

# On the Extension of Xception to Support Software Fault Models

Telmo Menezes, Diamantino Costa and Miguel Tavares

Critical Software, S.A.

{telmo, dino, mtavares}@criticalsoftware.com

## 1. Introduction

Software faults are recognized as the major cause of system outages. The two possible approaches to overcome this problem are fault avoidance and fault tolerance. Quality assurance techniques fail to attain the zero defects mark, making fault tolerance vital to assure mission and business critical systems dependability. One major issue is the difficulty in the verification and validation of software fault tolerance modules. This is where software fault injection comes in. We propose the extension of the Xception [3] SWIFI (Software Implemented Fault Injection) tool to emulate software faults. Xception was designed to emulate hardware faults, but recent studies ([1] and [4]) showed promising results in applying the same technology on the emulation of software faults.

## 2. Emulation of Software Faults Using SWIFI

The available approaches to emulate software faults using SWIFI might be classified according to the time instant chosen to insert the errors deemed to be caused by faults. Two classes exist: compile time instrumented software implemented fault injection (CTSWIFI) and run-time software implemented fault-injection (RTSWIFI) [4]. Techniques based on compile time instrumentation can only be used when the target system source code is available. SafetyNet [5] and other mutation tools [6] are remarkable examples of the CTSWIFI approach. Most COTS components do not have available source code, making for a great limitation on the usability of this class of tool. RTSWIFI can be seen as an extension of low intrusive SWIFI techniques that have been used so far to emulate software faults, as is the case of Xception.

The Orthogonal Defect Classification proposed in [2] divides software defects on the following types: Function; Assignment; Interface; Checking, Algorithm and Timing/serialization. Two more classes are considered (Build/package/merge and Documentation) but these types of defects are out of the scope of our current work. An experimental study [1] has been performed to access the capabilities of the Xception SWIFI tool to emulate software faults. This study revealed three kinds of software faults from this perspective: i) faults that can be accurately emulated; ii) faults that can be emulated if the tool is improved with extra fault triggers, fault models and tools for automatic fault definition; iii) faults that could never

be emulated. This experimental study determined the current ability of the Xception SWIFI tool to emulate the ODC classes of software faults.

Fault Type	Emulation Status
Function	Impossible with any SWIFI tool
Assignment	Some possible / others possible with Xception improvements
Interface	Possible with Xception improvements
Algorithm	Impossible with any SWIFI tool
Checking	Possible
Timing/Serialization	Uncertain
Trigger	Emulation Status
Normal mode	Already supported
Recovery/exception handling; System start/restart; Workload volume/stress; Hardware/software configuration	Possible to implement

**Table 1 - Current Xception's ability to emulate software faults**

Checking faults and most assignment and interface faults may easily be emulated by changing machine code in memory as soon as the program text is loaded<sup>1</sup> or by changing the operand fetch every time the relevant instruction is executed. This later case might be modeled as a data bus fault or a memory fault in Xception [3]. The first option seems to be more interesting since it uses a simpler trigger. It is important to note that the current implementation of Xception emulates transient hardware faults, while software faults are permanent by nature. Changing machine code instructions in memory as soon as the code is loaded provides permanent faults without any change in the Xception architecture, assuming that the code does not perform mutations on itself (which is a quite reasonable assumption with modern programming practices). Function and algorithm faults cause considerable changes at the machine code-level and cannot be emulated by SWIFI tools.

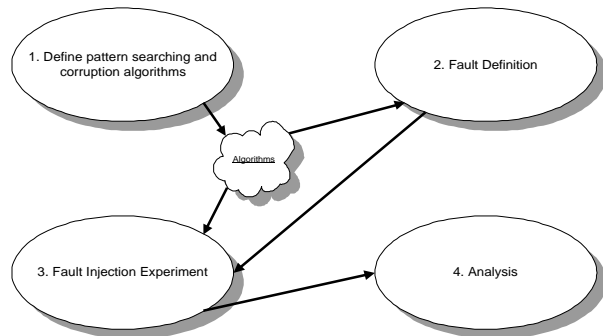
According to the ODC model a trigger condition is also needed to characterize a software defect. The current version of the Xception tool provides adequate triggers for the emulation of single event upset hardware faults, but these triggers do not represent

<sup>1</sup> In practice this may be implemented by setting the fault trigger to an opcode fetch from the first program code address.

realistic software fault triggers. The ODC model recognizes three sets of triggers. Two of them fall out of the scope of this work since they are related to review, inspection and test cases. System test triggers are the ones that matter when dealing with software fault injection. The only types of software fault triggers that can be defined with Xception at present are normal mode triggers. Normal mode triggers represent faults that result in failures during system operation in normal conditions.

### 3. SFEFI Methodology

Figure 1 depicts the workflow associated with the SFEFI (Software Fault Emulation by Fault Injection) technique. This is the technique we propose to implement software fault emulation on Xception. The layout is fairly similar to traditional software implemented fault injection.



**Figure 1 - SFEFI basic processes**

Step 1 is only needed when targeting a new architecture, compiler and/or source language and consists in defining a set of algorithms to be used for picking candidate fault/error locations and to modify the instructions according to predefined error types. It is important to note we use the designation “error type” here. That is because fault types must be instantiated to specific error types. An assignment fault, for example, may be instantiated to an off-by-one initialization error, a no initialization error or others. Step 1 requires the largest part of human skills in the whole process, being necessary to find how high-level language constructs map to machine instructions and define the algorithms to insert the errors according to the fault models. In step 2 the set of locations and error types are produced, based on the defined operational profile. The system is loaded with the application(s) and input data specified for the experiment and the SFEFI engine enters search mode. In this mode, a lightweight operating system kernel-mode extension samples periodically the instruction stream for known instruction patterns. When a match is found the instruction pointer is saved for posterior processing and optionally an error type might be settled according to the fault type in presence with an also optionally predefined error distribution. Steps 3 and 4 are analogous to the fault injection and analysis phases in SWIFI hardware fault injection.

### 4. Current and Future Work

Work is currently underway to implement a Xception interface with the GDB (the GNU Debugger). The immediate use of this interface is to provide an easy hardware fault definition tool, but it will also constitute an important step in the definition of software faults. The GDB delivers information on memory addresses of program functions and variables as well as profiling information, as long as the binary contains debugging symbols. Profiling information may also be used to assess the code coverage of a fault injection experiment during the analysis step.

We are committed to research ways of implementing software triggers other than normal mode in the near future. Some work has already been done on the Linux and Windows NT version of Xception to access the current processor load, leading the way to the definition of workload volume/stress triggers.

To overcome the problem of algorithmic/functional fault injection impossibility under SWIFI we intend to research the possibility of translating the machine code to a higher-level representation where algorithmic structures are easier to identify and mutate. This would be done during the fault definition step but the feasibility of this procedure is yet uncertain. During the fault injection step, the machine code would be changed in memory as soon as the code is loaded.

### References

- [1] H. Madeira, D. Costa and M.Vieira, “On the Emulation of Software Faults by Software Fault Injection”, Proc. of Intl. Conference on Dependable Systems and Networks, June 25-28, New York, USA, 2000.
- [2] R. Chillarege, “Orthogonal Defect Classification, Chapter 9 of “Handbook of Software Reliability Engineering”, Michael R. Lyu Ed., IEEE Computer Society Press, McGraw-Hill, pp.359-401, 1995.
- [3] D. Costa, J. Carreira, J. Cunha, T. Menezes et al., “Xception: Professional Fault Injection”, White Paper, Critical Software, 1999, available at <http://www.criticalsoftware.com>
- [4] D. Costa, T. Rilho, M. Vieira and H. Madeira, “SFEFI – A novel technique for the emulation of software faults”, Technical Report, CISUC – Center for Informatics and Systems of the University of Coimbra, June 2000.
- [5] J. Voas, F. Charron, G. McGraw, K. Miller, & M. Friedman, “Predicting How Badly “Good” Software can Behave”, IEEE Software, July 1997, Volume 14, Number 4, pp. 73-83.
- [6] Y. Crouzet, P. Thévenod-Fosse and H. Waeselynck, “Validation of Software Testing by Fault Injection: The SESAME Tool”, in Proc. 11th Conf. On Reliability and Maintainability, (Arcachon, France), pp.551-559, SEE, September 1998.
- [7] R. Stallman, R. Pesch, S. Shebs et al., “Debugging with GDB – The GNU Source-Level Debugger”, Free Software Foundation, Inc., 8<sup>th</sup> edition, March 2000.