TIMERMETER USER AND PROGRAMMER GUIDE: Quantifying Accuracy and Invocation Costs of Software Timers accessed from Java Bytecode

Michael Kuperberg [1]

Abstract

TIMERMETER [1] is an novel approach for systematically obtaining the invocation cost and the accuracy (resolution) of timer methods. [1] describes the principles of TIMERMETER and a case study which applied a Java implementation of it to different timer methods. This manual documents this Java implementation of TimerMeter and provides guidelines on extending it. It also describes implemented enhancements and extensions that were included into TimerMeter after [1] was submitted. Finally, it provides a list of known issues and planned development steps, as well as a FAQ list.

Keywords: Timers, counters, timer methods, precision, accuracy, invocation costs, fine-granular measurements, clustering

1 Introduction

Text

2 Applying TimerMeter and Evaluating the Results

Text

3 Project Structure and Libraries

Text

4 Code Description and Documentation

Text

[1] Chair for Software Design and Quality, Universität Karlsruhe (TH), Karlsruhe, Germany, Email:mkuper@ipd.uka.de

This paper is electronically published in
TODO
URL: TODO

5 Extending and Modifying TimerMeter

Andere Timer; andere Plattformen; andere Sprachen

5.1 In Java 5 und später

(i) In the constructor, set `timerClassName`, `timerMethodName`, `timerMethodUnit` and `timerMethodIsStatic`
  - if the timer method has a non-time `timerMethodUnit` [2] (e.g. ”ticks” with a certain frequency), that frequency can be set using `setExternallyDeterminedTimerFrequency`
  - they are declared but not initialised in the `AbstractTimerMeter` class
  - note that the method name is not allowed to contain a dot (”.”), a semicolon (”;”), or a parenthesis (( ) { } < > [ ])
  - the `timerClassName` should be a fully-qualified class name referring to an non-abstract class. If `timerMethodIsStatic` is true, `timerClassName` does not need to have a constructor; otherwise, either (i) at least one [3] parameterless constructor of that class must be declared and accessible for TimerMeter or (ii) the `initialiseInvokableTimerMethod` must be overwritten [4] . The `initialiseInvokableTimerMethod` method is called internally by TimerMeter and is implemented in the `AbstractTimerMeter` class.

(ii) `obtainMeasurementsUsingDirectInvocation` must be overwritten (it is abstract in `AbstractTimerMeter` class)

(iii) `timerMethodToOverride` must be overwritten (it is called by `obtainMeasurementsUsingOverridenTimerMethod`, and it is abstract in `AbstractTimerMeter` class)

(iv) if the timer method's resolution is higher than its invocation cost and if that invocation cost has been determined using a more accurate timer (cf. `java.lang.System.currentTimeMillis`, s. case study in [1]), it is possible to set that invocation cost a `double` value using `setExternallyDeterminedTimerInvocationCost`.

(v) if the timer method's vendor provides a built-in facility to query the timer method's accuracy or resolution cost, this values can be stored in `AbstractTimerMeter` subclass using the methods `setOfficialTimerMethodAccuracy` and `setOfficialTimerMethodInvocationCost`, so that the unit (stored in `timerMethodUnit`) confirms with this information.

Using the command-line parameters passed to `run(String[] args)`, the execution of the algorithm can be controlled (w.r.t. number of measurements, clustering algorithm, etc.). The concrete subclasses of `AbstractTimerMeter` class are expected to implement `public static void main(String[] args)` by initialising their declaring class (a subclass of `AbstractTimerMeter`), and by invoking `run` on the initialised instance.

[2] It is planned to replace this field's current type (`java.lang.String`) with an `enum`

[3] TimerMeter takes the first parameterless constructor by default, as returned by the Java Reflection API; if another constructor must be taken, the `initialiseInvokableTimerMethod` method can be overwritten.

[4] It remains to be tested whether run(String args[]) that is final in class AbstractTimerMeter will use the subclass'

For modifying the algorithm itself, see the comments in the source code of the `AbstractTimerMeter` class.

5.2 For Java SE Versions 1.4 and earlier

RetroTranslator? Remove generics?

5.3 For Java ME and EE

Speicherbedenken? API-Einschränkungen?

5.4 In anderen Programmiersprachen mit Java Bytecode als Ziel

Ruby etc.? Jython?

5.5 In anderen Programmiersprachen

.NET? C/C++? (Windows-Timer wiederverwerten... :-))

6 Assumptions and Limitations

Text

7 Contributing and Future Development

Text

References

[1] Michael Kuperberg, Martin Krogmann, and Ralf Reussner. TimerMeter: Quantifying Accuracy of Software Timers for System Analysis. In *QAPL'09*, 2009.