Practical Dependability Analysis of SOAP Based Systems

Nik Looker,
Department of Computer Science, University of Durham, UK
n.e.looker@durham.ac.uk,

Malcolm Munro,
Department of Computer Science, University of Durham, UK
malcolm.munro@durham.ac.uk

Jie Xu
School of Computing University of Leeds, UK
jxu@comp.leeds.ac.uk

Abstract

This paper provides a detailed example of our FIT (Fault Injection Technology) package. We show how our GUI driven package can be applied to a complex SOAP based system, generate test scripts from WSDL and detect defects in this system. Finally we discuss how our fault injection tools can be used as part of a specification based certification process.

1. Introduction

Testing is an often-neglected area of a product release cycle. A common planning mistake is to confuse debugging, development testing and unit testing with the integration-testing phase. Consequently not enough time/resources are allocated to this vital phase of a project.

Nowhere is testing more important than in the deployment of middleware products. It is vital that middleware products be reliable since any defects will be propagated through into any applications that utilize them. Even slight defects in the middleware platform can become more serious because most designs assume a reliable 'bug free' middleware platform so much time can be spent by developers of middleware applications tracking failures that they assume are in their applications when in fact they are cause by defects in the middleware.

Whilst formal methods of program proof are useful in verifying design implementation and that designs meet a given specification there are a number of issues with their use: 1) They have yet to achieve wide scale usage by developers; 2) A generally accepted method has yet to be applied to distributed problems.

Fault injection is a well-proven method of assessing the dependability of a system and our previous research has indicated its usefulness to testing web service based systems [Looker and Xu 2003a; b] utilizing the SOAP protocol [Curbera, Duftler et al. 2002].

2. Dependability Assessment

System dependability can be assessed using either model or measurement techniques [Marsden, Fabre et al. 2002]. Both techniques have their merits but measurement techniques are useful because they can be applied to existing systems without requiring access to source code or design documentation. There are two main measurement techniques:

Observation: measurements can be performed by the observation of errors and failures in a large set of deployed systems. This technique uses existing logs. Analysis of the data can obtain information on the frequency of faults and the activity that was in progress when they occurred. Since failures and errors may occur infrequently data must be collected over a long period of time and from a large number of systems. It is unlikely that this technique will catch rarely seen errors [Hecht and Hecht 1996] since they may take a very long time to occur (in the order of years).

Fault Injection: since it may take a very long time for some errors to occur, fault injection attempts to speed up this process by injecting faults into a running system. There are a number of different methods of fault injection, both hardware and software based methods but this research concentrates on software implemented fault injection.

All software methods attempt to cause the execution of seldom used control pathways within a system. By doing this either an error condition will be generated or the systems fault tolerance mechanism will handle the fault.

2.1. Network Level Fault Injection

This technique is concerned with the corruption, loss or reordering of network packets at the network interface. It is possible to use SWIFI tools to inject faults by instrumenting the operating system protocol stack as in [Dawson, Jahanian et al. 1997] but this runs the risk of being detected and rejected by the receiving systems protocol stack. It is therefore preferable to inject the fault at the application level before this stage [Marsden and Fabre 2001]. The faults injected are based on corrupting packet header information and injecting random byte errors.

3. WS-FIT

WS-FIT (Web Service – Fault Injection Technology) is a fault injector that allows

network level fault injection to be used to test SOAP based web service systems. WS-FIT has been implemented specifically to test SOAP based web service systems and a detailed description of its design is given in [Looker and Xu 2003b]. This implementation specifically handles the problems associated with modifying SOAP messages when signing and encryption are being used.

Perturbation is the modification of input parameters to a function. This technique is useful in testing such things as fault tolerance mechanisms. As stated above network level fault injection is usually based upon more or less random corruption of bytes within a network packet. Our method extends this method to make meaningful perturbation to a network packet, e.g. our method can target a single parameter within an RPC message sequence.

Our method uses an instrumented SOAP API that includes two small pieces of hook code. One hook intercepts outgoing messages, transmits them via a socket to the fault injector engine and receives a modified message from the fault injector. This message is then transmitted normally to the original destination. There is a similar hook for incoming messages, which can be processed in the same way.

By instrumenting the SOAP stacks on strategic machines this method can be used as part of the certification testing for individual components within a production system without the need for a test harness.

Whilst a number of existing fault injectors could be used to do this, notably DOCTOR [Han, Shin et al. 1995] and Orchestra [Dawson, Jahanian et al. 1996], these tools are designed for general purpose protocol testing. WS-FIT was designed around an engine to decode SOAP messages and presents an interface at the script API level with the information included in a SOAP RPC easily accessible. WSDL is utilized to produce a framework for the scripts.

4. Case Study

This simple experiment demonstrates the value of WS-FIT in providing a framework for certification testing of web services. Our simple experiment simulates a complex real-world scenario and uses the tools to locate a defect in the design of the system.

4.1. Scenario

Our system is based on a simulation of a self-regulating heating system (see Figure 1). The system is composed of three main elements: 1) a heater coil, 2) a thermocouple and 3) a controller.

Web services are provided as an interface to both the thermocouple and the heater coil. These act in the same way as a hardware driver. These drivers are hosted on separate servers (In a real world application these could be embedded devices). The heater coil hardware is designed to allow only small stepped changes to the power. It also has an upper limit to its power output of 100˚C, if set above this limit the coil will burn out.

The controller is hosted on a third server. It allows a required temperature to be set. The controller runs a continuous polling loop that periodically polls the thermocouple to check that the actual temperature is equivalent to the required temperature. If it isn't the controller increments or decrements the power to the heater coil to increase or decrease the temperature.

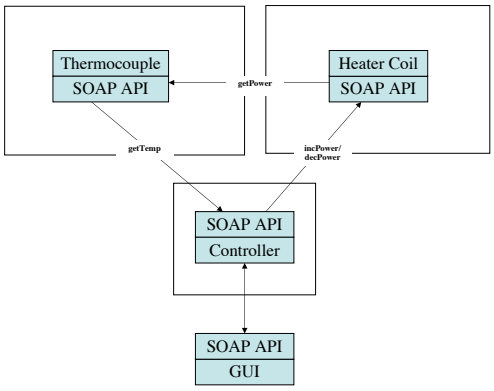

Figure 1: System Configuration

In our simulated system the thermocouple requests the currently set power from the heater coil and calculates the temperature based on this, in the real system this information would come from the thermocouple hardware.

A GUI Client is provided to exercise the system. A simple state machine is implemented by the client to first increase the temperature to 10˚C, and then decrease the temperature to 5˚C and finally to increase and hold the temperature at 7˚C.

4.2. Installing Hook Code on Servers

A small amount of hook code must be installed on any server on which faults are to be injected. By strategically positioning this hook code on certain machine WS-FIT can be used as part of the certification process for individual components of a system. For example if the instrumented SOAP stack is positioned on the

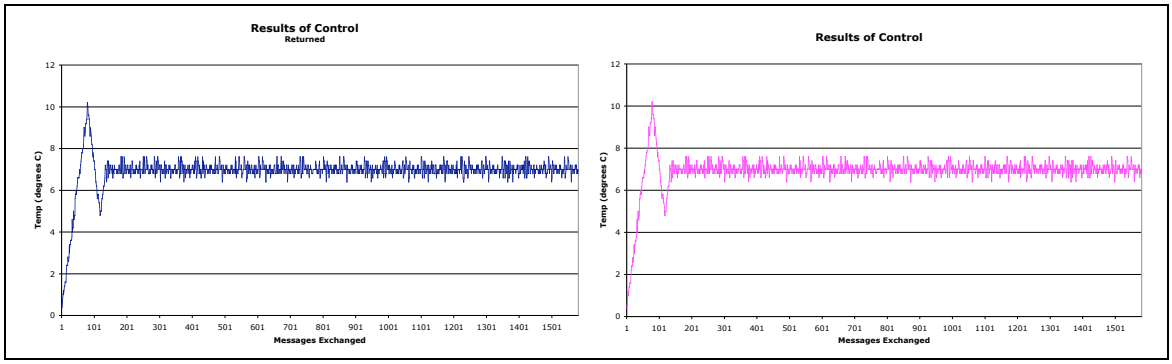

Figure 2: Returned and Actual Temperature From Control Experiment

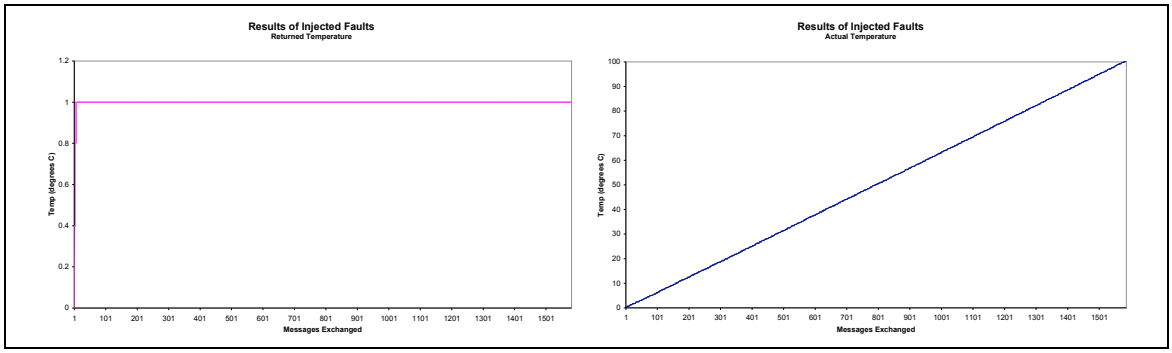

Figure 3: Returned and Actual Temperature From WS-FIT Experiment

machine running the heater coil service it could be used to certification test the thermocouple or controller.

Our system is set up to certification test the heater coil service so we have chosen to position the instrumented SOAP stack on the machine running the thermocouple service (see Figure 1). In this way we can monitor the output of the thermocouple driver and inject faults into the messages received from the heater coil (without modifying the heater coil code or environment).

4.3. Constructing Test Scripts

WS-FIT uses a script to provide both a trigger for fault injection and also the fault injection itself. The WS-FIT GUI can be used to create skeleton scripts by parsing the WSDL for a web service and allowing the user to set triggers on messages and parameters. The user can then complete the test script by entering fragments of script to perform fault injection. The GUI can generate a complete test script from this information.

In our scenario we will construct a script to monitor temperature response messages between the thermocouple and the controller. We will also construct a trigger to inject a fault into the get power responses received by the thermocouple from the heater coil after the temperature has reached a certain limit. By modifying this response to give a constantly low value we will attempt to force the controller to continually increase the power emitted by the heater coil, thus causing the heater coil to exceed its maximum power.

4.4. Running Experiment

Test scripts generated by the WS-FIT GUI can be executed natively from the GUI. Test scripts are python based so the Jython package [Pedroni and Rapping 2002] is used to execute them and allow interoperability with the Java JVM used to implement WS-FIT.

Our experiment uses two test scripts. The first is a control script that passes all messages through unaltered to their destination. It is used for monitoring SOAP flows. The second script is the one described in Section 4.3.

While running the test script, the fault injector framework logs a variety of data including unmodified and modified messages in an XML log file.

4.5. Analysis Of Data

From the above test runs we collected two series of data, one for the control experiment and the second for the fault injection experiment. We then created an analysis program to extract the

required data from the log files, basing this upon a supplied skeleton analysis program. Again the analysis program could be run natively within the GUI under Jython.

For both series of data our analysis program extracted both modified and unmodified power. It also extracted the temperature returned by the thermocouple to the controller. This data was then plotted graphically.

The control data shown in Figure 2 clearly shows that the system is functioning according to specification with only random variations (introduced deliberately as part of the simulation).

The fault injection data in Figure 3 shows a clear problem with the design of the system. At the point where the fault injection starts the temperature drops to 1˚C and holds at this temperature. The controller is written in a relatively simple fashion. According to its criteria the temperature is to low so it keeps ramping the power to increase the temperature. The heater coil soon exceeds its maximum operating temperature and in a real system would have malfunctioned.

This scenario could be caused, under real world conditions, by a thermocouple malfunctioning and thus causing an invalid reading to be received. It would indicate that some form of fault tolerance mechanism is required in the system, e.g. a piece of guard code in the heater coil driver service.

5. Conclusions and Future Work

Promising work has been carried out on applying network level fault injection to SOAP RPC based systems, both web service based and OGSA GRID [Looker and Xu 2003a; b]. WS-FIT is designed to integrate into SOAP based web service systems. It injects meaningful faults into specific SOAP messages via a script based trigger system. By injecting meaningful faults into RPC SOAP messages WS-FIT can simulate API level faults without the need for modifying code or running a test harness. With strategic placement of instrumented SOAP stacks on web servers WS-FIT can be used as part of a certification process for a running production system.

Our research using WS-FIT has shown that it can effectively be used to detect defects in system designs by exercising seldom used code pathways via fault injection techniques.

6. Acknowledgements

This work was supported by the EPSRC through a DTA studentship.

7. References

Curbera, F., M. Duftler, et al. (2002). "Unraveling the Web Services Web: An Introduction to SOAP, WSDL, and UDDI." IEEE Internet Computing **6**: 86-93.

Dawson, S., F. Jahanian, et al. (1996). "ORCHESTRA: A Probing and Fault Injection Environment for Testing Protocol Implementations". International computer performance and dependability symposium, Urbana-Champaign; IL, Los Alamitos CA.

Dawson, S., F. Jahanian, et al. (1997). "Experiments on Six Commercial TCP Implementations Using a Software Fault Injection Tool." Software Practice and Experience **27**: 1385-1410.

Han, S., K. G. Shin, et al. (1995). "DOCTOR: An IntegrateD SOftware Fault InjeCTiOn EnviRonment for Distributed Real-time Systems". International computer performance and dependability symposium, Erlangen; Germany, Los Alamitos CA.

Hecht, H. and M. Hecht (1996). "Qualitative Interpretation of Software Test Data". Computer-aided design, test, and evaluation for dependability, Beijing, International Academic Publishers.

Looker, N. and J. Xu (2003a). "Assessing the Dependability of OGSA Middleware by Fault Injection". The 22nd Symposium on Reliable Distributed Systems, Florence, Italy, IEEE.

Looker, N. and J. Xu (2003b). "Assessing the Dependability of SOAP-RPC-Based Web Services by Fault Injection." 9th IEEE International Workshop on Object-oriented Real-time Dependable Systems: 163-170.

Marsden, E., J. Fabre, et al. (2002). "Dependability of CORBA Systems: Service Characterization by Fault Injection". Symposium on reliable distributed systems, Osaka, Japan, IEEE Computer Society.

Marsden, E. and J. C. Fabre (2001). "Failure Mode Analysis of CORBA Service Implementations." Lecture Notes in Computer Science(2218): 216-231.

Pedroni, S. and N. Rapping (2002). "Jython essentials". Sebastopol, Calif., O'Reilly.