Technical Report

Short Paper: Experiences on the Implementation of a Cooperative Embedded System Framework

Cláudio Maia

Luis Miguel Nogueira

Luis Miguel Pinho

HURRAY-TR-100801

Version:

Date: 08-19-2010

Short Paper: Experiences on the Implementation of a Cooperative Embedded System Framework

Cláudio Maia, Luis Miguel Nogueira, Luis Miguel Pinho

IPP-HURRAY!

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8340509

E-mail:

http://www.hurray.isep.ipp.pt

Abstract

As the complexity of embedded systems increases, multiple services have to compete for the limited resources of a single device. This situation is particularly critical for small embedded devices used in consumer electronics, telecommunication, industrial automation, or automotive systems. In fact, in order to satisfy a set of constraints related to weight, space, and energy consumption, these systems are typically built using microprocessors with lower processing power and limited resources.

The CooperatES framework has recently been proposed to tackle these challenges, allowing resource constrained devices to collectively execute services with their neighbours in order to fulfil the complex Quality of Service (QoS) constraints imposed by users and applications. In order to demonstrate the framework's concepts, a prototype is being implemented in the Android platform. This paper discusses key challenges that must be addressed and possible directions to incorporate the desired real-time behaviour in Android.

Short Paper: Experiences on the Implementation of a Cooperative Embedded System Framework

Cláudio Maia
CISTER Research Centre
School of Engineering of the
Polytechnic Institute of Porto
Porto, Portugal
crrm@isep.ipp.pt

Luís Nogueira
CISTER Research Centre
School of Engineering of the
Polytechnic Institute of Porto
Porto, Portugal
lmn@isep.ipp.pt

Luís Miguel Pinho
CISTER Research Centre
School of Engineering of the
Polytechnic Institute of Porto
Porto, Portugal
lmp@isep.ipp.pt

ABSTRACT

As the complexity of embedded systems increases, multiple services have to compete for the limited resources of a single device. This situation is particularly critical for small embedded devices used in consumer electronics, telecommunication, industrial automation, or automotive systems. In fact, in order to satisfy a set of constraints related to weight, space, and energy consumption, these systems are typically built using microprocessors with lower processing power and limited resources.

The CooperatES framework has recently been proposed to tackle these challenges, allowing resource constrained devices to collectively execute services with their neighbours in order to fulfil the complex Quality of Service (QoS) constraints imposed by users and applications. In order to demonstrate the framework's concepts, a prototype is being implemented in the Android platform. This paper discusses key challenges that must be addressed and possible directions to incorporate the desired real-time behaviour in Android.

Categories and Subject Descriptors

D.1.3 [**Programming Techniques**]: Concurrent Programming — Distributed programming; D.2.11 [**Software Engineering**]: Software architectures; J.7 [**Computers in Other Systems**]: Real-Time

Keywords

Distributed Real-Time Embedded Systems, Cooperative Computing, Android

1. INTRODUCTION

During the past years the embedded device industry has faced a huge growth and the tendency is to grow even more in the next years [10]. Following this tendency, new applications, functionalities and more diverse devices are becoming available to the general audience in a fast pace, therefore bringing new technological and scientific challenges.

One of these challenges is how to efficiently execute complex applications in these new embedded systems while meeting non-functional requirements, such as timeliness, robustness, dependability, performance, etc. In this context, a cooperative QoS-aware execution of resource intensive services among neighbour nodes seems a promising solution to address these increasingly complex demands on resources and desirable performance. The CooperatES (Cooperative Embedded Systems) framework [8] facilitates the cooperation among neighbours when a particular set of QoS constraints cannot be satisfyingly answered by a single node. Nodes dynamically group themselves into a new coalition, allocating resources to each new service and establishing an initial Service Level Agreement (SLA). The proposed SLA maximises the satisfaction of the QoS constraints associated with the new service and minimises the impact on the global QoS caused by the new service's arrival.

This paper builds on our previous work [7] and presents the motivation and the key challenges concerning the implementation of the CooperatES framework in the Android mobile platform [1]. The paper allows one to better understand the current limitations of the Android platform and identify possible directions for a better support of a cooperative QoS-aware execution of resource intensive applications in Android.

The remainder of this paper is organised as follows: Section II briefly presents the CooperatES framework. Section III points out the key challenges and focus in one of the possible extensions to incorporate real-time behaviour in Android. Finally, Section IV concludes the paper.

2. THE COOPERATES FRAMEWORK

Many embedded real-time systems evolved to open and dynamic environments where the characteristics of the computational load cannot be predicted in advance and resource needs are usually data dependent and may vary over time, as tasks enter and leave the system. Nevertheless, response to events still has to be provided within precise timing constraints in order to guarantee a desired level of performance.

In this context, a cooperative QoS-aware execution among neighbour nodes seems a promising solution. The goal of the CooperatES framework [8] is to satisfy multiple QoS dimensions imposed by users and applicatons by forming temporary coalitions of heterogeneous nodes whenever a particular set of QoS constraints cannot be satisfyingly answered

JTRES'10 August 19-21, 2010 Prague, Czech Republic

by a single node. Nodes may either cooperate because they cannot deal alone with the resource allocation demands imposed by users' QoS constraints or because they can reduce the associated cost of execution by working together.

Resource reservation is the basis for supporting QoS mechanisms. It is not possible to provide stable QoS characteristics to an application without some guarantees on the available amount of resources. While individual resource management is an important factor for an efficient QoS management, we believe that it is not sufficient for the ultimate end-users who experience the resulting QoS. In order to achieve the users' acceptance requirements and to satisfy the imposed constraints, the framework enables a QoS-aware resource management scheme that supports QoS negotiation, admission, and reservation mechanisms in an integrated and accessible way.

However, the CooperatES framework differs from other QoS-aware frameworks by considering, due to the increasing size and complexity of distributed embedded real-time systems, the needed trade-off between the level of optimisation and the usefulness of an optimal runtime system's adaptation behaviour. Nodes start by negotiating partial, acceptable service proposals that are latter refined if time permits, in contrast to a traditional QoS optimisation approach that either runs to completion or is not able to provide a useful solution. Thanks to the anytime nature of the proposed approach, it is possible to interrupt the optimisation process at any point in its execution and still be able to obtain a service solution and a measure of its quality, which is expected to improve as the run time of the algorithms increases.

Taking into consideration works made in the past such as [9, 4], either concerning the Linux kernel or Virtual Machine (VM) environments, there is the possibility of introducing temporal guarantees allied with QoS guarantees in each of the aforementioned layers, or even in both, in a way that a possible integration may be achieved, fulfilling the temporal constraints imposed by the applications.

In order to verify the feasibility of the framework's concepts, namely the cooperative execution allied with real-time characteristics as means to satisfy the QoS constraints imposed by applications, it was decided by the project's team to explore the capabilities of the Android platform. Among the positive features, one may find: (i) its software licence; (ii) the target devices in which Android can be run, i.e. mobile devices and devices based on x86 architecture [2], useful to prove the heterogeneous capabilities of the proposed framework; (iii) its Linux-based architecture; and finally, (iv) its VM environment. Also, the work described in [6] shows that the implementation of a resource management framework is possible in Android.

3. EXTENDING ANDROID FOR REAL-TIME EMBEDDED SYSTEMS

The Android's architecture is composed by five layers: Applications, Application Framework, Libraries, Android Runtime and finally the Linux kernel.

For the purpose of this paper it is just considered the interaction between the two bottommost layers - Android Runtime and the Linux Kernel. Regarding the Android Runtime, Android provides its own VM named Dalvik. Dalvik [3] was designed from scratch and it is specifically targeted for memory-constrained and CPU-constrained devices. It runs Java applications on top of it and unlike the standard Java VMs, which are stack-based, Dalvik is an infinite register-based machine. Dalvik uses its own byte-code format name Dalvik Executable (*.dex*), with the ability to perform several optimisations during *dex* generation when concerning the internal storage of types and constants by using principles such as minimal repetition; per-type pools; and implicit labelling.

In [7], we have discussed the suitability of Android for open embedded real-time systems, analysed its architecture internals, pointed out its current limitations, and proposed four possible directions to incorporate real-time behaviour into the Android platform, namely:

- Inclusion of a real-time VM, besides Dalvik, along with the inclusion of a real-time Operating System (OS);
- Extension of Dalvik with real-time features based on the Real-Time Specification for Java [5], as well as the inclusion of a real-time OS;
- Inclusion of a real-time OS in order to allow only native applications to have the desired real-time behaviour;
- Inclusion of a real-time hypervisor that parallelises the execution of Android and real-time applications. Both run as guests over the hypervisor which is responsible for handling the scheduling and memory management operations.

This paper focus the discussion on the first two directions. The first possible direction, depicted in Figure 1, replaces the standard Linux kernel with a real-time OS and includes a real-time Java VM together with Dalvik. It is then possible to have bounded memory management, real-time scheduling within the VM, and better synchronisation mechanisms. The real-time VM interacts directly with the OS's kernel, meaning that most of the operations provided by the real-time Java VM are limited to the integration between the VM's supported features and the supported OS's features.

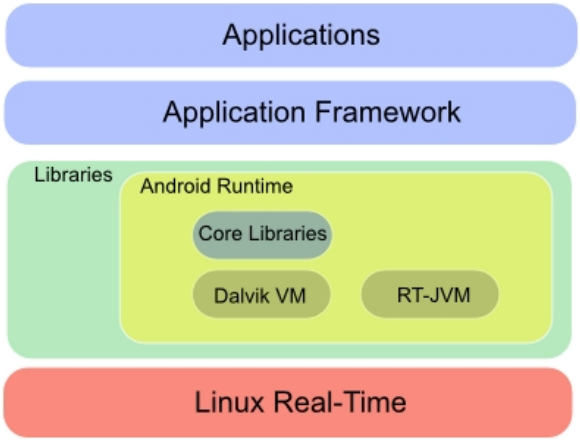

Figure 1: Android full Real-Time

The second possible direction, presented in Figure 2, introduces modifications both in the OS and VM environments. The major difference between this direction and the previous

one lies on the extension of Dalvik with real-time capabilities based on the Real-Time Specification for Java (RTSJ) [5].

By extending Dalvik with RTSJ features we are referring to the addition of the following API classes: *RealTimeThread*, *NoHeapRealTimeThread*, as well as the implementation of generic objects related to real-time scheduling and memory management, such as *Scheduler* and *MemoryAreas*. All of these objects will enable the implementation of real-time garbage collection algorithms, synchronisation algorithms and finally, asynchronous event handling algorithms.

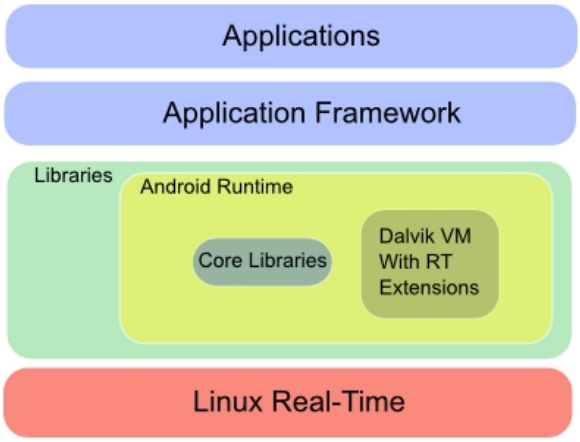

Figure 2: Android Extended

The acquired experiences with the directions proposed in [7] allows us to conclude that the first direction is the one that causes less impact in the system as a whole. It allows the possibility of having Dalvik VM serving the needs of any native Android application, while at the same time, the real-time VM can handle the specific requests made by any real-time application. Nevertheless, while the inclusion of this second VM brings the desired real-time behaviour to the Android platform, it also brings important challenges that should be considered. One may think of how the scheduling operations between both VMs are mapped into the OS, how the memory management operations will be managed in order to take advantage from the system's resources, and finally, how to handle thread synchronisation and asynchronous events in this dual VM environment.

Regarding scheduling, it must be assured by the OS that all the real-time tasks have higher priority than the normal Android tasks. This can be achieved by having a mechanism that maps each of the real-time tasks to a higher priority OS task. Then, the OS scheduler is responsible for assuring that these tasks are dispatched earlier than the remaining tasks. Thus, at a lower end limit a simple mapping mechanism must exist to perform this operation.

As for the memory management, one possible solution to consider would be to have a memory management abstraction layer that handles all the memory operations requested by both VMs, i.e. allocation and deallocation through the use of a smart garbage collector. The main benefit from this layer would come from the fact that it would be possible to have a single heap where all the objects would be managed and thus, the system's resources to deal with the dual VM environment would be optimised. The disadvantage lies in the way that Dalvik performs. Each Android application runs on its own Linux process with its own VM and garbage collector instances. Also, there is a part of the heap that is shared among all the processes. This *modus operandi* entails the need to, at least, integrate Dalvik with the abstraction layer and at the same time to modify its behaviour related to the per-process garbage collector instances.

Regarding thread synchronisation, as long as the real-time threads do not have the need to communicate with Dalvik threads, it is assured that this will not pose any kind of problems. However, if this communication is desired, a protection mechanism must be implemented in order to assure that a real-time thread will not block on a Dalvik thread and that priority inversion does not happen.

In terms of asynchronous events, a mapping mechanism must be sufficient to assure that the task that is waiting for the event will receive it in a bounded time interval. This mechanism must be implemented at the OS level in order to forward the events to the correct VM. Both VMs just need to implement the handlers for the events.

4. CONCLUSION

Android was chosen as a testbed for demonstrating the feasibility of the CooperatES framework, a QoS-aware framework that addresses the increasing demands on resources and performance by allowing services to be executed by temporary coalitions of nodes.

Android was designed to serve the mobile industry purposes which clearly has an impact on the way that the platform behaves. However, with some effort, as discussed in this paper, it is possible to incorporate the desired real-time behaviour and resource reservation mechanisms in an integrated and accessible way. These capabilities may suit specific applications by providing them the ability of taking advantage of temporal and resource guarantees, and therefore, to behave in a more predictable manner.

Acknowledgements

This work was supported by FCT through projects CooperatES (PTDC/ EIA/ 71624/ 2006) and RESCUE (PTDC/EIA/65862/2006), and by the European Commission through the ArtistDesign NoE (IST-FP7-214373).

5. REFERENCES

[1] Android. Home page, Jan. 2010.
[2] Android-x86. Android-x86 project, Jan. 2010.
[3] D. Bornstein. Dalvik vm internals, Mar. 2010.
[4] A. Corsaro. jrate home page, Mar. 2010.
[5] R.-T. S. for Java. Rtsj 1.0.2, Jan. 2010.
[6] R. Guerra, S. Schorr, and G. Fohler. Adaptive resource management for mobile terminals - the actors approach. In *Proceedings of 1st Workshop on Adaptive Resource Management (WARM10)*, Stockholm, Sweden, April 2010.
[7] C. Maia, L. Nogueira, and L. M. Pinho. Evaluating android os for embedded real-time systems. In *Proceedings of the 6th International Workshop on Operating Systems Platforms for Embedded Real-Time Applications*, Brussels, Belgium, July 2010.
[8] L. Nogueira and L. M. Pinho. Time-bounded distributed qos-aware service configuration in heterogeneous cooperative environments. *Journal of Parallel and Distributed Computing*, 69(6):491–507, June 2009.
[9] RTMACH. Linux/rk, Mar. 2010.
[10] M. Stanley. The mobile internet report, Jan. 2010.