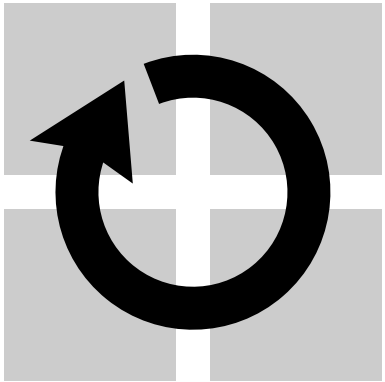
Michael Golm, Jürgen Kleinöder

Ubiquitous Computing and the Need for a New Operating System Architecture

Technical Report TR-I4-01-09 ◆ September 2001

Department of Computer Sciences 4
Distributed Systems and Operating Systems



**Friedrich-Alexander-Universität
Erlangen-Nürnberg**

Univ. Erlangen-Nürnberg • Informatik 4 • Martensstr. 1 • 91058 Erlangen • Germany
Phone: +49.9131.85.27277 • Fax: +49.9131.85.28732
E-Mail: i4@informatik.uni-erlangen.de • URL: http://www4.informatik.uni-erlangen.de

TECHNISCHE FAKULTÄT
(Faculty of Engineering Sciences)

Submitted to:

UbiTools ‘01 - Workshop on Application Models and Programming Tools for Ubiquitous Computing, September 30, 2001, Atlanta GA, USA
Held as part of UBICOMP 2001

Ubiquitous Computing and the Need for a New Operating System Architecture

Michael Golm, Jürgen Kleinöder
Dept. of Computer Science
University of Erlangen-Nürnberg
{golm, kleinoeder}@informatik.uni-erlangen.de

Abstract

Traditional operating system architectures are not able to cope with the demands of ubiquitous computing. These demands include mobility of code and running applications, precise resource control, robustness, and user-friendly failure handling. Furthermore, traditional operating systems were designed for desktop or server use and contain functionality that is ballast for a ubiquitous system. One example for such ballast is the file system.
We describe JX, our own operating system architecture, that has the desired properties, follows a microkernel approach, and structures the operating system as a set of components.

1 Introduction

Ubiquitous computing and mobile code systems impose high demands on their execution environment. Mobility of code and running applications, exact resource control, robustness, and user-friendly failure handling must be supported. These features are lacking in traditional operating systems. Many mobile code systems assume that deficits of operating systems can be corrected by using a smart middleware. We believe that this is impossible and that the properties of the operating system and even the properties of the underlying hardware shine through all middleware systems. The only sound approach is to select proper hardware and a suitable OS architecture [17].

Several virtualizations that are provided by the OS make it impossible for a runtime system to control resources. An example for such a virtualization is the virtual address space that is backed up by physical memory according to strategies that are deep inside the OS. Techniques to make these strategies externally changable [23] have not found their way into traditional operating systems.

The paper is structured as follows. Section 2 describes the essential challenges of ubiquitous computing to operating systems. Section 3 introduces the basic architecture of the JX operating system an how JX meets the challenges, described in Section 2.

2 Challenges of Ubiquitous Computing

This section gives an overview about the essential challenges of ubiquitous computing to operating systems. We describe the basic problems and sketch research results that contribute ideas to solve these problems.

Resource Control. Resource management is the central task of an operating system. All mobile code systems must control the resource consumption of the downloaded code. Traditional operating systems have problems controlling the resource consumption of processes, because they employ best-effort resource management strategies. They are especially limited, when the resource principle spawns multiple processes. However, various concepts for controlling resources have been published: Resource containers [2] are an extension for Unix-like systems to control resources when the resource principal extends into the kernel. The Rialto scheduler [15] allows to give CPU QoS guarantees in Windows 2000. Eclipse [6] is a system based on Plan9, with QoS-aware scheduler. Nemesis [18] is a vertically structured QoS OS. Scout [14] introduces techniques to manage resource consumption along invocation paths. JRes is a resource accounting extension for Java. As it operates on bytecode level, it can only account for resources that are visible at this level. Memory used for stacks or operating system resources can not be accounted.

Failure Handling. Failures are a fact of life in computer systems. The promise of ubiquitous computing is that the computer system becomes invisible for the human user. To fulfill this promise, the number of runtime failures must be reduced to a minimum and the failure handling must be user friendly. Traditional operating systems base their protection on a memory management unit (MMU). A program error is detected only when the program causes a protection violation. Until then the error may have caused trashing of unrelated data structures. The situation is even more catastrophic if the error occurs inside the OS. Thus, traditional operating systems have difficulties locating the original error and reporting it to the user in a comprehensible way.

Reliability must be assured at all software levels of the system, from low-level device drivers up to applications and mobile code. Only a statically safe language like Java is able to achive this reliability [21].

Protection and Security. On current systems protection is achieved by using Software Fault Isolation [22] or by running an interpreter for a safe language (TCL [12][13], Java, Telescript [9], Inferno/Limbo [19], Omniware [8]) .

Typical hardware for ubiquitous computing are small devices, such as personal digital assistants (PDAs). An MMU is not available on such devices.

Security requires control of communication between agents, including the control of covert channels.

Communication between agents on the same machine involves at least a local RPC between two processes. In some agent systems the inter-agent communication costs on one machine are higher than a network RPC [13]. Systems that run all agents in one process usually have difficulties isolating the agents.

Mobility. Mobile code must be platform-independent to be executable on different hardware. Most mobile code systems use an interpreter that runs as a process on top of the OS. When the agent needs to access a service of the OS it must cross an expensive protection boundary.

When mobile code is migrated to another machine, an execution environment must be created for this code. On traditional systems this means creating a new process that runs the interpreter. Although the overhead can be reduced by using a process pool [13], it is still too large for short-living agents.

A ubiquitous computing environment must support the migration of a running application, aka. strong migration [10]. To support strong migration, the state of the application, including the application-relevant OS state, must be transferred.

Process migration has been investigated for load balancing in a Unix cluster ([3], [5]). Processes can only migrate between homogenous machines. In a ubiquitous computing environment we do not have the homogenity that is common in a cluster.

Although several agent systems support strong migration ([12],[20], [9]) they pay a high price: they all use interpretation. A system using a JIT has problems when migrating running applications between different hardware architectures, because of different stack layouts, register sets, etc.

3 The JX Operating System

JX [11] is a single-address–space OS which is almost completely written in Java. Only a small core (about 100kBytes) is written in C and assembler. The system is structured into domains, which are the unit of protection and resource management. A special domain - DomainZero - allows access to the core and provides basic services, such as a name service. The Java bytecode is compiled to machine code when it is loaded into a domain. Interaction between domains is performed via portals. Because all domains execute in the same physical address space, a portal call is very fast (about 600 cycles).

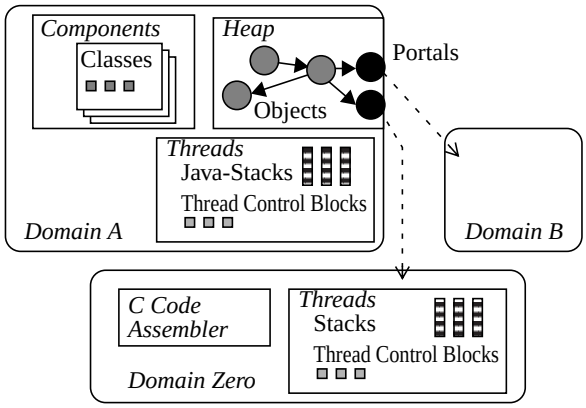


Figure 1: JX - Structure of a domain

Performance. A new OS should deliver a performance that is in the same range as established, optimized systems. We measured the file system and NFS performance, because these numbers can be compared with a traditional monolithic UNIX system, such as Linux, running on the same hardware.

We wrote a simple benchmark that sends *gettattr* requests to an NFS server. JX acting as the NFS server can process about 3000 request/sec. while Linux running on the same computer gives about 4000 requests/sec. A file system system benchmark (re-reading a 4 kByte file) measured a throughput of about 400 MBytes/s for Linux and about 200 MBytes/s for JX.

The overall performance is not as good as that of a Linux. But given the young age and limited number of developers of JX it is a very promising performance.

In the case performance is less crucial than memory consumption, the alternative to interpret the bytecode remains.

Resource management. While the overall resource consumption (CPU, memory, network bandwidth, etc.) of a domain can be precisely controlled, the domain is allowed to manage its own resources using policies that match the application's needs. When a domain depends on a service that is provided by another domain, it can even transfer resources, for example CPU time, to that domain. The service provider domain can use the donated resources for an improved service quality.

Failure handling. As usual in Java, an error is indicated by an exception. The affected thread can respond to the exception by trying to work around the problem. If this is not possible another part of the system, usually another domain that interacts with the user, can be informed.

Protection and security. Protection is based on the use of a type-safe intermediate code, the Java bytecode. Domains are completely isolated from each other. The communication between two protection domains in JX is significantly faster than a local RPC [11].

Mobility. Being a Java runtime system JX supports mobile code out of the box. Compared to other agent systems, the integration of the mobile code into the host system is very smooth, because the agent is written in the same language as the OS. We are currently investigating the necessary changes to support strong migration. In JX strong migration means migrating a running domain to another JX host. First the domain must be stopped. The complete state of the domain must be transferred. This includes the stacks of all threads. When the CPUs are different the stack frames must be converted into an architecture-independent format. In JX this is possible at GC safe points [1]. At these points the GC knows whether a value on the stack is a reference or primitive data and it can establish a mapping between these values and the (virtual) Java stack. A mapping between native code address and bytecode position is already maintained for debugging purposes and exception handling.

4 Conclusion

Ubiquitous computing imposes new demands on the flexibility of an operating system architecture.

JX is an architecture for an OS framework. It allows the fine-grained combination of arbitrary OS mechanisms and strategies. Based on Java, it is highly architecture-independent. Automatic compilation, which also may be domain-specific, provides reasonable performance.

Providing a maximum of flexibility, the JX architecture is one example for a basis for the implementation of the various solutions of the problems of ubiquitous computing.

5 References

[1] O. Agesen, D. Detlefs,j. Moss, B. Eliot.: Garbage Collection and Local Variable Type-Precision and Liveness in Java Virtual Machines, *PLDI 98*, pp. 269-279

[2] G. Banga, P. Druschel: Resource containers: A new facility for resource management in server systems, *OSDI 1999*

[3] A. Barak and O. La'adan: The MOSIX Multicomputer Operating System for High Performance Cluster Computing, *Journal of Future Generation Computer Systems*, Vol. 13, No. 4-5, pp. 361-372, March 1998.

[4] J. Baumann, F. Hohl, K. Rothermel, M. Straer: Mole - Concepts of a Mobile Agent System. *WWW Journal*, *Special Issue on Applications and Techniques of Web Agents*, 1(3):123-137, pp 2-13, Baltzer Science Publishers

[5] P. Bepari: *Distributed Process Management Protocol: A Protocol for Transparent Remote Execution of Processes in a Cluster of Heterogeneous Unix Systems*, Master's thesis, 1999

[6] John Bruno, Eran Gabber, Banu Ozden and Abraham Silberschatz: The Eclipse Operating System: Providing Quality of Service via Reservation Domains, In *Proc. of Usenix 1998*

[7] D. Chess, C. Harrison, A. Kershenbaum: *Mobile Agents: Are They a Good Idea?*, IBM Research Division, T.J. Watson Research Center, Yorktown Heights, New York, March 1995

[8] Colusa Software: *Omniware Technical Overview*, www.colusa.com

[9] General Magic, Inc.: *The Telescript Language Reference*, Oct. 1995

[10] C. Ghezzi, G. Vigna: Mobile Code Paradigms and Technologies: A Case Study. In K. Rothermel and R. Popescu-Zeletin, editors, *Proceedings of the 1st International Workshop on Mobile Agents (MA '97)*, LNCS 129, Springer, April 1997.

[11] M. Golm, J. Kleinöder, F. Bellosa: Beyond Address Spaces - Flexibility, Performance, Protection, and Resource Management in the Type-Safe JX Operating System. *Proc. of HotOS 2001*, May 20-23, 2001, Schloß Elmau, Germany.

[12] R. S. Gray. Agent Tcl: A transportable agent system. In *Proc. of the CIKM Workshop on Intelligent Information Agents, Fourth International Conference on Information and Knowledge Management (CIKM 95)*, Baltimore, Maryland, December, 1995

[13] R. S. Gray, George Cybenko, David Kotz D. Rus: Mobile Agents: Motivations and State-of-the-Art Systems. In Jeffrey M. Bradshaw, editor, *Handbook of Agent Technology*, AAAI/MIT Press, 2000

[14] J.H. Hartman, A.B. Montz, D. Mosberger, S.W. O'Malley, L.L. Peterson, and T.A. Proebsting: *Scout: A communication-oriented operating system.* TR 94-20, University of Arizona, Tucson, AZ, June 1994

[15] M. B. Jones, J. Regehr, S. Saroiu: Two Case Studies in Predictable Application Scheduling Using Rialto/NT, *RTAS 2001*

[16] T. Jaeger, N. Islam, R. Anand, A. Prakash, and J. Liedtke. Flexible Control of Downloaded Executable Content. *ACM Transactions on Information and System Security*, Vol. 2, Iss. 2 , 1999

[17] J. Lepreau, B. Ford, M. Hibler: The persistent relevance of the local operating system to global applications. *SIGOPS European Workshop*, 1996

[18] I. M. Leslie, D. McAuley, R. Black, T. Roscoe, P. Barham, D. Evers, R. Fairbairns, and E. Hyden: The design and implementation of an operating system to support distributed multimedia applications, *IEEE Journal on Selected Areas in Communications* 14(7), pp. 1280-1297, 1996

[19] Lucent Technologies. *The Limbo programming language*, 1997. http://inferno.lucent.com/inferno/limbo.html.

[20] H. Peine, T. Stolpmann: The Architecture of the Ara Platform for Mobile Agents, In: Rothermel K., Popescu-Zeletin R. (Eds.), *Mobile Agents, Proc. of MA'97*, Springer Verlag, Berlin, April 7-8, LNCS 1219, pp 50-61

[21] C. Szyperski , J. Gough: The role of programming languages in the life-cycle of safe systems. *Proc. of the Intl Conf on Safety Through Quality*, Kennedy Space Center, FL, USA, October 1995.

[22] R. Wahbe, S. Lucco, T. E. Anderson, and S. L. Graham. Software fault isolation. In *14th ACM Symposium on Operating System Principles*, vol. 27, pp 203-216, Dec. 1993

[23] M. Young, A. Tevanian, R. Rashid, D. Golub, J. Eppinger, J. Chew, W. Bolosky, D. Black, R. Baron: The duality of memory and communication in the implementation of a mulitprocessor operating system. *Proc. of the 11th SOSP*, pp. 63-76, Nov. 1987