

Experiment Management for the Xception Fault Injection Technology

Telmo Menezes
Critical Software, Lda.
telmo@criticalsoftware.com

Diamantino Costa
Dept. of Computer Engineering,
University of Coimbra, Portugal
dino@dei.uc.pt

1. Introduction

This abstract presents the general architecture of the Experiment Management Environment of Xception, a commercial Software Implemented Fault Injection (SWIFI) tool. Fault injection experiments are carried out in the Xception testbed, which comprises the target system (the system to evaluate) as well as a host system, and the fault injection software framework itself - Xception.

Xception consists of two main modules: the Injection Run Controller (IRC) module, and the experiment management (EM) module. The IRC runs on the target system and contains the actual fault injection code. The experiment management module runs on a remote host system, and will be described throughout this document.

The EM's high-level, 100% platform independent nature contrasts with the IRC. In fact, the IRC has dependencies on the target processor and OS, thus requiring some degree of customization (although the IRC kernel itself is written in portable C, there's assembly glue code to deal with). While work is in progress to customize Xception to several operating systems/machine architectures, the EM offers a unifying user interface to the technology.

2. Background

The failure of critical computer-driven systems has dire consequences. It threatens human life in aerospace, railway control, medical life-support, industrial plant control, nuclear power plants, and defense. On the other hand, even a few minutes of downtime can inflict tremendous economic losses on business at all levels, but especially in telecommunications, banking, insurance, and any industry that runs 24 hours a day, 365 days a year. Nowadays, computer systems have to provide the expected service despite the presence of faults--this is the purpose of fault tolerance. But before a fault-tolerant system is deployed, it must be tested and validated. This is the purpose of Xception.

The actual trend in fault injection technology goes through the use of SWIFI

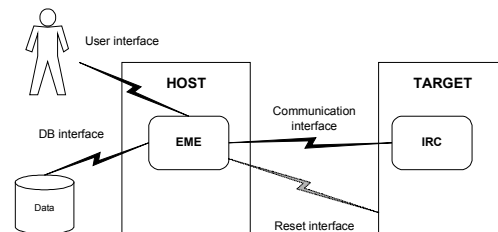


Figure 1. Xception Testbed Architecture

tools. The advantages of SWIFI tools are manifold. They use real hardware and software, they are less complex and costly, and incur in less development effort than other techniques. They are easily expanded (for new classes of faults), quite portable, and finally there is no problem with physical interferences or risk of damaging the target system as in physical fault injection.

3. Features

The Xception EM is the component of the Xception framework that runs on the Testbed host system, as opposite to the IRC that runs on the target system (where faults are actually injected). As shown in *Figure 1*, the EM runs in a host computer, controlling the IRC (Injection Run Controller) both through a network connection and a reset interface (in case one is present). It also connects to an SQL database through JDBC and provides a graphical user interface.

Specifically, The EM handles: (1) fault injection experiment definition, (2) fault injection execution, and (3) fault injection outcome archive and analysis.

During the experiment definition phase the operator provides the set of parameters that completely define a fault injection experiment, such as the number, type, location, and trigger of faults, among many others.

During the experiment execution phase, the EM systematically executes the so-called *fault injection runs*. In each *run*, a user-defined workload (benchmarks) is executed in the target system, a predefined fault is injected (may be a burst of faults), and its impact in the target system monitored. Meaningful results are then collected, and the target system reset before

proceeding to the next *fault injection run* and starting all over again.

Finally, in a third phase, fault definition data as well as experiment results archived in the systems' SQL database may be evaluated offline by the operator in order to obtain statistical figures, spot system weaknesses, or to accomplish other specific experiment goals.

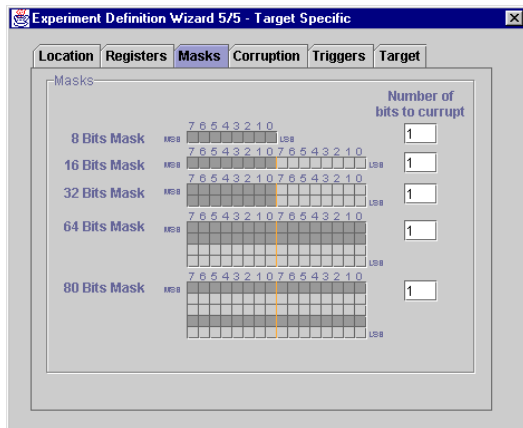


Figure 2. Experiment Manager GUI Screenshot

Figure 2 shows a screenshot from the experiment definition wizard.

4. Architecture

The EM was designed to be as much platform independent as possible and to be easily expandable to support any target system architecture and network protocol.

Java's portability makes it the natural choice to address the first requirement, and its object-oriented nature and dynamic class loading features a great choice to address the second one.

Little has to be said about Java portability and its Write-Once-Run-Everywhere capabilities. It's fitness for writing "plug-and-play" software modules, on the other hand, is not so straightforward.

The EM architecture enables specific target system support (for new processors or new OS's) to be implemented through independent software modules. Each of these software modules is a directory containing a set of compiled Java class files. Installing a new module is as simple as placing this directory under EM's main class directory. When the EM is started, the new module is detected, classes dynamically loaded, and made available at runtime.

Furthermore, the API for developing new target support modules is defined by a set of abstract classes. These classes must be extended, in order to implement methods which define target specific experiment definition and

injection procedures (e.g. a new target processor register file).

Target specific modules are therefore cleanly integrated while not being part of the application core, as they are dynamically loaded at runtime.

The EM GUI is based on the Swing package, which is becoming a *de facto* standard in Java GUI development. On the other hand, database connectivity was provided through JDBC. It is worth to note that both JDBC and Swing are part of the current Java Development Kit (JDK) distribution. In fact, it has been a requirement to the design of the EM to make use only of standard, 100% pure Java classes. The goal is to make deployment of the EM as simple and problem free as possible.

5. Xception on the Field

Xception has already been used in many and distinct target system architectures, from parallel to embedded systems, and from desktop to real-time OS's. The increasing maturity achieved by the tool expresses Critical Software's commitment to provide long-term support to the Xception technology. One challenging effort currently underway is the customization of Xception for use on the aim of the Jet Propulsion Laboratory's REE project [5].

6. References

- [1] João Carreira, Henrique Madeira and João Gabriel Silva, "Xception: A Technique for the Experimental Evaluation of Dependability in Modern Computers", in *IEEE Transactions on Software Engineering*, February 1998, pp. 125-135.
- [2] João Carreira and João Gabriel Silva, "Why do Some (weird) People Inject Faults?", *ACM Software Engineering Notes*, January 1998, pp.42-43.
- [3] João Carreira, Diamantino Costa, João Gabriel Silva, "Fault-injection spot-checks computer system dependability", *IEEE Spectrum*, Vol. 36, No. 8, August 1999, pp. 50-55.
- [4] Diamantino Costa, João Carreira, João Cunha, Telmo Menezes, "Xception: Professional Fault Injection", *White Paper, Critical Software*, 1999, available at <http://www.criticalsoftware.com>
- [5] John A. Rohr, "Software-Implemented Fault Tolerance for Supercomputing in Space", FTCS-28 - *International Symposium on Fault-Tolerant Computing*, June 1998