# Performance Validation on Multicore Mobile Devices

Thomas Hubbard, Raimondas Lencevicius, Edu Metz, Gopal Raghavan

Nokia Research Center, 5 Wayside Road, Burlington, MA 01803, USA
Thomas.Hubbard@nokia.com Raimondas.Lencevicius@nokia.com
Edu.Metz@nokia.com Gopal.Raghavan@nokia.com

**Abstract.** The validation of modern software systems on mobile devices needs to incorporate both functional and non-functional requirements. While some progress has been made in validating performance (including power consumption) on current mobile devices, future mobile devices will incorporate multiple processing units, more complex software and hardware that will raise additional challenges. This paper outlines ideas for future directions in performance validation on multicore devices, based on the current work in model-based validation, application state monitoring and performance assertions.

## 1 Motivation

Today personal communication devices are more than voice call terminals. Mobile phones serve as platforms for a variety of mobile applications including text and picture messaging as well as personal information management, including data synchronization with remote servers and desktop computers. Many mobile phones today are equipped with imaging devices and are capable of taking still images and video clips. The images may be sent over wireless networks to other phones or may be transferred to a remote server or a desktop computer for storage or forwarding. Mobile phones also have a number of local connectivity interfaces such as USB, Bluetooth, and WLAN that can be used for a variety of applications involving local data transfer, remote execution, and other types of interaction with surrounding computing resources. For example, a phone can serve as a wireless modem for a laptop computer over Bluetooth connecting it to a wide area network over circuit switched data call or GPRS (General Packet Radio Service) packet data connection.

The above description shows that mobile devices host complex software systems subject to numerous non-functional requirements. This paper concentrates on performance requirements. In particular we focus on two areas of performance: software response time and power consumption, although our methods may be also applicable to other performance areas, such as memory consumption.

Software response time and power consumption requirements are somewhat at odds with each other: decreasing response time by using more powerful hardware usually means increased power consumption, while decreasing power consumption may lead to slower response times. However, they are tightly connected, since they both depend on the hardware specifications and on the software execution.

Both of these requirements have been partially ignored in the generic computing environments subject to Moore's law and constantly connected to power outlets. In a mobile environment, both are very important. Mobile device processors lack the speed of their personal computer counterparts for cost and size reasons, while the mobile software is as complicated as the software running on desktop computers. This forces mobile application developers to spend a lot of time optimizing software performance. Mobile device power consumption is primarily important for two critical reasons: battery lifetime and heat dissipation. Extending battery lifetime supports other non-functional requirements such as usability and availability of the device since the device does not have to be connected to the power outlet and charged as often. Heat dissipation is a key requirement because if too much heat is dissipated in too short amount of time, the device may overheat. It then has to be shut down or it risks being destroyed.

While some approaches for performance validation in mobile devices have been proposed [9][11], additional research is needed in the future. An important issue in the near future is the upcoming use of multiple processors on a single mobile device. While some mobile devices already use two general CPUs and two DSPs, devices with even more processors are projected in the future. More and more complex software is developed for mobile devices, which requires high performance processors. However, such processors consume a lot of power. A single processor needed to satisfy the application power needs in the next couple years would consume about 800mW, which is a considerable amount of power from battery capacity and heat dissipation point of view. Replacing such a processor with multiple slower and more energy efficient processors becomes a realistic alternative. ARM has already released MPCore [1] multicore solution that supports such approach. Even in the PC industry Intel and AMD are moving towards multiprocessor solutions in their future processor roadmaps [6].

This move towards multicore systems brings new challenges to non-functional requirement validation. The rest of this paper describes these challenges and proposes possible adaptation and extension of current research techniques for multicore systems.

Section 2 outlines an existing method of validating power consumption on mobile devices. Section 3 then describes the challenges of applying current methods to complex devices in the future. Sections 4 and 5 propose two new methods to deal with complexities described: application state monitoring and performance assertions. We finish with related work and conclusions.

## 2 Energy Consumption Validation With Power Profiles

It may be possible to create performance validation methods for multicore mobile devices by extending current approaches [9][11]. Already now, performance validation has to take into account the multiple hardware devices controlled by the software. These hardware devices have a large influence on the functionality and performance of the system. For example, an audio player is a simple application that uses a number of devices: LCD, backlight, flash storage, RAM memory, CPU, DSP,

speakers, keypad, and headset. If the audio is fetched from the network then the wireless modem must be used as well. All of these devices affect response time, throughput and power consumed by the audio player. Validating performance for such a device may be not very different from validation for upcoming multicore devices. The remainder of this section reviews the energy validation approach we have proposed previously [9].

Our energy consumption validation approach consists of three parts: power requirements, a device power model, and system power measurements.

Using this approach, the device's software is modeled as a set of run-time functions that are executed, possibly in parallel. A set of hardware devices is associated with software functions. The efficient energy consumption requirement is that only hardware devices associated with active software functions should be active. In other words, any device not associated with the active software function should be inactive.

The device being validated is modeled as a collection of hierarchical state machines [2] that represent the power consuming devices in the system. Each device, or a component, is described as a state machine containing a set of power functions.

For example, let us consider the model of an audio subsystem. Figure 1 represents the state machine of an abstract audio subsystem component. It shows that the subsystem can be in the "Audio Idle" state or in the "Audio Standby" state. The "Audio Standby" state shows the nesting of parallel sub-states that describe microphone and earpiece activity. Looking at the parallel states, it is clear that the microphone and the earpiece can transition independently between active and idle states when the audio subsystem is in the "Audio Standby" state.

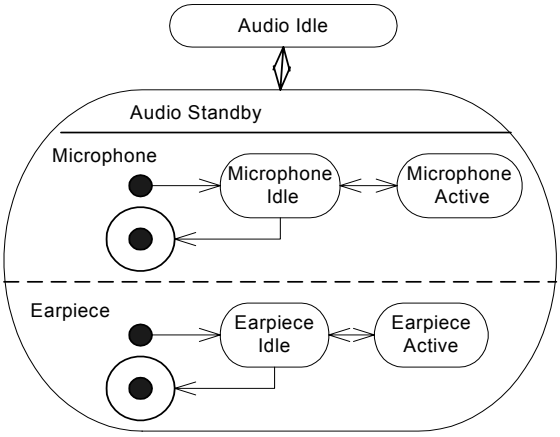


Figure 1. Audio subsystem state transition model

Every state and transition in the state machine has an associated power level function. In the simplest case, the power level function is a constant for a state and zero for a transition. The power consumption in a state may also depend on parameters. For example, the power consumption in a processor depends on executed

instructions, cache misses, memory accesses, etc. The power consumption in a radio antenna depends on the transmission frequency, signal strength, transmission protocol and other parameters. It is possible to use only constant power functions and to model any parameter dependencies by introducing additional states and transitions. However, such approach is cumbersome. For example, while a very detailed state transition diagram could potentially model the processor power consumption, such diagram would have enormous number of states and would be difficult to construct and understand. Non-constant power functions allow us to use higher-level system abstractions in state transition diagrams.

The power function of a composite state combines the power functions of the state's components. For example, the Audio Standby state's power function combines the constant standby power of the audio subsystem $PConstant_{AudioStandby}$ and the power functions of the microphone $P_{microphone}$ and the earpiece $P_{earpiece}$:

$P_{AudioStandby} = PConstant_{AudioStandby} \circ P_{microphone} \circ P_{earpiece}$

In simple cases, the composite function is a sum of the component functions.

Transitions between the power states in the model are controlled by global system events. These events are applied to all concurrent parts of the state machine so that when an event occurs, all parts perform the state transition triggered by the event. The overall power consumption of the entire system can then easily be determined by examining the power function of each state in the system.

In order to perform energy requirement validation the following information is needed: the model described above, the observation of the global system event triggers that generate state transitions, and the power measurements at the time of the trigger. The events are observed by using software traces. The power measurements are part of a power measurement test framework. When validation is being performed, the events observed from the trace are applied to the model in time sequence. At each point in time, the calculated power function of the model is compared to the measured power of the system. If there is any discrepancy between the actual and the modeled power levels, it means that the efficient energy consumption requirement is violated. Further analysis can indicate whether the system implementation is faulty or the system model is incorrect.

## 3 Challenges of Validating Complex Devices

Adjusting the approach described in section 2 for multicore devices is simple in principle. It only requires inclusion of the multicore hardware model into the hierarchical state machines. However, in practice this is not straightforward, because the hardware model and the relationship of the software to the hardware become much more complex in multicore devices. For example, resource conflicts have to be modeled when multiple processors access common bus, memory or peripherals. Timing relationships among multiple processors have to be modeled at very fine grain level, which leads to very fine grain (machine instruction level) modeling of software as well.

These issues are not unique to multicore devices. General complexity of hardware devices is increasing due to advanced features and power savings functionality. For

example, new processors allow multiple voltage and frequency settings based on the current processing demands of the device. Modeling of these processors will be more complex, since the transition times between voltage and frequency levels must be modeled as well as any lag time when moving from high frequency to low frequency states. WLAN adapters have different modes of operation, such as power save mode, constant aware mode, and so on. Each of these modes has different capabilities and power profiles. This means that hardware models become much more complex and building them may become a bottleneck in power validation process.

To illustrate these issues, let us consider an example of validating the power consumption for a web browser downloading five web pages from the network. The browser consists of 2 processes and uses the CPU, RAM, flash storage, LCD, backlight (at various brightness levels), keypad, and wireless modem over the entire test. The test takes 5 minutes to complete. During that 5-minute period on a typical mobile device with no other foreground application running, 42 threads belonging to 33 different processes are scheduled, not counting the browser. Also due to the asynchronous nature of wireless protocols and embedded devices many interrupts are scheduled causing the browser to suspend. On some executions of the tests, some thread not belonging to the browser may have previously activated the wireless modem. In such case, a browser thread does not have to do this again. In other test runs the browser may have to perform this power costly function thereby changing the power consumption. This results in a very complex system whose performance still needs to be validated.

Matching the software with the hardware is a non-trivial issue. It is easy to do for single core and multicore CPUs, since the operating system scheduler generally keeps track of which process or thread is active. For other resources reconciling the software using a resource with the resource requires a few levels of indirection. For example, under Linux it takes one call to enter system space and interact with the driver, and then the return call requires a callback to the kernel followed by a thread reschedule to the caller. In other, microkernel operating systems, for example, Symbian, this involves more levels of indirection and the application-resource mapping is difficult to track due to interactions with intermediate system servers.

Due to these considerations we cannot directly use the approach from section 2 for complex multicore devices. We need to find ways to adjust it so that it becomes applicable in practice. Sections 4 and 5 describe two possible directions.

## 4 Application State Based Validation

One way to resolve the issues raised in previous section would be to abstract the power consumption validation to the level of application states.

Generally, applications are specified as a set of features and requirements on those features. A logical state of an application can be viewed as a set of paths that combine to accomplish a goal and has a meaningful characteristic use of power and resources. This can be represented as a simple grammar in Bachus Naur form:

<application> ::= <state> { <state> }
<state> ::= <path> {<path>}
<path> ::= <instruction> {<instruction>}

The application level is too coarse for performance validation, while paths and instructions are too complex and granular to validate effectively. For this reason we propose to validate power and resource use at the application state level. An application can be instrumented to track application states and their transitions together with resources used in every state.

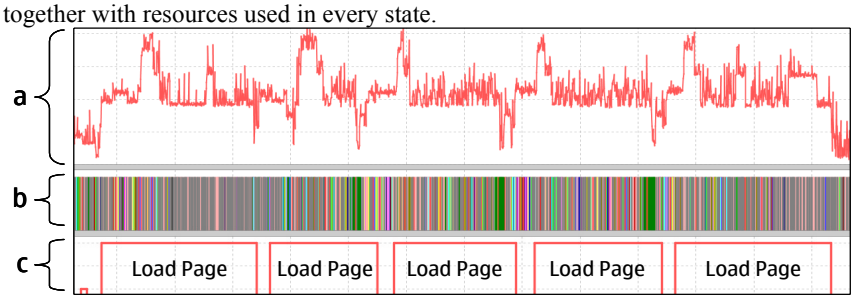


Figure 2: Application State Technique

Consider Figure 2 that shows the information obtained from an executed application, including the application state information and power measurements. In Figure 2, a) represents the power measurements taken from the browser test described in section 3, b) represents time slices of activity of the system (each color in the actual visualization represents a different thread in the system), and c) represents the different states that the browser enters during the verification process. In this example, the "Load Page" application state uses the CPU, RAM disk (for caching), Backlight (level 2), LCD, and wireless modem. Information about all the application states in the system along with software traces indicating which application is using which resource makes finding violations of power requirements more straightforward. The power consumed by the devices that an application is using in each state can be compared to the power consumption model abstracted to the application state level.

A possible drawback of this approach is that application states are more abstract than paths or instructions. Therefore the modeled power consumption may be approximate. The comparison of the model to the measurements would need to use some analytic approximate comparison techniques.

## 5 Assertion Based Validation

An alternative approach to the application-state based validation is to use performance assertions. Assertions have long been used to validate the functionality of software systems [4][12]. Because assertions became well-known and easy to use tool, researchers and practitioners tried to extend them for validation of non-functional requirements, such as performance [10][15]. Such assertions track the

software events corresponding to the ones specified in requirements and check the performance constraints. For example, a constraint "*Deleting a number of scheduled meetings in a calendar application should take less than 1 second plus 10 ms for each deleted meeting*", can be checked by a performance assertion:

```
int Calendar::calendarStartDeleteFunction (…)
{
        …
        pa_start(CALENDAR_DELETE);
        …
}

int Calendar::calendarFinishedDeletingFunction (…)
{
        …
        pa_end(CALENDAR_DELETE,
        assertion_interval(CALENDAR_DELETE)
         < 1000 + numberdeleted * 10);
        …
}
```

Here the pa_start event starts the assertion interval and pa_end event finishes it and checks the assertion. We have proposed a performance assertion framework for mobile devices that handles assertions in a multitasking environment (for example, Symbian OS [14] that uses client-server model inside the device) and resolves a number of other issues with previous performance assertion proposals [8].

Performance assertions allow programmers to specify performance constraints corresponding to the requirements directly in program code. For example, the following constraints may be specified and checked:

1. *During GPRS session lower layer packets arrive every 10 ms.*
2. *Screen redraw should take no more than 10% of the time needed to insert an appointment into a calendar application*
3. *Opening a scheduled meeting in a calendar application should take less than 1 second*
4. *Reading of a file should take at most 10ms multiplied by the number of blocks read and multiplied by the ratio of total and consecutive blocks in the file*

Assertions incorporate all the parts needed for validation: requirements are expressed as the assertion constraint and the device model is implicit in the constraint. The response time is communicated to the assertion checker by the assertion mechanism. Performance assertions avoid the complexity of profile-based validation by specifying and checking very specific requirements in a single spot. The complexity of building the model of the whole system is split into the more manageable complexity of using implicit partial models inside the assertion formulas. It remains to be seen whether such shift is justifiable in terms of overall complexity when numerous assertions are present.

Performance assertions can be adjusted to specify power constraints. Consider the example constraint we discussed above "*Deleting a number of scheduled meetings in a calendar application should take less than 1 second plus 10 ms for each deleted meeting*". It could be rewritten in terms of energy model: "*Deleting a number of scheduled meetings in a calendar application should take less than 100mW plus*

*100μW for each deleted meeting*". The assertion start and end events in the code would remain almost the same as before:

```
int Calendar::calendarStartDeleteFunction (…)
{
        …
        pa_start(CALENDAR_DELETE);
        …
}

int Calendar::calendarFinishedDeletingFunction (…)
{
        …
        pa_end(CALENDAR_DELETE,
        assertion_interval_power(CALENDAR_DELETE)
         < 100 + numberdeleted * 0.1);
        …
}
```

Instead of time measurements, power measurements would need to be provided to the assertion checker. Obtaining such measurements inside a device may be complicated. Additionally care should be taken to exclude energy spent in tasks unrelated to the one mentioned in a constraint. This is similar to tracking "process specific" time intervals, where time spent in other processes is excluded from the total time.

## 6 Related Work

Researchers from Duke University have proposed an energy consumption model for a Palm™ device [3], which they used for the Palm™ device simulation, but not for the validation. We formalized and extended their model using state machine diagrams and extended message sequence charts [9].

A research group at MIT has implemented JouleTrack [13] - a web based system for software energy profiling. JouleTrack simulates only the energy used by a processor in application execution. Such a system could provide a detailed model and power function for a processor.

Real-time system modeling is a large research field. Our power consumption model has some similarities with timed transition systems [5] and modecharts [7]. Neither timed transition systems nor modecharts were previously used to model the energy consumption of the real-time systems.

## 7 Conclusions

In this paper we presented the position that non-functional requirement validation is very important for future complex multicore mobile devices. We discussed the expected validation issues and our ideas for solving these issues based on the current work in model-based validation, application state monitoring, and performance

assertions. We believe that these approaches outline a feasible path towards solving the non-functional requirement validation issues in the future.

## References


[1] ARM MPCore, http://www.arm.com/products/CPUs/ARM11MPCoreMultiprocessor.html , 2005

[2] Booch, G., Rumbaugh, J., Jacobson, I.: *The Unified Modeling Language User Guide* (Addison-Wesley, 1999).

[3] Cignetti, T., Komarov, K., Ellis, C.: Energy Estimation Tools for the Palm™, *Proceedings of ACM MSWiM 2000: Modeling, Analysis and Simulation of Wireless and Mobile Systems*, August 2000.

[4] Floyd, R.W., Assigning Meanings to Programs, *Proceedings of the Symposium in Applied Mathematics*, Vol XIX, pp 19-32, American Mathematical Society, April 1967.

[5] Henzinger, T. A., Manna, Z., Pnueli, A.: Temporal proof methodologies for timed transition systems. *Information and Computation*, 112, pp. 273-337, 1994.

[6] Intel Platform 2015, http://www.intel.com/technology/computing/archinnov/platform2015/?iid=search& , 2005.

[7] Jahanian, F., Mok, A.: Modechart: A Specification Language for Real-Time Systems, *IEEE Transactions on Software Engineering*, vol. 20, no. 12, December 1994.

[8] Lencevicius, R., Metz, E.: Proposal of Performance Assertions for Mobile Devices, *Technical Report*, 2005.

[9] Lencevicius, R., Metz, E., Ran, A., Software Validation using Power Profiles, *20th IASTED International Conference on Applied Informatics (AI 2002)*, Feb 2002.

[10] Perl, S. E., Performance Assertion Checking, *Ph.D. Thesis*, MIT, 1992.

[11] Raghavan, G., Salomaki, A., Lencevicius, R., Model Based Estimation and Verification of Mobile Device Performance, *Proceedings of the International Conference on Embedded Software (EMSOFT 2004)*, September 2004.

[12] Rosenblum, D. S., Towards a method of programming with assertions, *Proceedings of the 14th international conference on Software Engineering*, Melbourne, Australia, pp. 92 – 104, 1992

[13] Sinha, A., Chandrakasan, A.: JouleTrack - A Web Based Tool for Software Energy Profiling, *Proceedings of the 38th Design Automation Conference*, Las Vegas, June 2001.

[14] Symbian OS, www.symbian.com, 2005.

[15] Vetter, J.; Worley, P.H., Asserting Performance Expectations, *Proceedings of the SC2002*, 2002.