Runtime Verification of Real-Time Properties of JamaicaVM

Clemens Ballarin
aicas GmbH

COOPERATION

ARTEMIS

Background

JamaicaVM

- Java platform for embedded systems
- Realtime-capable garbage collection
- Implements the Realtime Specification for Java (RTSJ)
- Available on Linux, QNX, VxWorks, Windows CE, ...

Scheduler

- Decides, which threads get executed
- Priority-based (fixed-priority scheduling)
- Co-operates with the scheduler of the OS

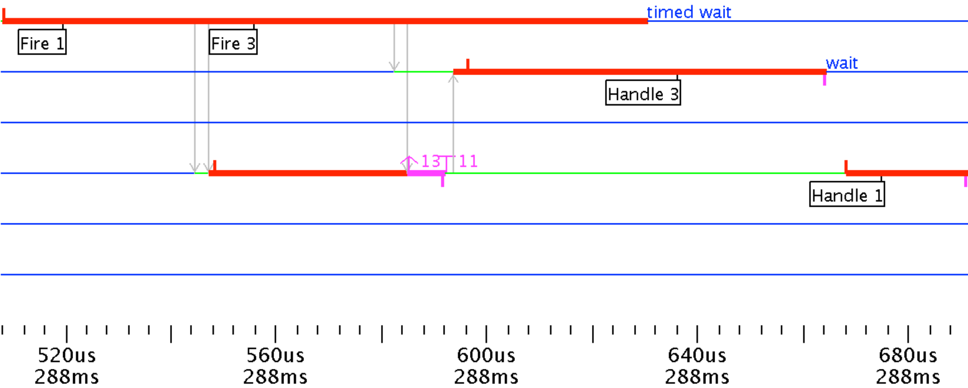
Goals

Systematic tests of JamaicaVM

- Implementation of the scheduler
- Implementation of the RTSJ
- Analysis of event sequences emitted by JamaicaVM

Development tool for JamaicaVM

- Detect insufficient synchronisation in application code



timed wait
Fire 1
Fire 3
wait
Handle 3
13
11
Handle 1
520us 288ms
560us 288ms
600us 288ms
640us 288ms
680us 288ms

Runtime Verification

Formal Methods

Static analysis

- Examines the model of a system
- Can prove the absence of errors in the model
- Cannot ensure the correctness of the actual system

Runtime verification

- Examines the actual system
- Cannot prove the absence of errors
- Can reliably identify errors when they occur

(Complex Event Processing is very much related.)

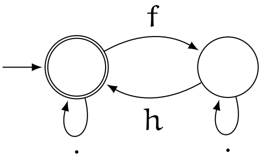
Modelling an Activity

Event sequences are words

$$\cdot\, f \cdot h \cdots f \cdot \cdot\, h \cdot \cdot$$

Properties can be decided with automata

Every “Fire” is followed by a “Handle”



Observable Properties

Problem statement

- "Every 'Fire' is followed by a 'Handle'" cannot be decided through observation:

$$\cdot\cdot\cdot\cdot\cdot\cdot f \cdot\cdot\cdot\cdot\cdot\cdot h \cdot\cdot\cdot\cdot \qquad \surd$$

$$\cdot\cdot\cdot\cdot\cdot\cdot f \cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot \qquad ?$$

Safety

- "no error occurs" ● ● ● ● ● ● ● ● ● ● ● · · ·
- observable

Liveness

- "something happens eventually" ● ● ● ● ● ● ● ● ● ● ● · · ·
- not observable

Slicing

Slicing

Splits the event sequence in subsequences

- So activities can be examined separately
- So various simple monitors can be used

Monitors are based on

- Regular languages
- Linear temporal logic (LTL)
- Statistical methods (for example, jitter analysis)
- ...

Usually Events Contain Data

Java Collections

While an iterator is in use, the underlying data structure may not be modified.

$c(D, I)$	iterator I created for data structure D
$m(D)$	data structure D modified
$s(I)$	iterator I steps to next element

Example

$c(l_1, i_1)\ \ c(l_2, i_2)\ \ m(l_1)\ \ c(l_1, i_3)\ \ s(i_1)\ \ s(i_2)$

$c(l_1, i_1)$		$m(l_1)$		$s(i_1)$		$D \mapsto l_1, I \mapsto i_1$
		$m(l_1)$	$c(l_1, i_3)$			$D \mapsto l_1, I \mapsto i_3$
	$c(l_2, i_2)$				$s(i_2)$	$D \mapsto l_2, I \mapsto i_2$

JavaMOP

Base algorithm A [Roşu and Chen, 2007]

- Separates event sequence based on contained data
- Allocates a suitable monitor for each slice
- Optimised variants exist (B, C, C+, D)

Extension [Ballarin 2014]

- Combination of events with unrelated data into a slice
- Previously believed to be an inherent limitation
- Essential for applying slicing to scheduling problems

Events Related, but not through Data

Priority-based scheduling

When activities X and Y are fired, the higher-priority activity shall be handled first.

$f(X), f(Y)$	activity X or Y fired
$h(X), h(Y)$	activity X or Y handled

Example

$f(1) \quad f(3) \quad f(2) \quad h(3) \quad h(2) \quad h(1)$

Priority of $3 > 2 > 1$

$f(1)$		$f(2)$		$h(2)$	$h(1)$	$X \mapsto 2, Y \mapsto 1$
$f(1)$	$f(3)$		$h(3)$		$h(1)$	$X \mapsto 3, Y \mapsto 1$
	$f(3)$	$f(2)$	$h(3)$	$h(2)$		$X \mapsto 3, Y \mapsto 2$

JavaMOP

Base algorithm A [Roșu and Chen, 2007]

- Separates event sequence based on contained data
- Allocates a suitable monitor for each slice
- Optimised variants exist (B, C, C+, D)

Extension [Ballarin 2014]

- Combination of events with unrelated data into a slice
- Previously believed to be an inherent limitation
- Essential for applying slicing to scheduling problems

Implementation

Monitor for JamaicaVM

- Implements extended version of Algorithm A
- Monitors based on finite automata
- Several defects in JamaicaVM's scheduler for multicore systems identified

JavaMOP / RV

- Comprehensive library of safety properties
- Runtime overhead for the DaCapo 9.12 benchmark suite: JavaMOP 360%, RV 140%
- `http://fsl.cs.illinois.edu/index.php/JavaMOP4`

Summary

Runtime verification

- Systematic tests of actual systems
- Based on formal methods
- Particularly suitable for safety properties

Slicing

- Computes subsequences of event traces based on data
- Can be extended to problems where slices are not evident from the data — for example, scheduling

www.aicas.com