Degree project specification

Adam Renberg
adamre@kth.se

June 7, 2012

School: School of Computer Science and Communication (CSC) at KTH
Company: Valtech, http://www.valtech.se/

Supervisor, Valtech:	Erland Ranvinge	erland.ranvinge@valtech.se
Supervisor, CSC:	Narges Khakpour	nargeskh@kth.se
Examiner:	Johan Håstad	johanh@kth.se
Coordinator, KTH:	Ann Bengtsson	ann@csc.kth.se

Contents

1 Short Background

Runtime verification (RV) is a light-weight formal verification technique, see e.g. [1, 2]. It verifies that specified properties hold during the execution of a program. It operates on a *trace* of the *current execution*, either while it is running (online), or at some other time and/or place (offline).

The properties that should be verified are specified in a constructed language, often a logic/calculus, such as Linear Temporal Logic [3]. An RV framework then instruments the target program with monitors so that the properties can be checked, and the execution verified against the specification.

When a violation against the specification occurs, simple actions can be taken (e.g. log the error, send emails, etc.), or more complex responses initiated, resulting in a *self-healing* or *self-adapting* system (see e.g. [4]).

Unit testing is the concept of writing small tests, or test suites, for the units in a program, be it functions, classes, etc. These tests are used during development to test the functionality of the units. They aim to reduce the risk of breaking existing functionality when developing new features or modifying existing code (by preventing regression).

Writing unit tests, often using unit testing *frameworks* such as JUnit [5] for Java and unittest [6] for Python, is a common practice on many development teams.

2 Problem

2.1 Problem Definition

How can runtime verification specifications be written in a manner that resembles the syntax of unit tests?

Suggested title: *Test-oriented runtime verification - using a unit test-like specification syntax for runtime verification.*

2.2 Motivation

Verification that a program works correctly is of great interest to software developers, and formal formal verification techniques can often help. Traditional approaches, such as *theorem proving*, *model checking* can be impractical with larger programs, and verification by *testing* is informal. Runtime verification can here be a lightweight addition to the list.

An important part of RV specifications is that they should be formal, and thus that the properties they specify yield a formal proof of correctness for the current execution.

The specification languages used by RV implementations is therefore often based on some formal logic and not written in the target program's programming language. In contrast, offline unit testing frameworks often utilize the programming language to great effect, and their use is wide spread.

If RV specifications more resembled unit tests, and were written in the target program's programming language, it might popularize the use of runtime verification.

3 Literature

3.1 Previous Research

Runtime verification is a somewhat new area of research, but the research on verification and formal methods goes back several centuries.

Unit testing is also quite young, perhaps having started in the 90s, and it isn't as much researched as formal methods.

The work on the Linear Temporal Logic, on runtime verification in general and its applications, on code instrumentation (e.g. [7, 8]), and on unit testing and their frameworks will lay the foundation of this work.

3.2 Examination

The literature study will be examined as a part of the report, as the report will contain both a general description of RV and unit testing, and more detailed sections on research related to the problem.

4 Method

The work will consist of three parts:

4.1 Related Work

I will do a background and state-of-the-art inventory of RV and unit testing. The focus will lie on the syntax used for writing the specifications/tests, and how they are executed and the target program instrumented.

4.2 Approach/Investigation

In this investigative part I will answer the following questions:

1. How should the syntax for the specification be defined, so that it looks similar to that for unit tests, but works for RV? Which language could be used? Which unit testing framework to take inspiration from?
2. How can this be related to a formal logic, to ensure the correctness of the specification? What formal logic should be used?
3. How should the code be instrumented to monitor the system and build the system model? How should the specification be transformed so it can be executed?
4. How will this be used to (online) verify the system against the specification?

I will focus on item 1 and 3.

4.3 Evaluation/Experiments/Implementation

In this part I will implement an RV framework prototype, based on the previous investigation.

If possible, I will also attempt to use the resulting framework to enable runtime verification on a project at Valtech.

5 Resources

If I attempt to try the prototype framework on a project at Valtech, I need to get access to such a project. One suggested possible project is the Valtech Intranet.

6 Scope

The the report will contain a section on how to relate the new specification syntax to a formal logic, thus showing that specifications written in the new syntax can be used for runtime verification.

TODO: What should be here? What will *not* be in the report?

7 Working Procedure

I will work on the degree project 50% and 50% on projects at Valtech, in periods of two weeks degree project, two weeks work. This fits well into the iteration-planning at Valtech. During the summer I will almost exclusively work on the degree project, and also use some of my vacation hours to work on it.

Other required work related to the degree project, such as doing the opposition of another student's work, will take time from the work above and require flexibility in planning.

I will keep a diary, `http://tgwizard.github.com/thesis`, in which I will write on my progress.

7.1 Approximate Time Plan

Start	Weeks		Work
May	*1w in May/June*	**1w**	Writing, and doing the research for, this specification.
May	*2w in May*, v23, v26, v27	**5w**	Doing background & research.
9/7	v28	**1w**	Writing the background part of the report.
16/7	v29, v30, v31, v32, v33	**5w**	Investigating and evaluating "what to do".
13/8	v34	**1w**	Writing about the investigation.
20/8	v35, v36, v37, v38	**4w**	Implementing and evaluating the RV framework. Possibly testing and analysing the RV framework on a project.
24/9	v39, v40, v41	**3w**	Writing and finishing the report.

I will take the weeks v24 and v25 off, as well as about 1 week in august (not planned). **The weeks I will be working at Valtech haven't been written into the time table.**

Total: 20 weeks. End date: **12 October 2012** (very optimistic).

8 Licensing

I will license any resulting code under some open source license, and any documentation, including the report, under Creative Commons Attribution 3.0 Unported License [9] (or something similar).

References

[1] Martin Leucker, Christian Schallhart. A brief account of runtime verification. In *The Journal of Logic and Algebraic Programming 78*. 2009, pages 293-303.

[2] Nelly Delgado, Ann Q. Gates, Steve Roach. A Taxonomy and Catalog of Runtime Software-Fault Monitoring Tools. In *IEEE Transactions on Software Engineering*. IEEE, 2004.

[3] Amir Pnueli. The temporal logic of programs. In *Proceedings of the 18th IEEE Symposium on the Foundations of Computer Science (FOCS-77)*. IEEE, 1977

[4] Markus C. Huebscher, Julie A. McCann. A survey of Autonomic Computing degrees, models and applications. In *ACM Computing Surveys*, Volume 40 Issue 3. ACM, 2008.

[5] Kent Beck, Eric Gamma, David Saff. JUnit. `http://junit.sourceforge.net/`, June 2012.

[6] Python Software Foundation. unittest. `http://docs.python.org/library/unittest.html`, June 2012.

[7] AspectJ. `http://www.eclipse.org/aspectj/`, June 2012

[8] Martin Matusiak. Strategies for aspect oriented programming in Python. 2009

[9] Creative Commons. Creative Commons Attribution 3.0 Unported License. `http://creativecommons.org/licenses/by/3.0/`