STUDY ON USE CASE MODEL FOR SERVICE ORIENTED ARCHITECTURE DEVELOPMENT

N.SASIKALA, Research Scholar, MTWU

Dr.K.RANGARAJAN, HOD,DEPT.OF MCA,BHARATH UNIVERSITY.

Abstract

The recent trends in the computer industry are the one and only thing i.e., web services. Because of the common availability and open technologies web services are relevant to all. Service-oriented architecture (SOA) helps organizations to transform their business processes for high performance by simplifying the underlying information systems. The most challenging aspect of building successful software applications is clearly understanding and specifying the requirements that an application must satisfy. Use case modeling is an increasingly popular approach for identifying and defining requirements for software applications of all types. Use cases describe the behavior of the system as its users interact with it. This approach helps to place the software requirements in the framework of a user doing some useful work with the system. This type of approach helps to map software requirements to the relevant end-user business processes, a very powerful concept. This paper presents how the relationship between use case model and Service oriented architecture.

Index Terms—Service Oriented Architecture, Component Based Architecture, SOA entities, SOA characteristics, Use cases, Relationships

1. Introduction to SOA

Companies are in need to integrate existing systems in order to implement information technology (IT) support for their business processes which cover all present and prospective systems requirements needed to run the business end-to-end. A variety of designs serve this kind of service ranging from rigid point-to-point electronic data interchange (EDI) interactions to web auctions. This will be done by updating older technologies, for example by Internet-enabling EDI-based systems, companies can make their IT systems available to internal or external customers; but the resulting systems have not proven flexible enough to meet business demands, which require a flexible, standardized architecture to better support the connection of various applications and the sharing of data. SOA offers one such prospect. It unifies business processes by structuring large applications as an ad hoc collection of smaller modules called "services".

Service Oriented Architecture (SOA) can be defined as an evolution of the Component Based Architecture, Interface Based Design (Object Oriented) and Distributed Systems. A Component Based Architecture is an architecture where the functionality of the whole is divided into smaller functions, each encapsulated in a component. A Distributed System is an extension of components-based architecture and refers to components that may exist in different physical locations. This paper is organized as follows: Section 2 describes about the SOA entities and its description. It also talks about the guiding principles for the development, maintenance, and usage of the SOA. Section 3 describes the necessity for software employees to adopt SOA and the benefits which they can gain. Section 4 demonstrates about the use cases and its benefits of use cases. Section 5 describes two different examples to talk about the relationships between the use cases and SOA. Conclusions are presented in Section 7.

2. SOA Entities

SOA consists of the following six entities configured together to support the find, bind, and execute paradigm. The “find, bind, and execute” paradigm as shown in Figure 1 allows the consumer of a service to ask a third-party registry for the service that matches its...
criteria. If the registry has such a service, it gives the consumer a contract and an endpoint address for the service.

Figure 1:

Service Provider

Service consumer

Contract

Registry

Service Proxy

The service proxy is a convenience entity for the service consumer. The service proxy can enhance performance by caching remote references and data. When a proxy caches a remote reference, subsequent service calls will not require additional registry calls. By storing service contracts locally, the consumer reduces the number of network hops required to execute the service.

Service Lease

The service lease, which the registry grants the service consumer, specifies the amount of time the contract is valid: only from the time the consumer requests it from the registry to the time specified by the lease (Sun Microsystems, Jini Technology Core Specification, 2001). When the lease runs out, the consumer must request a new lease from the registry. The lease is necessary for services that need to maintain state information about the binding between the consumer and provider. The lease defines the time for which the state may be maintained.

2.1 Principles

The following guiding principles define the ground rules for development, maintenance, and usage of the SOA:

- Reuse, granularity, modularity, composability, componentization, portability, and interoperability
- Standards compliance (both common and industry-specific)
3. Essential Need for the adoption of SOA

In general the Information Technology (IT) workers face many challenges like,

- Limited budgets for the modeling system.
- Constantly changing technologies.
- Evolving technologies for the same business function.
- Business requirements that demand applications and technology silos that need to be integrated with each other.
- Application functionality that must be extended to reach outside an enterprise firewall

By these factors we can come for the conclusion that the system require integration. For example, when a core banking system must be accessed and used by customers through the Internet, a demand is placed on the system to expose some of its functionality to the application that drives the Internet access.

If the integration is always in native format (specific to an application’s programming language or environment), then integration projects will require a lot of custom integration work. In addition, the ability to connect multiple systems together quickly will require highly specialized programming capabilities and knowledge of the nuances of the individual systems. Because each enterprise application may be implemented on a single server that provides the same functionality to many clients, it makes sense to use common protocols to access the functionality. IT staff can lower costs dramatically by using common protocols and standards, and common methodologies such as SOA.

Figure 2 shows the distinct philosophies of each step that business has taken to adapt to requirements integration:

1. Vertical silos of integration – keeping all applications and systems with similar functionality integrated with each other, but not accounting for applications that may wish to use their core functionality in the future.

2. Horizontal integration – integration of some but not all similar functionality across vertical systems; for example using a common purchasing system for raw materials, shipping needs and office supplies.

3. The SOA – an environment of ubiquitous service providers and service consumers interoperating with each other in a secure and consistent manner.

Figure 2: Business views of system integration

Many IT workers are placing requirements for this type of integration on their software vendors. Most of the large and medium-sized software vendors have either announced or incorporated SOA methodology in their software.

While the basic patterns of integration remain the same, the specific technology to implement it does vary, depending largely on the software vendor. While the technical barriers to integration are easy to overcome, there are vastly differing business policies and legal aspects to be enforced at runtime, but no widely accepted standards exist for them yet. Many software vendors continue to pursue different methodologies and individual solutions for these components of the SOA.

4. Use Cases

Use cases are a powerful technique for capturing and communicating functional requirements for software development. A use case describes a set of possible executions of the system. It describes a complete flow through the system until the resulting value is achieved for some actor. I can be described as follows:
• **“Sequence of actions”**: Atomic activities, decisions, and requests. Each action is performed fully or not at all.
• **“Performed by a system”**: The actions performed by the system are the functional requirements.
• **“ Observable result of value”**: Make sure that the use case results in an observable value. Why would anybody use the system if it does not achieve a result of value? If nobody receives value from the use case, then the use case is probably too small. It needs to be combined with other use cases to provide a complete set of steps for achieving some goal for an actor.
• **“To an actor”**: Decide which particular actor receives the value and helps avoid use from declarative statements in two primary ways. They describe functional requirements from the user perspective rather than the system perspective, and they provide a coherent goal focused flow of events rather than a set of discrete declarative statements.

5. Relationship between SOA and Usecase model

In formulating an SOA, we can start with operation. Here the focus is how end users, systems, or applications use services. Use cases in Unified Modeling Language (UML) [5] describe the external behavior of a service as seen or utilized by an actor (user, system, or application). The boundary of the service in a use case is clearly delineated. The interaction of the actor with the service is described without revealing the internal details of the service. Use case is, therefore, a natural tool for describing operational activities in an SOA. Based on the operational concept, the scope of services, and the high-level requirements, one may identify a set of high-level critical use cases. These are the use cases that the architecture must support to meet the minimal requirements. Use cases are not requirements. Nevertheless, they illustrate what functions architecture provides and highlight the requirements. Therefore, use cases are the first step in formulating an SOA (see Figure 3).

In each use case, typically two or more nodes interact with each other by exchanging information. If a node is a service consumer, then it is an actor in the use case for that service. If it is a provider, then it is a component providing that service. Traditionally, a node in the C4ISR framework represents a role, an organization, an operational facility, etc. For an SOA, its scope is expanded to include shared resources and services. Hence a service node interacts with consumer nodes to provide services. Use case and node are, therefore, the primary objects in describing the operational aspects of an SOA, as shown in Figure 3.

Once the operational aspects are identified, the next action is to find the solution that satisfies the operational requirements. In an SOA, each service provides a set of well-defined functions useful to its users, or consumers.

5.1 Operational View

Let us consider an enterprise messaging system, in Command, Control, Communications, Computers, Intelligence, Surveillance, and Reconnaissance (C4ISR), which encompasses e-mail, instant messaging (IM), chat, and presence services. The critical use cases are send and receive emails and instant messages, participate in a chat session, subscribe to and receive presence notifications, etc. They are shown in the use case diagram in Figure 4. In addition, the administrator configures and administers the services.
For each use case, you may describe a sequence of events or activities. These activities may be presented in a hierarchy, as in the standard activity model operational view. Here, however, the use cases provide a natural grouping of those related activities. Additionally, a use case highlights the actors and system/service boundary, allowing you to delineate roles and nodes easily. Hence, include use case as part of OV and consider it an essential product for an SOA.

For an SOA, the use-case diagrams (such as Figure 4) often identify the nodes. These nodes are roles, organizations, shared resources, or service nodes. You can further draw the connections (i.e., the need lines) between the nodes, thereby forming the operational node connectivity description (OV). An example is given in Figure 5.

Using UML techniques to supplement the traditional C4ISR framework, I have elucidated an approach for formulating an SOA. On the operational side, it starts with use cases, which involve the interaction of two or more operational or service nodes. Mission functions are provided through applications, which are implemented by a set of services. The high-level operational concept graphics still applies to an SOA. This, operational views together encompasses the concepts of operation, the use cases from user's viewpoint, the connectivity between operational nodes, and their information exchanges. They therefore characterize the essential operational aspects of an SOA. Furthermore, since operational nodes include shared resources and services, dynamic and collaborative operational activities are properly captured.

6. Conclusions:
By this paper I can also conclude that well-written use case descriptions are a powerful technique for capturing and communicating functional requirements in many software development paradigms. In fact, many software development organizations have adopted use case techniques for their requirements management efforts on projects that are not object oriented or using any other UML constructs. SOA is based on the use of distributed objects and components and is the next evolutionary step in computing environments. An SOA may also implement optional concepts that include a service consumer, a service client, acceptance of the service contract and invoking the service. There are many business drivers affecting the development of a standardized SOA reference.
model. Once this is achieved, SOA will likely be part of the solution for many business and world issues. They are by no means a silver bullet for requirements and UI design, and they certainly have their pitfalls, but overall they can be a powerful tool for most projects. The secret is to keep it simple, and to involve the users right the way along in the identification and design of use cases. Remember that our aim is to eliminate rework due to requirements misunderstandings, and so we should be aiming to reach a point where there are no surprises for the users. Use cases, in conjunction with SOA techniques help to build an explicit shared understanding that everyone can take away with them. users, developers, testers, technical authors, and others.

7. References
https://jdl.jwfc.jfcom.mil/.
The next version is the DoD Architecture Framework Vers. 1.0. 15 Aug. 2003 www.aiitcnet.org/dodfw