2.

CONF-930401--2

Considerations for Control System Software Verification and Validation Specific to Implementations Using Distributed Processor Architectures

CONF-930401--2

DE93 007920

John K. Munro, Jr.
Instrumentation and Controls Division
Oak Ridge National Laboratory*
P.O. Box 2008
Oak Ridge, Tennessee 37831

DISCLAIMER

Submitted to the American Nuclear Society Topical Meeting
Nuclear Plant Instrumentation, Control and Man-Machine Interface Technologies
April 18-21, 1993

---

*Managed by Martin Marietta Energy Systems, Inc., for the U.S. Department of Energy under Contract DE-AC05-84OR21400

CONSIDERATIONS FOR CONTROL SYSTEM SOFTWARE VERIFICATION AND VALIDATION SPECIFIC TO IMPLEMENTATIONS USING DISTRIBUTED PROCESSOR ARCHITECTURES

John K. Munro, Jr.
Instrumentation and Controls Division
Oak Ridge National Laboratory*
P.O. Box 2008
Oak Ridge, Tennessee 37831
(615) 574-0635

ABSTRACT

Until recently, digital control systems have been implemented on centralized processing systems to function in one of several ways: (1) as a single processor control system; (2) as a supervisor at the top of a hierarchical network of multiple processors; or (3) in a client-server mode. Each of these architectures uses a very different set of communication protocols. The latter two architectures also belong to the category of distributed control systems. Distributed control systems can have a central focus, as in the cases just cited, or be quite decentralized in a loosely coupled, shared responsibility arrangement. This last architecture is analogous to autonomous hosts on a local area network. Each of the architectures identified above will have a different set of architecture-associated issues to be addressed in the verification and validation activities during software development. This paper summarizes results of efforts to identify, describe, contrast, and compare these issues.

INTRODUCTION

In the last half of the 1980s a number of new types of multiprocessor computer systems reached the commercial market. Advances in computing technology opened up new options for processing/communications architectures, and the availability of networking hardware and software also increased. Processor architecture options now available for use in nuclear power plant control system applications bring advantages such as greater flexibility in design, more effective resource allocation and use, improved performance, possibilities for improved plant maintenance, and greater system availability, reliability, and fault-tolerance. Through efforts of federal agencies, initiatives to develop and adopt operating system and communications standards opened up the commercial marketplace so that a user would not be tied to the product line of a single vendor. All these factors contributed significantly to opening up a range of distributed processing options for plant control and management.

However, the proliferation of available processor architecture options, while offering advantages such as greater flexibility for design and implementation, also brings greater complexity to system specification and design efforts. This has consequences for the application software development life cycle, in particular for verification and testing activities, first during the development process, and then later during maintenance.

Digital control system software verification and validation has been an area of ongoing investigation and activity in the Reactor Systems Section of the Instrumentation and Controls Division at Oak Ridge National Laboratory. New possibilities presented by distributed processing

---

*Managed by Martin Marietta Energy Systems, Inc., for the U.S. Department of Energy under Contract DE-AC05-84OR21400.

applied to plant control are of great interest because of the potential advantages offered and the challenges posed, as well as concern to find ways to ensure the reliability of software developed for control.

The work reported here describes initial efforts to establish a systematic approach to the analysis, evaluation, verification, and testing of digital control systems with respect to consequences of choice of system architecture. The range of multiprocessor architecture options is characterized in terms of a progression from a highly centralized system to a collection of independent (fully distributed) systems communicating over a network. Examples chosen reflect architectures available commercially. The intent of this study is to provide a framework for use in planning control system upgrades for existing commercial nuclear power plants. Verification and testing activities will need to cover architecture-related issues identified in this report. Work to date on this topic has not progressed far enough to provide specific details of the architecture dependencies, the primary goal of this initiative.

BACKGROUND

A distributed computer/control system consists of "multiple autonomous processing elements cooperating in a common purpose or to achieve a common goal."[1] Distributed processing architectures for plant control systems are of interest for many reasons, such as

- Concern about obsolescence of equipment (vendor and supplier reliability),
- Greater use of modularity to increase reliability (operations and maintenance),
- Access to multivendor support (open systems, lower costs),
- Improved fault tolerance through redundancy and distribution of processors,
- More effective isolation (common mode failure protection) of failures,
- Capability to reconfigure plant resources for improved availability,
- Evolutionary upgrades to improve performance and capacity, and
- Greater flexibility in implementing different control algorithms and strategies (discrete, continuous, optimal).

Advantages from distributed processing introduce an associated increase in the level of system complexity, thus requiring closer attention to verification and validation (V&V) activities. This paper identifies architecture-dependent issues that verification and testing activities must consider—some of the factors contributing to cost in terms of design, verification, and validation activities.

The range of processor and communications architecture options for plant control extends from highly centralized systems to those in which all processors operate almost totally independently (minimal communication occurs). A set of five systems has been selected for consideration in this paper as a vehicle to compare and contrast the issues considered:

(a) High-performance central CPU with multiple I/O channels,
(b) Master/Slave system with centrally controlled processing or decision-making (strong hierarchical control),
(c) Cooperative processing (supervisory control),
(d) Client/Server system, and
(e) Fully distributed processing system.

These systems are shown schematically in Fig. 1.

Large central computers have been the exclusive architecture used until very recently. These computers are typically a large mainframe system (a single processor or a small number of tightly coupled processors) with large numbers of I/O channels connected to peripheral devices such as disks, tapes, display panels, printers, and multiplexors (e.g., terminal concentrators). Plant signals are digitized and sent directly to the central computer, which makes control decisions and sends control commands to the actuators.

The next evolutionary configuration (Master/Slave) adds "intelligence" (limited local processing capability under central control) to the I/O devices. This architecture makes extensive use of

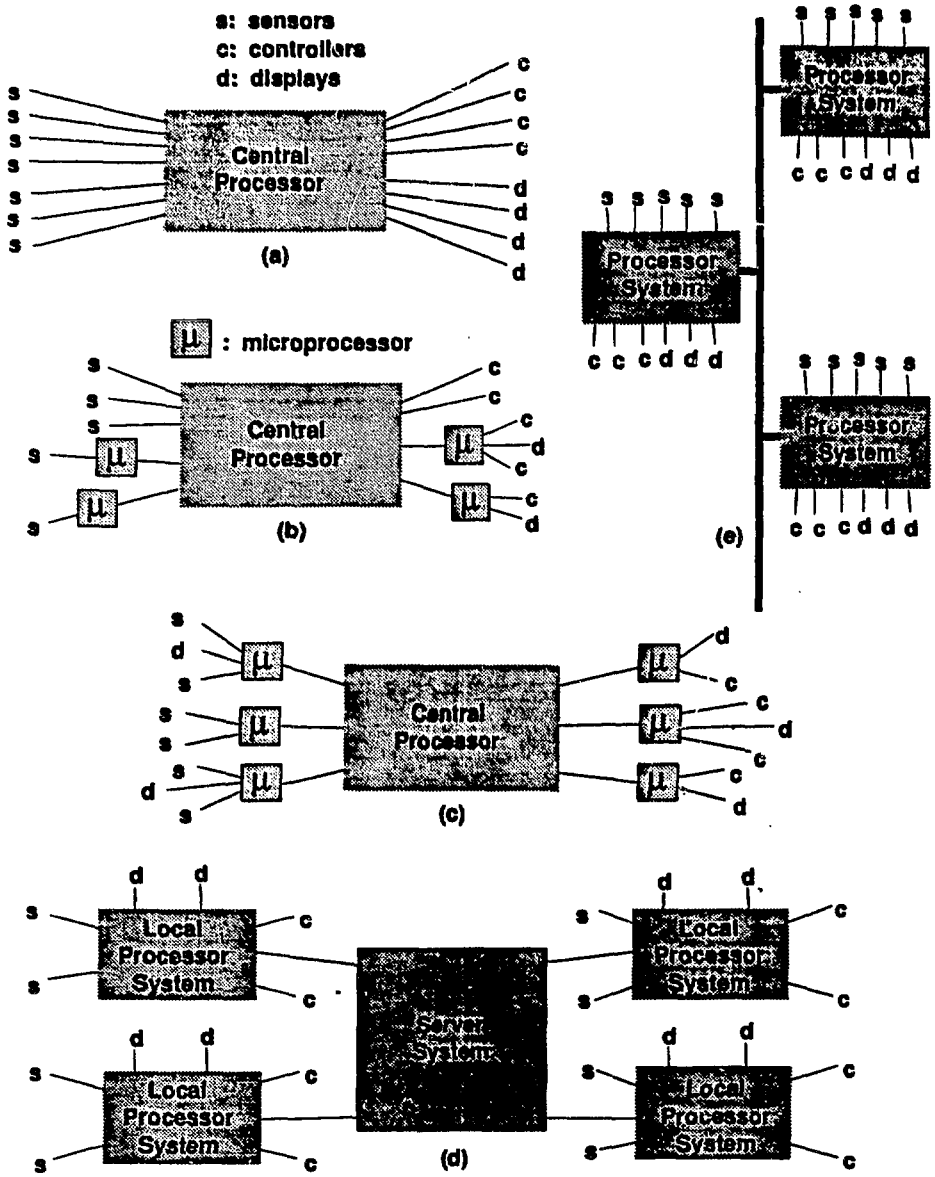

Fig. 1. Range of architecture options: (a) high-performance central system, (b) master-slave, (c) cooperative processing, (d) client-server, and (e) fully distributed.

embedded applications; that is, a processor integrated into a device that performs specific functions. Plant signals can be monitored locally for problems (signal validation) before being sent to the central computer system. Likewise, higher level commands can be sent to actuator processors that monitor the status of the actuator and check that the command sent is consistent with permissible actions. If not, the central system can be notified that a possible problem exists. All primary aspects of control are still associated with the large central system, so this architecture is suitable for strong hierarchical control schemes. All data are centrally maintained and managed.

The third progression in architecture is represented by many cooperative processing schemes in which a significant amount of computer processing functionality is shifted to local microcomputer systems. Copies of data needed to support local execution of applications are sent to these systems and stored on them as long as needed. High-level coordination of control decisions and functions is still done on a large central computer. Such an architecture is well-suited for supervisory control strategies. Selection of appropriate communications protocols and networks will permit use of heterogeneous computer systems.

The fourth category of plant control options is the Client/Server architecture, which is currently very popular for distributed database and transaction processing applications. The server functions as a vehicle for coordinating (synchronizing) activity and maintaining master copies of all data needed by more than a single client. Client systems often differ from servers only in terms of storage capacities of memory and archival media. Each client does a substantial amount of processing, and most control decisions are made locally. This architecture accommodates heterogeneous computer systems as long as data access protocols are available in addition to the necessary communications protocols.

The final category of architecture options is a fully distributed system of autonomous hosts connected to a common network used to communicate high-level plant status information and to synchronize control actions. Use of appropriate communications and data access protocols permits heterogeneous systems to be used for this architecture as well.

Selecting a vendor who will provide homogeneous computer systems for any of the architectures described above does not guarantee reliable long-term support. Plant lifetimes are measured in many decades, while the commercial lifetime of a particular chip is one-tenth of that or less. Even large mainframe system lifetimes are typically 7–10 years. A system used for more than 15 years is an exception. No vendor can guarantee a homogeneous computing solution to a plant control system for a significant fraction of the plant's lifetime. Such considerations have provided the impetus to develop open systems architectures and open systems standards that can be used as the basis for procurement of computer systems from multiple vendors. Greater use of standards will be an important factor in the design of future digital control systems.

Not all the standards needed for use of heterogeneous computer systems in a control system exist, although the International Standards Organization (ISO) has developed some standards that can be used. A large effort is underway in the United States to develop a comprehensive suite of related standards (POSIX) covering operating systems, file structures, communications, applications programming languages, and more. Application of these standards is intended to provide some protection against obsolescence.

Manufacturing costs of some types of computer components have decreased dramatically in the last few years, especially memory and long-term data storage media. This decrease in costs can strongly affect resource management design choices, such as use of shared or local data storage.

Many of the reliability-related issues affecting the design, verification, and testing of distributed digital control systems have much in common with these same areas for distributed operating systems. The same technical problems are present with both, but with variations in emphasis. Stankovic provides a list of techniques to improve reliability of distributed operating systems:[2]

- Error (fault) confinement to limit damage to operation,
- Fault detection,

- Fault masking,
- Retry,
- Diagnosis,
- Reconfiguration,
- Recovery,
- Restart,
- Repair, and
- Reintegration.

This list is worth considering in designing control systems. Each item in the list is discussed in the collection of papers selected by Stankovic.[2]

PLANNING FOR SYSTEM VERIFICATION AND VALIDATION

Advantages offered by distributed processing present a temptation to select one of the distributed system architectures and to assume that experience with centralized systems can be transferred to the analysis, design, and early development stages of the new control system. An increased level of processor parallelism is assumed to provide adequate support to distribute the processing load. However, the most serious consequences of greater use of distributed processing may not show up until late in the development cycle. Items that could introduce unexpected surprises are identified in this section, because they all affect V&V planning.

Two items stand out as being the most important: system response time and system configuration management. Of these two issues, time is by far the most important. Time is not only important for completing critical tasks by required deadlines, but also to record and store event sequences in the correct order by use of synchronization.

Event synchronization is a very important factor in data validation. It can be critically important during rapid excursions in monitoring any kind of physical flow (heat, material) because control actions are predicated on application of conservation laws. Lack of synchronization can produce the opposite control action to what is intended. Control algorithms to be used on distributed processors will have to be examined for sensitivity to synchronization latency, that is, the degree of accuracy in synchronization that can be maintained, so that problems do not develop as result of unexpected excursions.

Scheduling involves timing, task prioritization, resource management, and possibly communications management. A scheduling analysis must look for possible race conditions that can produce artificial states that may cause program execution to halt or to lose synchronization, and lead to system failure. Analysis tools for system design will have to incorporate graph theory used in understanding aspects of network behavior, queuing theory and other topics used widely in operations research, Petri nets used in digital communications design,[3] and finite-state (discrete event) simulation models. These tools may also be needed to assist with verification activities. New approaches such as Timed-Transition Logic or predicate logic languages (e.g., Prolog) with temporal extensions may be needed to achieve a clearer understanding of system scheduling behavior.[4,5]

Use of simulation for re-evaluation needs to be included as part of the maintenance support environment for distributed control systems. This is particularly important for any part of the system where timing and scheduling strategies are critical to reliable control. Performance matching is another timing-related consideration.

Configuration management must be considered at two quite different levels: plant management and control system operation. At the plant management level, attention must be paid to use of development and maintenance tools, for example, compilers (including version, release, patch number). Verification and testing activities needed are those typically associated with software engineering. At the control system implementation level, attention must be given to the use of additional processors to provide redundancy for increased system availability and reliability.

Verification and testing activities needed here must focus on system timing and scheduling behavior affected by any control task changes.

Communication protocols become significant design issues as the level of use of distributed processors increases. Choice or design of protocols must consider performance (timing), fault-tolerance, reliability, security, and data integrity.

Complexity management can become a major concern very quickly. Digital technology opens up many attractive possibilities for control and diagnostics, but it first needs to be used to improve reliability of existing designs and implementations before tackling systems of greater complexity. Doing smarter things with computers frequently requires making more measurements and passing more messages back and forth between processors. These activities use resources, such as processor cycles, and increase system overhead.

Complexity can be managed in many ways. The most common approach is to partition a complex problem into smaller units, each of which presumably will be easier to handle. Verification and validation activities must include an analysis of the partitioning choices made, especially to determine that partitioning of the problem (as well as tasks) for the development process does not occur too soon.[6] In the case of application programs in a distributed processing environment, two options are available: one program partitioned into many concurrently executing tasks, or many programs running separately (synchronously, if necessary) but communicating through some common operating system. The single program can be implemented in either of two ways: partitioned by the operating system at execution time (post-partitioned), or by the application designer during development (pre-partitioned).

In safety or mission-critical control applications, a developer must be careful not to partition the system too soon in the development sequence. In very complex applications, the move to simplify the problem by partitioning too soon introduces serious problems, especially with regard to completeness of requirements and design.[a] If the problem is so complex and difficult that development of the whole system cannot proceed through the early development phases without partitioning, then either the existing technology is not yet mature enough to support development or some other solution must be found.

Software reuse for real-time control systems must be evaluated with much greater care than for other application domains. Attention especially must be given to the original application for which it was written, to understand all the design assumptions made and the environment in which it was used. Care must also be given to use of software on a different processor chip than the one used initially or in cases where the same chip as the original is used, but running at a different clock speed. Even if no changes must be made to the source code, timing relations can be modified. Likewise, the same compiler must be used to guarantee that the machine-level instructions are the same.

Software maintenance must deal with nearly all the same issues associated with software reuse. Correcting even simple errors can result in new code with different timing sequences. Such a problem is difficult enough in a homogeneous processor environment; it can easily be worse in a heterogeneous processor environment.

Data integrity (data validation) can be affected by synchronization failures discussed above. In distributed systems, assuring consistency in the use of units and units conversion becomes a greater problem. Other attributes of data that are easy to verify on a single processor system become more difficult to verify on a large distributed system. Greater care is needed in describing

---

[a]A growing body of recent anecdotal material in the RISKS-FORUM news group of Netnews (on the Internet) suggests that the emphasis placed on problems with coding errors in safety-critical software has diverted attention from a more serious source of deficiencies: inadequate (including incomplete) requirements and/or design specifications.

data items in databases, especially when limitations significant to plant operations may apply. As the degree of distribution and heterogeneity in processor systems increases, the care required to describe widely accessible data and to maintain concurrency also increases.[7]

Data storage strategy can affect data integrity. An obvious example is too heavy reliance on (volatile) memory (RAM) located on the CPU board. Plant data must be written frequently to other storage media to ensure stable data storage needed either for recovery of control or transition to a degraded control mode. Stable data storage must be tied to system reconfiguration strategies.

Use of different data representations can cause difficulty. This can happen, for example, when using a particular data value that has been stored in computers from different vendors. When the value from the different systems is used by a third computer in an arithmetic difference, the result may not be the expected null. Decision logic dependent on getting a null result in this case might not give the correct response. The different systems considered here could conceivably be processors on two different controller cards plugged into a common backplane of a minicomputer or workstation.

METHODS FOR EVALUATION AND TESTING

Models, simulations, and formal methods are the primary tools recommended for system analysis. They are also needed to assist with system testing during development and system integration. They are needed again to assist with maintenance and periodic testing in support of operations to verify that unauthorized or accidental system changes have not occurred. Discrete event simulation tools used in the analysis and design of the distributed control system must continue to be used as a regular part of maintenance, especially when enhancements are made or a different chip design is used. Formal methods are very valuable for challenging assumptions, especially when they are not recognized. Hence, they help to reveal unintended functions.

Recent anecdotal material strongly supports the need to conduct broader system hazard analyses and reviews periodically that extend beyond the particular subsystems affected by maintenance activities, especially when changes are made to the system.[6] Such a practice is expected to be even more important as the level of processing distribution (hence, system complexity) increases.

CONCLUSIONS

Possible advantages identified in the Introduction that favor greater use of distributed processors for control applications are not necessarily straightforward to implement. These advantages are possible because an increase in system complexity is possible. They appear to be attractive because they are conceptually within reach; but they are more difficult to verify and test thoroughly, making reliability more difficult to assure. This paper identifies architecture issues expected to vary with the degree of processor distribution and to influence strongly the requirements for verification and testing activities. A detailed picture of the architecture dependence of each issue is still being developed.

Many of the tools needed to design and maintain highly distributed reliable control systems do not yet exist and this is an area that needs attention. Reports about research examples of tool development and use are difficult to locate in the literature.

Two new data types should be considered as the processor architecture becomes more distributed: measurement and location. The reasons for this are as follows:

a. Measurement data type: Data integrity is harder to ensure the more the data are distributed. When integrity is critical and depends on knowing and being able to verify certain attributes of the data, a reliable way to keep the necessary attributes bound tightly together needs to be considered. One way to do this with computers is to code the important information into an indivisible unit or representation such as a particular long word type. Attributes bound to a measured value must include

- time the measurement was made, and
- physical units of the measured value.

Other important attributes include

- time the value is made available for shared access, and
- time period measurement is valid for control decisions (retention time).

b. Location data type: Each processor system must have some way to locate data—its own local address scheme. Another level of reference is required to locate data across a local network. A large distributed processor system may require multiple levels for addresses. Correct data addressing is critically important for high-reliability systems, enough so to warrant introduction of a separate data type for use in real-time control applications.

As others have noted,[8] if improved technology or methods were first used to build systems like those already in use, but with improved reliability, existing systems could be made much safer. However, too often anticipated benefits from new technology are used to justify proceeding with development of much more complex systems than turns out to be warranted. This is a contributing factor to the "software crisis," especially for process control applications.

One strategy to consider for use with distributed digital systems (because the processing capability may more likely permit it) is to operate the processors in a mode as close as possible to an analog system. Analog systems provide a continuous response. A digital system can approach a continuous response using finer and finer time slices and executing instructions continuously under conditions of maximum load. Logical tests and branching would have to be prohibited except for permitting the system to continue operation or to shut it off. Control logic choices permitted with mechanical relay systems should initially determine the limits of complexity of system design, especially for critical systems. There appears to be no fundamental reason so far to exclude the introduction of greater complexity. Limitations are due to insufficient experience with the complexities involved and lack of analysis methods and tools needed to meet the challenges associates with these complexities.

REFERENCES

1. ALAN BURNS and ANDY WELLINGS, *Real-Time Systems and Their Programming Languages*, Addison-Wesley Publishing Company, Reading, MA (1990).

2. JOHN A. STANKOVIC (Ed.), *Reliable Distributed System Software*, IEEE Computer Society, Computer Society Press, Silver Spring, MD (1985).

3. NANCY G. LEVESON and JANICE L. STOLZY, "Safety Analysis Using Petri Nets," *IEEE Trans. on Software Engineering, SE-13* (3), 386–397 (March 1987).

4. JONATHAN S. OSTROFF, "A Framework for Real-Time Discrete Event Control," *IEEE Transactions on Automatic Control, 35* (4), 386–397 (April 1990); JONATHAN S. OSTROFF, "Deciding Properties of Timed Transition Models," *IEEE Transactions on Parallel and Distributed Systems, 1* (2), 170–183 (April 1990).

5. ALEXANDER TUZHILIN, "Templar: A Knowledge-Based Language for Software Specifications Using Temporal Logic," Information Systems Department, Stern School of Business, NYU (preprint) (1990).

6. NANCY G. LEVESON and CLARK S. TURNER, "An Investigation of the Therac-25 Accidents," *UCI Technical Report # 92-108*, Information and Computer Science Department, University of California, Irvine (December 1992).

7. JURIS HARTMANIS and HERBERT LIN (Eds.), *Computing the Future: A Broader Agenda for Computer Science and Engineering*, National Academy Press, Washington, DC (1992).

8. C. JONES, *Systematic Software Development with VDM*, 2nd Edition, Prentice-Hall (1990).