Salma Elmalaki *University of California, Los Angeles* **Lucas Wanner** *University of Campinas, Brazil*
Mani Srivastava *University of California, Los Angeles*

Editors: Robin Kravets and Nic Lane

CAreDroid:
Adaptation Framework for Android Context-Aware Applications

Excerpted from "CAreDroid: Adaptation Framework for Android Context-Aware Applications," from *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*. http://dl.acm.org/citation.cfm?id=2790108 © ACM 2015

Context-awareness is the ability of software systems to sense and adapt to their physical environment. Many contemporary mobile applications adapt to changing locations, connectivity states, and available energy resources. Nevertheless, there is little systematic support for context-awareness in mobile operating systems. Because of this, application developers must build their own context-awareness adaptation engines, dealing directly with sensors and polluting application code with complex adaptation decisions. However, with adequate support from the runtime system, context monitoring could be performed efficiently in the background and adaptation could happen automatically [1]. Application developers would then only be required to implement methods tailored to different contexts. Just as file and socket abstractions help applications handle traditional input, output, and communication; a context-aware runtime system could help applications adapt according to user behavior and physical context.

Motivated by this idea, we introduce CAreDroid. CAreDroid is a framework that is designed to decouple the application logic from the complex adaptation decisions in Android context-aware applications. In this framework, developers are required only to focus on the application logic by providing a list of methods that are sensitive to certain contexts along with the permissible operating ranges under those contexts. At runtime, CAreDroid monitors the context of the physical environment and intercepts calls to sensitive methods, activating only the blocks of code that best fit the current context.

CAreDroid ARCHITECTURE

The main objective of CAreDroid is to provide the application developer with support to easily build adaptation in context-aware applications. Hence, from a developer perspective, the design of CAreDroid needs to satisfy the following properties:

1. Usability: CAreDroid needs to add minimal overhead on the application developer at development time.

2. Performance: The adaptation engine needs to add minimal execution overhead when the application is running.

Motivated by these two design objectives, we designed CAreDroid. A conceptual overview of CAreDroid architecture is shown in Figure 1. Applications normally call methods f and g. Each method is aliased to one of its versions (f1 and g2, respectively, in the example). A configuration file describes rules that determine under what context each of the method versions should be used. Method f1 could define, for example, two rules stating that Wi-Fi connectivity and battery charging status should be equal to true, while f2 could define one rule stating that remaining battery capacity should be between 0% and 20%. Rules are assigned priorities that help determine which of the versions should be used when multiple rules are valid.

In the system layer, CAreDroid parses the configuration file to discover adaptable methods and their rules of operation. A context-monitoring module abstracts the various sensors in the system, and exposes context information to an adaptation engine. When changes in context occur, the adaptation engine changes the aliasing of the adaptable methods according to the rules. If more than one version of a method matches the current context, the priorities of the rules are used to choose between them. CAreDroid is organized in three modules: CAreDroid Configuration File, CAreDroid Context Monitoring, and CAreDroid Adaptation Engine. In the following section we will describe each module in detail.

CAreDroid ALLOWS APPLICATION DEVELOPERS TO DEVELOP CONTEXT-AWARE APPLICATIONS WITHOUT HAVING TO DEAL DIRECTLY WITH CONTEXT MONITORING AND CONTEXT ADAPTATION IN THE APPLICATION CODE

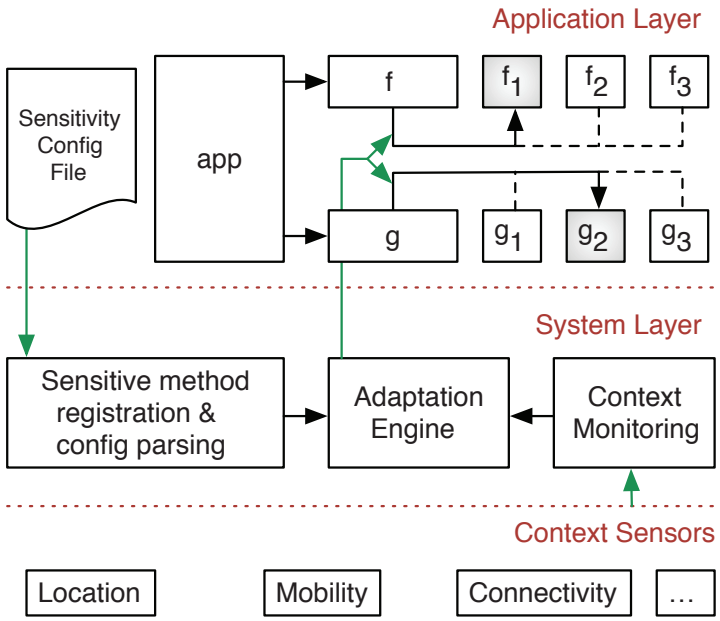


FIGURE 1. CAreDroid System Architecture: The developer provides a set of methods and provides a configuration file describing how these methods shall be called. At runtime, CAreDroid monitors the phone context and adapts the application behavior accordingly.

CAreDroid CONFIGURATION FILE

For each context-aware application, a configuration file maps methods to contexts is defined. The file is a straightforward description of acceptable ranges of operation for each sensitive method under different contexts.

The configuration file is an XML file that is supplied by the developer. For each sensitive method, the developer provides different method implementations. Each method version is described by a name, a tag, and a priority. The name corresponds to the method name in source code. The tag associates different implementations of a method with one another. For example, if methods f1 and f2 are different implementations of the same method, then they must be associated with the same tag, for example, f. Finally, the priority for a method helps the system resolve ambiguities when multiple versions of a method satisfy the current context. For each different implementation, the developer assigns a sensitivity list. This sensitivity list is the list of context states for which this implementation shall be triggered.

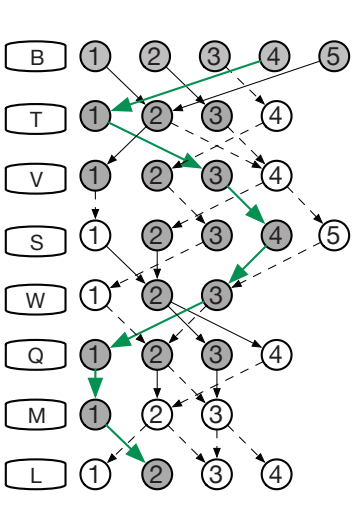
To illustrate the construction of the configuration file, we provide a small example in Figure 2. In this example, we have two sensitive methods for adjusting the camera parameters under different contexts. One method, AdjustCameraPowerAware, is designed to save energy. Hence, its BatteryCapacity range is from 0% up to 25%, and it can execute whether Wi-Fi is on or off. The second method is dedicated to adjusting the camera while the device user is running. For example, this method should adjust the focus and the scene parameters of the camera to give a better quality image. Accordingly, the mobility is assigned to be run. Currently in CAreDroid we support eight contexts; three power-related contexts (battery capacity, battery temperature, and battery voltage), and three connectivity-related contexts (signal strength, Wi-Fi connectivity, and quality of the signal) and two mobility-related contexts (current activity, and current location).

CAreDroid CONTEXT MONITORING

CAreDroid has a dedicated module that continuously probes the current phone context. CAreDroid supports both raw

```
<Method>
  <MethodName>AdjustCameraPowerAware
  </MethodName>
  <priority>1</priority>
  <tag>cameraAdjust</tag>
  <batterycapacity>
    <vstart>0</vstart>
    <vend>25</vend>
  </batterycapacity>
</Method>
<Method>
  <MethodName>AdjustCameraWhileRunning
  </MethodName>
  <priority>2</priority>
  <tag>cameraAdjust</tag>
  <batterycapacity>
    <vstart>20</vstart>
    <vend>100</vend>
  </batterycapacity>
   <mobility>run</mobility>
</Method>
```

FIGURE 2. Snippet of CAreDroid configuration file.



M1	M2	M3	M4	M5
1	2	3	4	5
2	3	4	1	2
1	4	2	3	4
1	2	3	4	5
2	2	1	3	3
4	3	2	1	2
2	3	3	1	2
1	4	3	2	3

FIGURE 3. An example of Decision Graph: The nodes at each level correspond to the five methods shown in the right table with their defined context range. The shaded nodes correspond to the range that matches the current context. The solid arrows correspond to the active paths of the methods that match the current context. Finally, the path marked in green corresponds to the method that satisfies all the ranges, and therefore this method is the one picked by CAreDroid for execution.

contexts that can be directly known by reading the state of the hardware (e.g. WiFi connectivity, battery level) as well as inferred contexts, such as mobility status (e.g., walking, running) that require advanced processing of sensor information.

In this section, we describe how CAreDroid acquires the current context at runtime with less overhead than Android Java APIs. The work in this section can be indeed complemented by any of the context monitoring systems that appeared currently in the literature.

Raw Context Monitoring: Android exposes sensor information to the software stack through a Hardware Abstraction Layer (HAL). The HAL features a set of sensor managers that work as an intermediate layer between the low-level drivers and the high-level applications. In order to reduce the overhead, we need to bypass the HAL layer and the associated sensor managers. This can be done by snooping on the interface between the HAL and the low-level device drivers through the sysfs virtual file system. In our work, we create an internal Dalvik VM thread that continuously reads these files.

Inferred Context: Mobility state detection is calculated by processing the raw accelerometer data obtained by the internal VM thread described above. In order to infer the mobility state, we adapt the classification procedure described in [2] to detect whether the user is stationary, walking, or running.

CAreDroid ADAPTATION ENGINE

Alternative implementations of sensitive methods are connected together through a map that lists all candidate methods that can be used for a method call. Whenever more than one candidate implementation fits the current context, CAreDroid uses a conflict resolution mechanism to pick the method with the highest priority. Because it frees developers from having to implement adaptation strategies in the application layer, the CAreDroid adaptation engine is the main factor in meeting our goal of decreasing development complexity for context-aware applications.

The adaptation Engine is the core of CAreDroid. It is where the method replacement happens at runtime. In this section, we explain how CAreDroid extends the execution phase of the Android flow to automatically and transparently switch between methods.

Which method Implementation to pick?

In order to choose the correct method that best suits the current context, our framework utilizes the data supplied by the developer in the configuration file.

Best Fit Vs Must Fit: The first step is to choose a set of candidate methods. We allow for two policies. In the first policy, Must Fit, a method is considered a valid candidate if the current context satisfies all the operation ranges for all sensitive contexts. In the second policy, Best Fit, a method is a valid candidate if the current context satisfies at least one operation range of the sensitive contexts. The choice of policy is determined in the configuration file.

Decision Graph: We use a Directed Acyclic Graph (DAG) to choose the candidate method. Each level of the graph marks one sensitive context (e.g. battery capacity, mobility state). The sensitive contexts are ordered based on their priority as defined in the configuration file. For each sensitive context, we create nodes for all the distinct operation ranges for each context. For example, if the range of battery capacity for one sensitive method is 20% - 50% while it is form 0% - 30% for the second sensitive method, and 40% -100% for a third one, then we have three distinct ranges for battery capacity. Distict ranges for all the contexts are created and a graph is constructed to connect between these nodes to mark one path of distinct ranges of context for each method. An example for the decision graph is shown in Figure 3.

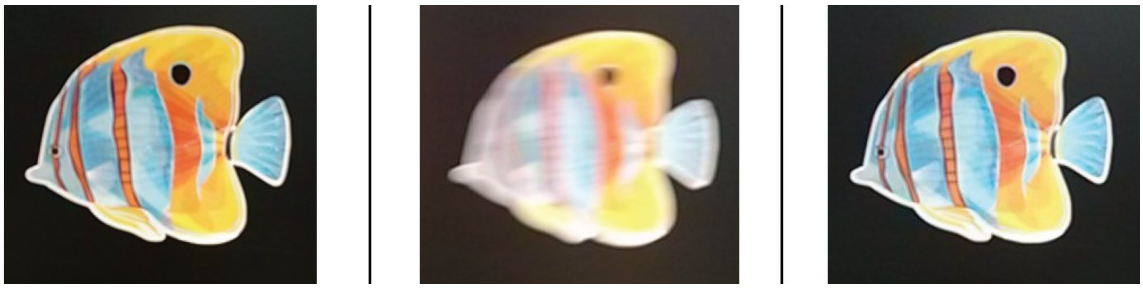


FIGURE 4. Photos taken by the Smart Camera application: Photo taken while the phone holder is standing still (left); photo taken while the phone holder is walking and no context-awareness is taking place (middle); and photo taken while the phone holder is walking and using the Smart Camera application built on top of CareDroid (right).

At runtime, when the phone context is reported, some paths in the graph become active and some become inactive. The final step is to use the method that satisfies the selected active path and pass it to the Android interpreter to execute it.

EVALUATION

We evaluate CAreDroid on Nexus 4 phone running a modified system image for platform 4.2 API 17. The overhead in the system image that supports CAreDroid is 4.6%.

Case Study: Context-Aware Camera

In this case study, we build a context-aware camera application. The camera adjusts its features based on the phone context. We have five different methods that CAreDroid alternates between. The objective of this application is to make the focus, scene mode and flash mode adaptive to the context. However, this can be extended to handle all the camera features. The five modes are listed as follows:

- Default: Configure the focus mode to "default."
- When running: adjust the focus mode to the "continuous picture" mode.
- When walking: adjust the scene mode to the "steady photo" mode.
- When still: adjust the focus mode to the "fixed" mode.
- Power saver: (1) adjust the flash mode to "off," (2) adjust the focus mode to "fixed" mode, and (3) adjust the quality of the picture to "minimum."

We implemented a different method for each of these modes and the objective is to call the appropriate method based on the phone context.

The results of the implemented application are shown in Figure 4. Figure 4(a) shows the original photo captured from a standstill position. Figure 4(b) shows the captured photo while the phone holder is walking and no context-awareness processing is taking place, and finally, Figure 4(c) shows the captured photo while the user is walking and using the developed context-aware camera with CAreDroid. The reduction in SLOC when CAreDroid context-aware Camera application is used versus normal Android flow is more than a factor of 3×.

CONCLUSION

Context-aware computing is a powerful technique for physically coupled software. It can enhance functionality and improve resource usage of applications by adapting them to context. In this paper, we present CAreDroid, an adaptation framework for context-aware applications in Android. CAreDroid allows application developers to develop context-aware applications without having to deal directly with context monitoring and context adaptation in the application code. In CAreDroid, multiple versions of methods that are sensitive to context are dynamically and transparently replaced with each other according to application-specific configuration file. ■

Salma Elmalaki is a PhD candidate in the Electrical Engineering department at UCLA. Her research interests include designing context-aware systems with special focus on performance, reliability and security and privacy issues. She is the recipient of the Best Paper Award at MobiCom 2015, Best Community Paper Award at MobiCom 2015, and the 2016/2017 Microsoft PhD fellowship.

Lucas Wanner is a faculty member of the Institute of Computing at the University of Campinas, Brazil. He received a PhD in Computer Science in 2014 from UCLA. His research is on runtime systems and energy optimization for embedded applications.

Mani Srivastava is on the faculty at UCLA. His research is in networked human-cyber-physical systems, and spans problems across the entire spectrum of applications, architectures, algorithms, and technologies. He is a Fellow of the IEEE. More information about his group's research can be found at http://nesl.ee.ucla.edu.

REFERENCES

[1] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos. Context aware computing for the internet of things: A survey. Communications Surveys Tutorials, IEEE, 16(1):414–454, First 2014.

[2] J. Ryder, B. Longstaff, S. Reddy, and D. Estrin. Ambulation: A tool for monitoring mobility patterns over time using mobile phones. In Computational Science and Engineering, 2009. CSE'09. International Conference on, volume 4, pages 927–931. IEEE, 2009.