The use of such high-level instructions is discussed here only as an assembly-level process, and only two levels of abstraction (the components and their assembly into a task list) are allowed. The designer is thus constrained to think in terms of rather simple building blocks, which may not be the appropriate level of abstraction for his problem. Research into more powerful system specification languages, for which the term system compiler would be appropriate, is needed.

Because a system's global structure and scheduling characteristics are separated from the details of the implementation and execution of the components, the prototyping, development, validation, and maintenance of an application can be viewed more simply and more systematically. This offers potential for automatic generation of software from high-level specifications.

Particular virtues to be exploited here are the guarantee that concurrent tasks will not interfere with one another and that different implementations (for example, multiprocessor versus uniprocessor) use the same application programs without change.

The architecture supports application systems requiring backing storage with automatic management of swapping. In fact, resource requirements and their management are overlain onto the application system rather than built into it. Potential extensions in fault tolerance and adaptive hardware and software reconfiguration areas are promising.

Also of significant promise are extensions to include host architectures with multiple heterogeneous general- and special-purpose processors, all orchestrated by a single high-level system program. There are no inherent limitations in the architecture that preclude such an approach.

Finally, adaptations of the architecture to accommodate standard programming languages such as Ada seem promising. One interesting possibility is the automatic translation of the native structures of such a language into procedure-like chunks that can be managed by an executive. This technique could be used to impose more precise management of time-critical computations than the standard language itself provides.

Another possibility is the automatic generation (or selection from a library) of manageable chunks, coded in a language like Ada, to meet a specification expressed in a higher level language. We are exploring this path, both to exploit the standard executive architecture and to gain more understanding of the future role of Ada in real-time applications.

Acknowledgments

The Washington Technology Center, the University of Washington, and Boeing Aerospace supported the writing of this article. The authors are also indebted to the referees for their constructive comments.

References


Theodore P. Baker is a professor in the Computer Science Department at Florida State University. While coauthoring this article, he was on sabbatical leave at the University of Washington, where he studied tool design for real-time software development.

Baker received his BA in 1970 and PhD in 1974 in computer science from Cornell University. His address is Department of Computer Science, Florida State University, Tallahassee, FL 32306-4019.

Gregory M. Scallon is an industrial research fellow with the Washington Technology Center, working on real-time software development methodologies. At Boeing Aerospace, he is applying advanced software research to a Strategic Defense Initiative program.

Previously, he worked for ITT and Texas Instruments. Scallon received a BSEE from Seattle University in 1965.

Scallon's address is Boeing Aerospace, PO Box 3999, Seattle, WA 98124.