WCET Analysis of Java Bytecode Featuring Common Execution Environments

Kasper Søe Luckow
Joint work with
Christian Frost, Casper Svenning Jensen, and Bent Thomsen

Department of Computer Science,
Aalborg University

JTRES, September 2011

Motivation

- The Worst Case Execution Time (WCET) forms an integral component in schedulability analysis and, thus, in determining temporal correctness
- For hard real-time systems, the estimated WCETs must be *safe*, that is, must be at least as high as the actual WCET
- Safe WCET (and schedulability) analysis is currently possible on hardware implementations of the JVM using WCA[1] for JOP (and SARTS[2] for schedulability analysis)
- Portable WCET analysis has been proposed by XRTJ, but relies on a measurement-based technique for low-level WCET analysis
- We want to extend WCET analysis of hard real-time Java systems to execution environments with software implementations of the JVM and common embedded processors using a static approach

[1] http://www.jopdesign.com
[2] http://sarts.boegholm.dk/

TetaJ at a Glance

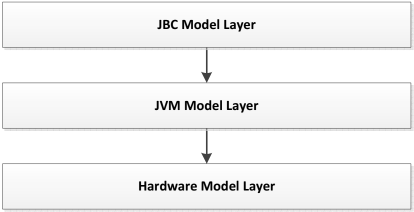
- Capable of conducting WCET analysis of Java bytecode taking into account a software implementation of the JVM, and common embedded hardware

Design goals

- Flexibility
  - To address Java's portability goal
- Safe and precise WCET estimates
  - Safety: necessitated by hard real-time systems
  - Precision: to make fewest computational resources suffice while still being temporally correct
- Applicable for iterative development
  - Analyses on method level
  - Analysis time and memory consumption are reasonably low

WCET Estimation Technique

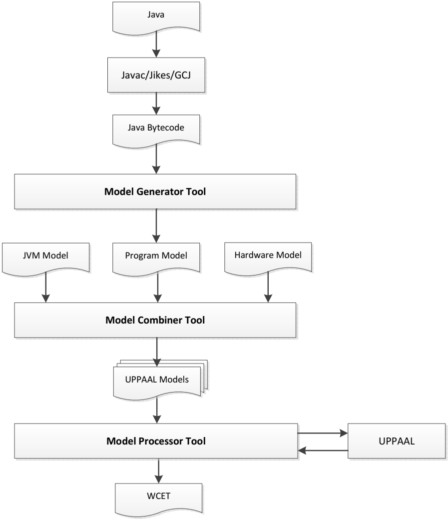
- TetaJ employs a static analysis approach for WCET estimation
- The program analysis problem of determining WCET is viewed as a model checking problem
- Java bytecode program, JVM, and hardware are modelled as a Network of Timed Automata (NTA) (UPPAAL[3])
- Program and execution environment are viewed as three independent layers of models
  - Interact through pre-defined interfaces
  - Achieves high flexibility



[3] http://www.uppaal.com

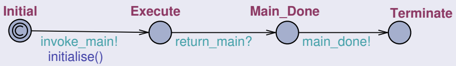
Architecture of TetaJ

- TetaJ Control Flow Graphs (TCFGs)
  - Achieves reusability
- Extendible support:
  - AVR $\rightarrow$ TCFG
  - Java Bytecode $\rightarrow$ TCFG
- CFG analyses for e.g.
  - Loop detection
  - Condition optimisation
- Transform TCFG to NTA
- Combine models into one
- Interact with UPPAAL to obtain WCET

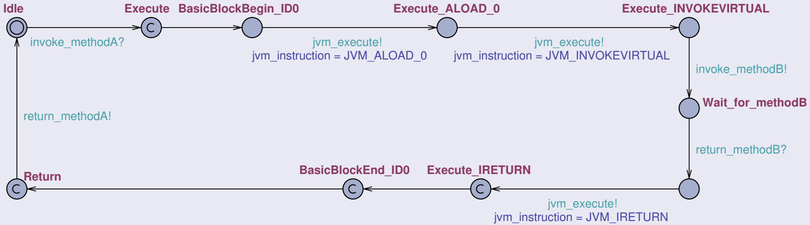


UPPAAL Models

Initialisation Model

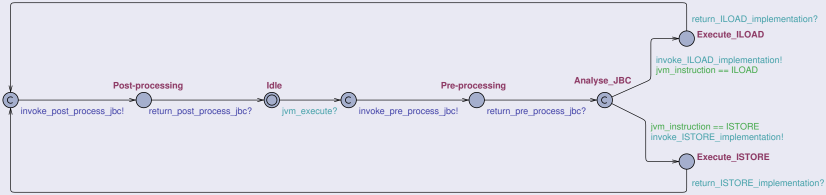


Program Model



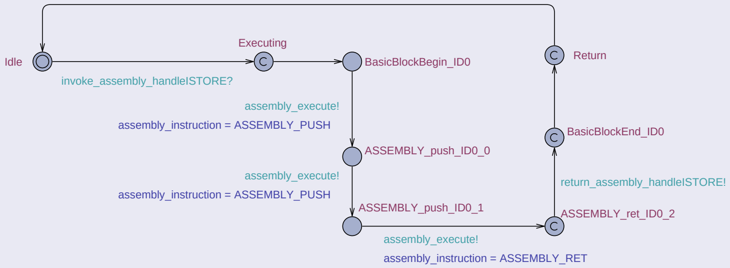
UPPAAL Models Cont'd

JVM Model (excerpt)

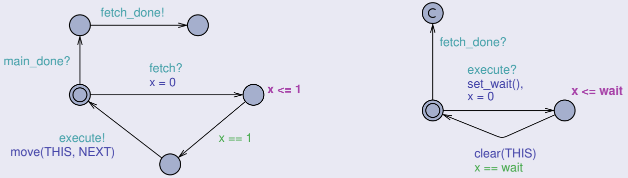


UPPAAL Models Cont'd

Java Bytecode Implementation



Hardware Models (From METAMOC)



Evaluation of Optimisations

Optimisation	Analysis time	States explored	Memory usage
No optimisations	14h 51m 17s	41854143	3,905 MB
Only state space reduction	13h 33m 21s	41854143	2,426 MB
Only condition optimisation	1m 16s	53732	294 MB
Only template reduction	4h 46m 41s	41854143	3,851 MB
All optimisations	19s	57553	144 MB

- Based on a trivial program containing, among others, a loop and variable assignments
- Optimisations decrease resources needed substantially
  - Analysis time decreases from 15 hours to 19 seconds
  - Memory consumption decreases from 3.9GB to 144MB
- Template reduction and condition optimisations are the prime contributors to this decrease

Evaluation of Safety and Precision

Algorithm	Meas. WCET	TetaJ WCET	Pessimism
Iterative Fibonacci	46,642	46,933	0.6%
Factorial	39,726	40,939	3.1%
Reverse Ordering	64,436	81,919	27.1%
Bubble Sort	907,103	2,270,401	150.3%
Binary Search	54,430	99,301	82.4%
Insertion Sort	849,353	3,740,769	440.4%

- Safety and precision are evaluated by using a measurement-based method
  - Formally not safe. Hence, this evaluation only provides indications
- Generally: TetaJ WCET $>$ Meas. WCET $\rightarrow$ indicates safety
- Precision: as low as 0.6%
- Imprecise results, e.g. Bubble Sort (150%), are attributed lack of annotation possibilities for stating dependencies among loop bounds

Mine Pump

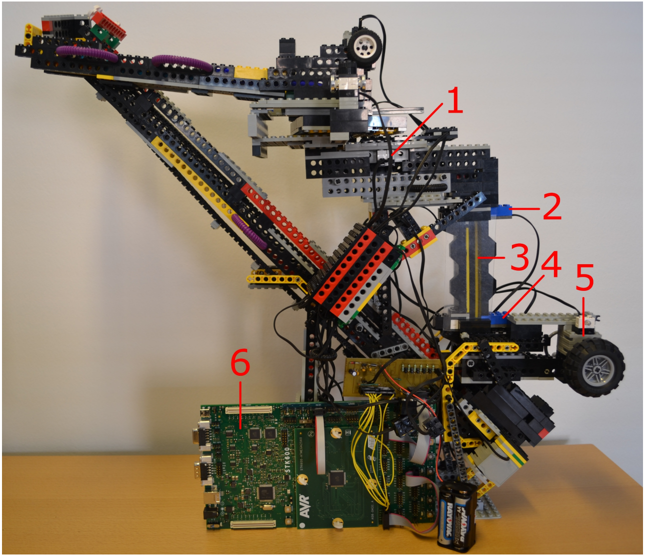
- Classic text-book example of a real-time system
- A water pump is responsible for removing excess water to avoid endangering the lives of the mine workers

Requirements

- Pump starts whenever the water level reaches the high level marker
- Pump stops whenever the water level reaches the low level marker
- Pump must not run when the methane concentration is too high
- If the methane level is not critical, the mine pump must never be flooded

Task	Period/Deadline
Methane	$56ms$
Water	$40ms$

The LEGO Mine Pump



The LEGO Mine Pump Cont'd

- Implemented in Java
- 429 lines of code
- 18 classes
- McCabe cyclomatic complexity of 6
- RTSJ, SCJ, PJ etc. have not been used

Execution Platform

- Based on a modified version of the Hardware near Virtual Machine (HVM)[4] and the Atmel AVR ATmega2560 processor
- Representative execution platform for an embedded system using Java Bytecode

HVM

- Emphasises portability (currently has support for ATmega2560, CR16C, and x86)
- Supports systems with as low as 256 kB of flash, and 8 kB of RAM
- Employs iterative interpretation of Java Bytecode to machine code
- Originally not amenable to static WCET analysis

Atmel AVR ATmega2560

- Deterministic behaviour
- No caching, nor branch prediction
- Features a simple two-stage pipepline

[4] http://www.icelab.dk

The Modified HVM

- Modifications of the HVM comprise e.g.:
  - Constant time analyse stage
  - Eliminating recursive solutions
  - Constant time type compatibility check
- The NTA of the modified HVM can be automatically constructed by the provision of its binary
- A timed automaton captures the conceptual model of iterative interpretation, that is, the *fetch*, *analyse*, and *execute* stages
- This automaton forms an NTA with the timed automata constructed from each of the Java Bytecode implementations

Applying TetaJ in the Case Study

Task	Analysis time	Memory usage	TetaJ WCET	Meas. WCET
Methane 1	1m 19s	140 MB	41,644	8,901
Methane 2	1m 16s	105 MB	68,436	29,449
Methane 3	29s	75 MB	12,552	3,896
Water	6m 40s	271 MB	70,712	32,421

- Again TetaJ WCET $>$ measured WCET
- Analysis times and memory consumptions are reasonably low
- A cyclic executive is constructed for scheduling the two tasks
    - Minor cycle is 8ms; major cycle is 280ms
    - A schedule can be constructed, thus we conclude that the system is schedulable (proof by construction)

Conclusion

- Shown that WCET analysis of Java Bytecode executed on common embedded hardware is feasible using model checking with UPPAAL
- From the case study, we have shown that TetaJ may be adopted in an iterative development method
  - Analysis time and resource consumption are reasonably low which can be attributed the optimisations
  - TetaJ provides analysis on method level by providing the fully qualified name of the method of interest
- We have indications that the WCET estimates obtained by TetaJ are safe
  - May therefore be appropriate for analysing hard real-time systems
- Precise WCETs can be obtained using TetaJ

Future Work

- Compare TetaJ with WCA
- Evaluate scalability of TetaJ
  - In general, apply TetaJ on more case studies
- Determine whether TetaJ still applies for systems that use e.g. the SCJ
- Merge TetaJ with SARTS