Component-Based and Service-Oriented Software Engineering: Key Concepts and Principles

Hongyu Pei Breivold, Magnus Larsson
ABB AB, Corporate Research, 721 78 Västerås, Sweden
{hongyu.pei-breivold, magnus.larsson}@se.abb.com

Abstract

Component-based software engineering (CBSE) and service-oriented software engineering (SOSE) are two of the most dominant engineering paradigms in current software community and industry. Although they have continued their development tracks in parallel and have different focus, both paradigms have similarities in many senses, which also have resulted in confusion in understanding and applying similar concepts or the same concepts designated differently. In this paper, we present a comparison analysis framework of CBSE and SOSE and analyze them from a variety of perspectives. We discuss as well the possibility of combining the strengths of the two paradigms to meet non-functional requirements.

The contribution of this paper is to clarify the characteristics of CBSE and SOSE, shorten the gap between them and bring the two worlds together so that researchers and practitioners become aware of essential issues of both paradigms, which may serve as inputs for further utilizing them in a reasonable and complementary way.

1. Introduction

Today, designing and implementing a large scale and complex system has been a challenging task. Two of the most well recognized software engineering paradigms coping with this challenge are: component-based software engineering and service-oriented software engineering.

Component-based software engineering (CBSE) provides support for building systems through the composition and assembly of software components. It is an established approach in many engineering domains, such as distributed and web based systems, desktop and graphical applications and recently in embedded systems domains. CBSE technologies facilitate effective management of complexity, significantly increase reusability and shorten time to market. On the other hand, the growing demands for Internet computing and emerging network-based business applications and systems are the driving forces for the evolvement of service-oriented software engineering (SOSE). Service-oriented design utilizes services as fundamental elements for developing applications and software solutions. Service-oriented design technologies offer great feasibility of integrating distributed systems that are built on various platforms and technologies and further push focus on reusability and software development efficiency.

SOSE has evolved from CBSE frameworks and object oriented computing [16] to face the challenges of open environments. Therefore, CBSE and SOSE are similar to each other in many senses. Both use similar approaches and technologies. Both have software architecture as the common source and base. Meanwhile, both paradigms have continued with their development tracks in parallel and have different focus. Consequently, the mixture of similarities and specialized utilization of concepts in CBSE and SOSE have also resulted in confusion in understanding and applying concepts in a correct way. This may lead to less efficient utilization and combination of these paradigms. Furthermore, since both CBSE and SOSE can co-exist in enterprise systems and complement each other [17], any divided understanding and different interpretation of the terminologies would lead to less efficient combination and adaptation of these paradigms in future software development. For these reasons, it is important to clarify the concepts, principles and characteristics of CBSE and SOSE, shorten the gap between them and bring these worlds together so that researchers and practitioners can become aware of both sides. This clarification may serve as inputs to the subsequent investigation in how to take advantages of the strengths of these two paradigms, how to adapt and integrate the component-based and service-oriented technologies, concepts and their strengths so that both component-base and service-oriented software engineering can complement each other to the ultimate extent.

The goal of this paper is to provide a clarification framework of the component-based and service-oriented software engineering to avoid any misunderstandings and misuses. A brief discussion of

reasonable utilization, combination and adaptation of the two paradigms is also outlined through looking into a set of research studies in how they have been used. These studies have exampled the benefits of improved quality attributes of software solutions through combining CBSE and SOSE. Since both paradigms are evolving rapidly, there exists increasing research interest in further exploration of their combination potentials. We contend that a good understanding of respective characteristics is a necessary step for this exploration.

The remainder of the paper is structured as follows. Section 2 presents overview of component-based and service-oriented software engineering. Section 3 gives a comparison analysis framework of the two paradigms from different perspectives, including key concepts and principles, process, technology and composition. Section 4 discusses state of the art research in combining the strengths of CBSE and SOSE. Section 5 concludes the paper.

2. Overview of component-based and service-oriented software engineering

Component-based software engineering (CBSE) is a software engineering paradigm that aims to accelerate software development and promote software reusability and maintenance through assembling components to software systems that meet certain business requirements. The prerequisite requirements that enable components to be integrated and work together are component models and component framework [20]. Component models specify the standards and conventions that components need to follow during component composition and interaction. Component framework provides design time and run time infrastructure.

Numerous component models exist nowadays. Some examples are COM/DCOM/COM+, .Net component model, JavaBeans, Enterprise JavaBeans and CORBA component model. Examples of component models that have been developed specifically for applications to embedded systems include Koala [11], Rubus [21], PECOS [18].

Important areas of research within CBSE include, but not limited to, determination and specification of QoS (Quality of Service), predictability of non-functional properties, component interference and process related activities such as component classification, identification and selection, component adaptation, testing and deployment techniques.

Although CBSE has proved to be successful for software reuse and maintainability, it does not address all of the complexities software developers are facing today, such as varying platforms, varying protocols, various devices, the Internet, etc [7]. Service-oriented software engineering paradigm has emerged to address these issues.

Service-oriented software engineering (SOSE) is a software engineering paradigm that aims to support the development of rapid, low-cost and easy composition of distributed applications even in heterogeneous environments [13]. It utilizes services as fundamental elements for developing applications and solutions.

Important areas of research within SOSE include service foundations, service composition, service management and monitoring and service-oriented engineering [13]. Service foundations provide service-oriented communication technologies to support run time service-oriented infrastructure and connect heterogeneous systems and applications. These communication technologies provide the communication mechanisms between service providers and service requesters; they differ with respect to service description techniques and messaging functions [6]. Service composition encompasses necessary roles and functionality to support service composition [13]. The dynamic composition feature in SOSE makes QoS a major challenge. Different initiatives have emerged such as orchestration and choreography. Service management encompasses the control and monitoring of SOA-based applications throughout life cycle.

A key element in SOSE is the service-oriented interaction pattern, i.e. service-oriented architecture (SOA), which enables a collection of services to communicate with each other. SOA is a way of designing a software system to provide services to applications or other services through published and discoverable interfaces. The basic elements of service-oriented architecture are illustrated in Figure 1.

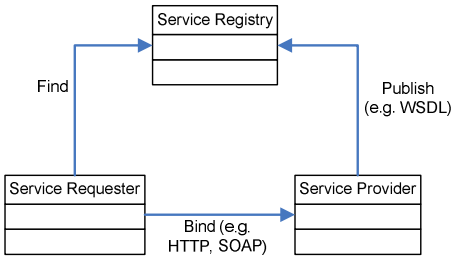

Figure 1 Service-oriented interaction pattern

As shown in Figure 1, SOA has three main actors: a service provider, a service requester and a service registry. The service provider defines service descriptions of a collection of services, supplies services with functionalities and publishes the descriptions of the services so as to make the services discoverable. The service registry contains service descriptions and references to service providers and

provides mechanisms for service publishing and discovery [14], e.g. Universal Description, Discovery and Integration (UDDI). The service requester is a client that calls a service provider. It can be an end-user application or other services. A service requester searches in the service registry for a specific service via the service interface description. When the service interfaces match with the criteria of the service requester, the service requester will use the service description and make a dynamic binding with the service provider, invoke the service and interact directly with the service.

3. Classification of component-based and service-oriented software engineering

The main concepts and principles of CBSE and SOSE may look similar at the first sight, but differences exist in mechanisms, approaches and implementations. Therefore, we group particular characteristics that have similar concerns to describe the same or related aspects of CBSE and SOSE. The categories in the comparison framework that we are going to address are: key concepts and principles, process concerns, technology concerns, quality and composition.

3.1. Key concepts

A summary of the key concepts in CBSE and SOSE is listed in Table 1.

Table 1 Comparison of key concepts in CBSE and SOSE

Concepts	CBSE	SOSE
Module	Component	Service
Specification	Component contract	Service description
Interface	Component interface	Service interface
Assembly	Component composition	Service composition

3.1.1. Module. In CBSE, components are the building blocks that can be deployed independently and are subject to composition by third party [4]. Based on the formulation by Clemens Szyperski [15], a software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. It can be both fine-grained providing specific functionality and coarse-grained encompassing complicated logics.

In SOSE, services are the building blocks that can be reused and offer particular functionalities. They are generally implemented as coarse-grained discoverable software entities [2], operating on larger data sets, encapsulating business functionality and exposing the functionality to any source that requests the functionality through well-defined interfaces. Thus, the services can be reused and accessed at various levels of the enterprise application and even across enterprises boundaries.

3.1.2. Specification. In CBSE, the component specification provides for the clients the definition of the component's interface, i.e. the operations and context dependencies. Furthermore, an abstract definition of the component's internal structure is specified for the component providers [4].

In SOSE, the service description is a service contract that advertises the following information: (i) service capabilities - stating the conceptual purpose and expected results of the service; (ii) interface - describing the service signatures of a set of operations that are available to the service requester for invocation; (iii) behavior - describing the expected behavior of a service during its execution; and (iv) quality - describing important functional and non-functional service quality attributes [12].

3.1.3. Interface. Although both CBSE and SOSE are interface-based in the sense that interfaces are the specifications of access points, the separation between service descriptions and service implementation is more explicit than the separation between component specification and implementation.

3.1.4. Assembly. In CBSE, component composition is the process of assembling components using connectors or glue code to form an assembly, a larger component or an application. The components are assembled through the component interfaces and the composition is made out of several component instances that are connected and interact together.

In SOSE, the composite services are built by composing service descriptions. The realization of the service composition is during run time when the service providers are discovered and bound.

3.2. Key principles

A summary of the key principles of implementation in CBSE and SOSE is listed in Table2.

Table 2 Comparison of key principles of implementation in CBSE and SOSE

PRINCIPLES	CBSE	SOSE
Coupling	Loose and tight coupling	Loose coupling
Self describing	Component specification	Service descriptions
Self contained	yes	yes
State	Stateless/stateful	Stateless/stateful
Location transparency	In some component models e.g. DCOM	yes

3.2.1. Coupling. CBSE enables both loose coupling and tight coupling. As a component is used within the scope of a component model, it needs to conform to the rules specified by the component model. A component model often uses one particular interaction style, such as broadcasting, asynchronous connection and connection-oriented style. All these interaction styles imply some kind of coupling between components, such as referential coupling and temporal coupling.

In contrast to CBSE, SOSE enables only loose coupling, with minimized dependencies between service providers and service requesters. The service providers need not to know anything about the service requesters or any other services. They have great flexibility in choosing their design and deployment environment to offer their services. Likewise, the service requesters or calling applications need not to know anything about underlying logic of the service implementation and service deployment except the service descriptions. The service descriptions are the only communication channel between service requesters and service providers. Service loose coupling is enabled through the use of service descriptions that allow services to interact within predefined parameters [5].

3.2.2. Self describing. Both CBSE and SOSE share the same self describing characteristic with their own specialization. In CBSE, the component specification is the key to the component's self describing characteristic and specifies the rules that the components must conform to.

In SOSE, the service description is the key to the service's self describing characteristic. The service provides its clients with all the relevant information in the service descriptions, which contain combinations of syntactic, semantic and behavioral information.

3.2.3. Self contained. In CBSE, components can be self contained. For example, for CCM, a component is 'a self-contained unit of software code consisting of its own data and logic, with well-defined connections or interfaces exposed for communication. It is designed for repeated use in developing applications; either with or without customization' [22].

In SOSE, services are self contained. The services provide the same functionality regardless of the other services, even if any other services may fail for some reason.

3.2.4. Stateless. Both components and services can be stateful or stateless. In SOSE, stateless services are used to meet the performance requirements and in some circumstances, the stateless property is optimal for services' reusability. As a result, the services should minimize the amount of state information they manage and the duration for holding the message information. Otherwise, the services would not be able to timely correspond to other service requesters. On the other hand, there are circumstances when stateful services are necessary so as to maintain states across several method calls by the same service requester. The service object creation policy determines whether a stateful service can be returned.

3.2.5. Location transparency. In CBSE, some component models can provide location transparency, e.g. DCOM allows component-based applications to be distributed across memory spaces or physical machines using proxies and stubs.

In SOSE, since services have their descriptions and location information stored in the service registry through e.g. UDDI, which is accessible to a variety of service requesters, services can be invoked by service requesters from different locations.

3.3. Development process concerns

Three aspects related to development process are identified for further comparison.

3.3.1. Building from pre-existing entities (components or services). The main idea for CBSE is to build systems from pre-existing components. This feature applies in the same way for SOSE in the sense that systems can be built from composing appropriate pre-existing services to meet certain business functionality.

3.3.2. Separation of development process of system and entities (components or services). In CBSE, the development process of component-based systems is separated from the development process of components. This feature applies in the same way for SOSE in the sense that services can be developed by various service providers across organizational boundaries and the service requesters need only to discover and invoke the services.

3.3.3. Development process. In CBSE, engineering a component-based software system is a process of finding components, evaluating and selecting proper components, testing, adapting if necessary and integrating the components into the software system, e.g. in the COTS-based development process. In SOSE, engineering a service-oriented computing system is a process of discovering and composing the appropriate services to satisfy a specification [8]. The process of service discovering, matching, planning and composing is essential. Service-oriented engineering process focuses more on run-time activities, such as dynamically adding, discovering and composing services illustrated in Figure 2.

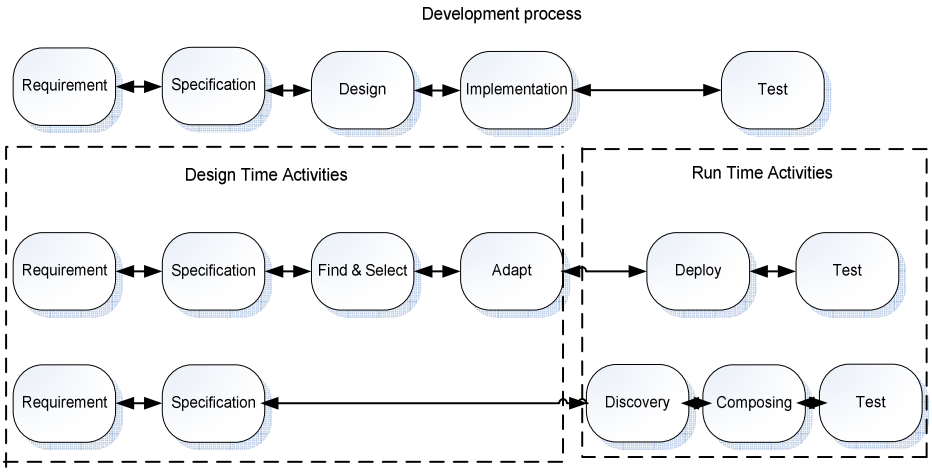

Figure 2 Comparison of typical activities during development process in CBSE and SOSE

3.4. Technology concerns

Three aspects are identified for further comparison: technology neutrality, encapsulation and static or dynamic behavior.

3.4.1. Technology neutrality. In CBSE, components need to conform and follow the rules that are set up by a specific component model. As a result, the feasibility to compose components of different component models is relatively limited. On the other hand, compliance to a certain technology may also lead to advantages in the sense that many solutions can be optimized since they can be directly supported by the specific technology.

In contrast to CBSE, SOSE provides the feasibility for services to be implemented in diverse technologies and for multiple applications running on different platforms to communicate with each other. This feasibility is enabled through applying commonly accepted message standards for interface descriptions to the services. Hence, the enterprise applications or solutions can cut across technology and platform boundaries, performing business functionalities by composing services from different sources of service providers.

3.4.2. Encapsulation. Encapsulation means that the business logic and implementation are shielded from the outside world. CBSE supports a variety of encapsulation types, ranging from white box exposing all the implementation, or gray box exposing parts of component implementation to black box. In the cases of white box and gray box, the component clients have the flexibility to make modifications to the component in order to meet specific needs in their solutions.

In contrast to CBSE, SOSE supports only black box encapsulation. The logical view of a service consists of one or a set of service interfaces and service implementation. A service can be regarded as a business logic entity which can be accessed and executed through the well-defined and formal interfaces by any service requester that wants to use the service. This is called the service interface level abstraction [5], which enables the services to act as black boxes, leading to the inflexibility of service requesters to modify services.

3.4.3. Static vs. dynamic. Two aspects are concerned:

(1) Binding

There are two types of binding: early binding and late binding. Early binding allows clients to obtain compile-time type information from the component's type library. Late binding allows clients to bind to components at run time and the compiler has no clue during build time about the method calls that are to be made at run time.

CBSE allows static early binding and supports dynamic late binding in some component models. An example is early and late binding to COM components. In early binding, the components are instantiated as needed and invocations of operations are based on the interface definitions, statically checked and bound to by the compiler. In late binding, components are bound by invoking IDispatch methods in COM that redirects dynamically to the sought interface. The choice of static or dynamic binding has both pros and cons, and consequently need to be taken into consideration during design. Static binding between components may lead to the disadvantage of less flexibility in facilitating changes, but it allows for stronger type checking during compile time and is much faster than the late binding approach.

SOSE allows only dynamic binding. The service requesters make targeted named calls and search in the service registry for a specific service. When the service

requesters find the services that match certain criteria, the service requester will use the service description to make a dynamic binding with the service provider.

(2) Dynamic discovery and availability

Discovery implies the ability that an entity (component or service) is discovered for use. Availability is the ability that an entity (component or service) is operational or accessible when required for use. In CBSE, dynamic discovery and dynamic availability of components are not the major concerns [3].

In SOSE, services exhibit the feature of dynamic availability, since they can be added or removed from the service registry at any time. Consequently, services are readily available running entities and need to be dynamically discovered and composed in run time.

3.5. Quality concerns

Quality attributes can be classified into life cycle properties and run time properties. Hundreds of quality properties exist and we can not analyze all of them. Therefore, we choose only quality attributes that are of common or related interest to CBSE and SOSE.

3.5.1. Reusability as life cycle property. CBSE emerged to accelerate reusability of software. However, there are some constraints in achieving component reusability, such as component specification should be explicit, no architectural mismatches among composed components, etc.

Similar to CBSE, services can be reused to construct applications. In SOSE, the concern in having similar architecture needs not to be taken into consideration because of the technology neutrality, platform independence and interoperability characteristics of SOSE. On the other hand, extra emphasis is put on having explicit service descriptions.

There are several factors that contribute to the reusability of components and services. Firstly, both components and services are composable. This implies that the level of granularity of components and services need to be considered when taking reusability into account. The design of operations should be in a standardized manner and with appropriate level of granularity [5] so that the components or services can be reused and composed. Secondly, the separations between component/service development and applications also promote component and service reusability.

Recently, researchers have been active in investigating the possibilities of enhancing service reusability with service-oriented architectures. One study is presented by Zhu in [19], where he proposed the idea that services are new types of components and service-oriented architectures may provide more chances for the development of reusable components.

3.5.2. Substitutability as life cycle property. Substitutability means that alterative entity (component or service) implementation may be used with the constraints that the system can still meet the requirements on functional level and non-functional level. According to [15], white box and gray box reuse very likely prevents the component substitutability. In such cases, explicit conventions about the implementation information and changes that are made in components are required to achieve substitutability [4].

In SOSE, since the service-oriented interaction pattern enables the loose coupling characteristic between a service requester and service providers, services can be substituted with new services as long as the service descriptions fulfill the criteria from service requesters.

3.5.3. Interoperability as runtime property. The main idea in CBSE is to assemble components together to perform certain functionality. However, each component conforms to a certain component model that specifies different rules from another component model. Therefore, interoperability between heterogeneous components is still a challenging issue in CBSE. Although in some circumstances, interoperability can be achieved through implementing wrapper class or proxies.

On the other hand, broad interoperability among different vendors' applications and solutions can be achieved in SOSE through the use of well accepted standards. For instance, WSDL, UDDI, SOAP, XML [23]. These descriptions are independent of underlying platform, programming languages and implementation details and therefore promote interoperability.

3.6. Composition concerns

3.6.1. Heterogeneous vs. homogeneous composition. In CBSE, components can only be assembled according to the rules specified by a specific component model; there is not much feasibility to assemble components that conform to different component models.

In SOSE, services which access and combine information and functions from different sources of service providers can be assembled into composite services to perform particular tasks [12]. The service-oriented software engineering principles, such as services are platform independent and loosely coupled, offer the feasibility that services from different sources of service providers can be used in the same composite service.

3.6.2. Design time/run time composition and composition mechanisms. In CBSE, components can be composed at design time and run time. Design time composition allows for optimization [4]. A component detaches its interface from its implementation, and conceals its implementation details, hence permitting composition without need to know the component implementation details [1]. The mechanisms for component composition vary from method calls, to pipes and filters or event mechanism [4]. Furthermore, component models provide also general architecture and mechanism for component composition. For example, component models require components to support introspective operations to enable component composition at assembly time or run time [17], e.g. the functionality and properties of the components can be discovered and utilized automatically at assembly time or run time.

In SOSE, services are composed at run time. Several mechanisms exist to compose services, such as pipe and filter which can direct the output of one service into the input of another service, orchestration and choreography. Orchestration utilizes a high-level scripting language to control the sequence and flow of service execution. It describes the behavior and interactions of a specific service provider with other involved services. BPEL4WS (Business Process Execution Language for Web Services) and WSCI (Web Service Conversation Interface) are examples of web service orchestration languages. Choreography describes the interactions between service providers that are collaborated for achieving business functionality. WS-CDL (Web Service Choreography Description Language) [24] is one example of choreography languages.

3.6.3. Predictability. In CBSE, the predictability of non-functional properties of the composition components from the properties of components remains to be a challenging issue. However, compared with SOSE, the use of static binding in CBSE may provide to a certain extent better predictability because of the clarification of interface-based design during assembly time.

To some extent, SOSE faces even more challenges in predictability because of its dynamic discovery and dynamic availability behaviors. Some of the examples of the challenges include how to predict the quality of service when services are discovered and invoked dynamically during run time, how to predict the quality properties when services are composed at run time? These are still interesting open research issues.

Based on the above comparison analysis, the main similarities and differences between CBSE and SOSE are summarized in Table 3.

Table 3 Summary of similarities and differences of CBSE and SOSE

	CBSE	**SOSE**
Process	Building system from pre-existing components. Separate development process of components and system. More activities involved in design time	Building systems from pre-existing services. Separate development process of services and system. More activities involved in run time
Technology	Constrained by component models. Ranging from white box, gray box to black box. Static and dynamic binding between components. Dynamic discoverability is not a major concern	Platform independency. Black box. Only dynamic binding between services. Dynamic discoverability
Quality	Interoperability concern between heterogeneous components. Achieve component substitutability through explicit specifications. Better predictability	Interoperability through universally accepted standards. Achieve service substitutability through service descriptions. Predictability issue
Composition	Homogenous composition. Design time and run time composition and design time composition allows for optimization. Pipe and filter; event mechanism etc. Composition is made out of several component instances	Heterogeneous composition. Services are composed at run time. Pipe and filter; orchestration etc. Composite services are built by composing service descriptions

4. Discussions

Because of the diverse nature of software systems, it is unlikely that systems will be developed using a purely service or component-based approach [10]. Therefore, the ability to combine the strength of CBSE and SOSE and use them in a complementary manner becomes essential. So far, a lot of research has been done in combining the strength of CBSE and SOSE for improved quality attributes of software solutions. Jiang and Willey proposed a multi-tiered architecture [9] that offers flexible and scalable solutions to the design and integration of large and distributed systems, where the architecture makes use of both services and components as architectural elements, offering flexibility and scalability in large distributed systems

and meanwhile remaining the system performance. Wang and Fung [17] proposed an idea of organizing enterprise functions as services and implementing them as component-based systems in order to offer flexible, extensible and value-added services. Cervantes and Hall [3] addressed introducing service-oriented concepts into component model to provide support for late binding and dynamic component availability in component models. Since CBSE and SOSE keep on developing rapidly, exploring their combination potentials is still one interesting research topic.

5. Summary

In this paper, we have presented a comparison framework for component-based and service-oriented software engineering and discussed briefly the research efforts that have been done in combining the strengths of CBSE and SOSE for improved quality attributes.

An explicit clarification of the concepts, principles and characteristics of CBSE and SOSE is the first necessary step before further exploration in efficient utilization and reasonable combination of them in future applications. Discussions on state of the art research with respect to how to combine the two technologies in a complementary way can be helpful for further investigation of the long term advantages in introducing service-oriented architecture into component-based development, and integrating component-based and service-oriented architecture to offer added value in system development.

6. References

[1] M. Aoyama, "New Age of Software Development: How Component-Based Software Engineering Changes the Way of Software Development", Proceedings of 1st workshop on Component Based Software Engineering, 1998.

[2] A. Brown, S. Johnston, and K. Kelly, "Using Service-Oriented Architecture and Component-Based Development to Build Web Service Applications", A Rational Software White Paper, 2002.

[3] H. Cervantes, and R. S. Hall, "Autonomous Adaptation to Dynamic Availability Using a Service-Oriented Component Model", 2004.

[4] I. Crnkovic, and M. Larsson, Building Reliable Component-Based Software Systems, Artech House Publishers, 2002.

[5] I. Crnkovic, S. Larsson, and M. Chaudron, "Component-Based Development Process and Component Lifecycle", Information Technology Interface, 2005.

[6] R. M. Dijkman et al, "The State of the Art in Service-Oriented Computing and Design", 2003.

[7] S. Hashimi, "Service-Oriented Architecture Explained", http://www.ondotnet.com/, 2003.

[8] M. N. Huhns, and M. P. Singh, "Service-Oriented Computing: Key Concepts and Principles", IEEE Internet Computing, Service-Oriented Computing Track, 2005.

[9] M. Jiang, and A. Willy, "Architecting Systems with Components and Services", Information Reuse and Intergration, 2005.

[10] G. Kotonya, J. Hutchinson, and B. Bloin, "A Method for Formulating and Architecting Component and Service-Oriented Systems", http://scse.comp.lancs.ac.uk/pubs/KotonyaHutchinsonBloin_SOSEBook.pdf, visited 2007.

[11] R. van Ommering, F. van der Linden, and J. Kramer. "*The koala component model for consumer electronics software*", In IEEE Computer, pages 78–85. IEEE, March 2000

[12] M. P. Papazoglou, "Service-Oriented Computing: Concepts, Characteristics and Directions", Proceedings of the Fourth International Conference on Web Information Systems Engineering (WISE), 2003.

[13] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, "Service-Oriented Computing Research Roadmap", 2006.

[14] Z. Stojanovic, and A. Dahanayake, Service-Oriented Software System Engineering: Challenges and Practices, Idea Group, U.S, 2004.

[15] C. Szyperski. Component Software – Beyond Object-Oriented Programming, Addison-Wesley, 2002.

[16] W. T. Tsai. "Service-Oriented System Engineering: A New Paradigm", Proceedings of the 2005 IEEE International Workshop on Service-Oriented System Engineering (SOSE), 2005.

[17] G. Wang, and C. K. Fung, "Architecture Paradigms and Their Influences and Impacts on Component-Based Software Systems", Proceedings of the 37th Hawaii International Conference on Systems Sciences, 2004.

[18] M. Winter, C. Zeidler, C. Stich, "*The PECOS Software Process*", Workshop on Components-based Software Development Processes, ICSR 7 2002

[19] H. Zhu, "Building Reusable Components with Service-Oriented Architectures", Information Reuse and Integration, 2005.

[20] Component-Based Design and Integration Platforms, http://www.artist-embedded.org/, 2002.

[21] Arcticus Systems, Rubus component model, http://www.arcticus-systems.com

[22] OMG. CORBA Components. Report ORBOS/99-02-01.

[23] W3C. World-Wide-Web Consortium: XML, SOAP, WSDL, http://www.w3c.org/

[24] W3C World Wide Web Consortium, Web Services Choreography Working Group, http://www.w3.org