Critical (of) issues in real-time systems*

A Position Paper

May 3, 1988

Fred B. Schneider
Department of Computer Science
Cornell University
Ithaca, New York 14853

It is time to place the development of real-time systems on a firm scientific basis. Unlike other engineering disciplines, our methods are not founded on science. Real-time systems are built one way or another because that was the way the "last one" was built. And, since the "last one" worked, we hope that the next one will.

True, the area has benefited from rigorous mathematical work analyzing various scheduling policies for processors and other shared resources. It is argued that managing time is what makes real-time systems different, so such results allow us to put the discipline on a firm scientific basis. While I certainly agree that closed-form analytical equations to allow task-response time predictions under various assumptions are useful, the existence of such results does not seem to move the discipline closer to a science. Why is this the case?

The view that we should add a "time dimension" to all of what computing systems research has deemed "good" is not justified and, as far as I can tell, has never been critically examined. It should be. Abstractions, like processes, fairness, and finite progress, are useful in connection with concurrent programs and operating systems. Because they are abstractions, they suppress irrelevant details and allow us to concentrate on the relevant issues. Relevance, however, is in the eye of the beholder. Processes, fairness, and finite progress are useful in writing a concurrent program exactly because they allow us to ignore details of process interleavings and execution times. In real-time systems, however, these "details" are no longer irrelevant. For example, the importance of execution time is obvious and reasoning about process interleavings is useful in

*This material is based on work supported in part by the Office of Naval Research under contract N00014-86-K-0092 and the National Science Foundation under Grant No. CCR-8701103. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author and do not reflect the views of the Office of Naval Research or National Science Foundation.

avoiding priority inversion. Thus, the usual abstractions might not be the appropriate starting point for real-time programming.

It is time to reconsider our abstractions and devise some that are better suited to our needs. Adding semantic Band-Aids to extant abstractions is probably not the best way to devise suitable abstractions. Completely ignoring existing abstractions is not prudent, either. The new abstractions should build on and generalize the old ones. Thus, when the time dimension is ignored, the new abstractions should degenerate into the old ones, allowing us to exploit the strengths of the old ones without being subject to their weaknesses.

Let me go a step further. Recently, I have begun to question an implicit belief that pervades the field—that time is fundamental to real-time programs. All the systems with which I am familiar can be modeled as a collection of processes. Some of the processes are described by physical laws and can only be controlled indirectly by reading and writing certain variables (sensors and actuators). Other processes are described by program code that has been designed to ensure that the entire computation continues to satisfy some constraints. When controlling a reactor, we are interested in its temperature and pressure; when controlling a train, we are interested in velocity as a function of track position. The control loop for these applications is

```
do true → await bad state;
          perform corrective action
od
```

and contains no explicit mention of time. Time is *one* method for implementing "**await** bad state" when we know something about process execution speeds and the rates of change for physical parameters. But observe that this is little more than using time as a way of implementing synchronization between processes. The notion of "priority" in real-time software can also be seen as a synchronization method for asynchronous processes, although it is rarely described in this manner.

I have hinted at a new paradigm for viewing the area: synchronizing asynchronous processes, not all of which are under program control. Other paradigms should also be investigated, always with an eye towards explaining our current methods and problems in new ways. Whether or not the new paradigms prove useful, the problems with our current paradigm remain. We must, therefore, develop appropriate abstractions for our current paradigm or for alternative paradigms.

Associated with any new paradigm for programming, we would expect to find three elements:

(1) a notation for specifying requirements,

(2) a notation for describing computations, and

(3) a method for verifying that a computation satisfies requirements.

It is important that these elements—specification, description, and verification—all be addressed using an integrated approach. A single of these three elements can make sense and be useful only in concert with the other two. Moreover, experience in sequential programming and concurrent

programming has shown that methods for *a posteriori* verification, as required by (3), can lead to program development calculi. It is the existence of such calculi that will put the field on a firm scientific basis.

Acknowledgments

Andre van Tilborg and Keith Marzullo provided helpful comments on an earlier version of this note.