USAGE OF DESIGN BY CONTRACT
From Component-based Engineering to SOA Design

Diana Berberova and Boyan Bontchev
Dep. of Software Engineering, Sofia University "St. Kl. Ohridski", Sofia, Bulgaria

Keywords: Design by Contract, Service Contract, SOA Design.

Abstract: Modern SOA research is focused more and more on fundamental service design issues such as means for creation of formal, standardised service contracts. A possible way for achieving standardised service contracts goes through application of the design by contract approach. Such an approach promises offering a lot of benefits especially when applied for description and management of quality of services. The paper tries to reveal the potential advantages of design by contract when applied for SOA design. It discusses similarities and differences between component based software engineering and SOA, as far as design by contract has been successfully used for component design. Also, it shows the importance of service contract and traces usage of design by contracts for Web service design and how it could be applied for SOA.

1 INTRODUCTION

Design by contract (DbC) is a methodology based on the principle that the interfaces between modules of a software system, especially the critical ones, should be governed by precise specifications (Meyer, 1991). The main goal of Design by Contract (DbC) is to improve correctness and robustness of software systems. The contracts cover mutual obligations called pre-conditions, benefits called post-conditions, and consistency constraints called invariants. DbC was first introduced by Bertrand Meyer in Eiffel programming language. This methodology is very powerful and has already been approved as a technique for building high-quality, reliable solutions in object-oriented architectures. The concept solves an age-old problem of having accessible, up-to-date and readable documentation of the code without requiring additional resources. In addition, exception handling is guided by precise definition of "normal" and "abnormal" cases (Meyer, 1999). The faults occur close to their cause and all this helps to find the problems earlier and faster.

In the last years, software technologies have changed tremendously and the software products have become more complex and more critical than ever. New paradigms like Component Based Software Engineering (CBSE) and Service Oriented Architecture (SOA) have come in the place of object-oriented programming. With the evolution of software architectures, DbC concept has also evolved. It was successfully used in CBSE (Owe, Schneider and Steffen, 2007) and, currently, it is applied in Web Service development (Warmer and Kleppe, 2007). It seems obvious that next step is to apply the DbC concepts in SOA but still there is no solution that provides support for DbC on a conceptual level.

As fas as DbC has been used in CBSE, which relates in many aspects to SOA, some good practices regarding DbC usage can be transffered from CBSE to SOA. Thus, article describes principles of CBSE and SOA and relation between these two methodologies. It continues with the nature of service contract and finished with explaination of DbC usage for Web services and its potential benefits when applied for service contracts.

2 RELATIONS BETWEEN SOA AND CBSE

2.1 Principles of CBSE

The CBSE introduces a new software development paradigm in which systems are no longer implemented from scratch, but glued together from existing components. Component Based Software Engineering is concerned with the assembly of pre-

Berberova D. and Bontchev B. (2009).
USAGE OF DESIGN BY CONTRACT - From Component-based Engineering to SOA Design.
In *Proceedings of the 4th International Conference on Software and Data Technologies*, pages 299-304
DOI: 10.5220/0002244402990304

existing software components into larger pieces of software.

CBSE emerged from the failure of object-oriented development to support effective reuse (Sommerville, 2004). Single object classes in OOP are too detailed and specific. Components are more abstract entities than classes. Each component is a single standalone service provider. Component based software development goal is to reduce development costs by promoting rapid development of software systems that are agile, flexible and easily maintainable.

Component Based Software Development encompasses two processes:

- developing reusable components.
- assembling software systems from software components

Some of the main principles of Component Based Software Engineering are as follows (Breivold and Larsson, 2007):

- Multiple reuse
- Composability (with other components)
- Encapsulation and hidden component implementation – the component is accessible only through its interfaces
- Exactly specified communication interfaces
- Independent deployment and versioning
- Component independence – the components do not interfere with each other

2.2 Principles of SOA

2.2.1 SOA Definitions

There are different definitions of Service Oriented Architecture. One of the most widespread understandings of that SOA provides a framework for design and implementation of rapid, agile, high quality and low-cost systems. Another definition is that SOA unifies business processes by structuring large applications as a collection of services. The goal of SOA is to provide a way of software development in which the developers do not need to provide redundantly the same functionality over and over again.

The interfaces of software applications that provide common logic should have the same look and feel and the same level and type of input data validation.

SOAs build applications using software services. Services are independent components of functionality. A service can be a simple business capability, a more complex business transaction or a system service. Each service no matter if smaller or bigger does not call implicitly other services. Services provide a new way of software reuse on a larger scale. The goal of SOA is to build applications of fairly large pieces of functionality. This goal leads to some amount of processing overhead so a thorough performance consideration should be done.

There are three types of roles in SOA environment:

- Service provider - owns and provides the business information content of services, and in addition a specification for accessing these services.
- Service consumer - incorporates services into applications and designs the flow of the services
- Service registry – used by Service provider to register and make accessible the service and by Service consumer to find a service and access it.

In a SOA environment, the service consumers can access independent services without knowledge of their underlying platform implementation.

2.2.2 Service Interface and Service Contracts

The service in SOA is a stand-alone unit of functionality available only via a formally defined interface. The interface defines the required parameters and the result - the nature of the service, not the technology used to implement it. The system must manage the invocation of the service and management includes many aspects (Erl, 2007):

- Security - authorizaton of requests, encryption and decryption of data, data validation
- Deployment - allows the service to be moved around to maximize performance and provide maximum availability
- Logging - provides auditing and metering capabilities
- Dynamic rerouting - provides fail-over or load-balancing capabilities
- Maintenance - manages efficiently the new versions of the service

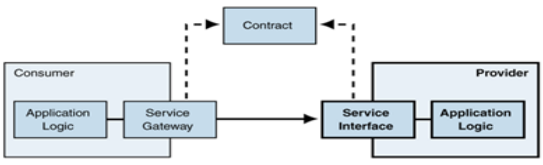

Figure 1: Service interface.

The Service interface describes the behaviour of the service and the messages required to interact with that service. The service interface should describe:

- The operations that a service provides
- The format for information being passed to and from operations
- The message exchange patterns that the service implements (request/reply, one-way, and duplex)

2.2.3 Principles

The guiding principles of SOA concern the rules for development, maintenance and usage of services. These common principles include issues as follows:

- Reuse, granularity, modularity, composability, componentization, portability, and interoperability (Erl, 2007)
- Standards compliance (both common and industry-specific)
- Services identification, categorization, provisioning and delivery, monitoring and tracking

There are also specific architectural principles that concern the system's design and behaviour (Erl, 2007):

- Service statelessness – in order to promote reusability and scalability, state data should be carried by exchanged messages but not retained at the service
- Service encapsulation – logic is encapsulated by a service so that it becomes an enterprise resource capable of functioning beyond the boundary for which it is initially delivered
- Service loose coupling - services relationship should have minimum dependencies and only requires that the services maintain an awareness of each other
- Service contract - services adhere to a communications agreement
- Service abstraction - services hide all logic from the outside world that is not included in the service contract
- Service reusability – logic is divided into services with the intention to be reused
- Service composability - collections of services can be assembled to form composite services
- Service autonomy – services have control over the logic they encapsulate
- Service optimization – all else equal, high-quality services are generally considered preferable to low-quality ones
- Service discoverability – services are designed to be most descriptive so that they can be found and accessed via available discovery mechanisms
- Services independence - service operates as "black box". Service consumers do not know or care how the services perform

2.3 Comparison Study

There is no clear division between Service Oriented Architecture and Component Based Architecture (Petritsch, 2006). SOA can be considered as enhancement of components methodology. Each service is a single component and can be linked to gain new business logic, new services or a new component.

2.3.1 Differences

The big difference between SOA is CBA is the connection between units and the possibilities of offering single services for third parties. Other important differences are:

- The services provide better granularity of the functionality, than the components
- The services provide better abstraction and easier usage
- The services are more loosely coupled than components and hide completely their implementation
- In SOA the division between server and provider is bigger and more clearly specified
- In SOA the communication is message driven and in CBA is communication is object oriented
- The services provide a more dynamic linking of resources, than components

2.3.2 Similarities

Component based architectures and Service-oriented architectures seem to have the same goals. They provide a foundation for loosely coupled and highly interoperable software architecture. They enable efficient, error-free, highly reusable software development. The similarities are:

- Main goal is reuse of software functionality
- Both services and components provide a way to communicate units developed on different operational systems, programming languages and hardware platforms.
- Both components and services offer predefined services

- Both services and components are a higher abstraction of the objects
- Both techniques lead to decreased performance due to the processor and system communication
- Both architectures use the idea of the Facade software design pattern

An exact differentiation between service oriented architecture and component based architecture is not easy. These techniques have the same goals but provide a different way of handling the same issues but in different situation. In this SOA can be seen as a normal extension of CBSE.

3 SERVICE CONTRACTS

3.1 The Service Contract Concept

A service contract represents the terms and conditions by which a service is provided and consumed (Berre, 2008). The service contract is the specification of collaboration between the provider and consumer – it specifies the roles each party plays, the interfaces they offer and the behavior of enacting the service. It is an exchange agreement between specific bound participants, also called *signed contract*. The service contract represents service's collective metadata is a single specification of a service without regard for implementation or dependencies and is defined by the service provider (fig. 3). This representation is required to be in formal, standardized form (Erl, 2007). Thus, the service contract consists of several documents describing technical contract (i.e. technical content such as WSDL, XSD and WS policies – consumed run time) and non-technical information such as Service Level Agreement (SLA). It plays cornerstone role in SOA, as far as it supports other principles in a consistent way (table 1). For example, a standardized contract improve consistency and coupling between services. It should be well balanced, as far as very detailed contracts restrict service abstraction. It is required for a consistent composability but its constraints restrict the reuse.

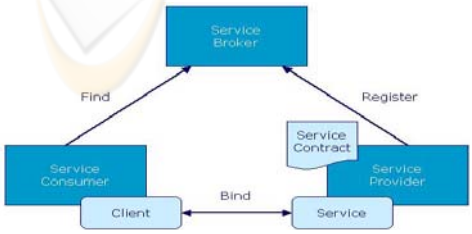

Figure 2: The place of service contract.

Table 1: Service contract's support of other SOA principles.

Service contract		
	facilitates	loose coupling and independence
	provides metadata for	service abstraction
	is used for searching, i.e. for	discoverability, interoperability
	is required for	composability
	determines	granularity and categorization
	controls	delivery, monitoring and tracking
	restricts	reuse

A service contract needs to have the following components:

- Header – containing name of the service, version of this service contract, owner and persons in charge of the service (Responsible, Accountable, Consulted, Informed)
- Type - the type of the service helps for distinguishing the layer in which it resides. Different implementations will have different service types such as Presentation, Process, Business, Data and Integration.
- Functional requirements – service operations and their invocations
- Non-functional requirements – information about security constraints, Quality of Service, SLA, semantics and process.

Services express their purpose and capabilities via a service contract. The principle for Standardized Service Contract is probably the most fundamental one in service orientation design. It specifies that specific considerations should be taken into account when the service's public technical interface is designed.

The goal of this principle goal is to define the specific aspects of contract design in way that service contracts are optimized and in standardized, granular and in the same way usable, consistent granular and governable.

3.2 SOA Patterns Related to Service Contract

The software patterns provide description of a solution for a problem that appears often in software design, architecture and development. There are two SOA design patterns introduced (Erl, 2008) that relate directly to the service contract creation and management: *Contract Denormalization* and *Contract centralization* patterns.

3.2.1 Contract Centralization Pattern

Contract centralization pattern resolves the issue of direct coupling between consumer and implementation. The service consumer programs can access the functionality provided by a service using different entry points. This leads to different forms of implementation dependencies that can influence on the service in a way that the service is not able to change and evolve.

The solution provided by the Contract centralization pattern is to limit the access to service logic and in this way force the customers to avoid implementation coupling and references. This pattern of course can lead to performance overhead and also requires additional effort for standardization. To implement it properly the Service Abstraction principle should be applied when service is designed and created.

3.2.2 Contract Denormalization Pattern

Contract denormalization pattern provides solution for the service contract to facilitate consumer programs with differing data exchange requirements. A problem that often rises in SOA development is that services which have strictly normalized contracts can impose unnecessary functional and performace demands on some service consumer programs.

The solution that is provided is to denormalize the service contracts to a measured extent. By this contract denormalization it will be possible that provide multiple core functions of the service in different ways and multiple capabilities. Each type of capability will be created for a different type of service consumer programs. This pattern should be used extremely cautiously because if overused, it can increase enormously the service contract size and make it difficult to interpret and use effectively.

4 DESIGN BY CONTRACT FOR SERVICES

4.1 Usage of Design by Contract for Web Services

Design by Contract concept is similar to the notion of establishing a legal contract. The contract describes what a service component expects of its clients and what the service clients can expect. The service contract defines the responsibilities of both parties and how these are accomplished.

The notions of contracts for Web services allow the usage of Design by Contract both on provider and requestor's side.

To inform the service consumer about its rights and obligations, the service contracts have to be transferred from the service provider to the service consumer. To achieve this, the representation of contracts during the development process of a Web Service can appear on three different levels (Heckela and Lomanna, 2005) - implementation level, XML level and Model level.

There are two approaches that can be used for developing Web services contracts - code-first and contract-first approaches. The schema based contract-first design is the better approach as it defines the service messages in XML using SOAP standard and makes the services more interoperable. It is easier to define the contract first and then generate the code specific for the concrete platform and programming language. The biggest challenge in this approach is the lack of modelling concepts and tools support. Generally there are five steps in the contract-first design approach:

- Modelling data and defining data structures which should be exchanged in messages
- Modelling messages that should be exchanged by using XML Schema
- Modelling interface and operations provided to the Web service consumer.
- Generate code skeletons and defines message and interface contracts code
- Iterate contract design and code generation.

The Web Service Contract First (WSCF) tool is a Schema-Based Contract-First Web Services code generator and WSDL wizard. It supports designing messages, interfaces and data for contract-first style web services, acting as a replacement for XSD and WSDL. It removes all exhaustive details of WSDL and does not allow making errors and wrong assumptions when trying to use and apply the original WSDL specification. WSCF provides means for defining the schema for service types, messages and WSDL definitions before creating implementation. There are two steps in WSCF – first design contract's data, messages and interface and then generate code from the contract.

4.2 SOA DbC Concept

For achieving standardised service contracts, DbC i.e. "contract-first" design approach should be taken. It should be supported by tools importing customized service contracts. One of the great challenges here is Quality of Service (QoS)

management. The service oriented enterprise systems should be dynamic, flexible, secure and high quality. To achieve all these characteristics QoS management must be integrated into service-oriented enterprise architectures. It must support the most common and valuable QoS characteristics and provide comprehensive services.

SLA is used to establish agreements on the quality of a service between a service provider and a service consumer. SLA sets the expectations between the consumer and provider and helps defining the relationship between the two parties. Properly defined SLAs cover the following aspects (Bianco, Lewis and Merson, 2008):

- The provider obligations
- How delivery of the service at the specified level of quality will be realized
- Which metrics will be collected, and how
- Actions and penalties to be taken when the service is not delivered at specified QoS
- How and whether the SLA will evolve as technology changes.

Thus, the service level agreement defines mutual understandings and expectations of a service between the service provider and service consumers. The service guarantees are about what transactions need to be executed and how well they are executed.

The parameters and metrics defined in an SLA define the Quality of Service of the service. QoS parameters may include response time, availability, throughput, latency, etc. The QoS defines contracts, and obligations between service provider and consumer. To perform this task the service level agreements include negotiation, agreement, quantifying service levels, and clarification of responsibilities (Blackwell and Dixon, 2005). The Quality of Service provisioning and management is of great importance for development of Service Oriented Architecture solutions. It is also one of the greatest challenges and there is still no solution for continuous handling of QoS attributes. There are works in the area of definition and enforcement of QoS service management, but there is no systematic way to do this. The existing gap with QoS may be filled by several efforts:

1) Creation of a unified QoS specification supporting the most important QoS characteristics and providing a way to describe the requirements, contracts and policies based on them;

2) A language and compliant tools that supports SLA and QoS attribute management;

3) Intensive usage of contract-first approach which will facilitate QoS management and will bring better consistency, maintenance and reuse in SOA.

5 CONCLUSIONS

Design by Contract methodology has already been proved as a technique for building high-quality and reliable solutions in Object Oriented and Component based architectures. With the evolution of software architectures DbC concepts have also evolved and currently they are successfully applied in Web Service development. The next step is to apply the Design by Contract concepts in service oriented development but still there is no solution that provides support for DbC on a conceptual level.

The Design by Contract can extend the service contracts with additional logic about the processes and quality attributes which cannot be expressed with simple schema descriptions. There are issues of modeling, implementing and assuring the QoS characteristics in large scale infrastructures and domain services that should be resolved. The goal set for our future investigations and practical work is integration of QoS management in SOA architecture using Design by Contract concept.

REFERENCES

Sommerville I., 2004. *Software Engineering*, Addison Wesley. 7th edition, ISBN-13: 978-0321210265.

Petritsch H., 2006. *Service-Oriented Architecture (SOA) vs. Component Based Architecture*, VTU int. report.

Berre A. J., 2008. *Services Oriented Architecture Profile SOA-Pro: A UML Profile and Metamodel for Services*, SINTEF internal report.

Erl T., 2007. *SOA: Principles of Service Design*, Prentice Hall, ISBN-13: 978-0132344821.

Erl T., 2008. *SOA Design patterns*, Prentice Hall, ISBN-13: 978-0136135166.

Heckela R., M. Lohmanna, 2005. *Towards Contract-based Testing of Web Services*, Electronic Notes in Theoretical Comp. Sci., Vol. 116, 2005, pp.145-156.

Blackwell M., J. Dixon, 2005. *Service level agreements: a framework for the quality management and improvement of central support services*, Monash.

Breivold H.P., M Larsson, 2007. *Component-Based and Service-Oriented Software Engineering: Key Concepts and Principles*, Proc. of 33rd EUROMICRO, pp.13-20.

Meyer B. 1999, *DESIGN BY CONTRACT and Genericity for Java*, http://archive.eiffel.com/doc/talks/sigs/

Meyer B. 1991, *Design by Contract, in Advances in Object-Oriented Software Engineering.* Prentice Hall.

Owe O., G. Schneider and M. Steffen 2007., *Components, Objects, and Contracts*, ACM SIGSOFT Symposium on the Foundations of Software.

Warmer J., A. Kleppe, 2007. *The Object Constraint Language: Precise Modeling With UML.*

Bianco P., G. A. Lewis, P. Merson 2008. *Service Level Agreements in Service-Oriented Architecture Environments*, Techn. Note, CMU/SEI-2008-TN-021.