Using Quicktrace to collect runtime execution traces easily and automatically

Vincent Nelis
Luís Miguel Pinho
Using Quicktrace to collect runtime execution traces easily and automatically

Vincent Nelis, Luís Miguel Pinho

CISTER Research Center
Polytechnic Institute of Porto (ISEP-IPP)
Rua Dr. António Bernardino de Almeida, 431
4200-072 Porto
Portugal
Tel.: +351.22.8340509, Fax: +351.22.8321159
E-mail: 
http://www.cister.isep.ipp.pt

Abstract
Using Quicktrace to collect runtime execution traces easily and automatically

Vincent Nelis  
CISTER/INESC-TEC, ISEP  
Porto, Portugal  
Email: nelis@isep.ipp.pt

Luís Miguel Pinho  
CISTER/INESC-TEC, ISEP  
Porto, Portugal  
Email: lmp@isep.ipp.pt

In many application domains, it is an elementary step in the design of a software, typically before its deployment, to exercise parts of its functionality by running some of its code on the target platform and collect informations about its runtime behaviour. Those informations may be used for debugging purpose or to assess the responsiveness of the application for example.

Taking measurements and collecting runtime data and is typically a tedious process that comprises many smaller but more specific tasks such as, e.g. (1) upload the source code to be tested on the target device, (2) compile the code remotely, (3) parametrize the device and set up the execution conditions, (4) run the compiled program to be analysed, (5) download the collected information (called traces hereafter), (6) process and analyse those traces, and (7) display the results of the analysis. Those are just a few example steps that are typically found as part of a testing process.

Besides the tremendous time that it takes to execute all these subtasks, it is also very difficult to automatize the whole process, mostly because all those steps are generally implemented using different programming languages. For example it is not uncommon that all the remote operations such as compiling, linking, and running the program to be analysed are basic shell scripts run on the target device through SSH, whereas the scripts used to analyse and display the traces are typically written in Matlab, R, or any human-friendly language capable of generating nice-looking plots. As a result, each time new measurements are needed it is required to manually re-execute the whole chain of subtasks, which may be very tedious, error-prone, and time-consuming.

This paper introduces and show-cases our new and innovative tool called “Quicktrace”. Quicktrace has been developed in the scope of the European FP7 STREP project P-SOCRATES\(^1\) to ease the process of collecting runtime execution traces. However, the tool can be used for many other purposes as it offers a generic interface to easily implement, connect, and run together a set of scripts. Those scripts may be written in different programming languages and yet they can share common variables. Quicktrace is based on three main concepts: commands, actions, and variables.

\(^1\)The P-SOCRATES Consortium, P-SOCRATES (Parallel Software Framework for Time-Critical Many-core Systems): http://p-socrates.eu

1) The **commands** are the basic blocks of the tool. A command is defined in a specific programming language and has a pre-defined type. A command is typically a small script that is used to perform a very specific task and its type describes how it must be executed. For example, one can define the commands “Upload the source code to the testing device” or “Compile the code remotely”. The former is of type “SFTP – Put” and its code lists the files to be uploaded (following a pre-defined syntax), whereas the latter can be a Shell script of type “SSH - Remote command[s]”. The current version of the tool supports scripts written in R as well as in any shell script supported by the machine running the script. The next version will provide support for running scripts written in python and matlab. Files can be sent and received through SSH or SFTP and Shell scripts can be executed remotely through SSH. Figure 2 shows the “Commands” panel of the tool.

2) The **Variables** are defined by the user. They are simply characterised by a name, a type, and a value. Before a command is executed, Quicktrace performs a simple “search and replace” on the code of the command to replace every reference to a variable with its value. Therefore, variables can be used and accessed by every command, irrespective of its programming language, simply by referring to it as “@ {variable_name}” in the command’s code. Figure 1 shows the “Variables” panel of the tool.

3) The **Actions** are the mean to connect the commands together. They are defined by a set of commands and executing an action simply runs all its commands in the order defined by the user. Note that Quicktrace also provides special control-flow commands that allow to implement loops and if-statements. Those control-flow commands are kept relatively simple as the ambition of the tool is (for now) not to design a new programming language. Figure 3 shows the “Actions” panel of the tool.

**ACKNOWLEDGEMENT**

This work was partially supported by National Funds through FCT/MEC (Portuguese Foundation for Science and Technology) and co-financed by ERDF (European Regional Development Fund) under the PT2020 Partnership, within project UID/CEC/04234/2013 (CISTER Research Centre); also by the European Union under the Seventh Framework Programme (FP7/2007-2013), grant agreement n° 611016 (P-SOCRATES).
Fig. 1. In the “Variables” panel, the user can define all the variables that will then be shared by all the commands, irrespective of their programming language. Note that variables can be used within variables to allow for more flexibility.

Fig. 2. In the “Commands” panel, the user defines all the basic blocks/subtasks that will later be assembled to compose an action.

Fig. 3. In the “Actions” panel, the user defines new actions, the set of commands that compose them, and how they are sequenced within each action.