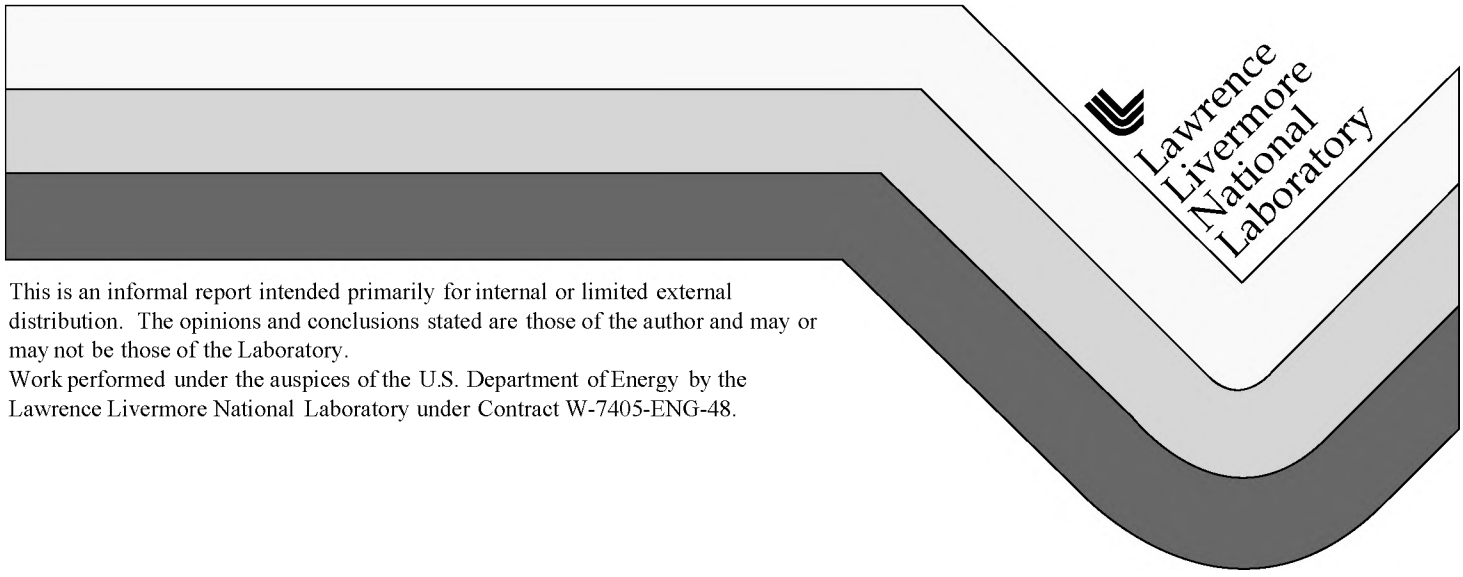
UCRL-ID-133243

Integrated Computer Control System CORBA-based Simulator

P. J. Van Arsdall
R. M. Bryant
F. W. Holloway

January 15, 1999



Work performed under the auspices of the U.S. Department of Energy by the Lawrence Livermore National Laboratory under Contract W-7405-ENG-48.

Integrated Computer Control System

CORBA-based Simulator

FY98 LDRD Project
Final Summary Report

P. J. Van Arsdall
R. M. Bryant
F. W. Holloway

January 15, 1999

Introduction

The CORBA-based Simulator was a Laboratory Directed Research and Development (LDRD) project that applied simulation techniques to explore critical questions about distributed control architecture. The simulator project used a three-prong approach comprised of a study of object-oriented distribution tools, computer network modeling, and simulation of key control system scenarios. This summary report highlights the findings of the team and provides the architectural context of the study. For more information, the reader is referred to detailed project reports listed in the references.

For the last several years LLNL has been developing the Integrated Computer Control System (ICCS), which is an abstract object-oriented software framework for constructing distributed systems. The framework is capable of implementing large event-driven control systems for mission-critical facilities such as the National Ignition Facility (NIF) [1] [Figure 1]. Tools developed in this project were applied to the NIF example architecture in order to gain experience with a complex system and derive immediate benefits from this LDRD.

The ICCS integrates data acquisition and control hardware with a supervisory system, and reduces the amount of new coding and testing necessary by providing prebuilt components that can be reused and extended to accommodate specific additional requirements. The framework integrates control point hardware with a supervisory system by providing the services needed for distributed control such as database persistence, system start-up and configuration, graphical user interface, status monitoring, event logging, scripting language, alert management, and access control. The design is interoperable among computers of different kinds and provides plug-in software connections by leveraging a common object request brokering architecture (CORBA) to transparently distribute software objects across the network of computers.

Figure 1. Computer rendering of the NIF control room.

Because object broker distribution applied to control systems is relatively new and its inherent performance is roughly threefold less than traditional point-to-point communications, CORBA presented a certain risk to designers. This LDRD thus evaluated CORBA to determine its performance and scaling properties and to optimize its

use within the ICCS. Both UNIX (Sun Solaris) and real-time (Wind River VxWorks) operating systems were studied. Performance of ICCS deployment was estimated by measuring software prototypes on a distributed computer testbed and then scaled to the desired operating regime by discrete-event simulation techniques. A study of CORBA protocols continues to guide software optimization as NIF software is being implemented and tested.

The message-driven nature of distributed control places heavy demands on computers and network switches, so a complementary simulation of network architectures for several protocols was undertaken using a network modeling tool (OPNET Modeler). Additional workflow simulations were developed in a general simulation tool (Simprocess) to assess system behavior of high-stress operational scenarios. Understanding the risks and decisions that trade-off in designing the framework and supporting hardware architecture was enhanced by a concurrent program of simulation and prototype validation of the ICCS applied to the NIF example.

Integrated Computer Control System Architecture

The ICCS is a layered architecture consisting of front-end processors (FEP) coordinated by a supervisory system [Figure 2]. Supervisory controls, which are hosted on UNIX workstations, provide centralized operator controls and status, data archiving, and integration services. FEP units are typically constructed from VME-bus or PCI-bus crates of interfaces and embedded controllers that attach to control points (e.g. stepping motors, photodiode sensors, and pulse power supplies). FEP software provides the distributed services needed by the supervisory system to operate the control points. The software is distributed among the computers and provides plug-in software extensibility for attaching control points and other software services by using the CORBA protocol. Functions requiring hard real-time implementation do not communicate over the network and are allocated to software resident within the FEPs or embedded controllers. Precise triggering of fast instrumentation is handled by an independent timing system, which relieves the software and network from supporting hard real-time communications.

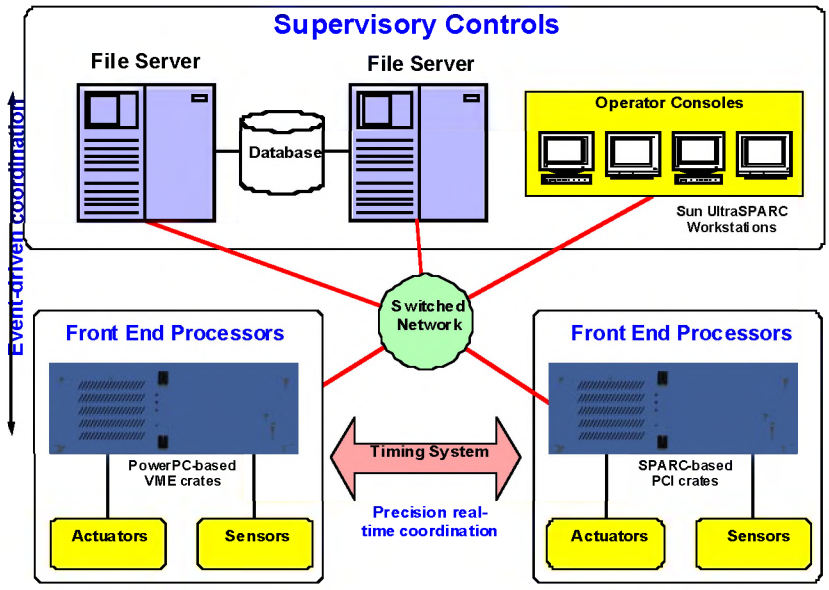


Figure 2. Simplified Integrated Computer Control System architecture.

The operator console provides the human interface in the form of operator displays, data retrieval and processing, and coordination of control functions. Supervisory software is partitioned into several cohesive subsystems, each of which controls a primary subsystem (some NIF examples are alignment controls, laser diagnostics, and power conditioning). A dual server configuration provides enhanced performance with the added benefit of greater availability in the event one server fails. Several databases are incorporated to manage both experimental data and data used during operations and maintenance. The subsystems are integrated to coordinate operation of distributed equipment. Some of the performance requirements that drove the simulation study are listed in Table 1.

Requirement	Performance
Computer system start-up	< 30 minutes
Respond to broad-view status updates	< 10 seconds
Respond to alerts	< 1 second
Perform automatic alignment	< 20 minutes
Transfer and display digital motion video	10 frames per second
Human-in-the-loop controls response	< 100 ms

Table 1. Typical ICCS performance requirements.

Front-end processors implement the distributed portion of the ICCS by interfacing to equipment control points. The FEP software performs sequencing, data acquisition and

reduction, instrumentation control, and input/output operations. The software framework includes a standard way for FEP units to be integrated into the supervisory system by providing the common distribution mechanism (i.e. CORBA) coupled with software patterns for hardware configuration, command, and status monitoring.

To reduce complexity and ease implementation, large hierarchical control systems are often partitioned into a number of vertical slices each of which contain a related subset of supervisory software and associated front-end processors (FEP). Coupling is reduced because little or no message traffic needs to bridge between slices. The slices are constructed by extending the reusable framework components to meet additional functional requirements imposed by its unique equipment. Each slice may also have different performance and physical distribution requirements.

In the NIF example, which is fairly complex, there are eight supervisory software applications that conduct laser shots in collaboration with 19 kinds of front-end processor as shown in Figure 4. Seven of the subsystems are shown as vertical slices comprised of a supervisor and associated FEP that partition the ICCS into smaller systems that are easier to operate and maintain. The eighth supervisor is the shot director, which is responsible for conducting the shot plan, distributing the countdown clock, and coordinating the other seven.

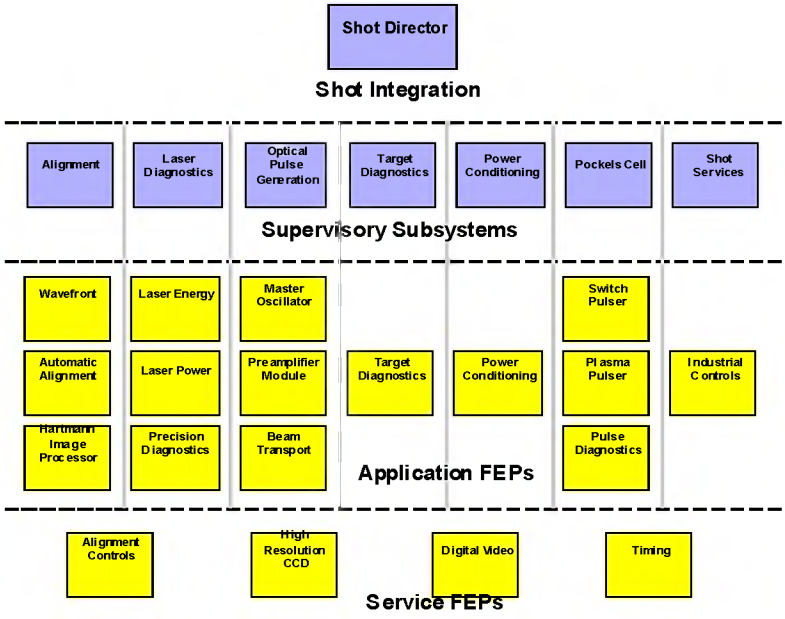


Figure 3. Software responsibilities are partitioned to form loosely coupled vertical slices as shown for the NIF example control system.

ICCS Software Framework

The ICCS supervisory software framework [2] is a collection of collaborating abstractions that are used to construct the application software. The framework promotes code reuse by providing standard coding templates that interconnect via CORBA. Components in the ICCS framework are deployed onto the file servers, workstations, and

FEPs, as shown generically in Figure 4. Engineers specialize the framework (through object-oriented inheritance) for each application to handle different kinds of control points, controllers, user interfaces, and unique functionality. The framework concept enables the cost-effective software construction and provides the basis for long-term maintainability and upgrades by virtue of object technology.

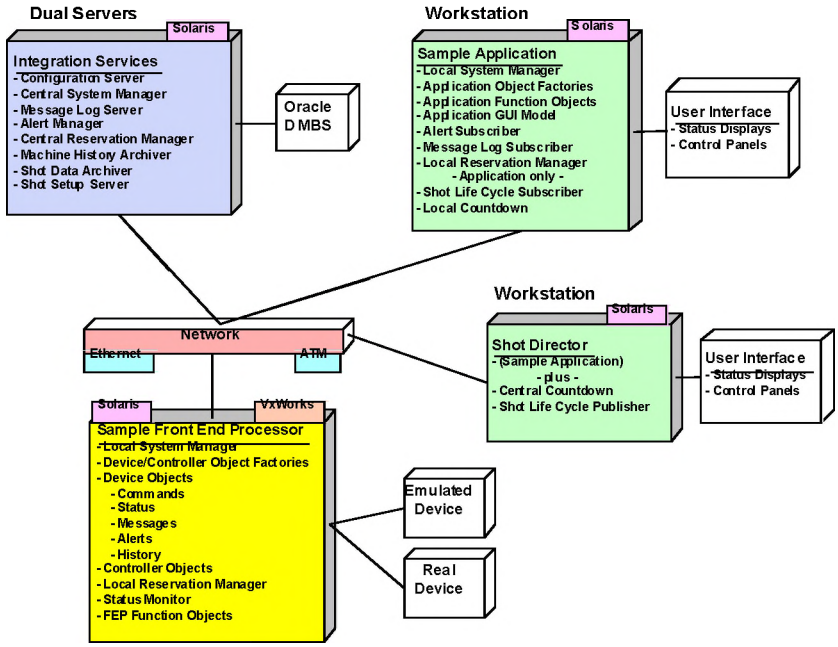


Figure 4. The ICCS reference architecture illustrates deployment of software objects into a server, application workstation and front-end processor on the network.

The following discussion introduces the framework components that form the basis of the ICCS software.

Configuration - a hierarchical organization for the static data that define the hardware control points. Configuration provides a taxonomic system that is used as the key by which clients locate devices (and other software services) brokered by CORBA. During normal operation, configuration provides to clients the CORBA references to all distributed objects. An important responsibility of configuration is the initialization of front-end processors during start-up. Configuration data are stored in the database and describe how and where the control hardware is installed in the system. Sensor calibration data, operating setpoints, and device channels assigned to interface boards are examples of static data managed by configuration. During ICCS start-up, this framework collaborates with an "object factory" located in the FEP to create, initialize, and report the CORBA reference for each control/monitor object.

Status Monitor - provides generalized services for broad-view operator display of device status information using the publisher-subscriber model of event notification. The status monitor operates within the FEP observing devices and notifying other parts of the system when the status changes by a significant amount. Network messages are only generated when changes of interest occur.

Sequence Control Language - used to create custom scripting languages for the NIF applications. The service automates sequences of commands executed on the distributed control points (or other software artifacts). Operators, rather than programmers, create and maintain sequence scripts.

Graphical User Interface (GUI) – All human interaction with ICCS is via graphical user interfaces displayed upon operator workstations. The GUI is implemented as a framework in order to ensure consistency across the applications. Commercial GUI development tools are used to construct the display graphics. This framework consists of guidelines for look and feel as well as many common graphical elements. This component is implemented at the edge of the architecture so that the GUI tool can be replaced with minimal impact if necessary.

Message Log - provides event notification and archiving services to all subsystems or clients within the ICCS. A central server collects incoming messages and associated attributes from processes on the network, writes them to appropriate persistent stores, and also forwards copies to interested observers such as GUI windows.

Alert System - any application encountering a situation that requires immediate attention raises an alert, which then requires interaction with an operator for the control system to proceed. The alert system records its transactions so that the data can be analyzed after the fact.

Reservation - manages access to devices by giving one client exclusive rights to control or otherwise alter a control/monitor point. The framework uses a lock-and-key model. Reserved devices that are "locked" can only be manipulated if and when a client presents the "key".

System Manager - provides services essential for the integrated management of the ICCS computer network. This component ensures necessary processes and computers are operating and communicating properly. Services include orderly system start-up, shutdown, and watchdog process monitoring.

Machine History - gathers information about operational performance for later analysis in order to improve efficiency and reliability. Examples of such information are component adjustments, abnormal conditions, operating service, periodic readings of sensors, and reference images.

Generic FEP - pulls together the distributed aspects of the other frameworks (in particular the system manager, configuration, status monitor, and reservation frameworks) by adding unique classes for supporting device and controller interfacing. These classes are responsible for hooking in CORBA distribution as well as implementing the creation, initialization, and connection of device and I/O controller objects. The generic FEP also defines a common hardware basis including the processor architecture, interface board inventory, device drivers, and field-bus support. The FEP application developer extends the base software classes to incorporate specific functionality and control logic.

Data Archive - The ICCS is responsible for collecting the data from diagnostics, making the data immediately available for "quick look" analysis, and delivering the data to an

archive. The data archive provides a server that works in collaboration with the system manager framework to assure that requested data are delivered to a disk staging area.

CORBA simulation

Past architectural approaches to distributed controls have relied on the technique of building large application programming interface libraries and point-to-point socket communications to give client applications access to server functions implemented throughout the computer network. This practice results in large numbers of interconnections that quickly increases the system complexity and makes software modification much more difficult. To address this problem in the ICCS, software objects are distributed using CORBA.

CORBA is a standard developed by a consortium of major computer vendors to propel the dominance of distributed objects on local area networks and the World Wide Web. The best way to think of CORBA is as a universal "software bus". CORBA provides a standardized backplane into which software objects can "plug and play" to interoperate with one another, even when made by different vendors. By design, CORBA objects interact when implemented with different languages, operating systems, and networks.

At a greatly simplified level, the major parts of CORBA are shown in Figure 5. The interface types and methods provided by the server objects and used by the clients are defined by an industry standard Interface Definition Language (IDL). The IDL compiler examines the interface specification and generates the necessary interface code and templates into which user-specific code is added. The code in the client that makes use of CORBA objects is written as if the server was locally available and directly callable. Each computer on the network has an object request broker that determines the location of remote objects and transparently handles all communication tasks.

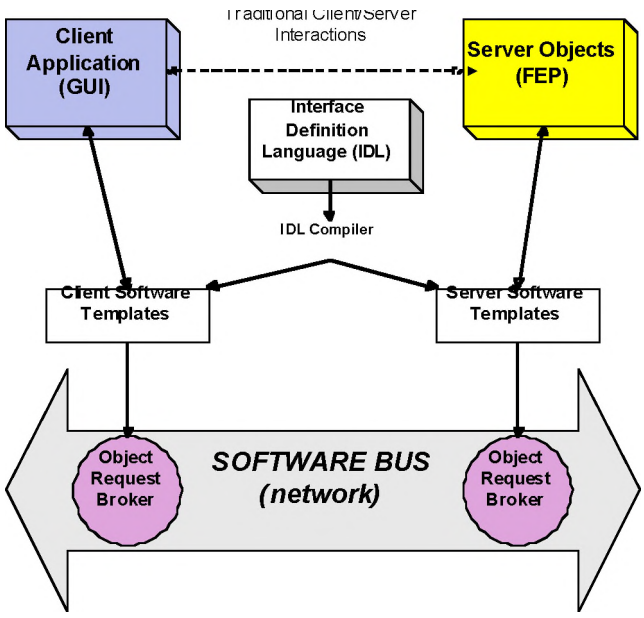


Figure 5. CORBA distribution effectively implements a software bus on the network.

Because CORBA handles the data format conversion necessary to interoperate with diverse computer systems, it is a more heavyweight protocol than previously used for control system distribution. Other researchers have measured object request brokers and shown CORBA to perform about three times slower than point-to-point communication schemes (e.g. sockets). For this reason, a significant testing capability [3] was established to predict the operational performance of a multi-threaded CORBA designed to support concurrent Ada programming (ORBexpress by Objective Interface Systems) under various deployment schemes.

The server program executed on a Sun Enterprise 3000 (2x250MHz Ultra-I), and creates 5 worker tasks to process incoming messages. The client program ran on a Sun Ultra Creator 3D (300 MHz Ultra-II), and used 40 tasks to make calls to the server. For each length $x=2^n$ (where n=0 to n=16) 10,000 sequences of octets were sent to the server (250 per client task). The network utilized 100 Mbit/s Ethernet links. Clock time, CPU time on the client machine, and CPU time on the server machine were recorded for each test.

Performance of CORBA varies according to several factors including client and server processor speeds, network speed and loading, and the efficiency of the particular Ada compiler run-time implementation. Results shown in Figure 6 gave the highest performance out of three Ada compilers tested, obtained with the freely available GNAT Ada translator. Figure 7 shows the same test operated on the same computer and network hardware, but compiled with the Rational Apex Ada compiler. Figure 8 shows results for a real-time (i.e., more deterministic behavior) CORBA implementation, where the server computer was replaced with a PowerPC processor (300 MHz) in a VME crate. The test server code in this case was compiled with the Rational Ada cross-compiler for the VxWorks operating system.

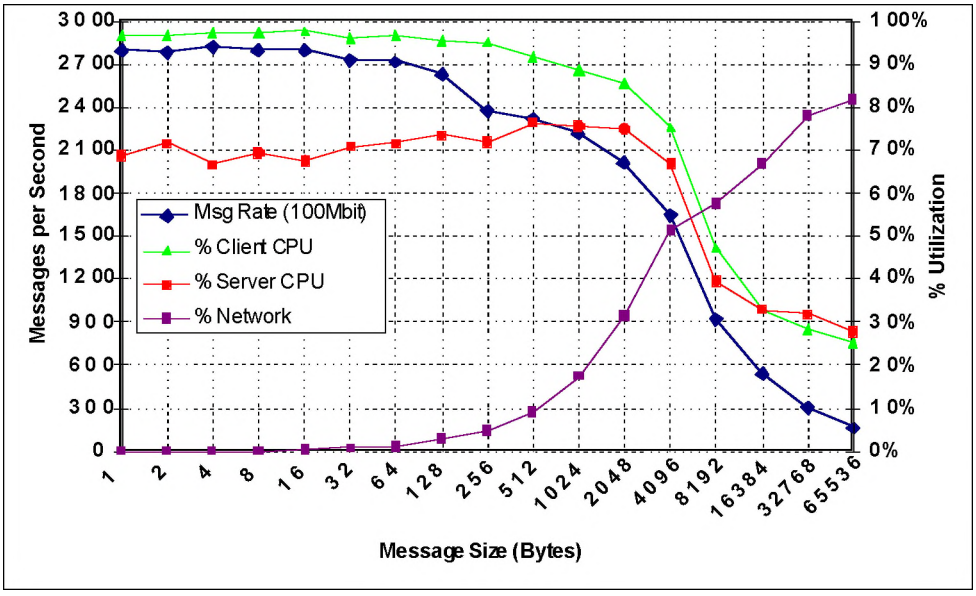


Figure 6. CORBA performance tests of code compiled with GNAT Ada Solaris/Sparc compiler.

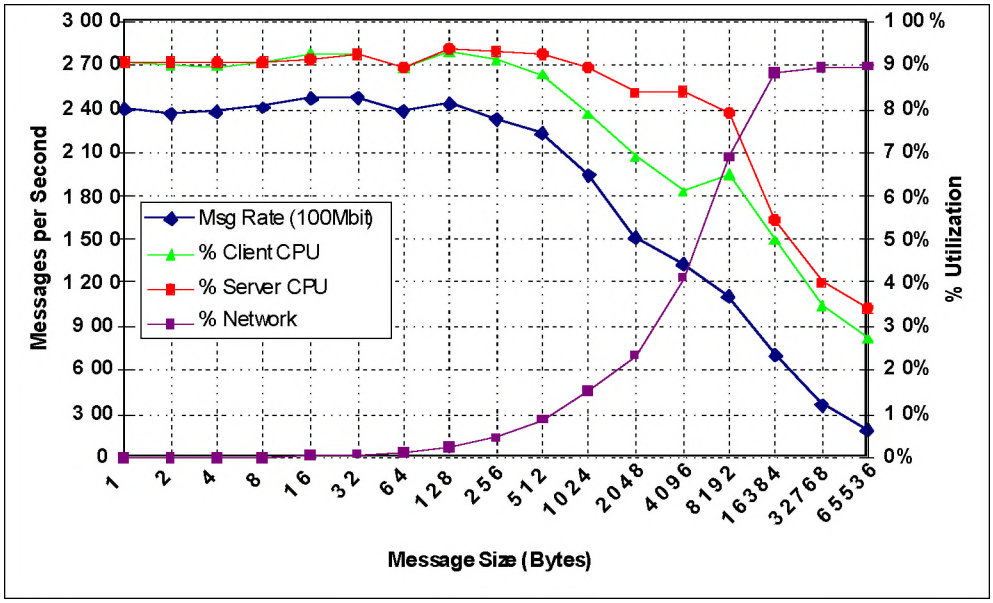


Figure 7. CORBA transaction performance measured for test code compiled with Rational Apex Ada Solaris/Sparc compiler.

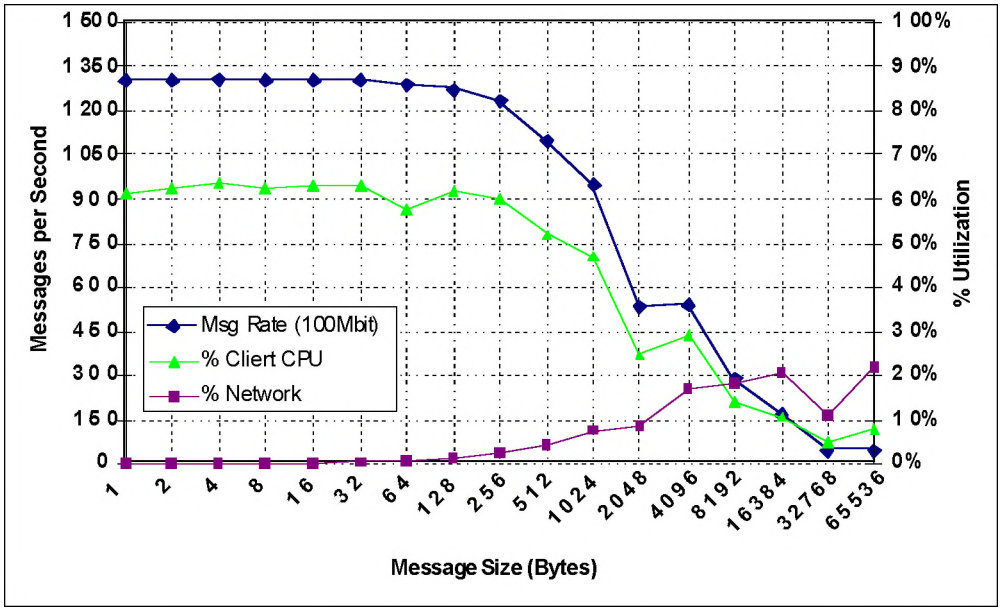


Figure 8. CORBA transaction performance measured for test code compiled with Rational Apex Ada for VxWorks/PowerPC cross compiler.

In typical ICCS deployments, most control and monitoring transactions utilize on the order of 100 byte messages. For this case, CORBA can transact at from 1300 to 2700 messages per second depending on the compiler and target processor tested. The reason for the wide difference in CPU utilization is generally attributed to variations in compiler and run-time library efficiencies and may well be addressed by ongoing improvements in these relatively new products. For small message sizes, the CPU is the limiting resource,

as the network is not heavily utilized (note that other computers need to use the remaining network bandwidth). However, as messages become larger (e.g., during image data transfers) the network does become the limiting factor.

For specific cases of ICCS deployment, subsystems are partitioned into loosely coupled subsystems such that the message rate design point will average around 500 control transactions per second. This approach provides a four- to fivefold capacity margin in the supervisory computers to accommodate episodic bursts of message activity that are occasionally expected in an event driven architecture.

Network communications simulation

Network modeling was performed to study and optimize different switch architectures, transport technologies, and protocols. Figure 9 shows the general computer network studied, which for the NIF example is comprised of 30 workstations, 300 FEPs, and several hundred embedded controllers. Note that other implementations of ICCS could be modeled by straightforward adaptations of this topology. The main control room contains eight graphics consoles, each of which houses two workstations with dual displays. Each software application is assigned to operate on one primary console, although the software can be operated from alternate consoles including remote terminals located near the equipment. File servers provide disk storage and archival databases for the entire system as well as well as hosting centralized management and device naming services necessary for coordinating facility operation.

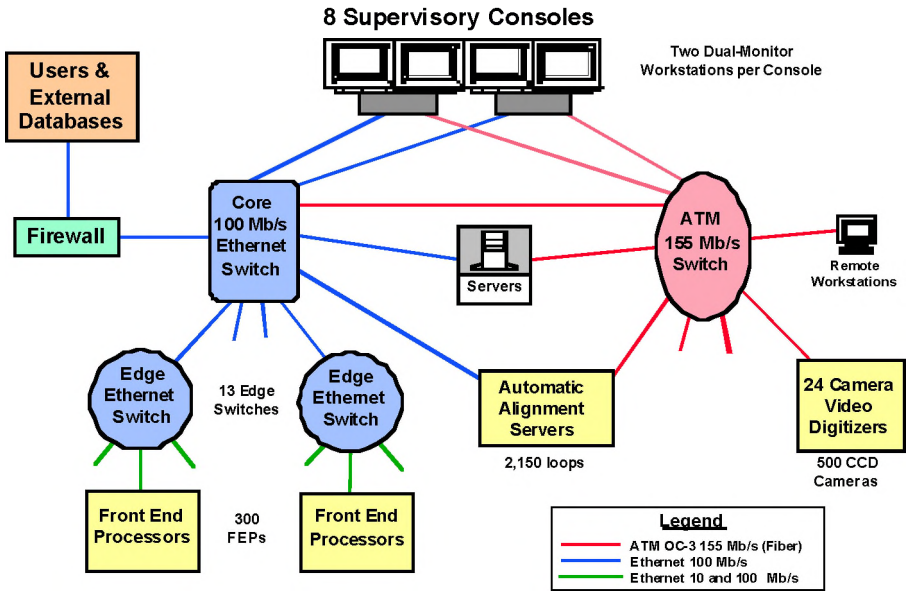


Figure 9. ICCS computer system and network architecture as deployed for NIF.

Key portions of the network were analyzed by applying the OPNET Modeler discrete-event simulation package to simulate network operation [4]. Network analyzers (Network Associates Distributed Sniffers) were used to collect actual performance data from the testbed.

The network design studied is both flexible and high-performance, utilizing Ethernet (10 and 100 Mbit/s) and Asynchronous Transfer Mode (ATM) technologies. ATM is utilized on systems having time-sensitive video requirements. A core ATM switching system provides connectivity to all ATM end-systems via 155 Mbit/s multi-mode fiber links. Ethernet provides connectivity to all end-systems in the network, including the ones that also have ATM connections. A core Ethernet switch provides 100 Mbit/s connectivity and distributed Ethernet switches provide a mix of 10 Mbit/s and 100 Mbit/s connectivity.

TCP/IP (transport control protocol / Internet protocol) is the protocol used for reliable data transport between systems, either over Ethernet or ATM. TCP provides retransmission of packets in the event that one is lost or received in error. In the NIF example network, the only traffic not using TCP will be digitized video and software network triggers. Video is transferred using the ATM adaptation layer 5 (AAL5) protocol. Network triggers are broadcast to many end-nodes simultaneously using multicast protocols.

The ICCS network supports the transport of digitized motion video in addition to the more typical control, status, and large data sets. The network transports video images of 640 x 480 x 8 bits/pixel at 10 frames per second from video FEPs (which digitize camera images) to operator displays. Each uncompressed video stream requires about 25 Mbit/s of network bandwidth. Operator workstations can display at least two video streams. As many as 3 simultaneous video streams are supported from a single FEP, which requires 75 Mbit/s of bandwidth. Because the time to retransmit lost packets using TCP is excessive the protocol is not suitable for use with video traffic. Also, video transmissions need to be multicast to more than one operator console, which TCP does not support.

Digitized video is sent via the ATM application programming interface (API) using ATM's quality of service capability. The API provides an efficient method of moving large, time-sensitive data streams, resulting in higher frames/sec rates with lower CPU utilization than alternative approaches, which is an important consideration for streaming video traffic. Performance testing of a prototype video distribution system indicate that 55% of the FEP CPU (300 MHz UltraSparc AXI) is used to broadcast 3 concurrent streams while 10% of the operator workstation CPU (300 MHz UltraSparc 3D Creator) is utilized for each playback stream.

The first problem studied with OPNET was to evaluate the latency of "trigger" signals sent over the Ethernet and ATM switches. If the latency of the networks were low enough, then separate dedicated "networks" would not be needed to support these triggers. Simulations indicated that the latency would be much less than 1 millisecond, even under the background load necessary to operate the distributed control system. The largest contributor to trigger latency was not the network switching but rather trigger packet processing in the source and destination processors.

In order to study operational scenarios, specific node models were developed in OPNET Modeler for interesting subsystems in the network. This included 32-port ATM and 50-port Ethernet switches, supervisory workstation, automatic alignment system, video FEP, and alignment motor controller FEP. The supervisory workstation was particularly challenging because the model needed to support concurrent clients using several

transport protocols over both ATM and Ethernet. Note that since the network is fully switched (i.e. every end-system has its own non-shared network link), traffic is isolated within the switches to only the ports that are communicating. This allows various subsystems to be independently modeled, which decreases simulation runtime to practical levels.

The component models were used to simulate network operation during NIF's alignment phase, which is when the network carries the most traffic. Traffic levels were defined between the client and server modules in the "workstation" nodes. Traffic statistics were collected during simulation runs and used to evaluate throughputs, latencies, packet loss, and link utilization at various points in the network. Special "filters" were created which allowed viewing the statistics in alternative ways. For example, a filter was created to generate windowed averages of link utilization.

A computer display of the top-level alignment network model is shown in Figure 10. The topology is a subset of the full system comprised of one each of an alignment supervisor workstation (Align_Sup1), automatic alignment FEP (AA_Sys1), video FEP (VFEP1), and alignment control FEP (AC_FEP1). The heavy traffic load that was applied in the simulation is given in Table 2. These traffic levels were obtained from estimates made by design engineers. The load of 5 requests/sec from an automatic alignment system to a video FEP represents a heavy load. The load of 108 requests/sec to the AC_FEP greatly exceeds what any AC_FEP can expect but is more indicative of a load on the alignment supervisor and automatic alignment systems, which communicate with 108 alignment control FEPs.

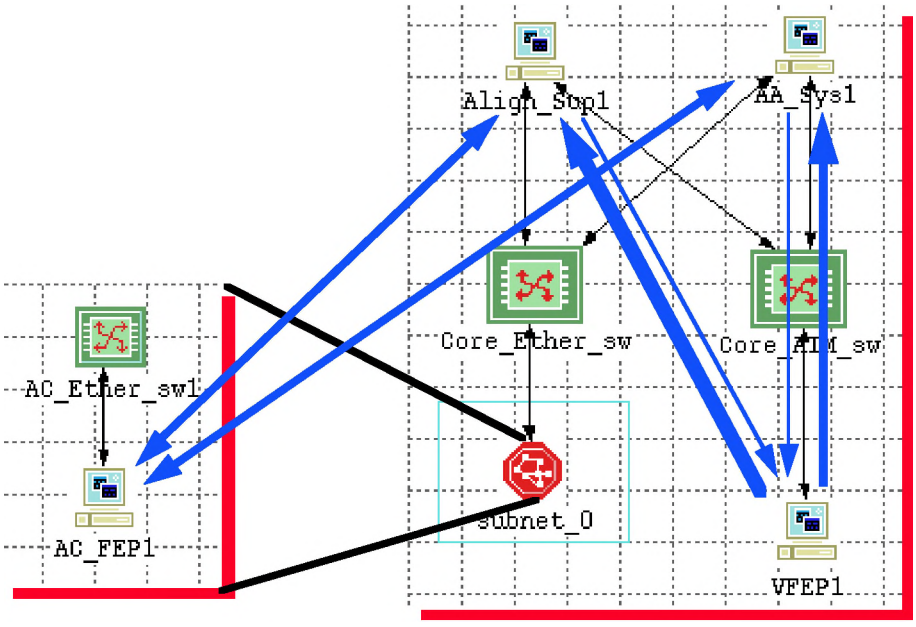


Figure 10. Top-level of hierarchical OPNET network model of NIF's alignment control system.

Client Nodes	Server Nodes	Network	Protocol	Requests[3] / sec	dist.	Req. size Bytes	dist.	Resp. size Bytes[4]	dist.	Server time, msec
Align_Sup	VFEP[1]	ATM	AAL5	10	Const.	1	Const.	307,200	Const.	20.48
	VFEP[1]	ATM	AAL5	10	Const.	1	Const.	307,200	Const.	20.48
	VFEP[1]	ATM	AAL5	10	Const.	1	Const.	307,200	Const.	20.48
	AC_FEP[2]	Ethernet	TCP/IP	108	Normal	200	Const.	64	Const.	0.0625
AA_Sys	VFEP[2]	ATM	TCP/IP	5	Normal	200	Const.	307,200	Const.	20.00
	AC_FEP[2]	Ethernet	TCP/IP	108	Normal	200	Const.	64	Const.	0.0625

[1] An Align_Sup requests 3 video streams from one VFEP (worst case)
[2] Clients randomly select a server
[3] Requests made per each Client node (i.e. Align_Sup or AA_Sys)
[4] 307,200 Bytes = single camera image (640 x 480 x 1 Byte)

Table 2. Load traffic used in the alignment network simulation.

The "dist." columns represent the distributions used for the traffic. "Const." means a constant distribution is applied (i.e. all are the same). "Normal" means that a normal distribution is applied providing a level of randomness to the traffic.

The three video streams between the video FEP and the alignment supervisor are flowing simultaneously and each one is allocated 30 Mbit/s of the 155 Mbit/s ATM bandwidth. Therefore the video streams are spread out in time rather than being burst at the maximum ATM rate. This allows the video FEP to respond without potentially large delays to image requests from the automatic alignment system. Figure 11 shows the service time distribution for video FEP frame requests (i.e., for single images) over a 15-second simulation run. The video FEP service times vary from the minimum of 20.48 ms to a maximum of about 45 ms, which occurs when image frame requests from the automatic alignment system overlap with streaming video requests.

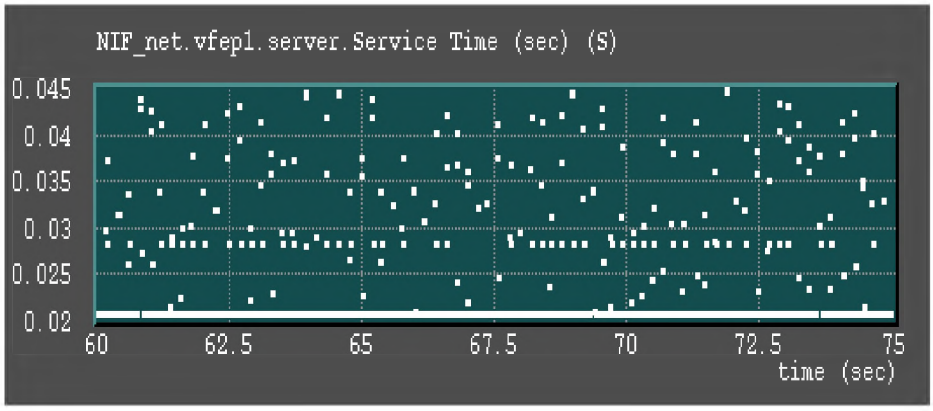


Figure 11. Video FEP single-frame service times over a 15-second interval.

Figure 12 shows the response times for streaming video frames from the video FEP to the alignment control supervisor. The graph shows that the minimum response time is about 111 ms and the maximum is about 135 ms. This latency is the sum of the service time and the transmission time of the image data (about 2.46 Mbits), which at 30 Mbit/s takes about 90 ms. This ~100 ms latency is not overly significant since video is only being

viewed by a person at the supervisory console. The most critical factor is that the jitter variation between images does not lag control enough to become tedious for the operator.

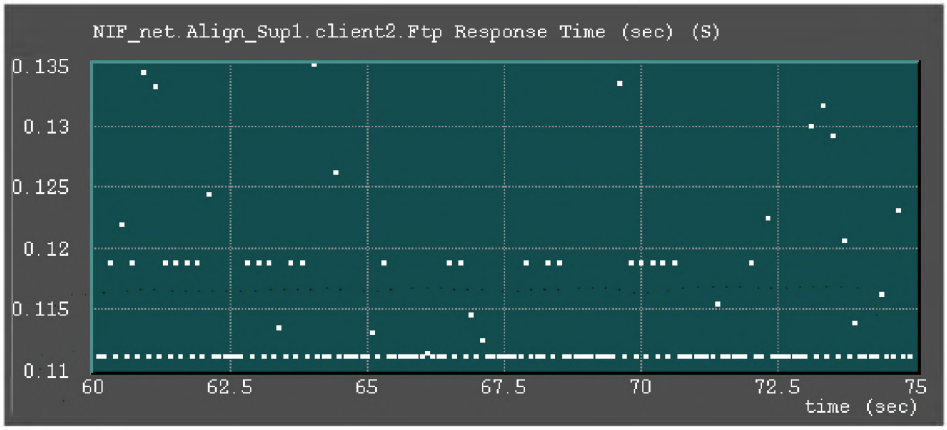


Figure 12. Video FEP streaming video service times.

Figure 13 shows the response time of image frames transferred from the video FEP to the automatic alignment system. In this simulation, since 90 Mbit/s is allocated to the 3 video streams going to the alignment supervisor workstation, the rest of the ATM bandwidth (65 Mbit/s) is available for image transfers to the automatic alignment system. The simulation predicts a response time between about 60 and 75 ms. A minimum of 20.48 ms is the video FEP service time and a minimum of about 40 ms is for the image transmission time, including TCP/IP protocol processing. The automatic alignment system will be receiving images from multiple video FEPs, so the overlapped processing of other images mitigates this latency. Note that if streaming video were turned off so that all ATM bandwidth is available to the automatic alignment system, then these response times could be reduced to less than 40 ms (of which 20 ms is for video FEP service time).

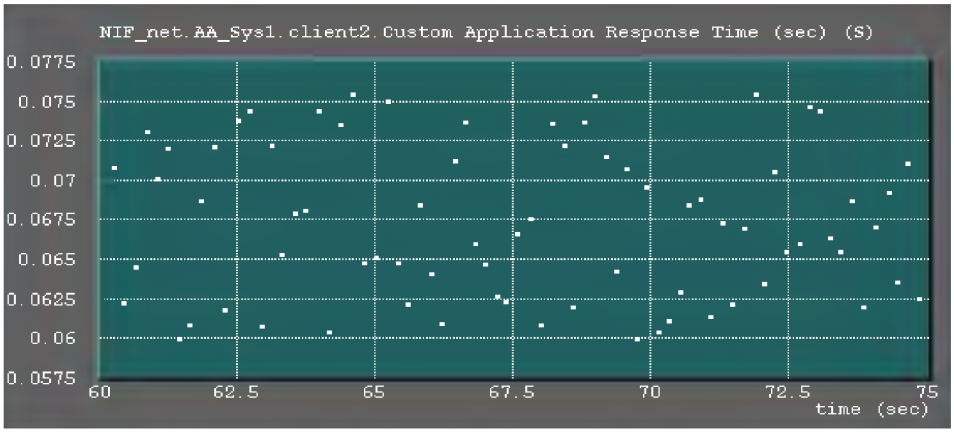


Figure 13. Video image transfer times to the automatic alignment system.

Overall simulation results indicate that the network will be capable of meeting the throughput and latency requirements of the NIF laser alignment process, which presents the highest traffic to network switches. Expected traffic involves the simultaneous transfer of motion video to operator stations at 75 Mbit/s and sensor image transfers at 60

Mbit/s, as well as control messages—all between 100 computers comprised of image processors, supervisory workstations, video digitizers, and motion control systems. The overall requirement is to provide a steady stream of video images to operators while automatic closed-loop processing choreographs the precise adjustment of over 9,000 motorized optics within a 20-minute period.

Control system operational workflow simulations

Several process models of typical control system operation were developed for determining optimal performance configurations of hardware and software in the ICCS architecture. Software tasks, computer hardware, operator interactions, and the collaboration between the functional elements were modeled using a commercial tool, Simprocess. These models help pull together results from the efforts described above. Some performance data were obtained from the network and CORBA simulations as well as testbed measurements.

Since the ICCS framework design is object-oriented, software classes from the development effort are useful for defining simulation models. Of particular interest was the performance of the ICCS under scenarios for computer system start-up [5], status monitoring [6], shot setup messages [7], and distributed processes such as automatic alignment [8].

Restarting all computers in a large system from a power-down condition is a serious concern. For NIF, it is specified to take less than 30 minutes. ICCS will run under both the Solaris and VxWorks operating systems. In most cases the operating system will be downloaded to a computer, rather than reside on local disks. The computer start-up process is shown as a Simprocess model diagram in Figure 14.

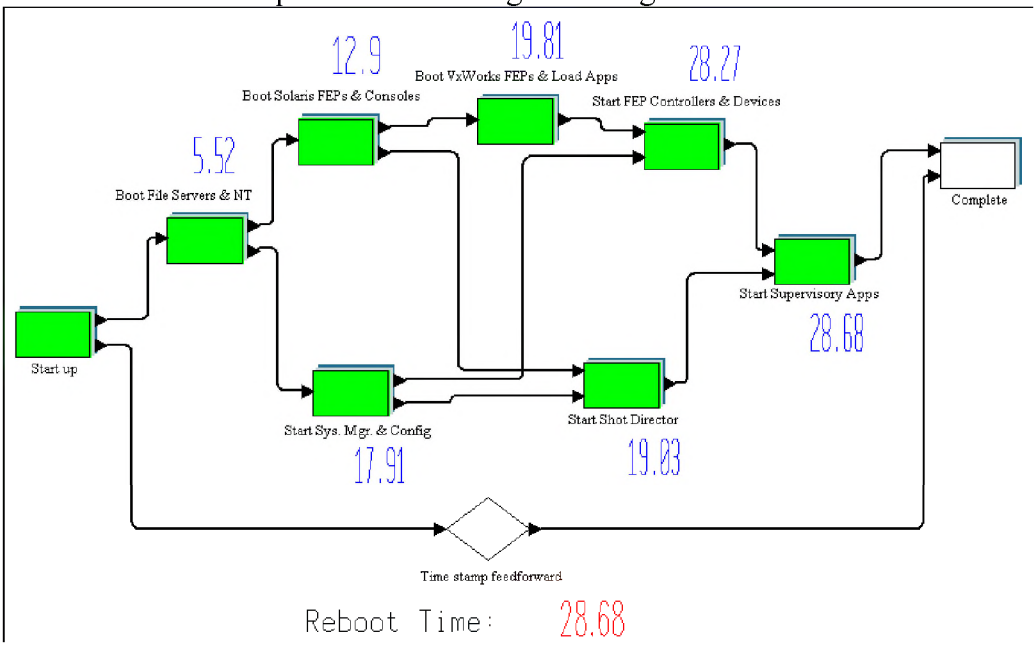


Figure 14. Simprocess discrete-event model for starting all processors in the NIF example configuration.

The simulation passes control system events (e.g., requesting CORBA addresses) through a sequence of activities (e.g. retrieve the operating system from disk, download the

application, etc.). Each activity has among its other attributes, a delay. This delay defines how long it takes each subprocess to finish the activity. The sum of these delays from beginning to end is the total time required to complete the process modeled by the simulation. Delays can be constants or random numbers chosen from a statistical distribution. Each rectangular box in the figure represents a set of activities through which the start-up event must pass. The connecting lines indicate process dependencies. Resources (e.g., network capacity and file server capacity) are also modeled in order to determine resource-constrained bottlenecks in the process.

Each time is given in minutes and is the total elapsed time when the subprocess represented by the box completed. The total time to start-up the NIF system is 28.7 minutes for this replication. A series of 100 replications yielded a maximum of 30.0 minutes, minimum of 28.5 minutes, an average of 29.0 minutes, and a standard deviation of 0.5 minutes

As a consequence of poor performance revealed by initial simulations, the architecture had to be adjusted so that all available Solaris workstations would participate as boot servers for the VxWorks FEPs. This was a trade-off in that there is a system administration cost for maintaining many mirrored locations of the boot information. Nevertheless, there are 455 VxWorks FEPs and the simulation showed savings of 25.3 minutes in the system start-up time over maintaining the information in a single server.

The simulation also showed that having each Solaris computer with its own copy of the operating system saves another 7.4 minutes in the system start-up time. This would require an increase in the system administration effort to maintain over 60 copies of the Solaris operating system. If needed, this change would further reduce the start-up time.

A countdown simulation was constructed on the NIF example architecture to assess FEP resource utilization during the 5-minute time frame preceding a target shot. CORBA messages were modeled between control subsystems including the plasma electrode Pockels cell, laser wavefront correction, and power conditioning. Message format, rate and size were obtained from design engineers as inputs to the simulation. Front-end processor and 10 Mbit/s Ethernet latencies were experimentally evaluated and formulas were created to predict communication delays to within ±10% accuracy. The simulation predicts that the control system will be relatively lightly loaded for communication tasks, running from less than 10% resource utilization to about 30% for the wavefront correction system, which must continuously transmit video sensor images between collaborating processors. The remaining resource is of course needed for processing data. Engineers found this type of simulation very useful for visualizing system operation very early in the design cycle.

Another detailed simulation was prepared of NIF's automatic alignment system that modeled the interactions between motor controllers, video digitizers, and image processing computers. This study was particularly interesting and important because of the system size – some 9,000 motorized actuators and 500 video sensors – and the dramatic impact closed-loop alignment has on facility throughput. Simulation results revealed a major defect in the mechanical design of the beam transport system, which was subsequently corrected. The simulation guided design optimization that should lead to acceptable performance, as shown in Figures 15 and 16.

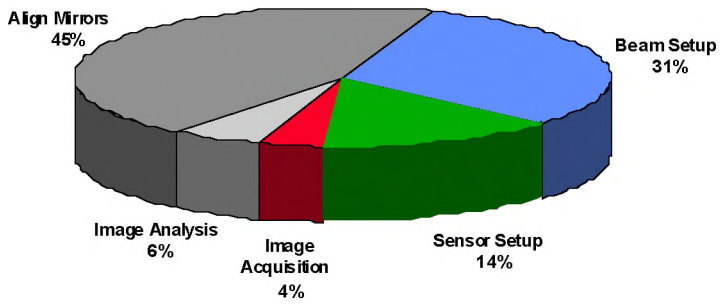


Figure 15. Control system activity during 20-minute NIF alignment is dominated by mechanical actuation times.

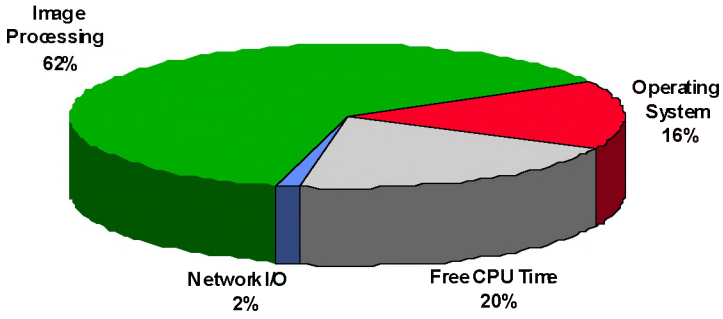


Figure 16. Processor time utilization shows CPU resources do not limit operations.

Note that in this scenario network traffic uses only a small percentage of processor resources even though 48,000 CORBA transactions are required in a 20-minute interval. Two milliseconds of processor time per transaction is easy to achieve. With four processors assigned to automatic alignment in NIF, only 2% of each computational resource is needed for CORBA. The dominant processor utilization is image processing, while the dominant clock time impact is waiting for mechanical actuators to complete their assigned movements. The parallel nature of aligning NIF's 192 laser beams allows multiple processors to divide up the work such that other beams can be processed whenever another is waiting on mechanical hardware.

Suggestions for Continuing Work

The simulation work summarized in this report focused on control system design and architectural issues. Additions and enhancements to control systems have traditionally been expensive and time consuming. However if experience with prototypes and operator feedback is obtained early, the object-oriented nature of the ICCS design should allow modifications to be incorporated at reduced cost during the iterative implementation that leads up to full deployment. More realistic and thorough simulations driven by actual

operator controls are needed to continue providing pre-deployment experience. As control systems are installed, testing and operator training becomes a complex undertaking that can be assisted by simulation techniques.

Two basic techniques for operating controls without hardware are emulation and simulation. Emulation supports testing by faking interactions with the facility equipment at the control point level. In this context, simulation supports operator training by incorporating special software written to predict a more realistic integrated response of the equipment, such as determining sensor readouts under various operating conditions.

Figure 17 illustrates a 3-year timeline starting with this LDRD and leading toward the first deployment of ICCS in NIF.

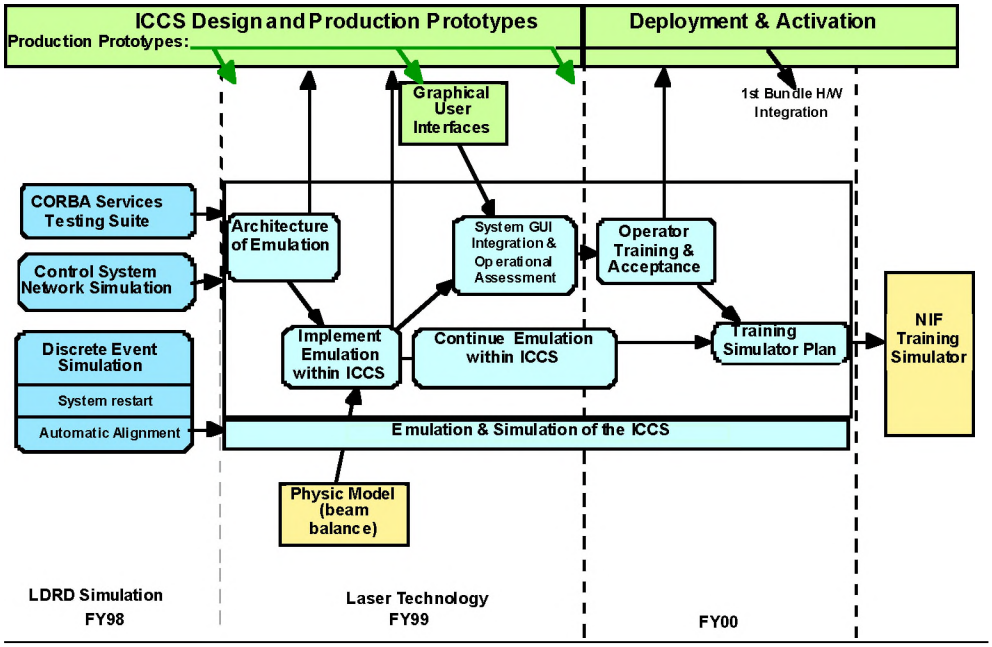


Figure 17 Roadmap to future ICCS simulation development within the NIF project

Future work should determine how an abstract emulation capability fits within the ICCS architecture. Emulation could then be used to test the software at increased "simulated" scale and provide validation data for refining existing discrete-event models. A technique for integrating equipment simulation models into the ICCS architecture using CORBA should also be developed. In addition, an assessment of the integration and operational characteristics of graphical user interface displays applied to training simulators is warranted. The design of the control room, optimal use of computer displays, and interactions with operators should be included in the assessment.

Several topics that enhance ICCS simulation merit further work:

- Where to implement emulation in the distributed object architecture.
- How to configure emulation during start-up of the control system while maintaining confidence that "real" configurations use "real" hardware.
- Methods to reconfigure emulation and simulation on individual devices (or device groups) while the system is running.
- Algorithms and logic to simulate off-normal signals and conditions.
- Continued refinement and validation of models and parameters.

Summary

Building distributed object-oriented control systems, particularly those as large and fully featured as required for NIF, is state of the art. The three coordinated simulation activities comprising this LDRD successfully guided the design of the ICCS as it will be deployed for NIF. Actual performance data collected from the ICCS computer and network testbed verified many simulation parameters.

CORBA testing provided both functional and performance evaluations of products under development to the vendor, thus helping to ensure availability of an off-the-shelf distribution mechanism for the ICCS architecture. Key portions of the NIF network were analyzed by simulating network operation and assessing its performance under worst-case conditions. Workflow models supported redesign of the operational aspect of restarting the ICCS system within the time required, and were used to study a model of controlled equipment for the automatic alignment system. Simulation estimates revealed initially unsatisfactory system performance that led to improvements in the software deployment and modifications to the optical-mechanical hardware.

Additional simulation results also showed that some kinds of trigger signals could be reliably broadcast over the large network, obviating the need for separate trigger distribution hardware. For example, using the NIF network in this capability will save approximately $250K over conventional methods.

The ICCS framework is designed for reuse in future control systems, so the strategies developed under this LDRD can also assist those developers in determining optimal distribution of software among planned computer resources.

Acknowledgements

The principal investigators wish to acknowledge the work of collaborators drawn from several divisions across the Laboratory: C. E. Annese, G. H. Armstrong, R. V. Claybourn, R. R. Johnson, B. M. Kettering, M. G. Miller, and E. A. Stout.

References

1. J. A. Paisner and J. R. Murray "The National Ignition Facility for Inertial Confinement Fusion," 17th IEEE/NPSS Symposium on Fusion Engineering, San Diego, CA, October 6-10, 1997.
2. J. P. Woodruff "A Large Distributed Control System Using Ada in Fusion Research," UCRL-JC-130569, April 21, 1998.
3. F. W. Holloway, "Evaluation of CORBA for Use in Distributed Control Systems," UCRL-ID-133254, February 18, 1999.
4. R. M. Bryant, "ICCS Network Simulation LDRD Project Final Report," UCRL-ID-133085, January 9, 1999.
5. B. M. Kettering, "ICCS Computer System Startup Simulation," UCRL-ID-133148, February 4, 1999.

6. B. M. Kettering, "ICCS Status Monitor Simulation," UCRL-ID-133147, September 18, 1998.

7. C. E. Annese, "ICCS Countdown Status Messages Simulation," UCRL-ID-133242, October 1998.

8. M. G. Miller, C. E. Annese, "Simulation of Beamline Alignment Operations," UCRL-ID-133248, February 2, 1999.

Work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract W-7405-ENG-48.