ABSTRACTS OF THE IEEE COMPUTER SOCIETY

Fourth Workshop on Real-Time Operating Systems

July 2-3, 1987
Cambridge, Massachusetts

Sponsored by:
IEEE Computer Society

☆TP3
W14

ABSTRACTS OF THE IEEE COMPUTER SOCIETY

Fourth Workshop on Real-Time Operating Systems

July 23, 1987
Cambridge, Massachusetts

Sponsored by:
IEEE Computer Society

Chairmen's Remarks

Welcome to the fourth workshop on real-time operating systems. As the program for the workshop indicates, this workshop has brought together researchers, designers, implementors, and users of real-time operating systems.

This year we are experimenting with a new format for the workshop. All sessions have been organized in the form of panels each devoted to a specific topic and headed by a chair to stimulate and coordinate the technical interactions and discussions among panelists and the audience. To provide sufficient time for discussions, each panelist has been requested to limit his/her presentation to fifteen minutes.

Thanks for your participation in the workshop. We welcome your comments and look forward to your suggestions to make it even more useful to the participants of future workshops as well as to the field of real-time systems.

PROGRAM CHAIRMAN
Prof. Krithi Ramamritham
Department of Computer and Information Science
Graduate Research Center
University of Massachusetts
Amherst MA 01003
(413) 545-0196
e-mail: Krithi@cs.umass.edu

GENERAL CHAIRMAN
Prof. Al Mok
Dept. of Computer Science
TAY 3.140C
University of Texas
Austin TX 78712
(512) 471-9542
e-mail: Mok@sally.utexas.edu

IEEE Fourth Workshop on Real-Time Operating Systems

July 2-3, 1987

Friday, July 3, 1987

Towards Next Generation Distributed Real-Time Operating Systems

Hideyuki Tokuda, Lui Sha and John P. Lehoczky
Computer Science Department and
Department of Statistics
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213
(412) 268-7672
H.Tokuda@k.cs.cmu.edu

1 Introduction

There are many difficult problems in the construction of distributed real-time applications in a network of multiple processors. Unlike a non-real-time program, a real-time program must not only perform the computation correctly, but also meet its timing, fault-tolerance and reliability requirements. A general-purpose time sharing or network operating system can manage resources well when there is no hard deadlines; however, it is impossible for a user to request the operating system to manage resources to meet each task's timing constraints. Nor traditional real-time executives provide adequate supports. They are typically stripped down and optimized timing sharing operating systems. They do not provide adequate system functions for reliability and fault-tolerance in a distributed environment.

As a result, a designer of the distributed real-time program manage resources at the application level. Due to the lack of adequate operating system capabilities, the designer must use an application specific ad hoc solution and it complicates the program logic and structure. The objective of a distributed real-time operating system is to provide efficient and fault-tolerant time-driven resource management for its users in a network environment, so that programmers can concentrate on their applications. In the following, we first review our research efforts related to ARTS operating system. Then, we will overview the research issues which require for further investigated.

2 Review of ARTS Operating System Development

ARTS operating system is a distributed real-time operating system under development in the CMU ART (Advanced Real-Time Technology) Project. Unlike a commercial product, the goal of ARTS OS is to support research and experimentation of computation models, scheduling algorithms and reliability mechanisms that are essential to the construction of distributed real-time operating systems.

2.1 Computational Model

It is a unique problem to create a computational model suitable for designing time critical distributed programs. For instance, a software module concept commonly used in non-real-time systems is the notion of an abstract data type, which encapsulate both the data and its associated operations. However, it has yet to provide appropriate abstractions for the encapsulation of timing properties of the software module.

In ARTS operating system, we are developing an object-based model for a distributed computing environment [Tokuda85a]. An object is defined by a set of operations and private data and can be an "atomic", "permanent", or "normal" object. Each object can have more than one light-weight process which share the same address space within the object. The state of the object can be changed only by invoking its operation. Operation invocation semantics is extended to support not only a synchronous one-to-one invocation, but also asynchronous one-to-many invocations for managing replicated objects. A distinct feature of this ART object model is that it allows the propagation of timing abstractions through the notion of *value function*. For each object, we can define a value function, which represents the timing requirements of object in terms of the (semantic) value of the computation as a function of time. In the case of a server type object, the ARTS OS supports "value-function inheritance" during object invocations. The inherited value function provides the basis for server preemption.

2.2 Integrated Network-Wide Scheduler

ARTS OS is designed to support integrated scheduling algorithms for processor scheduling, I/O scheduling, and message communication scheduling, so that a distributed computation can meet its timing requirement. Since real-time applications can be divided into soft deadline and hard deadline, algorithms for both types of applications have been developed and supported.

As for processor scheduling, ARTS OS provides both a hard real-time virtual processor for periodic and sporadic tasks, and a soft real-time virtual processor for aperiodic tasks. Since bursty aperiodic tasks are incompatible with time division multiplex (TDM) type scheme, the multiplexing of the two virtual processors is based upon the bandwidth preservation algorithms developed in [Lehoczky87].[1]

In the context of the hard real-time virtual processor, the ART OS supports mission critical hard real-time applications:

- Guaranteed deadlines for critical periodic and sporadic tasks whose worst case utilization below the best of the three worst case bounds in [Lehoczky86].
- Stability under transient overload. That is, ARTS OS is designed to support the period transformation algorithm [Sha86], so that the guarantee to the critical tasks are honored even if the total processor load is impossible to schedule due to various exception conditions.
- Integrated processor scheduling and I/O handling. All schedulable entities in ARTS are prioritized and scheduled. In addition, priority inheritance in synchronization mechanisms is supported.

In the context of the soft real-time virtual processor, the ARTS OS supports value function based scheduling which maximizes the value of computation performed by aperiodic tasks. We have developed a time-driven scheduler (TDS) model for a single node environment and analyzed the model by a series of simulation runs [Locke85, 86]. Recently, we have also implemented a modified time-driven scheduler on a VAX 11/784 which consists of four VAX 11/780's connected by 8 Mbytes of shared memory [Tokuda87]. The basic scheduling policy is that when the soft real-time virtual processor is not overloaded, aperiodic tasks are scheduled by the earliest deadline algorithm. When overload is predicted or detected, tasks will be scheduled according to their expected contribution to the total value of the computations performed in the virtual processor.

---

[1] For example, supposed every 20 slots out 100 slots, i.e. (C_a = 20, P_a = 100, U_a = 0.2), are assigned to soft real-time virtual processor, the Deferral algorithm (one of the bandwidth preservation algorithms) permits aperiodic tasks use any 20 slots during a period of 100. This algorithm guarantees that no deadlines of periodic or sporadic tasks will be missed, provided that the hard real-time virtual processor supports tasks with utilization no more than 0.61 ($\ln((U_a+2)/(2U_a+1))$) and that the shortest period of periodic and sporadic tasks is greater than 120 ($P_a(1+U_a)$). The remaining 0.19 (1 - 0.2 - 0.61) of processing cycle can be used for background tasks.

2.3 Fault-Tolerance and Reliability

The basic problem which we are working on is related to system support for atomic transactions and replicated objects. The purpose of using atomic transactions is to share objects in a consistent, reliable and timely manner despite system failure. Based on the formal atomic data set model [Sha85] and a nested transaction model [Moss81], we have extended the transaction model and integrated it into our object model to develop a notion of "compensatable" atomic objects [Tokuda85b].

The basic idea of the compensation is similar to the notion of the forward recovery. If a transaction is aborted, the traditional transaction system performs "roll back" and bring the state of the object back into the previously consistent state. In our model, the aborted transaction will be compensated by executing a corresponding compensation operation and the net-effect of the transaction will be "cancelled".[2] We are also investigating a problem specific "real-time" transaction model together with the problem of integrating the transaction management protocol and real-time scheduling protocol.

The purpose of using replicated objects is to enhance the availability and reliability of the object. There are many ways to manage a replicated object by a user. However, it is not clear how many object attributes which should be related to replication control in the object specification. We are also investigating the minimum set of mechanisms and protocols to support efficient replicated object management at the operating system kernel level.

3 Research Issues

When we attempt to construct a real-time application for a network of homogeneous or heterogeneous processors, we must face many complex problems not only in the system architecture area, but also at every stage of the software life cycle such as problem specification, design, testing, debugging and maintenance. In ARTS OS development, our primary research effort focusses on the functionality and structure of an advanced distributed real-time operating system. In other words, we are studying the necessary advanced operating systems functions and structuring schemes.

3.1 Time-Driven Resource Management

The notion of time-driven resource management must be extended to other types of system resources such as memory, I/O, and communication. For instance, when many tasks are waiting for an access permission to a shared resource, the traditional approach is to give the permission in FIFO order. However, this approach is inconsistant with the real-time scheduling principles. One possible solution is to give the access permission according to the task's value function.

Another important aspect is "policy/mechanism" separation. In TDS scheduler, we have implemented the separation by introducing the policy dispatching module between the mechanism and the policy modules. However, it is important to investigate different types of separation schemes for other system resources.

[2] Whether an atomic object can be compensatable or not depends on the basic characteristic of each atomic operation. However, we also developed a state-insertion scheme so that a certain type of operation can be modified as compensatable operation.

3.2 Real-Time Transactions

A real-time transaction is a transaction which has real-time constraints. Because of the real-time constraints, we need a new type of recovery scheme such as "compensation" instead of "roll back".

A major issue in supporting real-time transaction is the integration issue between time-driven scheduler and transaction manager. As we stated in the previous section, transaction locking protocol, commit protocol and recovery protocol must be integrated together.

3.3 Real-Time Communication Protocols

Real-time communication protocols are very difficult to design, because the design spans from the media access level to the end-to-end level protocols. Furthermore, the design must be consistent with the integrated scheduling policies. In ART project, we are currently analysing various types of the media access level protocols.

Another problem is the implementation of protocol modules in general. Although protocol modules can be created at interrupt handling level, kernel level or process level in an operating system, it is important to develop better guide lines to implement a real-time protocol suite efficiently.

3.4 Real-Time Language Support

In ART project, we are investigating the use of Ada for distributed real-time applications. Even though many modern programming concepts are adapted in Ada, but some language features are inadequate [Dennis87]. For instance, we identified that FIFO nature of the task entry queue in rendezvous and the limitations of the task's priority assignments.

There are many other real-time languages, but many cases the language fails to provide "time" encapsulation mechanism. It also lacks an advanced compiler which can statically analyze the timing requirements among cooperating modules

4 Current Status and Plan

ARTS operating system is being designed and implemented on a network of SUN3 workstations. The current ARTS kernel can provide a multiprogramming environment, but the porting of the ime-driven scheduler from VAX11/784 has not been completed yet. However, every module is written in C, so the machine dependent portion only needs careful porting. The scheduling policy modules we have developed include

- SEPT (Shortest Estimated Processing Time First)
- MAXVAL (Maximum Value First)
- MAXVD (Maximum Value Density First)
- LSLACK (Least Slack Time First)
- DL (Earliest Dead Line First)
- DL2, ..., DL5 (Modified DL Scheduler with various heuristics)
- BE (Best-Effort Scheduling)

The implementation plan of ARTS OS is that we first focus on the development of the integrated time-driven scheduler in a single node. Then, we will coordinate with I/O and communication scheduler, and coordinate each node so that we can perform network-wide scheduling in a transparent manner.

5 Conclusion

The construction of a next generation distributed real-time operating system is essential to the systematic development of large-scale real-time systems. Since the operating system can manage the network-wide resources for its users to meet the timing and reliability constraints, programers can now concentrate on the design and implementation of real-time application.

Many related issues has been studied by various researchers [Mok83, Ramarithan84, Shaw86]. We have investigated some of the important issues in computational model, network-wide scheduling algorithms, and reliability mechanisms. However, many research problems still remain to be solved. In CMU ART project, we are actively studying these issues from both theoretical and systems points of views.

References

[Cornhill87] Cornhill, D., Sha, L., Lehoczky, J.P., Rajkumar, R., and Tokuda, H., "Limitations of Ada for Real-Time Scheduling", To appear in Proc. of ACM SIGAda, International Workshop on Real-Time Ada Issues, May 1987.

[Lehoczky86] Lehoczky, J.P. and Sha, L., "Performance of Real-Time Bus Scheduling Algorithms", ACM Performance Evaluation Review, Special Issue Vol. 14, No. 1 , May 1986.

[Lehoczky87] Lehoczky, J.P., Sha, L. and Strosnider, J.K., "Aperiodic Scheduling in A Hard Real-Time Environment", ART Tech. Report, Computer Science Department, Carnegie Mellon University, Apr. 1987.

[Locke85] Locke, C.D., Jensen, E.D. and Tokuda, H., "A Time-Driven Scheduling Model for Real-Time Operating Systems", Proc. IEEE Real-Time Systems Symposium, San Diego, Dec. 1985.

[Locke86] Locke, C.D., "Best-Effort Decision Making for Real-Time Scheduling," PhD thesis, Computer Science Department, Carnegie Mellon University, 1986.

[Mok83] Mok, A.K. "Fundamental Design Problems of Distributed Systems For The Hard Real Time Environment", PhD thesis, M.I.T., 1983.

[Moss81] Moss, E., "Nested Transactions: An Approach to Reliable Distributed Computing", Technical Report, M.I.T. Laboratory for Computer Science TR 260, Apr. 1981.

[Ramarithan84] Ramamrithan, K. and Stankovic, J.A., "Dynamic Task Scheduling in Hard Real-Time Distributed Systems",. IEEE Computer, Jul. 1984.

[Sha85] Sha, L., "Modular Concurrency Control and Failure Recovery -- Consistency, Correctness and Optimality," PhD thesis, Carnegie Mellon University, 1985.

[Sha86] Sha, L., Lehoczky, J.P. and Rajkumar, R. "Solutions for Some Practical Problems in Prioritized Preemptive Scheduling", Proc. of IEEE Real-Time Systems Symposium, New Orleans, Louisiana, Dec. 1986

[Shaw86] Shaw, A. C., "Software Clocks, Concurrent Programming, and Slice-Based Scheduling", Proc. of IEEE Real-Time Systems Symposium, New Orleans, Louisiana, Dec. 1986

[Tokuda85a] Tokuda, H., Locke, D.C. and Clark, R.K. "Client Interface Specification of ArchOS", Tech. Report, Computer Science Department, Carnegie Mellon Univ., Oct. 1985.

[Tokuda85b] Tokuda, H., "Compensatable Atomic Objects in Object-oriented Operating Systems", Proc. of Pacific Computer Communication Symposium, Seoul, Korea, Oct. 1985.

[Tokuda87] Tokuda, H., Wendorf, J.W., and Wang H.Y., "Implementation of Time-Driven Scheduler for Real-Time Operating Systems", ART Tech. Report, Computer Science Department, Carnegie Mellon University, Apr. 1987.

Topic: A Programmable Standard Executive for Multiprocessor Real-Time Systems

Greg Scallon and Becky Riley, Boeing Aerospace, Seattle, WA 98124-2499
M/S 82-53

Addressing:

- Issues in developing next generation real-time operating systems
- Implications for the design and development of real-time applications software

The increasing reliance on multiprocessor computer systems for medium-to-large scale real-time military applications raises a spectrum of design and performance issues which need to be addressed by next generation real-time operating systems and real-time software development methodologies. It is still current practice in the aerospace industry to build such systems from scratch for each new application. Typical systems incorporate a unique real-time executive, and are costly and time consuming to build and maintain. Furthermore, the complexity of such systems often precludes accurate or comprehensive design or performance trades.

The need to reduce the cost and development time of such systems, and to facilitate early, but accurate, real-time performance predictions, led to the present Boeing research effort. Based on a set of standards derived from experience with the design and development of multiprocessor real-time systems, the research has focussed on:

- a **standard** executive which is general enough to be used for a wide variety of multiprocessor resident and non-resident real-time applications, and which is

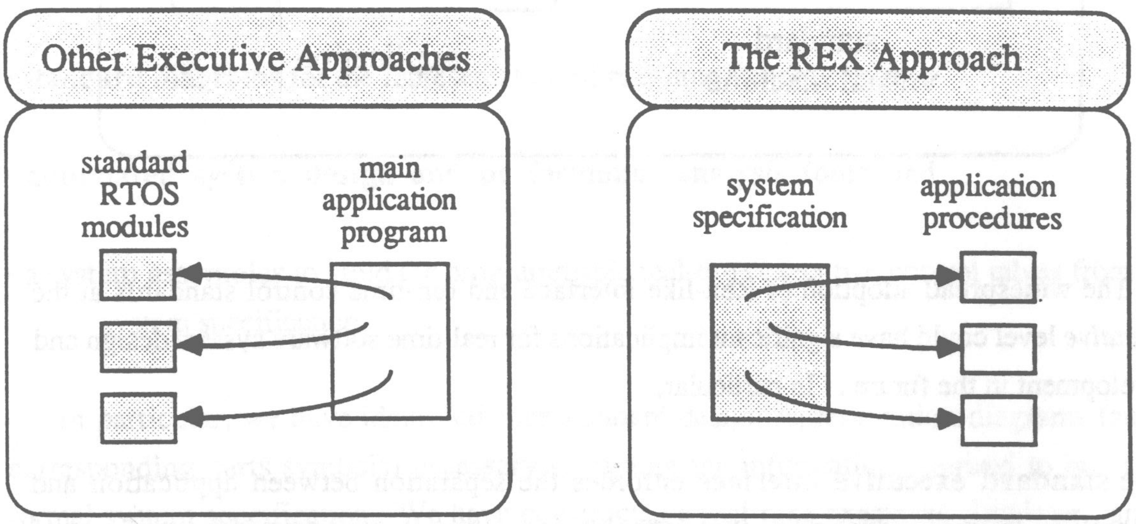
- **programmable** (and thus may be parametrically tailored to meet the particular requirements of a given system).

We are currently developing a prototype implementation of such a standard real-time executive (the Real-time Executive (REX)[1]) targeted to multiprocessor hardware configurations with hierarchical (and shared) memory resources. REX incorporates standards for the specification, organization and run-time management of application software components, and is programmed via a formal system specification language.[2] This language functions as a combined PDL (preliminary design language), MIL (module-interconnect language), JCL (Job Control Language), and system design language. However, it is *not* used to build *functional* application code. Designers use this language to write **system specification** programs which reference application **procedures** written in conventional programming languages. Such programs specify a plan for the organization of (and communications between) procedures and their run-time control on a given hardware configuration. Thus, the application consists of two levels:

• the **architectural level** (the formal system specification), and

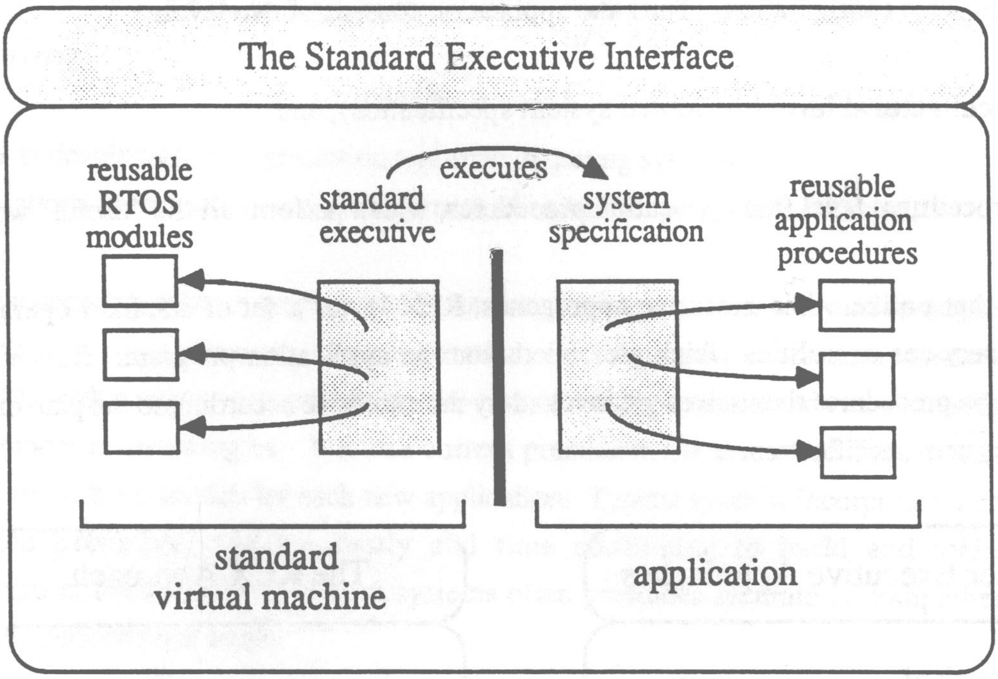
• the **procedural level** (the application procedures, which perform all the "useful" work)

Note that unlike other executive approaches, REX is *not* a set of standard operating system services or utilities which are called from an application program. Rather, the application procedures themselves are invoked by the executive according to the plan of the formal system specification.



This approach forces the designer to break the application into manageable sized procedure modules whose execution and communication with other procedures is then synchronized and controlled across multiprocessor hardware by the executive.

The standard executive performs many typical real-time operating system functions, and in fact, these services are implemented and supported by traditional lower level RTOS routines. The lower level RTOS services are dependent on the hardware configuration and specific to the host machine, but the executive provides a clean, standard interface between these lower level RTOS services and the application. It may be viewed as a **standard virtual machine** on which the application executes, effectively hiding the underlying hardware and operating system details from the designer.



The widespread adoption of Rex-like interface and run-time control standards at the *executive* level could have significant implications for real-time software system design and development in the future . In particular,

• the **standard executive interface** enforces the separation between application and operating system, ensuring that application procedures are independent of hardware/operating system configuration and implementation details, and thus potentially reusable between system designs. This also ensures that the designs themselves are portable between different, but REX-interfaced (multiprocessor) hardware configurations.

• **communication interface standards** force procedures to communicate via explicit, formal parameters, also facilitating their potential reusability. This suggests that the definition of additional standards for the specification of procedure argument *attributes* would make possible the use of automated tools to propagate such attributes through all specified interfaces within a system design to verify compatibilities. Procedures built to the proposed interface standards would then be inherently reusable, "plug-in" parts of a system specification.

• **run-time control standards** allow the operating system execution environment to be modeled with fidelity. Though the designer sacrifices some flexibility in the scheduling and real-time control schemes that may be used, the standard REX run-time control strategy is well-defined and can be accurately modeled. This makes feasible the use of automated tools for the performance testing of system designs. If the target system is interfaced with the same standard executive, then the real-time performance of a given system in that target environment can be accurately predicted. Furthermore, the tested system specification may theoretically be translated directly to product code for the target hardware, since it was developed in the same standard environment.

Following up on some of these possible implications, Boeing is researching and developing some of the companion tools necessary to exploit the automated design, testing and translation which the standard executive approach makes feasible. These include:

• **standard diagrams** for the representation of real-time system designs,

• **automated system design and performance analysis tools**, and

• a **system assembler** to build the programmable real-time executive control tables from a system specification.

In particular, we have identified four standard design representation diagrams (and corresponding parts symbols) necessary to capture the information required to build a formal system specification. We have constructed a real-time executive simulator, and extended the formal system specification language for system modeling. These form the exercisor and input language, respectively, of a computer-based rapid prototyping tool (the Architectural Design and Prototyping Tool (ADAPT)). ADAPT supports automated, static (e.g. parts interface) and dynamic (e.g. real-time performance) testing of system designs. It may be used to incrementally refine or expand the scope of system models. We have also designed (but not yet implemented) a system assembler (the Architectural Design to Operational Product Translator (ADOPT)). It translates a system specification into standard executive control tables, supporting the direct evolution of tested prototypes to executable code for a REX-interfaced target hardware configuration.

Our future plans are to extend ADAPT with fully automated design optimization capabilities, allowing the designer to specify objective functions and success criteria and have the tool identify optimal design solutions meeting those criteria. We also plan to build the ADOPT assembler. But more importantly, we are developing the concept of a real-time system design and development **methodology** based on the standard executive approach.

The above prototype tools (and yet to be constructed tools), made possible by the adoption of interface and run-time control standards at the executive level, can be integrated into an automated methodology to be applied across the real-time system design and development lifecycle. A parallel may be drawn between this approach, and that used for the design and development of hardware. Just as TTL parts and interface standards, MILSTD 806B standard logic symbols, and standard timing diagrams (all arbitrary, but *standard* conventions) made VLSI design automation processes feasible, so similar standards for software parts, interfaces, and run-time control make feasible (and justify the expense of) powerful tools for real-time system design and testing. Then, just as hardware is designed in an efficient top-down manner with "plug-in" standard (and tested) parts, and thoroughly tested prior to construction, so software systems can be built up from standard reusable (and tested) components and their real-time performance analyzed prior to construction. Furthermore, tested prototypes can be translated directly to product code.

We are already using parts of this new methodology to accomplish cost and development time reductions and performance prediction improvements on real-time system design work at Boeing. Ultimately, just as has happened in the hardware development world, we expect that full implementation and use of this methodology will help us realize a compounding of productivity improvements due to the interchangeability of parts and designs, and automated system design and testing prior to construction. We think the implications are revolutionary!