Verifying HLA RTIs

Susan Symington
Joseph Kaplan
Frederick Kuhl
John Tufarolo
Richard Weatherly
The MITRE Corporation
McLean, VA 22102
703 983-7209
susan@mitre.org

Jeff Nielsen
ObjectSpace, Inc.
14850 Quorum Dr., Suite 500
Dallas, TX 75240
jnielsen@objectspace.com

Keywords:
HLA, RTI, Verification, Verifier, Interface Specification, testing.

ABSTRACT: An RTI Verification Facility has been established by the Defense Modeling and Simulation Office (DMSO) to test the compliance of High Level Architecture (HLA) Run Time Infrastructure (RTI) implementations with the U.S DoD High Level Architecture Interface Specification, version 1.3. Using the custom-built "RTI Verifier" software, the Verification Facility has to date verified two RTI implementations from two different RTI developers. In this paper, we describe our experience with the verification process and the RTI Verifier. We explain the process in detail, including the effort involved, test failure analysis, and turnaround time. We also discuss how the RTI Verifier works and enhancements that have been made to support future versions of the HLA Interface Specification. Finally, we present some lessons learned as an aid to future RTI developers.

1. Introduction

To ensure consistent behavior of High Level Architecture (HLA) Run Time Infrastructure (RTI) systems and to encourage multiple independent RTI developers, the Defense Modeling and Simulation Office (DMSO) has established an RTI Verification Facility to test the compliance of RTI implementations with the U.S. DoD HLA Interface Specification. HLA users who select RTIs that have been verified by DMSO can have confidence that these products have been rigorously tested for compliance to the HLA standard.

The RTI Verification Facility consists of custom-built verification software—known as the *RTI Verifier*—and staff who operate this software, write tests, and analyze test results. The architecture of the RTI Verifier has been explained in detail in [1,2]. It includes a custom-built Script Definition Language (SDL) used to author test scripts; an application executive/controller/interpreter to parse and execute scripts; test federates that connect and interact with an RTI under test; and a database to maintain requirements, tests, and test results. Figure 1 presents a functional overview of this architecture. Since the RTI Verifier architecture was described in [1], extensive enhancements have been made to both the SDL syntax and the software itself to improve the execution, analysis, and reporting of tests and their results.

2. Verifier History

As with any test system, development of the Verifier required access to a working version of an RTI against which the Verifier could be tested. In the summer of 1998, the RTI Verification Facility began designing the RTI Verifier using the RTI 1.3 [3], as a test case. In February of 1999 the Verifier had matured to the

This work was supported by the Army under contract #DAAB07-00-C-C201 and is subject to the "Rights in Data" clause at DFARS 252.227-7013 (NOV 95). This work was performed at the MITRE Corporation. Jeff Nielsen was with The MITRE Corporation when the work was done.

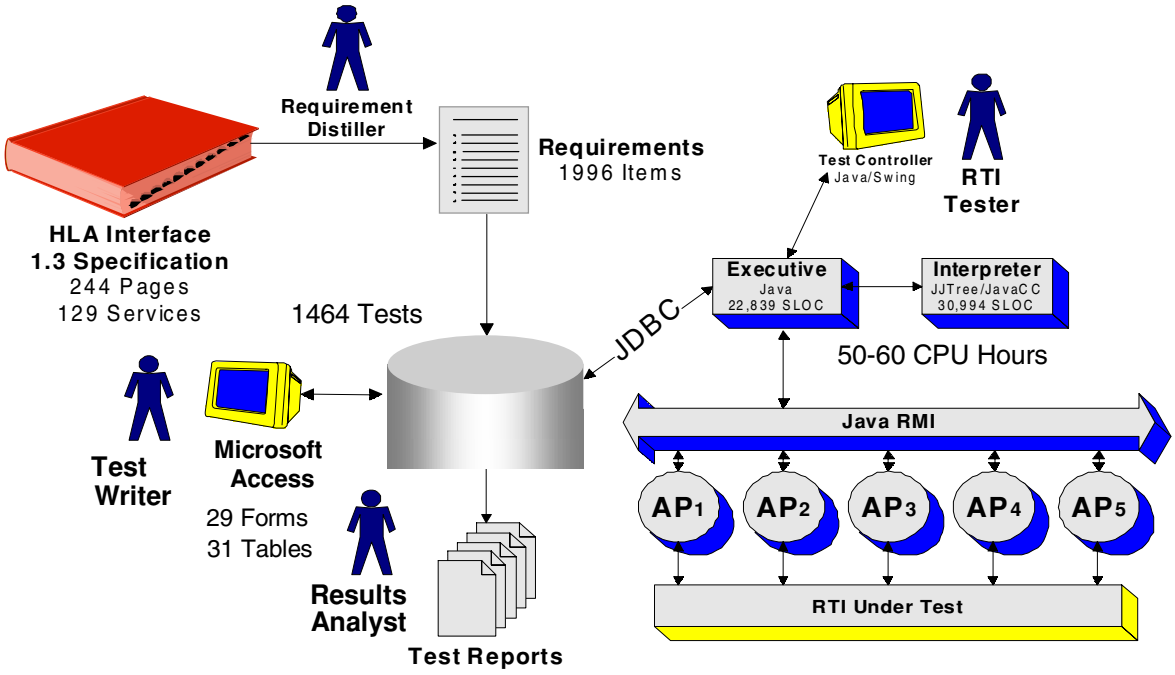

Figure 1: Verifier System Architecture

point that it could be used to test a beta version of the RTI 1.3 Next Generation (NG) software, developed by SAIC. During this time, the Verifier and RTI 1.3 NG were developed in parallel, enjoying a mutually beneficial relationship. Running the Verifier on the beta version of RTI 1.3 NG proved to be an excellent way to discover problems with Verifier software and test suites. Likewise, it was an excellent way to uncover early problems with RTI 1.3 NG.

In June of 1999, the Verification Facility began running the Verifier on an early version of a second RTI, pRTI, developed by the Swedish company Pitch AB. Having this second RTI, which was developed independently from RTI NG 1.3 and in a different language (Java as opposed to C++), enabled the Verification Facility to eliminate additional glitches and gain increased confidence in the Verifier.

Both of these RTI implementations, RTI 1.3 NG and pRTI, ultimately achieved the official status of *verified* RTIs. This means that, based on the results of verification testing, DMSO is confident that each adheres to the HLA Interface Specification, version 1.3.

2.1 Specification ambiguities

During this period of development, the Verifier and the RTIs were not the only elements being tested. In a sense, version 1.3 of the HLA Interface Specification was also under test. For the first time, two independent developers were trying to develop an implementation based completely on that specification to the extent that both RTIs would pass a comprehensive verification test suite. For the most part, the 1.3 Interface Specification proved itself to be clear and complete. However, as might be expected, several issues did arise regarding the expected behavior of an RTI. These issues arose because of ambiguous or missing text in the Interface Specification. As they arose, the Verification Facility provided feedback to the RTI developers informing them of the DoD interpretation of the Interface Specification and clarifying expected RTI behavior. These issues are documented on the DMSO web page [ftp://ftp.dmso.mil/pub/documents/initiatives/hla/rti/verify/spec_ambiguities.pdf] as guidance to future RTI developers.

3. Verification Policy and Process

Complete details on the HLA RTI Verification Policy can be found at the HLA web site [http://hla.dmso.mil/rti/verify/policy.html]. Included there are:

- a description of the verification policy,
- level 1 Test Procedures that should be self-administered by RTI developers before submitting their RTIs for verification testing,
- a form to register the RTI for verification testing, and
- a verification status board that lists
  - RTIs that have completed Level 1 testing
  - RTIs that are in Level 2 testing, and
  - RTIs that have been certified compliant.

From the viewpoint of the Verification Facility, the verification process officially begins when an RTI developer submits a formal request for verification using the automated form at the HLA web site. As part of the request, the developer specifies not only the particular RTI that is to be verified, but also the version of the HLA specification to which the RTI is intended to adhere, the HLA API, the operating system, and the language binding to be used. By submitting a formal request for verification, the RTI developer is asserting that the RTI has already successfully passed a set of fundamental tests that constitute Level 1 testing. Once the Verification Facility receives a request for verification, it contacts the developer to obtain a copy of the RTI so that testing can begin.

Access to RTI source code is not required for the verification process. Instead, the RTI Verification Facility links its own test federates with the runtime version of the RTI, executes the full suite of tests, and reports back to the RTI developers on every test that fails. It continues to run the tests on all successively submitted versions until a particular version of the RTI passes all of them.

When a single version of an RTI has passed all Verifier tests, the Verification Facility recommends that version of the RTI to the Director of DMSO for compliance certification. The Director of DMSO renders definitive judgement as to whether an RTI implementation is compliant. As indicated above, DMSO has to date certified two RTI implementations as compliant with the HLA Interface Specification version 1.3. These are:

- RTI Next Generation (NG), version 1.3, manufactured by SAIC, using the C++ API and a Sun C++ 4.2 compiler, on a Solaris 2.6 platform.
- pRTI, version 1.3, manufactured by Pitch AB, using the Java API and the JDK 1.2 compiler, on a Java 2 platform.

4. Testing Process

4.1 Phased Testing

The Verification Test Suite consists of approximately 1500 tests that perform functional testing of all areas of the HLA Interface Specification. These tests are divided into five phases for purposes of running tests and reporting test results. Thus, for any given version of an RTI under test, a developer must receive five test reports in order to have received a complete report on that RTI. The breakdown is as follows:

- Phase 1: Federation Management, Declaration Management, and Support Services
- Phase 2: Object and Ownership Management
- Phase 3: Time Management
- Phase 4: Data Distribution Management (DDM)
- Phase 5: Management Object Model (MOM)

The phases may be run in any order, with the sequence being determined by the Verification Facility.

4.2 Testing Process

Figure 2 shows a flowchart of the testing process that occurs for each of the five phases of testing with a given RTI version. First, the RTI is installed and the particular phase of the test suite is run. This may take anywhere from several hours to an entire day, depending on the phase being run and the characteristics of the RTI.

Once all tests in a given phase have been run, a Verification Facility staff member reviews all of the failed tests and their test logs in the database. This is a "quality assurance" measure in order to assure that all errors reported to the developer are truly problems with the RTI. If (as happened frequently in the early stages of development) the staff member determines that the cause of the failure was a problem with the test or the Verifier software, then the necessary corrections are made and the test is re-run. If the cause appears to be a genuine defect in the RTI, the staff member documents his failure analysis by inserting appropriate text for this test in the test report. Once all failed tests have been thus analyzed, the test report is generated and provided to the RTI developer. The test report includes a textual description of each test purpose and behavior, an analysis of where and why the test failed, and an automatically-generated log of every RTI service invoked by the test, complete with argument values.

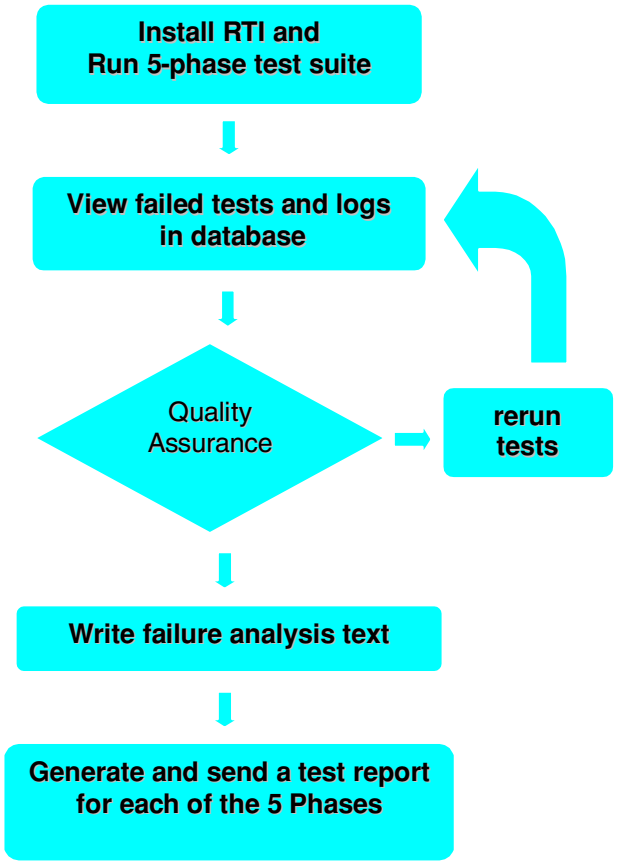

Figure 2: Testing Process Flowchart

As mentioned earlier, any given RTI version must go through this process for each of the five test phases in order for it to have been tested completely.

4.3 Costs

Given the verification process just described and the Verification Facility's experience having verified two RTIs so far, one may wonder what it costs to verify an RTI. Although the number of data points (two) is currently too small for any kind of statistically significant answer, this section attempts to present some relevant numbers.

We define a *pass* as one application of the verification test suite that includes each of the five phases of testing; and we define *turnaround time* as the amount of time elapsed from the moment the RTI is received until the moment that the last of the five phased test reports is provided to the RTI developer on a given pass. Simply running the entire suite of verification tests on a given RTI takes from 1 to 3 work days. After all tests have been run, all failed tests must be analyzed by a staff member. This quality assurance step is by far the most time and labor-intensive part of the testing process and as a result, turnaround time for a given RTI primarily depends on the number of tests that the RTI fails.

Data from two RTIs to date		
Avg. # releases delivered to Verification team	9	
	Avg. Number of Failures	Avg. Turnaround Time
Initial pass	333	~
Pass 1	179	5.5 weeks
Pass 2	57	2 weeks
Pass 3	29	6 days
Pass 4	5	4 days
Pass 5	1	3 days

Figure 3: Data from Two Verified RTIs

Our experience so far suggests that successfully completing verification testing requires about 5 passes, with the average turnaround time for the first pass being about five and a half weeks and turnaround time for each successive pass decreasing significantly. Figure 3 shows a chart of the failure rate and turnaround time data averaged from the two RTIs verified to date.

As can be seen from Figure 3, the first pass of testing typically has a large number of test failures and, consequently, a lengthy turnaround time. As the developers fix problems and as the Verification Facility corrects any problems found with the tests, subsequent passes exhibit a decrease in the number of failures, with a commensurate decrease in turnaround time. Although turnaround time is defined as the length of time required to report on the RTI's performance on the entire test suite, the test suite itself is broken down into a set of five phases. This means that RTI developers are not required to wait until all test phases have been run to receive test reports; instead, they receive the reports as the phases complete. So, although the first pass of testing may take five and a half weeks, the developer will receive its first test report much earlier than that (most likely after only one or two weeks). In a given pass, the developer can be working to eliminate errors found in one phase of testing while the Verification Facility is analyzing test results on other phases.

If an inordinate number of errors are found during a given phase of testing, a developer may choose to provide a new release of the RTI to the Verification Facility before it has reported the results of the entire pass. This choice would typically be made early during the pass, when it is discovered that a single error has widespread ramifications throughout the RTI and is causing a large number of errors. Such a choice saves both Verification Facility resources and minimizes developer wait time.

The option to provide a new release before the results of an entire pass have been received explains the data showing that an average of 9 releases of each RTI were delivered to the Verification Facility for testing, while only five of these releases were subjected to an entire pass of testing. The "Initial Pass" row of the table refers to the initial period of testing that was conducted while the Verifier itself was still being developed. Results from this early period of testing are not considered relevant to what can be expected from the typical verification process.

4.4 Testing Phenomena

Ideally, the number of failures will diminish with each pass of testing as RTI developers work to fix problems uncovered. However, there is always the chance that developers will inadvertently introduce errors when they make a change to the RTI in an attempt to fix it. In addition, there are some failures that persist despite attempts to eliminate them. Some interesting phenomena were discovered during our verification experiences that are not readily apparent from the data in Figure 3. These are

- single cause of many failures,
- failure masking, and
- labor-intensive latter failures.

Single cause of many failures: the drastic and continued decrease in verification turnaround time from the first through subsequent passes can be accounted for, in part, by the fact that in early phases of testing, many of the test failures were caused by only a few implementation errors. Hence, fixing one error can result in a large reduction in test failures in the next pass of testing. Developers who are responsive to the test reports, therefore, will tend to experience significant and systematic reductions in the number of errors, and therefore in the turnaround time, in the early passes of testing.

Failure masking: although most tests that fail in a given pass tend to be a subset of the tests that were reported as failing in previous passes, it is not always the case that tests fail for the *same reason* that they failed earlier. In fact, tests in latter passes frequently fail for an altogether different reason. For example, the table in figure 3 shows a reduction in test failures from pass 1 to pass 2 from 179 to 57. This does not mean that the developers chose to fix only the errors that were causing 122 tests to fail but not the errors that were causing the other 57 failures. On the contrary, typically the developer would deliver the RTI for its second pass of testing after having remedied all or most of the errors reported. However, in many cases, the error that had been causing the test to fail during pass 1 was merely masking a second (and perhaps third or fourth) error that was still present in the test, but was only uncovered when the first error was fixed. These errors appear on subsequent passes when the test is allowed to proceed far enough to encounter the additional error. So, no matter how responsive a developer may be to the test reports received, the five tests that failed pass 4 of the testing may have all failed each of the three previous passes. It merely required successive passes of testing to uncover failures that had been masked by previous failures to the point that all errors were uncovered.

Labor-intensive latter failures: Despite developer responsiveness, some tests failed successive passes of testing for a reason other than failure masking. In some cases, a particularly difficult or problematic failure would persist and show up repeatedly in test results despite developer attempts to eliminate it between passes. These failures, which tended to still be lurking in the latter passes, required significantly more labor on both the part of the developer and the part of the Verification Facility. These test failures could be difficult to reproduce, so that providing developers with enough information about them to enable them to be understood, tracked down, and fixed was not straightforward and often required considerable communication between the Verification Facility and the developer.

4.5 Quality Assurance Analysis

As was stated before, the quality assurance analysis performed by a Verification Facility staff member on all test failures is the most labor and time-intensive portion of the verification process. Figure 4 depicts a flowchart of this process.

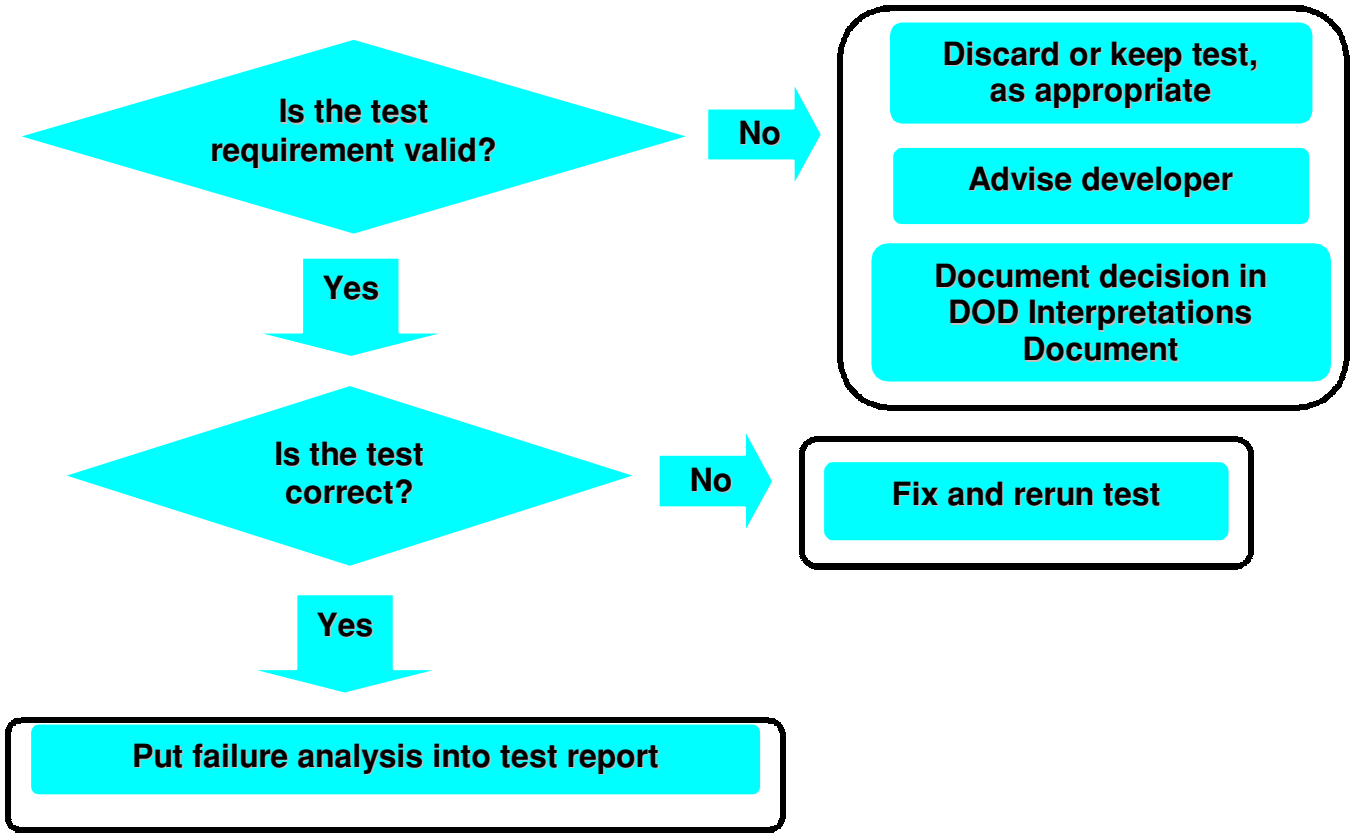

Figure 4: Quality Assurance Analysis Process Flowchart

The first step in the analysis of a failed test is to determine whether or not the requirement driving the test is itself valid. This requires combing the HLA Interface Specification and locating exactly that portion of text from which the requirement can be derived. If the requirement is not found to be derivable from the Interface Specification, then the Verification Facility makes a determination as to whether the test should be discarded or kept, as appropriate, and it advises the developer of this decision. All cases in which it is decided to keep a test and thereby enforce a requirement that can not be literally derived from the specification are documented and made publicly available on the DMSO web site at [ftp://ftp.dmso.mil/pub/documents/initiatives/hla/rti/verify/spec_ambiguities.pdf].

If the requirement is found to be valid based on the Interface Specification, the next step is to determine whether the test as written is correctly testing the requirement. If the test is found to be in error, then it is fixed and re-run. If the test is found to be correct, the analyst writes up a textual description of the test that describes the behavior that is expected and how it deviates from the behavior that was observed. This text is inserted automatically into the test report to be generated and provided to the developer.

5. Verification of Successive Releases

When an RTI is verified, it is verified according to a particular RTI version, HLA Version, HLA API, Operating System, and API language. If a new version of the RTI is released with any changes at all, that new version is not considered to have been verified. Any change made to a verified version of an RTI, no matter how seemingly small or insubstantial, has the potential to cause unanticipated problems with the way that the RTI operates, possibly resulting in an RTI that would not pass all tests in the verification test suite. Our recommendation, therefore, is that all new RTI versions should be verified before being released.

In order to accomplish verification prior to releasing a particular version of an RTI, the developer should budget an appropriate amount of time into the release schedule: one or two weeks for versions that do not incorporate substantial changes from a previous version that has already verified, and more time for

versions with more substantial changes. Several weeks may seem like too long a delay for a release that has been changed only minimally, but it is well worth the assurance that verification provides to the RTI users. The tradeoff is between a timely release and a correct release. Most users, we suspect, would rather have a correct release a little later than a buggy one earlier. Assurance that a newer version will function at least as well as the version currently in use would seem to be a minimal incentive required for a user to upgrade to a new version.

Sometimes it may make more sense to run a version through the verification test suite and note the failures in the release notes rather than subject that version to successive verification/revisions until verification is achieved. (This approach may be appropriate, for example, if the anticipated useful life of the RTI version is short, or if the RTI is not expected to be used to support any major federation executions.) This way, the users would be fully informed of the known problems with a new release and have the option of upgrading to it or not.

Practically speaking, however, budgeting an extra few weeks into the release cycle may be too much to ask. If there is not enough time to verify all releases, then what criteria should be used for determining whether or not to verify a given release? We recommend picking an amount of time, such as six months, and routinely verifying whichever version is released next after that period has elapsed. This would be a policy of periodic verification. It must be recognized, however, that a risk is being taken every time an unverified RTI is released. The potential damage includes user frustration, exasperation, and loss of confidence in the RTI developer and the Verification Facility. So, we further recommend that once a particular version of an RTI has been verified and advertised as such, every subsequent version of that RTI that is released be labeled either "verified" or "unverified". This will avoid having users fall into the trap of assuming that because one version of an RTI has been verified, all subsequent versions will automatically have been verified before being released. This will also provide users with the information that they will want to consider when deciding whether or not to upgrade to the next version.

5.1 Summary of our recommendations:

- Always verify (When in doubt, verify. With any code change, there is always doubt, so always verify.)
- Budget time for verification. View the issue as not whether to verify, but how much time to budget into the release schedule for verification. (One week for minimal changes to the RTI that aren't expected to introduce bugs, several weeks for more extensive changes.)
- Weigh the potential costs of not verifying a given RTI (including user frustration and loss of confidence that will be caused by potential bugs) against the cost of verifying.
- Practice full disclosure with new releases: label all new releases as verified or unverified. If a buggy RTI is released, fully disclose the tests that it fails in its release notes.

6. Ongoing Evolution of the RTI Verifier

Over the life of the RTI Verification Facility, the Verifier software has continued to evolve as we have gained a better understanding of what the tools need to do. With increased experience in the process of verifying RTIs has come a more complete articulation of the requirements for the Verifier system, which has necessitated ongoing enhancements. Such enhancements have helped to make the software more usable and the team more productive.

Many of the software enhancements were added incrementally, as the need for them was discovered. For example, the first time that we attempted to re-run the complete set of tests on an RTI in a short time frame, we discovered that we needed a less labor-intensive method of aggregating the results of a run for a particular RTI version. Around this same time we also realized that the generation of the test reports (with their associated logs) could be automated further. Other needs that surfaced as we began performing large numbers of tests included: being able to run the software remotely using a command-line interface, being able to easily re-run only those tests that had failed previously, and having the software automatically recover (i.e., “clean up”) from *any* kind of RTI or test-federate failure during a test.

6.1 Version 2.0 of the Verifier Software

With the successful verification of two RTIs becoming a reality early this year, we felt that the time was right for a more systematic overhaul of the system. A couple of factors precipitated this decision. First, it had become increasingly clear that the software requirements had grown significantly beyond the scope of the original design. Second, the impending

approval of the IEEE 1516 series of HLA standards meant that we would need to be prepared to verify RTIs written to newer versions of the HLA specification.

A complete re-design and re-implementation effort for both the database and Java-based tools was therefore initiated late in 1999. This effort resulted in version 2.0 of the RTI Verifier, which was largely complete as of June 2000. This version incorporates all of the previous enhancements and includes many other new or improved features. Some of the most significant are:

- *A new database format, database front-end tools, and Java GUI.*
- *Support for an enhanced SDL scripting language*. Our experience writing the existing 1500 tests convinced us that changes could be made to the SDL grammar in order to make it more concise and expressive in describing an RTI's expected behavior. The new grammar includes features like assertion-based constructs to painlessly check return values and callback parameters, automatic handle lookup and translation, and parameterized subroutines.
- *Backward compatibility with existing tests and databases.*
- *The ability to work with multiple RTI APIs.* The previous version of the Verifier had the RTI services from the 1.3 API "hard-coded" in both the Java code and the SDL interpreter. The current version takes advantage of the Java "reflection" capabilities to be completely independent of any particular API. An API is loaded dynamically at runtime and used to build a table of services, callbacks, and GUI menus. Invoked services (either from a script or from the GUI) and returned callbacks are then looked up by name.
- *A more modular design, allowing the different components of the system to be used independently*. For example, the system can read tests from and write results to either a database or a file system (the latter being useful if one wants to run without a database connection). It can likewise be run either with the GUI or in command-line mode. It can even be configured to behave as a single, local, API-independent test federate to work with any RTI.

The RTI Verifier 2.0 is thus both robust and flexible, and well equipped to meet the future needs of the Verification Facility. More complete details about its design and features will be presented in a forthcoming paper.

6.2 Preparing for the IEEE Specification

Although the Verifier software is versatile enough that it can be used to test RTIs that implement different versions of the HLA specification, many of the tests that the Verifier software runs must be written anew for each new version of the specification. A new specification implies a new set of requirements, which in turn requires a new set of tests. Even existing tests that remain valid must be translated to conform to a new specification's API. The Verification Facility staff is currently in the process of writing new requirements and their corresponding tests for the IEEE 1516 version of the HLA specification.

7. Lessons Learned & Recommendations

Given our experience thus far with successfully verifying two RTIs, we have the following recommendations for RTI developers:

- Read the HLA Interface Specification carefully. When in doubt, contact the Verification Facility for clarification.
- For RTIs being implemented to the 1.3 Interface Specification, read the DoD Interpretations document that specifies the expected behavior for those portions of the specification that are ambiguous [ftp://ftp.dmso.mil/pub/documents/initiatives/hla/rti/verify/spec_ambiguities.pdf].
- Schedule time early in your development cycle for verification. Do not tack 3-4 weeks on the end of your development cycle and expect verification to both begin and complete successfully in that time frame. Expect development to complete 4-6 months after verification begins. This is true because often failures in earlier runs through the Verifier mask failures that are uncovered later, after the early failures have been corrected. Also, fixing one problem may inadvertently cause another problem to be introduced, and our experience has shown that the bugs that are still in the RTI implementation near the end of the process are often the most difficult to fix. Furthermore, even if there is only one error in an implementation, the RTI must be fixed and the corrected version of the RTI must be run through all Verifier tests again. This takes time.

8. References

[1] Tufarolo, J., Nielsen J., Symington, S., Weatherly, R., Wilson, A., Ivers, J., and Hyon, T.: "Automated Distributed System Testing: Designing an RTI Verification System", *1999* Winter Simulation Conference, pp. 1094-1102, December 1999.

[2] Tufarolo, J., Nielsen J., Symington, S., Weatherly, R., Wilson, A., Ivers, J., and Hyon, T.: "Automated Distributed System Testing: Application of an RTI Verification System", 1999 Winter Simulation Conference, pp. 1103-1108, December, 1999.

[3] U.S. Department of Defense, Defense Modeling and Simulation Office, High Level Architecture Interface Specification, v1.3, April 2, 1998.

Acknowledgments

The authors would like to thank Dr. Ernest Page of The MITRE Corporation for his contribution to the development of Verifier software and Verification testing as a member of the Verification Facility staff.

Author Biographies

JOSEPH A. KAPLAN is a Senior Simulation and Modeling Engineer in the Information Systems and Technology Division at The MITRE Corporation in Reston, Virginia. He is currently supporting the HLA RTI Verification Facility as a member of the software development team. His professional interests include real-time software systems and distributed computing. Mr. Kaplan has a B.S. in Computer Science from Virginia Polytechnic Institute and State University and a M.S. in Computer Science from the College of William and Mary.

DR. FREDERICK KUHL is a senior principal engineer with The MITRE Corporation. He has been a leader since the beginning in the activities to prototype implementations of the HLA infrastructure. He presently leads the international standardization effort for the HLA. He has applied object-oriented programming to prototype air traffic control systems, and distributed object computing technology to distributed simulation. Dr. Kuhl holds the Ph.D. in computer science from Texas A&M University.

JEFF NIELSEN is a senior consultant and Infrastructure Specialist at ObjectSpace, Inc., where he helps organizations design and implement distributed object-based software systems. He was previously lead designer and developer for the HLA RTI Verifier software at the MITRE Corporation. He also participated in a variety of HLA-related activities, providing technical and management support to DMSO-sponsored federation efforts and contributing to the development and balloting of the HLA IEEE specification. Mr. Nielsen holds an M.S. in Computer Science and an M.A.Ed. in Instructional Technology.

SUSAN SYMINGTON is a Lead Scientist at the MITRE Corporation where she serves as the point of contact for the RTI Verification Facility. She is also the chair of the IEEE High Level Architecture Working Group that drafted the three M&S HLA draft standards: P1516, P1516.1, and P1516.2. She holds a B.A. in Mathematics and Philosophy from Yale University and an M.S. in Computer Science from the University of Maryland at College Park.

JOHN A. TUFAROLO is a Lead Simulation Systems Engineer for the MITRE Corporation in Reston, Virginia, where he is currently involved in High Level Architecture (HLA) testing and HLA federation development activities. Mr. Tufarolo is the Information Director for the Association of Computing Machinery (ACM) Special Interest Group on Simulation (SIGSIM), and a member of the ACM, IEEE CS, and SIGSIM. His professional interests include discrete event simulation, simulation systems development, and military modeling and simulation. Mr. Tufarolo has a B.S. degree in Electrical Engineering from Drexel University and an M.S. in Systems Engineering from George Mason University.

RICHARD WEATHERLY is a Consulting Engineer of The MITRE Corporation's Information Systems and Technology division where he leads various HLA infrastructure development and verification projects. Prior to that he was the project leader and designer of the Aggregate Level Simulation Protocol system. He is currently serving as the Joint Simulation System (JSIMS) Chief Engineer. He received his Ph.D. in electrical engineering from Clemson University in 1984.