JAVA TOOL FRAMEWORK FOR AUTOMATION OF HARDWARE COMMISSIONING AND MAINTENANCE PROCEDURES*

J. Ho#, J. Fisher, J. Gordon, L. Lagin, S. West
LLNL, Livermore, CA 94551, U.S.A.

Abstract

The National Ignition Facility (NIF) is a 192-beam laser system designed to study high energy density physics. Each beam line contains a variety of line replaceable units (LRUs) that contain optics, stepping motors, sensors and other devices to control and diagnose the laser. During commissioning and subsequent maintenance of the laser, LRUs undergo a qualification process using the Integrated Computer Control System (ICCS) to verify and calibrate the equipment. The commissioning processes are both repetitive and tedious when we use remote manual computer controls, making them ideal candidates for software automation. Maintenance and Commissioning Tool (MCT) software was developed to improve the efficiency of the qualification process. The tools are implemented in Java, leveraging ICCS services and CORBA to communicate with the control devices. The framework provides easy-to-use mechanisms for handling configuration data, task execution, task progress reporting, and generation of commissioning test reports. The tool framework design and application examples will be discussed.

INTRODUCTION

NIF is the largest and most energetic laser system in the world designed to study high energy density physics [1]. The stadium-sized complex contains 192 laser beams. The beam lines contains a total 6,200 LRUs, which house a combination of optics, motors, sensors, video cameras and other devices needed to align, control and diagnose the laser. Each LRU must undergo qualification procedures to verify and calibrate the hardware necessary to commission and maintain the unit.

MCT software automates the qualification process, which improves efficiency of commissioning and subsequent maintenance by directing the control system to carry out complex algorithms with minimal operator intervention. The tool architecture also supports simultaneous LRU qualifications, further improving efficiency. The automated tools are built on top of the ICCS framework [2], which encourages object oriented design, code re-use, and standardization. This establishes a structured environment where the developer may focus on commissioning algorithms rather than the low-level details necessary to integrate the tool into the ICCS software.

*This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.
#joyceho@llnl.gov

A typical qualification procedure entails moving a set of electro-mechanical devices, measuring the beam location on the video sensor, calculating the system characterization, and verifying the results. The process is repeated several times. This labor-intensive process is then applied to another set of devices in the same beam. For example, most electro-mechanical devices exhibit a degree of backlash due to tolerances and normal wear that must be measured to compensate for the effect in the control system. The process to calculate the backlash for a single motorized stage entails moving the motor in both directions, finding the beam centers, calculating the compensation, and then repeating the process a number of times to improve accuracy and assure reliability.

An automated tool eliminates most of the operator's interactions with the control system by encapsulating all the necessary actions within the tool's algorithm. A single operator command directs the tool to perform a complex sequence of moves, measurements, and calculations. A MCT performs commands faster because there is no noticeable delay between sequence steps. Furthermore, an operator can run several instances of the tool to qualify multiple beams simultaneously.

MCT FRAMEWORK DESIGN

The MCT software is implemented in the Java language (version 6.0) and packaged into a standard ICCS software release. The MCT framework is built on top of the ICCS User Interface (UI) framework, which provides NIF-specific logic, connection and display patterns, and services [3]. Several unique UI features distinguish the MCT. Standard ICCS UIs, by design, require operator actions to initiate commands and are generally stateless. MCTs incorporate command capability that can perform several actions or complex algorithms without operator involvement. The MCT framework provides a toolset for flexible configuration data management, thread management for algorithm execution, and status reporting for the MCT UI. The framework facilitates code re-use, object-oriented design, and consistency among all tool instances.

Configuration Data

Provision for adjustable configuration data limits the need to recompile software each time a particular device configuration is added, modified, or deleted. By managing the data outside the code, the tool can be updated within minutes to support a new hardware device. Any data that could potentially change, for instance device location or device specific properties, is stored in a single Extensible Markup Language (XML) formatted file.

The MCT framework parses the list of tools defined in an XML file. The list of tools is separated by subsystems that mirror the NIF laser subsystems. XML elements specify any configuration data shared among the devices within the same subsystem. In addition, a set of companion files such as spreadsheets can be associated with a particular tool. The format of the files is tool specific; the only requirement is that the tool contains logic to parse the file. The benefit of this design is flexibility to tailor configuration files to the individual needs of each tool.

Algorithm Execution

Automating a set of qualification procedures may require implementing several algorithms in an individual tool. Each algorithm is partitioned into one or more tasks, which encourages object-oriented design and code re-use. The MCT framework contains mechanisms for task execution, status propagation, and reporting the results. Figure 1 shows a diagram of the interaction between a tool's UI and the framework components.

Execution management of tasks that initiate tool algorithms is a key framework feature. The execution manager leverages the new Concurrency API in Java 6.0 for handling all necessary tasking, supporting parallel task consumption, stopping of running tasks, and providing thread debugging tools for troubleshooting purposes. These features eliminate the overhead of implementing a separate thread management system within each tool.

The MCT framework publishes algorithm progress information to any subscribed listener, such as the tool UI or other peer tasks, through a publish/subscribe observer pattern. For example, the information could be a completion percentage, an error message, a task action message, or an intermediate data result. By establishing a standard communication protocol to pass information between tasks, the framework promotes a division of labor among the tasks.

Documentation of measurements and calculations is an important requirement for many tools. The framework supplies mechanisms for saving images and data using a standard naming convention. In addition, an Excel-writer task can run in parallel with the algorithms to record data values, formulas, and other information necessary to perform the calculations manually. The spreadsheet format is an intuitive representation of the data that is familiar to operators and allows for further offline analysis.

Common Device Communication

ICCS software adheres to the Common Object Request Broker Architecture (CORBA) standard to communicate with distributed elements such as hardware devices. Within the ICCS framework is a layer to create connections to any device in NIF. A substantial percentage of MCT communication involves connections to motors, cameras, photodiodes, or automatic alignment supervisors [4]. The MCT framework adds another layer on the ICCS connection framework to encapsulate common function calls for these four connections. Benefits include standardized status reporting and error handling across the tools, enhanced code maintainability and minimized code repetition.

APPLICATION EXAMPLES

There are currently twenty-five tools implemented in the MCT software that are divided into five categories based on the general purpose:

- Alignment verification of laser beam position
- Timing optimal subsystem delays
- Device calibration
- Diagnostics to troubleshoot the laser
- Image analysis for beam characterization

Two tools will be highlighted, one from the device calibration category and another from the image analysis category. Algorithm automation as well as how the framework is leveraged will be discussed.

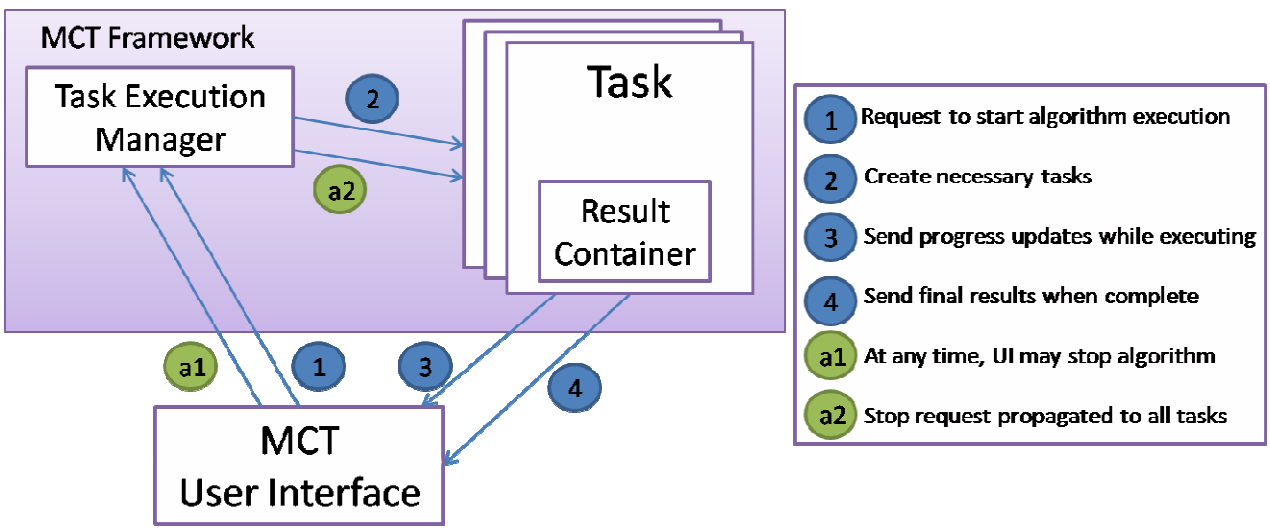

Figure 1: Diagram of the interaction between tool UI and the framework.

Cross-Coupled Motor Tool

An example device calibration tool is the cross-coupled (CC) motor tool. This tool computes a transformation matrix for positioning beam steering gimbals, which are multi-motor devices. The set of motors are coordinated such that the laser beam is either 1) centered within the clear optical aperture or 2) pointed to the next optic. The goal is to perform each operation without affecting the other as viewed on the alignment sensor. The algorithm moves individual motors separately and measures the response on the video sensor. Once all input axes are measured the CC matrix is computed.

Manual controls are inefficient and error prone because the operator must repeat a tedious process many times: specify distance, move motor, acquire sensor image, and locate beam position. The manually-operated procedure requires at least an hour per gimbal to complete the measurements, but the automated tool reduces this time to only 15 minutes.

The tool encapsulates configuration data in a spreadsheet to enumerate different gimbals. The data includes the default distances, tolerances for matrix verification, and the number of gimbal axes. The tool implements five tasks: query the device for the currently stored matrix, acquire the individual motors, measure response position, store new matrix in the device, and verify the new matrix is within tolerance limits. These tasks use framework connections to motors and an automatic alignment supervisor to communicate with the control devices. Progress status and measured values are published to the operator display.

Pinhole Overlap Tool

The pinhole overlap tool (Figure 2) relies on specialized image analysis to quantify the optical system installation. The purpose of the tool is to minimize beam clipping that may occur when the beam travels through pinholes that may not be perfectly aligned. The image algorithm measures beam ellipticity to characterize clipping and determine the pinhole location, while providing visual feedback to the operator for continuous adjustments. The decision to automate this procedure was primarily driven by lack of a consistent vision-based metric to quantify the clipping, as the human eye has difficulty detecting small misalignments. The manual adjustment period was about 45 minutes per beam, but the automated tool decreased the time to 10 minutes.

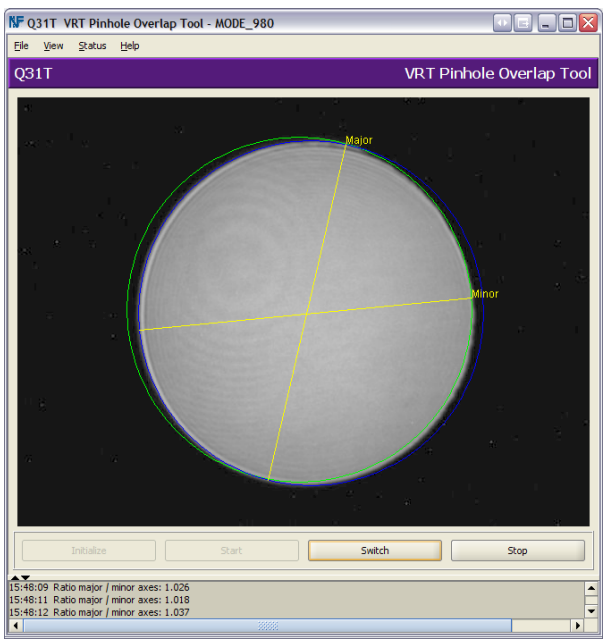

Figure 2: Pinhole overlap tool display.

The MCT framework limited new code development effort to the implementation of the specific image analysis, which condensed the development effort from 6 weeks to 3 weeks. To account for a potential camera location change, a configuration spreadsheet specifies the camera hardware. The common camera connection acquires the image for processing and publishes the image along with the ellipticity and location of the pinholes to the display through features in the tool architecture.

CONCLUSION

The MCT framework provides simple mechanisms for managing configuration data, algorithm execution, and common device communication. Tools built from the framework plug into the distributed control system without impacting normal operations. The framework promotes object-oriented design, code re-use, and standardization across all commissioning tools with a minimum of developer effort. These features support the automation of complex multi-step procedures with low operator intervention, allowing NIF to be commissioned and maintained at higher efficiency.

REFERENCES

[1] P. Van Arsdall, et al., “Status of the National Ignition Facility and Control System,” ICALEPCS 2005, Geneva, Switzerland, October 2005.

[2] R. Carey, et al., “Status of the Use of Large-scale CORBA-distributed Software Framework for NIF Controls,” ICALEPCS 2005, Geneva, Switzerland, October 2005.

[3] J. Fisher, et al., “User Interface Framework for the National Ignition Facility (NIF),” ICALEPCS 2007, Knoxville, October 2007.

[4] K. Wilhelmsen, “Automatic Alignment System for the National Ignition Facility,” ICALEPCS 2007, Knoxville, October 2007.

[5] A.P. Ludwigsen, “Software Engineering Process Used to Develop the National Ignition Facility Integrated Computer Control System,” ICALEPCS 2007, Knoxville, October 2007.