Nebojša Taušan
Pere Tumbas
Predrag Matković

Service and Object Orientation: Synergy in Information Systems Development

Article Info:

Management Information Systems,
Vol. 4 (2009), No. 2,
pp. 019-027

Received 12 Jun 2008
Accepted 24 April 2009

UDC 007:004 ; 004.272.45

Summary

At the end of the 1960s, demand for increasingly complex functionalities of information systems caused a software crisis, which was successfully surmounted by transition from structural to object-oriented programming. Until the end of the 1990s, object was the basis of every information system, and consequently, development methodologies that were mostly used were object-oriented methodologies such as Rational Unified Process – RUP.

At the end of the 1990s, a new set of information requirements was noticed. These were requirements for transparent system integration; namely, the need for one system to communicate with another internal or external system became one of the most dominant needs. As a response to the newly created state, there appeared a new concept, the service concept. The service imposes itself as a solution, which will try, like objects at the end of 1960s, to face the newly created information requirements.

This work will attempt to provide insight into similarities and differences between objects and service as the basic concepts within the object and service orientation, as well as the influence they exert on the overall information system development.

Key words

Service, web service, orientation, object, design, business process, interoperability, development.

Introduction

Although the service concept is an ideal almost 20 years old, it has been in the focus of IT community only since the beginning of this century. The reason for this should be sought in the way of information system (IS) development, i.e. in the phase of analysis and design. Traditionally, during development, some parts of the organization (finance, human resources, accounting, etc.) were analyzed, and the IS itself included parts, where every part of IS covered a part of the organization. Instead of the 'silos' approach, more modern approaches imply the process approach, where parts of the organization are not in the focus of analysis, but rather its business processes, which should be automated.

Business process can be defined as an arranged set of activities transforming inputs into outputs. These processes become the principal subject of analysis and the basis of IS development. Very quickly, it was noticed that some of the activities in the given business process represented tasks performed by the system user, and some activities can be mapped into the software component. These components can be part of our system or part of a partner, external system, and in order to communicate, these components must be visible on the network. Instead of structuring the IS following the organization scheme of the enterprise, in accounting, finance, human resources, and so on, the IS increasingly resembles a collection of independent software components which are mutually connected in the way defined by the business process. The basic question is how to design and develop the component, which would be available on the network, to communicate freely through the organization's firewall and which is independent of the operative system and hardware platform.

The use of technologies that had previously been used in distributed system development, represented the first and logical solution, but the shortcomings that each of them had prevented them from responding successfully to all requirements dictated by the new approach. As an answer to the newly created situation, the service concept appeared, and the web service (WS) technology was adopted as the dominant technology for service development.

Just as the object used to represent the key factor of most activities of analysis, design and implementation, the service concept has become the basis of contemporary IS. How to develop a service successfully, what are similarities and differences with the object, and how to transfer

experiences from object-oriented development into service-oriented one are some questions that the paper will deal with.

1. Web services - theoretical background

Service as a concept appeared at the beginning of the 1990s as a consequence of development of technologies intended to support distributive systems. CORBA, DCOM, and Java RMI are technologies with the primary aim to develop applications structured from many parts which communicate mutually and where each of these parts has the possibility to be executed on another machine which is in our or external domain.

Common Object Request Broker Architecture (CORBA) is a standard developed by Object Management Group (OMG) enabling applications written in different program languages to communicate mutually, thus opening the possibility for the application in one organization to communicate with the application in another organization or for the applications within the organization to communicate transparently regardless of the technology in which they were developed and the operative system on which they are performed.

Distributed Component Object Model (DCOM) is a Microsoft standard and a direct competitor to the CORBA technology. It implies objects, which are distributed on the network, and the application calls their routines when needed. DCOM objects are tightly coupled with the operating system and often rely on their services.

Java Remote Method Invocation (Java RMI) is a standard developed by Sun Microsystems, Inc., and it represents a system enabling the object performed on one virtual machine to call the object performed on another virtual machine.

There are numerous reasons why these technologies are not good candidates for service implementation. Some of these reasons include the impossibility of communication between components developed in different technologies, heterogeneity of middleware on which software components are executed, problems of passing the company's firewalls, property communication standards, weaker support for the needs of large organizations, and so on. The most used technology for service development is currently the web service technology.

According to the World Wide Web Consortium (W3C), the Web service is defined as a software system developed to support interoperable communication between two machines through the computer network. The web service means the existence of the standardized interface written in the language understandable to the machine (WSDL) and the exchange of messages in the SOAP format. The web service for message transfer mostly uses HTTP, although many of them such as SMTP, HTTPS are also supported. The idea that the web service can be a good candidate for implementing the service concept became increasingly evident at the end of the 1990s, when, in order to improve communications and data exchange, SOAP and WSDL were standardized. Based on these two specifications, as well as the fact that it uses the tested, open technologies (XML, HTTP...), the web service technology was supported by different development tools, thus contributing to the faster adaptation of this technology in the development community.

The web service is independent of the program language; it can be developed in any technology, and from the standpoint of IS, the software and hardware platform is not important. The web service can be viewed as a 'black box' whose functionalities can be called through the computer network from one or more IS in the agreed way. As their application has been increasing, a large number of specifications have appeared around the web service in order to arrange their use and development. The following table gives the overview of the most important specifications and standards, stemming from the need to use the web services as integral parts of information systems of large organizations.

Table 1 Web service standards and specifications.

Transport
HTTP/HTTPS, SMTP, UDP, TCP, ...
XML
XML, XML Schema, XML encryption, XML digital signature
Messages
SOAP, WS-Addressing, MTOM, WS-Event notification, WS-Enumeration, WS-Transfer, ...
Security
OASIS Web Services Security, WS-Secure Conversation, WS-trust, WS-Federation, Kerberos Token profile, Username Token profile, X.509 Token profile, etc
Reliable Messaging Protocols
WS-Reliability, WS-Reliable Messaging
Transaction
WS-Coordination, WS-Atomic Transaction, WS-Business Activity
Metadata
WSDL, WS-Policy, WS-Metadata Exchange

Each of these specifications is open, and is maintained by one of the organizations for standardization. Some of them are OASIS, W3C, IEEE, etc. Owing to defined specifications arranging every field of application, web services have become the dominant way of developing the service concept.

Although it is not their natural purpose, web services can be developed and used as part of the current IS. In these cases, they are mostly a sort of adapter through which other systems use some of IS functionalities for which the service was developed. The natural environment for services is the service-oriented architecture or SOA.

Service-oriented architecture (SOA) cannot be defined in a simple way. The reason for this is the fact that the concept includes the word 'orientation', which, like object orientation, points to the programming style or paradigm, and the same concept contains the word 'architecture', which, in turn, points to the structure of application, i.e. the architectural style. Having this in mind, one of the attempts to interpret SOA is that it represents a software system including interoperable services capable of communicating mutually. These interoperable services can be implemented with different technologies, but only web service technology is the one that can completely face all requirements expected from the service. Today, when we talk about SOA, in most cases, service technology is understood as web services. Information systems that are based on services and whose functionality is realized by connecting services in the way dictated by defined business processes is called the service-oriented information system or service-oriented application.

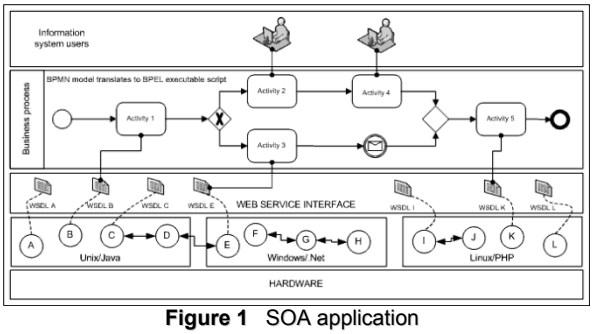

Figure 1 SOA application

Figure 1 illustrates a SOA application. The application includes several services marked by letters from A to L, which are developed and executed on different software platforms. Each of the services can communicate with the other services and every service can be accessed from the network; the ways and conditions of accessing are defined in the service interface, which can be read if its location, i.e. its URL is known. It must be noted that some of these services do not have to be in our domain, i.e. under our control at all. For example, the server, which hosts services I, J, K, and L on the Linux/PHP platform, is part of the information system of our business partner. Our business process has the possibility to use the process which is out of our domain, i.e. control (through the activity 5) in the identical way in which it is used in services that are in its domain, i.e. under its control. The business process itself does not know if the service is internal or external, nor does it have any knowledge of the technology in which it was developed, and these data are not important for process execution itself.

The business process is mostly modeled by means of the standardized Business Process Modeling Notation (BPMN). This notation, i.e. these diagrams have the possibility of translating into the execution language, Business Process Execution Language (BPEL), while the software called process server is the environment where translated diagrams are executed. The BPEL script has in itself descriptions and sequences of calling activities in the way as presented in the model, upon script initiation, it delegates tasks to system users and forwards service requests, which are part of the process.

Contemporary development tools have the possibility of automated taking out WSDL documents from the network and automated generation of the code of the class, through which they will call the functions of the service. Figure 1 also shows that some of the activities of the business process are suitable to business for which the user of IS is directly responsible, while some activities are appropriate to functionalities that one (web) service implements. The very fact that business processes are in the focus of the analysis phase and the services are inseparable part of process implementation, and also the fact that one service is not only part of a single process but it often happens that more processes share one service, speaks about the importance of high-quality choice and design of services which constitute this information system.

The basic service characteristics are the existence of contracts, loose coupling, abstraction, reusability, autonomy, statelessness, the possibility of discovery, and the possibility of service composition. Although we can find detailed descriptions of each of these characteristics in literature, as well as advice how to realize efficiently

the cited characteristics, there is still no consensus on how to notice, select and design in the best way the services which will meet all the cited characteristics. Many authors have identified activities and rules that can help to identify and select services based on the results of analysis, and based on business process models; however, the generally accepted set of steps that could guarantee the positive result still does not exist.

Similarities and differences between the object-based and service-based paradigm will be presented below in the paper. Comparing the basic principles of object orientation and development principles in general, we shall try to identify knowledge, mechanisms and experience that are applicable to service development or can be applied to them with minor changes. The comparison, presented in this way, would give development engineers, trained on the object-oriented paradigm, guidelines and instructions for as simple and fast transition to the service-based paradigm as possible.

2. Comparison of the object and service orientations

Object orientation is responsible for introducing order into unstructured development processes. The appearance of the object-oriented paradigm, at the end of the 1960s, interrupted the software crisis and fulfilled the vision of developing complete, reusable, scalable and flexible software solutions. Gradually, the object approach has been supported by defined development processes, conventions of UML languages, long-range practice, as well as a set of development patterns, turning this approach into the dominant development approach during information system development. Object orientation can be said to be the most complete and the most mature development framework today.

The object and service orientations try, through their methods, techniques and rules, to fulfill identical objectives. Both paradigms try to develop applications which will successfully service their users' requirements, but at the same time, to be simple to maintain and capable of coping successfully with continual changes, which characterize today's business.

Service orientation appeared as a response to the situation where traditional development approaches could not respond to current requirements in an acceptable way. Service orientation appeared on the foundations of object orientation and as such, has many similarities with it; however, they look at the common concepts such as the class, object, methods, attributes, interface and messages differently.

One of the most noticeable differences between the two paradigms is in the scope. Although in the world of object orientation there is no limit regarding the extent to which its principles can be applied, practice shows that objects have never exceeded the limits of the application or the group of applications for which they were developed. The reusability of objects has mostly been at the level of the users' components where the resulting component libraries were distributed through the applications being developed. Service orientation and the service as its basic material concept are indeed trying to exceed the limits of the application itself. Services are designed primarily to include one logically comprehensive business entity and to render their functionalities available on the network. Such autonomous services become parts of business processes, both internal and external. This was exactly the way to contribute to the flexibility of the software solution because most business changes are related to the business process itself, whose modification does not require changes on the service. Also, the fact that the functionalities of the service, which need not be part of our IS, can be directly mapped into the activity of internal business process solves one of the key requirements – the requirement for integration of both internal and external systems. Such a way of integration is independent of technology on which the service is developed, and the hardware-software platform on which the service is executed.

In the next section, comparison of the basic concepts, principles and techniques in the object and service orientations is given.

Class and service

The class, as a basic concept of object orientation, represents a tool for defining the static system structure. In the course of program execution, objects appear as the instances of the class. Class can be regarded as the template based on which objects are created, where, in the course of execution, each of them has its state and contains some data. The service also tries to organize the business logic in a complete entity, but it is not quite comparable with the class because of its specificities. The class encapsulates information and behavior, while the service contract defines only public information. The class defines attributes and operations which change them, and

calling these operations, the states of the object change. In order to maintain statelessness, services do not implement property; a service is expected not to have state, and to behave in the identical way for every call. Similarities are seen more clearly when comparing the service contract and the interface implemented by the class. The service contract means one or more technical service descriptions (WSDL, definition of XML scheme, description of WS-policy), intended for using in the course of service execution.

Interface and service contract

The interface represents abstraction including methods implemented in one or more classes. The interface encapsulates the methods of classes, rather than implementing. Each of encapsulated methods is implemented in its own class. When they are used, interfaces form an additional layer over the classes and represent the points of entry for accessing the classes. Based on the above, it can be concluded that interfaces abstract class implementation.

Service orientation also has a similar construct, the service contract. The service contract is one of the most important constructs of service orientation; all the operations implemented by the service can be seen through it, as well as the necessary data for access and service call. As well as the interface, the service contract forms an additional layer for access to service operations and abstract the service implementation itself. The service contract is mostly implemented through WSDL or Web Service Definition Language.

Method and possibilities

Classes in the object-oriented world implement methods and attributes. Methods represent the functionality that the classes have, while attributes represent the data that the class uses (the term *variables* is also used). The class properties represent the predefined states that the object can have. Methods and properties can be declared to be private or public; it is usual to declare as public those methods and properties which really need it. For the service, it can be said that it has possibilities. Possibilities are equivalent to the class methods. The service contract cannot define private operations, and in order to maintain statelessness, we try not to define attributes.

Object-oriented and service-oriented message

Communication between objects flows through message exchange. In this case, the 'message' represents an abstract notion, part of the OOAD vocabulary, and, as such, does not explain what the messages are like in the real word. As object orientation is mostly applied on the components whose communications are based on the non-standard communication protocols, messages are usually transferred as binary communication sets, which are exchanged synchronously. It is usually a RPC-based mechanism. Messages exchanged through service implementations are usually textual. Communication can be synchronous or asynchronous, and in this context, the message has a more usual meaning, such as, for example, the system of e-mail. Input and output values of (web) service operations are structured by means of the complex types of XML scheme. As the input and output values of service operations can be very complex and are always structured by the message, input and output operation parameters are never specified when representing services in the diagram.

The differences are also noticeable at the level of the granularity of the connection through which the communication is done. Objects try to implement very fine grained methods, in contrast to service operations, which are, as a rule, coarse grained. This is because the communication between objects (whether local or distant) is realized as persistent. Once it is established, the connection is maintained, and data exchange is undisturbed. Services, on the other hand, are often supported by one of standard communication protocols, such as HTTP, to exchange messages. The connection which is realized is unstable, so that data exchange is executed through sending messages. Having this in mind, service operations are designed in this way. As an argument, they receive a message which contains a series of complex types of data, and sometimes the whole business document. As in many cases the service represents the wrapper around a class, in Figure 2 we can see the difference between the class designs when they are developed as object- or service-oriented.

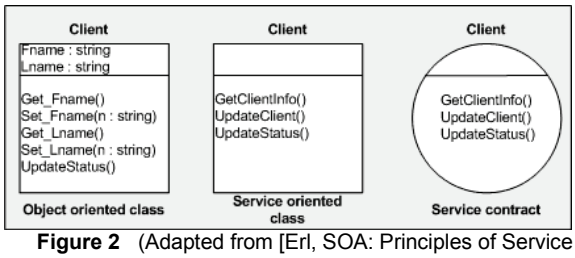

Figure 2 (Adapted from [Erl, SOA: Principles of Service Design, avgust 2005]).

Figure 2. illustrates how object and service orientation influences class design. The class designed according to object principles includes the private part with attributes, as well as the public one, which contains methods that are accessible. Access to class functionalities is designed in detail, therefore with object-oriented classes there are many more methods and attributes where each of them is responsible for a small job segment. These methods are said to be fine grained. The class designed according to service principles has no private part, while its methods are developed in order to include a bigger job segment. These methods are said to be coarse grained. Service classes are more oriented to messages, in contrast to the object ones, which are more oriented to the exchange of trivial data. The service contract, with which the service is typically represented in diagramming, includes only titles of operations which are available through this service. The service contract itself contains all needed data related to the service such as location, the way of connecting, operations, arguments of operations, etc.

Object-oriented and service-oriented encapsulation

In the broader sense, encapsulation denotes the inclusion of a smaller thing within a larger thing so that the included thing is not apparent. In object orientation, encapsulation means hiding information, i.e. encapsulation realizes one of the basic object principles – hiding information from the external world. The object is a container, and access to it is possible only through public methods, while everything else remains hidden. This principle can be compared to the principle of service abstraction, which also requires hiding information. As well as objects, services encapsulate logic and implementation; however, the term 'encapsulation' in the service world means what can be included by service. Encapsulation in the object paradigm represents hiding information about the class, while in the service paradigm, it makes decides which part of business logic is appropriate for implementing of the given service.

Inheritance

One of the principal ways of repeated use of the code, in the object paradigm, is inheritance mechanism. Two classes can form the parent-child relationship, where the child will have all the properties of the parent class and some additional personal properties. This process is called *specialization*. In the service paradigm, inheritance is not used because of the existence of the principle of service autonomy. Services do not implement one another and therefore do not form the parent-child relationship. During service design, it is necessary to aim at as loose coupling as possible. It is necessary to note that it is possible to inherit interface in the web service technology. This can be realized since WSDL v.2 became current through the attribute of *interface* element under the name of *extends*.

A well designed class of the higher level, often called the abstract class, enables creating a number of subclasses. Generalization is realized when the parent class (the class of higher level) is noticed and created, which will later be inherited by other classes. Other classes will specialize the class of higher level. A concept similar to generalization and specialization in the service world is related to granularity. As already mentioned, services do not implement one another so there is no specialization or generalization which appears as the relationship between the classes. In this context, it refers to the extent to which service operations are detailed (service granularity). The more special service is, the more detailed it is, i.e. granularity is at a higher level. Establishing the right degree of granularity of service operations is a very important task during service design. Services whose operations are detailed (fine grained), are referred to as specialized, while services whose operations are general, large (coarse grained), are referred to as generalized.

If we try to develop service from the existing systems, it is more sensible to create 'larger' services, which would include several functionalities of the existing systems. As a rule, one service should include operations working mutually and thus comprising a logical, functional entirety. In the case of service development from the existing applications where the observed functionalities are tight-coupling, services should be designed so as to include functionalities of tight coupling and make a logical entirety. These services can be referred to as coarse grained.

When we develop services from the beginning,

it is the rule to develop smaller services with a high possibility of repeated use. The price of services developed in this way is paid in orchestrations, which will use these services through a series of business processes (BPEL process). These services can be referred to as fine grained.

When determining which functionalities the service will include, attention must be paid to the following:

- Cut decoupling, i.e. breaking functionalities when the point is reached where a function will be broken into several tightly coupled functions. Decoupling is, therefore, done to the level when, as the result of breaking one function, functions appear between which there is no dependence.
- Decoupling should be done to the level where the resulting function is important for the future consumer of the function. Users usually do not need all the functions available in service.

How strong coupling between the client and the service is can be resolved through the following questions:

- How simply can logic inside the service be changed without changing the way of accessing the service?
- How much is the client protected from the changes coming with the increase in service possibilities?
- How simply can a service be orchestrated without changing the service itself?
- To what extent is the client dependent on the service availability?
- Can the client work if the service is unavailable?
- Is the service dependent on the state?

Polymorphism

Polymorphism means the capability of somebody or something to appear in several forms. Through object orientation, this concept can be realized in several ways. When several subclasses in the objective paradigm, inherited from one parent class, have the same name of a method, and every subclass applies different implementation of this method, such a phenomenon is called polymorphism. Each of these classes is a specialization of a superclass (parent class), and, as such, has different implementation. As a result of this phenomenon, when the same message is sent to the same method, and a different sub-class, obtained results in different subclasses are also different. Besides, there is the example of virtual functions through which it is also possible to realize the concept of polymorphism.

There is no identical designing principle in the service paradigm. The closest to this concept is the existence of a standardized principle of the service contract, whose application results in the existence of methods with similar or the same names. The typical example is the application of CRUD (Create, Read, Update, Delete) names in the entity services. It should be noted that if two or more services have the same names of methods, it does not mean that these services can receive the same message. Therefore, we can state that there is not a concept in the service world which could be compared in the right way to polymorphism in the object world.

3. Future research

There are currently many methodologies for software development. They can be described as a framework for structuring, planning and control of the information system development process. Each of the concrete methodologies has its advantages and disadvantages, and the choice of the methodology depends on the nature of the system being developed. Some of the specific methodologies are SSADM, SCRUM, RUP and SDLC.

Software Development Life Cycle (SDLC) is considered to be the oldest formalized methodology for information system development. This methodology implies a disciplined and methodical approach to each of the phases of the software life cycle, where transition from one phase to another is conditioned by ending the previous one. Today, there are many methodological approaches based on the development of the software life cycle; moreover, today the term SDLC means a family of methodologies based on the SDLC methodology.

The vision of making an IT solution, as a set of services rather than a set of hardware and software, permeates the whole ITIL methodology. Information Technology Infrastructure Library (ITIL®) is a methodology developed by the Government of Great Britain with the purpose of cost and time reduction and increasing the efficiency of information system development process. This methodology represents a catalogue of systematized best experiences intended for IT organizations. The current version is v3, consisting of five parts, where each of them contains

guidelines for one or more phases of the life cycle. This methodology is built on the process-based, i.e. holistic view of the control of organization management, and as such it imposes itself as the best candidate methodology for information system development today.

The increasing complexity of users' and technological requirements has influenced IT organizations to initiate the process of selecting and accepting a new methodology, which would enable them to face the newly created situation more successfully. Many indicators point to the fact that more and more IT organizations decide to adopt ITIL, in order to attain goals more successfully and efficiently. However, experiences have shown that during this process, development groups inside the organization (those who develop IS) still try to keep some of the tested SDLC approaches, while the infrastructure groups (all those who do not develop IS) easily accept the ITIL approach. The reason for this is noticed in the shortcomings observed in ITIL parts which are the most important for the development groups. The relevant issue is the design process. The development groups who tried to develop the system following ITIL very quickly concluded that the procedure was incomplete and unripe. Part of ITIL dealing with service design contains qualitative guidelines principles and advice for design, but there are no control mechanisms to define clearly the beginning, the middle and the end of the design process. In these circumstances, development groups are not willing to give up the well-developed and mature procedure, which they have been using so far and with which they have experience.

However, research efforts should not be oriented to finding the answer whether to keep traditional approaches and try to update them, or yield to a new ITIL development direction, and try to find the ways to overcome the noticed shortcomings. Future efforts should be channeled towards the transfer of the best from the traditional approach to the new one, ITIL approach, for example. The synergetic effect produced by this procedure would provide the development groups with a new methodology, which would enable them to cope successfully and continuously with the complex requirements and complex technologies. Indeed, noticing the similarities and differences between the object and the service, which are most frequently the focal point of design efforts of traditional and new methodologies, represents the basis and starting point in the future research.

4. Conclusion

Although the service and object orientation represent two different approaches, we will not make a mistake if we say that the service-oriented approach enriches and broadens the object-oriented one. Strategic goals, for example, attained by the application of object orientation such as flexibility, reusability and extensibility are completely taken over from the service-oriented approach. The service-oriented approach tries to enrich goals taken over from object orientation with a set of new goals originating from the new, process-oriented or holistic view of the organization for which the system is being developed. These goals include faster return on investment, organizational agility, system federation, business harmonization with software support, and the like. The analysis of these two approaches points to the following facts: that object orientation is now the most mature and the most used development procedure, and therefore, service orientation can be considered to be the extension or enrichment of object orientation, and that there is not yet a complete and accepted methodological approach for service development and service architecture. Having this in mind, as well as the interest in accepting service orientation as soon as possible, it is necessary to provide development groups with guidelines and advice for accepting the service concept, and, in perspective, a reliable methodology for service solution development. As the great majority of developers have acquired experience on object orientation and that the object is their natural central point, this advice and guidelines should interpret services and their properties through objects to the greatest possible extent. They also should try to apply the best practices of the object-oriented approach to service-oriented one, wherever possible.

References

Dirk Krafzig, K. B. (2004). *Enterprise SOA: Service-Oriented Architecture Best Practices.* (D. O'Hagan, Ed.) Upper Saddle River, NJ 07458, USA: Prentice Hall PTR.

Erl, T. (2005). *Service-Oriented Architecture: Concepts, Technology, and Design.* Upper Saddle River: Prentice Hall PTR.

Erl, T. (2005). *SOA: Principles of Service Design.* Upper Saddle River: Prentice Hall PTR.

M, N. (2007). *Josuttis SOA in Practice* (1st ed.). (S. St.Laurent, Ed.). Sebastopol, CA : O'Reilly.

Margolis, B., & Sharpe, J. (2007). *SOA for the Business Developer: Concepts, BPEL, and SCA*. Lewisville, TX: MC Press.

Nichols, D. (2009, February 3). *Establishing a Service Design Methodology*. Retrieved March 15, 2009, from itSM Solutions LLC: http://www.itsmsolutions.com/newsletters/DITYvol6iss16.htm

Nichols, D. (2008, June 3). *SDLC or ITIL? Wrong question*. Retrieved March 15, 2009, from itSM Solutions LLC: http://www.itsmsolutions.com/newsletters/DITYvol4iss23.htm

Sanjiva Weerawarana, F. C. (2005). Web Services Platform Architecture: SOAP, WSDL,WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More. Upper Saddle River: Prentice Hall PTR.

Nebojša Taušan
University of Novi Sad
Faculty of Economics Subotica
Segedinski put 9-11
24000 Subotica
Serbia
Email: nebojsa@ef.uns.ac.rs

Pere Tumbas
University of Novi Sad
Faculty of Economics Subotica
Segedinski put 9-11
24000 Subotica
Serbia
Email: ptumbas@ef.uns.ac.rs

Predrag Matković
University of Novi Sad
Faculty of Economics Subotica
Segedinski put 9-11
24000 Subotica
Serbia
Email: pedja_m@ef.uns.ac.rs