Developing Framework to do the Reliability Testing on webMethods Product Suite

[1]Harsh Chincholi, [1]Sandhya S, [3]Rameshchandra V,
[1] Computer Science & Engineering, R.V. College of Engineering, Bangalore
[2] Software AG Technologies Private Limited, Bangalore

Abstract
Over the past few years, use of integration server (IS) in critical situations has been increasing and need for high dependable integration server is necessary. Now, reliability as the primary characteristic of dependability is a key factor to evaluate integration server .So a Reliability framework is developed which is an automatic testing tool for integration server. Reliability framework examines the type information of a set of Java classes and constructs code fragments that will create instances of different types to test the behavior of public methods under random data. Reliability framework attempts to detect bugs by causing the program under test to "crash", that is, to throw an undeclared runtime exception. It also has the advantage of being completely automatic: no supervision is required except for off-line inspection of the test cases that have caused a crash.

Keywords: Integration server, reliability, test cases.

I. Introduction

webMethods product suite is an integrated set of design tools, run-time servers, registry/repositories, and Internet browser-based user interfaces that enables to develop and run integration solutions, create and manage a business-to-business integration network, develop and run composite applications, design and run business processes, develop and govern a service-oriented architecture, monitor and improve the performance and efficiency of business activity. Integration Server which is part of the webMethods Product Suite is one which receives requests from client applications and authenticates and authorizes the requesting users, invokes the appropriate services and passes them input data from the requesting clients, receives output data from the services and returns it to the clients and supports a wide range of established and emerging standards so that users can interact with virtually any business partner that is connected to the Internet.

With the proliferation of integration server (IS) in safety critical situations, need for high dependable integration server have been increasing. Now, reliability as the primary characteristic of dependability is a key factor to evaluate integration server. But, a hard problem of how to demonstrate the high reliability integration server needs urgently to be solved. The problem includes two parts: one is how to generate appropriate reliability testing cases considering the characteristics of integration server, and the other is how to control the testing process and evaluate the testing results. Unlike the functional testing for general application software, the reliability testing cases generating method for integration server is much more complex [1]because it must conform to the operation profile (or usage model) of integration server.

The above suggests that building reliable software systems requires understanding reliability of integration server at the architectural level. Several recent approaches have begun to quantify software reliability at the level of architectural models, or at least in terms of high-level system structure. All of these efforts focus on system-level reliability prediction. While they acknowledge that individual components reliabilities have a significant impact on system reliability, these approaches almost invariably assume that the reliabilities of the individual components in a system are known. The few approaches which do consider component-level reliability, assume that the reliabilities of a given component's elements, such as its services, are known. We do not believe these assumptions to be reasonable: it is unclear how the reliability of a component, or its services, is obtained in these approaches. The reliability would either have to be randomly guessed or the component would have to be implemented and one of the existing code-level reliability estimation techniques applied to it. None of these options is satisfying.

This paper strives to remedy the shortcomings of previous approaches by proposing a framework for predicting reliability of software components at architectural level. This is intended to be complementary to the existing on system-level reliability prediction. The purpose of the application is to implement a platform to do reliability testing on integration server across organization. The application provides a framework for all the teams working on various web methods products suite to generate test cases of integration server code, run the tests cases, which generates HTML or XML test results reports, which contains class name, number of tests checked, execution time, number of failures and customizes the reports generated.

II. Need for a New Reliability Framework for Is

The purpose of reliability framework is to discover potential problems with in the design as early as possible and provide confidence that the system meets its reliability requirements [2].

Reliability testing may be performed at several levels. Complex systems may be tested at component, assembly, subsystem and system levels. For example, performing environmental stress screening tests at lower levels, such as piece parts or small assemblies, catches problems before they cause failures at higher levels. Testing proceeds during each level of integration through full-up system testing, developmental testing, and operational testing, thereby reducing program risk. System reliability is calculated at each test level. Reliability growth techniques and failure reporting, analysis and corrective active systems (FRACAS) are often employed to improve reliability as testing progresses. The drawbacks to such extensive testing are time consuming and expense

It is not always feasible to test all system requirements. Some systems are prohibitively expensive to test; some failure modes may take years to observe; some complex interactions result in a huge number of possible test cases; and some tests require the use of limited test ranges or other resources. In such cases, different approaches to testing can be used, such as accelerated life testing [3], design of experiments, and simulations.

The desired level of statistical confidence also plays an important role in reliability testing. Statistical confidence is increased by increasing either the test time or the number of items tested. Reliability test plans are designed to achieve the specified reliability at the specified confidence level with the minimum number of test units and test time. Different test plans result in different levels of risk to the producer and consumer. The desired reliability, statistical confidence, and risk levels for each side influence the ultimate test plan. Good test requirements ensure that the customer and developer agree in advance on how reliability requirements will be tested.

A key aspect of reliability testing is to define "failure". Although this may seem obvious, there are many situations where it is not clear whether a failure is really the fault of the system. Variations in test conditions, operator differences, weather, and unexpected situations create differences between the customer and the system developer. One strategy to address this issue is to use a scoring conference process. Each test case is considered by the group and "scored" as a success or failure. This scoring is the official result used by the reliability engineer.

III. Current State Challenges

Now that Web-based applications like integration server influence everything from customer's experience to your relationship with vendors, reliable corporate information systems are increasingly critical to the company, the customers, and business partners. Because failure of an application can result in lost business and substantial recovery costs, and because the Internet is always open for business, companies are requesting 24 X 7 reliability for many enterprise applications. The Internet has made information error-free access to it.

An enterprise application is a collection of hardware, operating system services, software components, and (usually) human processes that are intended to cooperate to provide the expected business services. Reliability of the whole application depends very much on the reliability of the individual components. Because all components in a system are related, a failure in one component can affect the reliability of other components.

Application failures occur for many reasons like inadequate testing, change management problems, lack of ongoing monitoring and analysis, operations errors, weak code, interaction with external services or applications, different operating conditions (usage level changes, peak overloads),unusual events (security failures, broadcast storms).

Informally, software reliability is about how well an application accurately provides — without failure — the services that were defined in the original specification. In addition to how long the application runs before failure, reliability engineering is about providing correct results and handling error detection and recovery in order to avoid failures.

The consequence of failure can range from simply providing no service, to providing incorrect service, to causing corrupted data and damaged system states. While some failures are merely inconvenient (like when a Web site fails to respond), other failures range from time-wasting interruption.

The process of designing for reliability involves looking at the application's expected usage pattern, specifying the required reliability profile, and engineering the software architecture with the intention of meeting the profile. Reliability engineering requires in-depth consideration of an application's interactions with other system processes. Understanding the reliability problems and solutions for a system is not easy, but a good place to start is with analysis of currently running applications. Such analysis should reveal the failure frequency and distribution, root causes, and possible improvements for existing systems. Armed with current operational data [4], and leveraging the current reliability knowledge into a better design for the new system.

As a design concept, reliability is about an application's ability to operate failure free. This includes ensuring accurate data input and data transformations, error-free state management, and non-corrupting recovery from detected failure conditions. Creating a high-reliability application depends on the entire software development lifecycle from early design specification, through building and testing, to deployment and ongoing operational maintenance. Reliability can't be added onto an application just before deployment.

Currently functional testing frameworks are present which doesn't check reliability of integration server like lack of resilience, environmental failure at architectural level. In order to overcome the functional testing frameworks disadvantages this framework is developed which analyzes methods, produces test cases and generate test report.

IV. METHODOLOGY

The Reliability Framework can be modularized in to following modules as shown in figure 1: Data type definitions, Heuristics, Random test capability, Test case generator, Tester, Result monitor, Pattern recognizer/hypothesizer, Wrapper creation, Assertion templates.

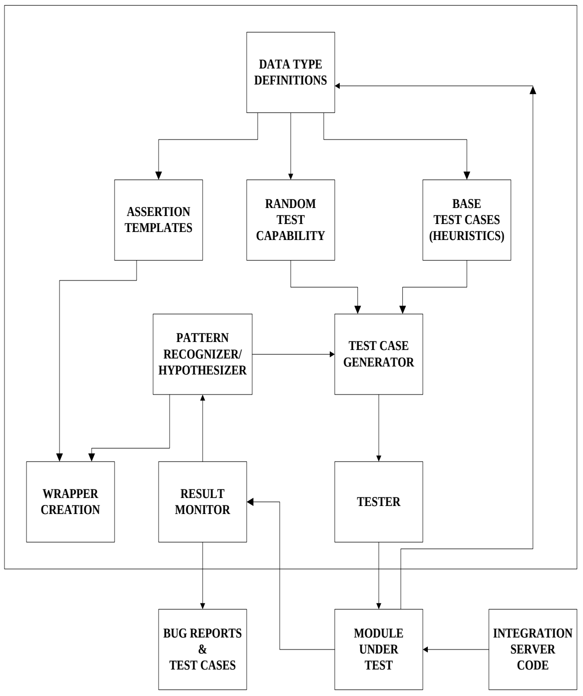

Figure 1: Modularized Reliability Framework

Data type definitions

Data type definitions are pre-defined data type information that provides base test cases, information about how to conduct random tests, and templates for use in assertion generation. Framework comes pre-seeded with a variety of data type definitions in the form of an inheritance hierarchy (*e.g.*, "file pointer" inherits from data type "int" and "pointer to buffer area" inherits from "generic pointer"). Thus, information from the interface declaration is provided in terms of defined data types.

In data type definitions, class loader reads the class bytecode from the file system, analyzes it using Java reflection .For each method declared by a given class, data type definitions uses Java reflection to identify method parameter types, types returned by methods, subtyping relations and visibility constraints.

Heuristics:

Heuristics consist of a set of manually-defined heuristics for testing a particular data type (the heuristics are accumulated over the inheritance hierarchy for that data type).These heuristics serve to find single-point discontinuities in the module response space in an efficient and effective manner.

Random test capability:

A random test capability is available to provide a randomly-distributed coverage of the module response space. It generates test cases in a pseudo-random manner (and thus is repeatable by resetting the pseudo-random generator seed), and provides starting points for the pattern recognizer to begin building response regions.

Random test capability does the parameter-graphing, test case selection and execution. The parameter-graph is an abstract representation of the parameter-space of the program's methods. Creating different parameter combinations for a method under test can be done by traversing the graph. The representation is useful because it allows us to easily bound the depth of method chaining. Each test case should provide a method with a different combination of its parameter-types' value-range. Result monitor can produce two or more test cases producing the same value. This can happen if a returning method returns the same value for different parameter combinations, or two returning methods have overlapping return-value ranges. Test cases are picked at random then a method with more parameter combinations is tested more often. This is advantageous, as more parameter combinations tend to produce a bigger variety of parameter state. And it generates more test cases for a more complicated method. Possible side-effects of methods via variables are ignored in test case selection. It does not attempt to deliberately search the space of possible side-effects by exploring all possible combinations of method calls. For this reason void-returning methods are currently excluded from the parameter-graph.By representing the parameter-space implicitly-via the parameter-graph, result monitor efficiently compute the number of test cases for a given search depth, that is, compute the size of the parameter-space.

Each test case consists of one or more method or constructor invocations contained in a single block. Each Exception thrown by a test case is caught and passed to the runtime. According to the heuristics the runtime either considers the exception a bug of the code under test or considers it an ill-formed test case-that is, failure to provide legal inputs-and suppresses the exception. These heuristics take into account the type of the exception and the method call history that led to the exception.

Test case generator

The test case generator has two modes of operation. In open-loop mode it selects both base test cases and a number of random test cases to test the module and report any robustness bugs found. In closed-loop mode it interacts with the pattern recognizer to map response regions.

Tester

The tester is a test harness that executes the module under test with a particular test case. In particular, the tester initializes any required data structures, executes the module under test, and then erases any remaining data structures after the test.

Result monitor

The result monitor tracks the status of the module under test and the computer system executing that module looking for crashes or hangs. Results with other than graceful behavior can be reported as bugs, and are fed to the pattern recognizer.

Pattern recognizer

The pattern recognizer/hypothesizer builds a map of the response regions of the module under test. It does so by using random and heuristic base test cases as a starting point to provided seeded response values. It then builds upon these values by "growing" regions of valid.

Wrapper Generation

Once a module has been tested and response regions have been characterized, a protective wrapper is synthesized to protect the module. The protection will be done with multi-dimensional assertions, which are executable statements that verify a parameter value is within acceptable ranges. Part of the data type definitions includes assertion templates, which are templates that contain information about how to build assertions for a particular data type.

The wrapper creation function combines information about response regions with the assertion templates for each data type used by the module under test. It collapses tests into efficient range-check style assertions (as opposed to an exhaustive process that has a separate test for every possible input value, which would obviously be to large or too slow to be useful). The wrapper creation function produces source code that compiled with or linked to the module under test to produce a hardened resultant module. In other words, if the wrapper is called instead of the original module under test, the result will be more robust in the face of exceptional input parameter values.

V .Conclusion

We presented Reliability Framework, a random testing tool for Java programs. Reliability Framework can be used to discover arbitrary program bugs and it is ideal for robustness testing of public interfaces-that is, making sure that the public methods of a program do not let erroneous inputs propagate far in the implementation.

Reliability Framework can be used either in batch mode, for random testing of large parts of application functionality, or interactively, as a plug-in to the Eclipse IDE. The Eclipse user can create simple, throw-away test cases with a couple of mouse clicks. The result is that Reliability Framework can be used interactively to test newly written code. In batch mode, Reliability Framework executes a huge number of tests looking to “crash" the application with an unexpected runtime exception. Both modes are useful for understanding the behavior of unknown code.

Reliability Framework is a valuable tool. It requires very little learning and automates the testing process without any need for program behavior specifications other than type system info. It represents a view where test cases are plentiful, and only good test cases become selected as eventually valuable assets and run these tests cases, which generates HTML or XML test results reports, which contains class name, number of tests checked, execution time, number of failures and customizes the reports generated.

VI. References

[1] Roberto Pietrantuono, Stefano Russo and Kishor S. Trivedi “Software Reliability and Testing Time Allocation: An Architecture-Based Approach”, IEEE transactions on software engineering, vol. 36, no. 3, may/june 2010, Digital Object Identifier**:** 10.1109/TSE.2010.6 pp.323-337

[2] Yumei Wu, Yongqi Zhang and Minyan Lu “Software Reliability Accelerated Testing Method Based on Mixed Testing”, Reliability and Maintainability Symposium (RAMS), IEEE 2010 Proceedings - Annual, Digital Object Identifier: 10.1109/RAMS.2010.5448017.

[3] Shuanqi Wang, Yumei Wu, Minyan Lu and Haifeng Li “Software Reliability Accelerated Testing Method Based on Test Coverage” Reliability and Maintainability Symposium (RAMS), IEEE 2011 Proceedings - Annual, Digital Object Identifier: 10.1109/RAMS.2011.5754463

[4] Erqiang Feng, Chang Liu and Jun Zheng “Software Reliability Accelerated Testing Based on the Combined Testing Method”,
Reliability, Maintainability and Safety (ICRMS), IEEE 2011 9th International Conference, Digital Object Identifier: 10.1109/ICRMS.2011.5979370 ,pp. 651 – 656