What Is Real-Time Embedded Software?

by Garth Gullekson
Rational Software

Welcome to the "eManagement" column, where The Rational Edge spotlights management issues and solutions. In this inaugural issue, I would like to define real-time embedded software and describe the major challenges associated with its development. In future columns I will describe in detail these aspects of Rational's approach to real-time embedded software solutions.

Real-Time Embedded Systems Defined

There are two obvious characteristics of real-time embedded systems that we should briefly examine. First of all, they are "real-time" systems, which means they are designed to process information "now, and not later." More specifically, real-time systems must respond to stimuli correctly and in a timely manner. For example, in an airplane the delay between pilot input and changes in control surfaces (e.g., rudders) must occur within certain acceptable time intervals or the plane will not be flyable. This contrasts with many business applications (e.g., payroll), for which time delays can be irritating but are rarely fatal!

Second, we are describing "embedded" systems, which means that their computing power is built into, or embedded in, the system. Embedded processors are usually designed for specific applications rather than general purposes. For example, a telephone system comprises many embedded processors for various functions such as handling terminals, controlling voice and data switches, etc.

Real-Time Embedded Systems in Business

Most real-time systems are embedded, and vice versa, so the industry often uses both terms when referring to this computing domain. Real-time embedded systems span a broad set of application types and sizes. Everything from programmable washing machines to vast distributed telecommunication networks can be classified as real-time embedded systems. Consumer awareness of these "hidden" computers increased substantially during the Y2K crisis, when the volumes of information regarding airline safety, telecommunications integrity, and proper functioning within a host of other industries demonstrated how reliant our
The Internet has accelerated the development of real-time embedded systems, in particular for Internet infrastructure (e.g., communication switches and routers) and Internet-connected devices (e.g., PDAs). The embedded-systems market is rapidly moving to connect almost all embedded devices to the Internet, including those in vehicles, home appliances, and medical devices.

Development Challenges

The growing use of real-time embedded software offers a particularly strong example of what Rational calls "the software development paradox": While companies need to reduce the time they spend on development, at the same time they need to deliver higher product quality. Because real-time embedded software is used in the world’s most critical systems (e.g., telecommunications, avionics, and medical equipment), software quality cannot be compromised. Yet this software is complex and difficult to develop, and embedded software is often difficult to upgrade. Real-time embedded software designs must address many challenges, including timeliness, event-driven stimuli, concurrency, distribution, dynamic structure, and dependability.

Timeliness is the most obvious requirement of the problem domain. Systems have to provide high overall performance and low latency (that is, minimum response time) to individual stimuli. However, timeliness is certainly not the only design challenge. These systems are highly "event-driven." Rather than simply providing functionality from start to finish based on a single command (e.g., printing a document, calculating income tax owed based on financial data), an event-driven system needs to continuously react to various discrete events such as input from users and messages from hardware peripherals. For example, a telephone call progresses from one state to another (e.g., providing dial tone, analyzing what number the user is dialing, ringing, etc.) as events stimulate the system (e.g., input from the user’s terminal). Such input usually comes from various uncoordinated independent sources, and can occur in random order. This makes designing the behavior of software components extremely challenging, because at every state in a component’s operation it must be able to respond properly to many different types of events. For example, in the middle of dialing a phone number, a user may hang up.

Most systems are highly concurrent, supporting multiple use cases (i.e., capabilities) running in parallel. The stimuli driving the systems are themselves often highly concurrent (e.g., multiple users of a system). This can cause design problems because software components are often involved in multiple use cases. Not only must the components be designed to handle each individual use case properly, but also they often must support the concurrent execution of multiple use cases. This includes managing the interactions among concurrent use cases. For example, a component handling a terminal in a telephone system must be prepared to properly handle the use case of taking a terminal out of service while simultaneously executing the use case of the terminal being involved in a call. In such a situation, the software must ensure that the terminal is not
taken out of service until the call is completed.

Many systems, especially for e-development, are also distributed, including multiple processors in the same box or geographical distribution across a network. For example, a photocopier exploits the power of multiple processors to control naturally concurrent activities such as paper feed, image capture, and output control. Distribution increases both the concurrency of stimuli to each embedded system and the complexity of the coordination required among systems. For example, the failure of a particular processor should not cause the other processors in a distributed system to fail.

The structure of real-time embedded systems is often highly dynamic, either based on configuration or dynamically as the system is running (e.g., objects representing telephone calls). Most systems have to be highly dependable, and some are life critical.

**Development Tool Requirements**

In addition to the above problems associated with the real-time embedded software domain, managers need to concern themselves with the following general development tool needs:

- Projects need a completely integrated environment that maximizes development team effectiveness. These environments need to span the complete development lifecycle, including requirements and design, code generation and debugging, configuration management, and ultimately deployment of the applications running on real-time operating systems (RTOSs). Out-of-the-box integration among these key tool elements is critical. Any effort spent by a development group on integration only detracts from the group's effort toward building the application.

- Project success requires a well-understood development process, given the complexity of the software to be developed and the risk of not delivering the product on time.

- Both the development process and tools must be scalable, from small local development teams to the increasingly large and geographically distributed teams so prevalent in today's e-development.

**The Rational Approach**

Rational is extremely well placed to serve the real-time embedded software market. The company's first product in 1981 was designed for real-time embedded development using the Ada programming language, and Rational has continued to serve leading real-time embedded companies such as Ericsson and Motorola. Fueled by the Internet and technology advances, real-time embedded software is becoming increasingly ubiquitous. And it is obvious from the above discussion that real-time embedded development presents numerous unique challenges that can benefit from an optimized solution.
Rational Software has taken an integrated approach to this problem. Our approach includes a graphical user interface for visual design and a development language (the Unified Modeling Language, or UML) that is optimized for the problem domain. As a complement, we also vend an integrated set of tools to meet complete project needs -- from requirements management through development to configuration management. These offerings are anchored by Rational Rose RealTime, cited by IDC as "a major contender as the de facto standard for real-time embedded software development." To accelerate project startup, reduce development risk, and manage complexity, we offer a proven development process and various services.

Until the next issue, if you have any questions, please contact me at garth.gullekson@rational.com.

For more information on the products or services discussed in this article, please click here and follow the instructions provided. Thank you!

Copyright Rational Software 2000 | Privacy/Legal Information