Agile Modeling and Design of Service-Oriented Component Architecture

Zoran Stojanovic, Ajantha Dahanayake, Henk Sol

Systems Engineering Group, Faculty of Technology, Policy and Management,
Delft University of Technology,
Jaffalaan 5, 2628 BX Delft, The Netherlands
`{Z.Stojanovic, A.Dahanayake, H.G.Sol}@tbm.tudelft.nl`

Abstract. Component-Based Development (CBD) and Web Services (WS) have been proposed as ways of building high quality and flexible enterprise-scale e-business solutions that fulfill business goals within a short time-to-market. However, current achievements in these areas at the level of modeling and design are much behind the technology ones. This paper presents how component-based modeling and design principles can be used as a basis for modeling a Service-Oriented Architecture (SOA). Proposed design approach is basically model-driven, but incorporates several agile development principles and practices that provide its flexibility and agility in today's ever-changing business and IT environments.

1 Introduction

During the last years first Component-Based Development (CBD) [1] and then Web Services (WS) [4] have been introduced as paradigms for building complex Web-based systems and providing effective inter- and intra-enterprise application integration. Besides technology developments, there is a need to architect component-based and service-oriented enterprise-scale software systems. Service-Oriented Architecture (SOA) is an approach to distributed computing that considers software resources as services available on the network that in collaboration provide comprehensive and flexible system solutions. CBD and WS technology platforms are naturally the ways of implementing SOA. However, developers and system architects cannot just start using technology such as EJB or .NET or standards such as XML and SOAP in realizing the SOA. Effective methods for modeling and design of such a complex architectural model are required. Among the other benefits, SOA design should provide a necessary support in deciding:

- what component of the system can be exposed as a service, that can be potentially used in intra- or inter-organization settings, offering a business value to the consumer, and at the same time being as much as possible decoupled from the rest of the system.

- what part of the system logical architecture can be realized by invoking a particular service over the Web, and how that part should interface with the existing organization's system solution.

The SOA modeling and design approach should provide a way of capturing given business requirements in the platform-independent system architecture that can be further mapped into the particular implementation solution, providing effective bi-directional traceability between business concepts and implementation assets. This is the main idea behind the current Object Management Group's (OMG) Model Driven Architecture (MDA) [5]. On the other hand, due to ever-changing business, principles and practices of another development paradigm called Agile Development (AD) must be considered as well [3]. While both AD and MDA provide solutions for building flexible solutions under the high change rates and within short time-to-market, their targets and proposed mechanisms are quite dissimilar. Therefore the balance between the two must be made in order to use the benefits of both paradigms.

The aim of the paper is to propose a service-oriented component modeling and design approach organized around the concepts of services and components in the Service-Oriented Architecture. The approach provides a paradigm shift from components as objects to components as service managers that makes component concepts capable for modeling the architecture of collaborating and coordinating loose-coupled business-valued services. The approach is flexible and agile, providing the way of balancing business and IT concerns, and adopting changes from both sides.

2 Related Work

SOA is an evolutionary, rather than revolutionary concept. A basis of SOA is the concept of a service as a functional representation of a real-world business activity meaningful to the end user and encapsulated in a software solution. Using the analogy between the concept of service and a business process, SOA provides that loosely coupled services are orchestrated into business processes that support organization's business goals. Components and services modeled in implementation-independent way represent an abstraction layer between business and technology. Business goals, rules, concepts and processes are captured by components and services at the specification level that are further mapped to technology artifacts providing effective bi-directional traceability between business and technology. The representation of the building blocks of SOA in a conceptual way provides the level of communication and understanding that is above the level of XML-based languages such as Web Services Description Language (WSDL) [8]. This is particularly important for providing common understanding and effective communication among the project stakeholders.

The natural starting points for SOA modeling and design are component-based and interface-based concepts and techniques, as well as the standard UML as a modeling notation. The current version of the UML (version 1.5) still treats components mainly as implementation units, rather than the main building blocks of the logical system architecture (although there are some improvements in that direction from the version

1.3) [6]. An improved support for components has been promised for the next major revision of the UML (version 2.0) scheduled for this year.

On the other hand, classical CBD methods do not provide thorough support for business-level concepts and services within the SOA [1]. Their focus is mainly on finer-grained components that closely map the underlying entities such as Customer, Order, and Product, rather than on larger-grained, business value added services and components as required by SOA. By treating components as binary-code packaging artifacts during implementation and deployment and as larger-grained business objects during analysis and design, these methods are not well equipped for modeling loosely coupled coarse-grained service-based components that offer business meaningful services organized in the SOA. Moreover, by defining a number of modeling artifacts as well as a complex and prescriptive way of using them proposed methods are often heavyweight and not flexible and adaptable enough to fit into agile business environments of today. A SOA modeling approach must be business service-driven rather than data-driven with strong requirements for modeling service interaction, coordination and dependencies at different levels of granularity. The collaboration and coordination of service components become as important as components themselves.

Therefore a SOA modeling and design approach should be naturally based on standard practices of component-based and object-oriented (OO) paradigms integrated with business process and workflow design concept and techniques. Business and system modeling and design are, more than ever before, integrated around the same set of service concepts and solutions.

3 Service-Based Component Concepts

Components were first introduced at the level of implementation and deployment through the component implementation models such as CORBA Components, Sun's Enterprise Java Beans, and Microsoft COM+/.NET. They have been defined as packages of binary and/or source code that can be deployed over the network nodes. Just recently components have become important analysis and design artifacts in creating logical system architecture.

On the other hand, Web services are self-contained self-describing, modular units providing location independent business or technical services that can be published, located and invoked across the Web. They are natural extension of component thinking. From a technical perspective the web service is essentially an extended and enhanced component interface constructs. Web services, as components, represent black-box functionality that can be reused without worrying about how the service is implemented.

While the component technology has been rather proprietary (divided basically into two camps - Microsoft and Java-community), Web services have provided standards and protocols for interoperability of loose-coupled software constructs across the Internet. Although these technology achievements such as XML, SOAP and UDDI are necessary for enabling true interoperability, the way of designing a system has not been changed. The basic design philosophy is still founded around component-based design techniques such as interface-based design, black-box modeling, design patterns, design by contract, dependency modeling etc. Therefore the component design concepts are a solid foundation of an approach for designing service-oriented architecture. While the classical objects in Object-Orientation are at too low level of granularity to be considered as a basis for defining Web services, larger-grained service-based business components represent a perfect mechanism for designing services in a SOA.

For the purpose of modeling the main building blocks of SOA we introduce the concept of *service component*. A service component is a self-contained service-based building block. It delivers services to its environment through the contract-like interface that abstracts its internal realization. Services can differ in granularity (coarse or fine-grained) and nature (provides a transformation, computation or information). Component collaborates with other service components in the single application space or across the Internet to provide a higher-level goal.

The service component meta-model can be divided into two parts. First part defines the basic concepts describing the very nature of a service component. At first place a component can be defined through the three basic aspects:

- Context (environment) inside which the component exists.
- Contract that is defined according to the component role in the context and that the component guaranties to fulfill.
- Content (interior) of the component that represents a realization of the component contract.

A component does not exist in isolation; it fulfils a particular role in a given context and actively communicates with it. A component participates in a composition with other components to form a higher-level component. At the same time every component can be represented as a composition of lower-level components. A component must collaborate and coordinate its activities with other components in a composition to achieve a higher-level goal. Well-defined behavioral dependencies and the coordination in time between components are of a great importance in achieving the goal.

The second part of the component meta-model defines the basic elements of the component contract as the main aspect of a service component. Component contract concepts represent the complete information about the component necessary for its consumer to use it without knowing its interior. This is an enriched and enhanced basic interface construct that now contains all the information about the component (or service) that must be known by its context in order to make use of it. In this way a component interface goes beyond simple operations' signatures to become a real business contract between a component as a service provider and the context as a service consumer. The following are the contractual concepts of a component:

- Component identification
  - Unique name in the naming space or identifier, the goal and purpose of a component (service).
- Component behavior
  - Operations (actions) provided and required,

  - Pre-conditions and post-conditions defined on these operations,
  - Events published and subscribed,
  - Coordination of operations and/or events to provide a higher-level behavior.
- Component information
  - Information types that the component uses or handles (not necessarily stores) mostly as parameters for services and operations the component provides and requires,
  - Invariants and constraints on these information objects.
- Configuration parameters
  - Parameters defined by the component that can adapt its contract to fit into possibly new requirements coming from the context, such as required Quality of Service (QoS), location in space, location in time, consumer profiles, etc.
- Non-functional parameters
  - Parameters that characterize the "quality" of component behavior in the context, such as performance, reliability, fault tolerance, priority, security etc.

The component contract can be fully specified using different mechanisms, from natural language to formal specification language and to XML-based language if we want a machine-readable specification of a component. On the other hand the component contract can be implemented using different implementation tools and techniques to provide the life of the component in the world of bits.

4 SOA Modeling Approach

Complexity of distributed enterprise systems raises the need for using the separation of concerns in specifying system architecture. Therefore, we use as underlying frameworks both OMG's MDA and ISO standard Reference Model of Open Distributed Processing (RM-ODP) [7] for defining the three architectural models that represent logical layers of our service-oriented component architecture:

- Business Architecture Model (BAM) – a model of the system as collaboration of components and services that offer business value.
- Application Architecture Model (AAM) – a model of the system that shows how business components and services are realized by the collaboration of finer-grained components and services.
- Implementation Architecture Model (IAM) – a model of the system that shows how business and application components and services can be realized using a particular implementation platform.

The BAM roughly corresponds to ODP Enterprise Viewpoint, AAM to ODP Computational Viewpoint, and IAM to Technology Viewpoint. Distribution concerns in the ODP described by the Engineering Viewpoint, and information semantics and dynamics in the ODP described by the Information Viewpoint are not treated separately in our application framework then integrated throughout all three architectural models. Thus distribution can be considered as business components distribution (virtual enterprises, legacy assets, web services), application distribution (logical distribution tiers) and implementation distribution (support by the particular middleware). Similar to this, a conceptual information model is defined in the BAM, a specification information model is fully specified in the AAM, and the ways of permanent data storage are considered in the IAM. The Figure 5 shows our architectural modeling framework.

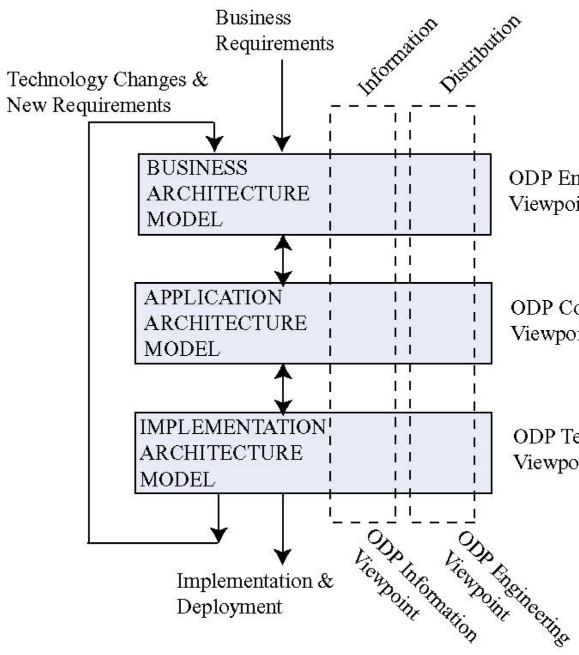

Business Requirements
Technology Changes & New Requirements
Information
Distribution
BUSINESS ARCHITECTURE MODEL
ODP Enterprise Viewpoint
APPLICATION ARCHITECTURE MODEL
ODP Computational Viewpoint
IMPLEMENTATION ARCHITECTURE MODEL
ODP Technology Viewpoint
Implementation & Deployment
ODP Information Viewpoint
ODP Engineering Viewpoint

Fig. 1. Architecture Modeling Framework

The BAM and AAM actually represents two levels of abstraction of a service-oriented MDA's Platform-Independent Model (PIM), while the IAM describes a service-oriented Platform Specific Model (PSM) for a particular technology platform. By focusing on two basic component stereotypes – Business Service Component and Application Service Component, we can define two levels of a Platform Independent Model. The first PIM level defines how business process is supported through contractual collaboration and coordination of service-based business components. The second level "opens" black-box business components and defines how their interior design is realized through collaboration and coordination of finer-grained application components and services. By defining all three models in a consistent manner, the whole system is specified and ready for implementation. The best result is achieved using constant iteration and small increments during design, as suggested by agile development principles.

The main goal of the BAM is to specify the behavior of the system in the context of the business for which it is implemented in terms of collaborating and coordinating chunks of business functionality represented as business service components. BAM starts with the following models: activity model that shows the flow of activities in the system, use case model and domain information object model. Based on use cases that fulfill business user goals (i.e. that correspond to Elementary Business Processes [2]) we define business services that system should provide, as well domain information objects used by these services. For each use case (and a service that supports it) the use cases that precede it, follow it, perform in parallel with it or be synchronized in other way with it should be defined, Figure 2. Furthermore, for each use case its superordinate and subordinate use cases should be defined providing a hierarchy of use cases, i.e. business goals. This can be illustrated using an activity diagram with use cases as action states of the diagram, or a sequence diagram enriched to express the action semantics (sequence, selection, loop, fork/join, etc.) with the use cases on the horizontal axis of the diagram. Domain information types are cross-referenced with the use cases defining, for each use case, what information types are needed for its performance.

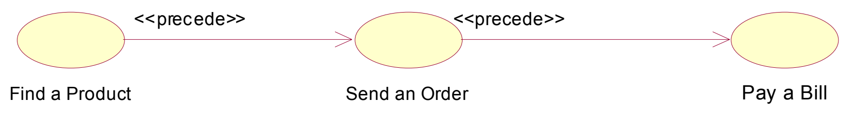
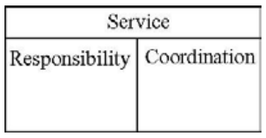


Fig. 2. The example of the relation <<precede>> between use cases

Services that support given use cases can be specified in two ways:

- in an agile-like manner using Service-Responsibility-Coordination (SRC) cards, Figure 3, as a variant of a CRC (Class-Responsibility-Collaborator) cards,
- by using more formal specification mechanisms derived from the use case specification template [2].



Service	
Responsibility	Coordination

Fig. 3. Service-Responsibility-Coordination (SRC) Card

Main elements of the SRC card are:

- Service – its name reflecting its goal, purpose and scope.
- Responsibility – description of its behavior preferably through lower-level services or activity steps it provides together with information objects that should be used by the service as some kind of parameters.
- Coordination – what services (events) precede or trigger this one, what services should follow this one, or what events should be emitted; furthermore what are eventual subordinate services and a superordinate service of this one.

By using the set of different business and technical criteria, such as semantic cohesiveness, shared data objects, market value, reusability potential, existing assets, etc., identified services are allocated to Service Cluster Units, which represent blueprints for business service components of the system. Again, business service components can be specified in a more formal contract-based manner, or, if the nature of the project suggests, in more agile way using Component-Responsibility-Collaborator-Coordination (CRCC) cards as another variant of classical CRC cards. Collaboration and coordination of business service components that form the system can be represented using the component stereotype of the sequence diagram enriched to express control flow mechanisms. Information about that is derived from the relationships among use cases that particular business service components support. The relations among the concepts of business components, services and business goal-oriented use cases are shown in Figure 4.

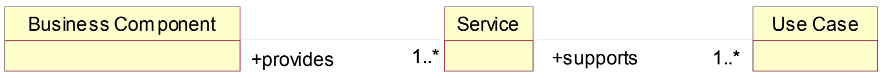


Fig. 4. Conceptual relations between business components, services (operations) and use cases

The goal of the AAM is to define how interior of business service components is realized in terms of collaboration of lower-level application components and services that do not provide a direct business value. There can be different types of application service components:

- Services that communicate with the business service component consumers by transferring their requests to the form understandable to the business logic and back. They should hide potentially different service consumers from business service logic.
- Services that provide some computation or data transformation logic;
- Services that represent contact points for information about business entities used by the given business component.
- Data access and data handler service components that hide variety of data storage formats from the business service logic.
- Service components that support included and extended use cases of the use case(s) realized by the given business service component;
- Coordination manager that coordinates other application service components inside the business component;
- Event manager component that manages the event subscription and notification mechanisms in an event-driven environment;
- Business rule manager component that handles business rules captured by the given business service component and maps these rules to pre-conditions, post-conditions, invariants, coordination conditions and other constraints defined on the component behavior and structure.

The result of the AAM is complete, fully specified, component-oriented platform-independent model that should be further considered for implementation on a particular technology platform. Functionality offered by a Business Component can be exposed as both inter- and intra-enterprise Web service in a SOA. On the other the services offered by an Application Component can be also used as Web services, but mainly internally to the enterprise.

IAM uses the complete business-driven component-based distributed system architecture specified through the previous models, and translates them to platform-specific models according to the chosen target implementation platform. To provide further flexibility of the architecture models we propose a technique called component refactoring which aims at reallocating and rearranging sub-components or sub-services of the component being addressed, while preserving its contractual behavior, analog to code refactoring used in agile development [3]. Application components are normally implemented using implementation components, language objects/classes or other programming constructs. Application Components can be directly or indirectly instantiated (addressable) depending on their granularity. On the other hand, Business Components are implemented as a composition of software constructs that realize their sub-components (in which case they are indirectly instantiated), or can be used as already built third-party software units, such as wrapped legacy assets, Web services or COTS components (in which case they are directly addressable in a general sense).

5 Conclusion

The SOA modeling arises certain requirements on top of the standard OO and CBD modeling methods. Therefore, straightforward applying of existing UML and CBD concepts for the purpose of modeling the SOA, although a good starting point, is not a feasible approach. The UML component concept as a natural basis for SOA modeling is still mainly implementation-related, while popular CBD methods are mainly focused on finer-grained entity-driven components. Due to the business-driven character of SOA, a proper modeling approach should combine component-based and object-oriented (OO) modeling concepts on one side with activity and workflow modeling mechanisms on the other side.

This paper presents a business-driven agile approach for modeling component- and service-oriented architecture. The approach provides a paradigm shift from components as objects to components as service managers. In this way the approach is capable of modeling the system architecture representing a contract-based collaboration and coordination of components and services. Since components and services are identified based on business requirements, goals and rules, then fully specified inside the logical system architecture and implemented using advanced CBD and WS technology, the approach provides bi-directional traceability between business concepts and implementation artifacts. The approach is basically model-driven but incorporates certain agile development concepts, principles and practice (e.g. cards, refactoring, user involvement) making an effective combination between the two in order to achieve the goals of adaptable process and solution, high-quality and on-time development products that closely reflect business goals and needs. The approach makes use of standards OMG MDA and RM-ODP to provide iterative and incremental architectural modeling and design through different architecture abstraction levels providing complete specification of the system ready for implementation in chosen platform.

References

1. D'Souza, D.F. and Wills, A.C.: Objects, Components, and Frameworks with UML: the Catalysis Approach. Addison-Wesley, (1999)
2. Cockburn, A.: Writing Effective Use Cases. Addison-Wesley (2001)
3. Cockburn, A.: Agile Software Development. Addison-Wesley, Boston MA (2002)
4. IBM Web Services, http://www.ibm/com/webservices.
5. OMG Object Management Group, MDA- Model Driven Architecture, information available at http://www.omg.org/mda/
6. OMG Object Management Group, UML- Unified Modeling Langauge, information available at http://www.omg.org/uml/
7. ODP, International Standard Organization (ISO), Reference model of Open Distributed Processing: Overview, Foundation, Architecture and Architecture semantics, ISO/IEC JTC1/SC07, 10746-1 to 4, ITU-T Recommendations X.901 to 904, 1996.
8. W3C. World-Wide-Web Consortium, WSDL (Web Services Description Language). Available: http://www.w3.org/TR/wsdl