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

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

***Abstract*—This paper presents how the tool TetaSARTS can be used to support the development of embedded hard real-time systems written in Java using the emerging Safety Critical Java (SCJ) profile. TetaSARTS facilitates control-flow sensitive schedulability analysis of a set of real-time tasks, and features a pluggable platform specification allowing analysis of systems including the hosting execution environment. This is achieved by approaching the analysis as a model checking problem by modelling the system using the Timed Automata formalism of the model checking tool UPPAAL. The resulting Timed Automata model facilitates easy adjustment of a wide variety of parameters that may be of interest such as processor frequency.**

This paper demonstrates that TetaSARTS can be used for tuning processor frequency, for conducting control-flow sensitive Worst Case Response Time analysis, and for conducting processor utilisation and idle time analysis.

I. INTRODUCTION

It is well-known, that the traditional Java run-time is unsuited for use in embedded real-time systems which is attributed issues related to 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) [6] and the Safety Critical Java (SCJ) [12] profile these issues have been accounted for thereby achieving a significant step towards use in embedded real-time systems development.

However, having a programming model as introduced by e.g. SCJ is not the only component in making Java a viable technology and competitor to C in the embedded real-time systems market. Of equal importance is the complementation of tools and analyses for verifying that the system is correct. For real-time systems this entails functional correctness but also temporal correctness for which showing that the system is schedulable, that is, showing that no deadline violations can occur, is of utmost importance.

Other analyses are also important. Since embedded systems are usually characterised by being produced in large quantities, mitigating the unit price is also an imperative. It is hence desirable showing that the system is schedulable on a platform containing fewest possible resources. Using a similar rationale, it is also desirable to reduce the running costs of the system after deployment. This can partly be achieved by limiting the energy consumption which in turn is partly a result of running the system with the lowest possible processor clock frequency while still ensuring that the system is schedulable.

In this paper, we present how TetaSARTS[1] can be used for conducting the presented analyses. The tool facilitates control-flow sensitive schedulability analysis of a set of SCJ real-time tasks analysed on a pluggable platform model that allows taking into account an exact, control-flow sensitive representation of the underlying execution environment, which in the case of Java Bytecode systems usually comprises a Java Virtual Machine (JVM) such as the Hardware near Virtual Machine (HVM) [13] running on embedded hardware such as Atmel's AVR range of microcontrollers. It does, however, also accommodate hardware implementations of the JVM such as the Java Optimized Processor (JOP) [16]. Moreover, it also allows different schedulers to be used and many parameters related to the execution of the system such as processor clock frequency of the hardware, are adjustable.

The rest of the paper is organised as follows; in Section II, we present related work, followed by an overview of the TetaSARTS tool in Section III. Afterwards, we present the capabilities of TetaSARTS and the analyses it supports in Section IV. In Section V, we present initial results of using the presented usages on two representative examples of real-time systems. Finally, in Section VI, we conclude on the results and make pointers to future development.

II. RELATED WORK

TetaSARTS is inspired by tools for timing analysis including TetaJ [11], METAMOC [8], SARTS [4], TIMES [1], and UPPAAL [2]. TetaJ is a Worst Case Execution Time (WCET) analysis tool for Java Bytecode systems compiled from SCJ programs and employs a model-based approach for analysis inspired by METAMOC which analyses C programs. In TetaJ, the Java Bytecode system, the JVM implementation, and the hardware are modelled as a Network of Timed Automata (NTA) amenable to model checking using UPPAAL. SARTS is a schedulability analysis tool employing a model-based approach inspired by the model-based ideas of TIMES. While TIMES relies on a static WCET component and is based on abstract descriptions of the behavior of the system, SARTS simulates a control-flow sensitive execution of the Java Bytecode system. It, however, assumes that the Java Bytecode execution times are fixed and that the execution environment is based on the JOP.

[1]TetaSARTS is available at http://people.cs.aau.dk/~luckow/tetasarts/

Harnessing the model used for schedulability analysis for other purposes, is to some extent inspired by the work of [15] and [10]. In this work, schedulability analysis is performed by constructing an NTA which simulates the behavior of the real-time tasks in the system. This simulation also allows for determining the Worst Case Response Time (WCRT) of tasks and processor utilisation and idle time. The models, however, are not directly generated from program source and real-time task parameters such as WCET and the behavior of the real-time tasks are manually encoded in the NTA.

III. TetaSARTS

TetaSARTS distinguishes itself from existing tools by incorporating a control-flow sensitive notion of the execution environment hosting the real-time system. In its current form, AVR and ARM hardware platforms are supported together with the HVM and JOP JVM implementations. Furthermore, it supports SCJ and real-time tasks with periodic or sporadic release patterns and also accounts for blocking as introduced by synchronised methods in Java.

It adopts model checking for schedulability analysis; the Java Bytecode system, and the JVM implementation are transformed to an NTA and combined with an NTA model simulating the hardware. This transformation is automatic, thereby ensuring that a tight correspondence is kept between the actual system and the model used for analysis. The transformation process draws many similarities with that of a traditional optimising compiler: the Java Bytecode system (or the JVM executable), is initially transformed into an intermediate representation (TIR), which is similar to a Control-Flow Graph structure. This forms the basis for various TA-independent analyses and transformations such as loop identification analysis. Afterwards, TIR is transformed to an NTA and analyses and optimisations are applied including *TA Inlining*, *Devirtualisation*, *JVM Specialisation*, and *Edge Aggregation*. These contribute to reductions in the size of the state and the state space. The details of these and their effect is documented in [14] which also contains a formalisation of the translation from Java Bytecode system to the NTA.

The architecture of the resulting NTA resembles the original architecture of a Java Bytecode system as depicted in Figure 1. TetaSARTS offers two representations of the execution environment; if the Java Bytecodes have statically fixed execution times as is the case on e.g. the JOP, the execution environment can be *inlined* thus reducing TA instantiations and communication overhead. In the other case where the execution times of the Java Bytecodes are dependent on the state of the JVM and hardware, the execution environment is *explicitly* represented.

The `Scheduler TA` simulates the adopted scheduling policy; currently FPS, EDF, and FIFO policies are supported but the support is extendible. This TA governs the execution of the periodic and sporadic tasks of the system each of which having a corresponding `Task Controller TA` that handles the execution of the task e.g. periodically releasing it (potentially after an offset), monitoring whether deadlines have been missed etc. In the `Program NTA`, TAs simulate a control-flow sensitive execution of each of the real-time tasks; for each method, there is a corresponding TA simulating its execution. Listing 1 shows a simple periodic task written in the SCJ profile and Figure 2 shows an excerpt of the corresponding TA demonstrating how the control-flow structure is captured for the conditional branch.

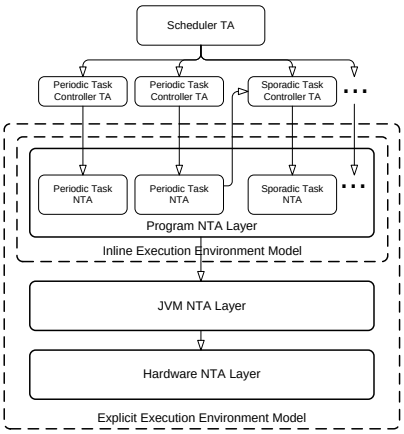

Fig. 1. The architecture of the resulting NTA.

```
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()*.

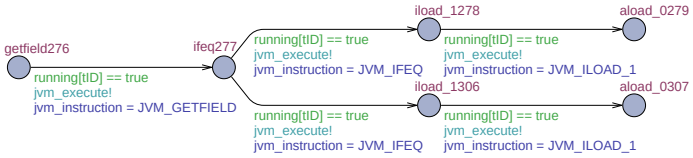

Fig. 2. Excerpt of a TA simulating the execution of Java Bytecodes.

Every firing of an edge in the TA, simulates an abstract execution of the particular instruction. In Figure 2, the execution environment is explicitly represented, thus the simulation of the Java Bytecode is transferred to the `JVM NTA` which receives on the *jvm_execute* synchronisation channel and consults the variable *jvm_instruction* which contains the Java Bytecode to simulate. The structure of the `JVM NTA` is similar to the `Program NTA`; each Java Bytecode simulation is enclosed in a separate TA simulating the execution of the machine instructions by consulting the `Hardware NTA`. TetaSARTS is capable of automatically constructing the `JVM NTA` provided the JVM executable and provided that the JVM has a certain structure. The `Hardware NTA`'s are reused from the METAMOC project [8].

IV. USAGE

A. Schedulability Analysis

The primary functionality of TetaSARTS is schedulability analysis of Java Bytecode real-time systems by extracting the real-time requirements of the real-time tasks from the source code, that is, period, offset, deadline, and release pattern. By configuring TetaSARTS in terms of specifying the execution environment constituents and scheduling policy, schedulability analysis is performed by simulating an abstract execution of the Java Bytecode real-time system on the hosting execution environment. The schedulability analysis is expressed using the UPPAAL query specification $\Box$ *not deadlock*, meaning that in all states, the system will never reach a deadlock state. This state can only be reached if a deadline is missed. In Figure 3, the `Task Controller TA` for a periodic task is shown. The

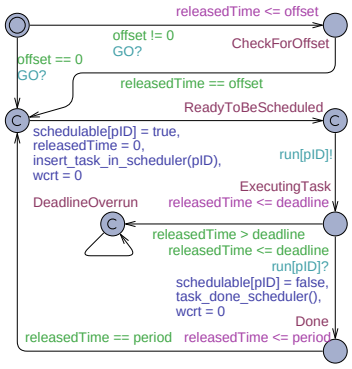

Fig. 3. `Task Controller TA` associated with a periodic task.

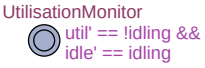

Fig. 4. `Utilisation TA`.

deadlock state occurs when the TA enters the *DeadlineOverrun* location which is only the case when the edge with the guard $releasedTime > deadline$ is true.

B. Processor Utilisation Analysis

TetaSARTS can also be used for determining worst case processor utilisation and idle time. When this option is enabled, TetaSARTS generates a new TA, shown in Figure 4, and adds two new clock variables; *idle* and *util* which are used as stop-watches [9]. When the scheduler has set a real-time task for execution, it also sets the *idling* variable to false which makes the *idle* clock stop progressing and makes the *util* progress. The opposite holds when the scheduler is idling, that is, when no task is eligible for execution. Determining the worst case processor utilisation and processor idle time, is hence a matter of determining the maximum values of the newly introduced clocks. This can be done by using the UPPAAL sup-queries $sup : util, idle$, which explore the entire state space, and returns the *supremum*, that is, the maximum observed value of the specified variables/clocks.

C. Worst Case Response Time Analysis

Traditional methods for WCRT analysis, such as [7], are usually based on a coarse, control-flow insensitive process model and do not include detailed information about the underlying hardware because these are included statically in the WCET component of the analysis. Further, they do not account for the release patterns of sporadic tasks which are regarded as periodic with period set to the minimum inter-arrival time. TetaSARTS, however, accommodates these shortcomings and is capable of conducting a control-flow sensitive WCRT analysis that also includes blocking as introduced by the *synchronized* keyword in Java. When this option is enabled, a new clock variable, *wcrt*, is introduced and is reset on the edge going to the *ReadyToBeScheduled* location and on the edge with destination in the *Done* location (See Figure 3). The WCRT of the task associated with the `Task Controller TA` can now be determined as the maximum observed value of the *wcrt* clock variable in the *ExecutingTask* location using the query $sup\{periodicThread.ExecutingTask\} : periodicThread.wcrt$. The same applies for sporadic tasks. This value will also account for blocking and cases where the task is pre-empted as governed by the scheduling policy.

D. CPU Clock Frequency Analysis

For many embedded systems, such as AVR and JOP, the CPU clock frequency is variable, thus, in case energy reductions are imperative, it is desirable to reduce this to a minimum while still guaranteeing that the system is schedulable. TetaSARTS currently offers an iterative process for determining the appropriate clock frequency; when generating the NTA, the clock frequency of the system used in the analysis can be specified. Using a method like the bisection method, the system can iteratively be analysed for schedulability by incrementing or decrementing the clock frequency until a desired precision is reached.

V. EVALUATION AND RESULTS

We demonstrate the usages of TetaSARTS using the textbook example of a Minepump [7], [3], [11] and the Real-Time Sorting Machine (RTSM) [4]. Both are representative of real-time systems written in Java. Since the SCJ specification is still a draft, the Minepump and the RTSM have been written in a variant of it. This, however, is only a syntactical matter and does not affect the validity of the results. A system containing an Intel Core i7-2620M @ 2.70GHz and 8 GB of memory has been used in the evaluation.

To demonstrate that TetaSARTS can be used for determining an appropriate CPU clock frequency, we have analysed the Minepump control system hosted by an execution environment consisting of the HVM running on an AVR ATmega2560 and on the JOP, respectively. The results are shown in Table I.

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

TABLE I
USING TETASARTS WITH VARIOUS EXECUTION ENVIRONMENTS.

For demonstrating that TetaSARTS can be used for processor utilisation and processor idle time analysis, we use an inline representation of the JOP execution environment. The results of the analysis are shown in Table II.

System	Clock Freq.	Proc. Util.	Proc. Idle
RTSM	100 MHz	48.5 μs	4.0 ms
RTSM	60 MHz	80.8 μs	4.0 ms
Minepump	100 MHz	25.9 μs	2.0 ms
Minepump	10 MHz	259 μs	11.8 ms

TABLE II
RESULTING PROCESSOR UTILISATION AND PROCESSOR IDLE TIMES.

Common to analysing the RTSM and the Minepump is that model checking times are only a few seconds. As expected, the processor is utilised more as the CPU clock frequency is reduced.

For demonstrating WCRT analysis, we use the RTSM system, an inline representation of the JOP, and the CPU clock frequency is set to 60 MHz. The results are shown in Table III.

RT Task	WCRT	Analysis 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 III
RESULTS OF WCRT ANALYSIS OF THE REAL-TIME TASKS OF THE RTSM.

As shown, conducting WCRT analysis can be done relatively quickly.

VI. CONCLUSION

In this paper, we have presented how the tool TetaSARTS can complement the development process of Java real-time embedded systems development. TetaSARTS is primarily targeted at control-flow sensitive schedulability analysis of Java Bytecode systems while taking into account the execution environment. While this analysis forms one of the cornerstones in guaranteeing temporal correctness of the system, TetaSARTS also allows for analysing other interesting parameters: processor utilisation and idle time, the schedulability of the system when adjusting the CPU clock frequency, and for Worst Case Response Time of the real-time tasks. This paper has demonstrated that TetaSARTS is applicable for these analyses using an evaluation featuring representative examples of real-time systems.

Supporting real-time systems development using TetaSARTS is an ongoing effort, and we envision a variety of extensions and improvements. Firstly, we want to examine the effect of using TetaSARTS on more case studies featuring real-life examples of systems with other, and potentially more complex, execution environments. This will entail the development of more hardware models and automating the process of constructing the `JVM NTA` from other JVM implementations. Secondly, we want to support other relevant analyses as well e.g. blocking time analysis, and memory related analyses such as stack height analysis and worst case heap consumption analysis. Finally, we also envision TetaSARTS to be extended with *Schedulability Abstractions* as defined in [5] where the interface of methods in a Java program is decorated with behavioural descriptions to capture requirements for individual methods. However, this requires an extension of the specification language to allow platform dependent timing constraints to be expressed.

REFERENCES

[1] T. Amnell, E. Fersman, L. Mokrushin, P. Pettersson, and W. Yi. Times - a tool for schedulability analysis and code generation of real-time systems. In *Formal Modeling and Analysis of Timed Systems*, volume 2791 of *Lecture Notes in Computer Science*. 2004.

[2] J. Bengtsson, K. Larsen, F. Larsson, P. Pettersson, and W. Yi. UPPAAL — a tool suite for automatic verification of real-time systems. Lecture Notes in Computer Science. 1996.

[3] T. Bøgholm, C. Frost, R. Hansen, C. Jensen, K. Luckow, A. Ravn, H. Søndergaard, and B. Thomsen. Towards harnessing theories through tool support for hard real-time java programming. *Innovations in Systems and Software Engineering*.

[4] T. Bøgholm, H. Kragh-Hansen, P. Olsen, B. Thomsen, and K. G. Larsen. Model-based schedulability analysis of safety critical hard real-time Java programs. In *JTRES '08: Proc. of the 6th int'l workshop on Java tech. for real-time and embedded systems*, JTRES '08, 2008.

[5] T. Bøgholm, B. Thomsen, K. Larsen, and A. Mycroft. Schedulability analysis abstractions for safety critical java. In *Proc. of ISORC'12*, 2012.

[6] G. Bollella and J. Gosling. The real-time specification for java. *Computer*, 33(6):47 –54, jun 2000.

[7] A. Burns and A. Wellings. *Real-Time Systems and Programming Languages: ADA 95, Real-Time Java, and Real-Time POSIX*. Addison-Wesley Educational Publishers Inc., 4th edition, 2009.

[8] A. E. Dalsgaard, M. C. Olesen, M. Toft, R. R. Hansen, and K. G. Larsen. METAMOC: Modular Execution Time Analysis using Model Checking. In *10th International Workshop on Worst-Case Execution Time Analysis (WCET 2010)*, 2010.

[9] A. David, J. Illum, K. Larsen, and A. Skou. *Model-Based Framework for Schedulability Analysis Using UPPAAL 4.1*. 2009.

[10] A. David, K. Larsen, A. Legay, and M. Mikučionis. Schedulability of herschel-planck revisited using statistical model checking. In *Leveraging Applications of Formal Methods, Verification and Validation. Applications and Case Studies*. 2012.

[11] C. Frost, C. S. Jensen, K. S. Luckow, and B. Thomsen. Wcet analysis of java bytecode featuring common execution environments. In *Proceedings of the 9th International Workshop on Java Technologies for Real-Time and Embedded Systems*, JTRES '11, 2011.

[12] JSR302. The java community process™program - jsrs: Java specification requests - detail jsr# 302, 2011. http://www.jcp.org/en/jsr/detail?id=302.

[13] S. Korsholm. Hvm lean java for small devices, 2011. www.icelab.dk.

[14] K. S. Luckow, T. Bøgholm, B. Thomsen, and K. G. Larsen. Flexible schedulability analysis of java bytecode real-time systems. 2013. Submitted to SAS 2013.

[15] M. Mikučionis, K. G. Larsen, J. I. Rasmussen, B. Nielsen, A. Skou, S. U. Palm, J. S. Pedersen, and P. Hougaard. Schedulability analysis using uppaal: Herschel-planck case study. In *Proc. of the 4th int'l conf. on Leveraging applications of formal methods, verification, and validation*, ISoLA'10, 2010.

[16] M. Schoeberl. JOP: A Java Optimized Processor. In *On the Move to Meaningful Internet Systems 2003: Workshop on Java Technologies for Real-Time and Embedded Systems (JTRES 2003)*, LNCS, Catania, Italy, 2003.