A Large Distributed Control System Using Ada in Fusion Research

John P. Woodruff
Lawrence Livermore National Laboratory
PO Box 808 MS L-493
Livermore, CA 94551-0808
925.422.4661
woodruff1@llnl.gov

Paul J. Van Arsdall
Lawrence Livermore National Laboratory
PO Box 808 MS L-493
Livermore, CA 94551-0808
925.422.4489
vanarsdall@llnl.gov

1. ABSTRACT

Construction of the National Ignition Facility laser at Lawrence Livermore National Laboratory features a distributed control system that uses object-oriented software engineering techniques. Control of 60,000 devices is effected using a network of some 500 computers. The software is being written in Ada and communicates through CORBA. Software controls are implemented in two layers: individual device controllers and a supervisory layer. The software architecture provides services in the form of frameworks that address issues common to event-driven control systems. Those services are allocated to levels that strictly prescribe their interdependency so the levels are separately reusable. The project has completed its final design review. The delivery of the first increment takes place in October 1998.

Keywords

Distributed control system, object-oriented development, CORBA, application frameworks, levels of abstraction

2. THE NATIONAL IGNITION FACILITY

The $1.2 billion National Ignition Facility (NIF) laser[8][10] is under construction at Lawrence Livermore National Laboratory in California. When completed in 2003, it will be housed in a building the size of a football stadium–704 feet long by 403 feet wide [Figure 1].

This laser is the latest in a series of experimental machines used to study inertial confinement fusion: nuclear fusion reactions produced in a plasma of deuterium and tritium that is compressed by a burst of laser energy[4][15]. The NIF laser, which will be the world's largest high power laser, will deliver 1.8 MegaJoule pulses of optical energy onto a BB-sized fusion fuel capsule in a pulse 25 nanoseconds long. In an experiment, the target will be heated to more than 100,000,000 degrees Celsius and compressed to a density more than 20 times that of lead.

The scientific data that NIF produces in support of the inertial confinement fusion program will support three diverse objectives. As a key component of the US Department of Energy's Stockpile Stewardship and Management Program, the NIF will enable the US to maintain its nuclear device stockpile without resorting to underground testing[11]. It will also collect preliminary data about fusion as an environmentally attractive energy source[7]. The ability to re-create conditions existing inside the sun and stars will significantly impact the science of astrophysics and high energy density physics[5].

NIF's Integrated Computer Control System (ICCS) must integrate more than 60,000 control points to manage 192 laser beamlines. Each shot of the NIF, which generates about 400 Mb of data, will require aligning all components of the laser so that all 192 beams propagate down 600-foot paths through their amplifiers and into the target chamber within 50 microns of their assigned spot on the centimeter-scale target — and so that all beams arrive at the center simultaneously within 30 picoseconds. The alignment process will involve approximately 9500 stepper motors, every one contributing to the position of a beam. This alignment activity is carried out by an automated system that analyzes some 3000 distinct images to command motion of the stepper motors. Only when a control loop fails to stabilize is it necessary for an operator to intervene.

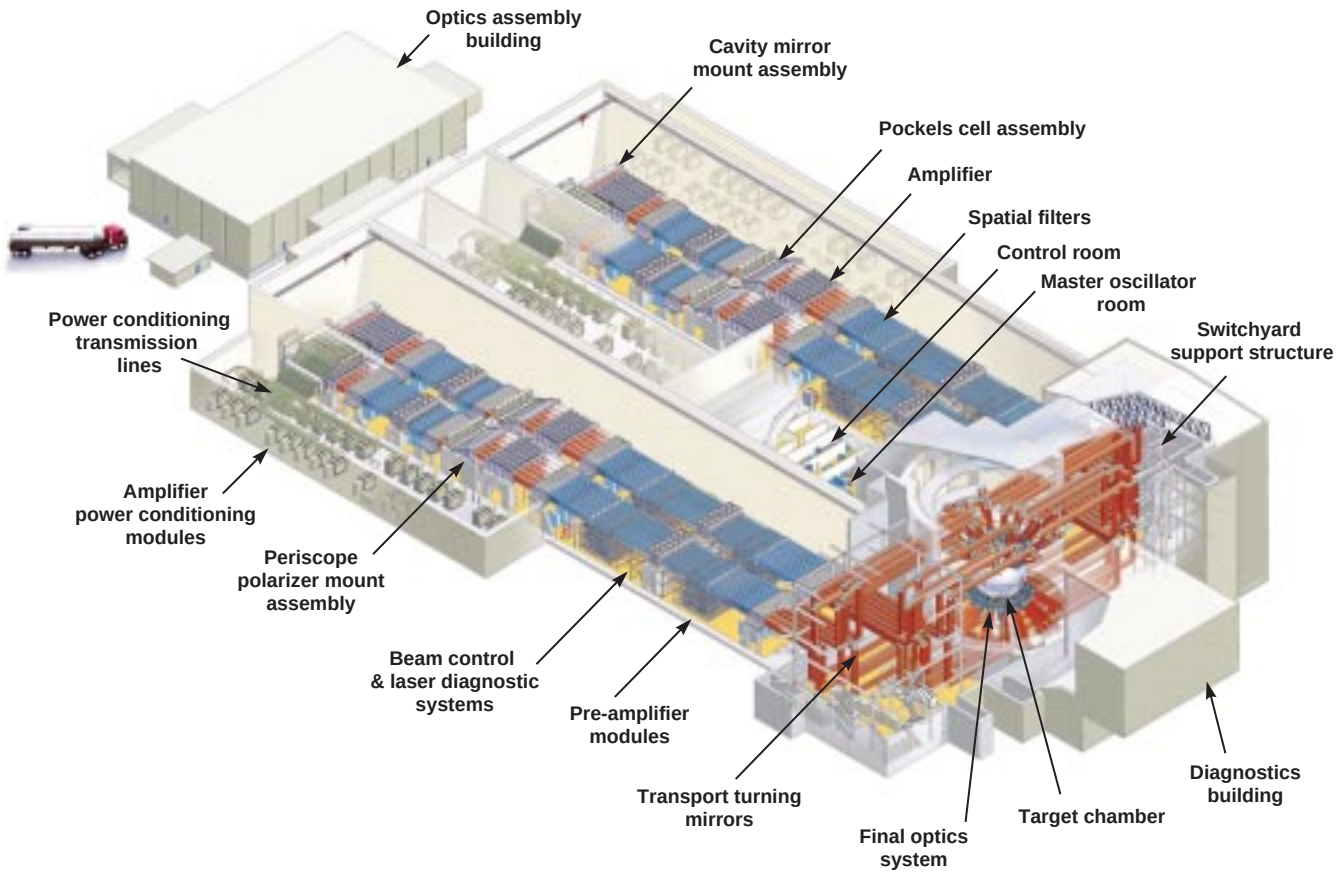

Figure 1: The National Ignition Facility

3. CONTROL SYSTEM REQUIREMENTS

Facilities such as the NIF represent major capital investments that will be operated, maintained, and upgraded for decades. The facility is, in a sense, a factory in which physics experiments are manufactured. Accordingly the computer and control subsystems must be relatively easy to extend as innovative experiments are planned. To assure a 30-year service life, the system must permit periodic replacement with newer technology. The system is being built using Ada on a modern object-oriented software framework that will be extensible and maintainable throughout the facility's life cycle.

The NIF controls must be highly automated and robust. The system will operate continuously around the clock with an allowed downtime of 7.5 days per year for unscheduled maintenance. A brief summary of requirements follows.

- The facility executes a physics shot once per 8 hours;
- The control system coordinates several experimental cycles concurrently to allow different sections of the laser to be used independently;
- The ICCS provides the necessary controls for a dozen operators and technicians to carry out coordinated activities on a machine composed of about 60,000 distinct control points;
- Automatic alignment of the entire laser, controlling actuators in 4000 control loops driven by image analysis, takes place in one hour;
- Data are collected to support operational and maintenance planning;
- Selected images from a population of 600 cameras can be displayed to an operator at ten frames per second;
- The operator's broad view of the entire machine's status is updated with a latency less than ten seconds;
- The system architecture must be flexible enough to absorb significant changes in requirements late in project construction;
- The system must be delivered on time and on budget.

4. LAYERED ARCHITECTURE

The ICCS layered architecture [Figure 2] was devised to address the general problem of providing distributed control for large scientific facilities that do not require real-time capability within the supervisory software. The resultant architecture consists of front-end processors (FEP) coordinated by a supervisory system, and includes commercially available components where possible.

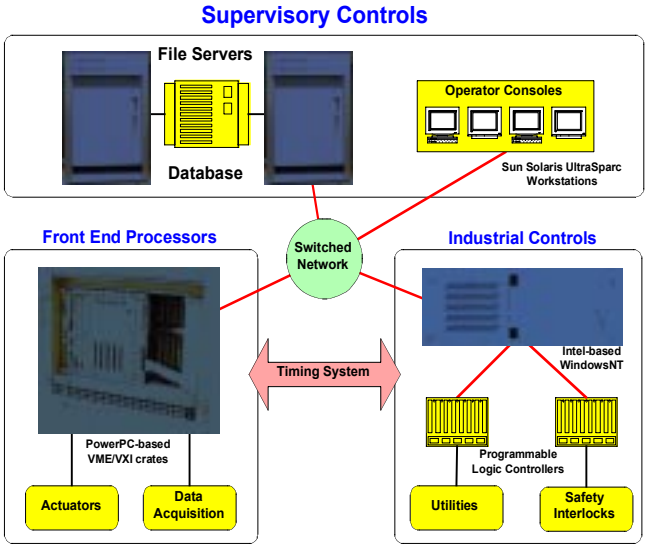

Figure 2: Control System Architecture

4.1 Supervisor layer

Supervisory controls, which are hosted on UNIX workstations housed in control consoles, provide centralized operator controls and status, data archiving, and integration services. Two high-performance servers sharing redundant disk storage provide enhanced performance for database operations, 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 supervisory software system is estimated to comprise 400 KSLOC. This layer provides a human interface in the form of operator displays, data retrieval, and processing that is used to coordinate control functions across laser and target area equipment. The ICCS supervisor is partitioned into eight cohesive subsystems, each of which controls a primary NIF subsystem such as beam control or power conditioning. The supervisory software is responsible for duties ranging from configuration and control sequencing to data processing and archival.

Interoperability among computers and operating systems is addressed by leveraging the international standard Common Object Request Broker Architecture (CORBA)[9]. CORBA is used to implement the communication within and between the supervisory and FEP layers and provides plug-in software extensibility for attaching control points and other software services [Figure 3].

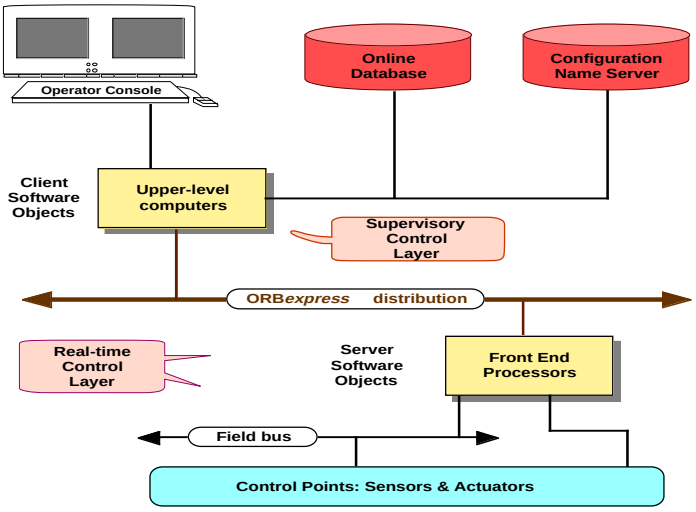

Figure 3: The Supervisor and the Front-end layer communicate using CORBA

4.2 Front-end layer

Front-end processors implement the distributed control functions of the ICCS by interfacing to the NIF control points. There are eighteen different types of FEP computers – some 450 computers in all – that differ by the devices that they control. These FEP units are constructed from VME/VXI-bus or PCI-bus crates of embedded controllers and interfaces that attach to control points.

The control points are sensors and actuators attached to interface boards plugged into an FEP backplane. In many cases, control points are handled by intelligent components that incorporate embedded controllers operated by small fixed programs. This firmware that runs in the embedded controller does much of the low-level work that would otherwise be allocated to an FEP. Example components implemented in this manner are stepping motor controllers, photodiode detectors, and power supply controls. Laser diagnostic sensors attach to FEP units by utilizing low-cost field bus microcontrollers.

The FEP software performs sequencing, instrumentation control, data acquisition and data reduction on control points that are collocated. The software required for all the FEP's is estimated to comprise 100 KSLOC. The architecture being described in this paper provides a standard way for FEP units to be integrated with the supervisory system by providing a common distribution mechanism coupled with software patterns for hardware configuration, command, and status monitoring functions.

Some real-time control is inevitably necessary. Functions requiring real-time implementation are allocated to software within a single FEP or to an embedded controller, so communication over the local area network is not obligated to meet hard-deadline schedules. Examples of real-time functionality include control loops that require deterministic response and diagnostic instruments that are triggered by an integrated timing system within two seconds of the laser shot.

A distinct segment of the control system contains industrial controls for which good commercial solutions already exist that can be integrated into the framework. This segment is comprised of a network of programmable logic controllers in the industrial control subsystem for safety interlocks and thermal, gas, or vacuum utility controls.

Potentially hazardous equipment is permitted to operate only when conditions are safe. Safety interlocks function autonomously from the rest of ICCS to ensure safety without dependency on the rest of the control system.

5 SOFTWARE DEVELOPMENT STRATEGY

Risks to the control system infrastructure are mitigated in the ICCS design by incorporating modularity, segmentation, and open-systems standards so that components and subsystems can be replaced at designated interface points if necessary. Software development of an expected 500,000 lines of code is carried out under an integrated software engineering process that covers the entire life cycle of design, implementation, and maintenance.

This strategy is being realized by building object-oriented software in Ada.

The ICCS employs Ada, CORBA, and object-oriented techniques to enhance the openness of the architecture and portability of the software. Ada was selected based on the favorable experience of the engineering team when the Nova laser software was constructed, then refitted, a decade in the past. Introduction of Ada 95 object-oriented techniques provide increased robustness through the ability to build shared abstractions. C++ is supported for the production of graphical user interfaces and the integration of commercial software.

The software development is managed under a software quality assurance plan that covers the entire life cycle of the software design, production, and maintenance. Central to the development are documentation standards traceable to IEEE standards for software requirement specifications[1] and software design descriptions[2]. Requirement specifications are formally controlled documents that define the contract between software engineer and the engineers who are designing NIF equipment. The design description is a narrative document that explains the object-oriented model and contains other information necessary for implementation. These documents are essential to the long-term maintainability of the software in view of periodic software upgrades and staffing turnover expected during the 30-year life of the NIF.

5.1 CORBA

Past architectural approaches to distributed controls have relied on the technique of building large application programming interface (API) libraries to give applications access to functions implemented throughout the architecture. 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 in a client-server architecture 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 worldwide web. The best way to think of CORBA is as the universal "software bus" [Figure 4]. CORBA is a series of sophisticated, but standard sockets into which software objects can "plug and play" to interoperate with one another. Even when made by different vendors at different times, the object interfaces are standard enough to coexist and interoperate. By design, CORBA objects can interact across different languages, operating systems, and networks.

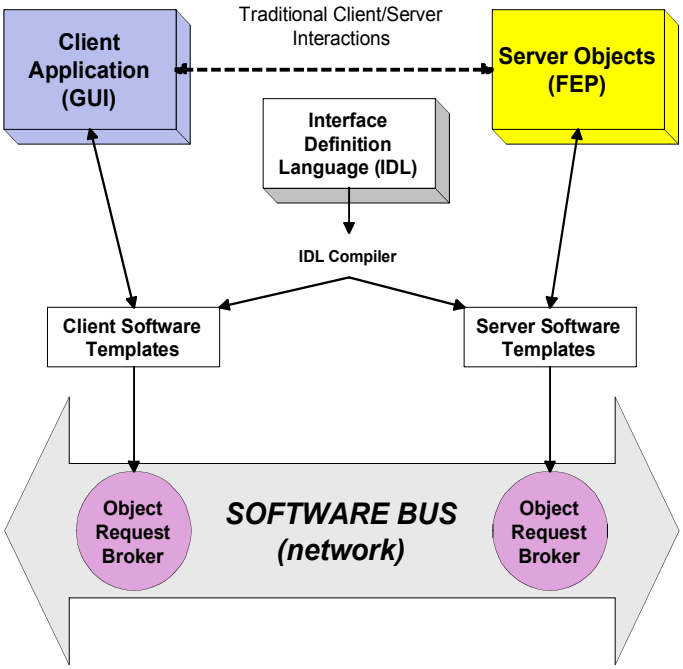

Figure 4: CORBA implements a software bus

5.2 Development Environment

Software engineering tools that have been proven capable of executing large object-oriented projects are being used to carry out the plan. The CASE tools selected include the Rational Rose modeling tool, and the Rational Apex Ada compiler and configuration management tool. Objective Interface System's ORBexpress CORBA distribution middleware and the Oracle 8 database management system both contribute to the planned system, as does the Builder Xcessory X-Windows based graphic interface construction tool.

The software engineering process uses models expressed in the Unified Modeling Language (UML) notation[12][14]. Detailed requirements expressed as use cases are analyzed by developers and result in classes being defined to implement the responsibilities of the software. The object-oriented design is captured in the Rose design tool using UML to maintain schematic drawings of the software architecture.

The Rose tool is used to model the interfaces and interactions between major software entities. Rose automatically generates Ada code specifications corresponding to the class interface and type definitions. The developer fills in the detailed coding necessary to implement each class [Figure 5].

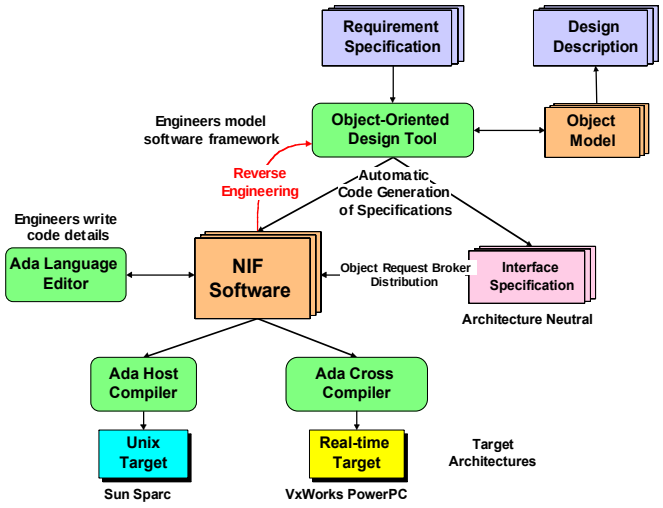

Figure 5: Modeling and program transformation tools manage source text

For classes that are distributed, Rose generates Interface Definition Language (IDL), which is passed through the IDL compiler to generate Ada skeleton code. The IDL compiler generates two families of packages for each IDL interface. One package defines the client section. The body of this package implements the proxy, which calls ORB routines to rendezvous with the remote object implementation. The other generated package implements the server side, which receives invocations from any and all clients and carries out the computations that implement the object's primitive procedures. The obligation of the developer is to specify the interface for every distributed object class and to implement the server-side package body for every primitive operation. The code in the client that makes use of CORBA objects is written as if the server was locally available and directly callable – CORBA takes care of all the rest.

Ada source code can be compiled for a variety of target processors and operating systems. Current development is self-hosted to Solaris on Sparc or cross-compiled for VxWorks on PowerPC. The models, sources, binaries, and run-time images are version-controlled by the Apex configuration management system, which allows the frameworks and applications to be independently developed by different engineers, each having a protected view of the other components.

5.3 Incremental Development

Development is guided by an iterative approach to software construction[3]. This technique is believed to be effective for projects whose requirements are not fully known until late in the project development.

At the earliest stages of the project, decisions were made based on exploratory programming exercises. The team examined the risks posed by such untested techniques as model-driven object-oriented development, Ada 95, and CORBA communication by designing and testing very small subsets of the system. Five iterations are planned prior to the first facility deployment of the ICCS software. Each new release will follow a plan aimed at addressing the greatest risks to the architecture while increasing the functionality delivered to the project.

Early demonstrations have already confirmed the basic architecture and we are now constructing the first of three prototype releases prior to delivery to NIF. The first release, which is being built during the summer of 1998, will deliver vertical slices of all applications in order to exercise the ICCS framework. Each subsystem, including both supervisors and FEP's, will endeavor to demonstrate just one of the several planned shared frameworks. The pulse power conditioning application will exhibit the shot countdown operation, automatic alignment will demonstrate the customizability provided by sequence scripts, and so on. Subsequent phases will be released to incorporate additional subsystem integration and automation during 1999, leading toward first deployment in the facility in the year 2000, when the first 8 of the 192 beams will be operated under ICCS control.

6. LEVELS OF ABSTRACTION

In a large software system, coupling between components can be managed by dividing the system into subsystems that are organized by levels of abstraction, and then enforcing rules for dependencies between subsystems.

When Ada packages are allocated into a collection of subsystems, the discipline of layers of abstraction prescribes that every package within an upper level subsystem may depend only on packages that are allocated to a specified set of lower level subsystems. The upper subsystem is said to "import" the lower subsystem. Cycles in the graph of imports are forbidden but within a subsystem, packages may mutually depend on each other to the extent allowed by Ada semantics.

There are two significant benefits to organizing Ada packages into leveled subsystems. The effect of changes in a package specification is limited to the packages contained in subsystems that import the changed package. Therefore a system's stability can be seen more readily as development proceeds, and work that is done on upper level subsystems cannot affect the lower levels. After the specifications in the lower levels stabilize, concern about system stability focuses on the higher levels.

The other benefit to organizing dependencies into levels comes when a team of developers works in a configuration-managed environment. Each developer or sub-group can import a controlled "view" (as provided by the Apex development tool) of lower level subsystems, and thus be insulated from ongoing work in the lower level. Workers on low-level packages can enhance and test their product, and release stable subsystems for upper level work according to a managed schedule; developers who import these lower levels can choose when to accept the newer release according to the status of their own work.

6.1 Levels in the ICCS Supervisory Layer

ICCS supervisory software has been structured following this principle. Some interesting issues arise when applying this principle in an architecture based on object-oriented programming because abstractions are extended by deriving new types and those derivations may cross subsystem boundaries. ICCS has devised some subsystem structuring heuristics that led to an architecture that is evidently reusable.

6.1.1 Lowest Level

At the lowest level ("Support") the COTS products for database management, GUI tools, CORBA libraries and operating system interfaces are installed; subsystems in every other level make free use of the services of the lowest level. This level holds the most widely reusable components since there is no control system specific information contained in these libraries.

6.1.2 Highest Level

The highest level ("ICCS Programs") contains those packages and library procedures needed to link the operational programs. All the executable processes that need to be installed on the target system are represented in this level. These include the front-end processor server programs (copies of which are allocated to several different processors), the supervisor processes at the "top of the food chain," and the programs that provide system services and are allocated to the central file server computers.

A configuration audit that accounts for some Computer System Configuration Item (CSCI) is performed starting at a subsystem in this highest level and constructing the closure of a compilation unit found there.

Graphic user interface processes are constructed as main programs that interact with supervisor client main programs through CORBA. The current technology for GUI's uses X-Windows libraries and COTS tools for building interfaces. The selected tools build programs that contain both C++ and Ada. These GUI processes are made into as thin as possible a layer so that they can evolve as either interface technology or styles of use dictate.

The eighteen different front-end processor programs are all built as instances of a single generic procedure. This generic procedure instantiation is located in the highest level, but its specification, as well as its actual parameters, are provided by types and their primitive procedures defined in intermediate levels of abstraction.

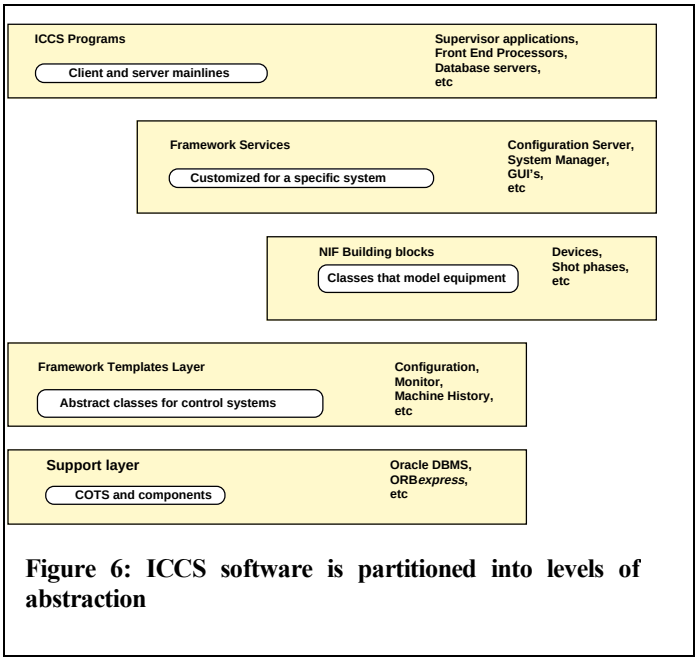

Figure 6: ICCS software is partitioned into levels of abstraction

6.1.3 Type Extension Splits Intermediate Levels

ICCS software is constructed using Ada's facilities for programming by extension: namely the ability to declare abstract base types and to extend those types into an inheritance hierarchy. Applying this facility has caused ICCS packages to be allocated into subsystems at three levels called Framework Templates, NIF Building Blocks, and Framework Services [Figure 6]. The subsystems in these three levels provide a low level of reusable templates, an intermediate level of concrete components, and an upper level that instantiates the templates into services that act on the components.

Subsystems in the Framework Templates level provide abstractions that arise from a domain analysis of control systems in experimental facilities. Control systems operate on "devices," so ICCS defines a class rooted at an abstract type to represent devices. This base type declares properties to be shared by all the control points implemented in FEP's: they possess references allowing distributed access via CORBA, they are initialized with data from a central datastore, and they can be reserved to assure exclusive operation by a single client. Facilities for writing and retrieving maintenance records about devices and for monitoring and publishing their status are in this level as well. However these facilities are incomplete (in the Templates level) since their services are defined in terms of the abstract type. Therefore the services could be reused by a different project where the domain analysis shows similar requirements.

The NIF Building Block level contains an inheritance hierarchy that extends the abstraction of the Device. These extensions implement all the diverse kinds of procedures that real physical devices – motors, power supplies, transient digitizers, and the like – provide for their users. The tactics of inheritance and aggregation are both used to define objects in this level. Several different kinds of commercially available stepping motors are modeled by subtypes of the motor class. Multi-axis motorized actuators used for coordinated motion of optical turning mirrors are built in software by aggregating motors and limit switches into a composite device.

These extensions reify the interfaces that were needed to fulfill the domain analysis. Therefore a motor device can report its present position to the monitoring framework, and the framework publishes that status. Similarly a power supply device can report how many faults have been detected, and this report becomes part of its maintenance record.

Numerous subsystems are defined in the NIF Building Block level to implement numerous specific details of the facility. Devices are the most conspicuous of these since they model the physical constituents of the laser. Additional building blocks include a definition of the shot phases that are enacted by an abstract state machine when the shot director orchestrates a coordinated experiment, and data structures used to record the outcome of physics experiments performed by the facility.

Classes that are extended in the building block layer are usually implemented as CORBA servers. These classes are declared in IDL and translated into a client-side and a server-side Ada package. Attempts are made to limit the coupling between types in this layer so that their services have little or no "policy" contained within them and thus can reliably be invoked for a variety of services. These concrete devices belong in the NIF Building Blocks level because they implement specific functions on specific hardware.

The complete set of control system functions is built on the levels above the Building Blocks so the services remain available when the device class is extended. Framework Services, the uppermost of the three layers, elaborates the services promised by the Templates layer.

The concrete packages that are defined here are extensible because they use the polymorphic types in the building block layer. And these packages, once extended, are the components from which the ICCS Programs in the highest level subsystems are built.

Operations that were specified as calls to the device abstract type are realized as dispatching calls into an appropriate subtype that has been provided by Building Blocks. The policies omitted from the lower layers are provided here. For example, framework services layer subsystems specify the frequency and precision for status reports, and actual physical maintenance records are stored.

It is the intention of the ICCS architecture to make Framework Services available for control of different scientific facilities whose requirements resemble NIF by constructing customized Main Programs at the very top of the abstraction tower. Such customizations might be generated by inserting appropriate Device subclasses with appropriate primitive operations and regenerating the Services for the new installation. At the very least, the strategy will serve the growth of NIF as innovative experimental equipment is introduced into the facility.

6.1.4 Instantiating the Generic FEP

The eighteen different front-end processors differ from each other only by the selection of devices that each implements. Because most contain some hardware controllers that are specific to the controlled devices, and because the platforms are diskless single board computers with limited memory, only the particular device subclasses needed in each FEP instance are loaded into the programs.

The method for configuring the suite of particular devices for each FEP is carried out using packages in the Building Blocks and the Framework Services levels. Objects that are distributed over CORBA are created using a variant of the Factory pattern[6]. For each distinct subtype of Device there is a corresponding subtype of the abstract type Config_Device. Just as with Device, Config_Device has its abstract definition in Framework Templates level and its particular subtypes in the Building Blocks level. The Config_Device object defines a method for using a factory – a non-dispatching call that depends on a reference parameter naming the specific installed factory for the FEP being built. Once the device is created, Config_Device then extracts the device's initial state from the datastore and calls the Initialize method on the device object.

A service defined within the (lower) Framework Templates level makes a dispatching call to the abstract Config_Device. However, the package that invokes this service is defined in the (upper) Framework Services level, so that the call dispatches to the proper member of the Config_Device class.

These operations take place strictly inside a process called Device_Configuration that runs within the central computer facility. It is the operations within the Config_Device that invoke remote servers: first the factory then the initialization of the created object.

So the feature that actually distinguishes one instance of FEP from another is the Factory, which is tailored to fabricate only those subtypes of Device that are allocated onto that platform. Therefore there exist eighteen different subtypes of Factory. Each inherits from the abstract Factory type defined in the Framework Template level, and each is dependent on a context that incorporates exactly the population of Device subtypes allocated to the FEP instance. The chosen Factory package is provided as a generic actual parameter to the generic FEP procedure and the context closure of the factory provides the intended functionality.

6.1.5 Separation of Client from Server

Subsystems between the lowest and the highest level are arranged to exploit the needs of type derivation and to incorporate several variations brought on by the CORBA distribution patterns. Classes that are implemented on CORBA occur at several of the different levels, so the pattern of IDL-generated packages, with proxy and implementation sections, is widespread.

When a software object is constructed using CORBA, no presumption is made that the object's implementation must be remote from the proxy. Therefore in the usual case the body of the proxy package depends on the implementation package, in case the particular object instance is local to the client. This is an appropriate presumption in most cases because the location of objects defined by IDL is intended to be transparent to the client.

However when the particular IDL class is a device which can only be implemented on an FEP platform equipped with special hardware, it makes no sense for the implementation of the primitive operations to be included in the proxy body, and in fact it may be impossible to link these operations if the client mainline and the server mainline must run under different operating systems.

It is appropriate to generate Ada packages for IDL interfaces in two different conditions: one proxy that can invoke its implementation directly (within its own process) and an alternate proxy that does not depend on its implementation. It is this alternate proxy that is included into the Device_Configuration process where Config_Devices are installed because that process on the central computer facility cannot possibly have local access to the implementation of the Devices it creates.

The two different proxies have exactly the same specification but differ in their dependencies, and thus are allocated to the same subsystem but to different views as defined by the configuration management system.

The ability for these views to share specifications is afforded by the Apex "history" facility.

7. SOFTWARE FRAMEWORKS

The ICCS supervisory software framework is the collection of collaborating abstractions that are used to construct the application software. Frameworks reduce the amount of coding necessary by providing pre-built components that can be extended to accommodate specific additional requirements. Engineers specialize the framework for each application to handle different kinds of control points, controllers, user interfaces, and functionality. The framework concept enables the cost-effective construction of the NIF software and provides the basis for long-term maintainability and upgrades.

The ICCS supervisory software framework delivers prebuilt components that are extended to accommodate specific additional requirements in the construction of the application software. The framework promotes code reuse by providing a standard model and interconnecting backplane that is shared from one application to the next. The following discussion introduces the framework components that form the basis of the ICCS software.

Configuration – defines the naming convention and manages the static data for the hardware control points that are accessible to the ICCS. Configuration provides a taxonomic system that is used as the key by which clients locate devices (and other software services) on the CORBA bus. 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. Calibration data for sensors, setpoints for alignment devices, and I/O channels used by devices on 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. Using the data and methods stored in the configuration database, the object factory creates, initializes, and determines the CORBA reference for each device and controller object in the FEP.

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 target processor architecture, backplane, I/O boards, device drivers, and field bus support. The FEP application developer extends the base classes to incorporate specific functionality and state machine controls.

System Manager – provides services essential for the integrated management of the ICCS network of hundreds of computers. This component ensures necessary processes and computers are operating and it provides failover services for processes that terminate during operation.

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

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 and writes them to appropriate persistent stores. Interested observers use DBMS retrieval techniques to update GUI windows on the screens of operators' consoles or produce historic audit trails of operations.

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

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 create and edit sequences by selecting icons that represent control constructs, Boolean functions, and user-supplied methods from a visual programming palette. The icons are then interconnected to program the sequence and any Boolean conditions or method arguments needed are defined to complete the sequence script.

Graphical User Interface – enables all human interaction with the ICCS, via graphical user interfaces displayed upon control room consoles or on X Terminals distributed throughout the facility. 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 common graphical elements for beam selection, laser map, status summary, and countdown clock.

Reservation – manages access to devices by giving one client exclusive rights to control or otherwise alter the device. 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".

Machine History – gathers information about the performance during operation of the NIF for analysis in order to improve efficiency and reliability. Examples of such information are installation and service of components, abnormal conditions, operating service time or usage count, periodic readings of sensors, and alignment reference images.

Shot Data Archive – collects the 400-Mb data generated by a shot, makes it immediately available for "quick look" analysis, and delivers it to an archive. The framework contains a server working in collaboration with the system manager to assure that requested shot data are delivered to a disk staging area. The archive server is responsible for building a table of contents file and then forwarding the table and all data files to the archive. The ICCS is not responsible for the permanent storage or in-depth study of shot data. The long-term study of experimental results is allocated to the scientific programs that utilize the Facility.

8. MEASUREMENTS THAT CONFIRM FEASIBILITY

Two cycles of exploratory programming were carried out before the project Final Design Review. At the completion of these preliminary iterations, approximately 5% of the estimated lines of code had been prototyped and two substantial development risks had been addressed. Object oriented techniques had been used to construct representative parts of ICCS in Ada, and these parts, as well as numerous standalone tests, had confirmed the ability of CORBA to support ICCS performance requirements.

The architecture was confirmed by the prototype. Several distinct kinds of devices were defined and elaborated by the configuration framework, including multi-axis gimbals implemented using stepper motors, photodiodes for laser energy measurement, and precision timing equipment. These device tests relied on data transmitted from an Oracle database into remote FEP's that were instantiated from the Generic FEP framework. Software infrastructures for system start-up and process management, for audit trail generation, and for database query were examined. The tests also demonstrated distributed control and monitoring provided by the architecture. NIF's automatic alignment design, which uses image analysis and closed-loop control to align optics without human intervention, was confirmed in an optical testbed.

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. Object request brokers have

been reported[13] to perform about three times slower than point-to-point communication schemes (e.g. sockets). For this reason, we established a significant testing capability to predict the operational performance of multi-threaded CORBA (ORBexpress by Objective Interface Systems) under various deployment schemes.

Results for different sizes of message streams are shown [Figure 7] for transactions between two UltraSparc machines (one was an Enterprise 3000 server and the other a 3D Creator workstation, both with 300 MHz processors). Measurements of CORBA on PowerPC/VxWorks are in progress, but performance is expected to be similar, as CPU performance is generally similar. Message rates for 100 Mb/s Ethernet are shown along with client and server CPU and network bandwidth utilization. An additional plot is shown for 10 Mb/s Ethernet, which is used to attach many of the FEP's.

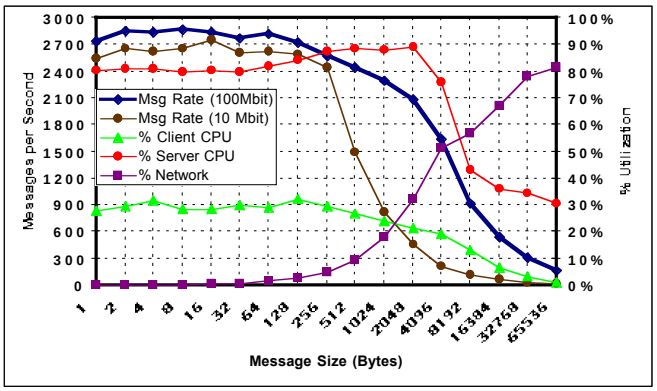

Figure 7: Performance results for CORBA

We determined that most control system transactions utilize on the order of 100 byte messages. For this case, CORBA can transact at about 2700 messages per second while utilizing 80% of the client CPU and 30% of the server CPU. The reason for the wide difference in CPU utilization between client and server is not fully understood and is under investigation. For small message sizes, the CPU is the limiting resource, as the network is not heavily utilized (note that other computers are using the remaining network bandwidth). However, as messages become larger (e.g., during post-shot data retrieval) the network becomes the limiting factor. In the ICCS design, we have partitioned our subsystems such that the message rate design point will average about 500 control transactions per second. This approach provides a fivefold capacity margin to accommodate episodic bursts of message activity that are to be expected occasionally in an event driven architecture. The performance of the ICCS deployment is estimated by measuring software prototypes on our distributed computer testbed and scaled to the NIF operating regime by discrete event simulation techniques.

Predeployment testing will continue through 1999, including the delivery of three increments of production prototype to an independent testing organization for preliminary reliability estimation. Nearly all control functions will be deployed to control the first "bundle" of eight beams in early 2000, and experiments on that laser will be carried out late that year. Project completion will occur in 2003.

9. ACKNOWLEDGEMENTS

We acknowledge the contributions of our colleagues without whose efforts this work would not be possible: G. Armstrong, R. Bettenhausen, R. Bryant, R. Carey, R. Claybourn, T. Dahlgren, F. Deadrick, R. Demaret, C. Estes, K. Fong, M. Gorvad, F. Holloway, J. Jones, C. Karlsen, R. Kyker, L. Lagin, G. Larkin, G. Michalak, P. McKay (Sandia National Laboratory, Albuquerque NM), M. Miller, V. Miller Kamm, C. Reynolds, R. Reed, R. A. Saroyan, W. Schaefer, J. Spann, E. Stout, W. Tapley, L. Van Atta, and S. West.

10. DISCLAIMER

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

11. REFERENCES

[1] ANSI/IEEE Std 830-1984, IEEE Guide to Software Requirements Specification

[2] ANSI/IEEE Standard 1016-1987, IEEE Recommended Practice for Software Design Descriptions.

[3] B.W.Boehm, "A Spiral Model of Software Development and Enhancement," IEEE Computer, May 1988, p 61.

[4] E. Michael Campbell, Neil C. Holmes, Steve B. Libby, Bruce A. Remington, and Edward Teller "The Evolution of High-Energy-Density Physics: from Nuclear Testing to the Superlasers" Laser and Particle Beams 1997, vol. 15, no. 4, pp 607-626.

[5] Keay Davidson "From Swords to Supernovae," Sky and Telescope, November 1997, p 36.

[6] E. Gamma, R. Helm, R. Johnson, J. Vlissides "Design Patterns: Elements of Reusable Object-Oriented Software," Addison-Wesley 1995.

[7] William J. Hogan, Roger O. Bangerter, and Charles P. Verdon "The Fire Next Time," The Sciences, Vol. 36 No. 5, September/October 1996, p 20.

[8] National Ignition Facility webpage http://lasers.llnl.gov/lasers/nif.html

[9] Object Management Group "The Common Object Request Broker: Architecture and Specification" John Wiley and Sons 1995.

[10] 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.

[11] Ted Perry and Bruce Remington "Nova Laser Experiments and Stockpile Stewardship," Science and Technology Review, September 1997 http://www.llnl.gov/str/Remington.html

[12] James Rumbaugh, Ivar Jacobson, and Grady Booch "Unified Modeling Language Reference Manual" Addison Wesley, expected 1998.

[13] Schmidt, D.C., Gokhale, A., Harrison, T.H., Parulkar, G., "A High-Performance End System Architecture for Real-Time CORBA," IEEE Communications Magazine Vol 14, Num 2, February 1997, pp72-77.

[14] "UML 1.1 Specification" http://www.rational.com/-uml/

[15] "Veil of Secrecy is Lifted From Parts of Livermore's Laser Fusion Program," Physics Today, September 1994, p 17.