

State of the art in developing applications using services: A review of basic concepts ¹

Mikko Raatikainen

Software Business and Engineering Institute (SoberIT)
Helsinki University of Technology
P.O. Box 9210, FIN-02015 HUT, Finland
Mikko.Raatikainen@hut.fi

Abstract. Developing application using services is increasingly commonly referred paradigm. The state of the art of the basic principles for developing applications using services are identified in this paper. These include concepts of service-oriented computing, a service, and a service-oriented architecture. The review shows that these concepts are relatively similarly understood in different studies, but the concept are not conceptually completely clear. A comparison of this service paradigm with more traditional software development paradigms shows similarities in particular with reuse-oriented software development paradigm such that in service paradigm is also a few unique characteristics.

1 Introduction

Developing application on a basis of services seems to be becoming more commonplace, or at least more often referred paradigm for constructing applications. It seems that service based software development is the next paradigm following the component based or component oriented software development paradigm. Further, implementing applications using services seems, at least intuitively, to require to follow in some extend the similar principles as construction of applications using those more traditional building blocks. That is, the entities that are used to construct such an application are needed to be, e.g., found, retrieved, and composed, used within a specified structure, and the quality of the resulted application needs to be assured. However, since services are the new paradigm, they also introduce new principles or modify the existing principles.

This report focuses on the problem of developing applications on a basis of services. This is done on a basis of the above mentioned assumption that development with services resembles to more traditional development approach. The main goal of the report is to find basic concept for constructing applications using services. In more detail, the report is set to answer following questions:

1. What is meant by developing application from services?

¹ This is a paper written for T-76.270 Seminar at Helsinki University of Technology.
©Mikko Raatikainen. All rights reserved.

2. What are the building blocks in constructing application on a basis of services?
3. What is the structure that such application follows?

The report focuses on services that are implemented as software in general. There are two major limitations in this report. First, the report excludes all other services than those implemented as software. Second, the report does not focus on certain kinds of services such as the web services or some particular standard of services.

The rest of the report is organized as follows. Section 2 introduces the used method and terminology for the rest of the report. Sections 3, 4, and 5 try to answer to the research question 1, 2, and 3, respectively. Section 6 summarizes the found concepts. In Section 7, we compare these concepts with software engineering in general. Section 8 includes discussion of the results of the review. Section 9 draws the conclusions.

2 Method and Terminology

This report aims to find state-of-the-art answers to the research questions described above. A means to do this was to search for scientific databases on a basis of key words. In addition, Internet based searches were conducted using Google search engine. However, the focus was on scientific articles, standards, and white papers and other reports on www-pages of different organizations such as white papers at IBM.

The limitation of the method in this report is that it does not aim at a systematic review. However, for each question and issues that the questions deal with are tried to find a few sources that would be as independent of each other as possible. Another limitation of this work is that it does not try to assess quality of the publications but relies on the papers publishers own procedures to assure quality of the publications. This limits also use of possible source such that personal www-pages that do not necessary have external quality assurance procedures are not included in the review.

The terminology use in this report is as follows. The term *application* is used of an entity that a user needs. In traditional software engineering, application would mean the software that the user gets to answers the needs. The term *construction* is used as a general term of the activities that take place between a request to meet user needs is invoked to the point when the user needs are answered.

3 Developing applications from service

Development of application from services originates, according to [1], application server providing (ASP). In this model, an application is simply provided over network. A special provider owns an application and keeps it in a certain server. A user the uses the application as a service instead of installing and running it

from own computer. In fact, this use of an application as a service over network instead of installing to local computer is the main and almost only difference between ASP and traditional software. The application in the ASP model is one entity similar with other software.

A more recent definition of what means to develop applications from services is Papazoglou's [1] definition of service-oriented computing:

"Service-oriented computing (SOC) is the computing paradigm that utilizes services as fundamental elements for developing applications."

The first observation of this definition, quite unsurprisingly, is that services are elements or building blocks in SOC for developing applications. Another observation is that the definition uses of the term utilize. This term is not further elaborated. However, later Papazoglou adds that services in SOC provide functionality that is in place and readily available. This excludes that utilization could refer to, for example, models that could be used to construct or generate the required functionality, but services are existing entities. On the other hand, according to this definition, it is not restricted by any other means how services can be utilized in developing applications than that the entities must exist and be finished work products. That is, the services can be, for example, accessed or referenced via web or used in composition similarly as components.

In addition to above definition Turner *et al.* [2] describe the *software as a service* (SaaS) concept in which the application are developed from services. The concept addresses "a demand-led software market in which businesses assemble and provide services when needed to address a particular requirement". A key in SaaS is separation of the possession and ownership of software from its use. That is, the owner of the software or its parts become irrelevant such that user can use software from any source as services. The application is then constructed from services and construction of the software is delayed as late as possible to run time and on demand.

The delaying of construction is further discussed in [3] where Bennett *et al.* introduce *service-based model of software* as follows:

[The] service-based model of software is one in which services are configured to meet a specific set of requirements at a point in time, executed and then disengaged.

The end of this definition refers to *ultra-late binding*. The ultra-late binding means that application is constructed when they executed and the services that are included in the application are removed after they are no longer needed. This construction takes place on demand depending on customer needs. Hence, each application is customized dynamically to customer every time it is executed. The development of such an application requires, in practice, that the services are not owned, but used as described above. That is, the application, or its user, does not own and buy the services but takes it for use and pays for the use.

4 Services as building blocks

The concept services are discussed by several authors. Some definitions emphasize certain aspect while others some other aspect. This does not mean that the definitions exclude other aspects, but the emphasis is only different. In the following, we will represent certain aspects of a service in the light of the authors that clearly emphasize the aspect.

Papazoglou defines service as follows [1]:

Services are self-describing, platform-agnostic computational elements that support rapid low-cost composition of distributed applications.

Self-describing means that a service must provide all the relevant information for its user that is required to use the service. This includes interfaces how to use the service, but also quality attributes. Platform-agnostic means that services are technology neutral such that they can be implemented such that they can be used from as many other typical infrastructures as possible. That is, the service should be able to invoke from other infrastructures that where the service is implemented. In addition to the platform-agnostic, services should be also location transparent meaning that they can be invoked from several different locations, for example, within an enterprise or even between organizations over different kinds of networks. The service should also be loosely coupled meaning roughly that services do not require knowledge of internal structure of other services.

Colan states:

[A service] is an application function packaged as a reusable component for use in a business process. It either provides information or facilitates a change to business data from one valid and consistent state to another.

This addresses another issues in services namely the granularity. This definition connects services to business processes. Another point of view is that software services are functional units in an application [3]. This means that the services are a large number of small functional units, each supporting a purposeful human activity or a business transaction. Any particular software supplier of software services can either assemble their services out of existing ones, or develop and evolve atomic services using traditional software development techniques.

Regardless of the granularity of a service, services can be simple services or composite services[1]. Composite services are such services that can consists of services that possibly utilize several service provides by combining them to a new service.

Hao focuses more on service-oriented architecture, which is discussed more in the next section, and defines service through roles as follows:

A service is a unit of work done by a service provider to achieve desired end results for a service consumer. Both provider and consumer are roles played by software agents on behalf of their owners. [4]

The key issue here is the introduction of the two roles of services: a provider and a consumer. A consumer is also sometimes called a client. Both these roles are logical constructs and an agent can be simultaneously both service provider and consumer. The main responsibility of a provider is to publish a description of the services and provide the service for use of other agents. Clients must be able to find the descriptions of the services they require and must be able to bind to them.

Another point of view to a service is given by Arsanjani as he argues [5]:

A service is a software resource (discoverable) with an externalized service description.

It is common to differentiate between this kind of service description and implementation [5] [1]. Service descriptions are used to search, bind, and invoke services. Service description documents the service capability, interface behavior, and quality. The interface simply provides the mechanism by which services communicate with applications and other services. Technically, the service interface is the description of the signatures of a set of operations that are available to the service client for invocation. Capability means the purpose and expected results of the service. Interface means the service signature. Quality describes, e.g., cost, performance metrics, and security.

Service implementation, in turn, is realization of the service that service provider does. Services may be realized by in house service design and implementation, purchasing, leasing, paying for services, outsourcing service design and implementation, or using wrappers that reuse legacy code by converting the legacy functionality and encapsulating it inside components or adapters that use legacy code in combination with newly developed code.

In addition, binding is also treated separately from realization [1][5]. Binding means that two services are connected together. This pertains to the fact that services are such loosely coupled. However, in more detail this is discussed in next section since this relates to the service-oriented architecture.

5 Service-oriented architecture

Above has been discussed services. The services are used in a special service-oriented architecture (SOA). The SOA is discussed in this section in more detail.

Hao [4] defines SOA as follows:

Service-oriented architecture is an architectural style whose goal is to achieve loose coupling among interacting software agents.

The agents in the above definition are services. The services the above mentioned a provider or a requester. Another important issue is the use of the term style. That is, SOA defines a style rather than requires special elements or protocols between services. In other words, as noted in [6] SOA is a logical way of designing a software system to provide services to applications through published

and discoverable interfaces. In fact, even existing applications, components or other software assets can be used, if adapted accordingly, in SOA as building blocks. This style that SOA defines is the key element in providing the service based applications described above which includes, for example, that the application are developed from services dynamically [5].

Another important aspect that this definition addresses is the loose coupling. The loose coupling means roughly that the agents do not depend on the functionality of other agent more than necessary; Applications are developed as independent sets of services. There is no requirement for tight coupling of services nor there is even a need for predefined agreements before use of services

SOA accomplishes, according to Hao [4], loose coupling by requiring two constraints. First, the agents have only a small set of simple interfaces. Second, through the interfaces are delivered only descriptive messages constrained by an extensible schema. Simple interfaces and extensible messages are required in order that services would be easy to use and usable over time. A descriptive message means that the message does not contain information how to accomplish, e.g., some data manipulation but only the data to be manipulated without caring how the data is to be manipulated. Consequently, in SOA in no need of convoluted point-to-point solutions based on inscrutable proprietary protocols [6].

The interaction between the agents in SOA is achieved through sending messages. Papazoglou [6] categorizes these messages into three broad categories: publish, find, and bind. Publish means that the description of the service is made available to be found and bound. Finding, in turn, means that a client asks, for example, from a broker to locate a service that accomplishes certain task. Finally, binding means that a client and provider are connected and they start to exchange the messages in order to accomplish, for example, a certain task or business process. This service interaction and communication require some kind of infrastructure. However, SOA does not specify this infrastructure, for example, in a form of a particular protocol or other means.

In addition to the simple model of provider, requester, broker model, sometimes SOA includes actors that extend this model. Service locator [7] which is a special kind of service has a registry of services and allows lookup for a specific service from its registry while a service broker refers to the service that can pass a service request to one or more additional service providers. The service broker seem to be similar with composite service. Arsanjani [5] uses the term service broker of the actor that keeps a registry of other services. Bennett *et al.* [3] use the term information service providers (ISP) of the services that provide information to other services e.g. the registry. They further identify contractor service providers (CSP) who have the ability to negotiate and assemble the necessary services to deliver a service to the end-user and software service providers (SSP) who are vendors that provide either the operational software services themselves, or descriptions of the components required and how they should be assembled.

6 Summary of the key concepts

In the above sections we have identified key concept of developing applications from services. These concepts are summarized in Table 1.

Table 1. Summary of basic concepts in developing applications using services

- In SOC services are major building blocks and utilized in applications as is in place and readily available entities
- In SOC application are bound ultra-late
- Services are used, not owned
- Applications are constructed on demand and constructed application is not stored but re-constructed of possible new services whenever required
- Characteristics of services: self-describing, platform-agnostic, location transparent
- Services are loosely coupled
- Services implement business functionality or processes
- Service have two major roles: a provider and consumer
- Service differentiate description and implementation
- Service differentiate binding and implementation
- SOA is an architectural style
- Main characteristic of SOA is loose coupling
- SOA consists of simple interfaces and descriptive messages
- Interaction in SOA through three message types: publish, find, and bind

7 Comparison with software engineering

We summarized the major characteristics of developing application using services in the previous section. When comparing these with the software engineering in general, characteristics become evident. First, a characteristic of service states that services are building block that in place and readily available entities. In traditional software engineering, application development includes development code. However, there is a special class of software engineering, namely reuse-based software engineering (RBSE), that is uses available entities in developing applications. Consequently, service based development resembles most RBSE, which is further discussed below. On the other hand, software engineering in general seems not to be feasible for comparison since service based development approach covers completele different model.

From the identified characteristics a clearly missing issues is development of the service per se. That is, the identified characteristics cover in contrast to reuse based software engineering end products of the development for reuse process, development with reuse process but lack the aspect of development of reusable software. This may pertain to the limitations of the review. However, a major reason for this might be also that development of services follows similar principles as any other development of reusable software with certain extra limitation

because of special requirements of characteristics of the services. Consequently, service based development can be further restricted to correspond to only the development for reuse phase in RBSE.

When comparing RBSE it seems that most of the identified characteristics of services can be implemented also in RBSE. However, typically RBSE has other design decisions available as well. In RBSE components correspond with services. The differences are that components in RBSE are typically owned, they are not as platform-agnostic or location transparent as services, they do not have special roles, and in RSEB is not defined such architectural style that has only three types of messages. With respect to all other aspects, RBSE can be implemented similarly as services. For example, component can be, but are not required to be, loosely coupled or applications can be bound dynamically and this binding can be released. That is, services can be regarded in many aspect a special class of RBSE, but not a true subset such that all characteristics of services are possible in RBSE as well.

8 Discussion and further work

Quite a surprising result of the review was that developing application from services in general has not been studied much. This is similar observation as in [8] that argues that there exists no systematic, methodological approach to service-oriented computing. Consequently, services in general terms seem to be still lacking extensive analysis and discussion.

Instead, in more detailed level exists a bit more studies. That is, the focus of studies seem to been more on the building blocks such as the characteristics of services and the service-oriented architecture. However, even for these issues does not seem to exist several studies.

It further seemed that a dominating issue was some special services such as web services. These services were even treated as service in general; a study found in the review that in the title or even in the abstract claimed to study services, turned out to be often about web services. Even these studies of some special services focus more on artifacts or their modeling and documentation similarly as services in general. There even seems to exist some methodological guidelines for some aspects for these special services [5].

There seems to be a relatively large consensus about what services are. The definitions did not have any drastic differences; different authors and studies seem to means the same when describing the issues related to services. Instead, the definitions emphasized different issues. The only major difference was the identification of actors in SOA: certain studies added some special actors to SOA.

However, the concepts of service-oriented computing, services, and service-oriented architecture seem to be mixed sometimes. Studies did not make a clear distinction between these similarly as in software engineering has been done between component, architectural components, and architecture.

Although the concepts are not yet completely clear, the paradigm is clear enough that it can be understood and applied. There seems to be today even standards that implement services such as the web service standards. These include the initial trio SOAP, WSDL and UDDI and some additional standards. However, the analysis of these standards was out of the scope of this paper.

Although there are standards that implements services, there seems to be a lack of empirical studies of actual use of service-oriented paradigm. It seems that the services are not yet matured to the level that they would be used in large scale in industry. However, it seems to be likely that service-based ideas are implemented in a small scale or at least such that certain conditions of the service based paradigm are violated. This, although we could not find any studies of these in the review.

In fact, it might be that current concepts per se are not feasible way to develop large scale commercial applications. For example, SOA does not give the provider any means to know about caller and there is not billing mechanisms. Hence, extensions for SOA are required. Some such extensions are already implemented for web services but are out of the scope of this paper. Another issue is to find the required services. Sometimes a principle is presented in which centralized brokers provide access to any kinds of services that are accessible through Internet by anyone, but in practice, it seems that realistic brokers are much smaller scale, e.g., for a certain application domain.

Nevertheless it remains to be seen does service based paradigm become a dominant paradigm for developing application. As discussed above, the major obstacles will be non-technical in nature. If compared to component based paradigm that is often regarded similar predominant paradigm in the 1990's, it seems that the promise of component based paradigm are not yet delivered to the practice at the full scale (cf. e.g. [9]). However, the component-based paradigm has influenced on application development. Hence, we believe that the service-based paradigm could influence on software engineering at least by making loose coupling and ultra-late and dynamic binding more typical solutions.

9 Conclusions

In this paper, we have presented a review of the basic concepts in service based application development. The identified basic concepts were as follows: In SOC services are major building blocks and utilized in applications as is in place and readily available entities; In SOC application are bound ultra-late; Services are used, not owned; Applications are constructed on demand and constructed application is not stored but re-constructed of possible new services whenever required; Characteristics of services: self-describing, platform-agnostic, location transparent; Services are loosely coupled; Services implement business functionality or processes; Service have two major roles: a provider and consumer; Service differentiate description and implementation; Service differentiate binding and implementation; SOA is an architectural style; Main characteristic of SOA is

loose coupling; SOA consists of simple interfaces and descriptive messages; Interaction in SOA through three message types: publish, find, and bind.

In addition to identifying service concepts, we compared the service-based development with software engineering, and reuse based software in particular. We found that some differences between these approaches exists, but are not drastic. Instead, service based development could be characterized as a special class of reuse-based software engineering with a few unique characteristics that are not possible in reuse-based software engineering.

Finally, we discussed the state of the practice of service-oriented paradigm. We argued that the service-based paradigm seems to be surprisingly little studied, the concept are similar in many studies, but not completely clear. We further argued that studies seem to focus more on some special services such as web services and that there is still lack of large scale industrial empirical data. Nevertheless, we argued that at least service based paradigm could have certain influences such as loose coupling and late binding on software engineering in general.

References

1. Papazoglou, M., Georgakopoulos, D.: Guest editor introduction: Service oriented computing. *ACM SIGSOFT Software Engineering Notes* **46** (2003) 24–28
2. Turner, M., Budgen, D., Brereton, P.: Turning software into a service. *IEEE Computer* **36** (2003) 38–44
3. Bennett, K.H., Munro, M., Xu, J., Gold, N., Layzell, P.J., Mehandjiev, N., Budgen, D., Brereton, P.: Prototype implementations of an architectural model for service-based flexible software. In: *Hawaii international conference on system sciences*. (2002)
4. He, H.: What is service-oriented architecture? (2003) <http://webservices.xml.com/pub/a/ws/2003/09/30/soa.html>.
5. Arsanjani, A.: Service-oriented modeling and architecture (2004) <http://www-128.ibm.com/developerworks/webservices/library/ws-soa-design1/>.
6. Papazoglou, M.: Service-oriented computing: concepts, characteristics and directions. In: *Proceedings of the Fourth International Conference on Web Information Systems Engineering. WISE*. (2003)
7. Endrei, M., Ang, J., Arsanjani, A., Chua, S., Comte, P., Krogdahl, P., Luo, M., Newling, T.: *Patterns: Service-Oriented Architecture and Web Services*. IBM Redbook (2004)
8. Krüger, I.H., Mathew, R.: Systematic development and exploration of service-oriented software architectures. In: *4th Working IEEE / IFIP Conference on Software Architecture (WICSA 2004)*, 12-15 June 2004, Oslo, Norway, IEEE Computer Society (2004) 177–187
9. Ravichandran, T., Rothenberger, M.A.: Software reuse strategies and component markets. *Communications of the ACM* **46** (2003) 109–114