Mobile Apps: It's Time to Move Up to CondOS

David Chu† Aman Kansal† Jie Liu† Feng Zhao◇

† Microsoft Research Redmond ◇ Microsoft Research Asia

1 Introduction

Sensing is a significant contributor to the current mobile computing revolution. Today's typical smartphone has more than eight sensors, including multiple mics, cameras, accelerometers, gyroscopes, a GPS, a digital compass, and proximity sensors. These sensors not only provide natural user interaction with the device, but also offer tantalizing opportunities for context-aware computing.

A rich history of work has investigated algorithms for converting raw sensor data into context, and their specific usages [3]. To cite just two examples: A restaurant finder app may adjust its search radius depending on whether a user is on foot, cycling, or driving, which can be inferred from GPS and IMU[1] readings [11]. A Twitter app might choose to alert the user of her latest updates at an interruptible moment such as when she is not engaged in conversation, which can be inferred from the mic's audio [5].

The ingredients for context appear ready: the sensing hardware, the data processing algorithms and the application scenarios are all primed. The question that emerges is: who is responsible for context generation?

One option is for apps to manage their own context generation. This approach appears appealing because apps are most familiar with their own context needs. However, many mobile OSs such as iPhone's iOS and Windows Phone's WP7 harbor legitimate energy concerns and severely restrict non-foreground processing. As a result, an app may generate context from immediately available sensor data, but is unable to maintain context while outside the scope of its execution. This can be as simple as missing the accelerometer's transition from *sitting* to *standing*, since sensing either state outside the transition period does not yield distinguishing information. Alternatively, Android apps may run in the background, but then the user is at the mercy of the flawless app developer to use resources intelligently. Another option is to simply ship all sensor data to the cloud for processing. However, the high power cost of radio communication and the latency requirements of certain apps like games quickly curtails this option.

[1]Inertial Measurement Unit, responsible for acceleration and orientation measurements

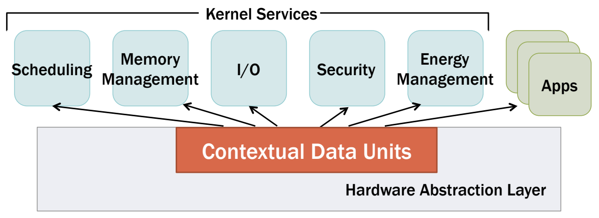

Figure 1: OS services and apps ought to benefit from context generated by the OS.

We take the unorthodox view that the operating system ought to shoulder responsibility for converting raw sensor data to *Contextual Data Units (CDUs)*. The CDU is simply an abstract data type that embodies a unit of context meaningful to applications. A concrete CDU such as *user motion state* might take values such as *sitting*, *standing*, *walking*, and *driving*. In contrast, traditional OSs offer apps opaque bytes, a very low level and unassuming interface to data. We suggest that the right approach for mobile systems is a much higher level of data abstraction: the CDU.

Though there will be challenges to incorporating context generation in the OS, it may lead to two main benefits. First, not only apps, but also core OS services can subscribe and respond to context changes (Figure 1). Our initial investigation suggests that memory management, scheduling, I/O and security may all benefit. Second, OS-managed CDUs offer tighter quality control of user experience, resulting in lower app battery impact and privacy exposure of raw sensor data. In turn, this can lead to interesting user-to-user context sharing opportunities. §2 discusses these two points in detail.

We propose the design of a new mobile phone operating system *Context Dataflow Operating System* – or simply CondOS – that embraces context generation. CondOS provides both applications and internal OS services the ability to query or subscribe to CDUs while protecting raw sensor data privacy. A concrete set of CDUs will form an initial context vocabulary. Though we are not yet prepared to finely delineate a base vocabulary, we hope our examples throughout this paper can facilitate the dialog. In complement, CondOS provides a way for apps to extend the CDU vocabulary by defining their own CDUs for inclusion in the kernel. CDUs are defined by their *CDU Generators*, which are graphs of processing components. CondOS incorporates CDU Generators into its composite context dataflow, akin to the Click Modular Router's packet dataflow [7]. CondOS's centralization of context generation and dataflow design permits several opportunities for energy reduction, such as shared dataflow processing, dataflow-to-hardware mapping, and principled flow degradation. §3 outlines the CondOS design.

2 Advantages of OS-managed Context

2.1 Context-Aware OS Services

CDUs can inform the OS as well as applications. This section looks at scenarios in which context may benefit traditional OS services on the phone.

Memory Management: Memory is scarce on mobile devices, yet users are demanding multi-tasking and faster app startup times. Like traditional OSs, mobile OSs only load applications into memory upon user request, which leads to slow load times, especially for data intensive apps like games.

Context can provide good hints when deciding which applications to preload. As an example, consider accelerometer-based activity detection [11]. Idle standing-around CDUs may suggest preloading a casual game app. Running or walking may suggest loading a music or workout app. Driving may suggest loading a maps or restaurant finder app. As another example, consider logical location CDUs such as *home* and *office*. It may be more appropriate to preload Twitter, Facebook, and gaming apps in the former, and calendar and productivity apps in the latter.

In a similar vein, context ought to aid memory eviction decisions. Mobile OSs such as Android and iOS do not use virtual memory but rather evict entire applications from memory when contention arises. CondOS can improve upon LRU eviction by evicting unlikely candidates for future access. Elaborating upon the example above, gaming while driving may be relatively rare, so once driving activity is detected, the OS ought to evict games from memory even if they were recently accessed.

Security: The OS can use contextual data hints to mix tight security while retaining convenience. For example, a sanctuary such as *home* could suggest relaxing the home screen's password lock, enabling Bluetooth device discovery and liberalizing access control to photos and music for sharing with other local devices.

Lending one's phone to friends and family, even temporarily, poses security risks due to the amount of personal information and apps on the phone. For example, biometric context [1] might help to differentiate the phone's owner (who ought to have full access), from the owner's child (who is authorized to access only educational content), from the owner's casual acquaintances (who are authorized to access only the phone dialer). Certainly a mobile OS with traditional access control mechanisms can offer the requisite security; the role of context is to make these mechanisms convenient and light-weight, which is especially important for mobile.

Scheduling: The scheduler manages the processor to ensure system responsiveness, fairness, throughput, and other key performance metrics. Most practical operating systems employ priority based preemptive schedulers where design parameters such as thread *priority* and *time quantum size* directly influence performance metrics. Mobile device OSs, including iOS and WP7, assign high priority to the user's foreground process, and often starve all other apps. This design helps improve device battery life and responsiveness of the foreground application but restricts application functionality and resultant user experience.

Context information can help overcome this restrictive design choice while limiting the impact on battery life and responsiveness. We suggest that context can directly influence process priorities and time quantum sizes. For instance, the activity *riding a bus* may increase the priority of background tasks such as download of social network updates and news – even in the presence of a foreground app – because the use of those background tasks is associated with this context. Location context, such as the user's presence at a weather-affected airport, may reduce the time quantum assigned to a foreground gaming app to make time for a flight status update app that has increased in priority at this location.

To enable context-aware scheduling, an application in CondOS may specify a priority level for each context in which it wishes to run. In lieu of distinguishing priority based on foreground or background status, the application moves among multiple priority queues as context changes. If time quanta get assigned to a priority queue in a certain context, the applications in that queue will receive background execution opportunities, improving functionality and user experience. These applications remain starved in other contexts, thus limiting their battery drain.

Energy Management: Critical battery outages cause headaches for mobile device users. Energy management is a key function of the mobile OS and can be assisted by context. Unfortunately, state of the art mobile OSs fail to avert low battery situations because the OS is not capable of foreseeing recharge opportunities.

Context can assist energy budgeting by predicting time-to-recharge. For example, the logical location *mall* or the activity *riding a bus* may hint that the next recharge may be some time off, whereas the logical location *home* may indicate liberal recharge opportunities. In response, the OS can shed workload. Examples include throttling tethering bandwidth and new email polling, and deferring network activities that are delay tolerant, such as uploading photos to social networking sites.

I/O: Phones are used in such a wide array of environments which makes it difficult to create a one-size-fits-all I/O solution. For example, the phone's basic alert mechanism, ringing, can easily be too loud in a meeting, or too soft at a party. Context can help by capturing and processing audio from the mic to determine the appropriate ring volume. Similarly, voice-based input may be most appropriate while driving, but not useful in a noisy bar. The displayed font size and touch screen keyboard could also be enlarged or reduced based on whether the user is sitting or walking. Network I/O responding to context is also intriguing. As an example, previous work has shown that optimal Wi-Fi and Bluetooth scanning periods and protocol window sizes depend upon IMU sensors distilling motion context such as *sitting* vs. *walking* [13].

2.2 Enhanced Sensor Privacy

Raw sensor data is powerful and can put users' privacy in jeopardy. Imagine a benign but buggy app that accidentally leaks your geolocation (lat/long) coordinates to the web, when it only intended to calculate whether you were at home or at work. In general, state of the art mobile OSs force the user to trust the app completely with raw data, even though the intention may be that the app only manipulates the sensor data to calculate and respond to a very narrowly-defined piece of context.

CondOS is well positioned to advance privacy-preserving sensor data processing. By delegating processing to CondOS, apps no longer touch raw sensor data. Nefarious and benign yet buggy apps can only receive CDUs at the kernel boundary.

Privacy authorization can also be of finer granularity in CondOS. At installation time, each mobile app might present a declaration of the context it desires, and the user can authorize each CDU type explicitly. If an app requests to install its own custom CDU Generator, then CondOS should fall back to asking the user for raw sensor data authorization on behalf of the app, since custom CDU Generators can disguise raw data through their CDUs. Similarly, apps that must have raw data (such as maps that use lat/long) can be serviced through custom CDU Generators. These measures enable higher resolution privacy control than the all-or-nothing sharing available today.

Moreover, CondOS could display a *CDU ticker*, an ambient display widget like a stock ticker that displays the latest CDUs generated [6]. The user can then visually inspect in-flight CDUs. This technique works well when the domain of the CDU is small (e.g. discrete text-based labels) and the domain of the raw data is large (e.g. continuous multi-dimensional signals), which is often the case. The CDU ticker is in the OS domain so the app has no way to manipulate it directly.

With CondOS, users need only trust the CDU conversion process, rather than the good intentions of a bug-free app. Moreover, users retain the opportunity to inspect the CDU ticker. As an alternative, taint tracking can also prevent leakage of raw sensor data [4], but its overhead is much more palpable, and its propagation of taints to CDUs may report high false positives. Our overhead is only that of regular kernel boundary protection.

The above mechanism can also be used to guard privacy when sharing sensor data among users. Imagine applications which wish to achieve wide- or local-area sensing goals. As an example, Alice may want to know which of her friends is partaking in an interesting social event. Audio signals from their phones can indicate which social event is the most lively. It is possible to ask for each phone's audio signal but this potentially intrudes on her friends' privacy. Instead, the CondOS approach calls for Alice to send her friends the *party on* CDU Generator, which – provided they trust her intentions – they can install and run locally on their phones. If *party on* is already installed, then each friend only needs to decide whether to share the CDU. Alice only gets to see the exported CDUs, and not the raw audio data. Furthermore, a sharing friend can inspect the generated CDUs and apply discretion before sharing.

Sharing CDUs and CDU processors rather than raw sensor data may have other advantages besides privacy. Since the domain of the former is far smaller than that of the latter, big compression and communication savings might be achieved. At the same time, the operating system can attest to the integrity of CDUs by extending trusted sensor data techniques [12] to trusted CDUs.

3 Toward a CondOS Design

Two challenges motivate the CondOS design. One challenge is to support a diverse and extensible CDU vocabulary while minimizing redundant CDU processing.

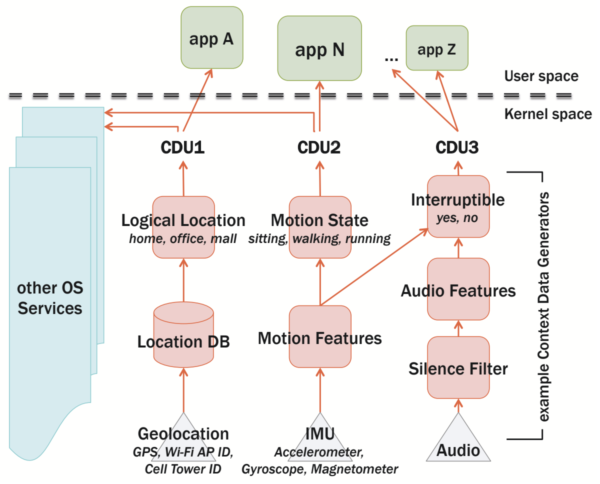

Figure 2: Example context dataflow installed in CondOS

A running system might have many concrete CDUs defined (possibly in proportion to the number of apps installed), yet CDU Generators will likely share much basic processing commonality. Prototypical cases of redundancy arise from popular signal processing subprocedures (e.g. mel-frequency cepstrum coefficient computations are widely used in audio processing), and hierarchically organized CDUs (e.g. sound context's *music* is refined by music context's *genre*). A second challenge is to guarantee system performance even while running app-defined CDU Generators. Predictability gives the OS a much better chance of anticipating and possibly avoiding low-energy or overload situations. Mobile OSs that fail to do this run the risk of failing users at critical junctures.

3.1 Design Proposal

We outline a design for CondOS that addresses the above challenges. Each CDU Generator consists of a directed acyclic graph of *components* connected to each other by producer-consumer interfaces. The head component(s) consume(s) data from one or more sensors and the tail component produces the CDU Generator's CDUs. Intermediate producer-consumer linkages define their own intermediate data types.

CDU Generators are managed by the OS's CDU Manager. The CDU Manager aggregates all CDU Generators into a composite dataflow, which is inspired by the Click Modular Router design [7]. Figure 2 illustrates an example dataflow with three CDU Generators installed: a Logical Location Generator that converts geolocation signals into logical locations such as *home* and *office*; a Motion State Generator that converts inertial movement readings into *sitting*, *walking*, etc., and; an Interruptible Generator that interprets audio and motion data to determine whether the user is interruptible.

Apps and OS services can make one time CDU requests, or subscribe to CDUs by simply wiring to the appropriate CDU Generator output. For example, a newer version of an OS service, such as the memory manager, may start to take advantage of the Interruptible Processor's CDU by simply wiring to it.

CondOS imposes constraints on dataflow components in order to maintain predictable CDU processing. Components are non-recursive, and may only be wired in a fixed number of producer/consumer relationships. Components are also sandboxed (e.g. no network communication) except at their input and output interfaces. Prior work [8] has shown that these constraints are sufficient for statically determining a performance profile such as memory, timing and energy usage at the time of CDU Generator installation.

Dataflow processing is activated in response to three types of events. First, an app or an OS service can explicitly ask for a CDU. Classic trade-offs between eager and lazy evaluation can be considered. Second, continuous CDUs require periodic maintenance. For example, *sitting* and *standing* may be hard to differentiate without periodic checks to identify *standing up* or *sitting down* motions. Lastly, CDU Manager may opportunisticly process a CDU if a subgraph of its CDU Generator graph has been otherwise activated for processing. The precise thresholds for piggyback processing may be set based on the cost savings expected from the performance profile, and the benefit expected from the CDU's activation frequency.

3.2 Energy Reduction Opportunities

Shared Dataflow Processing: Sharing both raw sensor data and intermediate results is a natural consequence of the dataflow architecture. Figure 2 shows an example of sharing occurring for the *Motion Features* component, which may be calculating an FFT on the raw accelerometer signal for both *CDU2* and *CDU3*. As new CDU Generators are installed, CDU Manager merges new graph components into the existing dataflow accordingly. As another example of sharing, a single app's interest in logical location may not warrant an expensive but accurate GPS request, but multiple apps that can all share in the benefits might. CondOS's challenge is to manage synchronization of these sharing opportunities.

Dataflow to Hardware Mapping: With the emergence of coprocessors and GPGPUs, mobile processor hardware is increasing in diversity. These platform variations cause headaches for app developers. CondOS's install-time static analysis provides an opportunity to map dataflow components to the user's specific phone hardware. For example, highly parallel audio and video processing routines may suit multiple cores [2], while highly geometric IMU calculations may be appropriate for GPGPUs. In particular, mapping dataflow to available hardware may be especially important for CDUs requiring continuous sensing. For example, although accelerometers nominally consume sub-milliwatts in power, reading data from them requires the main processor, which uses hundreds of milliwatts. Recent work has proposed using a dedicated low power sensing processor to offload sensing [10]. The sensing processor can accumulate data and apply simple filters. Only threshold events will activate the main processor for further processing. CondOS's dataflow components could provide a good level of granularity for such mapping to hardware.

Principled Flow Degradation: In CondOS, apps delegate handling of energy concerns during context generation to the OS. CondOS might efficiently service low energy situations by paring back CDU generation in a principled way that ultimately maintains the perception of an uninterrupted CDU stream. Degradation strategies include using cached or extrapolated results, throttling sensor sample-rates, and using variable fidelity CDU Generators that degrade context detection accuracy in return for energy savings. An example of the latter is [9]. CondOS's advantage is that predicting the trade-off from degradation of individual components may be feasible with dataflow static analysis.

4 Discussion and Conclusion

CondOS may provide the mechanism for OS services to subscribe to context, but it intentionally does not dictate the policy decisions resulting from context changes. One way to encode context-to-decision mappings is with a rule-based system, like a firewall's policy table. An example rule might be: *if the user is at home, the security subsystem should disable the password lock.* Another option is to use a small embedded machine learner to automatically extract mappings. The embedded learner may be more suitable for adapting to users' preferences provided there is a clear feedback loop for observing preferences. For example, in the context *standing*, the memory manager might learn over time to prefer preloading games for some users, and ebook apps for others since it is easy to identify what the user actually loads. Similarly, it could also learn that some context is simply not accurate enough to usefully predict a user's behavior, and adapt accordingly to safe defaults.

The context vocabulary will doubtless play an integral role to the utility of CondOS. We suspect that, like network protocols, a few CDUs will emerge as consensus defaults for inclusion in all mobile OSs. Since we are at the early stages of understanding what context will ultimately prove most useful, CondOS aims to make supporting any app- or OS service-defined context type easy.

Context extracted from raw sensor data has significant, unrealized potential to improve the entire mobile experience: from apps to the OS. The idea that the OS ought to manage (even app-specific) context generation at first appears counter-intuitive. Yet upon closer inspection we found several advantages to this approach, namely: better support for the OS itself to consume and benefit from context, better sensor privacy, and better centralized energy reduction opportunities. The design of CondOS, a context dataflow operating system, is a step toward realizing these benefits.

References

[1] Bamberg, S. J. M., Benbasat, A. Y., Scarborough, D. M., Krebs, D. E., and Paradiso, J. A. Gait analysis using a shoe-integrated wireless sensor system. In *IEEE Trans. on IT in Biomedicine* (2008).

[2] Bay, H., Ess, A., Tuytelaars, T., and Van Gool, L. Speeded-up robust features (surf). *Comput. Vis. Image Underst. 110*, 3 (2008), 346–359.

[3] Bishop, C. M. *Pattern Recognition and Machine Learning (Information Science and Statistics).* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[4] Enck, W., Gilbert, P., gon Chun, B., Cox, L., Jung, J., McDaniel, P., and Sheth, A. N. TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. In *Proceedings of OSDI* (2010).

[5] Fogarty, J., Hudson, S. E., Atkeson, C. G., Avrahami, D., Forlizzi, J., Kiesler, S., Lee, J. C., and Yang, J. Predicting human interruptibility with sensors. *ACM Trans. Comput.-Hum. Interact. 12* (March 2005), 119–146.

[6] Howell, J., and Schechter, S. What you see is what they get: Protecting users from unwanted use of microphones, cameras, and other sensors. In *Web 2.0 Security and Privacy, IEEE* (2010).

[7] Kohler, E., Morris, R., Chen, B., Jannotti, J., and Kaashoek, M. F. The click modular router. *ACM Transactions on Computer Systems 18*, 3 (August 2000), 263–297.

[8] Lee, E. A., and Messerschmitt, D. G. Synchronous data flow. *Proceedings of the IEEE 75* (Sep 1987), 1235–1245.

[9] Lin, K., Kansal, A., Lymberopoulos, D., and Zhao, F. Energy-accuracy trade-off for continuous mobile device location. In *MobiSys '10* (2010), pp. 285–298.

[10] Priyantha, N., Lymberopoulos, D., and Liu, J. Eers: Energy efficient responsive sleeping on mobile phones. In *PhoneSense* (2010).

[11] Ravi, N., D, N., Mysore, P., and Littman, M. L. Activity recognition from accelerometer data. In *IAAI '05* (2005), AAAI Press, pp. 1541–1546.

[12] Saroiu, S., and Wolman, A. I am a sensor, and i approve this message. In *HotMobile '10* (2010), pp. 37–42.

[13] Sivalingam, L. R., Newport, C., Balakrishnan, H., and Madden, S. "Extra-Sensory Perception" for Wireless Networks. In *HotNets-IX* (Monterey, CA, October 2010).