Supporting Development of Energy-Optimised Java Real-Time Systems using TetaSARTS

Luckow, Kasper Søe; Bøgholm, Thomas; Thomsen, Bent

Publication date:
2013

Document Version
Early version, also known as pre-print

Supporting Development of Energy-Optimised Real-Time Java Systems using TetaSARTS

AALBORG UNIVERSITY DENMARK

Kasper Søe Luckow, Thomas Bøgholm, and Bent Thomsen

Department of Computer Science, Aalborg University, Denmark

{luckow,boegholm,bt}@cs.aau.dk

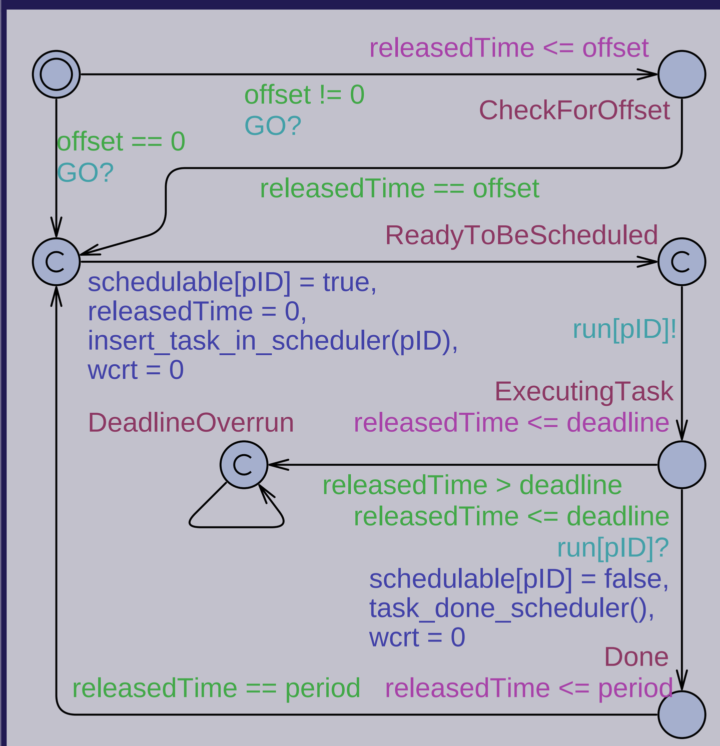
AALBORG UNIVERSITY DENMARK

Motivation

It is well-known, that the traditional Java run-time is unsuited for use in embedded real-time systems, which is attributed issues such as the lack of high-resolution real-time clocks and timers, insufficient thread semantics, and, most notably, memory management, which is traditionally handled by a garbage collector whose execution is highly unpredictable. However, with emerging standards such as the Real-Time Specification for Java (RTSJ) and the Safety Critical Java (SCJ) profile, these issues have been accounted for, thereby achieving a significant step towards use in embedded real-time systems development.

Having a suitable programming model as introduced by e.g. SCJ and RTSJ is not the only component in making Java a viable technological alternative and competitor to C in the embedded real-time systems domain. Equally important is the complementation of tools and analyses that support the development, and for verification purposes. For real-time systems, the latter comprises functional correctness, but also *temporal correctness*, which is the focus of this work.

Contributions



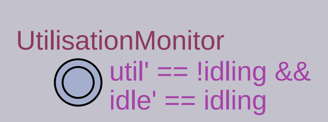


Fig. 1: Timed Automaton monitoring the state of the associated task. **Fig. 2:** Utilisation monitor.

We put forward the TetaSARTS tool whose capabilities include:

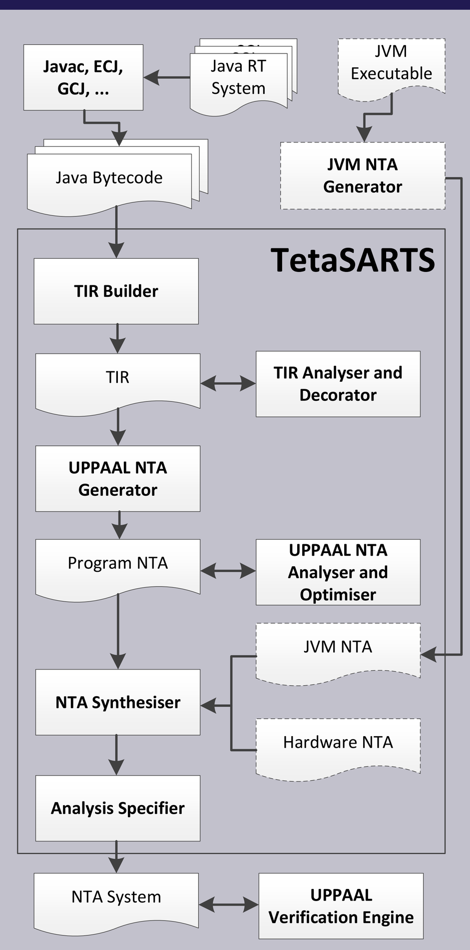
Schedulability Analysis The analysis is approached as a reachability, model checking problem by determining whether the Timed Automata model satisfies the specification *A*□ *not deadlock*. A deadlock can only be reached, if there exists a simulation reaching the `DeadlineOverrun` location for any of the real-time tasks (see Fig. 1).

Clock Frequency Adjustment The analysis facilitates adjustment of the clock frequency. Using a technique such as the bisection method, the appropriate clock frequency can be determined while still ensuring schedulability. Thus, potential energy reductions can be identified.

Worst-Case Response Time Analysis The Timed Automata model facilitates Worst-Case Response Time (WCRT) analysis to be conducted taking into account blocking time. Using the *sup-query* extension of the UPPAAL specification language, the maximum observed value of a clock variable can be found. Hence, e.g. for periodic tasks, WCRT analysis can be conducted using the specification $sup\{periodicThread.ExecutingTask\} : periodicThread.wcrt$ (see Fig. 1).

Processor Utilisation/Idle Time Analysis Adding the Timed Automaton shown in Fig. 2 allows monitoring the utilisation and idle time of the processor using the specification $sup : util, idle$.

TetaSARTS



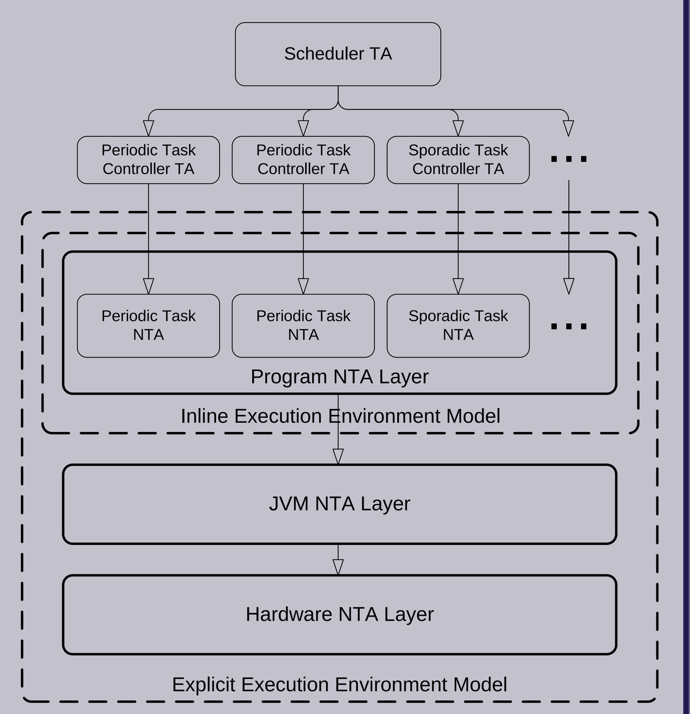


Fig. 3: TetaSARTS. **Fig. 4:** Resulting NTA.

TetaSARTS is a fully automated tool for conducting timing analyses of SCJ Java Bytecode programs taking into account the execution environment (see Fig. 3 for an overview). It accommodates hardware implementations of the JVM, and traditional execution environments with a software implementation of the JVM including the hosting hardware. It supports both periodic and sporadic real-time tasks, and from these, it generates Timed Automata simulating their control flow. The structure of the model and the communication between the Timed Automata using synchronisation actions is shown in Fig. 4. The analyses are performed using the UPPAAL model checker.

Results

We base the results on analysing two representative examples of real-time systems written in Java: the Real-Time Sorting Machine (RTSM) and the Minepump control system. The systems have been evaluated on the Java Optimized Processor (JOP), and an execution environment consisting of the Hardware near Virtual Machine (HVM) on an AVR ATmega2560 microcontroller. The results have been obtained on a system with an Intel Core i7-2620M @ 2.70GHz and 8 GB of memory. The results are shown in Table 1, 2, and 3.

Execution Env.	Clock Freq.	Schedulable
HVM + AVR	10 MHz	✓
HVM + AVR	5 MHz	×
JOP	2 MHz	✓
JOP	1 MHz	×

Table 1: Schedulability of the Minepump when varying the clock frequency.

System	Clock Freq.	Idle	Utilisation
RTSM	100 MHz	4.0 ms	48.5 μs (1.2%)
RTSM	60 MHz	4.0 ms	80.8 μs (2.0%)
Minep.	100 MHz	2.0 ms	25.9 μs (1.3%)
Minep.	10 MHz	11.8 ms	259.0 μs (2.1%)

Table 2: Processor idle and processor utilisation times when the system is schedulable on the JOP.

RT Task	WCRT	Veri. Time
Periodic Task 1	62.3μs	60s
Periodic Task 2	18.4μs	10s
Sporadic Task 1	4.5μs	10s
Sporadic Task 2	4.5μs	25s

Table 3: WCRT of the real-time tasks of the RTSM.

Timed Automata

Timed Automata are generated from Java Bytecode by identifying the event handlers of the real-time tasks such as the one in Listing 1.

```
public class MethaneCtrl extends PeriodicEventHandler {
    public void handleAsyncEvent() {
        if (this.methaneSensor.isCritMethaneLvlReached())
            this.waterpumpActuator.run();
        else
            this.waterpumpActuator.stop();
    }
}
```

Listing 1: SCJ event handler periodically firing *handleAsyncEvent()*.

From Java Bytecode, the control flow graph is reconstructed, producing the intermediate representation, *TIR*, on which various analyses and optimisations are applied. *TIR* is then translated to Timed Automata (see Fig. 5).

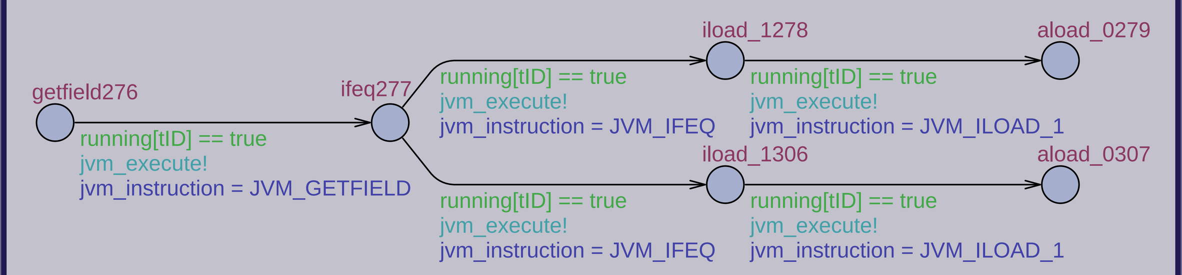


Fig. 5: Excerpt of the generated Timed Automaton.

Here, locations represent Java Bytecodes and the firing of the outgoing edges initiates simulation of that Java Bytecode by consulting the respective Timed Automaton in the JVM NTA Layer (see Fig. 6).

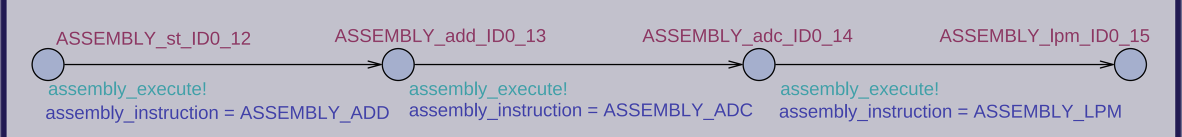


Fig. 6: Excerpt of simulating a Java Bytecode.

In turn, the Java Bytecode is simulated using the underlying Hardware NTA Layer (see Fig. 7).

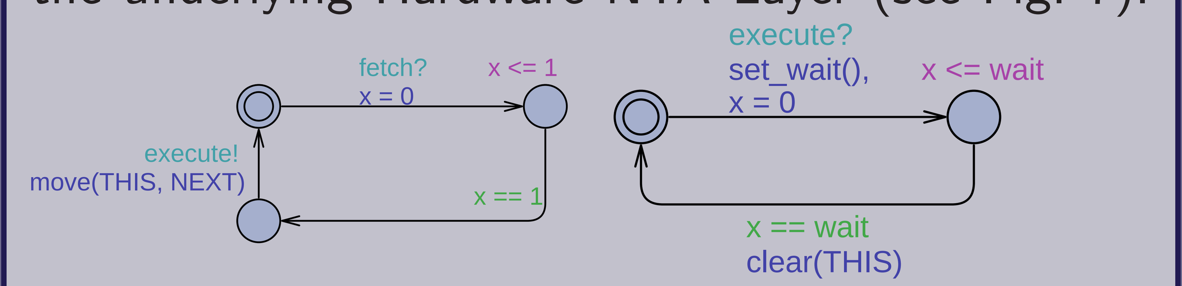


Fig. 7: Pipeline fetch and execute stage, respectively.

Future Work

Support other Analyses The Timed Automata model can be used for analysing other interesting properties e.g. worst case blocking time.

Statistical Model Checking[2] Can be used for analysing systems for which probabilistic guarantees of temporal correctness are sufficient. The technique can be used to avoid exhaustive exploration of the state-space of the model.

Schedulability Abstractions[1] Decorate Java interfaces with abstract, behavioural descriptions and generate Timed Automata accordingly.

Case Studies We want to further evaluate the applicability of TetaSARTS by using more complex real-time systems, and possibly other variations of the execution environment.

Download TetaSARTS

`http://people.cs.aau.dk/~luckow/tetasarts/`

References

[1] T. Bøgholm, B. Thomsen, K. Larsen, and A. Mycroft. Schedulability analysis abstractions for safety critical java. 2012.

[2] A. David, K. G. Larsen, A. Legay, M. Mikučionis, D. B. Poulsen, J. Van Vliet, and Z. Wang. Statistical model checking for networks of priced timed automata. 2011.