Application Validation on RTDroid

Yin Yan
University at Buffalo, State University of New York
yinyan@buffalo.edu

Lukasz Ziarek
University at Buffalo, State University of New York
lziarek@buffalo.edu

ABSTRACT

Android is becoming used more frequently in domains that expect some real-time guarantees. To facilitate the adoption of Android for programming real-time systems, this work presents a first effort to apply real-time scheduling theories to a real-time extension on Android, RTDroid. We integrate real-time properties specified in RTDroid's application manifest with an existing real-time scheduling framework, Cheddar. We leverage Cheddar to perform schedulability analysis and feasibility tests, based on the properties of RTDroid application components specified in the application manifest. This paper details our integration process and reports our experience of validating real-time properties in a real-time application developed on RTDroid.

CCS CONCEPTS

• **Software and its engineering → Real-time schedulability**; *Real-time systems software*; *Empirical software validation*;

KEYWORDS

real-time, application validation, schedulability analysis

ACM Reference Format:

Yin Yan and Lukasz Ziarek. 2017. Application Validation on RTDroid. In *Proceedings of ESWeek Workshop on Declarative Embedded and Cyber-Physical Systems (DECPS'17).* ACM, New York, NY, USA, 8 pages. https://doi.org/10.475/123_4

1 INTRODUCTION

Mobile devices have evolved far beyond the stereotypical smartphone or tablet and are now being employed in traditional real-time embedded contexts. As one of the most popular mobile systems, Android has seen the most widespread deployment outside of the consumer electronics market. Its open source nature has prompted the ubiquitous adoption of Android in sensing, medical, command and control, robotics, and automotive applications [3–6]. For example, NASA's PhoneSat project [43] uses Android-equipped device as a low cost profile for a satellite control unit. The medical industry has envisioned Android as a potential end-user device for remote patient monitoring. Example applications include cardio monitors, glucose analyzers, patient fall detectors [1, 7, 15, 16, 22]. In academia, a number of sensing applications have been proposed and developed, including seismic sensing, acoustic processing, localization and *etc* [8, 17, 36].

The benefits of using Android in these scenarios are four fold: (1) Android provides a rich set of APIs and supports various connectivities through Wi-Fi, Bluetooth, 3G, and 4G; (2) It provides native support for a large set of sensors, including GPS, accelerometer, camera, and gyroscope; (3) It fosters an intuitive user interface design through a touch screen and gestures. Control and medical apps typically require these functionalities and their development can be streamlined as well as standardized through the Android APIs and libraries. (4) Android's play store offers countless applications that are useful for real-time applications, such activity detection, environmental sensing, network-adaptive data transfer, *etc*. The reuse of these existing functionalities on Android is one of the most appealing, and also challenging. factors to adapt Android for real-time as well. Unfortunately, since Android is originally designed as a mobile system, it is not surprising that Android does not provide any real-time guarantees.

Our previous work on RTDroid [45, 47, 48] introduced a new programming model that retained a familiar style of Android programming by extending Android's basic application components and its application manifest schema with real-time capabilities. RTDroid is built using a real-time capable JVM, Fiji VM [35], and an existing RTOS, either RTEMS or RT Linux. RTDroid leverages the runtime isolation features of the Fiji MultiVM [44, 49] for the execution of multiple applications. It also supports sophisticated sensing abilities with bounded sampling latency [46]. Although RTDroid provides a familiar programming model for building real-time applications on Android, currently it is up to the programmer to validate that the configuration of the application as well as the RTDroid components form which the application is constructed meet real-time guarantees. This work explores the possibility of adding a static validation mechanism to build process for RTDroid. This mechanism validates the real-time configuration for tasks initialization and shared resource allocation during an RTDroid's application bootstrap.

More specifically, this paper presents our first attempt to enables an off-line validation mechanism for RTDroid's applications. We utilize real-time properties declared in RTDroid's application manifest for scheduling analysis and feasibility tests. RTDroid's compiler converts its application manifest to a format that understood by a real-time scheduling framework, named Cheddar [39]. We leverage Cheddar's built-in scheduling simulation and feasibility tests to validate whether an application is configured correctly to meet its timing constraints in term of response times, execution rates and

DECPS'17, October 19, 2017, Seoul, South Korea

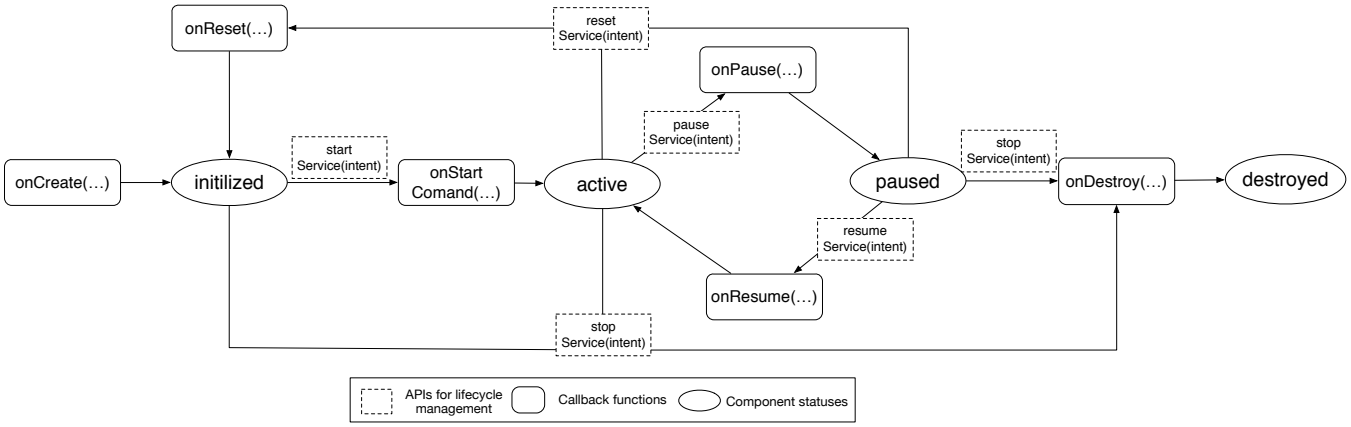


Figure 1: Lifecycle management of RTDroid's Real-time Service

the number of missed deadlines. Such validation mechanism serves two purposes: (1) It provides an automatic solution for application developers to verify if the application configuration is valid. (2) It provides a starting point for investigating a hybrid method for the worst case execution time.

The rest of the paper is organized as follows, Section 2 provides an overview of the task model for our application validation mechanism and discusses the compatibility between the RTDroid's components and existing real-time task models. Section 3 details the application validation process with the respect of task scheduling and bound checks of shared buffers and introduce how to enforce the memory bounds and buffer bounds during the boot process on RTDroid. Section 4 then shows a case study that applies the application validation process to a cochlear implant application running on RTDroid and reports our integration experiences.

2 BACKGROUND

RTDroid's programming model is derived from Android's programming model by extending Android's application components and the manifest schema. RTDroid inherits Android's event-driven nature and provides a familiar programing style to Android developers. Both Android and RTDroid programming models rely heavily on message passing based task communication, which significantly complicates the task model for validation. This section first introduces the design of the real-time components that form the bases of RTDroid's programming model. We then discuss how to model these components in a real-time task model that allows task communication.

2.1 Real-Time Components of RTDroid

RTDroid provides three basic application components—`RealTimeService`, `RealTimeReceiver` and `PeriodicTask`. To facilitate application validation, RTDroid synthesizes communication components for different types of message handling patterns. These are defines as RTDroid's real-time channels.

Similar to Android's components, RTDroid's real-time components are defined as abstract classes with a set of callback functions, these callback functions are used by developers to implement applications logic and invoked by lifecycle management APIs. Figure 1 depicts the lifecycle management of `RealtimeService`. Developers *must extend* the abstract class of `RealtimeService` and *implement* their application logic in callback functions. For example, `onStartCommand()`, which will be triggered, when other components or system services calls `startService(Intent)`. Then, once the execution of `onStartCommand()` is completed, `RealtimeService` enters into the state of "active". The following paragraphs describe the design of each component, respectively.

2.1.1 Real-time Service. The real-time service is a standalone component that serves as a controller over a group of nested components, such as real-time receivers and periodic tasks. Although every component in the same group has its own real-time properties, all component in a same real-time service share the same lifetime. This means that lifecycle management APIs listed in Figure 1 will start/stop/pause/resume both the real-time service as well as all of its nested components.

2.1.2 Real-time Receiver. The real-time receiver is used to listen to specific events from system services or even other applications. It can be implemented as a standalone component or a nested component of a real-time service. It has only two callback functions: `onReceive()`, which is used to implement responses to events, and `onReset()` to reset and clear data structures and memory associated with the real-time receiver. The callback function, `onReceive()`, is triggered by an broadcasting API—`sendBroadcast()`. The callback function, `onReset()`, can not be invoked programmatically, it is only invoked if the real-time receiver is reset by the RTDroid runtime system.

2.1.3 Periodic Task. The periodic task can be only be implemented as a nested component of a real-time service. The periodic task component differs from other RTDroid components as it does not have a corresponding component in Android. An instance of

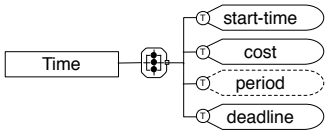


Figure 2: Timing Constrains

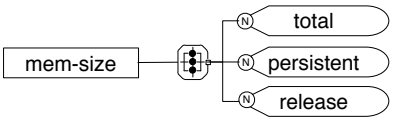


Figure 3: Memory Bounds

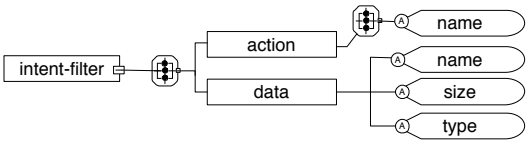
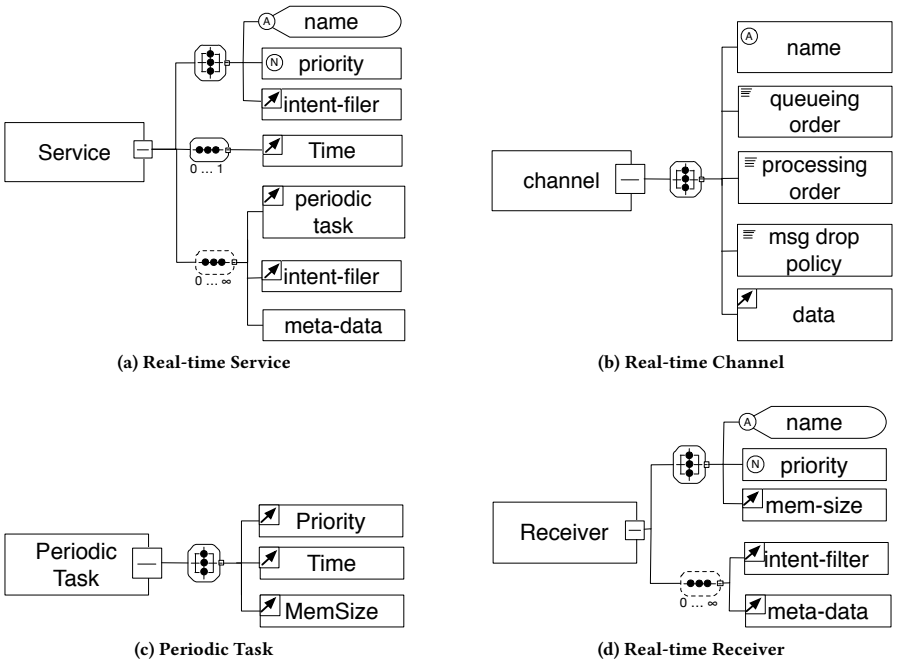


Figure 4: Channel Access



(a) Real-time Service

(b) Real-time Channel

(c) Periodic Task

(d) Real-time Receiver

Figure 5: Real-Time Schema for RTDroid's Components

periodic task has the same lifetime as the real-time service that contains it. `PeriodicTask` has only one interface, `onRelease()`, which embeds the logic to execute when the periodic task is released by the RTDroid scheduler.

2.1.4 Real-time Communication Channel. The real-time communication channels are built-in communication mechanisms on RTDroid. It is not an abstract class. To create a channel, a developer needs to declare the channel in the application manifest and specify the size of message sent in the channel, messages processing pattern and the maximum length of the message queue. If a component sends/receives messages to/from the channel, the component has to explicitly state itself as a producer/consumer of the channel and provides messages sending/receiving rate. There are four types of real-time channel built in RTDroid, we model them as tasks precedence and shared buffers. More details about the channel modeling will be illustrated as part of integration details Section 3. This section just focus on the discussion of the task model.

2.2 Task Model for RTDroid's Application Validation

Our validation mechanism utilizes a task model that accepts periodic tasks with hard deadlines as well as sporadic and aperiodic tasks with soft deadlines. It is based on a fixed-priority, periodic task model with rate monotonic scheduling. Feasibility tests are defined by [21, 29]. A number of assumptions were made to simplify the complexity of the feasibility tests. For example, every periodic task (τ_i) has an offset from system startup for the first release of the task and the length of time between successive releases must be a constant (T_i). Each release of a task must have a hard deadline (D_i). Most of the time such deadline (D_i) equals to T_i. Thereby, all tasks are independent of others and no interaction is allowed.

To support aperiodic and sporadic tasks and to enable task interactions, we adopt two extensions to the basic model: (1) The inclusion of aperiodic and sporadic tasks with soft deadlines, which

defines the periodicity of a sporadic task as the maximal interval of two release and utilizes an aperiodic server to repeatedly check requests of aperiodic releases [9, 12, 14, 41]. (2) Introduction of shared resources and task precedence in the feasibility analysis [11, 18, 23, 42].

Notice that the minimal execution unit of RTDroid's components is the callback function, so one execution of a callback function is referenced as a release of a real-time task in this work. `PeriodicTask` is directly mapped to a real-time periodic task with hard deadlines. If one release of a periodic task cannot be completed before the next release, it means that the application is not schedulable. We model real-time services and the real-time receivers as aperiodic tasks and sporadic tasks with soft deadlines since their releases are irregular. A real-time receiver is normally used to listening to an specific system/application event that occurs repeatedly within a minimum interval, so real-time receiver is modeled as the sporadic task. The real-time service is used to grouping a set of nesting components, it normally serves as a controller to manage the lifecycle of its nesting components. There is no specific interval between two releases, so we model real-time services as aperiodic tasks. The present implementation of RTDroid uses a *per-instance* server to execute `RealtimeService` or `RealtimeReceiver`, the server periodically checks release requests and invokes callback functions.

3 APPLICATION VALIDATION AND BOOTSTRAP

Section 2 has describe the task model in our validation process, this section presents integration details between RTDroid's real-time properties declared in its application manifest and feasibility tests in Cheddar [39]. Later, we also explain how the application boot process leverages the output of the application validation for task initialization and memory boundary enforcement.

3.1 Integration between RTDroid Manifest and Cheddar

RTDroid's compiler translates RTDroid's application manifest to a XML file understood by Cheddar [39] and performs the validation process, including scheduling simulation, calculating worst-case response times, and producing an upper bound of system utilization. Because RTDroid only supports an environment with a single core processor, all tests for this paper use uniprocessor hardware simulation in Cheddar. We also check memory bounds for real-time components and message bounds for real-time channels. These checks, however, are implemented in RTDroid's compiler and not in Cheddar.

3.1.1 Timing Constraints in Scheduling Simulation. Timing constraints of real-time components are modeled into a task model with fixed-priority scheduling consisting of three types of tasks: a periodic tasks with hard deadlines, sporadic tasks with soft deadlines, and aperiodic tasks with soft deadlines. Figure 5 shows a complete manifest schema. Each component *must* be declared with a unique name as its identifier; the importance of the component is defined by a `priority` element. Timing constraints are specified with a <time> element consisting of four child elements, given as timing parameters: `<start>`, `<cost>`, `<periodic>` and `<deadline>`, as shown in Figure 2.

Tags	Real-Time Tasks		
	Aperiodic Task	Sporadic Task	Periodic Task
<start>	-	-	The first release time
<cost>	Soft deadline	Soft deadline	The worst-cast time cost
<periodic>	-	MISR[1]	Periodicity
<deadline>	-	-	Hard deadline[2]

[1] The minimum interval between two successive releases.
[2] The period of a periodic task is equal to its deadline.

Table 1: Real-Time Semantics of Timing Elements

In real-time semantics, due to the restrict levels with deadlines, these timing parameters present slightly different meanings. *e.g.*, the element of `<cost>` is used as a soft deadline for the aperiodic and the sporadic task, but it is the worst-cast time cost of each release for a periodic task. Notice that the worst-case time cost is filled by application developers at the current version of RTDroid. Similarly, The element of `<periodic>` is the minimum interval between two successive releases for a sporadic task as listed in Table 1.

3.1.2 Feasibility Tests and Bounds Checking. There are two feasibility tests: (1) The schedulability test that produces an upper bound of processor utilization and worst-case response times of tasks; (2) The performance analysis with shared buffers/task precedences which leverages the theory of queuing systems to calculate two upper bounds—message waiting times and the maximum number of messages in a given buffer, respectively.

The schedulability test is based on a fixed-priority, preemptive rate monotonic scheduler in [21, 29]. The sporadic and aperiodic tasks are handled through a aperiodic server that periodically checks requests of aperiodic/sporadic tasks. There are different types serves proposed by Spuri *et. al.* [40, 42]. For performance analysis with shared buffers and task precedences, Table 2 shows the modeling of RTDroid's channel against queuing system theory [27, 28]. The message waiting times and the maximum number of messages are computed during the performance analysis, the later number is then used as the message bound of a real-time channel to check if any of its producer/consumer sends/receives message exceed this bound.

RTDroid utilizes on-stack memory management schema with two types of memory scopes: `<persistent>` and `<release>`. All real-time components *must* specify their expected memory usage via the element of `<mem−size>` defined in Figure 3 with three attributes:

Channel Name	**Channels in RTDroid**	**Real-Time Entities**
Message passing channel	Multiple producers. Consumers inherits the priority of the message producer. A consumer per priority.	A task precedence per priority.
Intent broadcasting channel	The broadcasting communication pattern with fixed number of participants. All participants declare their rates of message producing/consuming.	A shared buffer with M/M/s/∞/N.
Bulk data transfer channel	One-to-one data communication.	A receiver as the consumer, a task precedence. Periodic task as the consumer, shared buffer with M/D/1.
Cross-context channel	Unbound incoming messages, process messages with best efforts.	A shared buffer with M/G/1.

Table 2: Modeling between RTDroid's Channel to Real-Time Entities

(1) *Release:* The memory region is used to allocate temporal objects for the execution of callback functions, this region is reclaimed after each invocation.
(2) *Persistent:* The memory region is used to allocate objects that have the same lifetime as real-time components, this region is reclaimed when the associated component is destroyed.
(3) *Total:* The sum of the persistent, the release, and the memory usage of all children components if any exist.

Memory bound checking is performed at two levels. The first level is to check every single component if the sum of its persistent, its release, and its children's releases is equal to its total. The next level is to check the sum of total in all component is less than the amount of memory available for the real-time application itself within the RTDroid framework.

3.2 Application Bootstrap

The application bootstrap process is divided into two stages: a static compile time and a runtime boot procedure as depicted in Figure 6. The first step of static application compilation is to validate the real-time configuration to quantify temporal constrains and check any unbounded behavior over shared buffers defined in the application manifest. Then, RTDroid's compiler emits Java bytecode that overrides the constructor of each component in which application bootstrap instantiates an instance for declared real-time components, which assign timing parameters and allocates memory bounds or message objects for the runtime. Note, RTDroid does not allow any real-time component to manually change its timing properties at runtime. This is enforced by the programming model and exposed interfaces in RTDroid. The memory bound and utilization checks are enforced via instrumented bounds checks, inserted by the compiler, in the RTDroid runtime implementation. If memory constraints are violated, a pre-allocated runtime exception is generated by the runtime system.. For example, when the memory consumption of a real-time component exceeds its memory bound, the runtime checks will throw an `OutOfMemory` exception. This ensures clean failure semantics.

The bootstrap of a real-time application leverages results of static validation and makes the application ready for computation. As Figure 6 shows, after the JVM is initialized, it invokes an entrance function of the application and loads generated bytecode. Then, each component is allocated based on the real-time properties defined in the manifest and registered with an internal component manager for lifecycle management during the application runtime. Memory regions and message objects are preserved to guarantee a bounded response time with shared buffers and task precedences as discussed above.

4 CASE STUDY: COCHLEAR IMPLANT APPLICATION ON RTDROID

This section uses a simulated cochlear implant application as a case study to reports our experiences of integrating RTDroid's validation mechanism with an existing real-time framework, Cheddar. The cochlear implant can restore hearing abilities through a surgically inserted electronic device in a patient's inner ear. Figure 7 shows an external device used for capturing ambient audio and converting audio samples into digital signals, and an implanted device that translates signals into electrical energy and triggers implanted electrodes to simulate hearing nerves. Recently, there has been

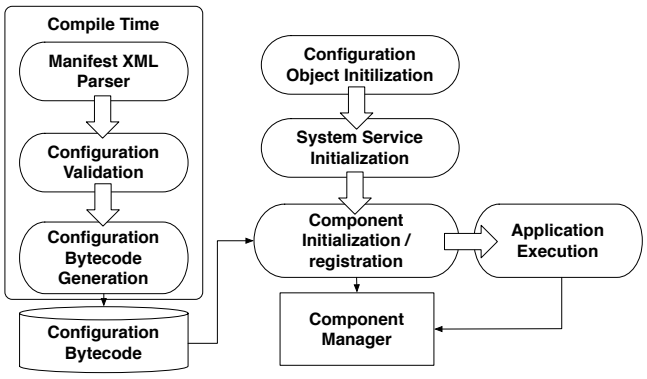


Figure 6: Bootstrap Procedure of RTDroid Application

interests in replacing the external device with a smartphone for audio sampling and processing to reduce the number of devices the patient must carry [2, 10] and enable the possibility of changing the built-in signal processing algorithms for better performance. To keep the patient rapidly response in daily conversation, a fixed amount of audio samples (*128 audio samples*) must be processed on the smartphone and delivered to the implanted device every 8ms.

Our previous work [47] implemented such cochlear implant application with three independent components on RTDroid, as listed in Table 3. It has two real-time services that control periodic tasks for audio recording and processing, respectively. The recording task sends captured audio samples to the processing task via a message passing channel (modeled as a task precedence). Audio samples are buffered and processed. After performing the signal processing algorithm, the processing task broadcast its processing signal to a real-time receiver (*a sporadic task*) as an output receiver. The receiver performs error-checking by simulating the behavior of implanted device. To enforce the timing constrain (8ms), we model the recording task, the processing task and the output receiver with descending priorities starting from 90 (the highest priority allowed in RTDroid application), while limits their periods with 8ms and time cost with 1ms. The cost time is estimated through an experimental measurement in [10], the time cost of computation for sampling, processing and error checking mostly less than 1ms.

We have implemented the integration process as part of the procedure in RTDroid's application compilation. The compiler translates RTDroid's application manifest to an input XML file loaded in Cheddar v3.3[1] There are a number of workaround that must be token to utilized the scheduling simulation and feasibility tests. According to our best knowledge, they are due to the fact of limitations of Cheddar's implementation, including the incompleteness of task precedence and shared buffer with the scheduling simulation and buffer feasibility tests. For example, the the simulation doesn't run with the present of aperiodic tasks. The implementation of message queuing models for shared buffers are not completed. We have made the following workaround to get task simulation and feasibility tests: (1) we manually convert the output receiver (a sporadic tasks) triggered by task precedences to a periodic task since it is triggered in every release by the processing service. (2) The feasibility tests on shared buffer can only produce meaningful results with P/P/1 model.

[1]Compiled from SVN repository revision 1835, last source code changed on 2015-08-19.

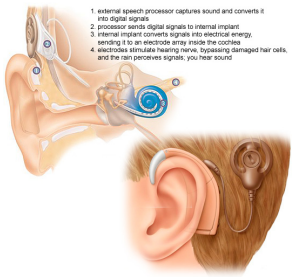


Figure 7: Cochlear Implant

Task	Priority	Start	Cost	Period (Deadline)
Recording Service	90	0	1	8
Processing Service	89	0	1	8
Output Receiver	88	0	1	8

Table 3: Real-Time Properties in Cochlear Implant

We report the scheduling simulation of the cochlear implant application in Cheddar, as shows in Figure 8. As they are in a descending order and scheduled with the policy of `POSIX_HIGHTEST_PRIORITY_FIRST`, the recording service is scheduled first, then processing service and the output receiver at last within 80ms repeatedly. Additionally, the scheduling feasibility test calculates processor utilization at 37.5 % and WCRTs are 1ms, 2ms, 3ms with the descending priorities. Given to the scheduling simulation, the message size bound of the massage passing channel is always 1, the maximum waiting time is 1 as well.

5 RELATED WORK

The study of real-time scheduling started with the hard deadline case in which any task with a missed deadline was considered to be a failure [13, 21, 29]. To build a system that would guarantee temporal constrains as well as logical behavior, it was necessary to consider a worst-case formulation. Later, Gardner *et. al.* [19, 20] model real-time constrains with soft deadlines. They are defined with soft deadlines and a lateness constraint. For example, it defines $\alpha(x)$ to be the long run fraction of jobs that miss their deadlines by more than x time units, then lateness constraints are typically of the form $\alpha(x) \leq \beta$.

Meanwhile, the real-time task model has also has been evolved to include with more real-world aspects, such as blocking due to synchronization, precedence constraints, mode changes, operating system overhead and architectural details [11, 18, 23, 42, 42]. The task model in our work requires a task model with the support of these aspects. Unfortunately, according to our experiences, although real-time scheduling theory has been most fully developed to predict whether a set of tasks will meet their hard deadlines given a certain set of assumption about the task set and system, fewer analytic results are available for soft deadlines.

There are a number of projects that proposed different architectures for a real-time capable Android paltform [30–32, 34, 37], but most of them require a strict separation between real-time and non real-time applications. Kang *et al.* [26] and Ruiz *et al.* [38] implemented such separation in the standard Linux kernel, assigning one or more cores for real-time tasks and isolating those cores from the rest of the system. Kalkov *et al.* [24] proposed to explicitly trigger the GC to reduce pause time during critical periods. The result was reduced latency in the system, but hard guarantees were still not possible to assure. RTDroid strives to make application interactions safe and avoids to manually invoke the garbage collection as choosing when to run the GC is difficult. Kalkov *et al.* also explored how components interact through intents, providing a mechanism to prioritize intents [25], but did not provide any memory bounds

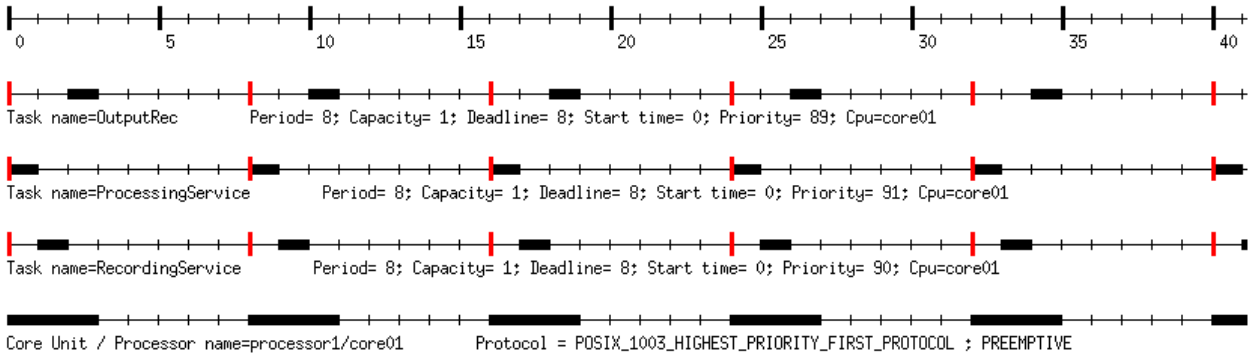


Figure 8: Scheduling Simulation of Cochlear Implant Application in Cheddar

on communication. RTDroid provides static bounds on memory consumption of communication between tasks, allows communication between real-time and non real-time tasks, and observers that only prioritizing intents can induce priority inversion in the callbacks that handle those intents. Other efforts have the left the Android framework unmodified [33], instead focusing on exposing the degree of jitter present in sensor data in the system so that applications can make necessary adjustments. RTDroid aims to eliminate such jitter.

6 CONCLUSION AND FUTURE WORK

In this paper we introduced our first automated mechanisms for application validation in RTDroid. Since RTDroid constructs are complex, internally constructed from potentially multiple components, their translation to corresponding tasks in a real-time task model is non trivial. We have shown a preliminary mapping of RTDroid constructs into a task model and provided a case study that validate real-time properties in a cochlear implant application and detailed our experiences as well as observed limitations in Cheddar.

Our next steps are overcoming the current limitations of our translation of RTDroid constructs into Cheddar. We also aim to explore typing mechanisms, specifically session types, to validate communication properties of the system. Specifically we will be exploring adding resource bound analysis to session types, to infer limits on the number of active messages in an application. Such information can be used to atomically infer the size of memory regions and channel queues in the RTDroid runtime.

REFERENCES

[1] Android and RTOS Together: The Dynamic Fuo for Today's Medical Devices. http://embedded-computing.com/articles/android-rtos-duo-todays-medical-devices/.
[2] Android-Based Research Platform for Cochlear Implants. http://goo.gl/EJuh1i.
[3] Android Based Robotics: Powerful, Flexible and Inexpensive Robots for Hobbyists, Educators, Students and Researchers. http://www.socsci.uci.edu/~jkrichma/ABR/abr_background.html.
[4] Float Sensor NetWork. http://float.berkeley.edu/.
[5] Robots and Androids—Tomorrow's Robotics Today. http://www.robots-and-androids.com/.
[6] SmartBot: your Smartphone robot. http://www.overdriverobotics.com/.
[7] Why Android Will Be The Biggest Selling Medical Devices in The World By The End of 2012. http://goo.gl/G5UXq.
[8] Xing and collaborators travel to Ecuador to monitor the Tungurahua Volcano. http://www.cse.msu.edu/About/Notable.php?Nid=423.
[9] T. F. Abdelzaher, V. Sharma, and C. Lu. A utilization bound for aperiodic tasks and priority driven scheduling. *IEEE Trans. Comput.*, 53(3):334–350, Mar. 2004.
[10] H. Ali, A. P. Lobo, and P. C. Loizou. Design and Evaluation of A Personal Digital Assistant-Based Research Platform for Cochlear Implants. *Biomedical Engineering, IEEE Transactions on*, 60(11):3060–3073, 2013.
[11] J. H. Anderson, S. Ramamurthy, and K. Jeffay. Real-time computing with lock-free shared objects. In *Proceedings of the 16th IEEE Real-Time Systems Symposium*, RTSS '95, page 28, Washington, DC, USA, 1995. IEEE Computer Society.
[12] S. K. Baruah, A. K. Mok, and L. E. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Real-Time Systems Symposium, 1990. Proceedings., 11th*, pages 182–190. IEEE, 1990.
[13] I. Bate and A. Burns. Schedulability analysis of fixed priority real-time systems with offsets. In *Real-Time Systems, 1997. Proceedings., Ninth Euromicro Workshop on*, pages 153–160. IEEE, 1997.
[14] G. Bernat and A. Burns. New results on fixed priority aperiodic servers. In *Proceedings of the 20th IEEE Real-Time Systems Symposium*, RTSS '99, pages 68–, Washington, DC, USA, 1999. IEEE Computer Society.
[15] W. C. Blog. What OS Is Best for a Medical Device? http://www.summitdata.com/blog/?p=68.
[16] cherylcoupe. Roving Reporter: Medical Device Manufacturers Improve Their Bedside Manner with Android. http://goo.gl/d2JF3.
[17] M. Faulkner, M. Olson, R. Chandy, J. Krause, K. M. Chandy, and A. Krause. The next big one: Detecting earthquakes and other rare events from community-based sensors. In *Information Processing in Sensor Networks (IPSN), 2011 10th International Conference on*, pages 13–24. IEEE, 2011.
[18] J. J. G. Garcia and M. G. Harbour. Optimized priority assignment for tasks and messages in distributed hard real-time systems. In *Proceedings of the 3rd Workshop on Parallel and Distributed Real-Time Systems*, WPDRTS '95, pages 124–, Washington, DC, USA, 1995. IEEE Computer Society.
[19] M. K. Gardner. *Probabilistic Analysis and Scheduling of Critical Soft Real-time Systems.* PhD thesis, Champaign, IL, USA, 1999. AAI9953022.
[20] M. K. Gardner and J. W.-S. Liu. Analyzing stochastic fixed-priority real-time systems. In *Proceedings of the 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems*, TACAS '99, pages 44–58, London, UK, UK, 1999. Springer-Verlag.
[21] R. Ha and J. W. Liu. Validating timing constraints in multiprocessor and distributed real-time systems. In *Distributed Computing Systems, 1994., Proceedings of the 14th International Conference on*, pages 162–171. IEEE, 1994.
[22] iOmniscient. Fall and Man Down Detection. http://iomniscient.com/index.php?option=com_content&view=article&id=155&Itemid=53.
[23] K. Jeffay. Scheduling sporadic tasks with shared resources in hard-real-time systems. In *Real-Time Systems Symposium, 1992*, pages 89–99. IEEE, 1992.
[24] I. Kalkov, D. Franke, J. F. Schommer, and S. Kowalewski. A Real-Time Extension to The Android Platform. In *Proceedings of the 10th International Workshop on Java Technologies for Real-time and Embedded Systems*, JTRES '12, pages 105–114, New York, NY, USA, 2012. ACM.
[25] I. Kalkov, A. Gurghian, and S. Kowalewski. Predictable Broadcasting of Parallel Intents in Real-Time Android. In *Proceedings of the 12th International Workshop on Java Technologies for Real-time and Embedded Systems*, JTRES '14, pages 57:57–57:66, New York, NY, USA, 2014. ACM.
[26] H. Kang, D. Kim, J. Kang, and K. Kim. Real-time motion control on android platform. *The Journal of Supercomputing*, 72(1):196–213, 2016.
[27] L. Kleinrock. *Queueing systems, volume 2: Computer applications*, volume 66. wiley New York, 1976.

[28] L. Klennrock. Queueing systems volume 1: theory. *New York*, 1975.
[29] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)*, 20(1):46–61, 1973.
[30] 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*, OSPERT '10, pages 63–70, 2010.
[31] W. Mauerer, G. Hillier, J. Sawallisch, S. Hönick, and S. Oberthür. Real-time Android: Deterministic Ease of Use. In *Proceedings of Embedded Linux Conference Europe, ELCE*, volume 12, 2012.
[32] H.-S. Oh, B.-J. Kim, H.-K. Choi, and S.-M. Moon. Evaluation of Android Dalvik Virtual Machine. In *Proceedings of the 10th International Workshop on Java Technologies for Real-time and Embedded Systems*, JTRES '12, pages 115–124, New York, NY, USA, 2012. ACM.
[33] E. Peguero, M. Labrador, and B. Cook. Assessing jitter in sensor time series from android mobile devices. In *2016 IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 1–8, May 2016.
[34] L. Perneel, H. Fayyad-Kazan, and M. Timmerman. Can Android Be Used for Real-Time Purposes? In *Computer Systems and Industrial Informatics (ICCSII), 2012 International Conference on*, pages 1–6. IEEE, 2012.
[35] F. Pizlo, L. Ziarek, E. Blanton, P. Maj, and J. Vitek. High-Level Programming of Embedded Hard Real-Time Devices. In *Proceedings of the 5th European conference on Computer systems*, EuroSys '10, pages 69–82, New York, NY, USA, 2010. ACM.
[36] J. Qiu, D. Chu, X. Meng, and T. Moscibroda. On the feasibility of real-time phone-to-phone 3d localization. In *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*, SenSys '11, pages 190–203, New York, NY, USA, 2011. ACM.
[37] G. J. Rajguru. Reliable Real-Time Applications on Android OS. *International Journal of Management, IT and Engineering*, 4(6):192, 2014.
[38] A. P. Ruiz, M. A. Rivas, and M. G. Harbour. Cpu isolation on the android os for running real-time applications. In *Proceedings of the 13th International Workshop on Java Technologies for Real-time and Embedded Systems*, JTRES '15, pages 6:1–6:7, New York, NY, USA, 2015. ACM.
[39] F. Singhoff, J. Legrand, L. Nana, and L. Marcé. Cheddar: a flexible real time scheduling framework. In *ACM SIGAda Ada Letters*, volume 24, pages 1–8. ACM, 2004.
[40] B. Sprunt. *Aperiodic task scheduling for real-time systems*. PhD thesis, PhD thesis, 1990.
[41] B. Sprunt, L. Sha, and J. Lehoczky. Aperiodic task scheduling for hard-real-time systems. *Real-Time Systems*, 1(1):27–60, Jun 1989.
[42] M. Spuri and J. A. Stankovic. How to integrate precedence constraints and shared resources in real-time scheduling. *IEEE Transactions on Computers*, 43(12):1407–1412, 1994.
[43] PhoneSat. http://www.nasa.gov/directorates/spacetech/small_spacecraft/phonesat.html.
[44] Y. Yan, C. Chen, K. Dantu, S. Y. Ko, and L. Ziarek. Using a multi-tasking vm for mobile applications. In *Proceedings of the 17th International Workshop on Mobile Computing Systems and Applications*, pages 93–98. ACM, 2016.
[45] Y. Yan, S. Cosgrove, V. Anand, A. Kulkarni, S. H. Konduri, S. Y. Ko, and L. Ziarek. Rtdroid: A design for real-time android. *IEEE Trans. Mob. Comput.*, 15(10):2564–2584, 2016.
[46] Y. Yan, S. Cosgrove, E. Blantont, S. Y. Ko, and L. Ziarek. Real-time sensing on android. In *Proceedings of the 12th International Workshop on Java Technologies for Real-time and Embedded Systems*, page 67. ACM, 2014.
[47] Y. Yan, K. Dantu, S. Ko, J. Vitek, and L. Ziarek. Making Android Run on Time. In *Real-Time and Embedded Technology and Application Symposium*, RTAS '17, Washington, DC, USA, April 2017. IEEE Computer Society.
[48] Y. Yan, S. H. Konduri, A. Kulkarni, V. Anand, S. Ko, and L. Ziarek. Real-Time Android with RTDroid. In *The 12th International Conference on Mobile Systems, Applications, and Services*, MOBISYS '14, New York, NY, USA, 2014. ACM.
[49] L. Ziarek and E. Blanton. The fiji multivm architecture. In *Proceedings of the 13th International Workshop on Java Technologies for Real-time and Embedded Systems*, JTRES '15, pages 9:1–9:10, New York, NY, USA, 2015. ACM.