

It's All a Matter of Time

In this article, we enumerate techniques to represent the architecture and timing requirements of a typical distributed process control system so that these techniques can be applied. We use an example mission-critical nuclear plant control system to illustrate the solution. Finally, we outline an extensible framework to describe process control system architectures so that timing analyses can be automated and other techniques can be added as 'plug-in' modules

Process control systems are used in many application domains including power plant control, chemical process plant controls, and discrete manufacturing such as automobile manufacturing, bottling, plant automation etc. Process Control architectures in these systems exhibit two distinct characteristics: (1) processing is distributed and (2) they must satisfy often critical timing requirements. Both these present challenging problems for real-time techniques used to design, develop, document and maintain these systems. The recent trend in process control systems is to distribute the processing functions in the system via smart sensors and devices coupled by more than one interconnecting bus (such as FieldBus, CANBus, Ethernet, FDDI etc.). Another trend is the widespread use of real-time operating systems (such as VxWorks, pSOS, VRTX) in programmable logic controllers. The software designer faces three challenges: (1) many concurrent software tasks, each perhaps with its own timing constraints, must be executed on a single processor (2) end-to-end event-response requirements spanning multiple processors and networks must be satisfied and (3) redundancy must be used to prevent the failure of a single component from bringing down the entire system.

Techniques such as Rate Monotonic Analysis (RMA) are used to analytically guarantee that timing requirements can be met under the worst-case conditions. However, other techniques including simulation are

required to account for the stochastic nature of task execution, and to evaluate average response times. Reliability analysis and simulations can also be used to evaluate failure mode characteristics and analyze 'what-if' scenarios with induced failures. In this article, we enumerate techniques to represent the architecture and timing requirements of a typical distributed process control system so that these techniques can be applied. We use an example mission-critical nuclear plant control system to illustrate the solution. Finally, we outline an extensible framework to describe process control system architectures so that timing analyses can be automated and other techniques can be added as 'plug-in' modules.

A PROCESS CONTROL EXAMPLE

Figure 1 illustrates a high-level view (or functional architecture) of a distributed process control system. A process such as a nuclear reactor is regulated by a set of control functions using operations such as open/close valve, start/stop pump etc. Operators at operator stations can monitor and modify system operations when necessary. The state of the system is also periodically captured for logging and printing. At regular intervals, the system state is also archived to longer-life media such as optical disks. Independently, a protection function continuously tracks the process. Its sole function is to detect any potential problems before they happen and to react by taking preventive action. For example, the protection system in a nuclear plant will look for symptoms of abnormally high temperatures within a reactor core and, if detected, initiate a shutdown of the reactor to prevent a reactor meltdown.

A set of high-level performance requirements must be met in such systems. These requirements are typically specified in terms of response times to real-world (physical) events. A setpoint is a threshold on the value of an input which, when exceeded, results in an action being taken by the control system. In the context of our example system, consider the following requirements placed on a single process input, the reactor temperature. Higher setpoints (e.g., High-3) represent increasing states of alert of the system.

1. Within 200 ms of the temperature exceeding the High-3 setpoint, the shutdown must be initiated.
2. Within 700 ms of the temperature exceeding the High-2 setpoint, the control system needs to be notified and the corresponding safety sequence must be initiated.
3. Within 2 seconds of the temperature exceeding High-1 setpoint, the corresponding alarm must be

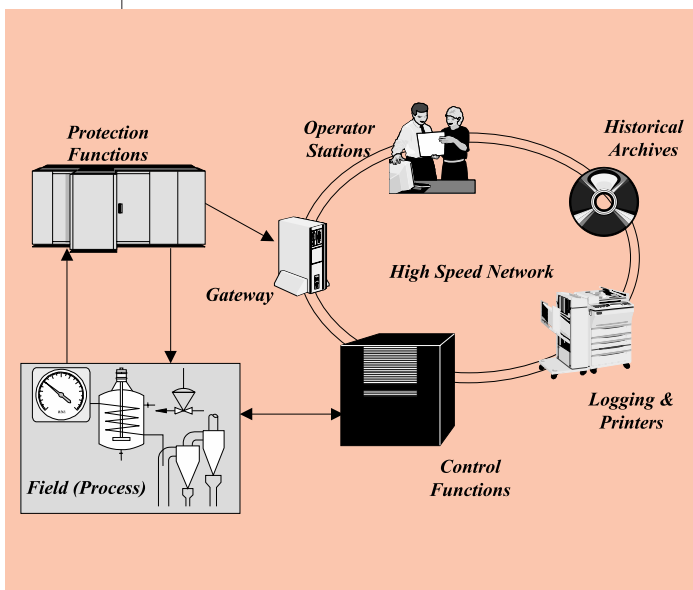


Figure 1: Functional Architecture of a Distributed Process Control System

displayed on the operator station, historically archived and printed.

THE PROBLEM

Several timing constraints such as above must be typically satisfied in distributed control systems. The design, implementation and integration of these systems is therefore a complex process. Maintenance changes can often occur after the system is deployed either due to enhancements to the computer system or due to plant enhancements. These changes can often overwhelm available system resources, resulting in changes that can sometimes affect the system architecture significantly. The design and implementation of control systems is an integration process that constantly attempts to assign required functions to available resources at any given point in time. Complicating the problem is that the response times and tolerance to failures must always be satisfied.

TIMING ANALYSIS

In the design and implementation of the process control system, a signal (or process variable) is processed by several software entities that execute in a distributed environment. Going back to our example, the reactor temperature input must be digitized, converted to engineering values and checked to see if the setpoint is exceeded. If so, appropriate actions must be initiated. These functions may or may not take place on the same processor. Nevertheless, the sum of latencies and processing time for each step must be within the stated requirements. Such delays may be due to

- i. Process scheduling delays in a real-time operating system environment
- ii. Interrupt latencies
- iii. Latencies due to asynchronous periodic tasks
- iv. Signal transmission delays

The timing analyses of these requirements depend on the nature of the 'signal path'. The signal path is defined as the path of the signal flow through the hardware and software elements of the system.

MISSION-CRITICAL PATHS

In high-integrity applications such as a Nuclear Power Plant Protection system, where interrupts and operating systems may not be used, only the last two delays need to be taken into account. In our example, as the temperature signal passes through the protection functions, it encounters the following delays:

1. An asynchronous A/D conversion process that executes on a dedicated processor with a loop time of 5 ms.
2. The output of the conversion is transferred via a backplane for further processing; hence the transmission delays may be ignored.
3. An asynchronous comparator process that executes on a dedicated processor with a loop time of 50 ms.
4. The output of the comparator is transferred via a backplane for further processing; hence the transmission delays may be ignored.
5. The serial message is formatted and processed by a process running at a loop time of 5 ms.
6. The output of the comparator is passed to a voter on a dedicated 64 K serial line within a message of size 100 bytes.
7. The incoming serial message is processed by a serial communication processor running at a loop time of 5 ms.
8. The message is transferred via a backplane for further processing; hence the transmission delays may be ignored.
9. An asynchronous voter process that executes on a dedicated processor with a loop time of 20 ms.
10. The output of the voter is digitally wired to the shutdown switchgear and delays may be ignored.

This is shown pictorially in Figure 2 utilizing hardware architecture with the signal path (in red) overlaid on top. System requirements state that the sum of all the delays must be smaller than 200 ms.

In the case of asynchronous periodic tasks, the worst-case time between message arrival and delivery depends on the implementation of these loops. The two methods of implementation are shown in Figure 3.

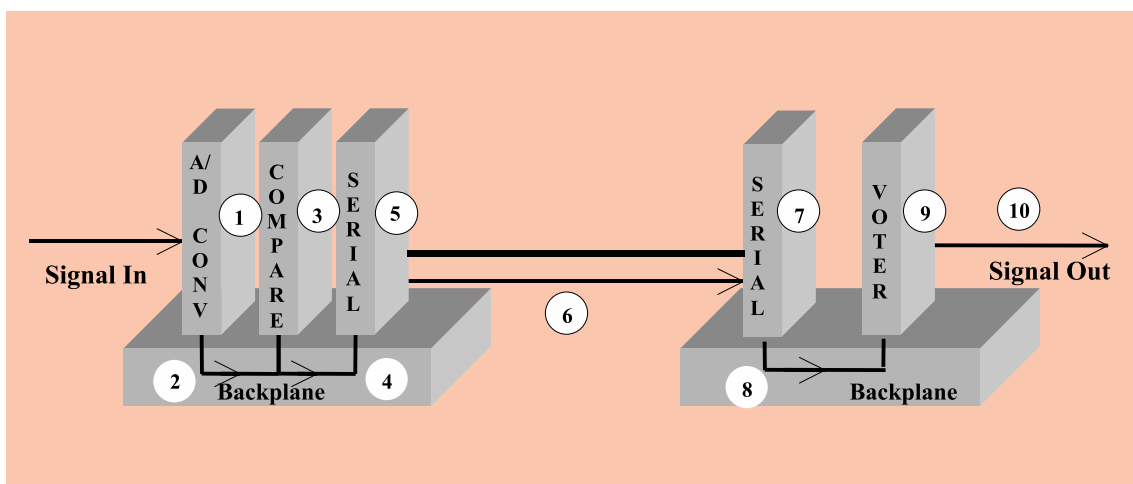


Figure 2: A portion of Hardware Architecture with the signal path overlaid

DISTRIBUTED CONTROL

In the first method shown in Figure 3A, each input or group of inputs is acquired, computations are made based upon the value and outputs are performed. This is then repeated for other inputs or groups. The worst-case scenario consists of the signal arriving just after the signal has been scanned and the resulting latency can be seen to be equal to one complete loop plus time taken for a single input to be acquired, computed and output. In the second method shown in Figure 3B, all inputs are acquired in the beginning of the loop, computations are performed and all outputs are processed at the end of the loop. The worst-case scenario again consists of a signal arriving after the signal has been scanned by the system. However, the latency is equal to two process loops.

From a timing point of view it may appear that the approach in Figure 3A is superior. But it has its drawbacks. The most serious drawback being that since each input is scanned and processed separately, the state of the plant which is represented by a combination of signals may be ambiguous due to data tearing across the signals. Data tearing is said to occur when a collection of several data items is 'torn' (i.e., processed at different times) due to finite transmission delays and process loop times. Even if related signals are grouped and implemented by chunks of code, decisions have to be made on grouping of inputs. As a result, maintenance-related changes can potentially have a direct impact on the timing behavior of many signals.

Assuming that the scenario in Figure 3B applies to our high integrity system, the total time response may be calculated (in msec) as

$$2 * (5 + 50 + 5 + 5 + 20) + (800/64000)*1000 = 170 +$$

$$12.5 = 182.5$$

When considering the worst-case scenario, we assumed that two cycles of each asynchronous periodic process are required to process the input and calculate the output. The average-case response time however can be much smaller since the signal can sometimes show up just before the acquire step. At other times, it can show up in between the acquire step and the output processing step. The worst-case response times therefore tend to be highly conservative; these conservative numbers are useful in the certification of operability of high integrity systems. However, in the majority of real-time situations, an average-case estimate is of operational use and these values are often obtained through simulation.

The above timing requirement definition and analysis deals with only one signal and its processing by the system. In a typical system in which there are hundreds of signals each with their own paths through the system, the representation and analysis is a non-trivial task. Even with the simplification that most signals can be 'grouped' into smaller number of groups with each group consisting of signals exhibiting the same timing characteristics, the resulting system description and analysis can become a time-consuming task, which must be repeated each time a change is made to the system. An automated method for defining and calculating these values will be highly useful. Determining the average-case behavior using simulation also needs to be automated.

High integrity systems also tend to have high degree of redundancy built-in. An example of a redundant architecture is a four way redundancy (2 out of 4 voting to determine safety outputs) and features to support

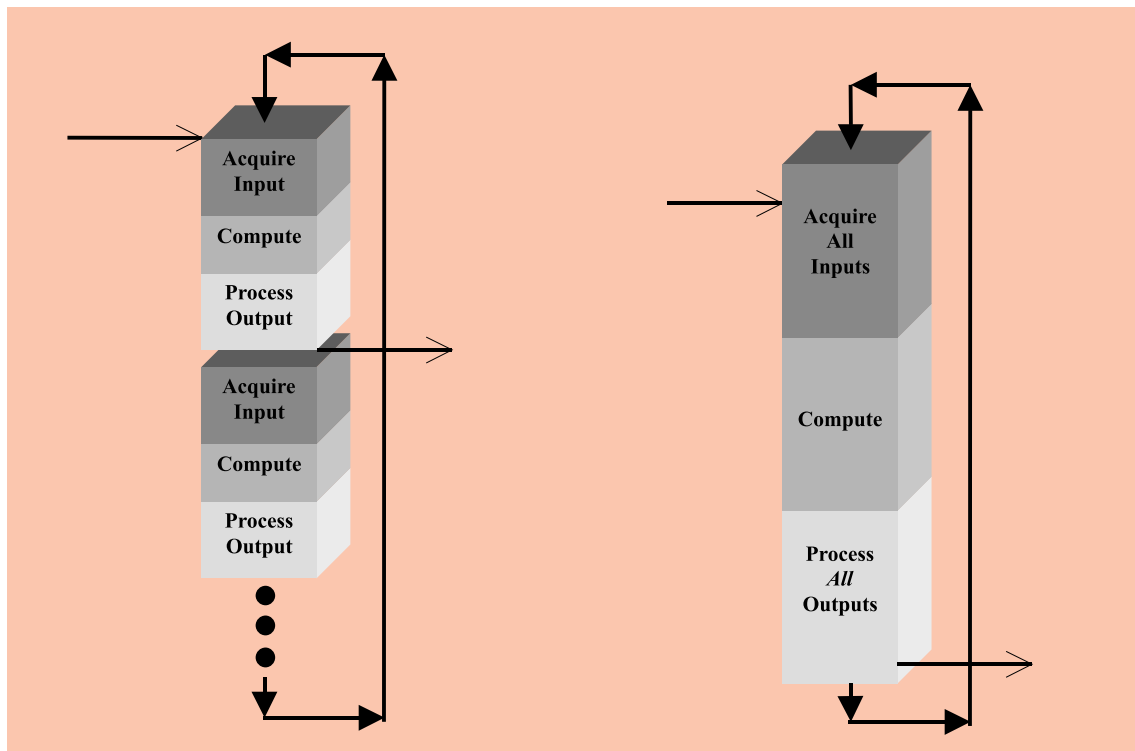


Figure 3A: The worst-case response to the arrival of a signal is an entire loop plus the time for one block.

Figure 3B: The worst-case response to the arrival of a signal is two entire loops.

**Ad
Imagination Syst.**

on-line servicing and operation in a reduced voting mode (such as 1 out of 2) under service conditions. This system-level redundancy must be maintained for all signal paths through the system. Failure mode analysis to determine the failure characteristics of each signal path in the system both using formal analytical techniques and simulation of induced failures is highly to system designers as well as verification and certification agencies.

PATHS THROUGH REAL-TIME OPERATING SYSTEMS

Control systems are typically implemented using programmable logic controllers (PLCs) running small executives or off-the-shelf Real-Time Operating Systems (RTOS) like VxWorks or pSOS. The timing delays when a signal passes through a control system are similar to the delays discussed above. The exception is that delays associated with task scheduling also need to be considered in determining the end-to-end event response time. **Rate Monotonic Analysis (RMA) based techniques can be applied to assess whether each of multiple concurrent timing constraints can be satisfied.** Consider the following signal path for the signal propagation through the control system in our example:

1. An asynchronous A/D conversion process that executes on a dedicated processor with a loop time of 5 ms.
2. The output of the conversion is transferred via a backplane for further processing, delays may be ignored.
3. The converted value is then processed by an Ethernet communications processor with a loop time of 100 ms which sends the value over an Ethernet with a Token Rotation time of 20 ms and a speed of 10 MB/sec within a message of 1000 bytes including all the overheads.
4. Messages are received and processed by a process (period 100 ms, execution 20 ms) running on VxWorks.
5. Control sequences are initiated by another process (period 200 ms, execution 40 ms) running on the same processor under the same operating system.
6. Outputs are performed to directly wired field devices and delays may be ignored.

The characteristics of the PLC need to be defined in terms of (i) the characteristics of the hardware platform, (ii) operating system used and (iii) other real-time processes on the PLC, in order to perform meaningful timing analysis.

RATE-MONOTONIC ANALYSIS

Rate Monotonic Analysis (RMA) is a set of mathematical techniques for performing timing analysis of complex real-time systems. It is described in the Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems. In its simplest form, it states that

A set of n independent periodic tasks scheduled by

the rate monotonic algorithm will always meet its deadlines, for all task phasings, if

$$S(C_i/T_i) \leq U(n) = n(2^{1/n} - 1)$$

where,

C_i = worst-case task execution time of task i

T_i = period of task i

$U(n)$ = utilization bound for n tasks

The above equation applies to the simple case of independent periodic tasks. Techniques are also presented to represent, model and analyze many more complex real-time situations including shared resources, jitter requirements, aperiodic tasks etc.

In order to apply the above techniques to our system, the characteristics of the operating system and associated scheduling, preemption and priority inheritance policies of the operating system and the timing characteristics of the other tasks executing in the machine must be known. Once the schedulability constraint is determined to be satisfied, the above described signal path timing analysis can be applied using the period of the processes in the signal path.

In our example, assuming no more processes exist, the schedulability is satisfied since

$$(20/100) + (40/200) = 0.4 \text{ is less than } U(2) = 2(2^{1/2} - 1) = 0.828$$

The total response time is then calculated as

$$2 * (5 + 100) + (20 + 8 * 1000 / 10000) + (100 + 20) + (200 + 40) = 591$$

Note that we have used the sum of one execution time and one period for the RTOS based process's response times. This is because for multi-process systems, the 'dead-time' (inactive time) for a process tends to be substantial compared to single processor, single process systems discussed in the previous section.

More often, the task execution time cannot be determined precisely, although the period needs to be specified as a parameter to the operating system. In these situations, worst-case schedulability and timing analysis may be performed using maximum estimated execution times. To arrive at average-case estimates, (i) the average estimates for execution times may be used and the process repeated, or (ii) a simulation, with the execution times varying by known statistical distributions or using actual data collected from a running system, may be performed.

OTHER PATHS

The analysis and representation of paths through processes that are not defined as 'real-time' in the operating system they execute or those that do not execute on a real-time operating system are often used in less critical paths through the system. Examples include support for historical archives, hard-copy printouts and supervisory displays. Although operator displays are a critical component in control system implementations, common implementations tend to be self-reacting control actions (automatic action through a dedicated PLC) with the displays provided for supervisory control.

Screen refresh rates, data storage rates, printing rates

and the characteristics of processes handling these tasks are often the determining factors in satisfying the timing requirements associated with these paths. However, the other delays due to signal scanning and signal transmission still need to be taken into account.

MAINTENANCE

Control system implementations rarely stay the same once they are implemented. Enhancements to the control system and enhancements to the plant or process present a constant challenge to satisfying the original formulated timing requirements and newer requirements. Changes need to be carefully reviewed to ensure that the timing and reliability characteristics of the implementation are still valid. Often, enhancements result in architectural changes, which in turn result in the addition of hardware components to the system, making migration of functionality from the previously overloaded processors possible. However, these additions also tend to increase the load on the network links; analysis is required to ensure that sufficient network bandwidth is available to satisfy the end-to-end timing requirement paths that pass through the network.

AN OBJECT-ORIENTED MODEL

Distributed control systems contain many diverse hardware and software elements that complicate the task of analyzing and predicting the worst-case and aver-

age-case time response. Approximations and assumptions are often required. However, fitting a generic hardware or software model will make the software unusable in specific situations. **Choosing a hardware or software model is typically a compromise between generality and usability.**

Fortunately, an object-oriented model offered by TimeSys Corporation can be used to describe the timing and reliability requirements flexibly in terms of objects in the system. Such a description is readily understood and related to the various entities found in distributed process control systems. These systems contain a wide variety of devices, busses and operating platforms such as PLCs, RTOSs, LANs, FieldBus, CANBus and other proprietary devices and protocols. The model can be applied to a wide variety of real-time applications by deliberately not containing hard-coded assumptions regarding the types of properties of objects. Designers, developers and **users can add custom attributes to any object** and save the object as a template (object type). Usability is therefore not compromised without losing the generality of the framework. Timing analyses based on RMA, simulations and reliability analyses are built-in. In addition, custom analysis and simulation modules can be 'plugged-in' by users, yielding an integrated but extensible framework.

TimeWiz™ from TimeSys Corporation, currently in Beta, supports this object-oriented model with a famil-

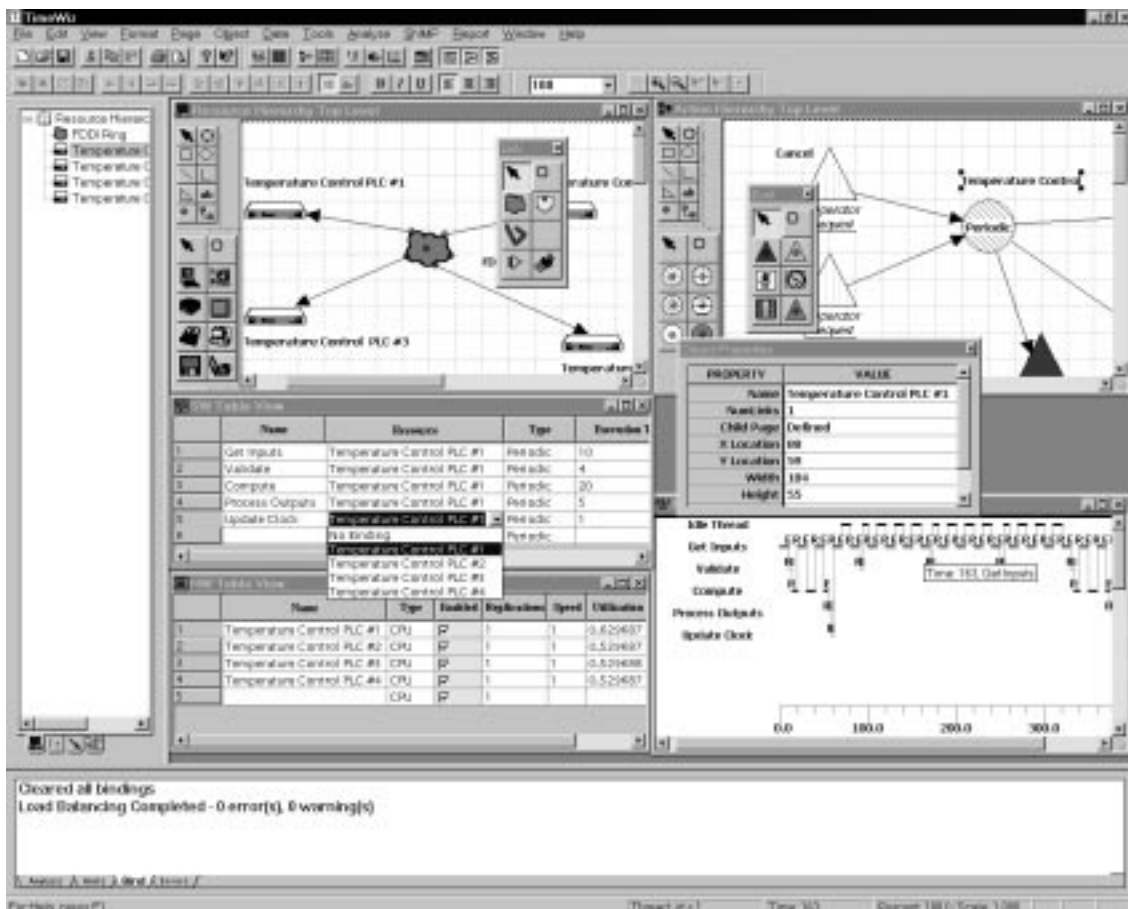


Figure 4: A screen image of TimeWiz software

lar user interface modeled after the Microsoft Visual C++ development environment. A snapshot of the screen during an example session with TimeWiz is shown in Figure 4. With this interface, the process control system can be visually and independently described in terms of its hardware and software architectures.

Elementary hardware objects consist of resources (such as processors) **interconnected by links** (such as buses and network links). **Elementary software objects consist of events** (such as "Temperature Alert") **and actions** (such as "Close Valve"). Relations between elementary objects are expressed and maintained as connections between visual objects. These elementary objects can also be grouped to organize hardware and software architectures hierarchically and more intuitively. Timing constraints within a hardware resource and end-to-end event-response requirements can be expressed easily and analyzed immediately to check whether schedulability requirements are met. Software components can be bound to hardware components manually or automatically, and can be changed at any time during the design, implementation or maintenance process. This allows maintenance changes and enhancements to be performed quickly using the tool. The resulting scenarios can then immediately be analyzed and simulated within the tool prior to their implementation. An object attribute inspector, a spreadsheet view, informative charts, execution timelines and integrated/customizable report generation provide easy and intuitive access to object data. A catalog designer enables the visual design of custom objects and drawing palettes. These objects can then be instantiated within a system configuration and custom analyzed via user-written scripts. With support for redundancy analysis, TimeWiz serves as a general-purpose and highly customizable tool to design and perform timing analysis of process control systems.

CONCLUSION

Timing requirements must be well-specified and analyzed in distributed process control systems. All end-to-end timing requirements for various signals must be met during the design and maintenance of the system. Established techniques such as RMA can be used in a multi-process system to determine the worst-case response. Simulation is used to determine average-case response times. With a well-defined problem model, other analysis techniques such as failure mode and reliability analyses can also be performed. Fortunately, there is a new breed of friendly object-oriented tools for real-time and embedded systems which

1. allow the user to represent the system both in terms of hardware and software architectures,
2. allow automated and/or manual 'binding' of software processes to hardware processors and network elements,
3. allow the specification of characteristics such as the use or non-use of operating systems and loop implementation characteristics,
4. perform worst-case timing analysis (thread-path tim-

ing and RMA based analyses), and

5. perform simulation of the distributed system.

As embedded devices (such as intelligent home automation, automotive electronics and personal communication devices) become ubiquitous, tools for standardized system description and analysis are necessary and timely. The process control industry can greatly be.

Mr. Srinivasan is the Chief Executive Officer of TimeSys Corporation located in Pittsburgh, USA. Working with Westinghouse Electric Corp., he actively participated in the design, implementation and analysis of the protection and control systems for the Sizewell B Nuclear Power Plant in the U.K. He has more than 10 years of industrial experience mainly in the area of industrial control systems. He received his M.S. degree in Chemical Engineering from the University of Houston, Texas. Over the years, he has worked with a wide variety of international companies handling increasing degree of responsibility within the organizations.

Dr. Raj Rajkumar is on the Research Faculty at the Department of Computer Science at Carnegie Mellon University in Pittsburgh, USA. He leads the Real-Time and Multimedia Laboratory, which develops the RT-Mach microkernel-based real-time operating system. He received a Ph.D. degree in Computer Engineering from Carnegie Mellon University in 1986 and 1989 respectively. From 1989 to 1992, he was a Research Staff Member at the IBM Thomas J. Watson Research Center at Yorktown Heights. He has been working on real-time systems since 1985, and his work on priority inversion and priority inheritance protocols is extremely well recognized in the real-time research community as well as among the practitioners. He is also the author of a book titled "Synchronization in Real-Time Systems: A Priority Inheritance Approach" on this topic.