Proceedings of the 1999 Winter Simulation Conference
P. A. Farrington, H. B. Nembhard, D. T. Sturrock, and G. W. Evans, eds.

AUTOMATED DISTRIBUTED SYSTEM TESTING: APPLICATION OF AN RTI VERIFICATION SYSTEM

John Tufarolo
Jeff Nielsen
Susan Symington
Richard Weatherly
Annette Wilson

The MITRE Corporation
1820 Dolley Madison Boulevard
McLean, VA 22102-3481, U.S.A.

James Ivers

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213, U.S.A.

Timothy C. Hyon

TRW International Defense
Simulation Systems
12902 Federal Systems Park Drive
Fairfax, VA 22033, U.S.A.

ABSTRACT

A new distributed test system is used to verify the Runtime Infrastructure (RTI) component of the High Level Architecture (HLA). As part of this effort, a test suite has been designed and implemented to provide a coordinated and automated approach to testing this distributed system. This paper describes the application of this test environment and its utility in verifying an HLA RTI.

1 INTRODUCTION

The High Level Architecture (HLA) was established as the standard technical architecture for all DoD simulations (U.S. Department of Defense 1996). A fundamental component of this architecture is known as the *HLA Interface Specification* (Defense Modeling and Simulation Office 1998). This specification describes the functional interface between simulation *federates* (simulation or simulation surrogate participating in a federation) and the runtime environment. An implementation complying with the HLA Interface specification is known as an *HLA Runtime Infrastructure* (RTI).

A verification system (*The Verifier*) has been designed and developed in order to test implementation of an RTI. See the companion paper, Tufarolo et. al, (1999), for a description of the design and implementation of The Verifier system.

This paper describes the application of the RTI verification system and the experiences therein. Section 2 presents the process of transforming the interface specifications into test scripts. Results and conclusions are included as sections 3 and 4.

2 USING THE VERIFIER

Distillation of the HLA Interface specification into a set of test requirements requires looking across HLA services, as many of the services are interrelated. A systematic approach is needed to confirm that all services were covered and that that the resulting test requirements could be easily traced back to the Interface Specification (in case a test is challenged, or the specification changes).

2.1 Interface Specification to Test Requirements

The test requirements are organized first by chapter of the interface specification, and then by service or closely related group of services (e.g., grouping the test requirements for the Send Interaction and Receive Interaction services together).

For each service or service group (hereafter just service), test cases are created that should succeed as well as those that should fail. Each pre-condition of the service is considered in producing a separate case that should fail. Each post-condition is checked for every case that should succeed.

The next consideration is how to account for *other* services that could interact with the service being tested. Sometimes examining these interactions takes the form of considering how the *results* of a service invocation differ based on the state of the system prior to making the service invocation. In these cases, it is the System State that is typically specified in the test requirement, not the sequence of services that established that state.

In other cases, these interactions are better considered by examining how the invocation of a service affects future service invocations (e.g., Does it cause something else to happen? Does it set up some transaction that must be completed in subsequent service invocations?).

A third category of services to consider represent cases in which an RTI should invoke a service when some condition becomes true. For such services, cases are included that verify the RTI invokes the proper service when the condition becomes true, as well as cases that check that an RTI does *not* invoke the service when the condition is not true. The latter is most frequently accomplished by setting up the system state such that the condition is "almost" true (e.g., three of four pre-conditions were satisfied). The SDL keyword ***other*** is very useful in catching unexpected RTI service invocations outside of these special cases.

Much of the specification service interaction information could be drawn from formal representations of the HLA, such as documented in (Allen, Garlan, and Ivers 1998). Other interactions could be found in the documentation of the services themselves, or the statecharts found in the HLA Interface Specification.

In producing test requirements for a given service, an important question is: how much is enough? Complete coverage is not attempted, a well-known impractical goal. A more realistic goal is more to produce a level of confidence that the hard cases and interrelationships among services are considered and correctly addressed.

In addition to including the cases that should fail, a decision is needed to determine how many cases to include that should succeed. In most cases, this decision is based on the possible combinations of the service arguments. For each argument, a set of possible abstract states that could be represented is identified. For example, an instance attribute could be owned, un-owned, in the process of being divested, or in the process of being acquired. Determining the abstract states is based, as often as possible, on the results of prior formal analyses or the statecharts in the specification.

In some cases, such as the Time Management services, this is a much more difficult task. The arguments are numbers and cannot be easily broken into abstract states (before, same, and after are a starting point, but there are several, related numbers to consider for each service and the combinations do get large). Complete coverage is clearly unachievable, even using the abstract state approach. Consequently, this area instead focuses on the most likely areas of complexity (as determined by formal analysis of the specification and understanding of possible algorithms to implement these services). For the cases deemed less complex, only some basic cases are identified.

2.2 Example Test Requirement Derivation

Take for example the RTI service *Request Federation Restore*. One of the pre-conditions for this service is:

> *The correct number of federates of the correct types that were joined to the federation execution when the save was accomplished are currently joined to the federation execution.*

This one pre-condition generates several test requirements, namely, verification that invocation of the *Request Federation Restore* service generates an exception in the cases that:

- the number of federates that are currently joined to the federation execution is fewer than the number of federates that were joined when the save was accomplished, and
- the number of federates that are currently joined to the federation execution is greater than the number of federates that were joined when the save was accomplished, and
- the total number of federates that are currently joined to the federation execution is equal to the total number of federates that were joined when the save was accomplished, but a different number of each type of federate is joined.

For Federate services, test requirements associated with preconditions typically involve verifying that if all preconditions for the service have not yet been met, the RTI will not invoke a callback for that service at the Federate. Judgement must be used on the part of the individual who is generating the test requirements to try to discern what sets of circumstances that don't satisfy the preconditions for a given federate service might, in an erroneously implemented RTI, generate a callback for that service at the federate.

Consider, for example, the federate service *Start Registration For Object Class*. One of the preconditions for this service is:

> *At least one of the class attributes that the federate is publishing at the specified object class is actively subscribed to at the specified object class or at a super-class of the specified object class by at least one other federate in the federation execution.*

This one pre-condition is capable of generating several test requirements. Namely, verification that a federate that is publishing one or more attributes of a given object class but is not yet registering instances of that class will not receive a *Start Registration For Object Class* callback as a result of another federate:

- passively subscribing to one or more of those attributes at that class or at a superclass of that class, or
- actively subscribing to one or more different attributes of that class or of a superclass of that class.

For each service, one or more test requirements are also needed to verify that after the RTI has executed a given service, the post-conditions for that service are met. For RTI services, test requirements associated with post-conditions typically involve verifying that after the given RTI service is invoked, the post-conditions for that service are met. Consider the RTI service *Unpublish Object Class*. One of the post-conditions for this service is:

The federate shall no longer own any instance attributes of object instances whose known class is the specified object class.

This one post-condition generates several test requirements, namely, verification that invocation of the *Unpublish Object Class* service by a given federate for a given class results in:

- Divestiture of all instance attributes whose known class at that federate is the given class and whose known class is equivalent to the registered class.
- Divestiture of all instance attributes whose known class at that federates is the given class and whose known class is not equivalent to the registered class.

Again, judgement must be used on the part of the individual generating the test requirements to determine that the distinctions between two cases such as those above make them each worth testing.

For Federate services, test requirements associated with post-conditions typically involve verifying that upon all pre-conditions for the service being met, the RTI will callback the Federate with that Federate service invocation. For example, consider the federate service *Stop Registration For Object Class.* Its post-condition is merely,

The Federate has been notified of the requirement to stop registration of object instances of the specified object class.

The invocation of the *Stop Registration For Object Class* service at a given federate constitutes the notification to that federate to stop registration. Test requirements generated from this post-condition, therefore, amount to requirements that verify that once all preconditions of the *Stop Registration For Object Class* service have been met, the service will in fact be invoked. Specifically, these test requirements include verification that a publishing federate will receive the *Stop Registration For Object Clas*s service callback in the following cases:

- When all subscriptions to published class attributes at a given class and at all superclasses of that class are removed, and there are no passive or active subscriptions to attributes that are not published.
- When all active subscriptions to published class attributes at a given class and at all super classes of that class are removed, but passive subscriptions to one or more published attributes exist.
- When all active subscriptions to published class attributes at a given class and at all super classes of that class are removed, but active subscriptions to one or more attributes that are not published exist.
- When there are active subscriptions to the published attributes of a given class and then the publishing federate changes the set of class attributes that it is publishing such that none of the class attributes that are published are subscribed at the given class or at any superclass of it.
- When there are active subscriptions to the published attributes of a given class and then the publishing federate changes the set of class attributes that it is publishing such that none of the class attributes that are published are actively subscribed at the given class or at any superclass of it, but some are passively subscribed at the given class or a superclass of it.

2.3 Test Requirement to Test Script

Once a test requirement has been generated, generally a single test is written to verify that the RTI under examination meets the test requirement. The test will typically consist of several scripts run in sequence. Consider the last test requirement listed above for the *Stop Registration For Object Class* service. A test to verify that an RTI meets this requirement would typically consist of a sequence of three scripts for accomplishing setup, requirement-specific testing, and cleanup. Setup and cleanup are accomplished via generic, re-usable scripts that set up and clean up the testing environment before and after the requirement-specific script is run. The setup script would, for example, create a federation execution and join three federates at attachment points ap1, ap2, and ap3. Variables would be declared and appropriate object class and attribute handles would be retrieved. The cleanup script would be designed to clean up the testing environment regardless of the whether the RTI under test met or failed to meet the requirement being tested. It would accept appropriate pending callbacks from the RTI, resign all federates from the execution, and destroy the federation execution. The requirement specific test would consist of the SDL code presented in Figure 1.

```
invoke ap1 publishObjectClass(class_ACD, attrs_xzws)
invoke ap2 subscribeObjectClass(class_ACD, attrs_xz)
accept any
startRegistration ap1 (whichClass) {
 if (whichClass != acd) then
  exitFailure (“incorrect class”);
 }
unsatisfied {
 exitFailure (“Did not receive expected callback”);
}
invoke ap3 subscribePasObjClass(class ACD, attrs_ws)
invoke ap1 publishObjectClass(class_ACD, attrs_ws)
accept any
 stopRegistration ap1 (whichClass) {
 if (whichClass !=acd) then
  exitFailure (“incorrect class”);
 }
 unsatisfied {exitFailure(“Did not receive expected callback”);}

exitSuccess (“Script completed successfully”);
```

Figure 1: Sample SDL Code

The requirement-specific script generates the following scenario: the federate at attachment point 1 (ap1) publishes class attributes x, z, w, and s of object class ACD. The federate at ap2 subscribes only to attributes x and z at class ACD. Because all of the preconditions for the *Start Registration For Object Class* service are now met, the federate at ap1 should receive the *Start Registration For Object Class* callback. If the callback is not received at ap1, the RTI is not functioning correctly and the test fails, and the script exits. If the federate at ap1 does receive the *Start Registration for Object Class* callback, then the object class for which it receives this callback should be class ACD. If it is not, then the test fails and the script exits. Next, the federate at ap3 subscribes, passively, to class attributes w and s at class ACD. Then the federate at ap1 changes the set of class attributes that it publishes at object class ACD such that none of the class attributes that it is publishing is actively subscribed at the specified object class or at a super-class of the specified object class by any other federate in the federation execution. This happens to be the last unmet precondition for the *Stop Registration For Object Class* service. Therefore, because of the federate at ap1 changing the set of class attributes that it is publishing, the federate at ap1 should receive the *Stop Registration For Object Class* callback for object class ACD. If it does not, then the RTI has failed the test for this requirement and the test is exited. If it does, the test completes successfully.

2.4 Executing Tests

To test whether an RTI meets a given test requirement, the following steps must be taken:

- The RTI to be tested must be initiated.
- The RTI Verifier system must be initiated.
- One or more test federates need to be initiated. (A maximum of five. The required number may be discerned from the particular scripts that are to be run.)
- Once the Verifier system has been initiated, the human tester must use the test controller GUI to associate the test federates that have been initiated with particular Verifier attachment points.
- The human tester must use the test controller GUI to select and initiate the particular script, test, series, or scenario to be executed.
- The human tester may monitor the ongoing execution of the script, test, series, or scenario and/or the results of this execution may be stored in the database.

Successful execution of a test script yields a test trace to the Test Controller GUI, as well as recorded results the database. A sample test trace (based on *the Stop Registration For Object Class* service example presented in section 2.2) is provided in Figures 2 and 3. Figure 2 shows the trace from the setup script, and Figure 3 the trace from the main part of this test.

3 PRELIMINARY RESULTS

The Verifier system has been implemented and script development for the RTI specification version 1.3 is in-progress. The system has successfully tested RTI Federation Management, Federation Support Services, Time Management, Object Management, and Ownership Management services. Other RTI service tests are pending completion of script development efforts. The Verifier helped to identify an number of legitimate (non-trivial) problems with an RTI currently under development. Moreover, the trace data from the failed tests helped to easily reproduce the error and to communicate the information to the RTI developer.

4 CONCLUSIONS

In conjunction with design and implementation of the Verifier, it was necessary to develop a set of test requirements that an RTI must meet and that therefore must be tested by the Verifier. These requirements were generated by systematically proceeding through each service defined in the Interface Specification and distilling a set of

```
Begin Test Execution: Stop_Reg_Test_5
Begin Script Execution: setup
        To AP 1: createFederationExecution
        Federation successfully created
        To AP 1: joinFederationExecution
        FedType: type1; FedEx : Verification
        Done. Returned handle 1.Federate 1 joined
        To AP 2: joinFederationExecution
        FedType: type2; FedEx : Verification
        Done. Returned handle 2.Federate 2 joined
        To AP 3: joinFederationExecution
        FedType: type2; FedEx : Verification
        Done. Returned handle 3.Federate 3 joined
        To AP 1: getObjectClassHandle; Name : A.C.D
        Done.Handle : 11
        To AP 1: getAttributeHandle; Name : X; Class : 11
        Done. Handle : 1
        To AP 1: getAttributeHandle; Name : Z; Class : 11
        Done. Handle : 2
        To AP 1: getAttributeHandle; Name : W; Class : 11
        Done. Handle : 3
        To AP 1: getAttributeHandle; Name : S; Class : 11
        Done. Handle : 4
        Script completed successfully
End Script Execution: setup
```

Figure 2: Setup Script Output

```
Begin Script Execution: Stop_Reg_Script_5
        To AP 1: publishObjectClass
         theObjectClass: 11
         theAttributes: { 1,2,3,4 }
        To AP 2: subscribeObjectClassAttributes
         Object Class: 11
         Attributes : { 1,2 }
        From AP 1: startRegistrationForObjectClass.
         theClass: 11
        Returning startRegistrationForObjectClass To AP: 1
        To AP 3: subscribeObjectClassAttributesPassively
         Object Class: 11
         Attributes : { 3,4 }
        To AP 1: publishObjectClass
         theObjectClass: 11
         theAttributes: { 3,4 }
        From AP 1: stopRegistrationForObjectClass.
         theClass: 11
        Returning stopRegistrationForObjectClass To AP: 1
End Script Execution: Stop_Reg_Script_5
```

Figure 3: Test Script Output

requirements that is sufficient to produce a level of confidence that the hard cases and interrelationships among services had been considered and correctly addressed by the RTI under test. Because the goal of achieving complete coverage of all requirements was not practical, formal analysis of the interface specification and an understanding of possible algorithms to implement each service had to be brought to bear in determining the specific requirements that would be tested.

Application of this system to date has proved to be very useful to meet the unique needs of testing an RTI. Future efforts include completing requirements generation and script development for the entire RTI 1.3 specification, and evolving the test requirements and existing scripts to address the IEEE Standard "P1516.1 Draft Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) – Federate Interface Specification" when it is approved as a formal standard.

ACKNOWLEDGEMENTS

The authors would like to acknowledge the efforts of Reed Little of the Software Engineering Institute (SEI) at Carnegie Mellon University and Dave Seidel of the MITRE Corporation for their assistance in creating and reviewing this paper. The U.S. Department of Defense (DoD) Defense Modeling and Simulation Office (DMSO) supports the work on the Verifier Project.

The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

REFERENCES

Allen, R.J., D. Garlan and J. Ivers. 1998. Formal modeling and analysis of the HLA component integration standard. *Proceedings of the Sixth International Symposium on the Foundations of Software Engineering* (FSE-6).

Defense Modeling and Simulation Office. 1998. High Level Architecture Interface Specification, v1.3.

Tufarolo, J., Nielsen J., Symington, S., Weatherly, R., Wilson, A., Ivers, J., and Hyon, T. 1999. Automated Distributed System Testing: Designing an RTI Verification System. In *1999 Winter Simulation Conference Proceedings*, ed. P.A. Farrington, H.B. Nembhard, D.T. Sturrock, and G.W. Evans. Institute of Electrical and Electronics Engineers, Piscataway, New Jersey.

U.S. Department of Defense. 1996. DoD High Level Architecture (HLA) for Simulations. U.S. Departement of Defense. Memorandum signed by USD(A&T).

AUTHOR BIOGRAPHIES

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 BS degree in Electrical Engineering from Drexel University and an MS in Systems Engineering from George Mason University.

TIMOTHY C. HYON is a Software Developer for the TRW Systems & Information Technology Group in Fairfax, Virginia, where he is currently developing an advanced simulation system known as the Planning Support Function (PSF). PSF will provide a software tool to help Japan's new system of centralized government better prepare itself to handle natural disasters, humanitarian assistance, and national defense. Prior to joining TRW, he was a Senior Simulation and Modeling engineer for the MITRE Corporation in Reston, Virginia, where he was a lead engineer for the HLA Runtime Infrastructure (RTI) and RTI Verifier system development projects. Mr. Hyon received a BS degree in Electrical Engineering from the University of Delaware and an MS in Electrical Engineering from Georgia Institute of Technology.

JAMES IVERS is a member of the technical staff at the Software Engineering Institute where he works in the Architecture Trade-off Analysis initiative. He is interested in formal methods and analysis, particularly as applied to software architectures. He received a BA in Computer Science and Mathematics from Transylvania University and an MSE in Software Engineering from Carnegie Mellon University.

JEFF NIELSEN is a Senior Software Systems Engineer at the MITRE Corporation. His current HLA-related activities include providing technical and management support to DMSO-sponsored federation efforts, participating in the HLA IEEE specification development, and developing the RTI Verifier test suite. Mr. Nielsen holds an MS in Computer Science and a M.A.Ed. in Instructional Technology, and is currently pursuing a Ph.D. in Computer Science. He is a member of IEEE.

SUSAN SYMINGTON is a Lead Scientist at the MITRE Corporation where she is involved in HLA testing activities using the Verifier. She is also the chair of the IEEE High Level Architecture Working Group that drafted the three HLA draft standards: P1516, P1516.1, and P1516.2. She holds a BA in Mathematics and Philosophy from Yale University and an MS in Computer Science from the University of Maryland at College Park.

RICHARD WEATHERLY is the Chief Engineer of the MITRE Corporation’s Information Systems and Technology division where he leads their DoD High Level Architecture (HLA) Runtime Infrastructure (RTI) software development team. He is a core member of the Defense Modeling and Simulation Office (DMSO) Technical Support Team and contributor to their HLA Interface Specification, Time Management, and Data Distribution Management working groups. He received his Ph.D. in Electrical Engineering from Clemson University. Please see http://ms.ie.org/weatherly for recent work.

ANNETTE L. WILSON is a Lead Modeling and Simulation Engineer in the Information Systems and Technology Division at the MITRE Corporation, McLean, VA. She is currently project engineer for the RTI Verifier, is a member of the RTI 1.3 development team, and supports the CADRE program by providing RTI expertise for participating federations. Ms. Wilson was one of the developers of the ALSP Infrastructure software (AIS) and chaired the AIS subgroup of the ALSP Interface Working Group. Ms. Wilson has a BS in Computer Science from Texas A&M University and is pursuing an MS in Computer Science at George Mason University.