Robustness Testing of A Distributed Simulation Backplane

Masters Thesis

Kimberly Fernsler

Advisor: Philip Koopman

Department of Electrical & Computer Engineering

Carnegie Mellon University

Pittsburgh, Pennsylvania 15213

kf@andrew.cmu.edu

May 3, 1999

Robustness Testing of A Distributed Simulation Backplane

Abstract

Creating robust software requires quantitative measurement in addition to careful specification and implementation. The Ballista software robustness testing service provides exception handling measurements for a variety of application domains. This thesis describes Ballista testing of the High Level Architecture Run-Time Infrastructure (HLA RTI), a general-purpose distributed simulation backplane developed by the Defense Modeling and Simulation Office that has been specifically designed for robust exception handling. While more robust than off-the-shelf POSIX operating systems, the RTI had normalized robustness failure rates as high as 10.2%. Non-robust testing responses included exception handling errors, hardware segmentation violations, “unknown” exceptions, and task hangs. Additionally, testing repeatedly crashed one version of the RTI client through an RTI service function call. Results obtained from testing the same version of the RTI on two different Unix operating system platforms demonstrate some difficulties in providing comparable exception handling coverage across platforms, suggesting that the underlying OS can have a significant effect on the way robustness failures manifest. Results obtained from testing the same RTI interface specification produced by two different development teams illustrate common robustness failures that programmers can overlook. Testing the RTI led to scalable extensions of the Ballista architecture for handling exception-based error reporting models, testing object-oriented software structures (including callbacks, pass by reference, and constructors), and operating in a state-rich, distributed system environment. Robustness testing has been demonstrated to be a useful adjunct to high-quality software development processes, and is being considered for adoption by the HLA RTI verification facility. The results of this testing yield insights into the types of robustness problems that can occur with an application specifically designed to be highly robust.

1. Introduction

Robustness in software is becoming essential as critical computer systems increasingly pervade our daily lives. It is not uncommon (and although annoying, usually not catastrophic) for desktop computing applications to crash on occasion. However, as more and more software applications become essential to the everyday functioning of our society, we are entering an era in which software crashes are becoming unacceptable in an increasing number of application areas.

Careful specification and implementation are required to create robust software (*i.e.*, software that responds gracefully to overload and exception conditions [IEEE90]). In particular, it is thought by some that exception handling is a significant source of operational software failures [Cristian95]. However, cost, time, and staffing constraints often limit software testing to the important area of functional correctness, while leaving few resources to determine a software system's robustness in the face of exceptional conditions.

The Ballista software robustness testing service provides a way for software modules to be automatically tested and characterized for robustness failures due to exception handling failures. This service provides a direct, repeatable, quantitative method to evaluate a software system's robustness. Ballista works by performing tests on the software based on traditional "black box" testing techniques (*i.e.*, behavioral rather than structural testing) to measure a system's responses when encountering exceptional parameter values (overload/stress testing is planned as future work). Previously the focus of Ballista was on testing the robustness of several implementations of the POSIX [IEEE93] operating system C language Application Programming Interface (API), and found a variety of robustness failures that included repeatable, complete system crashes that could be caused by a single line of source code [Kropp98].

This paper explores whether the Ballista testing approach works on an application area that is significantly different than an operating system API, testing the hypothesis that Ballista is a general-purpose testing approach that is scalable across multiple domains. The new application area selected for testing is the Department of Defense's High Level Architecture Run-Time Infrastructure (HLA RTI). The RTI is a

general-purpose simulation backplane system used for distributed military simulations, and is specifically designed for robust exception handling. The RTI was chosen as the next step in the development of Ballista because it not only has a significantly different implementation style than a C-language operating system API, but also because it has been intentionally designed to be very robust. HLA has been designed to be part of a DoD-wide effort to establish a common technical framework to facilitate the interoperability and reuse of all types of models and simulations, and represents the highest priority effort within the DoD modeling and simulation community [DMSO99]. Because RTI applications are envisioned to consist of large numbers of software modules integrated from many different vendors, robust operation is a key concern.

Extending the Ballista architecture to test the RTI involved stretching the architecture to address exception-based error reporting models, testing object-oriented software structures (including callbacks), incorporating necessary state-setting "scaffolding" code, and operating in a state-rich distributed system environment. Yet, this expansion of capabilities was accomplished with minimal changes to the base Ballista architecture.

Beyond demonstrating that the Ballista approach applies to more than one domain, the results of RTI testing themselves yield insights into the types of problems that can occur even with an application designed to be highly robust. Testing the RTI revealed a significant number of unhandled exception conditions, unintended exceptions, and processes that can be made to "hang" in the RTI. Additionally, problems were revealed in providing equivalent exception handling support when the RTI was ported to multiple platforms, potentially undermining attempts to design robust, portable application programs.

In the remainder of this paper, Section 2 discusses how Ballista works and what extensions were required to address the needs of RTI testing. Section 3 presents the experimental methodology. Section 4 presents the testing results, while Section 5 provides conclusions and a discussion of future work.

2. Extending Ballista for RTI testing

Ballista testing works by bombarding a software module with combinations of exceptional and acceptable input values. The reaction of the system is measured for either catastrophic failure (generally involving a machine reboot), a task “hang” (detected by a timer), or a task “abort” (detected by observing that a process terminates abnormally). The current implementation of Ballista draws upon lists of heuristic test values for each data type in a function call parameter list, and executes combinations of these values for testing purposes. In each test case, the function call under test is called a single time to determine whether it is robust when called with a particular set of parameter values.

2.1. Prior Work

The Ballista testing framework is based on several generations of previous work in both the software testing and fault-tolerance communities. The Crashme program and the University of Wisconsin Fuzz project are both prior examples of automated robustness testing. Crashme works by writing random data values to memory and then attempts to execute them as code by spawning a large number of concurrent processes [Carette96]. The Fuzz project injects random noise (or “fuzz”) into specific elements of an OS interface [Miller98]. Both methods find robustness problems in operating systems, although they are not specifically designed for a high degree of repeatability, and Crashme in particular is not generally applicable for testing software other than operating systems. While robustness under stressful environmental conditions is indeed an important issue, a desire to attain highly repeatable results has led the Ballista project to consider robustness issues in a single invocation of a software module from a single execution thread.

Approaches to robustness testing in the fault tolerance community are usually based on fault injection techniques, and include Fiat, FTAPE, and Ferrari. The Fiat system modifies the binary image of a process in memory [Barton90]. Ferrari, on the other hand, uses software traps to simulate specific hardware level faults, such as errors in data or address lines [Kanawati92]. FTAPE uses hardware-dependent device drivers to inject faults into a system running with a random load generator [Tsai95]. These approaches have

produced useful results, but were not intended to provide a scalable, portable quantification of robustness for software modules.

There are several commercial tools such as Purify [Pure Atria], Insure++ [Parasoft], and BoundsChecker [Numega] that test for robustness problems by instrumenting software and monitoring execution. They work by detecting exceptions that arise during development testing of the software. However, they are not able to find robustness failures in situations that are not tested. Additionally, they require access to source code, which is not necessarily available. In contrast, Ballista testing works by sending selected exceptional and acceptable input combinations directly into already-compiled software modules at the module testing level. Thus, Ballista is different from (and potentially complementary to) instrumentation-based robustness improvement tools.

The Ballista approach has been used to compare the robustness of off-the-shelf operating system robustness by automatically testing each of 15 different implementations of the POSIX[IEEE93] operating system application programming interface (API). The results demonstrated that there is significant opportunity for increasing robustness within current operating systems [DeVale99]. Questions left unanswered from the operating system studies were whether other APIs might be better suited to robust implementations, and whether the Ballista approach would work well in other application domains. This paper seeks to contribute to answering those questions.

2.2. General Ballista Robustness Testing

Ballista actively seeks out robustness failures by generating combinational tests of valid and invalid parameter values for system calls and functions. Rather than base testing on the behavioral specification of the function, Ballista instead uses only data type information to generate test cases. Because in many APIs there are fewer distinct data types than functions, this approach tends to scale test development costs sub-linearly with the number of functions to be tested. Additionally, an inheritance approach permits reusing test cases from one application area to another.

As an example of Ballista operation, Figure 1 shows the actual test values used to test the RTI function `rtiAmb.subscribeObjectClassAttributes`, which takes parameters specifying an `RTI::ObjectClassHandle` (which is type-defined to be an `RTI::ULong`), an `RTI::AttributeHandleSet`, and an `RTI::Boolean`. The fact that this particular RTI function takes three parameters of three different data types leads Ballista to draw test values from three separate test objects, each established for one of the three data types. For complete testing, all combinations of test values are used, in this case yielding 25 ULongs * 14 AttributeHandleSets * 12 Booleans = 4200 tests for this function (statistical sampling of combinations can be used for very large test sets, and has been found to be reasonably accurate in finding failure rates compared to exhaustive testing).

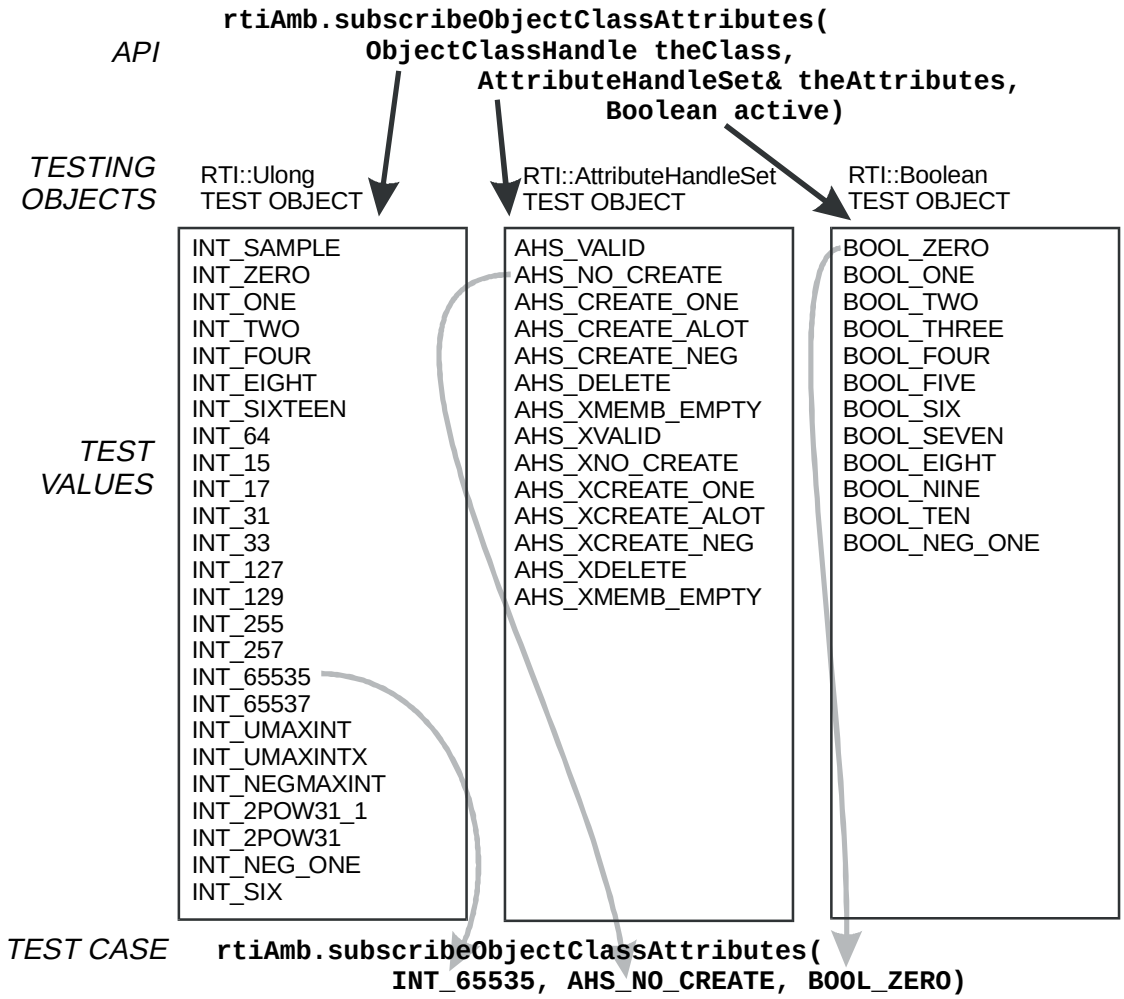

Figure 1: A Ballista example RTI test case for a function that allows the federate to subscribe to a set of object attributes.

A robustness failure is defined within the context of Ballista to be a test case which, when executed, produces a non-robust reaction such as a “hang”, a core dump, or generation of an illegal/undefined exception within the software being tested. In general, this corresponds to an implicit functional specification for all software modules being tested of “doesn’t crash the computer, doesn’t hang, and doesn’t abnormally terminate.” This very general functional specification is the key to minimizing the need for per-function test development effort, because all functions are considered to have a single identical functional specification -- the actual computation performed by any particular function is ignored for robustness testing purposes.

2.3. Enhancements for RTI testing

The previously tested POSIX operating systems represent only a small fraction of the types and variations of COTS software that could potentially benefit from robustness testing. So, a big question in the past has been whether a testing methodology initially developed using an example application of operating system testing would actually work in a different domain. Testing the RTI with Ballista did in fact require extensions to incorporate exception-based error reporting models, testing object-oriented software structures (including callbacks, passing objects by reference, class data types, private class member data, and constructors), addition of test scaffolding code, and operating with a state-rich, distributed software framework.

In previous Ballista testing, any call which resulted in a signal being thrown was considered a robustness failure by the test harness. However, the RTI throws an RTI-defined exception rather than using the POSIX strategy of return codes. This was readily accommodated by including a place for user-defined exception handling to be added to the general Ballista testing harness.

Because the RTI is a distributed system, a certain amount of setup code must be executed to set the distributed system state before a test can be executed. While in the operating system testing all such "scaffolding" was incorporated into constructors and destructors for each test value instance (such as creating a file for a read or write operation), in the RTI there were some function-specific operations required to create reasonable test starting points. While at first it seemed that distinct scaffolding would be required to test each and every RTI function, it turned out that we were able to group the RTI functions into 10 equivalence classes, with each class able to share the same test scaffolding code. This was incorporated into Ballista by inserting optional user-configurable setup and shutdown code to be applied before and after each test case, enabling clean set up and shutdown of the RTI environment for each specific test performed. While such scaffolding code was required for testing the RTI, its development effort was small relative to the number of software modules being tested.

The RTI spec requires that some RTI function calls be able to support a defined callback function. In a typical RTI simulation execution, there are many other simulation processes which need to communicate and share data with each other. For this reason, callbacks are necessary so that other simulation programs know when updates to data occur and so that the RTI can alert all simulations of certain global actions taking place. Testing the RTI showed that the Ballista framework is flexible enough to support the RTI callbacks without interfering with the testing process.

In addition, the RTI contains object oriented features such as passing by reference, user-defined class data types, constructors, and private class member data. In general these were able to be handled in relatively simple ways. Perhaps the most difficult situation that arose was how to test a function that took a pass-by-reference parameter of a class rather than an actual object. This problem was resolved by creating a *pointer* to the class as the data type, and modifying Ballista slightly to accommodate this approach. In general, all the problems that had previously seemed to be large obstacles by the development team and external reviewers alike compared to testing operating systems were resolved with similarly simple adjustments.

3. Experimental Methodology

The current Ballista implementation uses a portable testing client that is downloaded and run on a developer's machine along with the module under test. The client connects to the Ballista testing server at Carnegie Mellon that directs the client's testing of the module under test. This service allows software modules to be automatically and rapidly tested and characterized for robustness failures, and was particularly useful for testing RTI robustness on multiple platforms. In order to test the RTI on Digital Unix and SunOS, it was only necessary to recompile the Ballista client on each target machine, avoiding the need to port the server-side software.

3.1. Interfacing to the RTI for Testing

The HLA is a general-purpose architecture designed to provide a common technical framework to facilitate the reuse and interoperability of all types of software models and simulations. An individual software

simulation or set of simulations developed for one purpose can be applied to another application under the HLA concept of the federation: a composable set of interacting simulations. While the HLA is the architecture, the runtime infrastructure (RTI) software implements federate services to coordinate operations and data exchange during a runtime execution. Access to these services is defined by the HLA, which provides a specification of the functional interfaces between federates and the RTI. The RTI provides services to federates in a way that is analogous to how a distributed operating system provides services to applications.

RTI is a distributed system (Figure 2) that includes two global processes, the RTI Executive (RtiExec) and the Federation Executive (FedExec). The RtiExec's primary purpose is to manage the creation and destruction of federation executions. A FedExec is created for each federation to manage the joining and resigning of federates and perform distribution of reliable updates, interactions, and all RTI internal control messages. The Library (libRTI) implements the HLA API used by C++ programs, and is linked into each federate, with each federate potentially executing on a separate hardware platform.

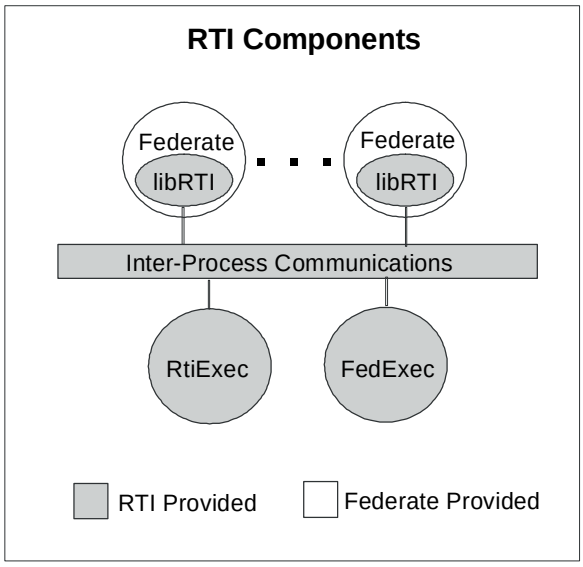

Figure 2: The HLA services are performed via communication between the 2 RTI processes, RtiExec and FedExec, and the federates (simulation programs).

3.2. RTI Testing Approach

A typical RTI function performs some type of data management operation involving either an object, ownership of that object, distribution of an object, a declaration, time management, or management of the federation itself. These function calls typically use complex structures, such as classes, as parameters, making testing the RTI functions more complex than simple operating system calls. Testing the RTI function calls involved creating a very simple application composed of a federation containing only one feder-

ate that was linked, along with the RTI libraries, to the Ballista testing client. However, setting up even this relatively simple system required creating a federation, creating a federate, having the federate join the federation, and so on. In fact, for every test run on every RTI function call, it was necessary to go through the following nine steps:

1. Ensure that the RTI server (RtiExec) was running
2. Create a federation: registers task with the RtiExec and starts up the FedExec process
3. Join the federation (the testing task executes as a federate)
4. Perform any setup functions associated with the current “scaffolding” equivalence class
5. Test the actual function
6. Free any memory that was allocated in the setup functions
7. Resign from the joined federation
8. Destroy the previously created federation to de-register from the RtiExec
9. Shut down the RtiExec if this is the last test or if an error occurred

3.3. Evaluating Test Results

Ballista seeks to determine instances in which exceptional parameter values produce a non-robust response, and is not focussed on the issue of correctness in normal operating conditions. In other words, it tests robustness in exceptional conditions, and is not concerned with whether the result of a function is “correct” (except insofar as the result should be a graceful response to an exceptional situation).

As part of the adaptations for testing the RTI, the previously used “CRASH” scale [Kropp98] had to be modified to account for the fact that the RTI API uses exceptions instead of error return codes used by operating systems, and that the RTI has an internal exception handler that attempts to catch hardware-generated signals and perform a “clean” shutdown rather than a raw core dump. The results for RTI testing fall into the following categories, loosely ranked from best to worst in terms of robustness (only the “Pass” categories are considered to be completely robust responses):

- **Pass** - The function call executed and returned normally, indicating that a nonexceptional combination of parameters happened to be generated.
- **Pass with Exception** - The function call resulted in a valid, HLA-defined exception being thrown, indicating a gracefully caught and handled exceptional condition.
- **RTI Internal Error** - The “RTI Internal Error” exception was thrown and caught, permitting the federate to free memory, resign from, and destroy the federation cleanly. However, the application did not have enough information to perform corrective action. This was a hardware segmentation violation successfully caught by the RTI.
- **Unknown Exception** - An unknown exception was thrown and caught internally to the RTI by a catch-all condition (as opposed to a hardware signal). It behaved similarly to the RTI Internal error, but was software-created instead of hardware-triggered.
- **Abort** - An error occurred that was not caught and the code exited immediately. No memory cleanup, resigning or destroying of the federation was allowed to take place, requiring a manual restart of the entire federation to resume operation.
- **Restart** - The function call did not return after an ample period of time (a “hang”).
- **Catastrophic** - The system was left in a state requiring rebooting the operating system to resume testing. This was experienced in some situations on Digital Unix with RTI 1.0.3, but was apparently due to testing conditions that were not deterministic enough to be tracked down to a particular function call by Ballista testing.

4. Testing Results

Testing was performed on four different versions of the RTI:

- Version 1.0.3 (an early version) for Digital Unix 4.0 on an Alphastation 21164
- Version 1.3.5 (current version) for Digital Unix 4.0 on an Alphastation 21164

- Version 1.3.5 (current version) for SunOS 5.6 on a Sparc Ultra-30
- Version 1.3NG (commercial version in beta test) for SunOS 5.6 on a Sparc Ultra-30

Overall a total of 88,296 data points was collected. This number depends on several factors: 1) the number of functions to be tested, 2) the number of parameters in each function, 3) the data types of the arguments, and 4) sampling for functions with very large test sets. Significant changes were made to the RTI functions in going from version 1.0.3 to 1.3.5, including adding functions and changing the parameter lists taken by existing functions. In addition, since the RTI 1.3 NG version was not developed by the DMSO as the other RTI versions were, there are likely to be several implementation differences.

4.1. Normalized Failure Rates

Table 1 reports the directly measured robustness failure rates. While it is tempting to simply use the raw number of tests that fail as a comparative metric, this approach is problematic. Versions 1.0.3 and 1.3.5 contain several completely different functions with different numbers and types of parameters. In addition, a single RTI function with a large number of test cases could significantly affect both the number of failures, and the ratio of failures to tests executed. Similarly, an RTI function with few test cases would have minimal effect on raw failure rates even if it demonstrated a large percentage of failures. Thus a normalized failure rate metric is reported in the last column of Table 1.

The normalized failure rate [Kropp98] first determines the proportion of robustness failures across tests for each function within each system being tested, then produces a uniformly weighted average across all the functions, permitting comparisons of the three implementations according to a single normalized metric. If one of the RTI implementations were being tested with a specific simulation program running, weightings would be used to reflect the dynamic frequency of calling each function to give an exposure metric that is potentially more accurate. In the absence of a particular workload, and given our experience that weighted failure rates vary dramatically depending on the workload (but were in some cases as high as 29% for the POSIX API), it is inappropriate for us to simply take an arbitrary application operating profile and use it here. However, as a simple common-sense check on these results, functions with high robust-

ness failure rates do in fact include commonly used features such as registering an object, publishing data, subscribing to data, and determining attribute ownership.

Table 1: Directly measured robustness failure rates.

	Functions tested	Functions with RTI Internal Error Exception Failures	Functions with Unknown Exception failures	Functions with Abort failures	Functions with no failures	Number of tests run	Number of RTI Internal Error Exception failures	Number of Unknown Exception failures	Number of Abort failures	Normalized (sum all aborts) rate
RTI 1.0.3 Dunix	76	19	18	0	41	40291	136	2611	0	6.41%
RTI 1.3.5 Dunix	86	20	32	0	43	22757	1255	1965	0	10.20%
RTI 1.3.5 SunOS	86	0	0	45	41	14291	0	0	2289	10.05%
RTI 1.3NG SunOS	84	4	0	43	40	10957	41	0	1444	8.44%

4.2. RTI 1.0.3 for Digital Unix

The types of robustness failures that were detected in RTI version 1.0.3 were RTI Internal Error and Unknown exceptions. Some of the RTI service functions have the ability to throw the exception "`RTIinternalError : Caught unknown exception`." However, in speaking with one of the developers, we learned that this is not supposed to ever occur. A more specific exception should have been thrown instead, rather than making the "RTIinternalError" exception serve as a "catch all" condition or default handler. This type of failure accounted for a 1.4% normalized failure rate, while the Unknown exceptions accounted for a 5.0% normalized failure rate.

As can be seen from Figure 3, the three functions `RTI::AttributeHandleSet->setUnion`, `RTI::AttributeHandleSet->removeSetIntersection`, and `RTI::AttributeHandleSet->setIntersection` responded the least robustly to our tests. All three of these functions took as their sole parameter an `RTI::AttributeHandleSet` class, and all three failed on exactly the same input parameters. In fact, all but one RTI 1.0.3 function we tested that had a failure rate of more than 20% took an `RTI::AttributeHandleSet` class as a parameter (labeled with "*****" in Figure 3). The

lower failure rates of the two functions with a "*****" at 20% were due to masking by a successful second exceptional parameter check before the `RTI::AttributeHandleSet` parameter was touched by the function.

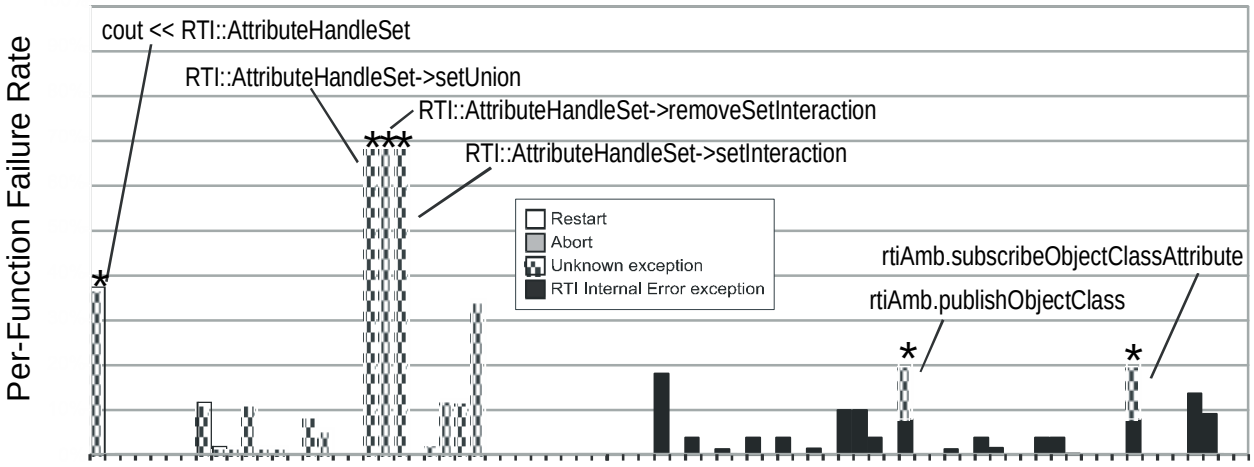

Figure 3: RTI 1.0.3 experienced failures of types RTI Internal Error and Unknown exceptions. Functions marked with "*" all took an RTI::AttributeHandleSet class as a parameter and all got failure rates greater than 20%.

An additional problem discovered while testing was the RTI client process crashing through an RTI service function call. This would occur randomly and the direct cause was never determined. While running a simulation, the following error would occur "`RTIinternalError: Invalid mutex object in RTIlocker::RTIlocker 14001`" for any RTI service function call made. Once in this error state it was impossible to run a federation execution until the machine was rebooted. This error is particularly nasty because it not only forces the user to quit the currently running federation execution without performing any memory clean up or shutdown code, but also requires rebooting the machine before any other RTI function can be executed. This particular problem was not encountered in the two RTI 1.3.5 versions.

4.3. RTI 1.3.5 for Digital Unix and SunOS

The types of robustness failures detected in the two RTI 1.3.5 versions were quite different in manifestation, but similar in profile (Figures 4 and 5). For the robustness failures that were detected in the Digital

Unix port, RTI Internal Errors accounted for a 2.6% normalized robustness failure rate, Unknown exceptions accounted for a 6.5% normalized robustness failure rate, Restarts obtained a 1.1% normalized robustness failure rate, and one test (0.02% normalized robustness failure rate, counted as an Abort), produced an exception system infinite loop failure which printed out the following 3-line message and terminated execution:

```
"Exception system exiting dues[sic] to multiple internal errors:
     exception dispatch or unwind stuck in infinite loop
     exception dispatch or unwind stuck in infinite loop".
```

In comparison, the robustness failures seen in RTI 1.3.5 for SunOS did not include RTI Internal Error or Unknown exceptions. Instead, two different reactions to exceptional inputs were seen. The first was a segmentation fault that would cause the federate process to exit immediately without properly resigning from and destroying the federation and cleaning up memory. This would result in a "zombie" federate registered with the federation executive. The presence of such zombies caused the subsequently joining federate (the next automatic test case in our situation) to hang in the `rtiAmb.joinFederationExecution` service. To remedy this it was necessary to manually resign from the federation by killing the FedExec and RtiExec processes. The other reaction to an exceptional input that was observed in the SunOS testing was similar to that segmentation fault, except instead of printing out "Segmentation fault" the following message would be displayed followed by execution termination:

```
"Run-time exception error; current exception: RTI internal error
     Unexpected exception thrown."
```

which appears to indicate an incomplete implementation of an RTI Internal Error. Both of these errors were considered to be Aborts, and accounted for an 8.9% normalized robustness failure rate. Restart failures accounted for a 1.1% normalized robustness failure rate.

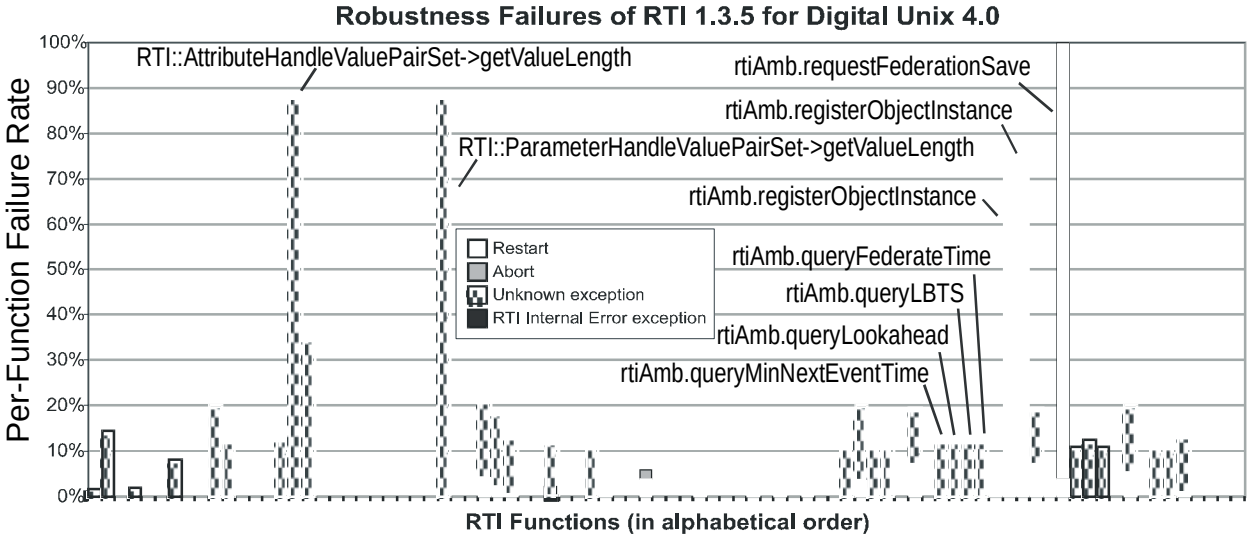

Figure 4: In addition to RTI Internal Error and Unknown Exceptions, RTI 1.3.5 for Digital Unix also had one function that experienced multiple restarts, and one function trigger the “stuck in infinite loop” error message.

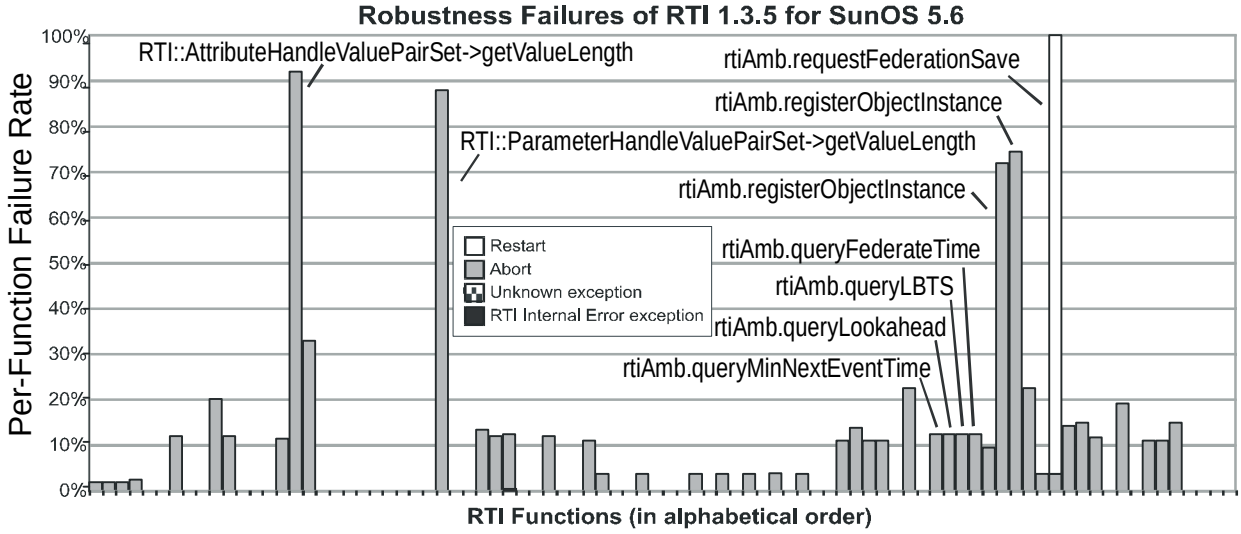

Figure 5: The RTI 1.3.5 for SunOS obtained Abort failures in the form of segmentation faults instead of RTI Internal Errors or Unknown exceptions.

4.3.1. Segmentation Faults vs. RTI Internal Error Exception

Comparing the two RTI 1.3.5 graphs shows that the robustness failure rates are essentially the same. However, in the SunOS port, any unanticipated signal apparently leaked through and was seen as a segmentation fault instead of being caught as an RTI Internal Error as in Digital Unix. The RTI Internal Error seen in the Digital Unix version allows recovery and cleanup, unlike a raw segmentation fault, which aborts the code. The SunOS version's inability to catch and handle segmentation faults could significantly disrupt the currently running federation execution because, due to the RTI design, other federates will not be informed properly that one federate has left. And, we can only speculate as to what might happen to the data that federate was sharing, and consequent effects on the rest of the distributed simulation. This example serves to illustrate possible problems in porting robust applications across platforms with different exception handling support.

4.3.2. Restart Failures

In both implementations, the Restart failures all occurred for the single-parameter function: `rtiAmb.requestFederationSave`. On both Digital Unix and SunOS, 50 of 52 tests hung. As an experiment, we let the `rtiAmb.requestFederationSave` function run for 8 hours, but it remained hung. It is interesting to note that both the two-parameter `rtiAmb.requestFederationSave` function of RTI 1.3.5 and the `rtiAmb.requestFederationSave` function in RTI version 1.0.3 did not have any Restarts, although there were significant changes in this function from version 1.0.3 to version 1.3.5.

4.3.3. Clusters of Similar Results

In both implementations of RTI 1.3.5 we see clusters of results where the normalized failure rates appear to be exactly the same. For example, the four sequential functions seen in both Figures 4 and 5, `rtiAmb.queryFederateTime`, `rtiAmb.queryLBTS`, `rtiAmb.queryLookahead`, and `rtiAmb.queryMinNextEventTime` all have failure rates of approximately 12%. These four functions all took an `RTI::FedTime` class as their only parameter. Eight tests were run on each of these

functions, and of these 8 the same test, DECODE_NEG, failed in the same way for all four of these functions on both implementations. By their names alone we can tell that these four functions are very similar - they all query-*something*. It is likely that they all use very similar code if not the exact same parent code. One fix to that code to check for the DECODE_NEG condition might eliminate these 4 functions as robustness problems. However, from a user/robustness point of view, we measure susceptibility to robustness failures rather than attempt to predict defect densities, so it is reasonable to charge all four functions with having robustness problems.

4.4. RTI Version 1.3NG for SunOS

Ballista robustness testing was also performed on the RTI 1.3NG (Next Generation). RTI 1.3NG is built to the same interface specification as RTI 1.3.5 and attempts to provide all of the services defined by the HLA 1.3 Interface Specification, However RTI 1.3NG was written by a group of commercial developers and not by the government labs (DMSO) that built all the other RTIs. This group of commercial developers includes Science Applications International Corp. (SAIC), Virtual Technology Corporation (VTC), Object Sciences Corporation (OSC), and Dynamic Animation Systems (DAS). RTI 1.3NG is in beta test at this time, which increases the potential benefit from Ballista testing.

RTI 1.3NG is currently only available on a limited number of platforms. We tested the SunOS 5.6 implementation which, interestingly, provides us with two implementations of the RTI 1.3 on the same platform built by two separate development teams. The types of robustness failures that were detected in RTI 1.3NG were RTI Internal Error exceptions, which accounted for a 1.1% normalized failure rate, and Abort failures in the form of segmentation faults which accounted for an 7.1% normalized failure rate.

Right off the bat we see that these results are different than the DMSO's RTI 1.3.5 version because the NG version produced RTI Internal Errors whereas the DMSO's version did not. In addition, no Restarts were encountered in RTI 1.3NG. For the most part, the results of testing RTI 1.3 NG were very similar to the DMSO's RTI 1.3.5 for SunOS. However, as can be seen in Figure 6, there were a few functions which exhibited quite different behavior.

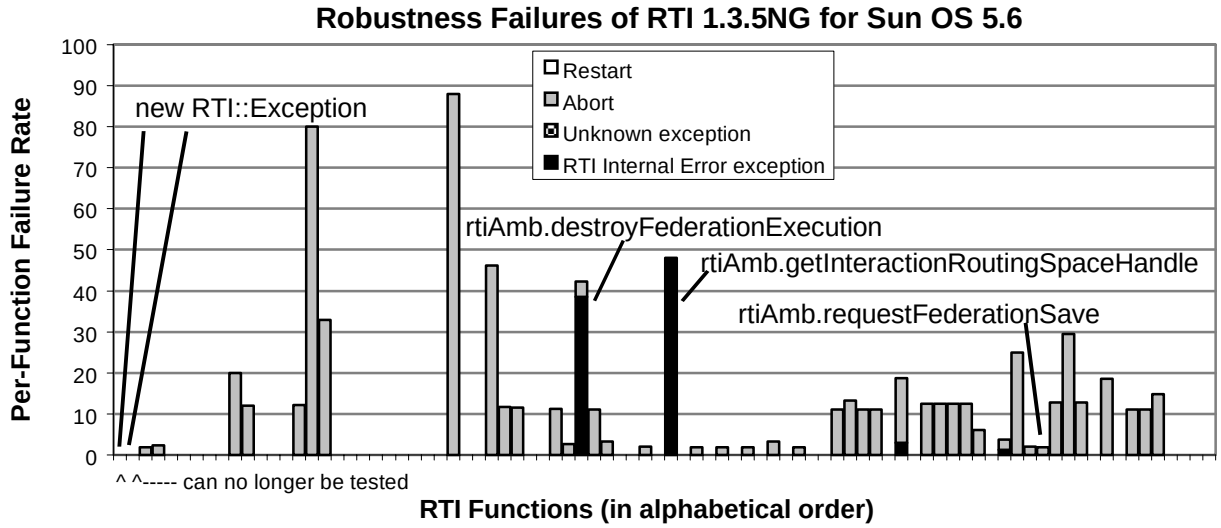

Figure 6: The RTI 1.3NG for SunOS obtained RTI Internal Error exceptions and Abort failures in the form of segmentation faults. Two functions can no longer be tested due to changes in the implementation.

4.4.1. RTI Internal Error Exception Sighting

The major difference seen in the results from testing RTI 1.3NG and the DMSO's RTI 1.3.5 is that while in RTI 1.3.5 robustness failures manifested themselves as segmentation fault Aborts, the RTI 1.3NG displayed the ability to also throw the RTI Internal Error exception which was not seen in the SunOS version of RTI 1.3.5. The function `rtiAmb.getInteractionRoutingSpaceHandle` threw this exception in almost 50% of the tests performed on it. However it appears that there may be some confusion in throwing this exception because when it occured for this function it is labelled as an `"Unexpected Exception!"` and displays the following message:

```
"RTIinternalError: "/vobs/rting/rti/priv/pkg/presMgt/priv/src/
RtiAmbassador2.cpp", line  2152: Unexpected exception!  Interac-
tionClassNotKnown: "/vobs/rting/rti/priv/pkg/fedDatabase/priv/
src/RtiFedDatabase.cpp", line 1417:"
```

It is unclear why this function did not simply throw the `"InteractionClassNotKnown"` exception instead since there was enough information to include reference to it in its output message. In addi-

tion, the function `rtiAmb.destroyFederationExecution` got an RTI Internal Error exception for almost 40% of the tests performed on it. In every case this was due to an empty string being passed to the function which resulted in the following message being displayed:

```
RTIinternalError: "/vobs/rting/rti/priv/pkg/presMgt/priv/src/
RtiAmbassador.cpp", line 255: Sorry, the use of a empty federation
name is not supported.
```

In this case, a simple check for an empty string could easily eliminate the robustness failures. Another function which also threw the RTI Internal Error exception in response to a null pointer being passed into the function being tested indicated that the cause was indeed due to a null pointer. Based on these examples of the occurence of the RTI Internal Error exception it is clear that enough information is available to determine the cause of and eliminate several robustness failures.

4.4.2. The “Save” Function and No Restarts

The only function in any of our previous tests which got Restart failures was the `rtiAmb.requestFederationSave` function which occurred in both of the RTI 1.3.5 versions. In the RTI 1.3NG version we see that this function does not get any Restart failures. Executing this function exactly the same way on both RTI 1.3.5 and RTI 1.3NG produces very dissimilar results and indicates a robustness improvement in the NG version which can most likely be attributed to the fact that this function was reworked for the RTI 1.3NG version.

4.4.3. Changes in the Implementation

In the course of testing, a few differences in the two RTI 1.3 implementations have become apparent. The main difference observered affects the two constructor functions for the `RTI::Exception` class. In RTI 1.3NG these two functions can no longer be directly tested because two pure virtual functions were added to this class making it an abstract class. We are still able to test these functions indirectly by testing the functions from another class which inherits functionality from the base `RTI::Exception` class. However this represents a change in the RTI 1.3 implementation.

4.5. Comparison to Operating System Results

The results obtained from RTI testing are much more robust than those we obtained testing POSIX operating systems, which typically had a robustness failure rate between 10.0% to 22.7% [Koopman99], compared to the RTI implementations which got between 6.4% and 10.2%. Several of the operating systems had catastrophic errors occur, which are failures that occur when the entire OS becomes corrupted or the machine crashes or reboots. In addition, almost every OS of the 15 tested encountered several functions that had Restart failures, whereas only one function in RTI 1.3.5 had Restart failures. We anticipated the results of Ballista testing of the RTI would indicate a lower failure rate than previous testing of operating systems primarily because the RTI, as well as the HLA, were specifically designed for robust operation.

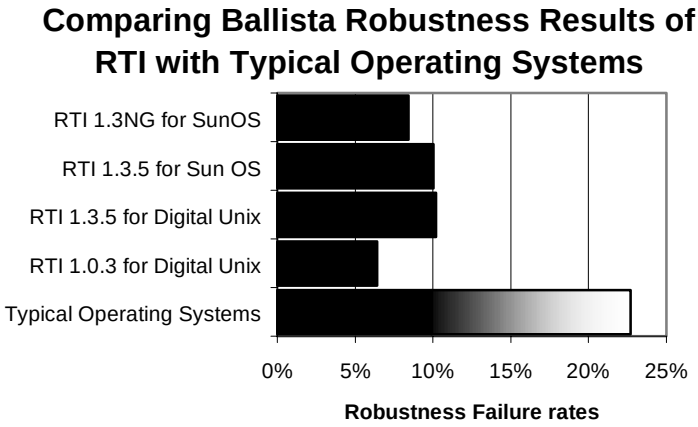

Figure 7: Overall comparison of failure rates of 4 implementations of the RTI to operating systems (OS failure rates ranged from 10.0% to 22.7%).

The average number of functions per RTI implementation that had no robustness failures is 49.8%. (In other words, about half of the functions were failure-free). This is similar to the operating system result of 48.3% functions being robustness failure-free. A breakdown of this is shown in Table 2.

System	Fns. Tested	Fns. With No Failures	% Fns. With No Failures
RTI 1.0.3 Digital Unix 4.0	76	41	54%
RTI 1.3.5 Digital Unix 4.0	86	43	50%
RTI 1.3.5 SunOS 5.6	86	41	48%
RTI 1.3NG SunOS 5.6	84	40	48%
AIX 4.1	186	108	58%
FreeBSD 2.2.5	175	77	44%
HPUX 9.05	186	98	53%
HPUX 10.20	185	92	50%
IRIX 5.3	189	90	48%
IRIX 6.2	225	131	58%
Linux 2.0.18	190	104	55%
Lynx 2.4.0	222	114	51%
NetBSD 1.3	182	83	46%
Digital Unix 3.2	232	96	41%
Digital Unix 4.0	233	109	47%
QNX 4.22	203	75	37%
QNX 4.24	206	77	37%
SunOS 4.13	189	85	45%
SunOS 5.5	233	129	55%

Table 2: Percent of RTI and operating system functions with no failures.

If we take a closer look at the data, we see that with the RTI functions there were typically a handful of functions, 5 or 6 depending on which implementation, which responded poorly (failure rate above 30%) and the rest had robustness failure levels down around 10-20%. In contrast, the OSes typically had more of a consistent distribution of failures across the functions with, taking Digital Unix 4.0 as a typical example, around 22% of the functions having failure rates above 30%. Thus we know that RTI has overall more robust functions and, for example, if the six least robust functions in RTI 1.3.5 for Digital Unix are removed from the calculation, the failure rate of RTI 1.3.5 for Digital Unix would go down from 10.2% to 4.88%.

4.6. Operational Profiling?

It is common in software reliability work to use an operational profile for weighting the severity of various problems encountered according to the expected execution frequency of functions (*e.g.*, [Musa96]).

Unfortunately, for the RTI, and indeed many general-purpose APIs, this type of profiling data is highly dependent which federation program(s) are running. While we did not have access to realistic RTI programs because that environment itself is still new, data on previous operating system testing showed that operational profile weightings still resulted in significant (10% or more) weighted robustness failure rates [Koopman99]. Additionally there is the issue that an operational profile for a normally running program is not necessarily applicable to exceptional situations (which are, by definition, abnormal). Thus we do not give detailed weighted failure rate results here, because to do so would risk inviting unwarranted, generalized conclusions by readers drawn from what would be very specific data.

5. Conclusions & Future Directions

This paper provides the results of Ballista robustness testing of the High Level Architecture Run-Time Infrastructure (HLA RTI), a general-purpose distributed simulation backplane which was specifically designed for robust exception handling. Testing the RTI required significant extensions of Ballista capabilities that were thought by some to be improbable to accomplish, including handling exception-based error reporting models, testing object-oriented software structures (including callbacks), incorporating necessary state-setting "scaffolding" code in a scalable manner, and operating in a state-rich distributed system environment. Moreover, these extensions were accommodated through small, natural evolutions of the basic Ballista architecture. This bodes well for extending Ballista to other application areas as well, according to the project goal of creating a general-purpose, scalable testing framework.

Robustness testing was performed on four different versions of the RTI, with a total of 88,296 data points collected. With a 6.4% to 10.2% normalized robustness failure rate, RTI appears to be significantly more robust than off-the-shelf POSIX operating systems, which had 10% to 22.7% normalized failure rates. As with operating system testing results, certain types of functions were robustness "bottlenecks," having significantly higher failure rates than most other functions. Thus, these testing results should aid in deciding how to allocate developer resources to improve robustness.

The particular robustness problems observed in four version/platform RTI pairs were internal exception handling errors (actually, a gracefully caught segmentation violation) ranging from a 1.1% to a 2.6% normalized failure rate, unknown exceptions (an exception handling software defect) found only on the two Digital Unix ports, with 5.0% to 6.5% normalized failure rates, and segmentation faults (exceptions that evaded the exception handlers) found only on the two SunOS ports, ranging from a 7.1% to a 8.9% normalized failure rate. Additionally, the Digital Unix port of RTI 1.3.5 suffered "multiple internal errors" on one particular function that required killing the testing task. Similarities in results obtained from testing the RTI 1.3.5 for SunOS with its commercial counterpart RTI 1.3NG for SunOS demonstrated common robustness failure traps that programmers can fall into. Unlike any other version tested, the RTI 1.3NG encountered both internal exception handling errors and segmentation faults. Finally, the Digital Unix port of RTI 1.0.3 could fail in a way that required rebooting the system to correct. All problems except for the RTI 1.0.3 reboot issue and the one "multiple internal errors" result were readily reproducible and were automatically reduced to simple "bug report" programs by the Ballista server.

These results indicate that it can be a difficult task to create "bullet-proof" code, even when that is a specifically stated development goal. Additionally, the problem with the SunOS port not catching segmentation faults indicates that it can be difficult to provide comparable exception handling capabilities for the same API across multiple platforms. One piece of good news, however, is that (except for the SunOS problem just mentioned), we did not find significant differences in exception handling coverage across platforms. This suggests, but certainly does not prove, that underlying variations in operating system robustness might not percolate up through well-written exception handling facilities to cause exception handling differences across platforms that would further complicate the task of writing portable, robust applications.

In the future, we are working to make Ballista part of the standard verification suite for RTI development. Additionally, we plan to explore issues of concurrent testing to find potentially more subtle bugs related to timing and resource sharing. However, even with a relatively straightforward static, single-

thread execution model, Ballista testing has been demonstrated to find exception handling problems in software specifically written to be highly robust.

6. Acknowledgements

This research was sponsored by DARPA contract DABT63-96-C-0064.

7. References

[Barton90] Barton, J., Czeck, E., Segall, Z., Siewiorek, D., “Fault injection experiments using FIAT,” *IEEE Transactions on Computers*, 39(4): 575-82, April 1990.

[Carette96] Carette, G., “CRASHME: Random input testing,” (no formal publication available) http://people.delphi.com/gjc/crashme.html, accessed 4/23/99.

[Cristian95] Cristian, Flaviu, "Exception Handling and Tolerance of Software Faults," In: *Software Fault Tolerance*, Michael R. Lyu (Ed.). Chichester: Wiley, 1995. pp. 81-107, Ch. 4.

[DMSO99] Defense Modeling and Simulation Office (DMSO). *HLA Homepage General Information*. http://hla.dmso.mil/hla/general/, accessed 4/1/99.

[HLA_IS98] U.S. Department of Defense, High Level Architecture, *Interface Specification version 1.3*, April 2, 1998. Available at http://hla.dmso.mil/hla/tech/ifspec/ifspec-d01-body.pdf.

[HLA_IS97] U.S. Department of Defense, High Level Architecture, *Interface Specification version 1.1*, Feb. 12, 1997. Available at http://hla.dmso.mil/hla/tech/ifspec/ifsp11.pdf.

[HLA_Prog98] Department of Defense, High Level Architecture, Run-Time Infrastructure, *Programmer’s Guide*, RTI 1.3 Version 5, Dec. 16, 1998, DMSO, MITRE, SAIC, Virtual Technology Corporation.

[HLA_Prog97] Department of Defense, High Level Architecture, Run-Time Infrastructure, *Programmer's Guide*, RTI 1.0 Version 3, Nov. 14, 1997, DMSO, MITRE, SAIC, Virtual Technology Corporation.

[HLA_RefA98] High Level Architecture, RTI 1.3 Version 5, *RTIambassador [API] Reference Manual*, Dec. 16, 1998. Available at http://www.dmso.mil/cgi-bin/hla_dev/hla_cat.pl.

[HLA_RefC98] High Level Architecture, RTI 1.3 Version 5, *Supporting Classes Reference Manual*, Dec. 16, 1998. Available at http://www.dmso.mil/cgi-bin/hla_dev/hla_cat.pl.

[IEEE90] *IEEE Standard Glossary of Software Engineering Terminology*, IEEE Std 610.12-1990, IEEE Computer Soc., Dec. 10, 1990.

[IEEE93] *IEEE Standard for Information Technology - Portable Operating System Interface (POSIX) Part 1: system Application Program Interface (API) Amendment 1: Realtime Extension [C language]*, IEEE Std 1003.1b-1993, IEEE Computer Society, 1994.

[Kanawati92] Kanawati, G., Kanawati, N. & Abraham, J., "FERRARI: a tool for the validation of system dependability properties," *1992 IEEE Workshop on Fault-Tolerant Parallel and Distributed Systems*. Amherst, MA, USA, July 1992, pp. 336-344.

[Koopman99] Koopman, P. & DeVale, J., "The Exception Handling Effectiveness of POSIX Operating Systems," submitted to *IEEE Transactions on Software Engineering*.

[Kropp98] Kropp, N., Koopman, P. & Siewiorek, D., "Automated Robustness Testing of Off-the-Shelf Software Components," *28th Fault Tolerant Computing Symposium*, pp. 230-239, June 23-25, 1998.

[Miller98] Miller, B., Koski, D., Lee, C., Magnanty, V., Murthy, R., Natarajan, A. & Steidl, J., *Fuzz Revisited: A Re-examination of the Reliability of UNIX Utilities and Services*, Computer Science Technical Report 1268, Univ. of Wisconsin-Madison, May 1998.

[Musa96] Musa, J., Fuoco, G., Irving, N. & Kropfl, D., Juhlin, B., “The Operational Profile”, in: Lyu, M.(ed.), *Handbook of Software Reliability Engineering*, McGraw-Hill/IEEE Computer Society Press, Los Alamitos CA, 1996, pp. 167-216.

[Numega] Numega. BoundsChecker. http://www.nemega.com/products/vc/vc.html.

[Parasoft] Parasoft. Insure++. http://www.parasoft.com/insure/index.html.

[Pure Atria] Pure Atria. Purify. http://www.pureatria.com/products/purify/index.html.

[Tsai95] Tsai, T., & R. Iyer, “Measuring Fault Tolerance with the FTAPE Fault Injection Tool,” *Proceedings eighth International Conference on Modeling Techniques and Tools for Computer Performance Evaluation*, Heidelberg, Germany, Sept. 20-22 1995, Springer-Verlag, pp. 26-40.

8. Appendices

- Appendix A contains a full listing of the results of testing RTI 1.0.3.
- Appendix B contains a full listing of the results of testing RTI 1.3.5 for Digital Unix.
- Appendix C contains a full listing of the results of testing RTI 1.3.5 for Sun OS.
- Appendix D contains a full listing of the results of testing RTI 1.3NG.

Appendix A: Results for RTI 1.0.3

Function	number of parameters	number of tests run	% of tests that passed clean	% of tests that passed with an exception	% of tests that got RTI Internal Error exception	% of tests that got Unknown exception	% of tests that got Aborts	% of tests that got Restarts
cout << RTI::AttributeHandleSet	1	16	62.50	0.00	0.00	37.50	0.00	0.00
cout << RTI::Boolean	1	12	100.00	0.00	0.00	0.00	0.00	0.00
cout << RTI::FederateStateType	1	7	100.00	0.00	0.00	0.00	0.00	0.00
cout << RTI::FederationStateType	1	12	100.00	0.00	0.00	0.00	0.00	0.00
cout << RTI::OrderType	1	7	100.00	0.00	0.00	0.00	0.00	0.00
cout << RTI::TimeManagementStateType	1	12	100.00	0.00	0.00	0.00	0.00	0.00
cout << RTI::TransportType	1	7	100.00	0.00	0.00	0.00	0.00	0.00
new RTI::Exception	1	17	88.24	0.00	0.00	11.76	0.00	0.00
new RTI::Exception	1	52	98.08	0.00	0.00	1.92	0.00	0.00
new RTI::Exception	2	1248	98.08	0.00	0.00	1.92	0.00	0.00
new RTI::RegionNotKnown	1	17	88.24	0.00	0.00	11.76	0.00	0.00
new RTI::RegionNotKnown	1	52	98.08	0.00	0.00	1.92	0.00	0.00
new RTI::RegionNotKnown	2	1248	98.08	0.00	0.00	1.92	0.00	0.00
RTI::AttributeHandleSet->add	1	22	86.36	13.64	0.00	0.00	0.00	0.00
RTI::ATtributeHandleSet->decode	1	11	90.91	0.00	0.00	9.09	0.00	0.00
RTI::ATtributeHandleSet->encode	1	51	94.12	0.00	0.00	5.88	0.00	0.00
RTI::ATtributeHandleSet->getHandle	1	24	4.17	95.83	0.00	0.00	0.00	0.00
RTI::ATtributeHandleSet->isMember	1	9	100.00	0.00	0.00	0.00	0.00	0.00
RTI::ATtributeHandleSet->removeSetIntersection	1	16	31.25	0.00	0.00	68.75	0.00	0.00
RTI::ATtributeHandleSet->setIntersection	1	16	31.25	0.00	0.00	68.75	0.00	0.00
RTI::ATtributeHandleSet->setUnion	1	16	31.25	0.00	0.00	68.75	0.00	0.00
RTI::AttributeHandleSetFactory::create	1	24	100.00	0.00	0.00	0.00	0.00	0.00
RTI::AttributeHandleValuePairSet->add	3	2015	69.13	28.19	0.00	2.68	0.00	0.00
RTI::AttributeHandleValuePairSet->getHandle	1	24	4.17	83.33	0.00	12.50	0.00	0.00
RTI::AttributeHandleValuePairSet->getValue	3	18367	3.90	83.61	0.00	12.49	0.00	0.00
RTI::AttributeHandleValuePairSet->moveFrom	2	216	1.85	63.43	0.00	34.72	0.00	0.00
RTI::AttributeHandleValuePairSet->next	1	24	100.00	0.00	0.00	0.00	0.00	0.00
RTI::AttributeHandleValuePairSet->valid	1	24	100.00	0.00	0.00	0.00	0.00	0.00
RTI::AttributeSetFactory::create	1	24	66.67	33.33	0.00	0.00	0.00	0.00
RTI::FederateHandleSetFactory::create	1	24	66.67	33.33	0.00	0.00	0.00	0.00
RTI::FederateHandleValuePairSet->add	1	22	100.00	0.00	0.00	0.00	0.00	0.00
RTI::FederateHandleValuePairSet->getHandle	1	24	4.17	95.83	0.00	0.00	0.00	0.00
RTI::ParameterSetFactory::create	1	24	66.67	33.33	0.00	0.00	0.00	0.00
rtiAmb.changeAttributeOrderType	3	2016	0.00	100.00	0.00	0.00	0.00	0.00
rtiAmb.changeAttributeTransportType	3	840	1.19	98.81	0.00	0.00	0.00	0.00
rtiAmb.changeInteractionOrderType	2	154	6.49	93.51	0.00	0.00	0.00	0.00
rtiAmb.changeInteractionTransportType	2	154	6.49	93.51	0.00	0.00	0.00	0.00

Function	number of parameters	number of tests run	% of tests that passed clean	% of tests that passed with an exception	% of tests that got RTI Internal Error exception	% of tests that got Unknown exception	% of tests that got Aborts	% of tests that got Restarts
rtiAmb.createFederationExecution	1	11	36.36	45.45	18.18	0.00	0.00	0.00
rtiAmb.dequeueFIFOasynchronously	1	12	100.00	0.00	0.00	0.00	0.00	0.00
rtiAmb.destroyFederationExecution	1	52	0.00	96.15	3.85	0.00	0.00	0.00
rtiAmb.flushQueueRequest	1	10	90.00	10.00	0.00	0.00	0.00	0.00
rtiAmb.getAttributeHandle	2	1144	0.00	98.78	1.22	0.00	0.00	0.00
rtiAmb.getAttributeName	2	484	6.40	93.60	0.00	0.00	0.00	0.00
rtiAmb.getInteractionClassHandle	1	52	0.00	96.15	3.85	0.00	0.00	0.00
rtiAmb.getInteractionClassName	1	22	50.00	50.00	0.00	0.00	0.00	0.00
rtiAmb.getObjectClassHandle	1	52	0.00	96.15	3.85	0.00	0.00	0.00
rtiAmb.getObjectClassName	1	22	31.82	68.18	0.00	0.00	0.00	0.00
rtiAmb.getParameterHandle	2	1144	0.00	98.60	1.40	0.00	0.00	0.00
rtiAmb.getParameterName	2	484	7.44	92.56	0.00	0.00	0.00	0.00
rtiAmb.nextEventRequest	1	10	90.00	0.00	10.00	0.00	0.00	0.00
rtiAmb.nextEventRequestAvailable	1	10	90.00	0.00	10.00	0.00	0.00	0.00
rtiAmb.pauseAchieved	1	52	32.69	63.46	3.85	0.00	0.00	0.00
rtiAmb.publishInteractionClass	1	22	50.00	50.00	0.00	0.00	0.00	0.00
rtiAmb.publishObjectClass	2	352	20.17	59.38	7.95	12.50	0.00	0.00
rtiAmb.registerObject	2	528	13.64	86.36	0.00	0.00	0.00	0.00
rtiAmb.requestAttributeOwnershipAcquisition	3	624	2.72	97.28	0.00	0.00	0.00	0.00
rtiAmb.requestAttributeOwnershipDivestiture	4	664	15.81	82.83	1.36	0.00	0.00	0.00
rtiAmb.requestClassAttributeValueUpdate	2	264	31.82	68.18	0.00	0.00	0.00	0.00
rtiAmb.requestFederationSave	1	52	96.15	0.00	3.85	0.00	0.00	0.00
rtiAmb.requestFederationSave	2	520	38.65	59.81	1.54	0.00	0.00	0.00
rtiAmb.requestID	3	3168	100.00	0.00	0.00	0.00	0.00	0.00
rtiAmb.requestObjectAttributeValueUpdate	2	120	100.00	0.00	0.00	0.00	0.00	0.00
rtiAmb.requestPause	1	52	96.15	0.00	3.85	0.00	0.00	0.00
rtiAmb.requestRestore	1	52	0.00	96.15	3.85	0.00	0.00	0.00
rtiAmb.sendInteraction	4	2860	21.12	78.53	0.35	0.00	0.00	0.00
rtiAmb.setLookahead	1	10	90.00	10.00	0.00	0.00	0.00	0.00
rtiAmb.setTimeConstrained	1	12	100.00	0.00	0.00	0.00	0.00	0.00
rtiAmb.subscribeInteractionClass	1	22	50.00	50.00	0.00	0.00	0.00	0.00
rtiAmb.subscribeObjectClassAttribute	2	352	20.17	59.38	7.95	12.50	0.00	0.00
rtiAmb.tick	2	100	100.00	0.00	0.00	0.00	0.00	0.00
rtiAmb.timeAdvanceRequest	1	10	90.00	10.00	0.00	0.00	0.00	0.00
rtiAmb.timeAdvanceRequestAvailable	1	10	90.00	10.00	0.00	0.00	0.00	0.00
rtiAmb.unpublishInteractionClass	1	22	36.36	50.00	13.64	0.00	0.00	0.00
rtiAmb.unpublishObjectClass	1	22	22.73	68.18	9.09	0.00	0.00	0.00
rtiAmb.unsubscribeInteractionClass	1	22	50.00	50.00	0.00	0.00	0.00	0.00
rtiAmb.unsubscribeObjectClassAttribute	1	22	31.82	68.18	0.00	0.00	0.00	0.00

Appendix B: Results for RTI 1.3.5 for Digital Unix

Function	number of parameters	number of tests run	% of tests that passed clean	% of tests that passed with an exception	% of tests that got RTI Internal Error exception	% of tests that got Unknown exception	% of tests that got Aborts	% of tests that got Restarts
new RTI::Exception	1	59	98.31	0.00	0.00	1.69	0.00	0.00
new RTI::Exception	2	174	85.63	0.00	0.00	14.37	0.00	0.00
new RTI::RegionNotKnown	1	2	100.00	0.00	0.00	0.00	0.00	0.00
new RTI::RegionNotKnown	2	1182	98.14	0.00	0.00	1.86	0.00	0.00
RTI::AttributeHandleSet->add	1	25	72.00	28.00	0.00	0.00	0.00	0.00
RTI::AttributeHandleSet->getHandle	1	24	4.17	95.83	0.00	0.00	0.00	0.00
RTI::AttributeHandleSet->isMember	1	25	92.00	0.00	0.00	8.00	0.00	0.00
RTI::AttributeHandleSet->remove	1	25	72.00	28.00	0.00	0.00	0.00	0.00
RTI::AttributeHandleSetFactory::create	1	24	100.00	0.00	0.00	0.00	0.00	0.00
RTI::AttributeHandleValuePairSet->add	3	1605	79.94	0.00	0.00	20.06	0.00	0.00
RTI::AttributeHandleValuePairSet->getHandle	1	24	4.17	83.33	0.00	12.50	0.00	0.00
RTI::AttributeHandleValuePairSet->getOrderType	1	25	0.00	100.00	0.00	0.00	0.00	0.00
RTI::AttributeHandleValuePairSet->getRegion	1	25	0.00	100.00	0.00	0.00	0.00	0.00
RTI::AttributeHandleValuePairSet->getTransportType	1	25	0.00	100.00	0.00	0.00	0.00	0.00
RTI::AttributeHandleValuePairSet->getValue	3	1907	4.04	83.27	0.00	12.69	0.00	0.00
RTI::AttributeHandleValuePairSet->getValueLength	1	25	12.00	0.00	0.00	88.00	0.00	0.00
RTI::AttributeHandleValuePairSet->moveFrom	2	225	1.78	63.56	0.00	34.67	0.00	0.00
RTI::AttributeHandleValuePairSet->next	1	24	100.00	0.00	0.00	0.00	0.00	0.00
RTI::AttributeHandleValuePairSet->remove	1	25	4.00	96.00	0.00	0.00	0.00	0.00
RTI::AttributeHandleValuePairSet->valid	1	24	100.00	0.00	0.00	0.00	0.00	0.00
RTI::AttributeSetFactory::create	1	24	70.83	29.17	0.00	0.00	0.00	0.00
RTI::FederateHandleSet->add	1	25	100.00	0.00	0.00	0.00	0.00	0.00
RTI::FederateHandleSet->getHandle	1	25	4.00	96.00	0.00	0.00	0.00	0.00
RTI::FederateHandleSet->isMember	1	25	100.00	0.00	0.00	0.00	0.00	0.00
RTI::FederateHandleSet->remove	1	24	4.17	95.83	0.00	0.00	0.00	0.00
RTI::FederateHandleSetFactory::create	1	24	54.17	45.83	0.00	0.00	0.00	0.00
RTI::ParameterHandleValuePairSet->getValueLength	1	25	12.00	0.00	0.00	88.00	0.00	0.00
RTI::ParameterHandleValuePairSet->remove	1	25	4.00	96.00	0.00	0.00	0.00	0.00
RTI::ParameterSetFactory::create	1	24	70.83	29.17	0.00	0.00	0.00	0.00
rtiAmb.attributeOwnershipAcquisition	3	1248	6.65	72.20	4.09	17.07	0.00	0.00
rtiAmb.changeAttributeOrderType	3	384	3.13	78.39	2.08	16.41	0.00	0.00
rtiAmb.changeAttributeTransportationType	3	2217	0.72	86.02	0.59	12.67	0.00	0.00
rtiAmb.changeInteractionOrderType	2	576	0.00	100.00	0.00	0.00	0.00	0.00
rtiAmb.changeInteractionTransportation	2	576	0.00	100.00	0.00	0.00	0.00	0.00
rtiAmb.deleteObjectInstance	3	1573	3.56	84.30	0.13	12.02	0.00	0.00
rtiAmb.deleteObjectInstance	2	1072	4.01	95.99	0.00	0.00	0.00	0.00
rtiAmb.destroyFederationExecution	1	52	0.00	100.00	0.00	0.00	0.00	0.00

Function	number of parameters	number of tests run	% of tests that passed clean	% of tests that passed with an exception	% of tests that got RTI Internal Error exception	% of tests that got Unknown exception	% of tests that got Aborts	% of tests that got Restarts
rtiAmb.flushQueueRequest	1	9	88.89	0.00	0.00	11.11	0.00	0.00
rtiAmb.getAttributeHandle	2	676	0.00	95.41	4.59	0.00	0.00	0.00
rtiAmb.getAttributeName	2	625	4.48	95.52	0.00	0.00	0.00	0.00
rtiAmb.getAttributeRoutingSpaceHandle	2	385	4.42	95.58	0.00	0.00	0.00	0.00
rtiAmb.getInteractionClassHandle	1	52	0.00	94.23	3.85	0.00	0.00	0.00
rtiAmb.getInteractionClassName	1	24	50.00	50.00	0.00	0.00	0.00	0.00
rtiAmb.getInteractionRoutingSpaceHandle	1	25	56.00	44.00	0.00	0.00	0.00	0.00
rtiAmb.getObjectClass	1	25	4.00	96.00	0.00	0.00	0.00	0.00
rtiAmb.getObjectClassHandle	1	52	0.00	96.15	3.85	0.00	0.00	0.00
rtiAmb.getObjectClassName	1	25	28.00	72.00	0.00	0.00	0.00	0.00
rtiAmb.getObjectInstanceHandle	1	7	14.29	85.71	0.00	0.00	0.00	0.00
rtiAmb.getObjectInstanceName	1	25	4.00	96.00	0.00	0.00	0.00	0.00
rtiAmb.getOrderingHandle	1	61	1.64	95.08	3.28	0.00	0.00	0.00
rtiAmb.getOrderingName	1	25	20.00	80.00	0.00	0.00	0.00	0.00
rtiAmb.getParameterHandle	2	178	0.00	94.38	5.62	0.00	0.00	0.00
rtiAmb.getParameterName	2	70	10.00	90.00	0.00	0.00	0.00	0.00
rtiAmb.getTransportationHandle	1	61	1.64	95.08	3.28	0.00	0.00	0.00
rtiAmb.getTransportationName	1	25	28.00	72.00	0.00	0.00	0.00	0.00
rtiAmb.isAttributeOwnedByFederate	2	373	0.80	99.20	0.00	0.00	0.00	0.00
rtiAmb.modifyLookahead	1	9	88.89	0.00	0.00	11.11	0.00	0.00
rtiAmb.negotiatedAttributeOwnershipDivestiture	3	810	5.56	74.32	3.70	16.42	0.00	0.00
rtiAmb.nextEventRequest	1	9	88.89	0.00	0.00	11.11	0.00	0.00
rtiAmb.nextEventRequestAvailable	1	9	88.89	0.00	0.00	11.11	0.00	0.00
rtiAmb.publishInteractionClass	1	24	54.17	45.83	0.00	0.00	0.00	0.00
rtiAmb.publishObjectClass	2	400	17.50	63.00	7.00	12.50	0.00	0.00
rtiAmb.queryAttributeOwnership	2	625	0.80	99.20	0.00	0.00	0.00	0.00
rtiAmb.queryFederateTime	1	8	87.50	0.00	0.00	12.50	0.00	0.00
rtiAmb.queryLBTS	1	8	87.50	0.00	0.00	12.50	0.00	0.00
rtiAmb.queryLookahead	1	8	87.50	0.00	0.00	12.50	0.00	0.00
rtiAmb.queryMinNextEventTime	1	8	87.50	0.00	0.00	12.50	0.00	0.00
rtiAmb.registerFederationSynchronization	2	415	90.36	0.00	9.64	0.00	0.00	0.00
rtiAmb.registerObjectInstance	1	25	4.00	24.00	72.00	0.00	0.00	0.00
rtiAmb.registerObjectInstance	2	1300	2.54	23.00	74.46	0.00	0.00	0.00
rtiAmb.requestClassAttributeValueUpdate	2	400	17.50	63.00	7.00	12.50	0.00	0.00
rtiAmb.requestFederationRestore	1	52	96.15	0.00	3.85	0.00	0.00	0.00
rtiAmb.requestFederationSave	1	52	0.00	0.00	3.85	0.00	0.00	96.15
rtiAmb.requestFederationSave	2	63	88.89	0.00	0.00	11.11	0.00	0.00
rtiAmb.requestObjectAttributeValueUpdate	2	400	87.50	0.00	0.00	12.50	0.00	0.00
rtiAmb.sendInteraction	3	892	0.00	89.01	0.00	10.99	0.00	0.00
rtiAmb.subscribeInteractionClass	2	288	54.17	45.83	0.00	0.00	0.00	0.00

Function	number of parameters	number of tests run	% of tests that passed clean	% of tests that passed with an exception	% of tests that got RTI Internal Error exception	% of tests that got Unknown exception	% of tests that got Aborts	% of tests that got Restarts
rtiAmb.subscribeObjectClassAttributes	3	219	16.44	63.47	5.48	14.61	0.00	0.00
rtiAmb.tick	2	100	100.00	0.00	0.00	0.00	0.00	0.00
rtiAmb.timeAdvanceRequest	1	9	88.89	0.00	0.00	11.11	0.00	0.00
rtiAmb.timeAdvanceRequestAvailable	1	9	88.89	0.00	0.00	11.11	0.00	0.00
rtiAmb.unconditionalAttributeOwnershipDivestiture	2	400	2.50	84.00	1.00	12.50	0.00	0.00
rtiAmb.unpublishInteractionClass	1	24	0.00	100.00	0.00	0.00	0.00	0.00
rtiAmb.unpublishObjectClass	1	25	4.00	96.00	0.00	0.00	0.00	0.00
rtiAmb.unsubscribeInteractionClass	1	25	0.00	100.00	0.00	0.00	0.00	0.00
rtiAmb.unsubscribeObjectClass	1	25	0.00	100.00	0.00	0.00	0.00	0.00

Appendix C: Results for RTI 1.3.5 for Sun OS

Function	number of parameters	number of tests run	% of tests that passed clean	% of tests that passed with an exception	% of tests that got RTI Internal Error exception	% of tests that got Unknown exception	% of tests that got Aborts	% of tests that got Restarts
new RTI::Exception	1	52	98.08	0.00	0.00	0.00	1.92	0.00
new RTI::Exception	2	1300	98.08	0.00	0.00	0.00	1.92	0.00
new RTI::RegionNotKnown	1	52	98.08	0.00	0.00	0.00	1.92	0.00
new RTI::RegionNotKnown	2	402	97.51	0.00	0.00	0.00	2.49	0.00
RTI::AttributeHandleSet->add	1	25	72.00	28.00	0.00	0.00	0.00	0.00
RTI::AttributeHandleSet->getHandle	1	25	4.00	96.00	0.00	0.00	0.00	0.00
RTI::AttributeHandleSet->isMember	1	25	88.00	0.00	0.00	0.00	12.00	0.00
RTI::AttributeHandleSet->remove	1	25	72.00	28.00	0.00	0.00	0.00	0.00
RTI::AttributeHandleSetFactory::create	1	25	100.00	0.00	0.00	0.00	0.00	0.00
RTI::AttributeHandleValuePairSet->add	3	1017	79.74	0.00	0.00	0.00	20.26	0.00
RTI::AttributeHandleValuePairSet->getHandle	1	25	4.00	84.00	0.00	0.00	12.00	0.00
RTI::AttributeHandleValuePairSet->getOrderType	1	25	0.00	100.00	0.00	0.00	0.00	0.00
RTI::AttributeHandleValuePairSet->getRegion	1	25	0.00	100.00	0.00	0.00	0.00	0.00
RTI::AttributeHandleValuePairSet->getTransportType	1	25	0.00	100.00	0.00	0.00	0.00	0.00
RTI::AttributeHandleValuePairSet->getValue	3	2070	3.72	84.83	0.00	0.00	11.45	0.00
RTI::AttributeHandleValuePairSet->getValueLength	1	25	8.00	0.00	0.00	0.00	92.00	0.00
RTI::AttributeHandleValuePairSet->moveFrom	2	225	3.56	63.56	0.00	0.00	32.89	0.00
RTI::AttributeHandleValuePairSet->next	1	25	100.00	0.00	0.00	0.00	0.00	0.00
RTI::AttributeHandleValuePairSet->remove	1	25	4.00	96.00	0.00	0.00	0.00	0.00
RTI::AttributeHandleValuePairSet->valid	1	25	100.00	0.00	0.00	0.00	0.00	0.00
RTI::AttributeSetFactory::create	1	25	72.00	28.00	0.00	0.00	0.00	0.00
RTI::FederateHandleSet->add	1	25	100.00	0.00	0.00	0.00	0.00	0.00
RTI::FederateHandleSet->getHandle	1	25	4.00	96.00	0.00	0.00	0.00	0.00
RTI::FederateHandleSet->isMember	1	25	100.00	0.00	0.00	0.00	0.00	0.00
RTI::FederateHandleSet->remove	1	25	4.00	96.00	0.00	0.00	0.00	0.00
RTI::FederateHandleSetFactory::create	1	25	56.00	44.00	0.00	0.00	0.00	0.00
RTI::ParameterHandleValuePairSet->getValueLength	1	25	12.00	0.00	0.00	0.00	88.00	0.00
RTI::ParameterHandleValuePairSet->remove	1	25	0.00	100.00	0.00	0.00	0.00	0.00
RTI::ParameterSetFactory::create	1	25	72.00	28.00	0.00	0.00	0.00	0.00
rtiAmb.attributeOwnershipAcquisition	3	606	2.15	84.32	0.00	0.00	13.53	0.00
rtiAmb.changeAttributeOrderType	3	630	0.63	87.30	0.00	0.00	12.06	0.00
rtiAmb.changeAttributeTransportationType	3	999	0.90	86.69	0.00	0.00	12.41	0.00
rtiAmb.changeInteractionOrderType	2	62	0.00	100.00	0.00	0.00	0.00	0.00
rtiAmb.changeInteractionTransportation	2	62	0.00	100.00	0.00	0.00	0.00	0.00
rtiAmb.deleteObjectInstance	3	689	3.48	84.47	0.00	0.00	12.05	0.00
rtiAmb.deleteObjectInstance	2	130	9.23	90.77	0.00	0.00	0.00	0.00
rtiAmb.destroyFederationExecution	1	52	0.00	100.00	0.00	0.00	0.00	0.00
rtiAmb.flushQueueRequest	1	9	88.89	0.00	0.00	0.00	11.11	0.00

Function	number of parameters	number of tests run	% of tests that passed clean	% of tests that passed with an exception	% of tests that got RTI Internal Error exception	% of tests that got Unknown exception	% of tests that got Aborts	% of tests that got Restarts
rtiAmb.getAttributeHandle	2	130	0.00	96.15	0.00	0.00	3.85	0.00
rtiAmb.getAttributeName	2	625	4.48	95.52	0.00	0.00	0.00	0.00
rtiAmb.getAttributeRoutingSpaceHandle	2	44	4.55	95.45	0.00	0.00	0.00	0.00
rtiAmb.getInteractionClassHandle	1	52	0.00	96.15	0.00	0.00	3.85	0.00
rtiAmb.getInteractionClassName	1	25	56.00	44.00	0.00	0.00	0.00	0.00
rtiAmb.getInteractionRoutingSpaceHandle	1	25	56.00	44.00	0.00	0.00	0.00	0.00
rtiAmb.getObjectClass	1	25	4.00	96.00	0.00	0.00	0.00	0.00
rtiAmb.getObjectClassHandle	1	52	0.00	96.15	0.00	0.00	3.85	0.00
rtiAmb.getObjectClassName	1	25	28.00	72.00	0.00	0.00	0.00	0.00
rtiAmb.getObjectInstanceHandle	1	52	0.00	96.15	0.00	0.00	3.85	0.00
rtiAmb.getObjectInstanceName	1	25	4.00	96.00	0.00	0.00	0.00	0.00
rtiAmb.getOrderingHandle	1	52	0.00	96.15	0.00	0.00	3.85	0.00
rtiAmb.getOrderingName	1	25	20.00	80.00	0.00	0.00	0.00	0.00
rtiAmb.getParameterHandle	2	129	0.00	96.12	0.00	0.00	3.88	0.00
rtiAmb.getParameterName	2	63	11.11	88.89	0.00	0.00	0.00	0.00
rtiAmb.getTransportationHandle	1	52	0.00	96.15	0.00	0.00	3.85	0.00
rtiAmb.getTransportationName	1	25	28.00	72.00	0.00	0.00	0.00	0.00
rtiAmb.isAttributeOwnedByFederate	2	62	1.61	98.39	0.00	0.00	0.00	0.00
rtiAmb.modifyLookahead	1	9	88.89	0.00	0.00	0.00	11.11	0.00
rtiAmb.negotiatedAttributeOwnershipDivestiture	3	166	2.41	83.73	0.00	0.00	13.86	0.00
rtiAmb.nextEventRequest	1	9	88.89	0.00	0.00	0.00	11.11	0.00
rtiAmb.nextEventRequestAvailable	1	9	88.89	0.00	0.00	0.00	11.11	0.00
rtiAmb.publishInteractionClass	1	25	56.00	44.00	0.00	0.00	0.00	0.00
rtiAmb.publishObjectClass	2	40	17.50	60.00	0.00	0.00	22.50	0.00
rtiAmb.queryAttributeOwnership	2	62	1.61	98.39	0.00	0.00	0.00	0.00
rtiAmb.queryFederateTime	1	8	87.50	0.00	0.00	0.00	12.50	0.00
rtiAmb.queryLBTS	1	8	87.50	0.00	0.00	0.00	12.50	0.00
rtiAmb.queryLookahead	1	8	87.50	0.00	0.00	0.00	12.50	0.00
rtiAmb.queryMinNextEventTime	1	8	87.50	0.00	0.00	0.00	12.50	0.00
rtiAmb.registerFederationSynchronizationPoint	2	397	90.43	0.00	0.00	0.00	9.57	0.00
rtiAmb.registerObjectInstance	1	25	4.00	24.00	0.00	0.00	72.00	0.00
rtiAmb.registerObjectInstance	2	1300	2.46	23.00	0.00	0.00	74.54	0.00
rtiAmb.requestClassAttributeValueUpdate	2	40	17.50	60.00	0.00	0.00	22.50	0.00
rtiAmb.requestFederationRestore	1	52	96.15	0.00	0.00	0.00	3.85	0.00
rtiAmb.requestFederationSave	1	52	0.00	0.00	0.00	0.00	3.85	96.15
rtiAmb.requestFederationSave	2	468	85.68	0.00	0.00	0.00	14.32	0.00
rtiAmb.requestObjectAttributeValueUpdate	2	40	85.00	0.00	0.00	0.00	15.00	0.00
rtiAmb.sendInteraction	3	408	0.00	88.24	0.00	0.00	11.76	0.00
rtiAmb.subscribeInteractionClass	1	25	56.00	44.00	0.00	0.00	0.00	0.00
rtiAmb.subscribeObjectClassAttributes	3	479	16.91	63.88	0.00	0.00	19.21	0.00

Function	number of parameters	number of tests run	% of tests that passed clean	% of tests that passed with an exception	% of tests that got RTI Internal Error exception	% of tests that got Unknown exception	% of tests that got Aborts	% of tests that got Restarts
rtiAmb.tick	2	100	100.00	0.00	0.00	0.00	0.00	0.00
rtiAmb.timeAdvanceRequest	1	9	88.89	0.00	0.00	0.00	11.11	0.00
rtiAmb.timeAdvanceRequestAvailable	1	9	88.89	0.00	0.00	0.00	11.11	0.00
rtiAmb.unconditionalAttributeOwnershipDivestiture	2	40	0.00	85.00	0.00	0.00	15.00	0.00
rtiAmb.unpublishInteractionClass	1	25	0.00	100.00	0.00	0.00	0.00	0.00
rtiAmb.unpublishObjectClass	1	25	4.00	96.00	0.00	0.00	0.00	0.00
rtiAmb.unsubscribeInteractionClass	1	25	0.00	100.00	0.00	0.00	0.00	0.00
rtiAmb.unsubscribeObjectClass	1	25	0.00	100.00	0.00	0.00	0.00	0.00

Appendix D: Results of testing RTI 1.3NG

Function	number of parameters	number of tests run	% of tests that passed clean	% of tests that passed with an exception	% of tests that got RTI Internal Error exception	% of tests that got Unknown exception	% of tests that got Aborts	% of tests that got Restarts
new RTI::Exception	1	X	X	X	X	X	X	X
new RTI::Exception	2	X	X	X	X	X	X	X
new RTI::RegionNotKnown	1	52	98.08	0.00	0.00	0.00	1.92	0.00
new RTI::RegionNotKnown	2	130	97.69	0.00	0.00	0.00	2.31	0.00
RTI::AttributeHandleSet->add	1	25	100.00	0.00	0.00	0.00	0.00	0.00
RTI::AttributeHandleSet->getHandle	1	25	4.00	96.00	0.00	0.00	0.00	0.00
RTI::AttributeHandleSet->isMember	1	25	100.00	0.00	0.00	0.00	0.00	0.00
RTI::AttributeHandleSet->remove	1	25	4.00	96.00	0.00	0.00	0.00	0.00
RTI::AttributeHandleSetFactory::create	1	25	88.00	12.00	0.00	0.00	0.00	0.00
RTI::AttributeHandleValuePairSet->add	3	1688	80.04	0.00	0.00	0.00	19.96	0.00
RTI::AttributeHandleValuePairSet->getHandle	1	25	4.00	84.00	0.00	0.00	12.00	0.00
RTI::AttributeHandleValuePairSet->getOrderType	1	25	0.00	100.00	0.00	0.00	0.00	0.00
RTI::AttributeHandleValuePairSet->getRegion	1	25	0.00	100.00	0.00	0.00	0.00	0.00
RTI::AttributeHandleValuePairSet->getTransportType	1	25	0.00	100.00	0.00	0.00	0.00	0.00
RTI::AttributeHandleValuePairSet->getValue	3	1091	3.67	84.23	0.00	0.00	12.10	0.00
RTI::AttributeHandleValuePairSet->getValueLength	1	25	20.00	0.00	0.00	0.00	80.00	0.00
RTI::AttributeHandleValuePairSet->moveFrom	2	225	3.56	63.56	0.00	0.00	32.89	0.00
RTI::AttributeHandleValuePairSet->next	1	25	100.00	0.00	0.00	0.00	0.00	0.00
RTI::AttributeHandleValuePairSet->remove	1	25	4.00	96.00	0.00	0.00	0.00	0.00
RTI::AttributeHandleValuePairSet->valid	1	25	100.00	0.00	0.00	0.00	0.00	0.00
RTI::AttributeSetFactory::create	1	25	72.00	28.00	0.00	0.00	0.00	0.00
RTI::FederateHandleSet->add	1	25	100.00	0.00	0.00	0.00	0.00	0.00
RTI::FederateHandleSet->getHandle	1	25	4.00	96.00	0.00	0.00	0.00	0.00
RTI::FederateHandleSet->isMember	1	25	100.00	0.00	0.00	0.00	0.00	0.00
RTI::FederateHandleSet->remove	1	25	4.00	96.00	0.00	0.00	0.00	0.00
RTI::FederateHandleSetFactory::create	1	25	72.00	28.00	0.00	0.00	0.00	0.00
RTI::ParameterHandleValuePairSet->getValueLength	1	25	12.00	0.00	0.00	0.00	88.00	0.00
RTI::ParameterHandleValuePairSet->remove	1	25	4.00	96.00	0.00	0.00	0.00	0.00
RTI::ParameterSetFactory::create	1	25	72.00	28.00	0.00	0.00	0.00	0.00
rtiAmb.attributeOwnershipAcquisition	3	501	46.11	13.77	0.00	0.00	40.12	0.00
rtiAmb.changeAttributeOrderType	3	51	0.00	88.24	0.00	0.00	11.76	0.00
rtiAmb.changeAttributeTransportationType	3	998	0.00	88.48	0.00	0.00	11.52	0.00
rtiAmb.changeInteractionOrderType	2	54	0.00	100.00	0.00	0.00	0.00	0.00
rtiAmb.changeInteractionTransportationType	2	54	0.00	100.00	0.00	0.00	0.00	0.00
rtiAmb.deleteObjectInstance	3	725	0.00	88.69	0.00	0.00	11.31	0.00
rtiAmb.deleteObjectInstance	2	193	5.18	92.23	0.00	0.00	2.59	0.00
rtiAmb.destroyfederationExecution	1	52	0.00	57.69	38.46	0.00	3.85	0.00

Function	number of parameters	number of tests run	% of tests that passed clean	% of tests that passed with an exception	% of tests that got RTI Internal Error exception	% of tests that got Unknown exception	% of tests that got Aborts	% of tests that got Restarts
rtiAmb.flushQueueRequest	1	9	88.89	0.00	0.00	0.00	11.11	0.00
rtiAmb.getAttributeHandle	2	121	0.00	96.69	0.00	0.00	3.31	0.00
rtiAmb.getAttributeName	2	91	7.69	92.31	0.00	0.00	0.00	0.00
rtiAmb.getAttributeRoutingSpaceHandle	2	92	29.35	70.65	0.00	0.00	0.00	0.00
rtiAmb.getInteractionClassHandle	1	50	0.00	98.00	0.00	0.00	2.00	0.00
rtiAmb.getInteractionClassName	1	25	52.00	48.00	0.00	0.00	0.00	0.00
rtiAmb.getInteractionRoutingSpaceHandle	1	25	52.00	0.00	48.00	0.00	0.00	0.00
rtiAmb.getObjectClass	1	25	4.00	96.00	0.00	0.00	0.00	0.00
rtiAmb.getObjectClassHandle	1	52	0.00	98.08	0.00	0.00	1.92	0.00
rtiAmb.getObjectClassName	1	25	24.00	76.00	0.00	0.00	0.00	0.00
rtiAmb.getObjectInstanceHandle	1	52	0.00	98.08	0.00	0.00	1.92	0.00
rtiAmb.getObjectInstanceName	1	25	4.00	96.00	0.00	0.00	0.00	0.00
rtiAmb.getOrderingHandle	1	52	0.00	98.08	0.00	0.00	1.92	0.00
rtiAmb.getOrderingName	1	25	16.00	84.00	0.00	0.00	0.00	0.00
rtiAmb.getParameterHandle	2	121	0.00	96.69	0.00	0.00	3.31	0.00
rtiAmb.getParameterName	2	50	8.00	92.00	0.00	0.00	0.00	0.00
rtiAmb.getTransportationHandle	1	52	0.00	98.08	0.00	0.00	1.92	0.00
rtiAmb.getTransportationName	1	25	16.00	84.00	0.00	0.00	0.00	0.00
rtiAmb.isAttributeOwnedByFederate	1	93	0.00	100.00	0.00	0.00	0.00	0.00
rtiAmb.modifyLookahead	1	9	88.89	0.00	0.00	0.00	11.11	0.00
rtiAmb.negotiatedAttributeOwnershipDivestiture	3	534	0.00	86.70	0.00	0.00	13.30	0.00
rtiAmb.nextEventRequest	1	9	88.89	0.00	0.00	0.00	11.11	0.00
rtiAmb.nextEventRequestAvailable	1	9	88.89	0.00	0.00	0.00	11.11	0.00
rtiAmb.publishInteractionClass	1	25	52.00	48.00	0.00	0.00	0.00	0.00
rtiAmb.publishObjectClass	2	171	2.34	78.95	2.92	0.00	15.79	0.00
rtiAmb.queryAttributeOwnership	2	92	0.00	100.00	0.00	0.00	0.00	0.00
rtiAmb.queryFederateTime	1	8	87.50	0.00	0.00	0.00	12.50	0.00
rtiAmb.queryLBTS	1	8	87.50	0.00	0.00	0.00	12.50	0.00
rtiAmb.queryLookahead	1	8	87.50	0.00	0.00	0.00	12.50	0.00
rtiAmb.queryMinNextEventTime	1	8	87.50	0.00	0.00	0.00	12.50	0.00
rtiAmb.registerFederationSynchronizationPoint	2	215	93.95	0.00	0.00	0.00	6.05	0.00
rtiAmb.registerObjectInstance	1	25	4.00	96.00	0.00	0.00	0.00	0.00
rtiAmb.registerObjectInstance	2	324	4.01	92.28	1.23	0.00	2.47	0.00
rtiAmb.requestClassAttributeValueUpdate	2	52	7.69	67.31	0.00	0.00	25.00	0.00
rtiAmb.requestFederationRestore	1	51	98.04	0.00	0.00	0.00	1.96	0.00
rtiAmb.requestFederationSave	1	52	98.08	0.00	0.00	0.00	1.92	0.00
rtiAmb.requestFederationSave	2	463	33.05	54.21	0.00	0.00	12.74	0.00
rtiAmb.requestObjectAttributeValueUpdate	2	173	53.76	16.76	0.00	0.00	29.48	0.00
rtiAmb.sendInteraction	3	416	0.00	87.26	0.00	0.00	12.74	0.00
rtiAmb.subscribeInteractionClass	1	54	44.44	55.56	0.00	0.00	0.00	0.00

Function	number of parameters	number of tests run	% of tests that passed clean	% of tests that passed with an exception	% of tests that got RTI Internal Error exception	% of tests that got Unknown exception	% of tests that got Aborts	% of tests that got Restarts
rtiAmb.subscribeObjectClassAttributes	3	479	13.15	68.27	0.00	0.00	18.58	0.00
rtiAmb.tick	2	63	100.00	0.00	0.00	0.00	0.00	0.00
rtiAmb.timeAdvanceRequest	1	9	88.89	0.00	0.00	0.00	11.11	0.00
rtiAmb.timeAdvanceRequestAvailable	1	9	88.89	0.00	0.00	0.00	11.11	0.00
rtiAmb.unconditionalAttributeOwnershipDivestiture	2	217	0.92	84.33	0.00	0.00	14.75	0.00
rtiAmb.unpublishInteractionClass	1	25	0.00	100.00	0.00	0.00	0.00	0.00
rtiAmb.unpublishObjectClass	1	25	4.00	96.00	0.00	0.00	0.00	0.00
rtiAmb.unsubscribeInteractionClass	1	25	0.00	100.00	0.00	0.00	0.00	0.00
rtiAmb.unsubscribeObjectClass	1	25	4.00	96.00	0.00	0.00	0.00	0.00