DEVELOPING A FAULT ONTOLOGY ENGINE FOR EVALUATION OF SERVICE-ORIENTED ARCHITECTURE USING A CASE STUDY SYSTEM

Binka Gwynne, Jie Xu
School of Computing, University of Leeds, UK

Keywords: Service-Oriented Architecture, Fault Injection Testing, Ontology.

Abstract: This paper reports on the current progress of research into the development and implementation of a Fault Ontology Engine. The engine was devised to facilitate the testing and evaluation of Service-Oriented Architecture (SOA), using ontologically supported software fault injection testing mechanisms. The aims of this research stem from the importance of system evaluation and the notion that testing and evaluation methods could be better supported for modern distributed systems by autonomous software machines, due to their potential dynamics, size, and complexity of SOA, and the variety of resources they offer. Fault injection testing is still generally in the domain of human expertise, experience and intuition, and machines require knowledge of testing mechanisms to develop testing strategies, with information that is formal, explicit, and in languages that they can interpret. This paper contains descriptions of experimental work carried out in order to generate information for modelling the fault and failure domains of a real-world case study system. Information from this case study system will be used to identify points of interest and target points for subsequent tests. It is hoped to show that inferences taken from known systems can be used for intelligent testing of unknown systems.

1 INTRODUCTION

This research brings together the concepts of ontology, Service-Oriented Architecture (SOA) and fault injection testing in order to develop a Fault Ontology Engine (FOE) with information gained from experiments on real-world case study systems.

Ontologies are formal and logical descriptions of domains used by software to support intelligent communication. There is a great diversity in the design of ontologies and how they represent the real world (Noy *et al.* 1997), but from a computing perspective, ontology is a logical theory which gives an explicit, partial account of a conceptualisation (Corcho *et al.* 2003).

An SOA is a type of software architecture comprising services, with emphasis on service interoperability and location transparency, with the aim of achieving loose coupling among interacting software resources. SOA middleware obscures the nature of resources by design, but this can lead to middleware that may be complex and problematic to implement, and systems that are difficult to evaluate. In consequence novel methods are required for testing and evaluation of SOAs.

Fault injection is a method of testing and evaluating computer hardware and software systems by deliberately inserting them with artificial faults in order to reveal faults more effectively than through observation methods.

2 BACKGROUND

2.1 System Boundaries

A system boundary describes a system's limit of self-determination and how interactions (inputs and outputs) occur between that system and the external side of its boundary. System boundaries are also concerned with those interactions existing between adjoining systems and how systems interact with their environment: the environment may be considered a system in its own right. System boundaries are important in testing and evaluation because boundary conditions are determinable and

Gwynne B. and Xu J. (2006).
DEVELOPING A FAULT ONTOLOGY ENGINE FOR EVALUATION OF SERVICE-ORIENTED ARCHITECTURE USING A CASE STUDY SYSTEM.
In *Proceedings of the First International Conference on Software and Data Technologies*, pages 353-356
DOI: 10.5220/0001318803530356

system failures are deemed to be observable deviations from a system's specified behaviour at its boundary.

2.2 Ontologies

Ontologies are essentially concerned with the nature of existence. They can vary from simple schema to controlled vocabularies, thesauri, hierarchies of terms, taxonomies, to full conceptualisations of domains. Ontologies are commonly used as intelligent communication media for machines (Mizoguchi and Ikeda 1997), and according to Duineveld et al. (1999), ontologies promise:

"*A shared and common understanding of some domain that can be communicated across people and computers*."

2.3 Service-Oriented Architecture

An SOA is a type of software architecture; it is essentially a collection of services with emphasis on interoperability and location transparency.

According to the World Wide Web Consortium (2006), an SOA is

"*A set of components which can be invoked, and whose interface descriptions can be published and discovered*."

An SOA is effectively a mechanism for connecting resources on demand by providing a more flexible, loose coupling of components than provided by traditional distributed systems architectures. SOA comprises three basic elements: service provider, service consumer and service repository. A service is defined by how it can be accessed by other components in its system. Access is achieved though a common interface by an exchange of messages with the interface hiding the details of how each service is actually implemented. Services are expected to be self-contained and not expected to have knowledge of the context of other services. Further, the granularity of services is unspecified and can vary, with a service hosted on a single machine or distributed over a network.

SOA middleware is software specifically designed to support interoperable machine-machine interactions over networks and provide common approaches for service definition, publication and use. SOAs can have problems associated with scale and dynamics and its middleware may be complex and difficult to implement (Looker *et al.* 2005).

2.4 Faults, Errors and Failures

A system failure occurs at a point in time where the condition of a system is different to its expected condition. Errors are responsible for causing failures, and in turn are due to the presence and activity of faults; a fault is a defect of some kind that exists within a system and may be active or dormant. An active fault, or combination of active faults, may produce an error, or combination of errors, whose visibility outside a system boundary is manifested as a failure. This is a well-established, recursive way to describe the relationship between failures, errors and faults (Avizienis *et al.* 2004). However, although inextricably linked, failures, errors, and faults are conceptually very different entities. A failure is a type of event and a fault a type of cause; both can exist as real-world entities. An error, however, is a type of state and therefore has a virtual existence.

2.5 Evaluating SOA Systems

Organisations offering computing resources need mechanisms to demonstrate the high dependability of their resources in comparison to rival systems. This is especially true for SOA based systems, where the details of service implementations are architecturally hidden.

There are a number of ways to evaluate systems including simulation modelling, stress testing and observation over time. Fault injection is a method of testing and evaluation where artificial faults are deliberately inserted into systems, in order to reveal faults more effectively than through observation (Voas and McGraw 1998), producing errors and observing how they propagate through to failures.

Fault injection is: 1) a faster and more logical way to detect faults than observation; 2) able to test systems at run time; 3) able to test dynamic, distributed systems; 4) allows the use of non–invasive methods such as interception of method calls and data corruption; and 5) able to evaluate how well different types of systems maintain their dependability over time (Arlat *et al.* 2003, Looker *et al.* 2005).

3 ONTOLOGICAL SUPPORT

By definition, autonomous software machines are self-managing and can adapt to changing conditions, which means they would provide good support in

evaluation of large or complex distributed systems such as SOAs as these systems can become dynamically modified during tasks. Fault injection is a fast and logical way to detect faults and can be used to test systems at run time, so is a suitable method for testing dynamic, distributed systems. It can also be non–invasive, using mechanisms such as interception of method calls and data corruption. However, fault injection is still generally in the domain of human expertise, experience and intuition, and machines require knowledge to develop their own testing strategies, and require information that is formal, explicit, and in languages they are able to interpret. By using ontologically supported fault injection we aim to: 1) make information on testing and evaluation methods available in machine readable form; 2) improve fault representativeness; 3) avoid spurious testing; 4) lessen the chances of producing undetected spawned faults; and 5) address problems associated with some traditional testing methods such as state and timing.

In our experiments fault and failure ontologies are used to guide more targeted fault injection tests through the analysis of cause and effect pairs as experimental data, including provenance, provides the information for FOE. In addition, is intended that FOE will be used in future in combination with other tried and tested software fault injection tools designed for SOA, such as WS-FIT (Looker *et al.* 2005).

4 A CASE STUDY

4.1 Secure Power System

A manufacturer of battery based AC and DC secure power systems was chosen for the case study. The principal markets for their products are industries involved in telecommunications, power generation and distribution, and petrochemicals. Uninterrupted, secure power units are vitally important to these industries, for example, the failure of PA systems on oil rigs was implicated as partially responsible for the loss of life in the Piper Alpha disaster, 1988 (UK Health and Safety Executive 2002).

The case study system is a secure power system that maintains a power supply for mobile telecommunications. The control units are mainly located on roof tops, are subject to sporadic mains power failures for generally short periods of time, with unit housing often subject to high ambient temperatures. The control units are remotely monitored.

These secure power systems can be maintained by one of two types of battery: Valve Regulated Lead Acid (VRLA) and Nickel-Cadmium (Ni-Cad). The VRLA battery is the newer technology, comprising a sealed system which does not require topping up by hand. However this battery requires more monitoring than the Ni-Cad battery due to physical vulnerabilities, although monitoring can be carried out remotely. In order to take advantage of the low maintenance required for VRLA, it is necessary for the system to maintain a high level of reliability through reliable monitoring.

Typical information monitored is temperature, voltage levels in batteries / cells, power input and power output to load etc., but monitors can also warn of such things as a door to the unit housing being open (a potential source of damaging heat input).

4.2 Fault Model

System fault and failure taxonomies were heuristically drawn up from known information on how the case study system behaves. Elements in the fault model were predicted failure-value pairs. For example: a voltage “Stuck-at” fault; this could lead to the alarm not sounding when the condition point is reached and the switch not closing to load at EOD and the hardware being damaged. The outcome of this example may be that the fault is sometime later observed; the failure could be classified as “Observed failure, non-recoverable, known cause, known effect”, and the scenario fault-failure pair would be “voltmeter stuck-at fault” and “hardware, cells/battery damaged”.

4.3 Experiments

A faulty voltmeter was simulated in the secure power system. The simulation involved injecting faulty values into a data set with the results logged in XML-based files. Each test carried out was given a unique ID and information was stored in logs, in order that information would be available and tests replicable later.

When fault injection testing is carried out there is an intention to guarantee that failures occur so that they can be observed. However, naturally occurring or programmed fault tolerance in a system can sometimes make it difficult for failures to occur where only one fault is present and it may become necessary to use a combination of faults, or composite fault, to bring about an observable failure. Consequently, experiments were also carried out that

simulate this situation, with each composite fault paired with its respective outcome.

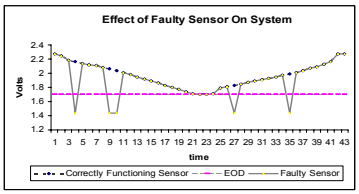

Figure 1: Injected Fault - Voltmeter Data.

In the next experiment the sensor tested was the thermometer for the unit housing, which is monitored in order to prevent damage to battery cells from high temperatures. An alarm fires when the temperature approaches 20 deg. Celsius, caused by a hardware fault in the unit housing, which leads to a temperature increase in the system. In this scenario the assumptions are that the sensor (thermometer), alarm and housing are all functioning correctly. The fault is in the refrigeration unit and for a short period of time the temperature exceeded 20 degrees C, the consequence is that there is slight hardware damage sustained before the fault is rectified externally (by human intervention). The fault is recovered, due to the correct function of the alarm and monitor, and some external action occurring to rectify the problem when the alarm sounds.

5 FUTURE STUDY

Information from these experiments is being analysed and stored in machine readable form, so that FOE can use it to predict future points of interest in the system fault domain. These points of interest will form the target points for further tests as FOE uses its information to make decisions on the next phase of testing and evaluation.

The information obtained from injecting faults with predictive outcomes into this known system is leading on to the design of further test scenarios to discover more of this system's fault and failure domains. It is hoped to build on to this information to develop new testing strategies for unknown systems and a new mechanism to glean information about their fault and failure domains.

Further experiments will be carried out that continue with the case study, and a new case study will be introduced, to determine where inferences from the first case study are useful to discovering information about other systems. This should demonstrate that experiments on known systems can be used investigate the fault and failure domains of unknown systems. The next progression in this research is to use this information to test and evaluate the reliability of SOA-based systems.

REFERENCES

Arlat, J., Crouzet, Y., Karlsonn, J., Folkesson, P., Fuchs, E., and Leber, G. H. (2003). Comparison of Physical and Software-Implemented Fault Injection Techniques. *IEEE Transactions on Computers* 52(9)

Avizienis, A., Laprie, J.-C., Randell, B., and Landwehr, C. (2004). Basic Concepts and Taxonomy of Dependable and Secure Computing *IEEE Dependable and Secure Computing*, vol. 1, pp. 11-33, 2004

Corcho, O., Fernandez-Lopez, M., and Gomez-Perez, A. (2003). Ontologies for Conceptual Modelling: Their Creation, Use, and Management. *Data and Knowledge Engineering* 46: 41 – 64

Duineveld, A. J., Stoter, R., Weiden, M. R., Kenepa, B., and Benjamins, V. R. (1999). Wondertools? A Comparative Study of Ontological Engineering Tools. *In Proceedings of Knowledge Acquisition, Modelling and Management (KAW99)*

Looker, N., Gwynne, B., Xu, J., and Munro, M. (2005). An Ontology-Based Approach for Determining the Dependability of Service-Oriented Architectures. *In Proceedings of the 10th IEEE International Workshop on Object-Oriented Real-time Dependable Systems*

Looker, N., Munro, M., and Xu, J. (2005). Simulating Errors in Web Services. *International Journal of Simulation: Systems, Science & Technology*

Mizoguchi, R. and Ikeda, M. (1997) Towards Ontology Engineering. *In: Proceedings PACES/ SPICIS International Conference on Intelligent Systems*

Noy, N. F., and Hafner, C. D. (1997). The State of the Art in Ontology Design. *American Association for Artificial Intelligence* (Fall): 53 - 74

UK Health and Safety Executive, Link Associates International Limited for UK HSE. (2002). *Inspecting and Auditing the Management of Emergency Response*. Norwich, UK: Crown Copyright

Voas, J. M. and McGraw, G. (1998). *Software Fault Injection* New York, USA: John Wiley

World Wide Web Consortium (2004). *Web Services Glossary* [online]. Available from: http://www.w3.org/TR/ws-gloss/ [Accessed: 7 February 2006]