ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEDPL01

IN-PLACE TECHNOLOGY REPLACEMENT OF A 24x7 OPERATIONAL FACILITY: KEY LESSONS LEARNED AND SUCCESS STRATEGIES FROM THE NIF CONTROL SYSTEM MODERNIZATION

M. Fedorov, G. Brunton, C. Estes, B. Fishler, M. Flegel, A. P. Ludwigsen, M. Paul, S. Townsend
Lawrence Livermore National Laboratory, Livermore, USA

Abstract

The National Ignition Facility (NIF) is the world's largest laser system for Inertial Confinement Fusion (ICF) and High Energy Density (HED) experiments. Design of the NIF control system started in the 1990s, incorporating established hardware and software technologies of that era. The architecture of the control system has stood the test of time, successfully scaling up to a full 192 laser beam configuration in 2009, and then transitioning to 24x7 operations and sustaining 400 shots annually since 2016. The control system has grown with NIF to add new major capabilities, such as cryogenic layering, a petawatt-class laser, 3D neutron imaging and others. In parallel, with scaling up and efficiency optimizations, the software had to adapt to changes dictated by the fast-paced computer industry. Some of our originally chosen technologies have become obsolete and replaced by new programming languages, frameworks and paradigms. In this paper, we discuss how the NIF control system has leveraged the strengths of its distributed, crossplatform architecture to successfully modernize "in-place" computing platforms and programming languages without impacting the demanding experiment schedule.

INTRODUCTION

Large experimental physics facilities embody significant investment of societal resources and they are expected to last, gainfully generating scientific knowledge for 20-40 years. Computer industry moves much faster, periodically forcing facilities' control systems into major modernizations to address technology obsolescence, cybersecurity and paradigm shifts in programming technologies.

Continuous pursuit of the experimenters' goals at these facilities may also mean that there will never be an extended downtime for a comprehensive "full rewrite" of the control system. The alternative is an incremental "in-place" upgrade in parallel with scientific operations, stretching the modernization over numerous small windows (2-4 hours) in the facility schedule. By overlapping with other maintenance activities, the "in-place" approach does help to minimize the overall downtime budget. However, each of these numerous upgrades carries a risk of an unexpected behavior change or a system incompatibility. While the planned downtime budget can be frugally negotiated, facilities have zero tolerance for unplanned downtime since it directly impacts the quantity and quality of scientific output.

We explain how we have approached the "in-place" upgrade by carefully inspecting the "pillars" supporting our control system architecture. Some of these pillars had to be removed and replaced, while others stayed and served as pivots which helped our team to propel NIF Integrated Computer Control System (ICCS) towards modern technologies.

To address the unplanned downtime risk, we have expanded our automated testing by adding focused verifications of the fidelity of the software migrations, assuring that new behaviors, timings and exceptions match those of the legacy software. Finally, we have leveraged the data-driven aspect of our architecture to develop a fast and reliable "conversion-reversion" process which assures that we can always undo a migration upgrade and return the facility to normal operations in a predictable time.

EVOLUTION OF ICCS TECHNOLOGIES

Early Days

The NIF control system was designed at the end of the 1990s. Reliability and scalability were the primary concerns for hardware and software architects, which resulted in selection of proven, well established technologies with solid industry support. For the low-level, hardware-facing Front-End-Processors (FEPs) the NIF team selected VME-bus, Motorola PowerPC diskless crates running VxWorks RTOS, Fig.1.

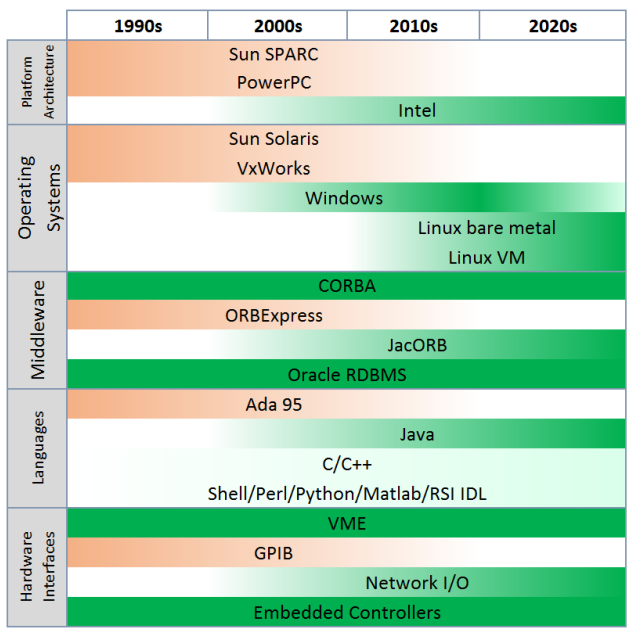

Figure 1: Evolution of ICCS technologies: fading red color indicates legacy technologies on their way to obsolescence. Deepening green illustrates growth of modern alternatives. Solid green highlights enduring pillars of our architecture.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEDPL01

Sun SPARC servers and workstations under the Sun Solaris UNIX OS were selected for supervisory and GUI applications [1].

In the software domain, a similar focus on type safety, modularity and scalability has driven commitment to Object Oriented Design (OOD), Ada 95 programming language and CORBA middleware. To accentuate software reuse and to support the never-ending expansion and modification of the NIF systems, the control system topology and configuration are data-driven, defined by structured database entries. Oracle RDBMS was selected as a storage for configuration data, persistence and history archive.

It is worth noting that all software tools and libraries in the ICCS 1997 design [1] were proprietary, closed source and expensive.

Java Moves In

Early in the prototype system development it became clear that Ada 95 was lacking a healthy ecosystem. Very few libraries were available outside of the core language, which was a severe limitation for a universal and integrated control system. ICCS needed a clean modern user interface, and Microsoft Windows became the preferable platform for the Control Room consoles due to its ease of use and standard office tools. There were no good cross-platform UI toolkits for Ada 95, proprietary or open-source. Tellingly, Oracle discontinued its Ada database binding library.

At the same time, the quickly rising Java programming language was offering consistent cross-platform UIs and solid enterprise-level database driver libraries. This is when ICCS team leveraged CORBA middleware as a cross-language tool, by connecting Java components to Ada using open-source JacORB middleware. The NIF control system became bilingual.

Soon after, Java became the language of choice for all new software development. ICCS software engineers preferred Java because of its clarity, efficient and free development tools and a whole universe of third-party libraries. Most of the new hires did not know Ada 95 and they were pleased not to learn the niche language.

FEP Challenge

By the early 2010s, all ICCS GUIs, and most of the supervisory and shot automation applications were Java. At the same time, our Ada 95 development toolchain became a real burden: obsolete, poorly supported and expensive licensing. Our hardware facing FEP platform of VxWorks 5.4 and PowerPC was obsolete. Oracle had acquired Sun Microsystems, tossing Sun Solaris OS and SPARC architecture towards obsolescence.

At this point, it became clear that we must entirely migrate away from Ada 95, VxWorks and Solaris. We already knew good alternatives to migrate to: Java, Linux and Intel.

Our Front-End-Processors (FEPs) became the last bastion of legacy Ada 95/VxWorks/Solaris platform, lagging because of the historical and technological reasons:

- Most of the FEP software was originally developed in Ada 95 during early phases of the NIF construction, before the ICCS Java stack became available.
- Controls hardware is attached to these FEPs, so the migration scope explodes into full replacement of the computing platform, OS and specialized drivers.
- FEPs are directly controlling powerful energies and expensive hardware, requiring formal re-verification of the machine safety requirements whenever software changes.
- Initially, Java platform was not as predictable for soft real time control applications as VxWorks/Ada 95. These concerns were addressed by newer JVMs, fast multi-core CPUs and garbage collection management.
- There was a loss of the domain expertise for systems designed years ago: many of the software engineers and their electrical, optical, mechanical counterparts have retired or moved on.

It is not surprising that the ICCS team was hesitant to migrate the FEPs, leaving them until later because of the complexity, impact and the risks. To eliminate legacy technologies from ICCS, we had to simultaneously rewrite FEP software in a new language, switch to a new OS, new CPU architecture and new drivers.

While NIF operations were supportive of the long-term sustainability goal, our proposal to make such radical changes to the core systems came when operations were the least willing to introduce any downtimes or risks.

OPERATIONAL ENVIRONMENT

Construction of the NIF had concluded in 2009, ten years ago, when 24x7 operations began with focus on maximizing the scientific output by increasing number of experiments (“shots”), maximizing availability and continuous innovation in laser, target and diagnostic capabilities, Fig. 2.

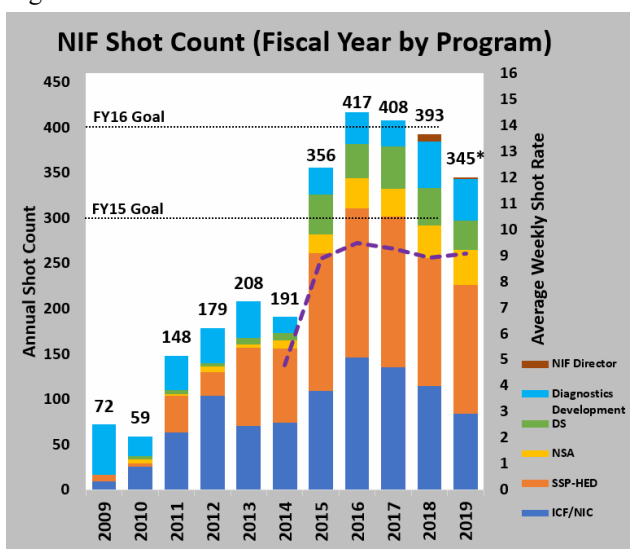

Figure 2: NIF Shot Rate year-to-year. 391 shots are expected in 2019 [2].

The NIF operational schedule is thoroughly managed, with experiment plans extending a year into the future and

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEDPL01

getting highly detailed down to hours and minutes on a week timescale. Within this schedule, a limited time is allocated to the ICCS team to rollout new releases and patches. We deploy 4 major releases per year and 4-6 minor patches. For a major release, ICCS is given 24 hours of NIF time, most of which is needed to perform online testing with real hardware and to exercise a full NIF laser shot. Rollout of patches is much faster, 2-4 hours, and we are normally not given time to exercise a full test shot.

NIF expects that after an ICCS software release or a patch is deployed, the control system software is fully qualified for operations – there is no allowance for debugging or tweaking after the release.

NIF experiments are conducted from the Control Room by a crew of 12 operators, led by a Shot Director. Operators are focused on safety, procedures and schedules. Control Room operators are not software engineers and they are not equipped to debug software during experimental campaigns. Moreover, ICCS software engineers are not qualified as NIF operators and they are not present in the Control Room during shot operations.

NIF's high expectations for availability and reliability dictate "zero tolerance for error":

- the downtime for the software upgrades and testing should be minimal
- the downtime should be predictable, since shot operations must resume normally after the allocated conversion time
- there should be no surprise changes in hardware and system behaviors
- machine safety requirements need to be explicitly re-verified
- there should be no significant change in operator interactions, otherwise procedures will need to be updated and operators re-trained.

Understanding these constraints rising from the need to assure uninterrupted scientific operations was the most formative factor in the development of our FEP Java modernization strategy.

STRATEGIES AND CONSTRAINTS

A control system modernization can be approached in several ways:

- New Generation or a Complete Rewrite, when a new control system is developed from scratch, and the legacy system is abandoned.
- SkunkWorks or a Tiger Team approach, when the new generation software is developed in a backroom while operating and maintaining the legacy system until the new software is ready to switchover.
- In-Place or Piecemeal incremental upgrade, when the new components are compatible with the existing system, they gradually phased in until the entire system is modernized.
- Gateway or a Federation, when the new software is not compatible with the legacy control system, but there is a software gateway which connects the parts. At extreme, this strategy produces a Federation of the control systems, working together, but drastically dissimilar in their protocols, control and data flows.

Each of these strategies has strengths or flexibility in some areas while they are more limiting in others. We have summarized our understanding of these benefits and constraints in Fig. 3. The control system modernization decision should be based on organizational preferences, clear understanding of the alternatives and assessment of operational, budget and workforce risks.

In our situation the modest In-Place upgrade strategy appears to be the only viable option, since minimizing facility downtime and assuring no unplanned outages are clearly the topmost concerns. Our organization budget has been flat for many years, we cannot hire a sizeable Tiger Team to work on the new technology while the core team continues to maintain the legacy system. Additionally, we would be concerned about loss of cohesion when multiple architectural paradigms coexist within the control system. Our team has considered the Gateway solution before and we like its technological flexibility, but we have decided that the added complexity does not justify the benefits.

	New Generation	Skunk Works	In-Place	Gateway
Total Technology Upgrade	++	+	-	+
Minimize Downtime	--	-	+	+
Predictable Downtime	+	--	+	+
Flat Budget	+	--	+	-
Architecture Cohesion	+	-	+	-

Figure 3: Control system upgrade strategies.

In exchange for the flexibility with downtime and budget, the In-Place strategy requires that new modernized software has to be compatible with the rest of the system. The extent of the required compatibility needs to be understood to determine which of ICCS architectural concepts should be preserved across migration and which will be replaced.

ARCHITECTURE OF CHANGE

CORBA-level Interfaces: Preserve

CORBA is the middleware technology which ICCS uses to connect components over the control system network. All ICCS Interface Definition Language (IDL) modules are compiled both to Ada and Java. Carrying over IDL interfaces intact from legacy Ada to new Java software is a vital requirement for achieving smooth In-Place migration.

Fortunately, CORBA has also been of great help supporting the migration, acting as a key modernization "pivoting pillar". Conceptually, in the CORBA OOD paradigm, Ada and Java variants of the software are just alternative poly-

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEDPL01

morphic implementations of the same interface, and therefore they are guaranteed to be fully interchangeable from the client point of view, Fig. 4.

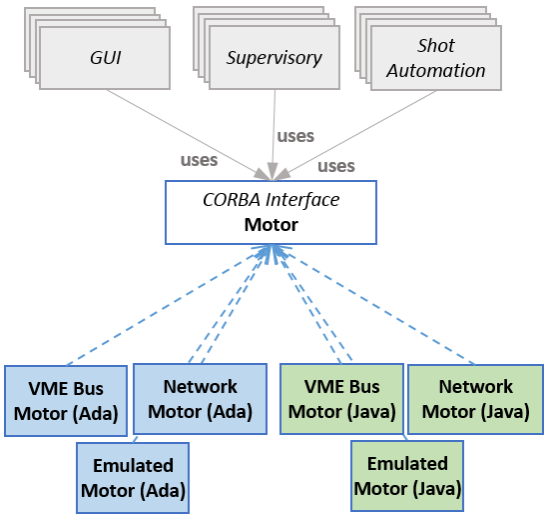

Figure 4: CORBA Interfaces "abstract away" Ada-to-Java transition, assuring client compatibility.

Database Schema: Preserve

Preserving database schemas (see Fig. 5), i.e. the structure of the tables, is not a strict technical requirement. It is entirely plausible that the modernized variant of the software will benefit from defining new datasets for its configuration settings, persistence and archive data.

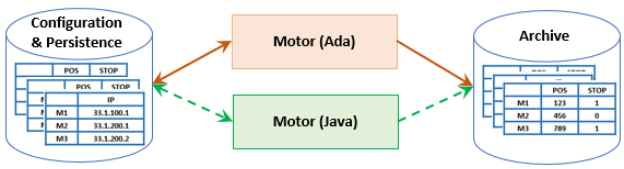

Figure 5: Reusing existing database tables between legacy and new software simplifies conversion-reversion process and it is transparent to facility operations and customers.

However, since the configuration and archive tables are part of a broader interface between the control system and the facility enterprise, maintaining schema consistency is desired to assure that the software modernization remains entirely transparent to facility operators, procedures and processes, external systems.

Additionally, reusing the same tables between legacy Ada and new Java implementations allows for easy switching between Ada/Java software by "repointing" the object identity.

Identity and Naming: Preserve

Every ICCS device is identified by a taxonomical name, or a taxon, consisting of four components**:** Subsystem, Location, Unit, Identifier. At runtime, all access to an ICCS CORBA object is performed via a taxon lookup. Taxons also serve as a database key for configuration, persistence and archive data.

To migrate a device, we just need to associate the device taxon with a Java FEP instead of an Ada FEP. The new Java device assumes all database data from the Ada device, and the rest of the control system will use the Java variant of the device without any additional changes.

Importantly, the change is easily reversible. If a problem is discovered with the newly migrated Java code, the taxon can be reconfigured back to the legacy Ada software. The Ada application will resume operations with up-to-date persistence data left by the Java device.

Implementation Level Design: Replace

In the ICCS control system, CORBA objects are "fat", each representing a non-trivial chunk of functionality. They can be thought of as a facade or microservice object. Internally, the implementation code for a single ICCS CORBA device may consists of dozens of classes and types.

While the goal of the Ada/Java migration is to preserve the external CORBA interface *exactly*, there is no such requirement for the design of the internal implementation. Although our legacy Ada 95 designs were also adherent to OOD principles, there have been significant rethinking of the OO practices and design patterns, with emphasis on composability, immutability and data flows inspired by Functional Programming. We have found that it is not worthwhile to follow the legacy designs even when the detailed documentation exists from the 1990s. Instead, we reimplement external facade interfaces using modern Java design and coding patterns.

Timing: Preserve (Reasonably)

Timing characteristics of object behaviours are important, especially in a control system, and they need to be preserved during the Ada/Java migration. Unfortunately, there is no provision in CORBA IDL to specify the timing aspects of an interface contract. Instead, we are relying on automated component testing to measure Ada call durations and then validate Java timings.

In most controls use cases, the timings are determined by the external hardware, so they are not computationally bound. While the precise reproduction of the call durations is not realistic and unnecessary, it is reasonable to expect that Ada/Java will have similar timing characteristic and any significant discrepancy should be a red flag for the developer.

Tasking, Concurrency, Synchronization: Replace

External timing properties of CORBA objects can be significantly affected by the implementation threading and synchronization details. The FEPs in the control system are multi-threaded, exposed to dozens of unsynchronized network clients and they are expected to respond promptly and predictably to these requests.

Unfortunately, Ada95 and Java have incompatible threading and synchronization primitives, therefore Ada's Tasks, Rendezvous and Conditional Entries have to be reimplemented with Java Threads, Executor Services and Queues.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEDPL01

User Interfaces and Interactions: Preserve

The in-place upgrade requires that user interfaces and interactions do not change at all when an FEP is migrated from Ada to Java. The control room operators should not be surprised and the FEP behaviours should not deviate from the existing facility procedures and documentation. It is common during the in-place migration process that some FEPs of certain type are running Java while the rest still operate with legacy Ada 95 software.

Since ICCS GUIs are CORBA clients, the preservation of the GUI Look and Feel is achieved automatically once we preserve the CORBA interfaces. The overall user behaviour is more complex since it includes operator expectations for business logic, input validation, timing and error reporting. Most of these properties can be verified at CORBA interface level with automated component test, without need for a manual GUI testing.

Logging: Mostly Replaced

Another valuable output of FEP software is its log files. The log files often contain the most detailed information about internal state of the program, its interactions with the controls hardware, clients and underlying infrastructure. Log files are used for monitoring of the system health, analysing off-normals and predicting future trends.

Log files have their consumers, and changes in the content or format of the logging may have a facility level impact. This is especially true when a log analysis tool, such as Splunk [3] is deployed at the facility to extract performance indicators, generate alerts and feed dashboard visualizations.

ICCS has standardized that both Ada and Java logs must contain certain key fields: timestamp, taxon, severity, thread id. However, the rest of the log message format is not formalized, and software developers can put any information they consider relevant. Since logging is coupled to the specifics of the implementation, the nomenclature of the log messages does differ significantly between legacy and modernized applications.

So, unfortunately, it was not realistic to preserve logging format across the Ada/Java migration. Some of our Splunk dashboards had to be redesigned to consume logging in new Java formats to support the migration.

Keepers and Goers

We summarize this section with diagram Fig. 6, mapping each concept to Idea – Implementation – Interaction hierarchy of the Design Process [4]: each system first comes into existence as a pure idea in the maker mind, it then gets implemented in silicon and code, and finally becomes complete when users interact with the system and thus with the original idea of the maker.

The diagram illustrates our focus on preservations of the original ideas and established user interactions, while "silicon and code" are being ripped out and replaced.

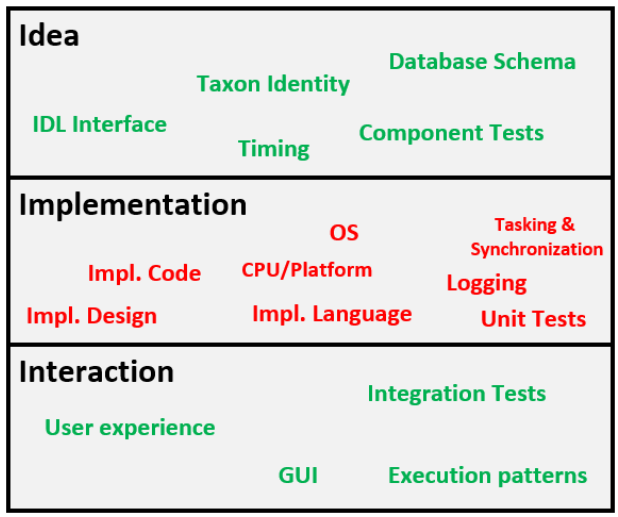

Figure 6: Summary of architecture concepts which are preserved (green) or replaced (red) over modernization.

ASSURING MIGRATION FIDELITY

When essentially everything is replaced at the "Implementation" level, how do we assure that modernized software is still compliant to the "Idea" level specifications and that operator "Interactions" do not change?

Thorough testing against the specification is desirable but difficult to implement for legacy software. The systems were developed 10+ years ago, and many of the original design documents became obsolete. Dozens or hundreds of change requests were implemented on top of the original designs, correcting and expanding software functionality. Without up-to-date and detailed requirements documents, it is difficult to define a comprehensive and accurate test plan.

Fortunately, we have realized that correctly operating instances of the control system can serve as a test fixture to supplement traditional testing processes.

Legacy Software as a Reference Implementation

We know that our legacy Ada 95 software works successfully. Its interfaces, behaviours, timings are what the rest of the system and operators expect.

We validate consistency of migrations with comprehensive automated CORBA component-level tests which rely on legacy software as a reference implementation. Developers capture expected behaviors by exercising tests against Ada 95. Correctness, error handling and execution timings are addressed by these tests.

Once the tests are "calibrated" against legacy code, they are used to validate new Java software, including method response times, Fig. 7.

Integration Testing

While the component level tests methodically exercise each method and compare them against their legacy variants, these individual tests are only as good as the developer's knowledge of the device functions. Given a decade-long gap since the original design, it is possible that this

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEDPL01

Test Case	Ada		Java (Real)		Java (Emul)	
	Status	Time	Status	Time	Status	Time
issueTriggerKey[EPOCH_	PASS	10.009	PASS	9.997	PASS	10.000
issueTriggerKey[EPOCH_	PASS	9.998	PASS	9.998	PASS	9.999
realTimeClockAccurate	PASS	0.012	PASS	0.009	PASS	0.009
realTimeClockUpdates	PASS	5.006	PASS	5.005	PASS	5.003
reinitDevice	PASS	1.216	PASS	2.650	PASS	1.058
restartFEP	PASS	150.335	PASS	15.559	PASS	30.186
setAndGetAsyncKey[CPU	PASS	0.049	PASS	0.120	PASS	0.005
setAndGetAsyncKey[CPU	PASS	0.064	PASS	0.120	PASS	0.004
setAndGetAsyncKey[ELEC	PASS	0.050	PASS	0.135	PASS	0.003
setAndGetAsyncKey[ELEC	PASS	0.049	PASS	0.119	PASS	0.003
triggerHTCAnyPermissive	PASS	10.009	PASS	10.001	PASS	9.999
triggerHTCAnyPermissive	PASS	9.989	PASS	9.998	PASS	0.000
triggerHTCAnyPermissive	PASS	9.999	PASS	10.009	PASS	9.997
triggerHTCAnyPermissive	PASS	9.998	PASS	9.989	PASS	0.002
triggerHTCSpecificPermis	PASS	8.560	PASS	11.410	IGNORED	

Figure 7: Fragment of Ada-Java migration fidelity report.

knowledge is incomplete, new code is deficient and the component level tests are blind to defects. In one example, a developer was not aware that a component is supposed to provide an optional "add-in" interface, so it was omitted entirely both from the new implementation and from the test suite.

This omission was discovered by a system-level integration test. ICCS software team is supported by several offline test instances of the control system, and our Automated Shot Test (AST) tool continuously runs the entire system through several variants of production-like experiments. We replace the first article of a legacy component with its modernized replacement and execute experiments with AST. The rest of the control system serves as a call pattern generator by driving new software through realistic scenarios and under realistic concurrent load. The system also performs the validation, since the control system already embeds many checks of state, values and timings, normally used to assure that production facility hardware performs as expected but leveraged in this first article test to assure correctness of the code migration.

Conversion and Reversion Scripts

As emphasized earlier, we need to enforce a firm time limit on our software deployment and testing activities in the production NIF environment. Whether new software works or not, we need to return the system to fully operational state in time for the next laser shot experiment. We are relying on data-driven architecture of the control system to switch between legacy Ada and new Java implementations.

This technique is implemented by preparing both forward conversion (Ada to Java) and reversion (Java to Ada) scripts and datasets. To assure robustness of these tests, we apply them in both directions in our integration and test environments. Shortly before a production release, we dry-run them against a copy of the production database.

CONCLUSION

The ICCS team is wrapping up our multi-year effort to modernize NIF control system platforms, Fig. 8. This summer of 2019 we migrated the last of VxWorks/PowerPC/Ada 95 FEPs to Linux/Intel/Java. ICCS is on schedule to convert all remaining Solaris/Ada systems by December 2019. At that point, we will remove Ada 95 from our nightly builds, eliminating dependency on the obsolete toolset.

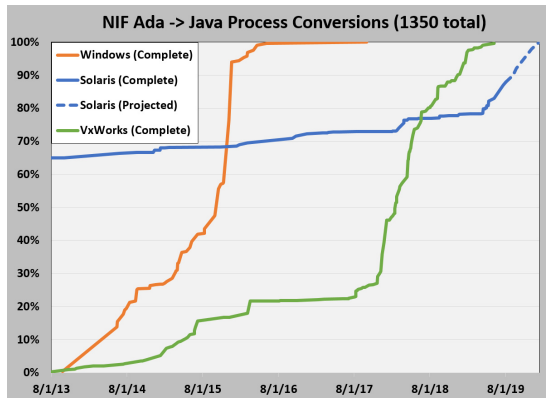

Figure 8: Progress of FEP modernization.

By adapting the "In-Place" strategy, our team has accomplished a comprehensive and deep upgrade of the control system while the NIF facility is running a busy 24x7 experimental schedule and continues to expand its scientific capabilities. Close coordination and shared values of excellent stewardship between ICCS and NIF Operations teams were essential for this success.

Looking forward, the modernized Linux/Java/Intel platform is well settled to serve NIF for the next 10-20 years. ICCS team is looking forward to apply our system migration expertise to new projects, such as hardware/firmware technology refresh of NIF Embedded Controllers.

ACKNOWLEDGMENT

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-CONF-739363.

REFERENCES

[1] P. Van Arsdall, "Integrated computer control system architectural overview", No. UCRL-ID-128811. *Lawrence Livermore National Lab.*, CA (United States), 1997.

[2] G. K. Brunton *et al.*, "Status of the National Ignition Facility (NIF) Integrated Computer Control and Information Systems", presented at the 17th Int. Conf. on Accelerator and Large Experimental Control Systems (ICALEPCS'19), New York, NY, USA, Oct. 2019, paper MOAPP04, this conference.

[3] M. A. Fedorov *et al.*, "Leveraging Splunk for Control System Monitoring and Management", in *Proc. ICALEPCS'17*, Barcelona, Spain, Oct. 2017, pp. 253-257. doi:10.18429/JACoW-ICALEPCS2017-TUCPA02

[4] Brooks Jr, Frederick P. The design of design: Essays from a computer scientist. Pearson Education, 2010.