CORE
Metadata, citation and similar papers at core.ac.uk
Provided by Waseda University Repository

Using Virtualized Operating Systems as a Ubiquitous Computing Infrastructure

Shuichi Oikawa, Midori Sugaya, Masatoshi Iwasaki and Tatsuo Nakajima
Graduate School of Science and Engineering, Waseda University
3-4-1 #61-505, Okubo, Shinjuku, Tokyo 169-8555, Japan
Email: {shui,tatsuo,pingoo,doly}@dcl.info.waseda.ac.jp

Abstract

The Personalization of computing environments is one of the key aspects of ubiquitous computing, and such personalization requires isolated computing environments for better security and stability. This paper describes our ubiquitous computing infrastructure architecture that is based on virtualized operating systems in order to enable ubiquitous devices and servers to be shared securely and reliably. Our architecture also includes CPU resource management mechanisms to support time sensitive applications and to make the execution of applications stable on shared devices.

1. Introduction

Ubiquitous computing is an emerging technology for the retrieval and processing of information from the real world that were not previously available and for the control of everyday interactions with various people and things. Ubiquitous computing environments employ a number of ubiquitous computing devices that are embedded in many things at many places. Examples of such devices include information and networked appliances, video cameras and monitors, controllers, sensors, and actuators. Using those devices, a new form of intelligent living environments can be created by making information access and processing easily available for everyone from everywhere at any time [11].

Personalization is one of the key aspects of ubiquitous computing. For example, context-awareness is one of the most important issues in ubiquitous computing. It enables the integration of physical and cyber spaces in order to personalize our living environments and to reduce the complexities in our daily living. Such environments are apparently not general ones, but are very specific to each user. A personalized environment takes into account a user's personal information, preferences, past activities and decisions, and the current contexts in order to augment intelligence and to ease one's personal life.

The personalization of ubiquitous computing environments requires the functionality of security, stability, and isolation. While malicious attacks should be prevented from intruding into a computing environment, they should not affect the other environments. A secure infrastructure should be able to contain security breaches in an infected environment. Time sensitive applications requires their stable execution to meet their timing requirements and to keep their QoS. Such provision of stability can make an computing environment not affected by the activities in the other environments. As described above, a computing environment for each user should be isolated from the others in terms of security and stability.

This paper presents our ubiquitous computing infrastructure architecture that is based on virtualized OSes (operating systems) [1, 4, 10]. By virtualizing OSes, their multiple instances can run on a single computer system. Each user can have one's own OS environment and construct a computing environment on top of it. Virtualized OSes on the same computer systems can communicate only through their network communication channels; thus, a virtualized OS can stay secure even if there is an intruded virtualized OS on the same system. Our architecture also include CPU resource management mechanisms to control the allocation of CPU times to virtualized OSes and applications running in them. The CPU resource management enables the stable execution of applications. Thus, a computing environment on top of a virtualized OS can be isolated from the others.

Our goal is to construct a ubiquitous computing infrastructure at the OS level focusing on the personalization of ubiquitous computing environments. There are many researches on application middleware, which creates a common infrastructure to build applications on top of it and to ease the application development. Only some of them have the focuses on the provision of OS level functionality. 2K [7] and Gaia [9] are the researches of which goal is the provision of distributed service infrastructures. 2K is a distributed OS, which aims at the management of dynamic

Proceedings of the Second IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems (WSTFEUS'04)
0-7695-2123-1/04 $ 20.00 © 2004 IEEE
IEEE COMPUTER SOCIETY

heterogeneous computing environment. Gaia is a meta OS, which provides services for the spontaneous networking and communication of devices. Those OSes are built on top of the existing OSes to provide the common services. In contrast to those researches, we enable the existing OSes to be used as our common service basis using OS virtualization technologies. By using the existing OSes, we can take advantage of a huge number of the existing software packages. By using virtualized OSes along with CPU resource management, we can provide better security, stability, and isolation.

The rest of this paper is organized as follows. Section 2 presents our ubiquitous computing infrastructure architecture that supports isolated personalization. Section 3 describes some application scenarios, and Section 5 concludes this paper.

2. Ubiquitous Computing Infrastructure

This section first describes the requirements for a ubiquitous computing infrastructure, and then proposes the use of virtualized OSes as such an infrastructure. It also describes the resource management to create stable computing environments, and the instantiation and configuration system to ease the use of virtualized OSes.

2.1. Requirements

We consider a practical ubiquitous computing environment where multiple users share the same environment. An environment constructed in a home is shared by its family members, and one in an office space is shared by its corporate employees. Users of a ubiquitous computing environment use ubiquitous computing devices embedded in it. Since the same environment is shared by multiple users, those devices are also shared by them. We call those shared computing devices servers. In such an environment where sharing servers can happen everywhere, security and stability are two major requirements to the infrastructure of computing environments.

A secure environment should be able to prevent malicious attacks from intruding into the environment. There, however, can be a hidden software bug that creates a security hole. Even if such a security hole was attacked and allowed an intrusion into the environment, it is required for a secure infrastructure to contain the security breach in the infected environment and not to affect the other environments. In personalized ubiquitous computing environments, it should also be possible to implement different levels of security for different users since they have different preferences and attitudes towards security.

The provision of stability is needed to support time sensitive applications. Time sensitive applications have their timing requirements and require certain amounts of CPU times in certain time frames to be allocated to them. The stable execution of other non-time sensitive applications, however, should not be disturbed by time sensitive applications.

PDAs are not shared, but the same requirements can be applied since they are also used in the same shared environments.

2.2. Virtualized Operating Systems

In order to meet the above requirements, we use virtualized OSes with the CPU resource management functionality. Virtualized OSes are OSes of which multiple instances can run on a single computer system. There are mainly two models to realize virtualized OSes. One is to use a VMM (Virtual Machine Monitor) that creates a VM (Virtual Machine), in which an OS runs [4]. The other is to use an OS personality server running on top of the host kernel, which can be a microkernel [5] or a monolithic kernel [1]. Those two models differ only in a way for the underlying layers to provide the abstractions of computing resources. From here on, we use the host kernel to describe the underlying layers that provides the abstractions of computing resources. In contrast to the host kernel, the OS kernel in a virtualized OS environment is called a guest kernel.

The virtualization of OSes can isolate one user's execution environment from the others that share the same ubiquitous server since users can own and use their personal installations of OSes. By doing so, the execution of an untrusted application can be contained in a separate virtualized OS. Moreover, virtualized OSes can add advanced security features by employing virtual machine based intrusion detection technologies [2, 3].

The other advantage of using virtualized OSes for personalized ubiquitous computing environments is that different virtualized OSes can run simultaneously on the same ubiquitous server. Such a feature makes it possible to leverage the existing applications and to support legacy applications; thus, it protects the past investments and enables phased transitions to new application platforms [6].

Figure 1 depicts the model that realizes virtualized OSes on top of the host kernel. The figure shows an example configuration on a ubiquitous server that is shared by User A and B. They use their own virtualized OSes. User A uses two virtualized OSes, VOS 1 and 2. User B uses one virtualized OS, VOS 3. There is another virtualized OS, VOS 4, which provides core system services. Applications in VOS 1 and 3 are communicating with a server in VOS 4 through internal connections provided by the host kernel. A separate virtualized OS, VOS 4, is used to run the core system services in order to be isolated from any faults and security breaches that can happen in user's virtualized OSes. Users can also employ a separate virtualized OS to run an

IEEE COMPUTER SOCIETY

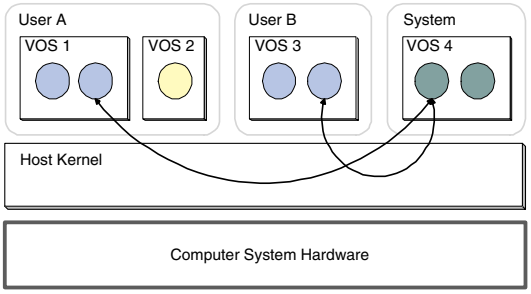


Figure 1. Virtualized Operating Systems

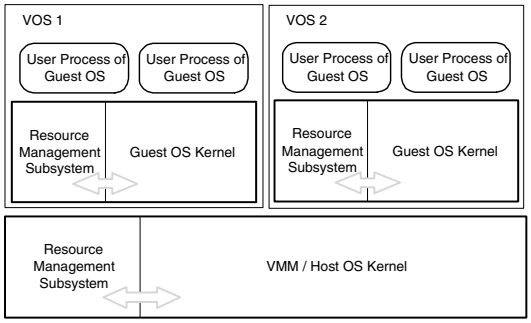


Figure 2. Architecture of Virtualized OS with Resource Management Mechanisms

untrusted application. For example, User A uses VOS 2 to run an untrusted application. The host kernel provides CPU resource management to guarantee and to limit the allocation of CPU times to each virtualized OS. A virtualized OS that runs time sensitive applications can always receive certain CPU times needed for their stable execution. A virtualized OS that runs untrusted applications can be configured to receive limited CPU times, so that their effect to the other virtualized OSes can be restricted even if they fall into an infinite loop.

More details of resource management is described in the next section, and more application scenarios using virtualized OSes are presented in Section 3.

2.3. Resource Management

In order to have an execution environment completely isolated from the others, the functionality of CPU resource reservation is required to protect CPU resources allocated for the environment. One computing environment created by a virtualized OS is completely isolated if the execution of programs in that environment is not affected by activities in the other virtualized OSes that share the same computer system. It means that programs in that environment can run as if its OS occupies a single computer system. Since personalized ubiquitous computing environments require guaranteed allocation of CPU times for their time sensitive applications as described above, those CPU times have to be allocated to their virtualized OSes that host those environments. Thus, the CPU resource reservation mechanisms are needed in the host kernel. Such reservations of CPU times also work as CPU resource protection especially if actual utilization of CPU times can be enforced to cap the maximum CPU time allocation. The enforcement of CPU time utilization prevents applications, and thus virtualized OSes, from overusing CPU times. It can limit the negative effects on CPU resource allocation to the other virtualized OSes.

Figure 2 depicts the overview of the architecture and its components. There are the resource management subsystems in both the host and guest kernels. The resource management mechanisms in the host and guest kernel control the allocation of CPU times to virtualized OSes and the guest kernel's user processes, respectively. An appropriate CPU time reservation is made for a virtualized OS in order to enable CPU time reservations in the virtualized OS.

2.4. Instantiation and Configuration

In order to be able to use virtualized OSes as our ubiquitous computing infrastructure, we also need a system that eases the instantiation and configuration of virtualized OSes. Our instantiation and configuration system helps users to create new virtualized OSes and to configure them for specific uses. For example, when a user would like to create a new virtualized OS to run an untrusted application that accesses Internet sites and sends back retrieved information, the virtualized OS needs an Internet connection and an internal connection to the originating virtualized OS. The user can use the instantiation and configuration system to automate the process of the creation and configuration of such a virtualized OS and the execution of the specified application in it.

3. Application Scenarios

This section describes two application scenarios of using virtualized OSes on a sever and a PDA.

A user can use multiple instances of virtualized OSes to configure environments to isolate untrusted applications and to contain them in separate virtualized OSes. Such separate virtualized OSes can be associated with limited CPU resources by using our resource management mechanisms, so that their overuse of CPU times does not affect the other applications. Virtualized OSes that run untrusted applications are exposed to risks of being intruded by viruses. Infected virtualized OSes can simply be removed without losing any important information. In contrast to untrusted ap-

IEEE COMPUTER SOCIETY

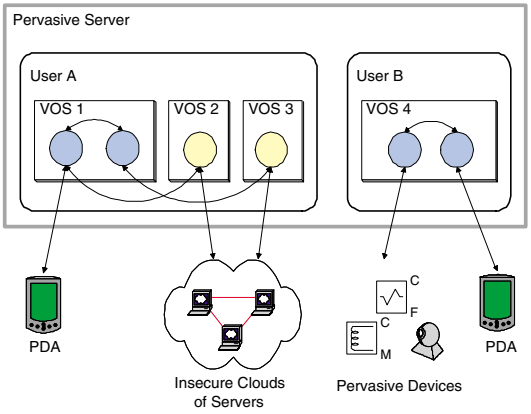


Figure 3. Application Scenario on a Server

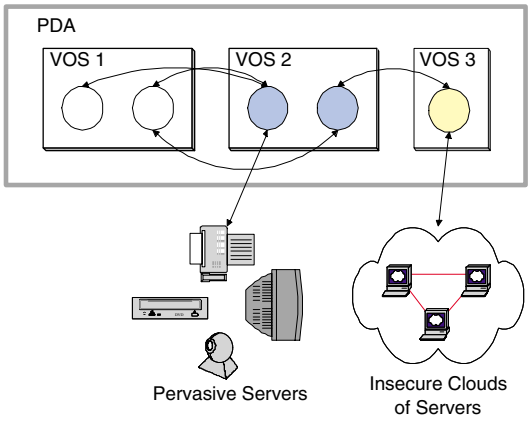


Figure 4. Application Scenario on a PDA

plications, trusted time sensitive applications can consume reserved CPU times to maintain the desired QoS specified by their users. Our resource management mechanisms guarantee the allocation of reserved CPU times to those applications.

Figure 3 depicts our application scenario on a server. User A uses three virtualized OSes, VOS 1, 2, and 3. User A runs untrusted applications, which were downloaded from the Internet, in VOS 2 and 3. The allocation of CPU times to VOS 2 and 3 is capped to limit their CPU resource usage. By using such a configuration, VOS 1 can be protected from the untrusted applications in VOS 2 and 3. Even if they have software bugs and fall into a busy infinite loop trying to use CPU times as much as possible, our CPU resource management mechanisms limit their CPU resource usage by certain amounts. If they were viruses and intruded VOS 2 and 3, the intrusion does not affect VOS 1 and those contaminated virtualized OSes can be simply abandoned. User B also uses the same server at the same time. User B uses only one virtualized OS, VOS 4, in which time sensitive applications, which control devices, are running. Our software architecture enables those time sensitive applications running in VOS4 to reserve CPU times and to guarantee their timely execution. Any activities of User A do not affect the execution of User B's applications. Therefore, User A and B can securely share the same server.

A PDA is a personal device that is not shared with other users; thus, there is no need to consider its sharing. Virtualized OSes, however, can be used to configure a PDA to internally realize protected domains by running applications and core services in different virtualized OSes. A firewall running in a separate virtualized OS can also be used to reinforce the security of the PDA.

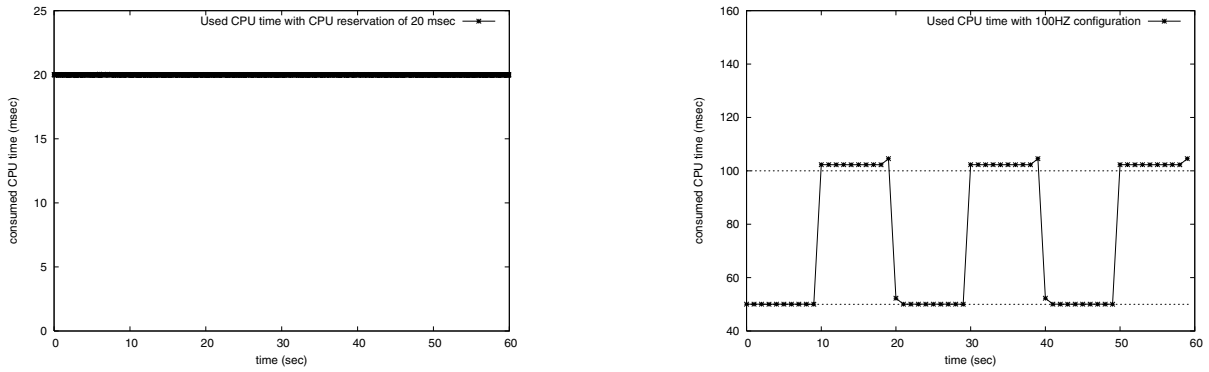
Figure 4 depicts our application scenario on a PDA. Core services, such as a storage service and a window system, are running in VOS 1. User applications are running in VOS 2. Connections to the Internet are provided through the firewall running in VOS 3. By employing the firewall and having it run in the separate virtualized OS, VOS 1 and VOS 2 can be protected from malicious attacks through Internet connections. By running user applications in VOS 2, core services running in VOS 1 can be isolated from software faults due to bugs in user applications running in VOS 2.

4. Results from the Current Prototype

We developed a prototype virtualized OS environment on Linux/RK [8] and by adapting its resource management mechanisms to UML (User-Mode Linux) [1]. We call UML with the resource management mechanisms UML/RK. This section evaluates our current prototype and shows a virtualized OS environment can isolate resource management from each other.

First, we show the share of the CPU resource used by a whole virtualized OS environment can be reserved and also limited in order to create an isolated execution environment. Figure 5 (a) shows the CPU times consumed by all processes that creates a virtualized OS environment of UML/RK in each period of CPU time replenishment. The virtualized OS environment created by UML/RK consisted of the total of 10 processes that includes the kernel, its supporting programs, and its user processes of UML/RK. The consumed times were calculated from the CPU cycles actually executed within each period. UML/RK was booted with the reservation parameter of 20 millisecond CPU time within 50 millisecond period. Figure 5 (a) shows the result of the execution of UML/RK in which a program that executed an infinite busy loop was running. We created 8 disturbing processes also being executed on Linux/RK aside of UML/RK. The disturbing processes executed an infinite busy loop trying to consume CPU cycles as much as possible. The results show that UML/RK received the reserved

IEEE COMPUTER SOCIETY



(a) (b)

Figure 5. CPU Time Reservation and Enforcement for a Virtualized Operating System Environment

CPU times even with disturbing processes running aside of it, and the received CPU times were also limited as specified; thus, the execution of UML/RK was correctly isolated.

Next, we show that CPU resource management is possible in a virtualized OS environment. In order to evaluate that CPU times can be effectively reserved and also be enforced by our CPU resource management mechanisms in UML/RK, a benchmark program ran with a reservation of the CPU resource in a UML/RK virtualized OS environment. Figure 5 (b) shows consumed CPU times calculated from the CPU cycles actually executed within each period. UML/RK was booted with the initial reservation parameter of 20 millisecond CPU time within 50 millisecond period. Figure 5 (b) shows the result of the execution started with the reservation of 50 millisecond every 1 second. After 10 seconds, the reservation parameter was changed to 100 millisecond every 1 second. 10 seconds later, the reservation parameter was changed again to 50 millisecond every 1 second. It repeatedly changed the reservation parameter every 10 seconds. The results show that UML/RK can rigidly enforce their CPU times and the CPU times obtained by processes change promptly when the reservation parameters are changed.

5. Conclusion

This paper described our ubiquitous computing infrastructure architecture that is based on virtualized OSes in order to realize secure, stable, and isolated computing environments. Our architecture enables ubiquitous devices and ubiquitous servers to be shared securely. Our CPU resource management mechanisms support time sensitive applications and make the execution of applications stable on shared devices.

We are actively working on enabling virtualized OSes to be our ubiquitous computing infrastructure. We implemented the resource management mechanisms in UML as our prototype. We are currently working on constructing a virtualized OS environment that can scale from small embedded systems to larger systems.

References

[1] J. Dike. A User-Mode Port of the Linux Kernel. In *Proceedings of the 2000 Linux Showcase and Conference*, October 2000.

[2] G. W. Dunlap, S. T. King, S. Cinar, M. Basrai, P. M. Chen. ReVirt: Enabling Intrusion Analysis through Virtual-Machine Logging and Replay. In *Proceedings of the 2002 Symposium on Operating Systems Design and Implementation*, December 2002.

[3] T. Garfinkel and M. Rosenblum. A Virtual Machine Introspection Based Architecture for Intrusion Detection. In *Proceedings of the Internet Society's 2003 Symposium on Network and Distributed System Security*, February 2003.

[4] R. P. Goldberg. Survey of Virtual Machine Research. *IEEE Computer Magazine*, pages 34–45, June 1974.

[5] D. Golub, R. Dean, A. Forin and R. Rashid. Unix as an Application Program. In *Proceeding of the Usenix Summer Conference*, June 1990.

[6] T. Kindberg and A. Fox. System Software for Ubiquitous Computing. *IEEE Pervasive Computing*, vol. 1, no. 1, January 2002.

[7] F. Kon, R. H. Campbell, M. D. Mickunas, K. Nahrstedt, and F. J. Ballesteros. 2K: A Distributed Operating System for Dynamic Heterogeneous Environments. In *Proceedings of the 9th IEEE International Symposium on High Performance Distributed Computing*, August 2000.

[8] S. Oikawa and R. Rajkumar. Portable RK: A Portable Resource Kernel for Guaranteed and Enforced Timing Behavior. In *Proceedings of IEEE Real Time Technology and Applications Symposium*, June 1999.

[9] M. Roman, C. Hess, R. Cerqueira, A. Ranganat, R. H. Campbell, and K. Nahrstedt. Gaia: A Middleware Infrastructure to Enable Active Spaces. *IEEE Pervasive Computing*, pages 74–83, October 2002.

[10] J. Sugerman, G. Venkitachalam, and B. H. Lim. Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor. In *Proceedings of 2001 USENIX Annual Technical Conference*, 2001.

[11] M. Weiser. The Computer for the Twenty-First Century. *Scientific American*, September 1991.

IEEE COMPUTER SOCIETY