed adjustments are converted into electrical voltage and applied to the electric motor, which in turn increases or decreases the speed of the train. A train speed sensor measures the current speed of the train – the controlled variable – and sends the measured speed to the software at regular intervals.

f) **Reactive systems**. Many real-time systems are reactive systems (Harel and Politi 1998). They are event driven and must respond to external stimuli. It is usually the case in reactive systems that the response made by the system to an input stimulus is state dependent; that is, the response depends not only on the stimulus itself but also on what has previously happened in the system, which is captured as the current state of the system.

g) **Concurrency**. Concurrent tasking is an effective solution to the design of real-time embedded systems because it reflects the natural parallelism that exists in the real-time problem domain, in which there are typically many real-world events occurring in parallel. For example, in an air traffic control system, the system is monitoring several aircraft, so many activities are occurring in parallel. Changes in weather conditions can lead to unexpected loads and unpredictable patterns of behavior in the system. A design emphasizing concurrent tasks is clearer and easier to understand because it is a more realistic model of the problem domain than a sequential program. In *multiprocessing systems*, such as *multicore systems*, concurrent tasks can take advantage of multiple CPUs, since any given task can execute in parallel with other tasks executing on other CPUs.

1.4 DISTRIBUTED REAL-TIME EMBEDDED SYSTEMS

Many real-time systems are also distributed. A distributed real-time embedded system executes in an environment consisting of multiple nodes that are in locally or geographically separated locations. In the example given in Figure 1.4, each node consists of a real-time embedded subsystem. Locally separated nodes are connected to each other by means of a *local area network*, while geographically separated nodes are connected to each other by means of a *wide area network*.

A distributed real-time embedded system has the following advantages:

Distributed control. Instead of being centralized, control is distributed among several interconnected nodes in configurations that can be hierarchical or peer-to-peer.

Improved availability. Operation is feasible in a reduced configuration in cases in which some nodes are temporarily unavailable. It is advantageous to design the system such that it has no single point of failure.

Flexible configuration. A given system can be configured in different ways by selecting the appropriate number of nodes for a given instance of the system.

Localized control and management. A distributed subsystem, executing on its own node, can be designed to be autonomous, so it can to a large extent execute independently relative to other subsystems on other nodes.

https://doi.org/10.1017/CBO9781139644532.002 Published online by Cambridge University Press

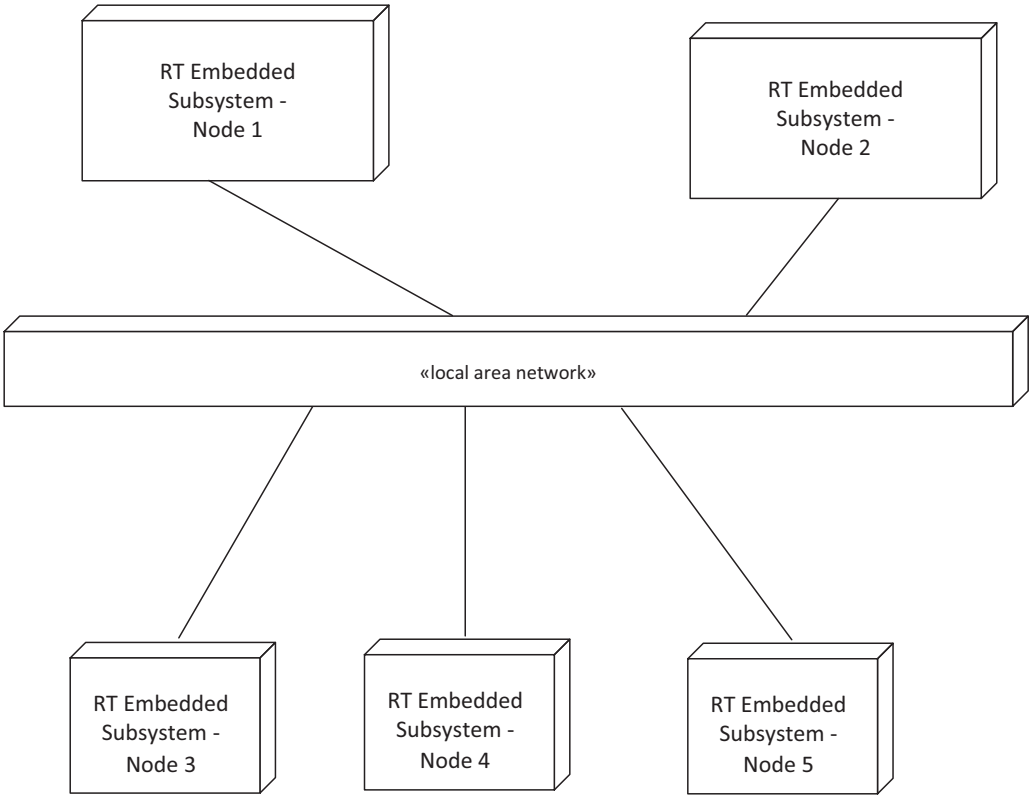


Figure 1.4. Example of distributed real-time embedded system.

Incremental system expansion. If the system gets overloaded, the system can be expanded by adding more nodes.

Load balancing. In some systems, the overall system load can be shared among several nodes and can be dynamically adjusted with varying loads.

Figure 1.5 depicts an example of a layered architecture for a distributed real-time embedded system in which the distributed nodes are interconnected by means of a local area network. Each node consists of several layers, which are the real-time embedded application software, middleware, real-time operating system, and communication software, with the computer and network hardware at the lowest layer. Compared to Figure 1.2, there are additional middleware and communication software layers, as well as additional network hardware in the hardware layer. The communication software allows distributed nodes to communicate with each other using network protocols, such as the Internet Protocol (IP). Middleware is a software layer that lies above the operating system and communication software to provide a uniform platform above which distributed applications can run (Bacon 2003), for example, to provide message communication between applications executing on different nodes. Distributed operating systems often integrate the middleware into the operating system.

https://doi.org/10.1017/CBO9781139644532.002 Published online by Cambridge University Press

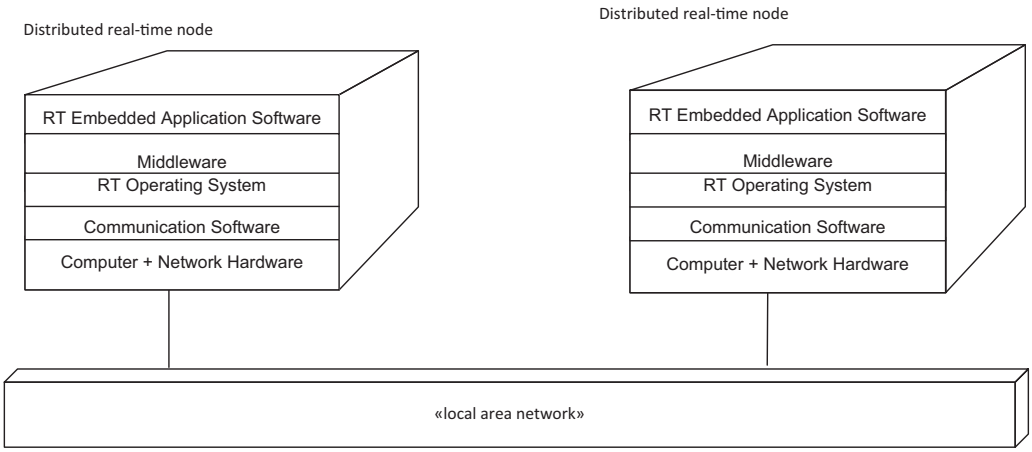


Figure 1.5. Example of layered architecture of a distributed real-time embedded system.

1.4.1 The Internet of Things

The **Internet of Things** (IoT) is a concept of interconnecting physical things to the Internet. This is achieved by connecting remote sensors and actuators to the Internet, with the objective of providing remote access to sensor data and remote control of physical devices over the Internet (Kopetz 2011). RFID is a technology that can be used to enable the connection of physical things (referred to as smart objects) to the Internet. A low-cost electronic RFID tag is attached to a physical product, allowing the product to become a smart object that can be uniquely identified over the Internet. The IoT provides a means for integrating real-time embedded systems with the Internet.

1.5 CYBER-PHYSICAL SYSTEMS

A National Science Foundation vision statement describes **cyber-physical systems** (CPS) as "smart networked systems with embedded sensors, processors and actuators that are designed to sense and interact with the physical world, and support real-time, guaranteed performance in safety-critical applications. In CPS systems, the joint behavior of the 'cyber' and 'physical' elements of the system is critical – computing, control, sensing and networking can be deeply integrated into every component, and the actions of components and systems must be safe and interoperable" (Lee and Seshia 2015).

The design of cyber-physical systems considers the design and integration of both the embedded cyber system and the physical processes. Furthermore, the real-time software design of cyber systems, which monitor and control the physical processes, is critical in the design of cyber-physical systems.

The automated driverless train described in Section 1.3 is an example of both an embedded system and a cyber-physical system. In the design of the train CPS, the design of physical systems such as the electric motor, braking system, speed control system, and transmission, etc. have to be considered in addition to the design of the embedded cyber system consisting of the computer hardware, real-time software,

https://doi.org/10.1017/CBO9781139644532.002 Published online by Cambridge University Press

and network. Computational algorithms need to be designed for controlling physical processes such as the electric motor and the braking system. Designers of these algorithms need to have an intimate knowledge of the design and operation of these physical systems.

1.6 REQUIREMENTS FOR REAL-TIME SOFTWARE DESIGN METHOD FOR EMBEDDED SYSTEMS

A real-time software design method for embedded systems needs to be capable of addressing the following characteristics of a real-time embedded system:

- **Structural modeling** – to model the problem domain, boundary of the total (hardware and software) system, interface between hardware and software components, and the boundary of the software system.
- **Dynamic (behavioral) modeling** – to model the interaction sequences between system and software artifacts at the requirements, analysis, and design levels.
- **State machines** – to react to external events as determined by both the input and the current state of the system.
- **Concurrency** – to handle multiple input sequences and unpredictable loads by modeling activities that execute in parallel with each other.
- **Component-based software architecture** – to provide an architecture consisting of concurrent object-oriented components and connectors, such that components can be deployed to different nodes in a distributed environment.
- **Performance analysis of real-time designs** – to analyze the performance of the real-time system before its implementation to provide an early determination of whether the system will meet its performance goals.

These requirements are all addressed by the COMET/RTE real-time software design method for embedded systems described in this book. How these requirements are addressed by COMET/RTE is described in Chapter 4. An overview of COMET/RTE is given next.

1.7 COMET/RTE: A REAL-TIME SOFTWARE DESIGN METHOD FOR EMBEDDED SYSTEMS

This book describes a software modeling and architectural design method called COMET/RTE (***C***oncurrent ***O***bject ***M***odeling and Architectural Design M***et***hod for **R**eal-**T**ime **E**mbedded Systems), which is tailored to the needs of real-time embedded systems. COMET/RTE is an iterative use case–driven and object-oriented method that addresses the requirements, analysis, and design modeling phases of the system and software development life cycle.

Structural modeling is used to analyze the problem domain from a systems engineering perspective, identifying the static structure of the total hardware/software system and then the boundary between hardware and software. *Requirements modeling* is used to determine the functional and nonfunctional requirements of the system. In *use case modeling*, the functional requirements are described in terms of actors and use cases. In *analysis modeling* for real-time embedded systems, the emphasis is on *dynamic modeling*. The use cases are realized to describe the objects that participate in the use case and their interactions. The state dependent parts of the

https://doi.org/10.1017/CBO9781139644532.002 Published online by Cambridge University Press

system are analyzed using state machines. In *design modeling*, the software architecture is developed, addressing issues of distribution, concurrency, and object orientation. Concurrent components use a blend of object-oriented and concurrency concepts to enable the distribution of components among several nodes in a distributed configuration.

1.8 VISUAL MODELING LANGUAGES: UML, SYSML, AND MARTE

The Unified Modeling Language (UML) is a standardized visual modeling language and notation for describing software requirements and designs. For the UML notation to be applied effectively, however, it needs to be used with an object-oriented analysis and design method. Although UML is sufficient for modeling most software applications, it needs to be supplemented for modeling real-time embedded systems. The Systems Modeling Language (SysML) is used to model the total hardware/software system from a systems engineering perspective. MARTE provides UML extensions for modeling real-time systems.

Modern object-oriented analysis and design methods are model-based and use a combination of use case modeling, static modeling, state machine modeling, and object interaction modeling. Almost all modern object-oriented methods (such as COMET, as described in Gomaa 2011) use the UML notation for describing software requirements, analysis, and design models (Booch et al. 2005; Fowler 2004; Rumbaugh et al. 2005). This book describes how COMET/RTE can be used to design real-time embedded systems using a blend of the UML, SysML, and MARTE modeling languages and notations.

1.9 SUMMARY

This chapter has described the characteristics of real-time embedded systems and applications. It has provided overviews of the COMET/RTE design method for real-time embedded systems and of its use of visual modeling languages and notations. Chapter 2 provides an overview of the UML, SysML, and MARTE modeling language and notations, in particular those parts that are used by COMET/RTE. Chapter 3 describes the fundamental design concepts on which concurrent object-oriented design for real-time embedded systems is based. It describes object-oriented concepts, the concurrent tasking concept including task communication and synchronization, as well as operating system support for concurrent tasks. Chapter 4 provides an overview of the COMET/RTE design method as well as the system and software life cycle for real-time embedded systems. Chapters 5 through 18 describe the details of the method, and Chapters 19 through 23 describe case studies of applying COMET/RTE to design real-time embedded systems.

A comprehensive and wide ranging textbook on real-time systems is Kopetz (2011). Other informative textbooks on real-time systems are Burns and Wellings (2009), Laplante (2011), Lee and Seshia (2015), and Li and Yao (2003).