Services and Service Composition – An Introduction

Services und Service Komposition – Eine Einführung

Schahram Dustdar, Technical University Vienna (Austria),
Mike P. Papazoglou, University Tilburg (The Netherlands)

Summary In this overview paper, we discuss the basic principles underlying service-oriented computing in general, and (Web) services in particular. We discuss the important differences between (Web) services and Web applications and other models in Internet computing. Finally, we discuss where we see the future research challenges in the area of service composition. ►►► **Zusammenfassung** In diesem Übersichtsartikel behandeln wir die Grundlagen von Service-oriented Computing und (Web) Services im Speziellen. Wir besprechen die wesentlichen Unterschiede zwischen (Web) Services und Web Applikationen sowie weiteren Modellen im Bereich des Internet Computing. Im letzten Teil des Artikels besprechen wir die zukünftigen Herausforderungen im Bereich der Service Composition Forschung.

KEYWORDS D [Software]; Web Services, Service-Oriented Computing, Service Composition, Service-Oriented Architectures, Service Engineering

1 What Are Services?

Services perform functions that can range from answering simple requests to executing sophisticated business processes requiring peer-to-peer relationships between service consumers and providers. Services are self-contained processes – deployed over standard middleware platforms, e. g., J2EE, or .NET – that can be described, published, located (discovered), and invoked over a network. Any piece of code and any application component deployed on a system can be transformed into a network available service. Services are most often built in a way that is independent of the context in which they are used. This means that the service provider and the consumers are loosely coupled.

In every new evolving field there is sometimes a mismatch regarding the terminology. Similarly, the terms Internet, Web, and Mosaic (the first well-known Web browser) were used synonymously in the early 1990ies. Here is how we use the terms in this paper. *Service-oriented Computing* (SOC) deals with all issues related to science, research, and technology in the field of services. *Service-oriented Architecture* (SOA) describes an *architectural style* consisting of the so called SOA-Triangle [12]. This style foresees three roles: service provider, service consumer, and service registry (proxy). An architectural style does not limit itself to *one* implementation technology. *Web services* are one set of technologies and composable standards with well-defined interfaces, for *implementing* an SOA. Obviously, it is possible to build service-oriented systems following an SOA without using Web services at all. However, it is clear that the advent of the Web services standards boosted the proliferation of the principles as well as the actual implementations of service-orientation. Thus we should bear in mind that service-orientation following an SOA including all questions being researched today under the umbrella of SOC are "here to stay", whereas the particular implementation technologies (e. g., today Web services) might change and evolve over the years to come.

A "service-oriented" approach to programming is based on the idea of composing applications by discovering and invoking network-available services rather than building new applications or by invoking available applications to accomplish some task [1]. SOC utilizes services as the constructs to support the development of rapid, low-cost and easy to compose distributed applications.

SOC is expected to have an impact on all aspect of software

construction as wide as that of object-oriented programming. The premise of its foundation is that an application can no longer be thought of as a single process running within a single organization. The value of an application is actually no longer solely measured by its functionality but by its ability to integrate with its surrounding environment. For instance, services can help integrate applications that were not written with the intent to be easily integrated with other applications and define architectures and techniques to build new functionality leveraging existing application functionality. A new type of applications can be based solely on sets of interacting services offering well-defined interfaces to their potential users. These applications are often referred as composite applications. Service orientation requires and enables (e. g., through providing composeable Web services standards) loosely coupled relationships between applications of transacting partners. At the middleware level, loose coupling requires that the "service-oriented" approach is independent of specific technologies or operating systems. The service-oriented model does not even mandate any kind of pre-determined agreements before the use of an offered service is allowed. The service model allows for a clear distinction to be made between service providers (organizations that provide the service implementations, supply their service descriptions, and provide related technical and business support), service clients (end-user organizations that use some service), and service aggregators (organizations that consolidate multiple services into a new, single service offering).

Service-oriented Architectures typically consist of three roles: service providers, consumers, and registries (brokers). Services are offered by service providers, which are organizations that procure the service implementations, supply their service descriptions, and provide related technical and business support. Since services may be offered by different enterprises and communicate over the Internet, they provide a distributed computing infrastructure for both intra and cross-enterprise application integration and collaboration. Clients of services can be other solutions or applications within an enterprise or clients outside the enterprise, whether these are external applications, processes or customers/users. This distinction between service providers and consumers is independent of the relationship between consumer and provider, which can be either client/server or peer to peer. For the SOC paradigm to exist, we must find ways for the services to be:

- *Technology neutral*: They must be invoked through standardized lowest common denominator technologies that are available to almost all IT environments. This implies that the invocation mechanisms (protocols, descriptions and discovery mechanisms) should comply with widely accepted standards.
- *Loosely coupled*: They must not require knowledge or any internal structures or conventions (context) at the client or service side.
- *Support location transparency*: Services should have their definitions and location information stored in a repository such as the *Universal Description and Discovery and Integration* (UDDI) repository and be accessible by a variety of clients that can locate and invoke the services irrespective of their location.

Services may be implemented on a single machine or a large number and variety of devices, and be distributed on a local area network or more widely across several wide area networks (including mobile and ad hoc networks). A particularly interesting case is when the services use the Internet (as the communication medium) and open Internet-based standards. This results in the concept of Web services, which share the characteristics of more general services, but they require special consideration as a result of using a public, insecure, low-fidelity mechanism for inter-service interactions.

Web services constitute a distributed computer infrastructure made up of many different modules trying to communicate over private or public networks (including the Internet and Web) to virtually form a single logical system. Web services are modular, self-describing, self-contained applications that are accessible over the Internet. They are the answer to the problems of rigid implementations of predefined relationships and isolated services scattered across the Internet. A Web service is a service available via a network, such as the Internet, that completes tasks, solves problems or conducts transactions on behalf of a user or application.

Web services can vary in function from simple requests (e. g., currency conversion, credit checking and authorization, inventory status checking, or a weather report) to complete business applications that access and combine information from multiple sources, such as an insurance brokering system, a travel planner, an insurance liability computation or a package tracking system. Enterprises can use a single Web service to accomplish a specific business task, such as billing or inventory control or they may compose several Web services together to create a distributed e-Business application such as customised ordering, customer support, procurement, and logistical support.

Web services efforts focus on wrapping existing applications (including legacy code) for lightweight integration with other applications, often motivated by the desire for new forms of sharing of services across lines of business, or between business partners. Typical

real-world examples of these efforts include:

- Opening up existing backend systems so that suppliers can access inventory data and hence perform more cost-effective inventory management.
- Exposing order management data for customer self-service in order tracking. Automate several types of transactions among a diverse and fragmented set of suppliers who previously performed transactions with paper, fax, and email.
- Providing a single enterprise-wide view of customer data (or manufacturing status or product design data) by aggregating data from several disparate sources.
- Enabling access to aggregated data by sales (or supply or demand) partners.

The eventual goal of Web services technology is to enable distributed applications that can be dynamically assembled according to changing business needs, and customised based on a device, network, e. g., cable, UMTS, XDSL, Bluetooth, etc., and user access while enabling wide utilization of any given piece of business logic wherever it is needed.

2 The Concept of Software as a Service

Web services are very different from Web pages that also provide access to applications across the Internet and across organisational boundaries. Web pages are targeted at human users, whereas Web services are developed for access by automated applications. Web services are about machine-to-machine communication, whereas Web pages are about human to machine communication. As terminology is often used very loosely, it is easy to confuse someone by describing a 'service' as a Web service when it is in fact not. Consequently, it is useful to examine first the concept of software as-a-service on which Web services technology builds upon and then compare Web services with Web server functionality.

The concept of *Software-as-a-Service* (SaaS) appeared first with the *Applications Service Provider* (ASP) software model. ASPs are organizations that package software and infrastructure elements together with business and professional services to create a complete solution that they present to the end customer as a service on a subscription basis. An ASP is a third party entity that deploys, hosts, and manages access to a packaged application and delivers software-based services and solutions across a network to multiple customers across a wide area network from a central data centre. In essence, ASPs were a way for companies to outsource some or even all aspects of their information technology needs. The ASP Industry Consortium defined that application service providers are service organizations that deploy, host, manage, and enhance software applications for customers at a centrally managed facility, offering application availability, performance, and security. End-users access these applications remotely using Internet or leased lines. The whole application is developed in terms of the user interface, workflow, business and data components that are all bound together to provide a working solution. As an ASP hosts the entire application, the customer has little opportunity to customize it beyond setting up tables, or perhaps the final appearance of the user interface (such as, e. g., adding company logos).

Access to the application for the customer is provided simply via browsing and manually initiated purchases and transactions occur by downloading reports. This activity can take place by means of a browser. This is not a very flexible solution – but offers considerable benefits in terms of deployment providing the customer is willing to accept it 'as is'. By providing a centrally hosted Internet application, the ASP takes primary responsibility for managing the software application on its infrastructure, using the Internet as the conduit between each customer and the primary software application. What this means for an enterprise is that the ASP maintains the application, the associated infrastructure, and the customer's data and ensures that the systems and data are available whenever needed. An alternative of this is where the ASP is providing a software module that is downloaded to the customer's site on demand – this is for situations where the software does not work in a client/server fashion, or can be operated remotely via a browser. This software module might be deleted at the end of the session, or may remain on the customer's machine until replaced by a new version, or the contract for using it expires.

Although the ASP model introduced the concept of software-as-a-service first, it suffered from several inherent limitations such as the inability to develop highly interactive applications, inability to provide complete customisable applications and inability to integrate applications. This resulted in monolithic architectures, highly fragile, customer-specific, non-reusable integration of applications based on tight coupling principles.

Today we are in the midst of another significant development in the evolution of software-as-a-service. The new architecture allows for loosely-coupled asynchronous interactions on the basis of *eXtensible Markup Language* (XML) standards with the intention of making access to, and communications between, applications over the Internet easier.

The SOC paradigm allows the software-as-a-service concept to expand to include the delivery of complex business processes and transactions as a service, while permitting applications to be constructed on the fly and services to be reused without any geographic or user-type restriction. Perceiving the relative benefits of service-oriented technol-

ogy many ASPs are modifying their technical infrastructures and business models to be more akin to those of Web service providers. The use of Web services provides a more flexible solution. The core of the application – the business and data components remain on the ASP's machines, but are now accessed programmatically via Web service interfaces. The customers can now build their own custom business processes and user interfaces, and are also free to select from a variety of Web services that are available over the network and satisfy their needs. When comparing Web services with Web-based applications we may distinguish four key differences [1]:

- Web services act as resources to other applications that can request and initiate those Web services, with or without human intervention. This means that Web services can call on other Web services to outsource parts of a complex transaction to those other Web services. This provides a high degree of flexibility and adaptability not available in today's Web-based applications.
- Web services are modular, self-aware and self-describing applications; a Web service knows what functions it can perform and what inputs it requires to produce its outputs and can describe this to potential users and to other Web services. A Web service can also describe its non-functional properties, e. g., the cost of invoking the service, the geographical areas the Web service covers, security measures involved in using the Web service, performance characteristics, contact information and more.
- Web services are more visible and manageable than Web-based applications; the state of a Web service can be monitored and managed at any time by using external application management and workflow systems. Despite the fact that a Web service may not run on an in-house (local) system or may be written in an unfamiliar programming language it still can be used by local applications, which may detect its state (active or available) and manage the status of its outcome.
- Web services may be brokered or auctioned. If several Web services perform the same task, then several applications may place bids for the opportunity to use the requested service. A broker can base its choice on the attributes of the "competing" Web services (cost, speed, degree of security).

3 Service Composition

The basic Web services infrastructure presented in the previous section suffices to implement simple interactions between a client and a Web service. If the implementation of a Web service's business login involves the invocation of other Web services, it is necessary to combine the functionality of several Web services. In this case we speak of a *composite service*. The process of developing a composite service in turn is called *service composition*. Service composition can be either performed by composing elementary or composite services. Composite services in turn are recursively defined as an aggregation of elementary and composite services. When composing Web services, the business logic of the client is implemented by several services. This is analogous to workflow management, where the application logic is realized by composing autonomous applications. This allows the definition of increasingly complex applications by progressively aggregating components at higher levels of abstraction. A client invoking a composite service can itself be exposed as a Web service.

Since it is a widely used approach to use conventional programming languages to link components to a composite Web service and thus bridge heterogeneous middleware platforms it becomes necessary to develop a Service Composition Middleware to support composition in terms of abstractions and infrastructure as well [2]. Programming languages focus on APIs rather than on the actual business logic. Different approaches and the need for workflow modeling have finally led to the development of the *Business Process Execution Language for Web Services* (WSBPEL, or BPEL, for short) [3]. A composition model and language to specify the services involved in the composition, a development environment with a graphical user interface to drag and drop Web service components and a run-time environment to execute the business logic can be identified as the three main elements of a Web services composition middleware. A service composition middleware requires the Web services to be precisely described in their functionality, interfaces and protocols they support. Conventional middleware lacks exactly those features. *Workflow Management Systems* (WfMS) are highly flexible and generic but on the other hand require the components to be aware of the WFMS API. Hence, components are system and vendor-specific and attended to additional development effort.

In [2] we describe six different dimensions of service composition models. A component model can make different assumptions of what a component is and what it is not. The advantage of a model making very basic assumptions, for example components only have to exchange messages via XML, is a more general model, while it has to deal with much more heterogeneity of the components.

An orchestration model defines abstractions and languages to define the order in which and the conditions under which Web services are invoked. Orchestration models (described later in this section) use process modeling languages, such as UML activity diagrams, Petri-

nets, state-charts, rule-based orchestration, activity hierarchies and π-calculus. Data and data access models define how data is specified and exchanged between components. The service selection model deals with static and dynamic binding that is how a Web service is selected as a component statically at design-time or dynamically during run-time. Transactions define which transactional semantics can be associated to the composition and how this is done. Finally, we must also consider a model for exception handling to handle exceptional states during the execution of the composite service without the service being aborted.

Services technologies address the problem of integrating simple services and business processes into composite added value services, since they support coordination and offer an asynchronous and message oriented way to communicate and interact with application logic. However, when looking at Web services, for example, it is important to differentiate between the baseline specifications of SOAP, UDDI and WSDL that provide the infrastructure that supports publishing, finding and binding operations in the service-oriented architecture and higher-level specifications required for e-Business integration. These higher-level specifications provide functionality that supports and leverages services and enables specifications for integrating automated business processes.

Currently, there are competing initiatives for developing business process definition specifications, which aim to define and manage business process activities and business interaction protocols comprising collaborating services. The terms "orchestration" and "choreography" have been widely used to describe business interaction protocols comprising collaborating services. Orchestration describes how services can interact with each other at the message level, including the business logic and execution order of the interactions from the perspective and under control of one of the business parties involved in the process.

The Business Process Execution Language for Web Services (BPEL) serves as the de facto standard for service orchestrations and is standarized by the OASIS. BPEL descriptions are XML documents, which describe the roles involved in the message exchange, supported port types and orchestration and correlation information as aspects of a process. BPEL is a service composition model which both supports composition and coordination protocols and consists of an activity-based component model, an orchestration model allowing the definition of structured activities, XML schema data types, a service selection model, and a mechanism for exception, event and compensation handling.

Choreography is typically associated with the public (globally visible) message exchanges, rules of interaction and agreements that occur between multiple business process endpoints, rather than a specific business process that is executed by a single party. Choreography is more collaborative in nature than orchestration. Service choreography is targeted by *Web Services Choreography Description Language* (WS-CDL), which specifies the common observable behaviour of all participants engaged in business collaboration.

This sharp distinction between orchestration and choreography is rather artificial and the consensus is that they should both coalesce in the confines of a single language and environment. On the research front, activities have mainly concentrated on dynamic compositions [4], on modularizing compositions [4; 5], on enhancing service descriptions (with, e. g., compositional assertions) so that compositions can be assessed and formally verified [6] and on providing context aware services to enable compositions. In the AI field there has been some work in the area of applying AI planning techniques to automate the retrieval and composition of Web services ([7–9]), verification [10], and monitoring of service oriented applications, and so forth, but these efforts are still either at the specification-level or at very preliminary stage of development. Many of the existing approaches towards service composition largely neglect the context in which composition takes place. It is only recently that research approaches have focussed on developing context-aware methodologies that take into account the business and social context of service compositions as the basis for process specification and verification [11]. For a survey on service composition approaches we refer to [2].

Service Composition Challenges Ahead

Service composition is one of the hot research topics since, by nature, this area involves difficult general problems combining areas including workflow research, coordination aspects, software engineering and others. Some of the most notable research challenges for the near future include (but are not limited to) (a) composability analysis for replaceability, compatibility, and conformance for dynamic and adaptive processes, (b) adaptive and emergent service compositions, (c) autonomic composition of services, (d) QoS-aware service compositions, (e) business-driven automated compositions, and (f) service governance, management, and administration. In the following section, we briefly touch on some of the above research challenges.

Composability analysis for replaceability, compatibility, and conformance for dynamic and adaptive processes to ensure the integrity of a composite service by matching its operations with those of its constituent component services imposes semantic constraints on the component services (e. g., to en-

sure enforcement of business rules), and ensures that constraints on data exchanged by component services are satisfied. Service conformance addresses both behavioral conformance as well as semantic conformance.

Adaptive service compositions: Service adaptivity is particularly useful for integrated supply chains as it implies that an integrated supply chain solution can leverage collaborative, monitoring and control abilities to manage product variability and successfully exploit the benefits of available-to-promise capabilities. All the tasks involve conversation between processes that span enterprises, with customized alerts set up across the network to track exceptions and provide manual intervention if necessary.

Autonomic composition of services: One of the main fundamental ideas of SOC is that applications should be developed by composing services that are available, e. g., on the Web. Given some business level and strategic requirements for the composition, the idea is to automatically generate the electronic business process implementing it. In this framework, the challenge is the autonomic composition of services, e. g., service composition that are self-configuring, self-optimizing, self-healing, and self-adapting. Self-configuring compositions are, e. g., composite services that are capable of automatically discovering new partners to interact with, to automatically select among available suppliers, to choose among different options available for contracts, etc. Self-optimizing Web service compositions should automatically select partners and options that would, e. g., maximize benefits and reduce costs. Self-healing compositions should be able to automatically detect that some business composition requirements are no longer satisfied by the implementation and react to requirement violations. Self-adapting service compositions should be able to function in spite of changes in behaviours of external composite services, they should reduce as much as possible the need of human intervention for adapting services to subsequent evolutions.

QoS-aware service compositions: To be successful service compositions need to be QoS-aware, i. e., understand and respect each other's policies, performance levels, security requirements, Service-leveal agreement (SLA) stipulations, and so forth. For example, knowing that a new business process adopts a Web services security standard such as one from the stack of WS-Security specifications is not enough information to enable successful composition. The client needs to know if the services in the business process actually require WS-Security, what kind of security tokens they are capable of processing, and which one they prefer. Moreover, the client must determine if the service should communicate using signed messages. If so, it must determine what token type must be used for the digital signatures. Finally, the client must decide on when to encrypt the messages, which algorithm to use, and how to exchange a shared key with the service. For example, a purchase order service in an order management process may indicate that it only accepts username tokens that are based signed messaged using X.509 certificate that is cryptographically endorsed by a third party.

4 Conclusions

In this overview paper, we outlined what we mean by Service-oriented Computing in general and (Web) services in particular. We discussed the differences between (Web) services and Web applications and other models in Internet Computing and furthermore discussed the principles of Service-oriented Computing and services. In the last section, we outlined where we see the future research challenges in the area of service composition.

References

[1] M. Papazoglou (2007) Web Services Principles and Technology, Prentice Hall, New Jersey, Aug 2007.

[2] S. Dustdar, W. Schreiner (2005) A Survey on Web services Composition. Int'l Journal of Web and Grid Services, 1(1), 1–30.

[3] A. Arkin et al. (2004) Web Services Business Process Execution Language Version 2.0, http://www.oasis-open.org/committees/download.php/10347/wsbpel-specification-draft-120204.htm

[4] J. Yang and M.P. Papazoglou (2004) Service Components for Managing the Life-Cycle of Service Compositions. Information Systems, 29 (2004) 2, pp. 97–125.

[5] A. Charfi, M. Mezini (2004) Hybrid Web Service Composition: Business Processes meet Business Rules. Int'l Conf. on Service Oriented Computing (ICSOC 2004), New York, Dec 2004.

[6] M. Solanki, A. Cau, H. Zedan (2004) Augmenting Semantic Web Service Descriptions with Compositional Specification. WWW '04: 13th Int'l Conf. on World Wide Web, New York, NY, USA, 2004. ACM Press.

[7] M. Paolucci et al. (2002) Semantic Matching of Web Services Capabilities. 1st Int'l Semantic Web Conf., Sardinia, Italy, 2002.

[8] A. Lazovik, M. Aiello, M.P. Papazoglou (2004) Associating Assertions with Business Processes and Monitoring their Execution. Int'l Conf. on Service Oriented Computing (ICSOC 2004), New York, Dec 2004.

[9] P. Traverso, M. Pistore (2004) Automatic Composition of Semantic Web Services into Executable Processes. Int'l Semantic Web Conf. (ISWC), Hiroshima, Japan, 2004.

[10] R. Kazhamiakin, M. Pistore (2006) A Parametric Communication Model for the Verification of BPEL4WS Compositions. Int'l World Wide Web Conf. (WWW), Edinburgh, Scotland, 2006.

[11] E. Colobo, J. Mylopoulos, P. Spoletini (2005) Modeling and Analyzing Context- Aware Composition of Services. 3rd Int'l Conf. on Service Oriented Computing, Springer, Amsterdam, The Netherlands, Dec 2005.

[12] A. Michlmayr, F. Rosenberg, C. Platzer, M. Treiber, S. Dustdar (2007) Towards Recovering the Broken SOA Triangle – A Software Engineering Perspective, 2nd Int'l Workshop on Service-oriented Software Engineering (IW-SOSWE'07), Dubrovnik, Croatia, Sep 2007, ACM Press.

1

2

1 Prof. Dr. Schahram Dustdar is full Professor of Computer Science and head of the Distributed Systems Group, Information Systems Institute, Vienna University of Technology (TU Wien). He is also Honorary Professor of Information Systems at the Department of Computing Science at the University of Groningen (RuG), The Netherlands. From 1999–2007 he worked as the co-founder and chief scientist of Caramba Labs Software AG (CarambaLabs.com) in Vienna, a venture capital co-funded software company focused on software for collaborative processes in teams. Caramba Labs was nominated for several (international and national) awards: World Technology Award in the category of Software (2001); Top-Startup companies in Austria (Cap Gemini Ernst & Young) (2002); MERCUR Innovationspreis der Wirtschaftskammer (2002). Currently, Prof. Dustdar is on the advisory board of Smart Information Systems and Sanaga Labs, two Austrian Start-up companies as well as on the management board of the Association of the alumni of the TU Wien.
Address: Technische Universität Wien, Information Systems Institute, Argentinierstraße 8/184-1, 1040 Wien, Austria,
E-Mail: dustdar@infosys.tuwien.ac.at

2 Prof. Dr. Mike P. Papazoglou is a full professor and director of the INFOLAB/CRISM at the Univ. of Tilburg in the Netherlands. He is also an honorary professor at the University of Trento, Italy. Mike Papazoglou serves on the editorial board of several scientific journals and is co-editor in charge of the prestigious MIT book series on Information Systems. He has chaired numerous well-known international scientific conferences in Computer Science. These include the Int'l Conf. on Data Engineering (ICDE), Int'l Conf. on Distributed Computing Systems (ICDCS), Int'l Conf. on Digital Libraries (ICDL), Int'l Conf. on Cooperative Information Systems (CoopIS), Int'l Conf. on Entity/Relationship Modelling. He is the founder of the Int'l Conf. on Cooperative Information Systems (CoopIS) in 1993 and more recently of the Int'l Conf. on Service Oriented Computing (ICSOC). Mike Papazoglou has authored/edited fifteen books and well over hundred and fifty scientific journal articles and refereed conference papers.
Address: University Tilburg, Faculty of Economics and Business Administration, P.O. Box 90153, 5000 LE Tilburg, The Netherlands, E-Mail: mikep@uvt.nl