Design Principles for Contactile Computing

Philip Machanick
Department of Computer Science
Rhodes University, Grahamstown
6140, South Africa
p.machanick@ru.ac.za

ABSTRACT

Contactile computing is a proposed new paradigm in which sensors and communications are integrated into a system model that integrates real-time support with conventional application support. The motivation for the model is the growth of mobile devices with varying degrees of sensor capability and varying levels of processing power. For smart phones with various sensors, a sophisticated operating system is warranted by the complexity of the resources being managed. At the same time, existing operating systems tend to be strong either in real-time support, or in conventional application support, not both. This paper proposes a preliminary design for a contactile system, based on insights from microkernel systems like L4ka and Minix 3, which have illustrated how designs with dissimilar layers may be pieced together.

Categories and Subject Descriptors

D.4.7 [**OPERATING SYSTEMS**]: Real-time systems and embedded systems—*mobile computing, highly interactive computing*

General Terms

Human Factors

Keywords

sensors, mobile computing, real-time system, human-computer interaction

1. INTRODUCTION

A fly performs amazing avionic feats with only a few hundred neurons, whereas a state-of-the-art military supersonic aircraft uses over a million lines of code to approximate this behaviour. How does a fly do it? A clue is in the fact that it has 80,000 sensors [16]. Meanwhile the conventional computer world is stagnating: Moore's Law is providing us with increasing challenges in doing something useful with the increasingly cheap performance of commodity processors. Challenges include the memory wall (the growing CPU-DRAM speed gap), using multiprocessor architectures and overcoming network latency.

One may argue that innovation in the conventional computing market is stagnating but mobile devices such as phones and games are proliferating with interesting designs like the Nintendo Wii, which uses an accelerometer in a hand-held device to implement a user interface which interprets movement, and smart phones with a range of interesting capabilities. However, each of these devices is designed for a narrow purpose and hence to a narrow vision. What is missing is a broader view of the kind of platform which can be used to implement not only these features, but a wider array of sensor-enabled applications in a device with a variety of communication interfaces.

All of these issues suggest exploring new approaches to system design. This paper presents a starting point for designing a mobile, sensor-rich platform. To give the idea a name, the name *contactile computing* is introduced. The word contactile is meant to convey a sense of tactile (perceptible by touch) and contact (interfaces to the outside world). The intended device will have senses other than touch, but other potential names like *sensory computing* [2] have already been taken.

Some of the ideas here are not new: real-time kernels have been around for a long time, and microkernel operating systems, after a hiatus, are back. However, the growth of hardware sophistication available in small-scale portable devices suggests re-examing architectural decisions of the past. Can, for example, the concepts of device drivers and real-time applications be merged, given they have similarities? Can the concept of conventional application-layer support be layered on top of a real-time kernel, rather than implementing a messy compromise to support both? A sensor-rich mobile platform is not the only case where some of these questions may be relevant, but provides a concrete example to focus design trade-offs.

The remainder of this paper is structured as follows. Section 2 reviews related work. Section 3 discusses how a contactile platform could be designed, and section 4 illustrates how such a platform could be used. The paper wraps up with conclusions.

2. BACKGROUND

Related work includes context-aware computing, smart phones, special-purpose devices and small-scale sensor devices in sensor networks. Sensor-enabled devices also have a following in the field of artificial intelligence. Each of these areas in considered in turn, noting any overlaps.

2.1 Context-Aware Computing

In context-aware computing, devices adapt their behaviour to external stimuli. Context-aware computing is

SAICSIT '11, October 3-5, Cape Town, South Africa

similar to contactile computing in responding to external stimuli. It differs in that contactile computing places reaction to sensors at the centre of system design.

Context-aware computing can include smart environments, in which objects in the environment can interact with a mobile device [3], an idea further discussed below as related to smart phones. The general idea is that the environment has devices with some logic, memory and communication capability.

Earlier work on context-aware computing has assumed communications protocols would be used to convey context [10]. This is by contrast with the contactile idea, where sensors would provide a significant amount of context information. More recent work on context awareness has included sensor data such as visual information [9]. The contactile idea takes this further in the form of a more general platform that can apply to the traditional concept of context-aware computing.

There has been significant work on infrastructure for context-aware computing including middleware [5, 7], illustrating the general maturity of the field. This project assumes this broader infrastructure work will continue, but is not essential to the deliverables.

2.2 Smart Phones

Smart phones are evolving to include some functionality that could form part of a contactile platform, such as cameras, touch screens, rotation sensing and a proximity sensors. They may also includes various modes of communication including wireless internet. Mobile phones are implemented with a variety of operating systems, including specialist systems like Symbian and versions of full-scale operating systems such as Linux and Windows [14], or platforms based on UNIX-like underpinnings such as Apple's iOs [1] and Google's Android. Such devices, while illustrating what can be put into a small package, are constrained by being designed primarily as phones with extra features, rather than as a general-purpose sensor-enabled platform (even if devices like the iPad are based on a similar platform).

Growing sophistication of phones can also be matched by smart environments, with which phones interact [11]. This latter idea is somewhat different to the contactile idea in that the environment contains devices that use a mobile phone as a tool for communicating with a user; the contactile idea is based on sensing the environment (more intelligence in the device, less in the environment).

2.3 Sensor Networks

There has been significant work on infrastructure for sensor networks, including middleware [6] – with some overlaps with work on context-aware computing. However, the emphasis in sensor networks is on smaller-scale devices, with most work focused on low power approaches to classic problems such as network routing [15] and processor design [17].

Sensor networks are based on largely immobile sensors that may be in the environment for a long time without the opportunity to charge of replace batteries, and therefore do not aim at the same niche as contactile computing, a mobile device of significant computing power.

2.4 Related Operating Systems

The L4ka microkernel has been used as a basis for a mobile phone platform, in which the microkernel handles security-sensitive functionality, and a version of Linux runs on top of the microkernel, to provide conventional application services [8]. The L4ka approach is a useful example of how to integrate conventional applications with a specialist kernel, but L4ka has no native application layer.

Real-time extensions exist in general-purpose operating systems such as Linux [12]. However, the assumption underlying such extensions is that the real-time activities will be a small fraction of the total workload, and so the overall system is not optimised to real-time requirements.

There are free real-time kernels, such as FreeRTOS[1], which are oriented towards pure real-time applications, i.e., without support for conventional applications.

Other real-time kernels are generally proprietary (e.g., QNX Neutrino RTOS[2]), and hence not suitable platforms for research. LynxOS-SE from LynuxWorks[3] is reasonably close to the required functionality in allowing virtual machines to run in different modes (native POSIX- compliant, Linux compatible binaries or native hard real time). However, a large fraction of the system is proprietary, and does not permit exploration of new models of system implementation, such as implementing device drivers as a special case of a real-time application.

2.5 Summary

There is a wide variety of work on sensor devices, both as lightweight networks of sensors, and sensors on larger devices. The growth of the sophisticated mobile devices in the form of smart phones and PDAs has created a niche large enough to justify a general-purpose platform tailored to this niche. The growing use of sensors on such devices suggests that a real-time platform designed to integrate sensors into a general programming model would suit a wide variety of applications and devices.

3. DESIGN PRINCIPLES

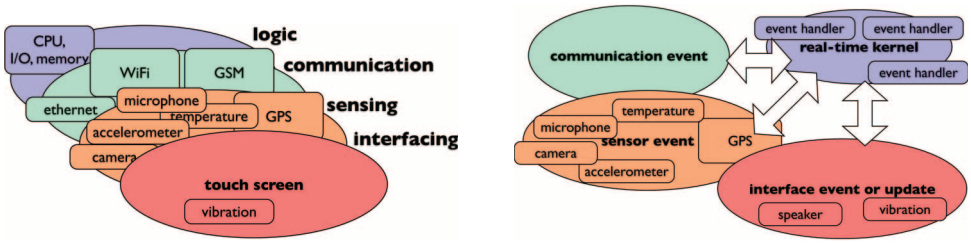
The major idea of a contactile device is to provide strong operating system support for sensors, some of which may have a hard real-time aspect, i.e., if they are not responded to in time, the system is broken. At the same time, as a general platform, it should provide strong support for a conventional application layer, one designed to maximise leverage from existing tool chains and popular applications. The latter is an important pragmatic principle – if writing applications is too hard, no one will develop for the platform, unless it is a very strong fit to a niche.

Further, the real-time aspect should not be a totally separate concept from normal application space, and there should be some flexibility in deciding whether to situate specific functionality in the real-time or the conventional layer. For example, telephony is a real-time application, but the most time-critical elements arise from delays on the network, and the occasional loss of information is tolerable, so adapting existing voice over IP solutions to the application layer would be a reasonable choice – they already exist, and work well enough. On the other hand, for a small, hand-held low-power device, it may make more sense to implement telephony on the real-time side, so a slower processor would achieve acceptable results.

To allow this kind of design flexibility, it should be possible to invoke real-time functionality from the "conventional" layer and vice-versa. For this reason, the system should support communication modes that cross the layers; the most obvious is a messaging primitive or interprocess communication (IPC).

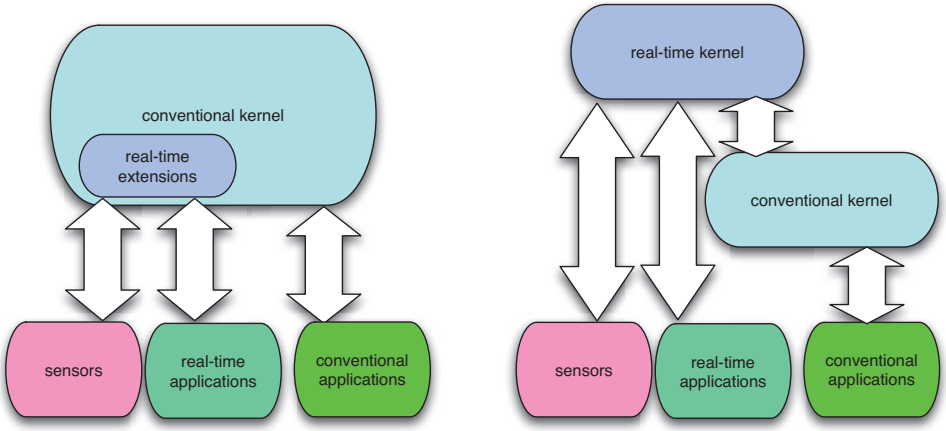
Figure 1 illustrates a conceptual design for a contactile device. The software architecture (1(b)) follows closely

[1] http://www.freertos.org/ [2] http://www.qnx.com [3] http://www.lynuxworks.com/



(a) Hardware (b) Software

Figure 1: Conceptual Design. ***Sensors and Interface are possibilities: specific options should form part of a detailed design.***



(a) Conventional (b) Contactile

Figure 2: Conventional versus Contactile Design. ***L4ka-Linux is closer to the contactile design but doesn't support inter-layer applications.***

from the hardware architecture (1(a)). Arrows in the software layer correspond to IPC information flows. The real-time and conventional kernels are aware of each other and communicate as shown. A consequence of this design is that device drivers may be put in the real-time layer, and hence have a consistent semantics for interacting with the rest of the system. As with the Minix 3 kernel, all the major subsystems can in principle be isolated from each other for reliability [4]. Also following Minix 3 design principles, any real-time response can be divided into that which has to be close to the hardware, and an application layer.

Minix 3 approaches system calls by dividing them into a user-level API, implemented as servers invoked via IPC or messages through the kernel, and kernel calls, which invoke functions only the kernel can perform. This division applied to real-time results in the following:

- kernel layer – low-level implementation of hardware-specific aspects of a device driver
- service layer – the rest of the real-time component of the application
- application layer – part of the application that does not require real-time guarantees (e.g., user interface)

As with Minix 3 device drivers, the real-time kernel layer would be isolated from other parts of the system, and would therefore be limited to interfacing to the hardware related to the real-time service.

Figure 2 illustrates how the proposed architecture for a contactile device differs from a conventional operating system designed to run standard applications, with a real-time aspect. The conventional operating system (2(a)) includes real-time support within a conventional kernel, where it must introduce compromises, versus the contactile model, where real-time is at the bottom layer (2(b)). Although the layers are separate, the contactile model allows inter-layer communication. Data flows are routed through the conventional kernel, to maintain partitioning.

What of the programming model? For the conventional layer could be similar to the Minix 3 kernel, with applications isolated from each other, and system functions implemented in servers outside the kernel. The big change proposed is having a separate real-time kernel, which becomes a base not only for new real-time functionality, but also creating new device drivers. This real-time kernel would have a subset of the rights of the conventional kernel, but would get the first bite at handling interrupts.

4. CASE STUDY

To put it all together, a case study illustrating the combination of features required for a contactile platform is useful. The example chosen includes a need for real-time responses to sensors, and conventional applications: a smart phone for emergency services.

The specification of the phone is illustrated in Figure 3. Some of the features (internet access, note pad) would be most easily implemented on a UNIX-like platform, using standard tools and technologies. Voice over IP can also be implemented on a UNIX-like platform (the real-time requirements are not stringent enough to be a problem

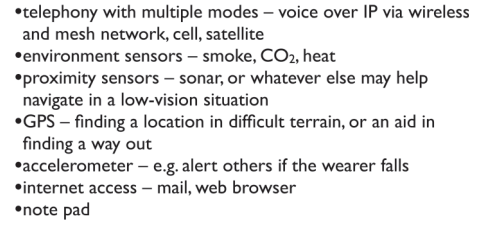


Figure 3: Basic Phone Specification. ***There could be more features; this list illustrates the principles.***

except on a heavily loaded machine). The GPS functionality too can be implemented in a standard application layer. Safety-related sensors on the other hand are best implemented in a hard real-time system (even if the latency requirements are not challenging) because missing a measurement could be life-threatening.

For this example, implementation on a contactile platform would include a "conventional" application layer supporting telephony, GPS, internet access and other standard kinds of application, like text editors. Safety-critical sensors would be implemented in the real-time layer. It is also possible to mix the functionality of the layers. For example, an effect of a CO_2 sensor exceeding a threshold could be to contact another member of the team by automatically initiating a phone call; the phone call would be in the "conventional" layer.

Why is a conventional operating system not ideal in this scenario? A non-real-time kernel such as Linux supports a conventional application layer. However, rapid response to external events requiring a hard real-time scheduler is difficult to retrofit onto a kernel not designed that way. On the other hand, a pure real-time kernel like FreeRTOS lacks the functionality and mature tool chains of a UNIX-like system for conventional applications.

5. CONCLUSIONS

The case study is but one example of a contactile computing application. It illustrates the essential problem: the need for a platform specific to computing with a significant number of sensors integrated into the user experience.

While other designs, like L4ka with Linux on top, illustrate the general possibilities, none actually implement the design proposed here. The L4ka-Linux combination works as two independent systems; L4ka passes all Linux system calls straight back to the Linux kernel, and the Linux kernel is not aware of the underlying microkernel. The contactile approach, on the other hand, requires that the layers be aware of each other, to allow implementation of services that combine functionalities.

The growth of and growing convergence between mobile devices and sensors provides support for developing a platform of this kind.

6. ACKNOWLEDGMENTS

This work was undertaken in the Distributed Multimedia CoE at Rhodes University, with financial support from Telkom SA, Comverse, Stortech, Tellabs, Eastel, Bright Ideas 39 and THRIP.

7. REFERENCES

[1] Apple Computer. *iOS Technology Overview*. Apple Computer, 2010.

[2] V. Brajovic and T. Kanade. Sensory computing. In *SPIE Proc. Vol. 4109, Critical Tech. for the Future of Computing*, San Diego, CA, 30 July–4 Aug. 2000.

[3] D. L. de Ipiña, I. Vázquez, D. García, J. Fernández, and I. García. A reflective middleware for controlling smart objects from mobile devices. In *sOc-EUSAI '05: Proc. 2005 joint conference on Smart objects and ambient intelligence*, pages 213–218, New York, NY, USA, 2005. ACM Press.

[4] C. Giuffrida, L. Cavallaro, and A. S. Tanenbaum. We crashed. now what? In *Proc. 6th Workshop on Hot Topics in System Dependability*, October 2010.

[5] T. Gu, H. K. Pung, and D. Q. Zhang. A middleware for building context-aware mobile services. In *Proc. 59th IEEE Vehicular Technology Conference VTC 2004*, volume 5, pages 2656–2660, May 2004.

[6] S. Hadim and N. Mohamed. Middleware: Middleware challenges and approaches for wireless sensor networks. *IEEE Distributed Systems Online*, 07(3), 2006.

[7] K. Henricksen, J. Indulska, T. McFadden, and S. Balasubramaniam. Middleware for distributed context-aware systems. In *Proc. Int. Symp. on Distributed Objects and Applications (DOA)*, pages 846–863. Springer, 2005.

[8] S.-M. Lee, C.-S. Nam, and T.-Y. L. andDong Ryeol Shin. The L4 microkernel based mobile middleware for the multiple virtual machines. In *Proc. Int. Conf. New Trends in Information and Service Science, 2009. NISS '09.*, pages 456–459, Beijing, 30 June–2 July 2009.

[9] P. M. Luley, L. Paletta, and A. Almer. Visual object detection from mobile phone imagery for context awareness. In *MobileHCI '05: Proc. 7th int. conf. on Human computer interaction with mobile devices & services*, pages 385–386, 2005.

[10] A. Schmidt and H. W. Gellersen. Context-aware mobile telephony. *SIGGROUP Bull.*, 22(1):19–21, 2001.

[11] F. Siegemund, C. Floerkemeier, and H. Vogt. The value of handhelds in smart environments. *Personal Ubiquitous Comput.*, 9(2):69–80, 2005.

[12] W. von Hagen. Real-time and performance improvements for the 2.6 Linux kernel. *Linux J.*, 2005(134):8, 2005.

[13] W. Wang, V. Srinivasan, and K.-C. Chua. Using mobile relays to prolong the lifetime of wireless sensor networks. In *MobiCom '05: Proc. 11th annual int. conf. on Mobile Computing and Networking*, pages 270–283, 2005.

[14] B. Weinberg. Mobile phones: the embedded Linux challenge. *Linux J.*, 2006(148):9, 2006.

[15] G. Xing, C. Lu, R. Pless, and Q. Huang. On greedy geographic routing algorithms in sensing-covered networks. In *MobiHoc '04: Proc. 5th ACM int. symp. on Mobile ad hoc networking and computing*, pages 31–42, 2004.

[16] R. Żbikowski. Fly like a fly. *IEEE Spectr.*, 42(11):46–51, 2005.

[17] B. Zhai, S. Pant, L. Nazhandali, S. Hanson, J. Olson, A. Reeves, M. Minuth, R. Helfand, T. Austin, D. Sylvesterand, and D. Blaauw. Energy-efficient subthreshold processor design. *IEEE Trans. VLSI Systems*, 17(8):1127–1137, Aug 2009.