Software Engineering Meets Services and Cloud Computing

Stephen S. Yau and Ho G. An, *Arizona State University*

Service-oriented software engineering incorporates the best features of both the services and cloud computing paradigms, offering many advantages for software development and applications, but also exacerbating old concerns.

Services and cloud computing have garnered much attention from both industry and academia because they enable the rapid development of large-scale distributed applications in areas such as collaborative research and development, e-business, healthcare, grid-enabled applications, enterprise computing infrastructures, military applications, and homeland security. These computing paradigms have made it easier and more economical to create everything from simple commercial software to complex mission-critical applications.

The two paradigms share concepts, such as resource outsourcing and transfer of IT management to service providers, but their emphasis on software engineering differs. Services computing focuses on architectural design that enables application development through service discovery and composition. Cloud computing focuses on the effective delivery of services to users through flexible and scalable resource virtualization and load balancing.

Service-oriented software engineering[1] incorporates the best of these two paradigms. Initially, SOSE was based on services computing, but it evolved to include cloud computing. In SOSE, a service-oriented architecture (SOA) provides the architectural style, standard protocols, and interfaces required for application development, and cloud computing delivers the needed services to users through virtualization and resource pooling. Combining services and cloud computing in a software engineering framework can help application developers and service providers meet the individual challenges of each paradigm.

Although SOSE is conceptually promising, its realization will require additional research in software engineering to address the challenges, such as security and quality-of-service (QoS) management, that arise in services or cloud computing.

SERVICES COMPUTING

Service developers follow SOA, an architectural model for creating and sharing computing processes, packaged as services.[2] Each service is an independent software entity with a well-defined standard interface that provides certain functions over networks. Developers can dynamically compose services as a workflow, which forms the basis of an application. In this context, software itself can be a service—a self-contained, stateless, and platform-independent entity with a URL, an interface, and functions that can be described and discovered as XML data.

Different organizations with different policies develop, manage, and govern services. Service-level agreements specify runtime requirements that govern a service's interactions with a user or with other services. A service's SLA describes that service and sets forth the terms, in es-

0018-9162/11/$26.00 © 2011 IEEE Published by the IEEE Computer Society

Table 1. Protocols and interfaces that enable SOSE.

Protocol or interface	Standards body (if applicable)	Purpose
XML	W3C	Represent data in SOA
Simple Object Access Protocol (SOAP)	OASIS	Invoke services remotely across networks and platforms
Representational State Transfer (REST)	Architectural style, not a standard (attributed to Roy Fielding)	Invoke services remotely across networks and platforms
Web Services Description Language (WSDL)	W3C	Describe interfaces and service functions
Universal description, discovery, and integration (UDDI)	OASIS	Automatically publish and discover services
Electronic Business XML (ebXML)	OASIS and United Nations Center for Trade Facilitation and Electronic Business (UN/CEFACT)	Automatically publish and discover
Business Process Execution Language (BPEL)	OASIS	Orchestrate services in a workflow

sence becoming a service contract that service providers must fulfill.

Using standard protocols and interfaces, application developers can dynamically search, discover, compose, test, verify, and execute services in their applications at runtime. SOA-based application development is through service discovery and composition, which involves three stakeholders:

- A *service provider* (or developer) is the party who develops and hosts the service.
- A *service consumer* is a person or program that uses a service to build an application.
- A *service broker* helps service providers publish and market their services and helps service consumers discover and use the available services.

Application developers need not integrate service code into applications because the service runs at its provider's site and is loosely coupled with applications through standard messaging protocols. Consequently, services and applications do not have to be in the same programming language or run on the same platform. Unlike an application, which provides a user interface, a service typically provides an application programming interface (API) so that an application or other services can invoke that service.

As this description implies, services have several attractive characteristics. They are

- *loosely coupled*—there are no direct dependencies among individual services;
- *abstract*—beyond the SLA description, a service hides its logic from the outside world;
- *reusable*—services aim to support potential reuse;
- *composable*—a service can comprise other services, and developers can coordinate and assemble services to form a composite;
- *stateless*—to remain loosely coupled, services do not maintain state information specific to an activity, such as a service request; and
- *discoverable*—services let a service consumer use mechanisms to discover and understand their descriptions.

When taken together, these characteristics empower the rapid development of applications in services computing.

Standards bodies, such as the Organization for the Advancement of Structured Information Standards (OASIS) and the World Wide Web Consortium (W3C) have established a variety of protocols and service interfaces that enable application development using SOA. Table 1 gives a sampling of these protocols and interfaces.

CLOUD COMPUTING

Cloud computing enables convenient, on-demand network access to a shared pool of configurable computing resources, such as networks, servers, storage, applications, and services, which the cloud system can rapidly provision and release automatically. Cloud computing lets a consumer (user or program) request computing capabilities as needed, across networks anytime, anywhere. Some researchers envision the future Internet as a kind of supercomputer that will depend heavily on cloud computing features, such as resource pooling and virtualization, on-demand service, and ubiquitous access.

There are four cloud types. A *public cloud* provides services to the public. A *private cloud* provides services to only users within one organization. A *community cloud* provides services to a specific community of organizations and individuals. A *hybrid cloud* is any combination of the first three types.

As Figure 1 shows, the cloud computing architecture has three layers: software as a service (SaaS), platform as a service (PaaS), and infrastructure as a service (IaaS). Developers can implement and use each layer as a service.

Software as a service

Software that performs various tasks is not on the client machine. Instead, third-party service providers host and manage the software services in the cloud. SaaS includes both software components (for application developers) and applications (for users). An SaaS application is often a service-oriented program so that it is easy to integrate with other SaaS applications.

Platform as a service

PaaS provides a development platform with services to assist application design, implementation, testing, deployment, monitoring, and hosting in the cloud. It requires no software download or installation and supports geographically distributed collaborative work.

Infrastructure as a service

IaaS virtualizes the data centers' computing power, storage, and network connectivity. Users can scale these computing resources up and down on demand.

APPLICATION DEVELOPMENT WITH COMBINED PARADIGMS

Services computing and cloud computing are two separate paradigms, and each provides many advantages for software development and application. Application developers can use services computing alone, cloud computing alone, or a combination of the two. We believe that combining these two paradigms in a software engineering framework will help alleviate some of the software engineering challenges that services and cloud computing have individually. For example, a major challenge of services computing is to manage the runtime QoS of loosely coupled services involving distributed service providers. Cloud computing can help meet that challenge through resource allocation and virtualization.

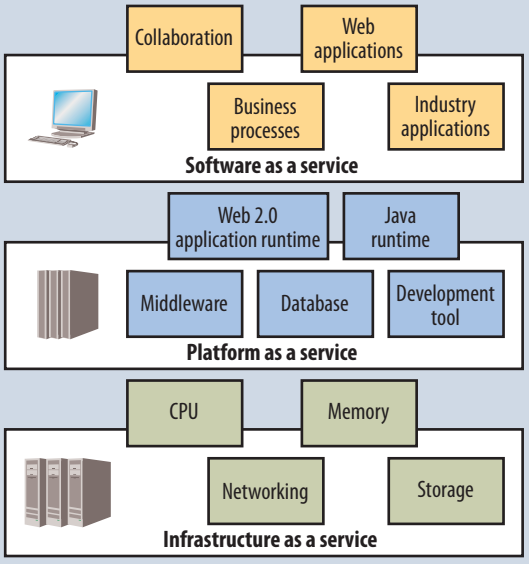

Figure 1. Cloud computing architecture. The top layer allows users and application developers to access services that third parties host and manage. The second layer consists of computing platforms and development services, and the third layer provides the computing resources that users can tap on demand.

On the other hand, cloud computing struggles both with providing interoperability across different clouds and with the rapid development of, and adaptation to, ever-changing business environments and requirements. SOA's standard interfaces and protocols could help address this interoperability challenge, while its dynamic service discovery and composition can provide the capabilities needed for dynamic adaptation in cloud computing environments.

Figure 2 shows the concept of developing applications using SOA and delivery through the cloud. Service providers could publish SaaS, PaaS, IaaS, and software

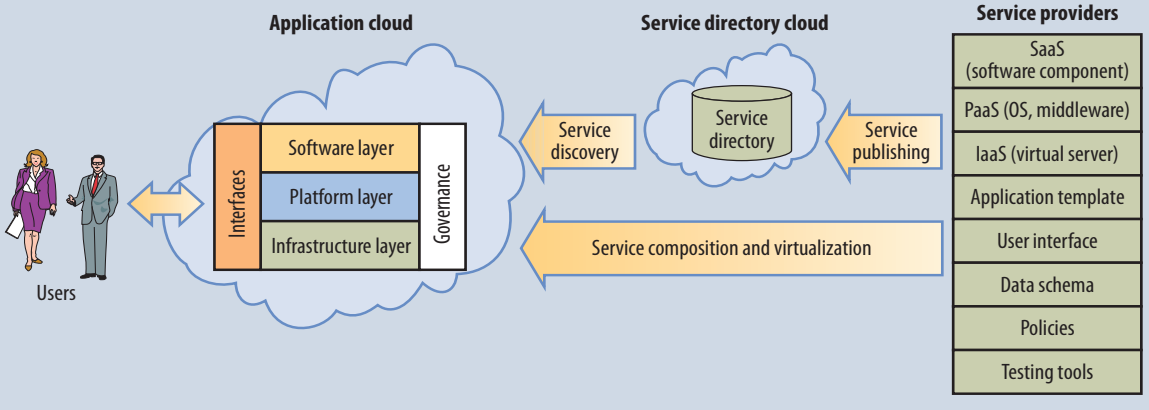

Figure 2. The SOSE vision. Service providers publish their services to a federated cloud directory. Application developers discover and compose the services to build an application, which is delivered to users through the cloud.

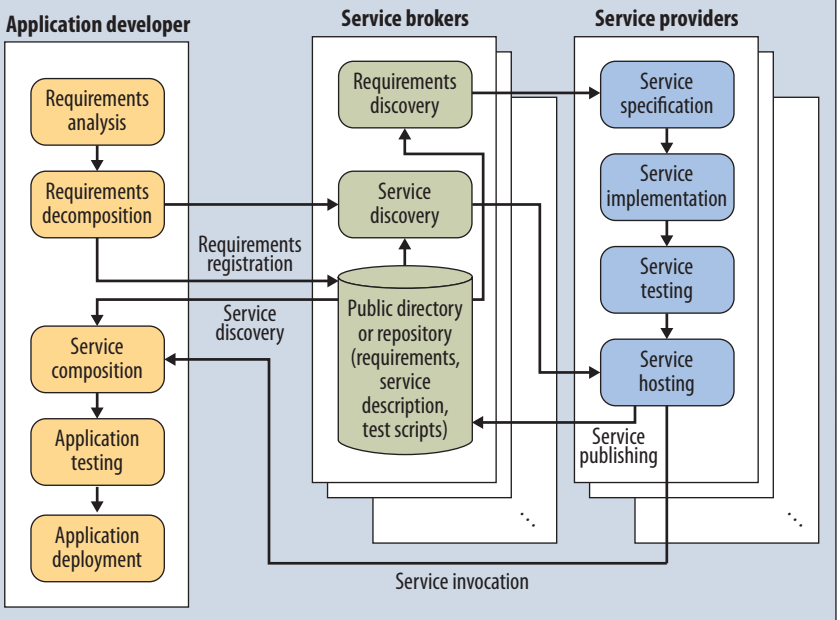

Figure 3. Interactions among an application developer, service brokers, and service providers in SOSE. The challenge for developing applications using SOA is to cope with the distributed nature of both the stakeholders and their activities, which can be in different organizations and locations.

artifacts, such as application templates, user interfaces, data schema, policies (including security policies), and testing tools in a service directory cloud. This federated service directory cloud could enable application developers to dynamically discover services in multiple distributed servers and compose these services using SOA and virtualization technologies.

Development with a service-oriented architecture

SOSE applies SOA to software development life-cycle stages, producing a cycle that includes not only the traditional requirements specification, design, and test phases, but also service implementation, discovery, and composition. Application development in SOA is different from software development approaches such as object-oriented programming, component-based software development, and aspect-oriented programming. The construction of an application from smaller software components in other software development methodologies is static and manual and depends on the components' technology and platform. In contrast, service composition in SOA is automated through standard protocols and interfaces, and thus does not depend on a specific technology or platform. In addition, service development, service publishing, and service composition (or application building) are parallel processes in SOA.

The challenge for developing applications using SOA is in addressing its distributed nature. Not only are the services under development distributed among different machines in various locations, but the development process is also distributed because the application developers, service brokers, and service providers work independently in different locations. Hence, these three stakeholders must collaborate through well-defined standards and interfaces. Figure 3 shows some key tasks and interactions among these individuals.

As in other software development methodologies, SOSE starts with requirements engineering. During this phase, the application developer develops a business model; works with the customer to analyze, clarify, and refine requirements; designs a workflow for the business model; and decomposes requirements.

The application developer then sends each decomposed part of the requirements to the service broker to find available services that satisfy these requirements parts. After successfully discovering all the needed services to satisfy each part, the application developer selects the needed services for all requirements parts and composes them into an application, essentially the business model workflow.

If no services are available for some parts, the application developer can register them in the service broker's directory and wait until the needed services are available. From the service providers' view, service development is similar to what happens in other software development processes, except that services must also comply with standard protocols and interfaces.

Software development in SOSE is highly flexible because SOA makes it possible to publish and reuse not only software services, but also numerous application development artifacts. Application developers can publish business models, application templates (workflow structures), requirements, services, application interfaces, testing tools, testing scripts, and policies in a service broker's directory, making them available for reuse. This flexibility facilitates the rapid development of large-scale distributed applications.

Delivery through cloud computing

Software engineering must address not only the software development processes, but also the effective delivery of the developed software to users, which includes software deployment and maintenance. However, SOA does not address how a developed application is to be delivered to users or how service providers will effectively manage

the applications during runtime. Cloud computing can help SOSE ensure effective application delivery by providing

- easy application deployment and maintenance for service providers through service virtualization,
- interfaces to facilitate users' access to and use of applications, and
- QoS management for service providers through dynamic resource virtualization and allocation.

These features illustrate the power of combining SOA-based development with cloud computing delivery.

APPLICATION DEVELOPMENT CHALLENGES

Achieving the vision of application development that Figure 2 depicts requires new approaches to effective virtualization and interoperability among SaaS, PaaS, and IaaS. It also requires revisiting software engineering issues, some of which are not new, but they are more severe in the context of services and cloud computing. We have identified seven areas that pose major challenges for application development using SOSE.

Confidentiality and integrity

In cloud computing, users have little control over data processing and storage, which is on remote machines that various service providers own and operate. Because this data is unencrypted, there is a risk that service providers or malicious users could disclose or alter it. Although techniques exist for confidentiality protection, they are not applicable to services and cloud computing systems because they are designed to protect data from malicious parties outside the systems. Services and cloud computing systems have many service providers inside the systems.

Thus, existing techniques for access control, identity management, end-to-end data confidentiality, and integrity assurance systems are not suitable. Although research is already addressing software engineering techniques for data confidentiality and integrity protection for services and cloud computing systems,[3,4] more work is needed in this area.

Service reliability and availability

Because users' businesses rely heavily on third-party service providers, there are serious concerns about how threats to service reliability and availability—from a service provider's unstable economic status to natural disasters and cyberattacks—could affect a service and consequently a cloud user's business. To alleviate these threats, service and cloud users should check their data backup plan, system robustness, contingency and recovery plans, end-of-service support, and incident history before using a particular service.

Cyberattacks are a particularly serious threat. Services and cloud computing systems rapidly and flexibly provide massive computing resources according to users' demands. For the users, the computing capabilities and resources often appear to be unlimited, since they are available for purchase at any time and in any quantity. However, cyberattackers also can buy huge amounts of computing resources, enabling them to launch more powerful cyberattacks. Attackers have already used the Amazon EC2 and Google AppEngine clouds, for example.[5,6]

To address this problem, services and cloud service providers need effective software engineering techniques to monitor and detect malicious user activities, as well as for strict user authentication and access control.

Security in a multitenant environment

In multitenancy a single software instance runs on a server that accommodates multiple users, or tenants. In a multitenant architecture, an application virtually partitions its data and configuration, and each user works with a customized virtual application instance.

Because users' businesses rely heavily on third-party service providers, there are serious concerns about how threats to service reliability and availability could affect a service and consequently a cloud user's business.

Services and cloud computing systems have multitenancy because multiple users share the application and a set of hardware. Security vulnerabilities are a major issue. Service providers use hypervisors that mediate access between virtual machines and hardware, but some hardware, such as CPU caches and GPUs, is not designed to offer strong isolation properties for a multitenancy architecture. Even virtual machine hypervisors can have flaws that allow one user's virtual machine to gain inappropriate control over others.[7] Recently, attackers have exploited numerous hypervisor vulnerabilities to influence other users' operations or to gain unauthorized data access.

Addressing these vulnerabilities requires developing software engineering techniques for securing multitenancy architectures in services and cloud computing systems, such as techniques for isolating and monitoring virtual machines.

Unknown risk profile

In services and cloud computing systems, users have limited access to information about the internal system architecture, software versions, configurations, operations, and security practices of service providers. This

limited access might enhance usability, but it also has serious implications for risk management. Risk management in software engineering ensures that the application developers identify and analyze threats to the application development process and that they use appropriate strategies to mitigate and control risks, such as failing to complete projects within the specified schedules and budget constraints and not meeting user requirements.

Because application developers lack information about the internal systems beneath the virtualized abstraction layer, they might not be able to conduct appropriate risk management. To address this problem, developers should ask service providers for three items:[8]

- partial or full disclosure of software design or infrastructure details;
- disclosure of applicable logs and data, such as network intrusion logs, anomaly detection logs, and security events logs; and
- disclosure of details of security policies and enforcement mechanisms.

Having these items will not eliminate risk, but the information should lead to much more effective risk management.

Quality-of-service monitoring

In services and cloud computing systems, managing a variety of QoS requirements is extremely difficult because numerous application developers are dynamically composing services over networks to form multiple workflows, and various providers with different techniques and policies are managing the services. Consequently, the QoS features of all services are tightly interrelated, and there are tradeoffs among them.

Features like throughput and service delay rely on system resource allocation at the applications' runtime. Often, the same server hosts multiple services, which compete for the server's CPU time, memory, and network bandwidth. In addition, service compositions, server resource status, workflow priorities, and QoS requirements are usually changing dynamically at runtime.

For these reasons, satisfying the QoS requirements of multiple workflows requires having effective techniques to adaptively allocate system resources to each service. To manage multiple QoS properties for such systems, services and cloud computing systems need situational awareness, context analysis and QoS estimation, and optimal resource allocation.[9-11]

Mobile computing

Services are available over networks, and users or programs on a range of devices—desktops, laptops, smartphones, tablets, and PDAs—can access the services on the networks at any time or in any location through standard protocols. Because identity theft and service hijacking are major threats, mobile services and cloud computing providers need rigorous software engineering techniques to secure ubiquitous access to services and data.

Legal issues

Those who use services and cloud computing systems do not know their data's exact physical locations because data processing and storage is often at unspecified geographic locations, both domestic and foreign. Legally, each location has a different jurisdiction. Service providers in foreign countries might not always guarantee regulatory compliance, such as protecting privacy, backing up data, or providing an audit trail. They might not be willing to assume liability for security incidents or for the failure to meet data backup requirements or to provide audit trails. They also might not protect intellectual property according to compliance standards.[12]

Application developers who establish SLAs with service providers need to check if the providers will commit to storing and processing data in specific jurisdictions, and if they will make a contractual commitment to comply with all regulatory requirements and liabilities in publishing and managing applications.

Although services computing and cloud computing have great promise in meeting the increasingly severe requirements of dynamic application development and use, fully realizing this potential requires some kinds of application development structure. Software engineering can help combine these computing paradigms and harness their considerable advantages for application development. Although many challenges remain in moving this idea from vision to implementation, the benefits of such an environment should serve to motivate the software engineering research that can meet those challenges. C

Acknowledgments

The work described in this article was partially supported by the National Science Foundation under grant CCF-0725340.

References

1. Y. Chen and W.-T. Tsai, *Service-Oriented Computing and Web Data Management: From Principles to Development*, Kendall Hunt, 2010.
2. K. Channabasavaiah, E. Tuggle, and K. Holley, "Migrating to a Service-Oriented Architecture," 16 Dec. 2003; www.ibm.com/developerworks/webservices/library/ws-migratesoa.
3. S.S. Yau and H.G. An, "Confidentiality Protection in Cloud Computing Systems," *Int'l J. Software Informatics*, vol. 4, no. 4, 2010, pp. 351-365.

4. Y. Zhu et al., "Dynamic Audit Services for Outsourced Storage in Clouds," to be published in *IEEE Trans. Services Computing*, 2011.
5. L. Whitney, "Amazon EC2 Cloud Service Hit by Botnet, Outage," *CNET News*, 11 Dec. 2009; http://news.cnet.com/8301-1009_3-10413951-83.html.
6. "Google Cloud Platform Used for Botnet Control," *Info Security*, 10 Nov. 2009; www.infosecurity-us.com/view/5115/google-cloud-platform-used-for-botnet-control.
7. C. Li, A. Raghunathan, and N. Jha, "Secure Virtual Machine Execution under an Untrusted Management OS," *Proc. IEEE 3rd Int'l Conf. Cloud Computing* (CLOUD 10), IEEE CS Press, 2010, pp. 172-179.
8. Cloud Security Alliance, "Top Threats to Cloud Computing V1.0," Mar. 2010; https://cloudsecurityalliance.org/topthreats/csathreats.v1.0.pdf.
9. S.S. Yau et al., "Automated Situation-Aware Service Composition in Service-Oriented Computing," *Int'l J. Web Services Research*, vol. 4, no. 4, 2007, pp. 59-82.
10. S.S. Yau et al., "Rapid Development of Adaptable Situation-Aware Service-Based Systems," *Web Services Research for Emerging Applications: Discoveries and Trends*, L.-J. Zhang, ed., Information Science Reference, IGI Global, 2010, pp. 104-139.
11. S.S. Yau et al., "Toward Development of Adaptive Service-Based Software Systems," *IEEE Trans. Services Computing*, vol. 2, no. 3, 2009, pp. 247-260.
12. B.T. Ward and J.C. Sipior, "The Internet Jurisdiction Risk of Cloud Computing," *Information Systems Management*, vol. 27, no. 4, 2010, pp. 334-339.

Stephen S. Yau *is the director of Arizona State University's Information Assurance Center and a professor of computer science. His research interests include software engineering, cybersecurity, distributed computing systems, service-based computing, and cloud computing systems. Yau received a PhD in electrical engineering from the University of Illinois at Urbana-Champaign. He is a Life Fellow of IEEE and a Fellow of the American Association for the Advancement of Science. Contact him at yau@asu.edu or at http://dpse.asu.edu/yau.*

Ho G. An *is a PhD student in the School of Computing, Informatics and Decision System Engineering at Arizona State University. His research interests include services and cloud computing, security, and privacy. An received an MS in computer science from Arizona State University. Contact him at ho.an@asu.edu.*