Test driven approach for porting software to mobile platforms

Pavel Zubarev, **Mark Zaslavskiy**, **Kirill Krinkin**
Open Source & Linux Lab
Russia
Email: {pavel.zubarev, mark.zaslavskiy}@gmail.com, kirill.krinkin@fruct.org

Abstract

Mobile platforms are growing day by day. With this growing they include more and more software. Test driven development is well known method which helps developers make products in time with high quality. Mobile development and software porting as well aren't exception and should be tested completely.

The presentation has four parts. First of all we will highligh main problems with mobile development, especialy cases with new software porting at MeeGo platform. The second part will discuss about simulator-based development advantages and disadvantages, describe aproach is introduced in MeeGo platform and targeted to delivery high quality. Third part will describe how test-automation tool for MeeGo platform, named Testrunner-lite, provides high quality for the platform. In the final part we will present experience of OpenSSL library integration into Testrunner-lite framework.

I. Introduction

Nowadays mobile devices got integrated to human life and we can say that they already became a part of human being. Any failure of mobile software or hardware could cause disbenefit for owner business of life. That is why manufacturers make high demands for hardware and software quality and robustness. This paper discusses automated software components testing on the MeeGo platform.

MeeGo is a software platform (it's better to say system platform) for wide class of devices like notebooks, tablets, mobile phones, TV sets, car computers and other embedded components. Wide spectrum of target devices requires unprecedented quality of software platform itself and applications which will be ran on it. There are hundreds of components are connected very hard into the MeeGo platform. Before deployment and publishing platform has to be verified so that manufacturer be sure that all requirements have been satisfied. Each component itself and platform have to pass set of verification procedures. Unit tests are used to check components quality, all applications, libraries and system functions have to be covered by tests.

II. Testing. Simulator vs Devices

We can consider MeeGo platform to reveal difficulties in verification of mobile platform, because the platform targets for big amount of different devices. In this case developer could use simulators for simulation device interfaces and behavior. One more advantage of using simulators is boosting testing speed (simulator can be ran on desktop, or even server, system which have more resources like CPU performance, memory size and so on). But unfortunately there is set of problems for testing mobile software on simulator.

The first of all, as usual simulator doesn't simulate hardware failures which can happened on the real device. That sort of failures could be connected with hardware characteristics of particular device. It's not a secret that every modern device has firmware which is stored into

controller and this firmware could be a reason of software errors. So, simulator is just an ideal model of the device. It could be useful for detecting logical software bugs but completely useless for testing error handling.

Second, simulator as a program can contain software bugs itself. This bugs impact on testing results and verification conclusion.

Obviously, the best approach to the testing mobiles software is the happy medium between these approaches: testing on simulator should be combined with testing on the real hardware. In this case it's required to unify tools and procedures for simulator and device.

III. TESTRUNNER FRAMEWORK

One of the main tool for unit testing for MeeGo platform is an utility testrunner-lite. It is a set of command line tools for testing on linux based platforms. There is a next terminology for test description:

- step – one step from testing script, minimal testing block. As usual it is ran by system shell ans testrunner detects return code for it
- test-case – set of several steps which are ran sequentially. It's possible not only Monitor overal success or failure, but even partial succes or failure for each step.
- set – logical union of set of test-cases
- test-suite – logical union of sets.

For test scenario unification special XLMSchema [3] has been developed. It allows describe all terms given above. This is a simple test case file:

```
<?xml version="1.0" encoding="UTF-8"?>
<testdefinition version="1.0">
  <suite name="simple test" domain="etc">
    <description>very simple test</description>
    <set name="simple tests set" feature="etc">
      <description>simple test</description>
      <case name="simple test 1" type="Functional" level="Component">
        <step>execute_test</step>
      </case>
    </set>
  </suite>
</testdefinition>
```

Testrunner-lite uses return codes for checking test success, which simplifies result analysis. Tool supports ability to specify expected return code for test scenario which as very flexible. Each scenario can contain number of commands. Separated steps could be combined in case and this cases could be combined in set. This multi-layered approach allows us to manage comprehensive test structures.

Report about test running results (test report[4]) also encapsulated in XMLSchema document. That approach allows verify concordance of testing plan to the fixed format.

File with tests description is required for running testrunner. It should be passed as first parameter in command line. Testing process is a sequence of test invoking in the current shell environment. The Testrunner creates record in results file for each running command, and stores return code. Full program output (stderr, stdout) is kept in the same file as well. This data can be used for failures investigation.

This characteristics of testrunner can be used for creating testing robot which will test full platform as a single object. There is one requirements for integration different modules

under testrunner scripting: each software component should except traditional deb package provide extra package with testing scripts. In this case testing robot will just install testing package and run testrunner against test descriptions which are provided by testing package. Initially testrunner has been created as cross-platform tool and from the begin it aimed to support desktop and embedded platforms. That is why this tool can be integrated to the MeeGo platform and it provide testing services and interfaces for developers not only on real hardware but on the simulator as well.

IV. OpenSSL integration with testrunner framework

OpenSSL is the one of most important component of huge amount of systems. This library provides basic functionality for security subsystem such as encrypting and decrypting. So, lots of applications from authorization tools to web-browsers depend on this library.

OpenSSL has been integrated with testrunner for MeeGo during a joint efforts Nokia and LETI and periodical testing has been arranged. This task is extremely important for platform stability. OpenSSL already has set of unit tests and we used this tests for integration with testrunner.

Initially test set has been developed on Perl and make. In the original OpenSSL Makefile has target test and this target is used for testing. That approach has next disadvantages:

- Tests are being ran in fixed order and it's impossible to test particular component collaboration;
- Tests are being ran during package creation, it means that testing is applied only for one (development) platform. It became important when we use crosscompiling.

Our main task was an integration existing testing system with testrunner. Development has been driven by next statements:

- current test system should be as is;
- all new tests have to be put into separated deb package and made autonomous;

The first stage of our work was investigation of existing OpenSSL test system. Actually it was test/Makefile analysis which contained test scenarios. Analysis showed that the project contains two types of tests: binary and scripts. Scripts are being used for testing for OpenSSL command line interface (CLI) and certificate management. Binary tests target encryption methods testing.

We had to port existing testing system for using with testrunner. The first action was replacing relative pathes in sources tree to absolute in target system scope. It has been domne by addind building time scripts which do replacement. So, OpenSSL testability was improved.

Next problem we met test dependency on some files in test set. Some tests fail due absence these files. We decided to add file copying to building process. Now deb packages with tests contains required files as well.

Another problem was connected with package naming scheme. On a target testing platform all packages have hardcoded names and it makes imposible to install two version of OpenSSL library, because it requires full platform rebuld that is useless task. We decided to add suffix ÏÏfor all files which was being integrated with testrunner on target platform.

Before this step test integration had proceed without openssl1-test package installation by specifying path to particular library version. Replacing conflict file names significantly simplified this process. But due new naming scheme we had to add adding suffix in the script tests.

On the porting stage, when we were extracting test invokes from Makefile to XML descriptions we weren't detect essential problems. Test scripts contained relative paths to the

console application openssl that was some obstacle for automated extracting for different running environment (when for instance, we run tests out of building directory).

On the package creation stage with tests for openssl and testrunner script came up a problem about dynamic paths substitution from relative to absolute. This was required for running testrunner and keeping legacy openssl testing system. Moreover, we had to determine all files which have been used by openssl tests so that we can avoid porting bugs.

During integration deb package with simulator wrong dependencies have been detected for testrunner-lite. The first installation attempt was failed because most of existing packages had hard dependencies on previous version fo openssl and cannot be rebuilt. We had decided to provide new naming scheme for OpenSSL so that we can Integration OpenSSL with MeeGo showed the same problems with names. It proves again that simulator is very important tool for detecting problems on pre-device testing.

V. What is next?

Our experience in OpenSSL integration with MeeGo platform testing framework revealed problems with existing OpenSSL building system. High level customization of this system makes maintenance too difficult. In the future we are going to add new tests and to make concomitant tools more flexible for testing on MeeGo.

VI. Acknowledgment

The authors would like to thank Finnish-Russian University Cooperation in Telecommunication Program (FRUCT, fruct.org) for provided support and especially professionals from Nokia Research Center for their consultancy work. This project is being developed in Open Source and Linux Lab (osll.fruct.org)

References

[1] Maemo Base Port requirements, http://wiki.forum.nokia.com/index.php/Maemo_Base_Port
[2] XmlSchema definition for TestRunner protocol, http://gitorious.org/qa-tools/test-definition/blobs/master/data/testdefinition-syntax.xsd
[3] Testrunner-lite sources, http://gitorious.org/qa-tools/testrunner-lite