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

Link to publication from Aalborg University

Citation for published version (APA):
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
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.

TetaSARTS
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 \textsc{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.

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