The Challenges of Embedded Systems

prof.dr.dipl-ing. Dieter Hammer

Embedded systems become increasingly important in almost all areas of business and daily life. It should thus not come as a surprise that we think about the introduction of an additional module on this subject in the Software Technology education (OOTI) program of our department. This contribution briefly addresses the following issues: what are embedded systems, what are the challenges in this area and how can we tackle this subject within the OOTI course. The aim of this contribution is not only to introduce the subject, but also to open the discussion about its realization within the OOTI.

What are Embedded Systems?

As suggested by their name, embedded systems are computer systems that are an integral part of a larger system like a production cell, a controller for a home heating system, an audio or television set, a copying machine, an airplane, a digital telephone exchange, etc. The term embedded system thus encompasses a broad class of systems, ranging from simple microcontrollers to large and complex multi-processor and distributed systems. Their embedding in the environment often hides their existence completely from the user; in other cases a dedicated user interface is provided. In general engineering terms embedded systems are used for the control of industrial or physical processes. Sensors continuously gather information from the environment. The service of the embedded system is to process of this information and to signal the actuators in accordance with the mechanisms of the controlled process. Embedded systems thus always have to be designed in the context of their environment. In computer science terms embedded systems are distributed reactive systems.

Of course, *distribution* is more a practical than a principal attribute. It reflects the fact that, for effectivity reasons, more and more businesses take an end-to-end view on their business processes. Think for example about a typical production process for a car consisting of (1) the production of the various parts (this is usually done by external suppliers), (2) the assembly of the parts by the car manufacturer, (3) the shipping of the finished car, and finally (4) the hand-over of the product to the customer. A typical usage process would be the handling of a television set: starting with (1) consulting a program menu (e.g., via teletext or via the catalogue of a video-on-demand source), continuing with (2) checking the various selection possibilities via multiple windows, (3) the selection of a channel, (4) the possible selection of a secondary information source (like a business news service) to be displayed in a small window, and finishing with (5) tuning the display parameters.

Such *processes* often involve a couple of different enterprises. This means that the embedded system is probably large and inhomogeneous, i.e., various different computer systems are involved which are possibly connected via various private and/or public networks. Note that this view is quite different from the more classical one. In the conventional view embedded systems are typically small homogeneous systems that comprise only a few processors that are distributed over a relatively small area, e.g., a control rack or a production cell. However, the introduction of broadband networks with small transmission delays will increasingly blur this distinction. Examples of such networks are ATM networks with gigabyte bandwidth and nanosecond cell switching delays.

There is another important difference between more classical and the currently evolving embedded systems. Conventional embedded systems are often supposed to operate in a deterministic environment. In that case all possible system inputs are known in advance. In principle (i.e., if formal methods are applied), it is then possible to verify the correctness of the system during construction. Of course, this is a very desirable property, especially for safety-critical systems. Consequently many research projects assume a deterministic environment. For example, in the DE-

DOS project that is conducted at our department, the hard real-time part also is deterministic. There is, however, another class of embedded systems that have to operate predictably in a dynamic and unpredictable environment. The larger and more complex the controlled process is, the larger the probability that it has to be adapted dynamically, i.e., during operation. Consequently, the behavior and the configuration of the embedded system have to be changed on-line, i.e., by adding new modules or jobs. An important problem in this area are on-line scheduling methods that guarantee the deadlines of all tasks dynamically, once a new task is accepted by the system. For example, the soft real-time part of the DEDOS project uses dynamic on-line scheduling methods. It should be clear that it is far from trivial to ensure the correctness of the functional, timing, and failure properties of an embedded system dynamically.

Predictability, whether statically or dynamically ensured, not only concerns the functionality but also many other aspects like timeliness (real-time behavior), reliability, availability, security, etc. These non-functional aspects are usually captured by the term *dependability*. Real-time behavior has always been an issue for embedded systems. It is, however, only during the last couple of years that attention is given to the systematic construction of embedded systems that have to provide several of the above mentioned dependability aspects. The reason is quite obvious. As an increasing number of increasingly complex processes depend on the correct operation of embedded systems, there is an increasing demand to avoid failures that lead to loss of money or human life. For the design of embedded systems, and especially of safety-critical systems, it is important to note that the controlled process usually obeys its own dynamics and does not allow the restoration of a previous state in case of failures. This means that erroneous transactions in the environment (e.g., too early or too late firing of the flash of a copying machine) that are caused by failures of the embedded system cannot be undone. Beside being correct, embedded systems must thus obey a well-defined failure behavior like *fail-stop* or *fail-soft* (graceful degradation).

I still need to explain the term *reactivity* in the definition of embedded systems that I gave earlier. There are basically two ways a system can interact with its environment. Transformational systems repeatedly or continuously transform an input into an output. For each input, they start from a given initial state and end via a finite number of transitions in a well-defined final state. Reactive systems, on the other hand, are involved in an unbounded interaction sequence with their environment. This interaction is often based on so-called events that reflect state transitions of the environment. Obviously, many multi-valued environment variables are inferred via the sensors. It is often required that several stimuli have to be handled (1) concurrently, (2) timely, and (3) reliable. The first requirement implies that the behavior of an embedded system cannot simply be described as a sequence of input - output interactions. The second requirement implies that the reactions to a given stimulus have to occur before a given deadline. Finally, the last requirement means that the functionality and timing constraints must also be guaranteed in the presence of failures. A common approach is to consider only failures of the execution platform (hardware and system software) on top of which the embedded software is built. This is certainly sufficient for provably correct software systems like, hopefully, the ones that will be built by the alumni of our OOTI course. Reality, however, is nasty. For safety-critical applications it is thus not unusual to include also certain classes of software faults in the fault hypothesis and to try to mask them, e.g., by means of multi-version programming.

What are the challenges?

The embedded systems field is rapidly growing. A considerable amount of progress has been achieved but a number of challenges still lie ahead. Among the most important ones are the following.

Adaptability versus predictability

As explained above, it is increasingly often the case that embedded systems have to work dependably in a dynamically changing environment. Predictability implies correctness. Correctness implies formal methods since exhaustive testing under mission conditions necessitates the construction of complex testbeds for the injection of faults and takes prohibitive long time. For static and predictable environments, formal methods start to emerge although they usually concentrate on only one dependability aspect and are only suitable for small problems like the specification and proof of a particular algorithm or protocol. For dynamic and non-predictable environments, no generally applicable methods are yet available.

Dependability
The different aspects of dependability are often contradictory. Reliability, for instance, requires the use of additional resources, which usually decreases the performance of the system. Up to now, design methods mainly concentrate on functionality. Dependability, however, cannot be added as an afterthought to a system. As yet, here are no integrated design methods and tool environments that support the systematic construction of both the functional and the non-functional system properties of an embedded system, including its verification against the specification. If we look for suitable formal methods the problem becomes even more severe.

Requirements engineering
As elaborated in the introduction, embedded systems have to support a production or usage process. This asks for a top-down approach that starts from a, possibly reengineered, business process. In the past, however, the designers of embedded systems used to start from a functional break-down of the environment. This is a static and structure-oriented view as opposed to a dynamic and process-oriented one. In addition, the design was was often driven by technological possibilities and choices and did not pay enough attention to the requirements of the process to be supported and to the users. This is a considerable drawback for the validation of a system against the requirements of the business process. In addition, this approach becomes very problematic if the environment changes or the design should be reused with different requirements.

Software productivity The number of applications of embedded systems increases rapidly. The same holds for the software share in the total system which increases with about 80% per year. A modern television set can have more than 1 Megabyte of software and a modern digital telephone exchange more than 100 Megabyte. Also the price/performance ratio of hardware has sharply decreased in the past: about a factor 1.5 per year. At the same time, the product-life cycles are decreased by about 25% per year. On the other hand, the software productivity has increased by not more than 20% per year. The construction of embedded software has thus become a serious bottleneck. Although various approaches have been advocated to tackle this problem, no real breakthrough has been achieved.

A discussion on the reasons behind this development is beyond the scope of this contribution. Nevertheless, the introduction of (1) object-oriented methods and (2) software process improvement models are steps in the right direction. The first subject, the use of object-oriented techniques, certainly improves the quality and maintainability of the product, enables concurrent engineering and is the basis for software reuse. But there are also challenging problems like the specification of dependability properties in the realm of inheritance and reuse. The second topic, software process improvement, addresses software productivity, software quality, and software configuration management by concentrating on the reproducibility, manageability, and efficiency of the construction process.

Some preliminary ideas about an optional Embedded Systems block

In my opinion, the block Embedded Systems will basically have the same structure as the other optional blocks: a small project of about 120 hours, supported by a number of relevant courses. The project should preferably be conducted in relation with a real application of embedded software. Our department is involved in the construction of two facilities that can be used to test software solutions for industrial applications: a scaled-down model of a production facility for printed circuit boards at the department for Industrial Engineering and a controllable railway model at our own premises. It will thus be important to acquire externally funded projects in which these test facilities can be used. In addition, there is the possibility to define interesting industry projects together with other departments like Physics, Electrical Engineering, Mechanical Engineering, and Industrial Engineering.

Fortunately, the block Formal Specification Methods and the block Software Engineering already form a sound basis for the construction of embedded systems. The additional courses will address a number of relevant subjects like design methods for embedded real-time systems, methods and techniques for the construction of distributed systems and dependable systems, distributed algorithms, real-time scheduling, etc. Although it would be nice to engage lecturers from industry who can transfer practical experience, this will not be possible in all cases. Another important consideration will be a good balance between contributions from other disciplines and a sound treatment of the computer science aspects.

Although there is no precise planning yet, I propose to introduce the new block in time for the people that start in September 1995, i.e., during the first half of 1996. This should give enough time to discuss the requirements with both industry and students and to evaluate possible realizations. □

Prof.dr.dipl-ing. Dieter K. Hammer is professor at the Department of Mathematics and Computing Science of Eindhoven University of Technology. He is one of the founders of OOTI.

Between pre- and postcondition...

FADS SELL

I *must admit, embedded software (or should I say embedded hardware) has dramatically simplified the user interface of my alarm clock, requiring only real adjustments when the power goes off. Soon there will be more software on my bedside cabinet than I have written for graduation. I am just waiting for the clock that monitors my sleeping habits and nocturnal peculiarities, notifying me when I snore (this will alleviate the pain caused by having to sleep with a tennis ball between my shoulder blades, hindering me from lying on my back), or preventing me from getting out of bed on the wrong side.*

But what is so irresistibly attractive about clogging a device's ROM with lines of code and rendering a product useless? Although direct manipulation is still advocated and proclaimed as the dominant interaction metaphor, it mostly looks more like negotiating with devices that have programmed minds on their own. Eliciting the right response from a high-end audio set by pushing buttons and shoving levers is rather something for the technologically savvy ones. Dozens and dozens of extraneous features of a ordinary tape recorder have to be repressed, before I actually can do the job for which I purchased the thing (illegally copying those damn high-priced CDs from friends). Basic functionality of any consumer product is hidden under an enormous quantity of add-on features provided by embedded software. Do not startle, when an elder tries to dry his pet in a micro wave. Users will do the stupidest things, because they are provided with the actual reasons for doing them.

A wide range of functionality is looming at the horizon fulfilling the needs we were never aware of even having them. Only very recently, designers have come to the conclusion that user interface design is something more than putting a dressing on a salad. But designers do not work for us. They work for the companies who pay their salaries and think about selling more products. A flashy shop-demonstration of a product with countless knick-knacks does much more to the public's first impression than a simple box with a single switch. Fads sell. □

Edsger & the Bugs Bunnies