mCrash: a Framework for the Evaluation of Mobile Devices' Trustworthiness Properties

José Ribeiro[1], Mário Zenha-Rela[2]

[1] Department of Informatics Engineering, Polytechnic Institute of Leiria,
2411-901 Leiria
jose.ribeiro@estg.ipleiria.pt
[2] Department of Informatics Engineering, University of Coimbra,
3030-290 Coimbra
mzrela@dei.uc.pt

Abstract. A rationale and framework for the evaluation of mobile devices' robustness and trustworthiness properties using a Windows Mobile 5.0 testbed is presented. The methodology followed includes employing software fault-injection techniques at the operating system's interface level and customising tests to the behaviour of the software.

1 Introduction

The philosophy for mobile devices has been evolving towards the "wallet" [1] paradigm: they contain important personal information, and virtually every adult carries one. They are true "proxies for the individual" [2]. Additionally, people are getting used to take care of their business affairs on these pervasive devices, since they are becoming increasingly more sophisticated and are able to handle most basic tasks. But not all mobile devices were designed with enterprise class security in mind, and even components which were specifically designed for mission-critical applications may prove to have problems if used in a different context. Retrofitting trust in any technology is considerably harder than building it in from the start [3], especially when users have already perceived it as invasive, intrusive, or dangerous. More emphasis should be placed on building robust systems that can adapt to aberrant behaviour.

However, software behaviour is a combination of many factors: which particular data states are created, what paths are exercised, how long execution takes, what outputs are produced, and so forth [4]. An operating system is, itself, a dynamic entity [5], as different services have diverse robustness properties; the way in which software makes use of those services will have impact on the robustness of their operations.

We believe that a tool for assessing robustness properties during the development phase by employing software fault-injection techniques will contribute decisively for the trustworthiness of the resulting software, as it is known that most of the computing devices' breakdowns are caused by residual software faults.

This framework's approach is that of integrating a fault-injection tool into the IDE – e.g. Microsoft Visual Studio 2005 – as a plug-in. The availability of the code under development allows for a profiling phase to take place, hence creating ground for making use of software fault-injection techniques to test the code in a tailored way and according to the developers' requirements. The methodology we will be following is that of applying fault injection instrumentation directly at the Windows Mobile 5.0 API - it is at the operating system's interfaces that corrupt data coming from the application under development will be simulated.
The next chapter outlines the proposed framework and provides an overview of its architecture; the third and last chapter presents topics for discussion and sets ground for future work.

2 Description of the Proposed Framework

Four fundamental modules embody this framework:

- The Faultload Database.
- The Input Generation and Fault Injection Module.
- The Postcondition Checker.
- The Execution Manager.

The Faultload Database

The process of building the Faultload database is executed offline, and must precede the actual testing phase, as a set of test values must be created for each unique parameter data type of every function made available by the Windows Mobile 5.0 SDK API. Test cases encompass both valid and exceptional values for the parameter data types, in order to mimic all sorts of events.
The first step is to catalogue all the API information, including all the functions, input and output parameters and their data types, and error codes. A parsing application that extracts information from the Windows Mobile SDK documentation automates this process.
The following step includes performing a domain analysis for each individual parameter in order to establish the Faultload. The DWORD data type, for example, is a typedef for an unsigned long. Therefore, test values could include its boundaries, and some randomly selected valid and invalid values. For pointer types, values such as NULL, -1 (cast to a pointer), pointer to freed memory, and pointers to malloc'ed buffers of various powers of two could be used [6].

Input Generation and Fault Injection Module

The Input Generator Component dynamically generates test cases for a given set of functions and their parameter data types. The function lists are fed to this component

by the Execution Manager as a result of a profiling phase; test cases are built by drawing values from the pre-defined Faultload Database.
The Fault Injection Component's approach is that of automatically and exhaustively testing combinations of parameter test cases by nested iteration. The testing code is generated given just the function name and a listing of parameter types. A test object is instantiated for each of the function's parameters data types, and is responsible for creating all testing infrastructure; the constructor runs the instructions needed to generate the test case. This methodology uses test objects to encapsulate all the test case generation complexity - hence avoiding the need to adapt test cases to a particular function - and allows for a considerable amount of system state to be set.

Postcondition Checker

The Postcondition Checker monitors the environment for unacceptable events. Assertions are put in two main places: the system level and the output level. The results yielded by the testing phase will be mapped to low-level categories (similar to those depicted in the C.R.A.S.H. Scale [7]).

Table 1. C.R.A.S.H. scale.

Value	Description
Catastrophic	the system crashes or hangs
Restart	the test process hangs
Abort	the test process terminates abnormally
Silent	the test process exits without an error code, but one should have been returned
Hindering	the test process exits with an error code not relevant to the situation
Pass	the module exits properly, possibly with an appropriate error code

Additionally, at the system level, global environmental events will be tracked using the State and Notifications Broker API [8] - Windows Mobile 5.0's state information store, which provides a standard architecture for monitoring state values for changes - which is a valuable tool for increasing the system's observability and for detecting actions that were uncalled for.
At the output level, Silent and Hindering failures values will be considered; this is why it is necessary - during the APIs cataloguing phase - for the output parameters and for the error codes to be identified.

Execution Manager

The Execution Manager is responsible for profiling the code - in order to correctly setup the Input Generation and Fault Injection Module with adequate usage scenarios - and for automating the collection and analysis of vulnerability information provided by the Postcondition Checker.

The profiling phase aims to identify the operating system's services used by the application under development – by means of an API tracing tool [9] - in order to generate ordered lists of the API functions; once the software behaviour is analysed,

it is possible to make use of software fault-injection techniques to test the code in a customized way. This phase is of paramount importance, as the number of test cases is determined by the number and type of input parameters, and is thus exponential with the number of functions.

The collection and analysis phase allows for the outputs monitored by the Post-condition Checker to be properly logged and mapped to the inputs previously produced by the Input Generation and Fault Injection Module.

3 Conclusions and Future Work

We believe that the presented framework will allow the detection robustness problems all the way through – and early on – the software development process, hopefully decreasing the need for adding software "wrappers" to baselined software, and thus reducing the possibility of introducing additional faults and lowering maintenance costs. However, several issues are still subject to discussion. Topics include:

- The feasibility of including all of Windows Mobile 5.0's APIs in the scope of the proposed framework.
- The relevance of implementing a parsing application to extract information from the Windows Mobile SDK documentation.
- The methodology for the generation of test cases.
- The tools to be employed during the profiling phase.
- The drawbacks and advantages of covering Silent and Hindering failures.

References

1. Satyanarayanan, M.: Swiss Army Knife or Wallet?. IEEE Pervasive Computing, vol. 4, number 2, pp. 2-3 (2005)
2. Abowd, D.: The Smart Phone: A First Platform for Pervasive Computing. IEEE Pervasive-Computing, vol. 4, number 2, pp. 18-19 (2005)
3. Langheinrich, M.: Privacy by Design - Principles of Privacy-Aware Ubiquitous Systems. Swiss Federal Institute of Technology, ETH Zurich, ch. 3. ACM UbiComp (2001)
4. Johansson, A., Suri, N.: Error Propagation Profiling of Operating Systems. International Conference on Dependable Systems and Networks (2005)
5. Voas, V., McGraw, G.: Software Fault Injection: Inoculating Programs Against Errors. John Wiley & Sons, Inc. (1998)
6. Kropp, N., Koopman, P., Siewiorek, D.: Automated Robustness Testing of Off-the-Shelf Software Components. Fault Tolerant Computing Symposium, Munich (1998)
7. Biyani, M., Santhanam, P.: TOFU: Test Optimizer for Functional Usage. Software Engineering Technical Brief (1997)
8. Wilson. J.: The State and Notifications Broker Part I. MSDN Library Online (2006)
9. Durães, J., Madeira, H.: Generic Faultloads Based on Software Faults for Dependability Benchmarking. International Conference on Dependable Systems and Networks (2004)