2018 IEEE 38th International Conference on Distributed Computing Systems

Operating Systems for Internetware: Challenges and Future Directions

Hong Mei, Yao Guo
Key Laboratory of High-Confidence Software Technologies (Ministry of Education),
Peking University, Beijing, 100871, China
Email: {meih, yaoguo}@pku.edu.cn

***Abstract*—An operating system is an essential layer of system software that is responsible for resource management and application support on a computer system. As the evolvement of computer systems, the concept of OSs has also been evolved into many new forms beyond the traditional OSs such as Linux and Windows. We call this new generation of OSs as ubiquitous operating systems (UOSs). Among many new types of UOSs, we are particularly interested in the operating systems for Internetware, i.e., Internetware Operating Systems. Internetware is a paradigm for new types of Internet applications that are autonomous, cooperative, situational, evolvable, and trustworthy. An Internetware OS represents our perspective on the OS for future Internet-based applications. This paper discusses the examples, technical challenges and our recent effort on Internetware OSs, as well as our vision on the future of Internetware OSs. We believe that, in the foreseeable future, Internetware OSs will become ubiquitous and could be built for many different types of computer systems and beyond.**

***Keywords*-Operating systems, Internetware, software-defined everything, ubiquitous operating systems.**

I. Introduction

An operating system (OS) is a layer of system software that runs between applications and computer hardware, managing hardware resources such as processors, memory and storage, while providing support to the software applications running above it [1]. Although the earliest computers such as ENIAC did not have an OS, most of the later computer systems require the installation of an OS to help bridge the gap between applications and hardware. It is unrealistic to build applications to run on modern bare-metal machines without OSs running in between.

As shown in Figure 1, an OS provides resource management capabilities on processors, storage, and peripheral device components, while providing systems calls and human-computer interfaces for applications and end-users. It exhibits different views when we look at an OS from different perspectives:

- From the perspective of a computer system, an OS can be viewed as a resource manager. An OS manages and coordinates the utilization of all kinds of low-level hardware and software resources, while supporting the efficient utilization of these resources. At the same time, an OS also bridges heterogeneous hardware resources through hardware drivers, in order to improve the interoperability among the whole computer system.

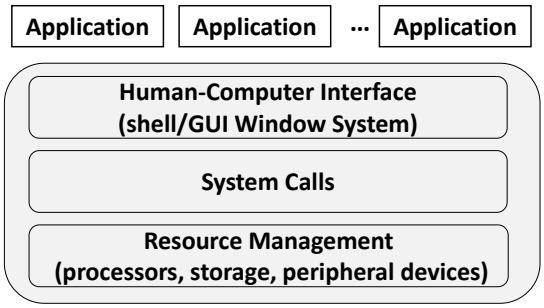

Figure 1. The architecture of a traditional OS.

- From the perspective of system users, an OS can be viewed as a virtual machine (VM). On one hand, an OS provides abstraction to hide the details in the hardware resources. On the other hand, it also provides user-friendly user interfaces (UI). For software developers, an OS also determines the programming model for the applications running above it.
- From the perspective of application software, an OS can be viewed as a development and execution platform. It provides necessary supports for the development and execution of all application software. For example, an OS typically includes an execution environment for application software, runtime resource management and scheduling, as well as tools for software development and maintenance.

A. A Brief History of OS

There were no OSs on the earliest computers, as software applications run directly on bare-metal machines. As the complexity of a computer system kept increasing, it becomes harder and harder to manage the resources on a computer directly in an application. As a result, more and more common functionality are abstracted as drivers and libraries, creating a system software layer running between applications and hardware, which can be shared between different applications. This software layer were then named operating systems as it was originally developed to abstract the operating capabilities to ease the tasks of computer operators. However, current OSs no longer focus on the operation of a computer system, instead

2575-8411/18/$31.00 ©2018 IEEE
DOI 10.1109/ICDCS.2018.00138

on resource management and application development and runtime support for a given system.

Modern OSs mostly employ the similar architecture as the UNIX OS, while customizing their functionalities for a particular type of computer systems. For example, desktop OSs such as Windows or MacOS focus on better graphical user interfaces (GUI) to provide better user experiences for personal users. Mobile OSs such as Android and iOS are also based on UNIX-like kernels, while adding a mobile platform layer to support the development and execution of mobile apps, in order to provide better experiences for mobile users.

As the fast adoption of computer networks since 1980s, it becomes critical for an OS to provide networking capabilities, thus creating many new OSs named as networking operating systems. The earliest networking OSs such as Novell Netware focused on the interconnection of computers within a local network. These OSs were later discontinued as most of these capabilities are incorporated in newer versions of desktop OSs, as connecting to the network has become a must-have for many users.

In addition to networking extensions to OSs, another important direction for networking support in a tradition OS is through networking middleware [2]. Middleware is defined as a layer between applications and the OS, providing a series of APIs for mainly networking support. Middleware help hide the complexity and heterogeneity of the low-level infrastructure (including networks, computers, OSs and programming languages), such that applications only need to concern about its main execution logics.

B. New Types of OSs

With the emergence of Internet, the talk of an Internet OS began to surface in mid-1990s during the war between Netscape and Microsoft. Marc Andreessen of Netscape announced a set of new products that would help transform their browser into an "Internet OS", which would provide the tools and programming interfaces for a new generation of Internet-based applications. The idea of Internet OS has kept floating around in the past 20 years, as many different implementations have been proposed, including the Java-based operating system JavaOS [3] and most recently, Google's browser-based operating system Chome OS [4].

Many of these OSs were focused on providing networking capabilities or incorporating Internet-related data (resource) management functionalities with OS-like structures. For example, taking the perspective of "Internet as a computer", an Internet OS focuses on providing better support on software applications running on the Internet. Many Internet applications do not run on a single computer, or even on a single computer cluster. Their components (or services) may run on computers on geographically distributed computer systems (or even virtual machines), providing specific services through Internet connections. These new kinds of networking OSs often run above existing OSs such as Windows or Linux, while constructing a new layer above the existing OSs to provide support for Internet-based applications and services. These networking OSs are structured as "*Meta-OSs*", which means that they are OSs running above existing (traditional) OSs.

Besides Internet OS, many new types of OSs are also Meta-OSs. For example, the most popular OS for robots, ROS [5], provides robot application development support and robot communication capabilities through a set of robot-related SDKs and libraries. Unlike Linux or Windows, ROS is a meta-OS running above Linux or Windows. However, similar to Linux or Windows, it includes OS-like resource abstractions and management capabilities, as well as application development and runtime support.

C. Ubiquitous Operating Systems

Mark Weiser envisioned a future of ubiquitous computing [6], where computing exists everywhere. To achieve true ubiquitous computing, every object and entity in the world may be programmable, for example, smart bulbs can sense the environment and change its colors accordingly. We have not reached the point where we need to build an OS for a smart bulb. However, we already need to build an OS for a robot, a smart home, or a network adaptor (i.e., SDN). As it seemed unrealistic when Mark Weiser proposed ubiquitous computing more than a quarter of a century ago, we argue that OSs will be ubiquitous too.

Ubiquitous operating systems (UOSs) [7] represent a new set of operating systems beyond the traditional OSs such as Linux and Windows. As the basic functionality of an OS is managing underlying resources and providing programming capability to the applications running above, a new layer of software can be regarded as a form of OS if it provides these kinds of functionalities. For example, Android itself is not a traditional OS, as it builds upon an existing Linux kernel. However, Android provides specialized resource management for smartphones, while providing a programming and runtime framework for Android apps. Thus, most people would regard Android as a mobile operating system (i.e., a UOS) itself, although it is essentially a middleware layer if we look at it from a traditional perspective.

Looking forward to the future, we believe that UOSs can be built for devices large and small, at the scale of single computers as well as clusters, at both the hardware and software levels, and for applications ranging from smart homes to smart cities. Although these OSs might look very different from one another, they all embody the same general principles and characteristics as traditional OSs – namely, resource virtualization and function programmability.

We can categorize existing and future UOSs into three different types:

- **UOSs for different computing devices.** These are mainly extended from (and include) traditional OSs

such as Linux and Windows. Examples include mobile operating systems such as iOS and Android, embedded OSs such as Android Thing, or robot OSs such as ROS. We have already witnessed that different OSs have been built for small embedded systems, mobile smartphones and tablets, traditional desktop PCs and laptops, standalone workstations and networked servers. In the future, OSs will be extended to almost all old and new IT systems, including both tiny computing devices in edge computing, and large computing environments such as supercomputers.

- **UOSs for new types of applications.** The previous type of UOSs focus on computing devices that a UOS needs to manage from a bottom-up view. In contrast, from a top-down view, we envision that a UOS can be built for each new type of applications in different domains. From this perspective, a UOS focuses on providing application programming interfaces (APIs) and library support for building new types of applications. For example, a UOS can be built for new application domains such as big data, Internet of Things, or even artificial intelligence.
- **UOSs for networked systems.** This view of UOSs is based on the observation that an OS can run across different networked devices such as a cloud. For example, A UOS can be built for a networked system including not only computation servers, but also mobile terminals. In such cases, a UOS needs to consider how to manage distributively networked resources, as well as how to provide programming functionality for new applications running on a networked system. These UOSs are mostly meta-OSs, which include OSs for cloud computing environments, as well as OSs for various enterprises, organizations, or institutions, as well as government agencies.

Please note that, sometimes, a UOS can be classified into not only one, but multiple categories. For example, mobile OSs such as Android can be considered as not only a UOS for new types of mobile devices, but also a UOS for new types of mobile apps. The categories here are provided only to help us understand the evolvement and implications of new types of UOSs.

II. Internetware Operating Systems

Internetware [8], [9], [10] is a new software paradigm for Internet computing. This term was coined to describe the new types of software applications on Internet with characteristics that are autonomous, cooperative, situational, evolvable, emergent, and trustworthy. The Internetware software model consists of a set of autonomous software entities distributed over the Internet, together with a set of connectors to enable collaborations among these entities in various ways. Internetware software entities can sense dynamic changes in the running environment and continuously adapt to these changes through structural and behavioral evolutions. The research of Internetware includes its models, development methodology and runtime support, and quality measurement and insurance [11].

Figure 2 shows the concept of an Internetware OS, which represents our perspective on the OS for future Internet-based applications. An Internetware OS is a type of UOS for networked systems. The structure of an Internetware OS is similar to the layered structure of a tradition OS, in the sense that it also provides the resource management functionalities and provides development and runtime support the applications running above it. However, because Internetware applications are not confined within a single computer system and could run on large-scale systems such as cloud, or even the Internet, thus an Internetware OS is a meta-OS running above existing OSs such as Windows and Linux. The resources managed by an Internetware OS are also generalized in the sense that it can be any types of resources existed on the Internet.

A. Examples of Internetware OSs

In this paper, as an Internetware OS is defined as an OS for new types of Internet applications, thus many existing OSs are qualified as Internetware OS, including:

- Robot OS: The Robot Operating System (ROS) [5] is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms. Although it is not a traditional OS, ROS provides a set of abstractions and APIs that can be used in developing robot applications across different robotic platforms. Thus, ROS is a meta-OS that provides development and runtime supports of robotic applications. ROS has become the most popular robot OS and has adopted and ported to dozens of different robotics systems.
- HomeOS: As a Microsoft initiative that aims to enable smarter homes for everyone [12], the major goal of HomeOS is to simplify the management of technology and to simplify the development of applications in the home. HomeOS provides a centralized, holistic control of devices in the home. It provides to users intuitive controls to manage their devices, and to developers higher-level abstractions to orchestrate the devices in the home. HomeOS even has its own app store called HomeStore, through which users can easily add and obtain applications that are compatible with devices in their homes and obtain any additional devices that are needed to enable desired applications. It was developed as a research prototype and deployed to more than a dozen homes.
- City OS: There are many initiatives that claim they are building different city OSs. One of the earliest city OSs is the PlanIT Urban Operating System [13],

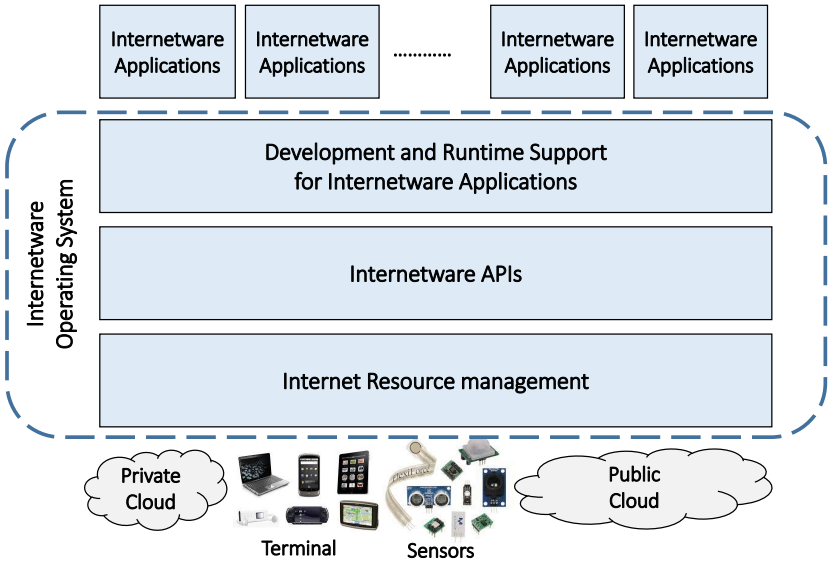

Figure 2. The concept of an Internetware OS.

which claims to be the smartest, most flexible way to converge infrastructure with a world of sensors, devices and people across developments of scale and entire cities. It provides abstractions and management interfaces to city systems such as energy, water, waste management, transportation, telecommunication, and healthcare, while providing programming APIs that ensure interoperability between different platforms.

- Cloud OS: Conceptually, a cloud OS [14] does what a traditional OS does C manage applications and hardware - but at the scope and scale of cloud computing. Compared to a traditional OS, a cloud OS replaces file systems with object storage, enabling infinitely scalable storage capacity and I/O throughput. Instead of managing processes, a cloud OS deals with virtual machines (VMs) and tasks. More importantly, a cloud OS also offers various cloud-based APIs, that can be used to in cloud applications to utilize the cloud resources. Some APIs, such as object storage APIs, have even become common across all major public cloud providers. Many cloud service providers have created their own cloud OSs, such as Microsoft Azure, Amazon AWS, and Huawei FusionSphere. There are also popular open-source cloud OSs such as OpenStack and Apache CloudStack.
- IoT OS: Google's Android Things (Brillo) OS [15]is an Android-based embedded operating system platform. It is aimed to be used with low-power and memory constrained Internet of Things (IoT) devices. Android Things allows developers to build a smart device using Android APIs and Google Services. It is similar to Android, as it can be applied with the usual Android development stackłAndroid Studio, the official SDK, and Google Play Services, however, it is customized to make it applicable to IoT applications. Android Things is a simplified and polished Android to make it run smoothly on low-power IoT devices. It also introduces a new communication protocol called Weave.

B. Technology Support for Internetware OSs

As a new type of UOS, Internetware OSs require the support of the following core techniques: virtualization, application programming interface (API), security and privacy, and customization.

1) Virtualization: Internet is developing into a huge virtualized computing platform, providing distributed storage and computing capability to the applications running on it (i.e., Internetware) and end-users. Internetware OS manages distributed software and hardware resources with virtualization, and offers scalable resource accesses and computing capabilities based on the application or user requirements.

Virtualization technologies appeared first in the IBM mainframe systems in the 1960s. These machines created virtual machines (VMs), which are managed by virtual machine monitors (VMMs) [16]. Each VM can run an operating system independently, thus many OSs can run on a single machine. Virtualization has become more popular in recent years, with the prevalence of multi-core system, clusters and cloud computing. Virtualization not only reduces the IT cost, it can also improves the security and reliability of the systems.

Today, the concept of virtualization is used not only for various computing resources, i.e., virtual machines. From the perspective of Internetware OS, virtualization means that we can abstract all the underlying software and hardware resources through virtualization, then offers them to the application software and users through unified application programming interfaces (APIs). As a result, we are referring to the "virtualization" in a generalized sense. Not only can we virtualize computing resources, we can also virtualize storage resources, data resources, sensors, network devices, terminal devices, and even user information.

There have been many advances in the area of virtualization for new computing models such as cloud computing and mobile computing, for example, OS kernel virtualization [17], data center virtualization [18], nested virtualization [19], [20], light-weight virtualization for smartphone devices [21], etc. In order to provide better support for Internetware OSs, we need to investigate on more general virtualization technologies for various devices and resources, as well as how to better optimize these systems.

2) Application Programming Interfaces (APIs): Application programming interface (API) provides a way to connect the different component within a software system. It can also be used to represent the programming interfaces provided by the operating system or function libraries. As the size the software system keeps increasing, we often need to split a complicated system into many smaller components, thus it becomes very important to design appropriate programming interfaces.

Tradition OSs are actually API providers themselves. An OS provides various libraries to applications running on it. An application typically requires calling the content of these libraries, or utilizing the hardware and resources, through the APIs provided by an OS. In the Internet era, Internetware rely heavily on APIs (or services) to access the storage and computing resources, which could reside locally or in a remote location.

The concept of *API economy* [22] has emerged in recent years, as the importance of APIs keeps increasing. Not only are APIs the interfaces between computer programs, they have also become the digital links for modern enterprises to provide services, applications and systems. API economy promotes open interfaces for data and computation, as well as numerous new applications based on these interfaces. From this angle, API economy promotes a type of Internetware OS that provides APIs and supports the development and execution of new types of applications.

3) Security and Privacy: In the Internet era, the openness and emergence of Internetware bring severe threats to security of privacy. More and more user data and computation have been moved to public cloud platforms such as Amazon AWS or AliCloud, thus introducing the potential leakage of privacy, as well as issues in data security [23]. In response to these threats, we need to investigate the following techniques:

- The security mechanisms for the Internetware OS infrastructure: how to ensure the confidentiality and integrity of data, how to ensure the security between the system layer and the application layer, etc.
- The data security and privacy protection for Internetware OS storage: how to reduce the safey risks of data, how to avoid the privacy risks, as well as privacy-related laws and regulations, etc.
- Identity authentication and access control for Internetware OSs: the trust boundary of, user management, mechanisms for identity authentication, access control models, etc.
- Security management of the application execution in Internetware OSs: including the security and availability management of cloud computing, security vulnerabilities, security protection, etc.

4) Customization: Due to the model of centralized computing, Internetware OSs might evolve into a management system comprising a huge number of resources and applications. However, for an OS oriented for end users, not all these resources are need because each user has different requirements. In addition, it is difficult for the users to find the specific resource they need because there are simply too many resources available. As a results, an Internetware OS needs to support personalized customization based on the huge repository of common resources. While it can support public software and services, it is also convenient to build an Internetware OS for public, enterprise, family or even an individual human being, all supported by the capability of personalized customization.

Componentization is software development mechanism aiming at automated software composition. Many researchers have proposed and developed various prototypes for componentized OSs in early years. In the Internet era, the diversity of Internetware introduces more challenges to the customization capabilities of Internetware OSs. We need to investigate different customization technologies including componentization techniques, in order to improve the adaptability of Internetware OSs for different application models and different user environments.

C. Our Practices

We have been researching and building an OS for Internetware that includes a set of software-defined features to abstract the low-level resource management functionalities of Internetware applications [24], [25]. Figure 3 shows the prototype architecture of our Internetware OS, which we regard as a prototype OS for future Internet-based applications. Within the Internetware OS, an Internetware application runs on top of the existing hardware systems including the cloud and edge devices. The Internetware OS core provides abstractions to manage both cloud and edge resources, while an application framework

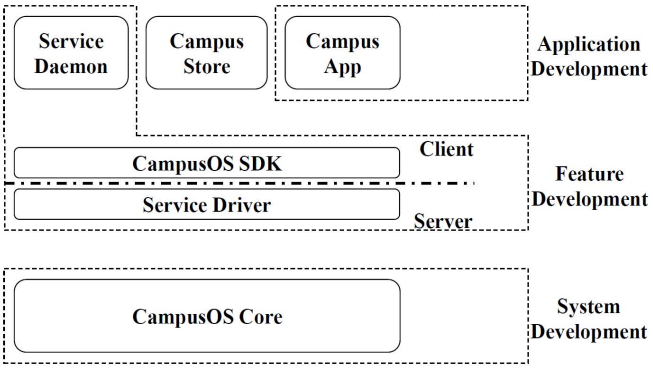

Figure 4. A prototype architecture for CampusOS. A prototype has been developed and deployed at Peking University to support the development and runtime management for various campus applications [27].

layer accommodates applications for different domains – for example, enterprise computing, mobile computing, and data as a service (DaaS).

1) YanCloud: As a cloud OS built for private cloud computing systems within an organization, YanCloud [26] is developed to support multiple types of virtualization technology, such that it is able to provide management of almost all existing virtual machines including Xen, VMWare and KVM, etc. One of the most prominent features of YanCloud is that it provides software-defined capabilities to generate cloud management applications with architecture-based model-driven runtime management.

YanCloud is designed as a middleware layer that provides management capabilities on different virtual machines, as well as programmability capability for cloud management applications. Thus it can be regarded as a UOS for cloud computing systems. YanCloud has been deployed in many business organizations, and has been installed as an OEM cloud OS for cloud servers from major manufacturers such as Lenovo and Founder.

2) CampusOS: We have also developed a prototype of CampusOS [27], which aims at providing OS-level support for campus applications within a university campus. CampusOS manages resources in a campus, including personal (student and faculty) information, course schedules, activity information, etc. It also provides abstractions to manage these resources, as well as software-defined APIs and SDKs to support campus application development and execution.

Figure 4 presents the hierarchical architecture of CampusOS. The bottom layer is the CampusOS Core and the underlying implementations of the whole system. This layer corresponds to the development of the CampusOS system. The middle layer consists of service drivers on the server side and the CampusOS SDK and service daemons on the client side. This layer corresponds to the development and extension of the CampusOS features. The top layer is based on the CampusOS SDK, including CampusStore (similar to app store) and campus applications.

We design CampusOS as an open, community-oriented operating system for university campuses. Just like traditional operating systems, CampusOS provides programming interfaces, libraries, and runtime management of applications, although at a higher level. Many components of CampusOS depend on traditional operating systems like Windows and Linux, as well as networked resources in the cloud and on the Internet.

CampusOS is intended to serve as an open platform for campus applications, which means that not only can developers publish new applications to CampusOS, but they can also add new features to CampusOS. CampusOS is designed with the principle of extensibility in mind, such that new features can be easily added into it. When some developers come up with interesting new features, they can choose to share the features with other developers by adding them to CampusOS. With more and more features added into CampusOS, more complex campus applications can be built, which will in turn attract more users and more developers, thus forming an ecosystem.

3) YanDaaS: Most recently, we have proposed and developed an instance of Internetware OS for data management and sharing between different types of legacy software systems. As its name implies, YanDaaS is an OS providing Data-as-a-Service functionalities.

The main goal of YanDaaS is to help connecting these isolated legacy software systems and applications through automated API generation and new application development without the source code of these legacy systems [28]. The data schemes and data access APIs of these legacy systems can be generated with a black-box runtime analysis mechanism, through computational reflection guided by software architectures. As a result, the recovered runtime software architecture can govern the structure and behavior of the legacy systems and enable their interoperability and integration with new systems.

YanDaaS can be regarded as a DaaS OS for legacy systems that provides software-defined data access and management capabilities. It has been successfully deployed to hundreds of industrial legacy systems covered by the national "Smart City" program in China.

III. Future Directions of Internetware OSs

We envision that some attractive Internetware OSs will emerge in the near future, which include:

- Enterprise OS: In future, we may need an OS to provide management capabilities for each and every organization. In order for an OS to manage a real-world organization, it should be able to support management of enterprise resources such as people, funds, and (real) machines, as well as providing support for (real) process management and output management.

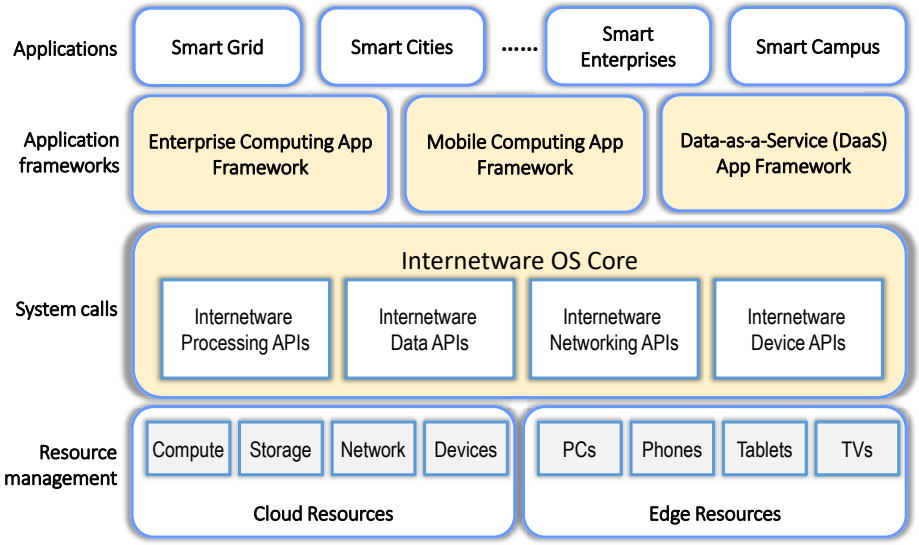

Figure 3. A prototype architecture for Internetware operating systems. Internetware applications run on top of the cloud and edge devices. The Internetware OS core provides abstractions to manage both cloud and edge resources, while an application framework layer accommodates applications for different domains.

Enterprise OSs can be built by reconstructing the current management systems (such as MIS or ERP) already deployed in many enterprises, while adding programming APIs to support flexible enterprise application development.

- OS for industrial control and manufacturing businesses: As a special type of enterprises, many manufacturers have long utilized software to control their manufacturing processes and pipelines. Many manufacturers have also deployed automated production systems as well as robotic control systems. Although many of these industrial control systems have been controlled with simple software layers such as embedded systems, new software-defined abstractions and communication capabilities will become inevitable as more intelligence and automation will be built into these industrial control systems in the near future. Building OSs for these manufacturers will help improve the efficiency and intelligence of their production control systems.
- OS for Human-Cyber-Physical convergence: There is a trend in software applications and systems where we have observed the convergence of three previously isolated domains: human beings, cyber systems, and the physical world. Human-cyber-physical convergence will bring many interesting applications beyond the current cyber-physical systems or Internet-of-Things. However, new software-defined abstractions and capabilities will be needed to support the management, application development, as well as communications within and between these human-cyber-physical systems.

IV. Concluding Remarks

With the generalization of the OS concept, we believe that OSs will become ubiquitous in the near future, resulting in many interesting ubiquitous OSs (UOSs). In the context of Internetware, we envision that Internetware OS, as an important type of UOSs, will emerge to support the efficient development and execution of new types of Internetware applications.

However, several key technical challenges still need to be resolved, including the architecture of Internetware OSs, performance and applicability issues, as well as security and privacy considerations. Nonetheless, we believe future Internetware OSs will keep emerging from new computing domains and beyond, including but not limited to, areas such as robotics, enterprise computing, manufacturing, big data, and artificial intelligence. Along this direction, our future work include developing Internetware OSs for new areas such as unmanned systems, manufacturing, as well as new computing models such as neuron computing.

Acknowledgments

This work was partly supported by the National Key Research and Development Program (No. 2017YFB1001904) and the National Natural Science Foundation of China (No. 61772042).

REFERENCES

[1] H. Mei and Y. Guo, "Network-oriented operating systems: Status and challenges," *Science China Information Sciences*, vol. 43, no. 3, pp. 303–321, mar 2013, (in Chinese).

[2] P. A. Bernstein, "Middleware:a model for distributed system services," *Communications of the Acm*, vol. 39, no. 2, pp. 86–98, 1996.

[3] P. W. Madany, S. Keohan, D. Kramer, and T. Saulpaugh, "Javaos: A standalone java environment," Sun Microsystems, Mountain View, CA, White Paper, 1996.

[4] S. Pichai and L. Upson, "Introducing the Google Chrome OS," The Official Google Blog 7, 2009.

[5] M. Quigley, B. P. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Y. Ng, "Ros: An open-source robot operating system," in *ICRA workshop on open source software*, 2009.

[6] M. Weiser, "The computer for the 21st century," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 3, no. 3, pp. 3–11, Jul. 1999. [Online]. Available: http://doi.acm.org/10.1145/329124.329126

[7] H. Mei and Y. Guo, "Toward ubiquitous operating systems: A software-defined perspective," *Computer*, vol. 51, no. 1, pp. 50–56, January 2018.

[8] F. Yang, J. Lv, and H. Mei, "Technical framework for internetware: An architecture centric approach," *:f*, vol. 51, no. 6, pp. 610–622, 2008.

[9] H. Mei, G. Huang, H. Zhao, and W. Jiao, "A software architecture centric engineering approach for internetware," *Science in China*, vol. 49, no. 6, pp. 702–730, 2006.

[10] H. Mei, G. Huang, and T. Xie, "Internetware: A software paradigm for internet computing," *Computer*, vol. 45, no. 6, pp. 26–31, June 2012.

[11] H. Mei and J. Lv, *Internetware: A New Software Paradigm for Internet Computing*. Springer, 2016.

[12] C. Dixon, R. Mahajan, S. Agarwal, A. J. Brush, B. Lee, S. Saroiu, and P. Bahl, "An operating system for the home," in *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 25–25.

[13] Living PlanIT SA, "Urban OS," http://living-planit.com/, 2017.

[14] D. G. Chandra and D. B. Malaya, "A study on cloud os," in *International Conference on Communication Systems and Network Technologies*, 2012, pp. 692–697.

[15] Google, "Android Things," https://developer.android.com/things/index.html, 2017.

[16] A. G. Olbert, *Extended control program support: VM/370: a hardware assist for the IBM virtual machine facility/370*. ACM, 1978.

[17] R. Nikolaev and G. Back, "Virtuos: an operating system with kernel virtualization," in *Twenty-Fourth ACM Symposium on Operating Systems Principles*, 2013, pp. 116–132.

[18] S. Angel, H. Ballani, T. Karagiannis, G. O'Shea, and E. Thereska, "End-to-end performance isolation through virtual datacenters," in *Usenix Conference on Operating Systems Design and Implementation*, 2014, pp. 233–248.

[19] M. D. Day, M. D. Day, Z. Dubitzky, M. Factor, N. Har'El, A. Liguori, A. Liguori, O. Wasserman, and B. A. Yassour, "The turtles project: design and implementation of nested virtualization," in *Usenix Conference on Operating Systems Design and Implementation*, 2010, pp. 423–436.

[20] F. Zhang, J. Chen, H. Chen, and B. Zang, "Cloudvisor: retrofitting protection of virtual machines in multi-tenant cloud with nested virtualization," in *ACM Symposium on Operating Systems Principles*, 2011, pp. 203–216.

[21] J. Andrus, C. Dall, A. V. Hof, O. Laadan, and J. Nieh, "Cells: A virtual mobile smartphone architecture," in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, ser. SOSP '11. New York, NY, USA: ACM, 2011, pp. 173–187. [Online]. Available: http://doi.acm.org/10.1145/2043556.2043574

[22] I. Gat, T. Remencius, A. Sillitti, G. Succi, and J. Vlasenko, "The api economy: Playing the devil's advocate," *Cutter IT Journal*, 2013.

[23] Z. Xiao and Y. Xiao, "Security and privacy in cloud computing," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 2, pp. 843–859, 2013.

[24] H. Mei, "Understanding "software-defined" from an os perspective: technical challenges and research issues," *Science China Information Sciences*, vol. 60, no. 12, p. 126101, 2017.

[25] H. Mei and Y. Guo, "Development and present situation of Internetware operating systems," *Science & Technology Review*, vol. 34, no. 14, pp. 33–41, 2016, (in Chinese).

[26] X. Chen, Y. Zhang, X. Zhang, Y. Wu, G. Huang, and H. Mei, "Towards runtime model based integrated management of cloud resources," in *Proceedings of the 5th Asia-Pacific Symposium on Internetware*, ser. Internetware '13. New York, NY, USA: ACM, 2013, pp. 1:1–1:10. [Online]. Available: http://doi.acm.org/10.1145/2532443.2532444

[27] P. Yuan, Y. Guo, and X. Chen, "Towards an operating system for the campus," in *Proceedings of the 5th Asia-Pacific Symposium on Internetware*, ser. Internetware '13. New York, NY, USA: ACM, 2013, pp. 24:1–24:4. [Online]. Available: http://doi.acm.org/10.1145/2532443.2532468

[28] G. Huang, X. Liu, X. Lu, Y. Ma, Y. Zhang, and Y. Xiong, "Programming situational mobile web applications with cloud-mobile convergence: An internetware-oriented approach," *IEEE Transactions on Services Computing*, vol. PP, no. 99, pp. 1–1, 2016.