

Operating System Issues in Wireless Ad-Hoc Networks

Invited paper

Kimmo E. E. Raatikainen, *Member, IEEE*

Abstract—Traditional monolithic operating systems have conceptually remained almost unchanged since the UNIX, that is, since the late 70s. Several experimental operating systems from the research community have been based on alternative paradigms. Today we are facing the dawn of mobile or wireless Internet. This new operational environment calls for new solutions. A paradigm shift in operating system design, as demonstrated by some experimental operating systems, can help us to lay the software foundation for reconfigurable end-user systems.

Index Terms—mobile computing, operating systems, wireless communication.

I. INTRODUCTION

OPERATING systems have been in the core of computer science from the very beginning. Therefore, one can ask is there some issues still open. If the answer is yes then one can ask why fifty years have not been enough in solving all relevant issues. Clearly operating systems have evolved a lot during the years. However, changes in operational requirements have changed so that today we need to reconsider even the fundamentals of operating systems.

The need of this reconsideration has its roots in the fundamental changes in usage patterns. Communication and computing devices move; users move and change their devices; (sub)networks in cars, trains and airplanes move; software moves from one execution environment to another.

In this paper we discuss some research challenges in operating systems for future wireless ad-hoc systems. We start by stating our assumptions about future mobile applications. In Section III we take a look at some milestones in operating systems. In Section IV we discuss the architectural challenge arising from the fundamental requirement of holistic solutions. In Section V we elaborate some fundamental research issues for future generations of operating systems.

Kimmo Raatikainen is a professor at the Helsinki University Computer Science Department, P.O. Box 68 (Gustaf Hällströmin katu 2b), FIN-00014 UNIVERSITY OF HELSINKI, Finland (kimmo.raatikainen@cs.helsinki.fi). He also works as a part-time research fellow at Nokia Research Center, P.O. Box 407 (Itämerenkatu 11-13), FIN-00045 NOKIA GROUP, Finland (kimmo.raatikainen@nokia.com) and a part-time principal scientist at Helsinki Institute for Information Technology, P.O. Box 9800 (Tamma-saarenkatu 3), FIN-02015 TKK, Finland (kimmo.raatikainen@hiit.fi).

II. FUNCTIONALITY IN FUTURE MOBILE SYSTEMS

In reconsideration we have taken into account various mission papers [1]-[7]. These visions can be summarized as follows: *Future applications will be context-sensitive, adaptive and personalized, and the systems will be reconfigurable.*

Until recent years, the computing and communication have been mainly driven by technology. Engineers have developed technologies that consumers have figured out how to use. It has worked, we have got positive surprises, but this path is close to an end. We believe that in the next ten years we must move from gadget (and technology) centricity to user-centricity. Producers want to be sure that their products will sell well enough.

The studies of end-user expectations set the requirements of infrastructure research. However, this is not a one-way road. The feedback loop gives cost estimates: how expensive certain features are when deployed locally, nation-wide, world-wide. In the forthcoming years we need to take a radically new attitude in order to realize Mark Wiser's vision of ubiquitous computing [1]: the computing and communication is here but you do not need to bother.

In bringing the dream of invisible computing into reality for mass markets, that is for hundreds of millions of people, we still have a lot to do. In fact our claim is that we must go back to the fundamentals; to reconsider the foundations of mobile computing and communications. As stated in the Introduction, the need of this reconsideration has its roots in the fundamental changes in usage patterns. These properties have been examined by WWRF [7] in details. Below we briefly elaborate the functional requirements behind the concepts of reconfigurability. In essence reconfigurability means that system's hardware and software configurations can seamlessly change in run-time.

A personal trusted device will be the core of the personal networking system. It probes its surrounding looking for suitable peripheral devices such as displays, input devices, processors, fast access memories and access points to communication channels. It dynamically builds up the most appropriate end-user system that can be autoconfigured. The device also probes for other similar devices in order to establish suitable ad-hoc communities and different kinds of

sensors in order to extract context information associated to the current smart place. It also tries to detect actuators which provide the means to affect properties of the smart place.

The scenario above implies that the system is able to do device detection and service discovery, as well as hardware and software configuration management. Efficient device detection and service discovery require that the system is able to monitor the environment and to deliver notifications when something new appears, something old disappears, and something existing changes its state. A notification should only be delivered to those who are interested in it. This, in turn, implies that notifications need to be filtered.

When the state of the systems or its environment changes, then the system needs to decide whether or not the system needs to be reconfigured. This decision engine needs to have a model of its environment (external context) and of its own configuration and capabilities. In other words, the system needs to be self-aware (or reflective). The decision engine also needs a “target function” that indicate how preferable different configurations are to the user. In that sense the target function reflects the personal preferences of the user. It is impossible to assume that all the capabilities and preferences used by the decision engine are complete from the very beginning. Therefore, the models need to be learned, and the system must have learning capabilities.

When the system reconfigures itself, it must maintain its integrity. This requires protection against unauthorized modification. When the system includes a new piece of code into its execution base, it must trust that the code is neither malfunctioning nor introducing undesirable side-effects. An alternative is that the system can verify or validate the code. In addition, the system needs to trust, verify, or validate the pieces of information so that the models used by the decision engine are coherent and reliable. When the system moves into a new administrative domain, it needs to establish a trust relationship in that domain. The establishment needs, at least, mutual authentication, usually also some kind of authorization to use certain resources and services.

III. SOME MILESTONES IN OPERATING SYSTEMS

The structure and abstractions (see Fig. 1) in the traditional monolithic operating systems have not practically evolved since the Unix [8]. Other milestones in operating systems include THE [9], “Nucleus” [10], Multics [11], and Hydra [12]. Edsger Dijkstra’s recollections [13] and Fernando Corbato’s retrospective [14] give interesting information about early problems in operating systems.

In the late 1980s the research community introduced the microkernel approach that minimized the size of the kernel and implemented most of operating system services as servers outside the kernel. The first generation of microkernels, such as Amoeba [15], Mach [16], and V [17], introduced too much overhead to be more than academic exercises. However, they laid the foundation for future developments.

The *x-kernel* [18] from the University of Arizona, L4

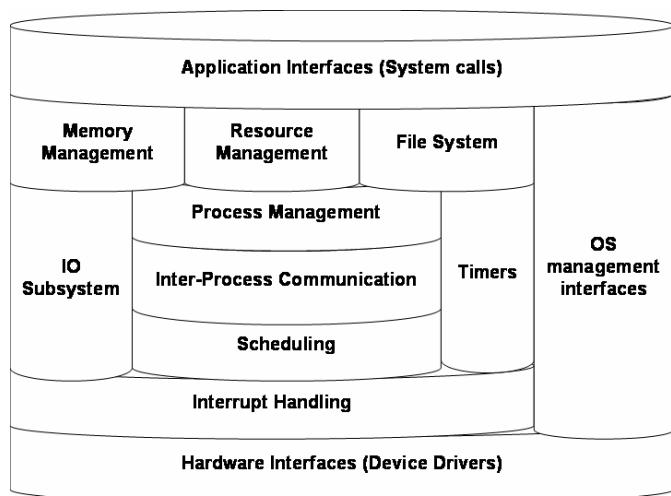


Fig. 1. Structure of traditional operating systems.

kernel [19] from GMD, *exokernel* [20] from MIT, and *TinyOS* [21] from UC Berkeley introduced interesting solutions and new concepts.

The *x-kernel* was designed to facilitate efficient implementation of communication protocols. It included components that manage processes, memory, and communication. The process and memory managers were quite similar to those in other operating systems. The novel aspect of the *x-kernel* was the communication manager that provides an object-oriented infrastructure for composing protocols.

The underlying design philosophy of *L4* is based on the claim that efficiency and flexibility require a minimal set of general microkernel abstractions and that microkernels are processor dependent. The main contribution of the L4 team was the fast message-based synchronous IPC.

A group at the MIT designed a new operating system architecture, called *exokernel*, which only securely multiplexes available hardware resources. All traditional operating system abstractions are implemented entirely at application level by untrusted software. The MIT team laid the foundation but the primary contribution later came from University of Cambridge. The *Nemesis* [22] and particularly the *Xen* [23] demonstrated that paravirtualization introduces only a small overhead, typically 2-4%.

TinyOS is a very small microthreaded operating system design for networks of sensor nodes (*SmartDust* [24]). The *TinyOS* is usually accompanied by *Maté* [25], a really minimalistic virtual machine. *TinyDB* [26] and the programming language *nesC* [27] belong to the same solution family. The “*Tiny solution*” family clearly indicates that a small and efficient operating system and development environment can be build. The primary lesson is similar to that from the *x-kernel*: Design the operating system for a specific target system. In addition, the *Maté* shows that a tiny virtual machine is useful for installing new applications.

The four examples presented above clearly demonstrate the advantages of a paradigm shift in operating system design. Other significant experiments in operating system research

include operating system frameworks (Flux OS Kit [28], Choices [29], Pebble [30]), portal-based operating systems (Kea [31], Space [32]), reflective operating systems (MetaOS [33]) and dynamically adaptive operating systems (Synthetix [34]). Additional interesting results have been in the area of software protection (Software Fault Isolation [35], Proof-Carrying Code [36]).

IV. ARCHITECTURE CHALLENGE

As from the new operational requirements: reconfigurability, context-awareness, adaptation, and personalization. The problems with the current plug-and-play are a clear indication that new approaches are needed. Our claim is that we need a paradigm shift: *forget the end-user terminal and start thinking about end-user systems!*

The fundamental challenge is that we must tackle reconfigurability and adaptation issues on hardware level and on all levels of software (operating system, protocol stack, middleware, applications). We must also remember that the system needs to be usable anywhere, anyhow, anytime, and by everybody. This implies that the end-user systems will also contain carry-on, battery-powered devices.

Future systems need a holistic approach in research and development. This is due to the fact that we need to address issues in the areas of hardware, software, and development tools. Therefore, architectures—blueprints of systems—are of fundamental importance when we try to construct a feasible solution for the future.

Architectures, however, are terribly overloaded words. According to Martin Folwer [37] “For many, the term ‘software architect’ fits perfectly with the smug, controlling image at the end of Matrix Reloaded.” Steve Vinoski [38] writes “Unfortunately, our old friend, the not-invented-here (NIH) syndrome, runs rampant throughout our industry. Sometimes, even when you put an architecture in place, managers and developers can still find ways to ignore it.”

Nevertheless, software architectures will be cornerstones in future mobile systems. They enable modular and independent development and define clear interfaces between different software components. There are two key challenges in software architecture:

- Architecture should not be too detailed preventing innovative solutions.
- Architecture should not be too loose creating interoperability nightmare.

OMG's Object Management Architecture (OMA) is a good example of balance that worked for years. However, the flat structure based on an object bus (ORB) has serious limitations to be applicable in future mobile world.

Service and software architectures are currently under wide development. Wireless World Research Forum (WWRF) has come up with an architectural reference model [39]. In addition, architecture related work has been carried out in the EC Wireless Strategic Initiative project [40] and in Open Mobile Alliance (OMA) [41]. In addition, there are regional

activities like mITF [42] in Japan. Moreover, industry has—examples include NTT DoKoMo [43] and Nokia [44]—contributed they proposals to various forums.

V. SOME RESEARCH CHALLENGES IN OPERATING SYSTEMS

As elaborated in the functional requirements for future mobile systems, the key enablers include:

- environment monitoring,
- device detection and service discovery,
- event notification and filtering,
- hardware and software configuration management
 - auto-configuration
- decisions engines
 - when and how to reconfigure
- modeling and learning capabilities
- maintaining system integrity

Clearly, all these issues are not solved on the operating system level alone. Support from hardware, protocol stack, and middleware is needed to for a feasible solution. Operating system support is needed in monitoring, detection, notifications, configuration management, and system integrity.

A. System Integrity

The success of the future Internet will totally depend on consumers' trust. The current Internet is vulnerable to worms, viruses, spam and fraud. It clearly demonstrates that “*fix it later*” does not work. Security, trust and privacy must be addressed from the very beginning of system design and on all levels: hardware, operating system, protocols, middleware.

Authenticated boot is a fundamental enabler for system integrity. Trusted Computing Group [45] has specified one way of providing the necessary chains of trust. In dynamically reconfigurable systems, chains of trust must be re-established each time the configuration changes. How this can be done efficiently enough is a research challenge. The re-establishment of the chains of trust is also needed each time a new piece of software is to be included. The Proof-Carrying Code [36] is a prominent way of verifying what the received piece of code will do. Alternatives include various sandboxing techniques.

Without sufficient hardware support, system integrity cannot be provided. For example, most game applications run their own logic on display processor. If the display processor is DMA-enabled, hardware support is necessary to prevent the application code from modifying internal data structures of operating system.

B. Power Management

In future systems, available energy, as embodied by the system battery, will have an increasingly important role. Despite wide-spread recognition of the importance of energy, current operating systems do not provide application developers a convenient abstraction of the energy resource. There have been broad efforts to better manage the energy use of individual devices, but there has been relatively little attention to managing energy as a resource.

In order to manage energy, the operating system must have a model for power consumption. There are two primary problems in addressing specific energy-related goals in operating system level. The first one is to develop resource management policies. The second one is related to adaptation in application behavior.

The Smart Battery interface in the ACPI specifications [46] is a fundamental enabler for operating system to manage the battery resource. It allows the system to query the status of the battery. However, the operation of querying the interface is too slow to be useful for gathering power consumption data at a sufficiently fine grain for resource management functions without introducing unacceptable overhead.

C. Other Issues

In order to be reconfigurable, the system needs to be self-aware. In other words, the system needs to understand its own state; the operating system must understand its own state and configuration, and the state and the configuration of the resources it is controlling and managing.

The understanding requires a model of hardware and operating system level software configuration. This model, or these models, will be used to describe the configurations and to reason about the configurations.

Operating systems have such models, but their use is limited to operating system's internal bookkeeping. Particularly in reasoning, the operating system needs to pass the model to other software.

In order to reconfigure a system, the system needs to detect new devices/subsystems and services that have come available. The system needs also to detect the subsystems or services that have disappeared, or whose properties have significantly changed.

Issues in this area include interrupt handling and event notifications. We need also revisit many optimization techniques currently used in operating systems. For example, lazy update should not be used if we must take into account that the persistent storage subsystem may disappear.

Most of detection in the operating system level is done in interrupt handlers and device drivers. Interrupt handling is also affected by sensors. When a carry-on device provide motion detection, we will get interrupts very frequently. The traditional way of handling interrupts would consume too much CPU cycles.

VI. CONCLUSIONS

We have shown that a paradigm shift in operating system design is necessary to meet the needs of future end-user system. We proposed to take reconfigurability as the main concern. This will be a similar to the design principles in x-kernel [18] and TinyOS [24].

If we are not all the time ready for a revolution, we may miss the train and we may find ourselves at the trap of basing next releases of our products on existing legacy. We do not claim that today is the right time to forget all legacy systems. However, tomorrow it is even more costly to replace them.

We should ask ourselves whether or not we want to produce pullovers for dinosaurs although the climate has already started to cool and sooner or later the dinosaurs will disappear.

In order to support reconfigurability, the operating system research must solve several research issues related to self-awareness, detection and notifications, system integrity, and power management. The fundamental challenge is to find a reasonable balance of solutions in the areas of hardware, system software (operating system, protocol suite, middleware), and development tools.

REFERENCES

- [1] M. Weiser, "The Computer for the Twenty-First Century," *Scientific American*, September 1991, pp. 94-104.
- [2] M. Weiser, "Some Computer Science Issues in Ubiquitous Computing," *Communications of the ACM*, July 1993, pp. 74-84.
- [3] R. Bagrodia, W.W. Chu and L. Kleinrock, "Vision, Issues, and Architecture for Nomadic Computing," *IEEE Personal Communications*, December 1995, pp. 14-27.
- [4] M. Satyanarayanan, "Pervasive Computing: Vision and Challenges," *IEEE Personal Communications*, August 2001, pp. 10-17.
- [5] K. Ducatel et al., *Scenarios for Ambient Intelligence in 2010*. Technical report, ISTAG, February 2001.
- [6] G. Banavar, J. Barton, N. Davies and K. Raatikainen, "Special feature on middleware for mobile & pervasive computing," *ACM SIGMOBILE Mobile Computing and Communications Review*, October 2002, pp. 16-24.
- [7] R. Tafazolli, Ed., *Technologies for the Wireless Future*. Wiley, 2005.
- [8] D. M. Ritchie and K. Thompson, "The UNIX Time-Sharing System," *Communications of the ACM*, July 1974, pp. 365-375.
- [9] E. W. Dijkstra, "The Structure of the "THE" Multiprogramming System," *Communications of the ACM*, May 1968, pp. 341-346.
- [10] P. Brinch Hansen, "The Nucleus of a Multiprogramming System," *Communications of the ACM*, April 1970, pp. 238-250.
- [11] E. I. Organick, *The Multics System: An Examination of its Structure*. MIT Press, 1972.
- [12] W. Wulf et al., "Hydra: The kernel of a multiprocessing operating system," *Communications of the ACM*, July 1974, pp. 337-345.
- [13] E. W. Dijkstra, "My Recollections of Operating System Design," Manuscript EWD1303 (Oct. 2000/Apr. 2001). Reprinted in *ACM SIGOPS Operating Systems Review*, April 2005, pp. 4-40.
- [14] F. J. Corbató, "On building systems that will fail," *Communications of the ACM*, September 1991, pp. 72-81.
- [15] A. S. Tanenbam et al., "Amoeba System," *Communications of the ACM*, December 1990, pp. 46-63.
- [16] D. Golub, R. Dean, A. Forin and R. Rashid, "Unix as an application program," *Proceedings of the Usenix Summer Conference* (Anaheim, Calif., June 1990). Usenix Association, 1990, pp. 87-96.
- [17] D. R. Cheriton, G. R. Whitehead and E. W. Szynter, "Binary emulation of Unix using the V kernel," *Proceedings of the Usenix Summer Conference* (Anaheim, Calif., June 1990), pp. 73-86.
- [18] N. C. Hutchinson and L. L. Peterson, "The x-Kernel: An Architecture for Implementing Network Protocols," *IEEE Transactions on Software Engineering*, January 1991, pp. 64-76.
- [19] J. Liedtke, "On μ -Kernel Construction," *Proceedings of the 15th ACM Symposium on Operating System Principles*, December 1995, pp. 237-250.
- [20] D. R. Engler et al., "Exokernel: An Operating System Architecture for Application-Level Resource Management," *Proceedings of the 15th ACM Symposium on Operating System Principles*, December 1995, pp. 251-266.
- [21] J. Hill et al., "System Architecture Directions for Networked Sensors," *Proceedings of ASPLOS 2000*, ACM, November 2000, pp. 93-104.
- [22] D. Reed, A. Donnelly and R. Fairbairns, eds., "Nemesis Kernel Overview," <http://www.cl.cam.ac.uk/Research/SRG/netos/old-projects/pegasus/publications/overview/brief-overview.html>.
- [23] P. Barham et al., "Xen and the Art of Virtualization," *Proceedings of the 19th ACM Symposium on Operating System Principles*, October 2003, pp. 164-177.

- [24] B. Warneke, M. Last, B. Liebowitz and K. S. J. Pister, "Smart Dust: Communicating with a Cubic-Millimeter Computer," *IEEE Computer Magazine*, January 2001, pp. 44-51.
- [25] P. Levis and D. Culler, "Maté: A Tiny Virtual Machine for Sensor Networks," *Proceedings of ASPLOS X*, ACM, October 2002, pp. 85-95.
- [26] S. Madden, M. J. Franklin, J. M. Hellerstein and Wei Hong, "TAG: A Tiny AGgregation Service for Ad-Hoc Sensor Networks," *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, December 9-11, 2002.
- [27] D. Gay et al., "The nesC Language: A Holistic Approach to Networked Embedded Systems," *Proceedings of PLDI'03*, June 9-11, 2003.
- [28] B. Ford et al., "The flux OSKit: A substrate for kernel and language research," *Proceedings of the 16th ACM Symposium on Operating Systems Principles*, 1997, pp. 38-51.
- [29] R. Campbell et al., "Designing and implementing Choices: an object-oriented system in C++," *Communications of the ACM*, September 1993, pp. 117-126.
- [30] K. Magoutis et al., "Building appliances out of reusable components using pebble," *Proceedings of the 9th ACM SIGOPS European Workshop*, 2000.
- [31] A. Veitch and N. Hutchinson, "Kea—a dynamically extensible and configurable operating system kernel," *Proceedings of the 3rd Conference on Configurable Distributed Systems*, 1996.
- [32] D. Probert, J. Bruno and M. Karzaorman, "Space: a new approach to operating system abstraction," *Proceedings of the International Workshop on Object Orientation in Operating Systems*, 1991, pp. 133-137.
- [33] M Horie et al., "Using meta-interfaces to support secure dynamic system reconfiguration," *Proceedings of the 4th International Conference on Configurable Distributed Systems*, 1998.
- [34] C. Pu et al., "Streamlining a commercial operating system," *Proceedings of the 15th ACM Symposium on Operating System Principles*, 1995.
- [35] R. Wahbe et al., "Efficient software-based fault isolation," *ACM SIGOPS Operating Systems Review*, December 1993, pp. 203-216.
- [36] G. C. Necula and P. Lee, "Safe kernel extensions without run-time checking," *Proceedings of the 2nd Symposium on Operating Systems Design and Implementation*, 1996, pp. 229-243.
- [37] M. Fowler, "Who Needs an Architect?" *IEEE Software*, September/October 2003, pp. 11-13
- [38] S. Vinoski, "Do You Know Where Your Architecture Is?" *IEEE Internet Computing*, September/October 2003, pp. 86-88.
- [39] S. Arbanowski et al. "I-centric Communications: Personalization, Ambient Awareness, and Adaptability for Future Mobile Services," *IEEE Communications Magazine*, September 2004, pp. 63-69.
- [40] Wireless Strategic Initiative, <http://www.ist-wsi.org/>
- [41] Open Mobile Alliance, <http://www.openmobilealliance.org/>
- [42] Mobile IT Forum, http://www.mitf.org/index_e.html
- [43] H. Yumiba, K. Imai and M. Yabusaki, "IP-Based IMT Platform," *IEEE Personal Communications*, October 2001, pp. 18-23.
- [44] *Mobile Internet Technical Architecture*, Parts 1-3. ISBN 951-826-671-9, IT Press, 2002.
- [45] Trusted Computing Group, *TCG Specification: Architecture Overview*, Revision 1.2, 28 April 2004, https://www.trustedcomputinggroup.org/downloads/TCG_1_0_Architecture_Overview.pdf
- [46] Hewlett-Packard Corporation et al., *Advanced Configuration and Power Interface Specification*, Revision 3.0, September 2, 2004.