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

Kasper Søe Luckow, Thomas Bøgholm, and Bent Thomsen
Department of Computer Science, Aalborg University, Denmark
{luckow,boegholm,bt}@cs.aau.dk

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 i7-2600M @ 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


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 UPPAAL model checker.

Results

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.

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

<table>
<thead>
<tr>
<th>System</th>
<th>Clock Freq</th>
<th>Idle Utilisation</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTSM 100 MHz</td>
<td>4.0 ms</td>
<td>48.5 μs (1.2%)</td>
</tr>
<tr>
<td>RTSM 60 MHz</td>
<td>4.0 ms</td>
<td>80.8 μs (2.0%)</td>
</tr>
<tr>
<td>Minip. 100 MHz</td>
<td>2.0 ms</td>
<td>25.9 μs (1.3%)</td>
</tr>
<tr>
<td>Minip. 10 MHz</td>
<td>11.8 ms</td>
<td>259.0 μs (2.1%)</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Processor Task</th>
<th>WCRT</th>
<th>Veri. Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Periodic Task 1</td>
<td>62.3 μs</td>
<td>60 s</td>
</tr>
<tr>
<td>Periodic Task 2</td>
<td>18.4 μs</td>
<td>10 s</td>
</tr>
<tr>
<td>Sporadic Task 1</td>
<td>4.5 μs</td>
<td>10 s</td>
</tr>
<tr>
<td>Sporadic Task 2</td>
<td>4.5 μs</td>
<td>25 s</td>
</tr>
</tbody>
</table>

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.

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

From Java Bytecode, the control flow graph is re-constructed, 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.

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/