Toward Ubiquitous Operating Systems: A Software-Defined Perspective

Hong Mei and Yao Guo, Peking University

In recent years, operating systems have expanded beyond traditional computing systems into the cloud, IoT devices, and other emerging technologies and will soon become ubiquitous. Despite the apparent differences among existing OSs, they all have in common so-called "software-defined" capabilities—namely, resource virtualization and function programmability.

n the January 2016 issue of *Computer*, Dejan Milojičić and Timothy Roscoe predicted what OSs would look like in a decade based on current hardware and application trends.[1] Whereas they focused on OSs for traditional computing systems such as PCs, servers, and embedded systems, we instead examine the future of OSs more broadly from a software-defined perspective.

In recent years, various OSs have been proposed and developed 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.

Resource virtualization and function programmability also lie at the heart of so-called "software-defined" systems including software-defined networks (SDNs),[2] software-defined storage (SDS), and software-defined datacenters (SDDCs). Just as a traditional OS manages a hardware system with software abstractions and provides runtime support for applications, we believe that future OSs will provide all of the software-defined capabilities for emerging technologies. Thus, an SDN is an OS for networking hardware, while a software-defined cloud is an OS for the cloud. We refer to these OSs as *ubiquitous operating systems* (UOSs).

A BRIEF HISTORY OF OPERATING SYSTEMS
An OS is a layer of system software that lies between applications and computer hardware, managing resources

TABLE 1. Evolution of traditional operating systems.			
Timeframe	Representative OS(s)	Computer system	Main characteristics
1956	GM-NAA I/O	IBM 704	The first practical OS Simple batch processing I/O management
1960s	IBM OS/360 series	IBM 360 series—mainframes	Time-sharing Multibatch processing Memory management Virtual machines (VM/370)
1970s	Unix	Minicomputers/workstations	First modern OS Developed with machine-independent languages (C) Provides standard interfaces Integrated development environment
1980s	Mac OS, Windows, Linux	Personal computers (PCs)	Provides modern GUI Improves usability for personal users
2000s	Apple iOS, Google Android, Windows Phone	Smartphones	Customization of traditional OSs Improves usability for mobile devices New app delivery model (App Store, Google Play)

such as processors, memory, and storage while providing support to the applications running above it.[3]

There were no OSs on the earliest computers, as software applications ran directly on bare-metal machines. However, as computing systems became increasingly complex, it became harder to manage the resources directly in an application. Consequently, more common functionalities were abstracted as drivers and libraries, creating a system software layer that could be shared among different applications. This software layer was called an "operating system" because it was originally developed to abstract a system's operating capabilities to ease the burden on operators. However, current OSs no longer emphasize system "operation" but instead handle resource management, application development, and runtime support for a given system.

Table 1 lists major traditional OSs and compares their key characteristics. Modern OSs mostly employ a Unix-based architecture, while customizing their functionalities for a particular type of system. For example, OSs such as Windows and macOS focus on GUIs to provide better user experiences for desktop users, while OSs such as Android and iOS include a layer to support mobile app development and execution to provide better experiences for mobile users.

With the rapid adoption of computer networks in the 1980s, it became critical for OSs to provide networking capabilities, leading to the creation of networking middleware and many new network OSs (NOSs). The earliest NOSs such as Novell Netware focused on connecting computers within a local network. These were later discontinued when connecting to the network became a necessity for many users and most such capabilities were incorporated in newer versions of desktop OSs.

Talk of an Internet OS began in the mid-1990s during the war between Microsoft and Netscape, which announced a set of new tools and programming interfaces for next-generation Internet-based applications. Since then, many Internet OS implementations have been proposed, including the Java-based JavaOS and, most recently, Google's browser-based Chome OS.

Many NOSs and Internet OSs offer networking capabilities or incorporate Internet-related data-management functionalities with OS-like structures. Their components might run on geographically distributed computer systems or even virtual machines (VMs), providing specific services through Internet connections. These new "meta-OSs" often run above traditional OSs such as Windows or Linux to provide support for Internet-based applications and services.

A SOFTWARE-DEFINED PERSPECTIVE

"Software-defined" has become one of the hottest buzzwords in both academia and industry. It describes a family of technologies including SDNs, SDS, and SDDCs that are collectively sometimes referred to as "software-defined everything" (SDX). In a software-defined system, hardware resources can be virtualized and managed by OS routines or the control plane, and users can write programs to access and manage the services provided by virtualized resources.[4]

We argue that OSs offer these same capabilities. For example, a traditional OS such as Linux or Windows provides virtualization of hardware resources through hardware drivers, and application development and runtime support through software development kits (SDKs) and libraries.
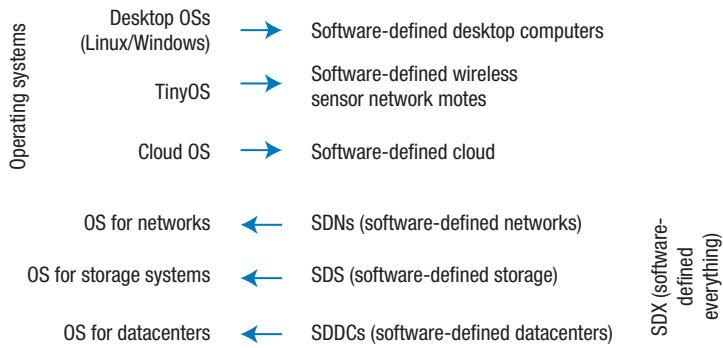
FIGURE 1. Operating systems and software-defined systems are mostly based on the same principles.

Mobile OSs such as Android provide an extra software-defined layer with higher-level abstractions for mobile apps, including management of mobile data (for example, contact and location data), a set of APIs for app development, and a set of libraries to support app execution. Whether OSs run on small devices (such as TinyOS[5]) or huge clusters (such as cloud OSs), they offer both resource virtualization and function programmability. Put another way, a "software-defined" technology is really just an OS for that technology. In an SDN, for example, the control plane provides the ability to write high-level applications to manage the networking functions, while the data plane virtualizes networking resources.

In sum, as Figure 1 shows, OSs and software-defined systems are mostly based on the same principles.

UBIQUITOUS OPERATING SYSTEMS

More than a quarter of a century ago, Mark Weiser envisioned a world in which computing was ubiquitous.[6] His prediction seemed unrealistic at the time but is becoming a reality with the emergence of the IoT and the programmability of everyday objects—for example, smart lights that sense the environment and brighten or darken accordingly. We likewise argue that OSs will become ubiquitous.

UOSs constitute a new type of OS for a software-defined world where software will be used to manage all aspects of our lives. To understand the enormous impact UOSs will have, consider these examples:

› *Web OSs*. Web OSs, also known as web desktops or webtops, provide a Linux-like environment within a browser for users to run applications and manage all their data and storage. They also include APIs for developers to create applications that can run within the browser. Example web OSs include Firefox OS, Chrome OS, eyeOS, YouOS, and G.ho.st.

› *The Robot Operating System*. ROS is a meta-OS that provides development and runtime support for complex and robust robotic applications.[7] Its extensive collection of open source tools, libraries, abstractions, and APIs can be used across a wide variety of platforms.

› *HomeOS*. A Microsoft initiative to enable "smarter homes for everyone," HomeOS aims to simplify the creation and management of home automation technology.[8] It provides both intuitive user controls and higher-level abstractions for device orchestration. Research prototypes of HomeOS have been deployed in more than a dozen homes.

› *City OSs*. There are many initiatives to create OSs to facilitate growth, energy use, and environmental sustainability.

For example, the Living PlanIT Urban Operating System (living-planit.com) provides abstractions and management interfaces for energy, water, waste management, transportation, telecommunications, and healthcare systems, as well as programming APIs to ensure interoperability among different platforms.

› *Cloud OSs*. Conceptually, a cloud OS does what a traditional OS does—manage applications and hardware—but at the scale of cloud computing, replacing file systems with object storage and enabling almost unlimited storage capacity and I/O throughput. Instead of managing processes on physical machines, a cloud OS manages tasks on VMs. More importantly, it offers various APIs for cloud apps to utilize cloud resources. Many cloud service providers have created their own cloud OS, including Microsoft Azure, Amazon Web Services (AWS), and Huawei FusionSphere. There are also popular open source cloud OSs such as Open-Stack and Apache CloudStack.

› *IoT OSs*. Google's Android Things (Brillo) is an embedded OS platform designed for low-power and memory-constrained IoT devices that uses Android APIs and Google Services.

Figure 2 shows a general architecture for UOSs, which is similar to that of traditional OSs. UOSs embody the same key concepts as traditional OSs—resource virtualization and function programmability—but these concepts are more generally defined for ubiquitous scenarios:
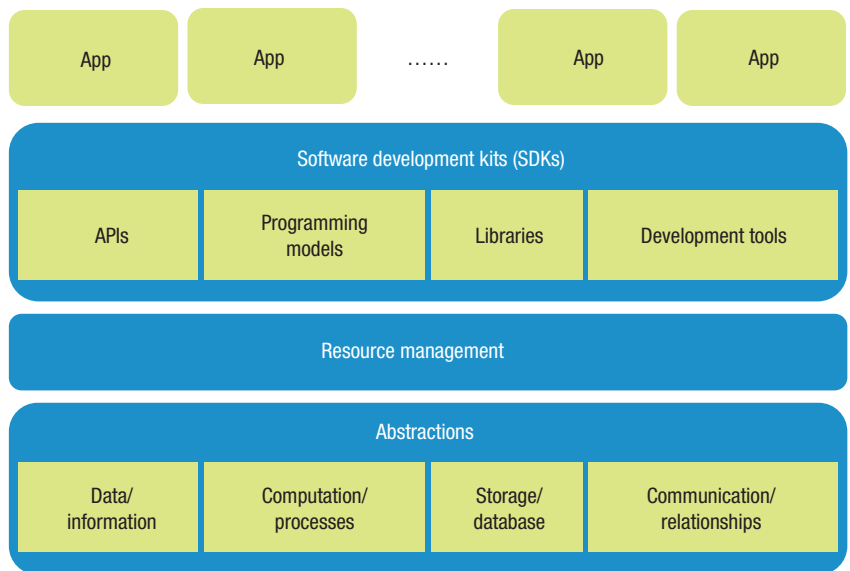
FIGURE 2. General ubiquitous operating system (UOS) architecture. A UOS provides abstractions to manage hardware and software as well as resource virtualization along with programming and runtime support for applications, especially those created by third-party developers.

❯ *Abstractions for resource management.* A UOS provides abstractions to manage various resources beyond traditional computing and storage resources. These function much like drivers or hardware abstraction layers in traditional OSs but enable resource virtualization more generally. APIs are also provided for users and applications to access these virtualized resources. For example, a UOS for social networks manages user information and relationships, as well as tracks user actions and communications between users.

❯ *Development and runtime support for ubiquitous applications.* A UOS provides APIs, programming models, libraries, and development tools for applications like a traditional OS. However, this support is at a higher level, as ubiquitous applications run atop the UOS, which in turn runs above traditional OSs such as Linux and Windows. The key difference is that UOSs support apps by third-party developers, whereas existing non-OS solutions are typically implemented as a proprietary layer on a system.

UOS OUTLOOK

As Figure 3 shows, we envision UOSs for many different entities, both real and virtual, as well as traditional IT systems.

UOS principles

Underlying this vision are three basic principles.

UOSs can be scaled to any size system. OSs have already been created for small embedded systems and mobile devices such as smartphones and tablets, traditional desktop PCs and laptops, standalone workstations and networked servers, and server clusters and clouds. We foresee OSs being extended to include almost all legacy and next-generation systems, from tiny edge computing devices to huge distributed computing environments that span continents. UOSs can also be built for emerging application domains such as big data and artificial intelligence.

A UOS can be built for every object (or collection of objects) in the physical world. The goal of ubiquitous
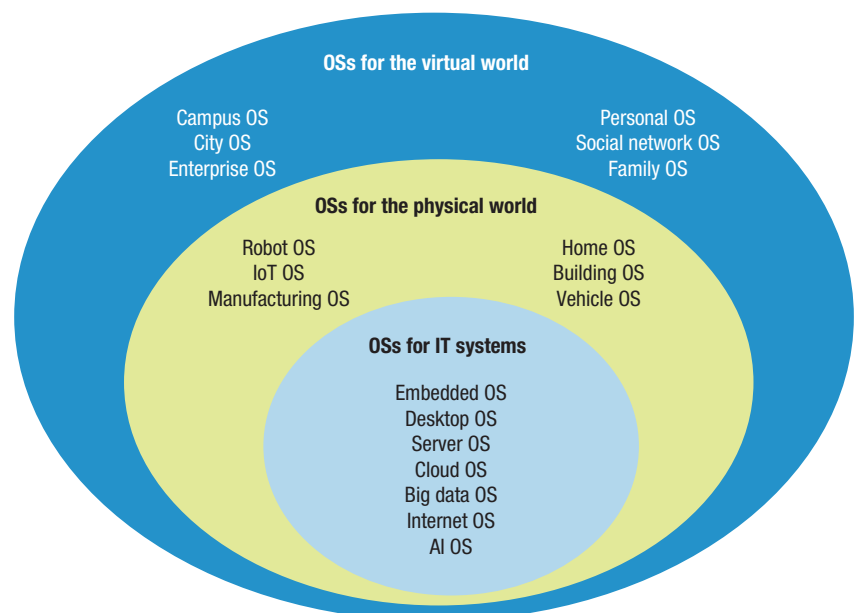

FIGURE 3. Different categories of UOSs for real and virtual entities as well as traditional IT systems.

computing is to expand computation capabilities beyond traditional IT systems to make all objects smarter. That will eventually mean making these entities programmable, which will require an OS. Robots (even Lego robots) already have OSs. In a smart home, all appliances—including TVs, washing machines, refrigerators, lights, microwave ovens, and coffee machines—will need an OS to become programmable. Every moving object including vehicles, drones, bicycles, wheelchairs, and even strollers will need a UOS as well.

A UOS can be built for each entity in the virtual world. In addition to physical objects and systems, OSs could also be created for entities in specific application domains. For example, organizations of various types and sizes including families, enterprises, institutions, and government agencies could be equipped with software-defined capabilities to manage personnel, information, schedules, and inventory. OSs would provide abstractions to manage resources as well as support for the development and execution of new applications.

UOS categories

Given these principles, we can expect to see many different categories of UOSs, for example:

- *Big data OS*. Big data applications have been built for a wide variety of domains. A big data OS could provide special functions for data abstraction and management, data access and management APIs, and programming models and languages for big data applications.
- *Enterprise OS*. Future enterprises or organizations might need an OS to support the efficient management of processes as well as resources including people, funds, and machines. Enterprise OSs could be created out of existing enterprise systems, such as management information systems (MISs) or enterprise resource planning (ERP) systems, by adding programming APIs to support flexible enterprise application development.
- *Industrial/manufacturing OS*. Many manufacturers have already deployed automated production and robotic control systems. Although many of these systems have been managed with simple embedded systems, new software-defined abstractions and communication capabilities will improve the systems' efficiency and intelligence.
- *Human-cyber-physical OS*. An emerging trend in computing is the convergence of three previously isolated domains: human beings, cyber systems, and the physical world. This will bring many interesting applications beyond current cyber-physical systems and the IoT. However, new software-defined abstractions and capabilities will be required to support human-cyber-physical system management, application development, and communications.
- *Artificial intelligence OS*. An OS will be needed to provide abstractions for machine learning or deep learning capabilities, as well as programming support for AI applications. Android cofounder Andy Rubin recently predicted that AI would be the next major OS breakthrough.[9] An AI OS will become essential infrastructure to the success of new types of intelligent applications.

Technical challenges

Despite their promise, UOSs present numerous technical challenges.

- *UOS models and architectures*. A generic UOS model and architecture likely will not be suitable for all UOSs. The most important factor is the granularity of abstractions and programming interfaces. Smaller granularity enables more flexibility, but at the potential cost of application runtime performance. Determining this tradeoff will be central to UOS architecture design.
- *Resource virtualization*. Virtualization is the key technology enabling all OSs and SDX. With UOSs, computing will be pushed from the central cloud to the edge, such as in smartphones and IoT devices. Thus, we need to investigate lightweight virtualization technologies to provide efficient OS abstractions and support software-defined edge computing.
- *Performance optimization*. In UOSs for small-scale computer systems or objects with weak computing capabilities, improving application execution performance will become critical. As more types of hardware, resources, and applications emerge, it will be a challenge to provide efficient services, especially for high-throughput and massively parallel scenarios.
- *Security and privacy*. Software is more vulnerable to security

FIGURE 4. Internetware OS architecture. 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.

threats than hardware. With a UOS in place, software becomes the control center of a system or environment, making it the main target of attackers. In addition, for the UOSs of systems that manage sensitive personal data or critical information, privacy will also become a first-order consideration.

› *Domain-specific programming languages.* Current high-level programming languages such as C/C++ and Java are designed with computers in mind. New domain-specific languages will be needed to develop more efficient apps for particular UOSs—for example, for an enterprise OS.

› *Achieving true intelligence.* Software is the basis for all intelligent applications. To achieve true intelligence, UOSs as well as applications must be able to "think"—to manage and execute intelligently.

INTERNETWARE OS: A PROTOTYPE UOS

Internetware is a paradigm for new types of Internet applications that are autonomous, cooperative, situational, evolvable, and trustworthy.[10,11] Internetware consists of a set of autonomous software entities distributed over the Internet, together with a set of connectors to enable various collaborations among these entities. Software entities sense dynamic changes in the runtime environment and continuously adapt to them through structural and behavioral evolutions.

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
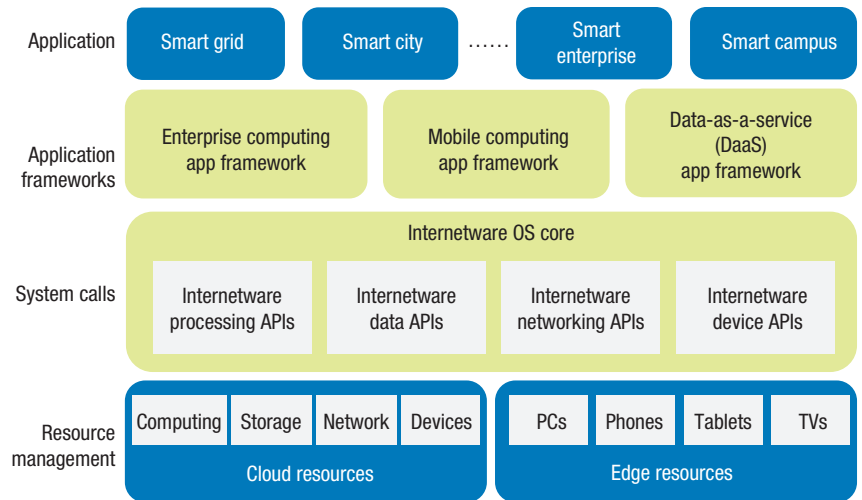
of Internetware applications.[12] Figure 4 shows the general architecture of our Internetware OS, which we regard as a prototype UOS for future Internet-based applications. Within the OS, an Internetware application runs on top of the existing hardware systems including the cloud and edge devices. The Internetware OS core provides abstractions to manage both cloud and edge resources, while an application framework layer accommodates applications for different domains—for example, enterprise computing, mobile computing, and data as a service (DaaS).

Examples of Internetware OS instances we have built include the following:

› *YanCloud.* A cloud OS for private cloud computing systems within an organization, YanCloud[13] supports almost all existing VM technologies including Xen,

VMware, and KVM. It features software-defined capabilities to generate cloud management applications with architecture-based, model-driven runtime management mechanisms. YanCloud has been deployed by many businesses as well as major cloud server manufacturers such as Lenovo and Founder.

› *CampusOS.* A prototype OS to support Internet-based applications at a university campus, CampusOS[14] manages resources including student and faculty personal information, course schedules, and school activities. It also provides abstractions to manage these resources, as well as software-defined APIs and SDKs to support campus application development and execution.

› *YanDaaS.* Most recently, we developed an Internetware OS for data management and sharing among

ABOUT THE AUTHORS

HONG MEI is a professor and director of the Chinese Ministry of Education (MOE) Key Laboratory of High-Confidence Software Technologies at Peking University as well as vice president of the Beijing Institute of Technology. His research interests include systems software and software engineering. Mei received a PhD in computer science from Shanghai Jiao Tong University. He is a member of the Chinese Academy of Sciences and a Fellow of IEEE, the China Computer Federation, and the World Academy of Sciences. Contact him at meih@pku.edu.cn.

YAO GUO is a professor at the Chinese MOE Key Laboratory of High Confidence Software Technologies and in the School of Electronics Engineering and Computer Science at Peking University. His research interests include mobile computing, operating systems, and mobile application analysis. Guo received a PhD in computer engineering from the University of Massachusetts Amherst. He is a member of IEEE and ACM. Contact him at yaoguo@pku.edu.cn.

different types of legacy software systems. As its name implies, YanDaaS provides DaaS functionalities. Its main goal is to connect isolated legacy software systems and applications through automated API generation and new application development without legacy source code.[15] YanDaaS has been successfully deployed within hundreds of industrial legacy systems covered by China's Smart City program.

With rapid IoT development, many UOSs will emerge to provide software-defined capabilities, especially resource virtualization and function programmability, to support the efficient deployment and management of new types of ubiquitous applications. However, several key technical challenges still must be resolved with respect to UOS architecture, system performance, and security and privacy. Nonetheless, we foresee UOSs appearing in various computing and computer-assisted domains including robotics, enterprise computing, manufacturing, big data, and AI. Toward this end, our future work includes developing Internetware OSs for new areas such as unmanned systems, industrial control, and brain-like computing. ▣

REFERENCES

1. D. Milojičić and T. Roscoe, "Outlook on Operating Systems," *Computer*, vol. 49, no. 1, 2016, pp. 43–51.
2. N. McKeown et al., "OpenFlow: Enabling Innovation in Campus Networks," *ACM SIGCOMM Computer Communication Rev.*, vol. 38, no. 2, 2008, pp. 69–74.
3. H. Mei and Y. Guo, "Network-Oriented Operating Systems: Status and Challenges," *Scientia Sinica Informationis*, vol. 43, no. 3, 2013, pp. 303–321 (in Chinese).
4. H. Mei et al., "Understanding Software-Defined from the Perspectives of Software Researchers," *Comm. China Computer Federation*, vol. 11, no. 1, 2015, pp. 68–72 (in Chinese).
5. P. Levis et al., "TinyOS: An Operating System for Sensor Networks," *Ambient intelligence*, W. Weber, J. Rabaey, and E. Aarts, eds., Springer, 2005, pp. 115–148.
6. M. Weiser, "The Computer for the 21st Century," *Scientific Am.*, vol. 265, no. 3, 1991, pp. 94–105.
7. M. Quigley et al., "ROS: An Open-Source Robot Operating System," *ICRA Workshop Open Source Software*, vol. 3, no. 3.2, 2009; www.willowgarage.com/sites/default/files/icraoss09-ROS.pdf.
8. C. Dixon et al. "An Operating System for the Home," *Proc. 9th USENIX Symp. Networked Systems Design and Implementation* (NSDI), 2012, pp. 25–25.
9. T. Haselton, "The Man behind Android Says A.I. Is the Next Major Operating System," CNBC, 18 Aug. 2017; www.cnbc.com/2017/08/18/andy-rubin-says-ai-is-next-big-operating-system.html.
10. H. Mei, G. Huang, and T. Xie, "Internetware: A Software Paradigm for Internet Computing," *Computer*, vol. 45, no. 6, 2012, pp. 26–31.
11. H. Mei and J. Lü, *Internetware: A New Software Paradigm for Internet Computing*, Springer, 2016.
12. H. Mei and Y. Guo, "Development and Present Situation of Internetware Operating Systems," *Science & Technology Rev.*, 2016, vol. 34, no. 14, pp. 33–41 (in Chinese).
13. X. Chen et al., "Towards Runtime Model Based Integrated Management of Cloud Resources," *Proc. 5th Asia-Pacific Symp. Internetware* (Internetware 13), 2013, article no. 1.
14. P. Yuan, Y. Guo, and X. Chen, "Towards an Operating System for the Campus," *Proc. 5th Asia-Pacific Symp. Internetware* (Internetware 13), 2013, article no. 24.
15. G. Huang et al., "Programming Situational Mobile Web Applications with Cloud-Mobile Convergence: An Internetware-Oriented Approach," *IEEE Trans. Services Computing*, 2016; doi:10.1109/TSC.2016.2587260.