Real World SOA: Analysis and Design

by Mohamad Halabi

Course Introduction

Introduction

Hi. This is Mohamad Halabi. Welcome to the Real World SOA: Analysis and Design course. If you want to learn all about SOA beyond the misconceptions and false expectations and if you want to understand when to implement SOA and how to implement it all within the context of real world problems and solutions, then you're at the right place.

A SOA Course! Isn't SOA Dead?

In 2009, an article titled SOA is Dead caused various discussions and many people agreed. In my opinion, whoever agreed did so for one of two reasons, either due to misconception about what SOA is or due to having false expectations from implementing SOA. The purpose of this course is to make sure you do not fall into this trap. It will teach you what SOA truly is, what problems it is intended to solve, when it is unfit for an organization, and how to implement it in a correct manner. By the end of the course, you will be convinced that what I call correct SOA is not dead. In fact, it's very much alive and will continue to be for a considerable time.

What's Special About this Course?

There aren't many high-quality SOA publications, but in my experience, most of them suffer from two main problems. First, and most common, is that the discussion takes a technical path right from the start. While this probably covers well the how aspect, it skips what are essential aspects of any architecture, the why and the what. Why are we pursuing SOA? Why is a sort of a solution for our problems? And what is the scope of SOA? What objectives are we trying to fulfill? Therefore, understanding business takes the front seat in SOA. The other problem is that most SOA publications take a theoretical approach. They typically leave the audience wondering how SOA is implemented in real world. In other words, what are the real-world challenges that you face as a SOA practitioner? This course tackles both these problems. It will make you understand the business behind SOA before at the jumps of the technology and it will do so presenting a real-life business case. The problems and solutions you will see in this course are all real-world problems and solutions. By the end of the course, you will be equipped with all the tools to implement the correct SOA.

Course Structure

As you'll get to see in detail later, there are a lot of information to be covered to properly understand SOA. Therefore, to make it easier for audience to follow through, the course is actually divided into two parts. The first course, this one, is about SOA analysis and overall architecture. Here I divide content into three logical areas. First, I cover the why aspect by setting the context. I will explain the real-world case study, what SOA is, why SOA was chosen as a suitable solution for our case study, and what are the challenges that will face us. In the next area, we cover the what aspect. As we start examining SOA's high-level architecture known as reference architecture, here you will learn what SOA is made of, what are its building blocks, what type of services will we create, and the overall structure and dynamics of a SOA. The next area dives into the how aspect as we study the process of SOA analysis, how we dissect the business as a pre-requisite for service identification, how we derive the services, how we classify the services, and how to apply a common information model. The next course, part two, is about the logical design and implementation. Here, we cover the remaining part of the how aspect including SOA design principles, quality attributes, service creation, and many more design and implementation topics.

Spot the Case Study

As I promised you, everything you will learn in this course will be presented in the context of a real-world case study. Within each module, any time you see a clip title starting with case study, this means you are about to apply the concepts that you have learned in our case study. Some concepts, however, are wide in scope; therefore, single eclips do not give them justice. In this case, you will find some module titles starting with case study. This means that the entire module is devoted to teaching you the application of SOA concepts in our case study.

The Case Study

Introduction

Welcome to module, The Case Study. You are watching Real-World SOA: Analysis and Design. As I said in the previous module, the content is divided into three areas. In this module, we start with the first area and that is understanding the context, and the first step is understanding the case study. This is very important because unless you understand the problems that we are trying to solve, you won't be able to understand the value of SOA.

Description

The case study is an adaptation of a real-world scenario of a government agency. This agency funds and manages the scholarship of post-graduate students who are studying overseas to earn their Masters or Ph. Ds. After completing scholarships, the agency also offers post-graduation services, which help the graduates find internships and jobs. The type of students, their selections, the allocated funds, and other complex business concerns are not relevant to this discussion. The high-level scenario goes as follows. First, there are three stages, the application stage where applicants submit their requests and get either approved or rejected, the scholarship stage, this is the core of their business where approved applicants get their entire scholarship managed in terms of finance and administration, and the graduate stage where students that have finished their scholarships get services that help them get internships and jobs. In a high level, an applicant applies to a scholarship. If approved, the student then travels to its nation country and starts the study lifecycle. During this time, the government agency manages everything related to this scholarship, such as funding, changing the major, grade striking, financial requests, pausing and resuming the scholarship, and so on. After the conclusion of the study phase, graduates go back to their home country and the government agency then manages the post-graduation lifecycle helping graduates lending, internships, and jobs offered by partner companies.

Problems

Of course, understanding the problem is the first step in identifying a solution. In this section, let's examine the problems that the organization is facing and throughout the course we're going to see how SOA help solving these problems. So when the government agency first started, the demand was to get the public services into production as soon as possible at all costs. So the team started by creating a couple of systems that provide certain services such as applicant services and student profile services. Over time, the scope started to get bigger and it was requested to present scholarship-related services, such as major changing and stopping a scholarship. Then later, financial services were requested as the agency decided to streamline the process by which they compensate students. At this stage,

the agency decided that as the range of business services being presented is increasing, then it must partition the business domain into a set of subdomains to ease management and planning. So a scholarship business domain was established and it included study, finance, and application sub-business domains. Each domain then will become the owner of the set of information systems that serve this domain. But the business model of the agency kept growing and now a major decision was taken to provide most scholarship services, first helping graduates land internships, then expanding the scope to help them land jobs. So again, they went on creating new business domains or units each responsible for their own information systems. Now during this expansion came a critical moment. Many of the public services exposed to customers now require the creation of cross-unit or cross-domain processes. It was no longer enough for each unit to serve customers separately, so units must now collaborate to present public services. This business processes use the functions and data presented by the information systems in order to perform their services and here lies the problem. As explained, due to time and resource constraints, systems initially were built in silos without a single holistic view of the entire architecture. This means that each system was created without taking into consideration the overall growth and architecture. As a result, the agency now has a set of siloed systems with redundant functionalities, redundant data across data stores, fragmented data across different stores, and systems that each has its own separate data model. So what the organization has is an interesting problem. Business processes that were from the start designed to cross unit boundaries and require different units to collaborate to achieve required outcomes, yet the supporting systems, the ones that provide functions and data to processes, are built in siloed manner with no regard to cross-unit collaboration. Because of this situation, the agency is now facing critical issues. The agency is not being able to keep up with the rate of changing requirements. Business processes relying on the current state information systems are not flexible in the responsible changes and the agency is facing a real problem with its lack of agility. This makes the agency slow to react to requirements, slow to launch new services, and slow to take advantage of new technical opportunities that could better enable the business. All this leads to a lack of customer satisfaction. IT costs are high since the agency spends valuable time and money working around the limitations caused by the IT state. Due to data and functionality redundancy, data aggregation, and different data models, both development and operation costs are high to keep the business running. And because functionality and data are redundant across units, customers are often required to enter the same information multiple times, such as provide information, for example, and over time, this information becomes inconsistent, which in turn, leads to unsatisfied customers and high IT costs.

Goals

So given the critical issues the agency is facing, it has a set of business goals that must be achieved. First, the agency wants to be agile in the response to new demands. It wants to improve the flexibility of business processes so that changes of the new requirements are applied in a timely manner and the time to market is decreased. The agency also wants to decrease the IT development and operation costs. It wants less development time and effort to be put

to respond to changes and less time and effort to operate existing systems and data in the response to these changes. The agency also wants to create a customer-centric enterprise. Customers must not feel as if they are dealing with different independent entities. The customer must have a single unified experience throughout the interaction with the agency regardless of which unit is offering the service. Now in addition to these business goals that are set to solve the current issues, one business goal is driven by an emerging business need. The agency wants to allow companies, which offer internships and jobs to be able themselves to post and edit information about internship and the job vacancies. In addition, companies should be able themselves to ask for certain set of skills and decide to offer graduates that fit their needs an opportunity for internship and possibly, later a job. As of now, these are done mostly by online communication and both the agency and the growing number of companies are looking for a more efficient and effective solution. This goal is essentially about extending the agency ecosystem to include companies. Just keep this goal in mind. Later it will become clear what's so special about it.

What Is SOA?

Introduction

Welcome to module, What is SOA. You are watching Real-World SOA: Analysis and Design. In the previous module, you understood the case study and the problems that are driving the organization to look for SOA as a possible solution. Now that the problem is clear, we first need to understand what exactly is SOA and to clear misconceptions and false expectations. Note that we're still setting the context here, so we'll not be talking about architecture just yet. We, first, want to understand what is SOA and why is it a sort of a solution to our problems.

Service-oriented Computing, Service-orientation, and SOA

The term Service Oriented Architecture or SOA has been used so broadly that it is sometimes mixed with the related terms, such as service oriented computing and service orientation. Knowing the difference between these terms is not necessarily an essential thing that you should do, but establishing a consistent vocabulary is important for clear communication. Service oriented computing presents a distributed computing platform that helps organizations achieve a set of strategic goals by helping organizations to be more flexible and agile in response to ongoing business demands. Service orientation, the first to a set of design principles that when applied to service design help achieve these strategic goals associated with service-oriented computing and service oriented architecture is an architectural style for building IT architectures based on services designed following the service orientation design principles. SOA decides the structure of these services, their interactions, and their interrelations with existing assets.

From now on, I will use the term SOA to talk about the architecture style. This will be the focus of our discussion. If I want to refer to any of the other terms, I will do so explicitly.

What Is a Service in SOA?

SOA is an architecture style, is technology independent, meaning that SOA aims to fulfill the strategic goals of service-oriented computing independent of any particular technology. This makes SOA capable of leveraging a new emerging technologies, as long as it continues to fulfill the strategic goals. So on principle, the concept of a service is not tied to any particular technology. It could be any piece of physically independent solution logic designed following the design principles of service orientation. Therefore, SOA services could be built for example as. NET components, Java components, or web services. So conceptual, a SOA service can be defined as a technology independent unit of solution logic with an independent functional context where functionality is exposed through a set of capabilities via an interface. The service interface dictates all aspects of communication between the service and its consumers, including the format of Exchange messages, technology requirements, quality of service, and policies. Now that being said, without a doubt, web services have been the dominant implementation choice of services for multiple reasons. First, the interoperability of web services eases up communication between different platforms. In addition, it allows us to build reusable services because they can be accessed by various communication frameworks, which increases the potential consumer base of the service. The nature of how web services expose contracts in a decoupled manner from the logic enhances loose coupling and abstraction. As we will see later, characteristics such as interoperability, reuse, loose coupling, and abstraction are critical to the successful server. Therefore, web services are definitely the dominant choice of SOA implementation. One important note I want to make here is that when I talk about web services, I'm not yet talking about what kind of services. Services can be SOAP-based or actually be based Web APIs. I will come back to this point in the next course, but I just wanted to give you an early hint because people tend to directly think SOAP whenever web service is mentioned. In this course, I talk about web services in general since we are not yet going into service design. In the next course, service design will be explained in detail.

Strategic Goals of Service Oriented Computing

As I said before, SOA is an architecture style aimed to fulfill the strategic goals of service-oriented computing. So what are these goals? The first and most important goal is to align IT with business goals and strategy. In other words, the IT solutions and services that you will create need to be traced back to business needs that are defined by the business goals and the business strategy set to reach these goals. As we will see later in detail, the first step in implementing SOA is to establish or articulate the business architecture. Business architecture is the first and most critical step in ensuring that IT architecture are aligned with business needs. Don't worry if this is not clear for now. I

will come back to this topic in detail later. But the important thing to understand for now is that SOA as an architecture gets its value by being able to align IT resources with business needs. This goal is the driving force behind SOA and all other goals exist to support this goal. Agility is a critical strategic goal that organizations strive to achieve. In current markets, as business needs evolve quickly environmental factors cause business disruptions and technology-based opportunities rise quickly. Therefore, organizations seek to be agile enough to quickly act to these factors in order to remain competitive and to quickly deliver value. Following service orientation principles allows an organization to automate its business processes based on interoperable, standardized, reusable, and composable services, which in turn allows the organization to be more flexible, to quickly adapt to a new or changing business events. Cost saving is another goal that SOA helps organizations achieve. As more and more interoperable and standardized services become available for use, the cost of building applications will continue to drop. A typical problem in organizations, especially ones that grew quickly, for example, as a result of measures and acquisitions, as the existence of silos of redundant functionality and data, SOA helps the consolidation of such redundancies, which means the reduction in development and operation costs. Another area of cost saving is that through SOA's promise of being platform independent, organizations get to choose the most suitable, cost effective platforms and technologies.

Examples of Strategic Goals and the Role of SOA

Now that you understand the main strategy goals that SOA helps to achieve, let's see some examples of how organizations layout the goal statements and let's take a high level look at how SOA helps achieve these goals. The first goal is to launch a new business processes in two weeks or less. SOA helps achieving this goal by defining a set of standardized and interoperable reusable services that are identified based on business needs. Business processes can then be assembled quickly without the need to recreate functionalities either by your using existing services or by composing a set of services together. In addition, by designing a common information model, services become semantically interoperable, which means services have common understanding of business entities and common message formats. This in turn means less time is required for these services to collaborate in a composition. Reusability composition and information modeling are all explained in detail later. Another goal would be given our limited resources, optimize the operation of our most critical business processes. SOA requires a top-down analysis approach to identify services based on business needs. This is the primary difference between a SOA and a non-SOA-based approach for service identification. In SOA, the driver behind service needs is business, not IT. As we do this top-down analysis, we get to understand business, its needs, and its strategy, and therefore, the business priorities. Then we identify the services that help optimize the operation of business processes that are most critical to the business strategy at this stage. Note that SOA requires also a bottom-up approach for service identification in order to expose existing system capabilities to support the business processes. Both top-down and bottom-up analysis and how SOA fulfills the business needs will be explained later in detail. One more goal statement would be

reduce our IT costs by 5%. SOA helps achieve this goal by providing a mechanism to quickly assemble business processes and applications through composition of existing services, which cuts down development costs. In addition, SOA promoted centralized functionality and unified access interface to data, which cuts down application and data operation costs. Of course, the common theme about all the previous examples is that they all work towards the most important goal of SOA, which is business/IT alignment. For all the previous business goals, SOA attempts to provide an IT architecture that is driven by business needs. Every service established, every asset exposed, every composition built, and every business process automated is done to achieve specific business goals. Once that happens, an important step is taken towards business IT alignment.

SOA Misconceptions

The previous sections have given you a high-level understanding of what SOA is. In this section, let's examine some of the most common misconceptions about SOA. As I said at the beginning, it's these misconceptions that lead to many people thinking that SOA is dead and to many SOA implementations to fail. SOA is not a bunch of web services talking to each other. Believe me, that's the most common way I've seen people describe SOA. Service oriented does not mean web services exchanging messages to fulfill a certain function. As you will see in detail later in this course, this has nothing to do with SOA. That's just web services talking to each other. Now keep in mind, I'm not saying this is a bad thing. Not every problem requires SOA, but just having an architecture where web services talk to each other is not SOA. It might be a great architecture, but it's just not SOA. SOA is driven by business goals. It's true that SOA is an IT architecture, which means that at the end, it affects the IT environment, but it should be driven by business needs. One of the problems of communicating the value of SOA to business people is actually the presence of the word architecture. As soon as business people hear this word, they immediately think IT and they judge the talk of being technical in nature. Therefore, it's important to communicate SOA to business people in the language they understand starting with the strategic goals we talked about and using the business architecture terms that will be discussed later. SOA is not integration. Actually, one of the technical benefits of SOA is the reduced need of integration through standardization and interoperability at both, syntactic and semantic levels. What role does integration play in SOA as discussed in detail later in this course. SOA does not require total IT redesign. A major part of SOA is reusing existing assets such as databases, legacy systems, applications, APIs, and so on. SOA would be an immediate failure if it had required to implement or throw away ease of investments. The using existing assets is part of the bottom-up analysis that is an important part of SOA and will be explained in detail later. SOA does not mandate a big bang approach. In fact, like everything in architecture, it's essential to carefully scope the work in order to provide value as quick as possible and reduce disruptions of business operations as much as possible. It's entirely possible to divide an enterprise into a set of interrelated business domains and scope the work of SOA on a business domain level. And this actually is the technique you will see when we discuss the case study. SOA is a long-term investment, or more specifically, it provides more value with time. Now this does not mean that you will only see value

after a long time. As I said in the previous slide, you must plan an iterations and provide value as soon as possible. However, this only means that the investment in SOA will pay off more with time as standardization, interoperability, reuse, and composition play an increasing role in automating business processes. Simply put, more time means more services, means more flexible business processes, which translates to increased organization agility, reduced costs, and overall business/IT alignment. And finally, SOA is not a vendor-based solution that can be used in a plug and play mode. Certainly many vendors want you to think this way, and in fact, market their SOA stack that way. This is simply not true and is actually one of the most important reasons SOA is often misunderstood. Sure vendor solutions help in different areas in a SOA lifecycle and SOA will eventually be implemented using vendor technology, but as you will see in this course, you will spend a lot of time learning about SOA without talking about technology. In fact, the entire SOA analysis and overall architecture phase is vendor independent. It's not until we reach the detailed design and implementation in the next course that we will start talking about technology. So next you get a visit from a SOA vendor promoting the so-called SOA stack make sure to let them understand that for you SOA is an architecture style. It's our approach and methodology, a way of doing things, and not simply some sort of a magical plug and play solution stack.

Case Study: Early Conceptual Demonstration

So now that you understand the concept behind SOA, how will SOA help the government agency achieve its goals? So our agency has a set of business processes that essentially run the business. This processes crosscut multiple business domains. This processes are in turn supported by a set of existing systems. These systems are built in siloed manner with redundant information, redundant functionalities, inconsistent data models, and the fragmented data. The processes access the systems through a set of web services and APIs that expose the same functionalities without regard to an overall business architecture, the usability, composability, and semantic interoperability. Because of this, the business processes are difficult to change. New business processes are difficult to be launched quickly enough. The development and operation costs are high and customers are having inconsistent experience dealing with different domains of the business. So given the organization's understanding of SOA, what it represents and what goals it is intended to achieve, the organization realized the importance of SOA and helping solving these problems, so it turned into SOA to increase agility, reduce development and operation costs, and provide a consistent customer view. As I explained, SOA used existing assets. So conceptually, it's actually sitting between the business processes and the information systems. So what are the building blocks of SOA, how these blocks are structured, and how do they interact, how our service is identified and what design of principles to apply. These and more topics will be explained in detail in the following modules.

Summary

SOA is a service-based architecture style that is targeted to achieve the strategic goals associated with service-oriented computing. Business IT alignment, agility, cost reduction, and customer centricity are major business goals that organizations look for. By relying on standardized, interoperable, reusable, and composable services, SOA increases flexibility of business processes. It reduces time and cost required to change and launch a new processes and provides a consistent view of data. SOA is often misunderstood of being an integration approach, a vendor-specific solution, or an IT-driven solution. This is not true. SOA is business-driven investment that provides a great value as time passes by and as SOA adoption increases. However, SOA can and should be planned iterations and reusing existing assets is essential in order to reduce the impact of change and provide value as soon as possible. Although a service in SOA is a generic concept, it's true that web services are the dominant technology due to their inherit interoperability, loose coupling, and abstraction. Okay, so now you understand SOA from a high-level point of view, but like any non-trivial decision you have to take, there will always be consequences and challenges and understanding this is a critical success factor. So in the next module, let's examine the challenges of implementing SOA.

Challenges of SOA's Success

Introduction

Welcome to module, Challenges of SOA's Success. You are watching Real World SOA: Analysis and Design. Despite all the benefits that SOA brings to an organization, successfully adopting SOA faces a series of challenges. Understanding these challenges is important for two core reasons. First, to decide if SOA is worth the effort, as you do a tradeoff between the benefits and the challenges, and then if you decide to go the SOA path, in order to have a strategy in place to overcome these challenges.

SOA Misunderstanding

First of all, SOA is often misunderstood. This misunderstanding often leads to wrong expectations from business owners and wrong implementations from architects. Please review section SOA Misconceptions in the previous module to see the sources of confusion.

Organizational Challenges

The main challenges facing reusability are organizational and political. A service lifecycle has to managed and maintained across organizational unit boundaries. For example, a recall from our case study that the business domain

is divided into a set of separate subdomains or units. When a service consumer that belongs to a unit uses a service that belongs to a different unit, the consumer has established a dependency on that service. This means that the service provider must guarantee a service level agreement where the service operates reliably, is supported and timely mannered in terms of bug fixes and operations, and responds to a change in business requirements and timely mannered. Of course, changes that are requested by a certain consumer must not cause existing consumers to stop working, which also mandates the presence of versioning strategy. Now when the provider and the consumer both belong to the same unit, this is somewhat easy to handle because both are owned by the same entity. Across different units, however, as you can see, these are non-trivial challenges. To tackle these challenges, the service provider must pay the following costs. Manage and maintain the service lifecycle across different organization boundaries. In other words, it's not enough to think of a service lifecycle with respect to the unit, which presents the service. A special consideration must be given to how the lifecycle affects and is affected by consumers from other units. The service provider must also guarantee service availability, reliability, and support as part of the terms of the service level agreement. Since this service is used from different units, it will have handle an increasing number of consumers, and therefore, must satisfy an increasing number of runtime and operation requests. This makes availability, reliability, and support ability requirements even more demanding. The service provider must design the service to be flexible enough for extensibility. Since the service is reused across different units, then it must be flexible enough to be changed or extended. Applying the appropriate SOA design principles, which will be explained in the next course, therefore, becomes a critical part of SOA implementation. The organizational challenges of SOA are explored in more detail when we talk about governance later.

New Development Paradigm

Why SOA decreases development costs through service they use. It requires a shift in development mindset. Services cannot be created in isolation anymore. In other words, in projects requiring a certain service cannot go in and just create the service and move on. Instead, it will need to consult a service inventor to check if the required functionality already exists, or if it can be achieved by composing a set of services, or if there is an existing service that can be edited, but while still be valid for other consumers, or finally, if there is a need to build a new service. This means that service development must now be part of an overall application architecture. This architecture is based on a hierarchy of services. It is directed by a business architecture and it conforms to a common information model. And development must also be guided by a special SOA methodology of service, analysis, and design. All this means more analysis and design cost than building individual services without SOA. Now it's important not to mix this with the cost savings strategic goal or service-oriented computing. Cost saving becomes more tangible over time as the services pool becomes bigger, but any surely the cost of development paradigm shift cannot be avoided. Over time, the cost savings gained by the services pool start to outweigh the cost penalty of the new development paradigm.

Establishing an Information Model

SOA requires the creation of a common information model. As we will see later, interoperability comes in two shapes, syntactic and semantic. Semantic interoperability mandates the presence of a shared information model, which guarantees that SOA services have common understanding and representation of entities, which speeds up service creation and enhances performance. Don't worry for now. We will talk about information modeling later in this course. However, why such model provides numerous advantages establishing it mandates extra time. In addition, at some stage, transformation must be done to map semantic objects to physical data models used by the existing information systems. This also impacts time and performance.

SOA Is A Long-term Investment

SOA is a long-term investment. While planning and alternations to provide value as soon as possible is a critical aspect of SOA planning, but the fact is the overall SOA benefits are mostly apparent with time, when reuse potentially grows. This takes time and requires an organization that is willing to put up front investment. For an organization to achieve this state of a true agile enterprise requires all the building blocks of SOA to fit together, organization changes, business building blocks, information model, services, and information systems, and these building blocks must also be supported by governance, business monitoring, and technical monitoring. As you can imagine, reaching this level of maturity is not an easy transformation and requires the willingness to do a long-term investment.

Key Takeaway

The key takeaway from this discussion is that you must know the costs that you will pay for a successful SOA implementation. It's only then whether you can tradeoff this costs with the benefits of SOA and it's only then when you can make a sound decision if SOA is the right approach.

Summary

In the previous module, you learned about the benefits of SOA and the strategic goals that SOA helps achieve. However, understanding the challenges is equally important. SOA is often misunderstood. Some people think it is just another way to do integration, others think it's a product from a vendor, why some people think that as long as you're creating web services, then you're doing SOA. Therefore, it's essential that you clear up the confusion and establish a common understanding of what SOA clearly is. It's important to understand that adopting SOA requires establishing governance policies, establishing an analysis and design method specific to SOA requires the creation of a common semantic information model, and that organizations must be willing to invest and see benefits as time passes by. It's only after understanding these challenges that you can do a tradeoff analysis with the benefits and then decide if

SOA is the right solution for you. In the next module, we'll wrap up our discussion around the context as we discuss some of SOA planning considerations.

SOA Planning Considerations

Introduction

Welcome to module, SOA Planning Considerations. You're watching Real World SOA: Analysis and Design. In this module, we'll look briefly at some of the core aspects of SOA planning. The purpose is not to discuss project planning in detail. Rather, I just want to briefly touch on some key concerns to keep in mind while planning for SOA.

Secure Executive Sponsorship

As you know by now, SOA is a business-driven transformation that helps organizations achieve strategic goals. It is a long-term investment. It requires organizational and business changes and it requires the adoption of special implementation methodologies. Therefore, something as big as SOA cannot succeed without executive sponsorship. Such sponsorship should come from the C-level executives, such as a CIO or even CEO depending on how an organization is structured. But the important thing is to have this sponsorship in the form of a signed document that specifies executive level information, such as the purpose, objectives, scope, milestones, and resources. Typically, this document is called a Charter. Without having this level of sponsorship, it's impossible for organizations to be able to embrace the needed business and IT changes and allocate the required budget.

SOA Team

One common mistake people make is to think that a SOA team consists only of people who create services. In other words, those people who analyze, design, implement, test, and deploy services. While these roles are of course core to SOA, they are not alone. Different organizations might have different roles, but the following list shows a common set of roles critical to SOA. As just explained, SOA needs executive sponsorship. These sponsors automatically become the most important people. If you can provide and show them business value, then SOA succeeds. As mentioned briefly when we talked about SOA challenges, business and IT changes across multiple business domains within an organization. Therefore, the owners or leaders of the business domains are also important members. They must collaborate with the implementation team, and in return, they must be shown how SOA is providing value for their own domains. As I will discuss shortly, this is why having a solid stakeholder management plan and working in increments are two very important planning aspects of SOA. As we will discuss in detail later, business architecture

plays a crucial role in SOA. Business architects need to articulate or even create a business architecture that is a prerequisite for deriving business aligned SOA services. A SOA analyst analyzes business processes and identifies SOA services. As we will see later, these services become part of a service inventory and are grouped into different layers. A SOA analyst also identifies the information model of SOA, which will also be explained in detail later. A SOA architect is responsible for the overall architecture of the SOA reference architecture. This includes things such as service interface design, message exchange patterns, security, service layering, and so on. Keep in mind here that we're talking about roles and not people, so it's very common to have the same person playing various roles or at least multiple activities from various roles. SOA services are eventually consumed by various challenges such as web and mobile applications, in addition to business process management engines. Each of these systems must have an architect role that designs the consumption of SOA services. In addition to all these roles, there are of course the more traditional roles of service designers or implementers, testers, deployment, support, and monitoring.

Scope Project

In addition, governance is a critical aspect to SOA and a governance team must exist. Because SOA requires business, organizational, and IT changes, attempting to implement SOA across the entire organization at once will most probably fail. This is of course especially true for large organizations and organizations that have low SOA maturity, for example, organizations that are implementing SOA for the first time. In this case, it's important to carefully scope SOA around one or a few interrelated business domains. Once SOA is scoped this way around domains, the effective changes are minimized and scoped. The mindset is gradually changed. Business and IT changes have boundaries. And most importantly, value of showing faster, which motivates other business domains to apply SOA. Of course, selecting the business domains to start with is not an easy task. It involves study of organization and business needs either defining business priorities and the planning is smooth transition while preserving business continuity. In the case study, I will show you the considerations that led to selecting the business domain, which scope SOA implementation.

Stakeholder Management

Stakeholder management is a well-known logic management activity. In SOA, it has an even more important role. They call one more time that SOA requires an organizational, business, and technology changes. Therefore, stakeholder management involves business and technical stakeholder groups having different expectations and concerns. For example, executives care about strategic objectives and return of investment, things such as agility, cost reduction, time to market, and so on. Line of business owners or business domain owners care about service level agreements, operation level agreements, business continuity, and so on. While system or application owners

care about system disruptions, time schedules, meeting business requirements and so on. Therefore, it's critical to provide these stakeholder groups with data they need that shows them value and to manage their expectations.

Plan in Increments

Even after scoping SOA for a business domain, planning in increments is still critical show value as early as possible. If you implement SOA in a business domain and say it takes you one year to release the first set of services, chances are you will lose the support of the domain owner and probably won't have the chance to continue anyway. Planning in increments is a well-known software engineering approach. In terms of SOA, this means identifying the most important services to start with. These are the top priority services. You'll build these services and start showing the value in terms of agility. You then move on for the second round of services and so on. Now the way to identify the priority of services requires understanding business needs and I will talk about this in detail later.

Case Study: Planning for SOA

The last time we saw a case study, we have identified that SOA is the correct approach that will help the organization achieve the business goals. So the first thing we have to do is to secure executive sponsorship. To do this, we have to communicate with executives using their own language. We talked to them agility, time to market, cost, and customer centricity. We then explained how our IT strategy based on SOA will help them achieve these goals. Next, we need to scope our work. We knew that applying SOA one shot across all domains or ___ because the changes will be just too much to digest and value would need a very long time to be proven. So based on business analysis, we decided that the study, application, and finance interrelated domains are the most critical, and therefore, they would be the scope of our SOA work. But still, there are a lot of processes crosscutting these domains and we also need to give priority to the processes that will be a part of our SOA analysis. Therefore, we create a list of processes that will be a part of our first SOA increment, second increment, and so on. And finally, having scoped our SOA and set up the priorities and increments, we now have a solid idea about our stakeholders, and consequently, can manage their needs and expectations. Executives are always the most important stakeholders as they look at business objectives across the entire organization, rather than a specific business domain. But since we selected the business domains, the business domain owners all become core stakeholders. In addition, system owners are also at the top of our stakeholder list.

Summary

Planning SOA requires project management skills just like any other project. In this module, we took a brief look at specific planning aspects that have special importance for SOA. SOA requires funding and change in organization methodologies. Therefore, senior management sponsorship is mandatory. And because SOA closes the business and

technical domains, the SOA team must be wrote so that business and technical rules are present. Scoping SOA projects and planning in increments are essential planning tools for SOA in order to limit the change impact and deliver value in a timely manner. And finally, again because SOA crosses the business and technical domains, stakeholder management is very important in order to gain and sustain the support of various stakeholder groups. This module completes the discussion around the context and the why aspect. In the next module, we'll do a quick recap and introduce the next area where we will discuss in detail the what aspect covering SOA overall architecture.

Pause and Recap

Pause and Recap

Let's now take a breather and have a quick recap of what we've covered and a look at what lies ahead. SOA is a very misunderstood term. I've met any people thinking that SOA is some kind of web service integration or a vendor product and many other misconceptions that we talk about in this module. Because SOA is misunderstood, there are so many implementations that are incorrectly called SOA or implementations that fail because of lack of understanding of business goals and challenges. Because of this, I've decided to spend time setting up the context of SOA. As we introduce the business domain, what is SOA and what is not SOA, why SOA is a suitable solution, and what are the expected challenges. Once you understand this context, then you have what you need to take a well-informed decision of SOA is right for you by trading off the business goals that SOA will help achieve versus the cost you have to pay to tackle the challenges associated with SOA. Next, we start a discussion about the what aspect of SOA as we will examine SOA as an architecture this time. In the next few modules, we talk about SOA reference architecture, the overall structure and interactions of SOA building blocks, service classifications, and service granularity.

SOA Fundamentals

Introduction

Welcome to module, SOA Fundamentals. You are watching Real World SOA: Analysis and Design. In this module, we'll start the journey of the what aspect as we understand what is SOA and what it is made of. In this module, we start by examining SOA as an architecture style. You'll learn the fundamental building blocks of SOA in terms of a SOA reference architecture.

The A in SOA

So far, we have established that SOA is an architecture style for building IT architectures based on services and these services are designed following service orientation design principles in order to realize a set of strategic goals. The architecture of a solution is about the structure and dynamics of the elements or building blocks that make up the solution. So what are these elements in a SOA solution? Now if you're thinking services, then this is only a partially true answer. Although SOA will at the end reduce a set of business aligned reusable and composable services, however, the entire SOA solution is composed of more than just services. A SOA solution is made up of the following building blocks. The business processes that services aim to automate, it's the flexibility of these processes that is a critical step towards an agile enterprise, then we have a set of business services that automate tasks within the business processes. The term business service, instead of just service, is to indicate that these services are identified based on business needs and analysis independent of any existing technologies, platforms, and systems. Then we have another set of supporting services that expose existing assets, such as existing applications and data. These services provide business services access to existing resources without forming coupling on these resources, again, something which is very important to achieve flexibility. The resources themselves are part of the entire SOA. Although they are not something that you will explicitly build as part of SOA, but the entire SOA solution depends on them, and as such, they are an essential element of the overall architecture. Messages exchanged by services are based on an information model that is in turn built on top of the organizations and formation architecture. Products and technologies are enablers for SOA. It's true that SOA as an architecture style is platform and technology independent, but just like any other architecture, it needs physical architecture to be implemented and the choice of platform, products, and technology is an important architectural decision. Now at this stage, you might be having a difficult time envisioning how all these elements relate to each other, so in the next section, let's examine a reference architecture that puts all these elements into perspective.

Reference Architecture

So a SOA includes a set of elements or building blocks, and as an architecture, you need to understand how these elements fit and interact together. A reference architecture acts as a blueprint for creating architecture and it provides templates and guidelines for making architecture design and implementation decisions. Have a reference architecture means having a starting point to your actual architecture creation. Now in terms of SOA, a SOA reference architecture should provide a blueprint for the structure and interactions of the SOA building blocks explained in the previous section. The best way to understand the reference architecture is to actually see it. First, we have a set of horizontal layers. These layers all have to do with service creation and consumption. The first layer in the reference architecture is the existing resources. This includes things such as legacy systems, ERPs, custom applications, data stores, APIs, and so on. As I said before, it's important to not convey SOA as an architecture that requires completely work of an organization IT architecture, therefore, the proper use of existing resources is a critical

consideration when designing SOA. The services layer is the heart of SOA. It's the identification, design, and implementation of services where we will spend the majority of our time. The services layer itself is divided into two groups of services. Business services are those that are meant to provide the business functionality required to automate the business processes. It's the standardization, interoperability, reuse, and composability of these services that leads to flexibility and agility. Business services eventually need to talk to the existing organization resources, however, they do so via another set of integration and utility services. This is important in order to loosely couple business services from existing systems, platforms, and technologies. This loose coupling is essential for achieving flexibility, and consequently, enterprise agility. This is just a high level of review of the service layer. The service layer is explained in more detail in the next section and further revisited as work out service identification. In addition, integration services have their own module in the next course since they are more into the detail Design area. Business processes indicate how the organization conducts its business. In other words, how it achieves the required business results or objectives. Business processes consume SOA services to automate certain tasks. Now it's important to note that the analysis and design of business processes is not part of SOA. Business process, analysis, and design typically falls under the umbrella of a separate, but related discipline known as Business Process Management or BPM. So having the business processes as a prerequisite for SOA service identification. Do not worry if this is not clear for now. I will revisit this topic later. The business process services layer simply indicates that business processes are exposed as services that can be consumed by systems. In other words, the entire business process is a service from the perspective of the business consumer. I will talk more about business process service layer and how it differs from the services layer in the next module. And then we have the channels layer, which indicates the mediums consuming our SOA services. These can be web applications, mobile apps, internet portals, and so on. Channels mostly consume business process services, but they can also consume services directly from the services layer. For example, a web application might invoke the change major business process service, which would in turn kick off an entire business process, which in turn utilize SOA services. But the web application might also directly use an in-mail utility service to send any mail or a service which returns information about the student entity. Next, we have a set of vertical layers that deal with the crosscutting concerns. The information model layer is about establishing a semantic information layer for SOA services. This topic will be explained in detail later in this course, but in a nutshell, SOA services must establish semantic interoperability where they have common understanding and representation of business entities regardless of how data is physically represented in existing systems. This greatly increases the flexibility of services. The integration layer is about providing a set of integration-related services, things such as mediation, transformation, orchestration, and so on. As I explained before, this layer is one of the most sources of SOA confusion. As vendors, and specifically enterprise service bus vendors, attempt to promote SOA as a product in the shape of an ESB offering. As I explained, this is not true and while an ESB is a tool that helps implement SOA, it's just a single building block in the entire reference architecture. The governance layer, the first of the actions required to make sure that policies, procedures, principles, guidelines, and standards are preserved during and after SOA implementation. This is essential to ensure that SOA is successful in delivering the business value it's supposed

to deliver. Governance is discussed in the next course. The management and monitoring layer is about management and monitoring of SOA in order to preserve the quality of service attributes, such as availability, security, and performance. This includes tools of technical nature, such as Microsoft System Center and tools of business nature, such as business activity monitoring. Business activity monitoring is discussed in the next course, while technical monitoring tools are outside of the scope of the course. Now before closing this section, keep in mind that reference architecture is organization-specific. Because different organizations might have different building blocks or implement different hierarchy or interaction between the building blocks, the reference architecture we saw here is common to SOA and is almost always a good starting point, but variations could happen based on your needs, so keep this in mind.

Service Inventory

The service inventory is a pool which contains all services identified and classified as part of our SOA. The inventory is typically partitioned as per our service classification. When we identify a need for a service, we first check inside the inventory. If the service already exists, then we reuse it. If it does not exist, then we look if we can build it by composing other existing services. Only if both options fail, then we know that we need to build a new service and this service then becomes itself part of the inventory. There's actually another option, which is to edit an existing service to match any of your needs, but editing services needs to be managed carefully because this means creating a new version of the service, so we have to manage which consumers use the new version and which consumers continue to use the old version until they become ready to use the new version. The impact of change can be minimized if the change is a non-breaking change. I will talk more about versioning in the next course. Notice that at the first stages of SOA implementation the rate of adding a new service to the inventory will be high, so reuse opportunities will be low. Over time, however, as the service inventory gets bigger, reuse opportunities will increase. Up until we reach a stage where most of the service required can be either reused or composed from existing services, flexibility and cost reduction gains are increased with time as the service inventory becomes more mature. SOA in summary, a service inventory has two major design goals. It is a single place where we look for services for reuse. It has information about the context and function of each service and how services relate to each other. By always looking into a central place for reusability, we reduce the possibility of creating redundant services, and this in turn, speeds up service discovery, and therefore, services can be more easily reused. It also helps in SOA governance since the inventory presents a confined boundary of services where it becomes easier to apply standardization, policies, procedures, and rules. Now how is a service inventory physically represented? This will be explained in the next course. For now, it's enough to know that a service inventory is a pool of services.

Reuse and Composition

Reuse and composition are related design concepts and are really at the heart of SOA. In simple terms, reusability is about having services that can be reused multiple times. So we'll say we have various business processes all requiring a piece of functionality. All of them can then reuse the same service. So the service is built one time and reused many times. Composition is when we need a certain functionality that is not available in any single service by itself, but can be achieved by combining capabilities from multiple services. Both reusability and composition allow SOA to help organizations move toward agility as it becomes easier and faster to introduce new business processes and change existing ones. However, usability and composition do not come without a price as they require a balanced application of SOA designed principles. For example, services will not be effectively used if they are not designed to be fast in execution because as they are more and more used, they will become a bottleneck, which will turn consumers away, and therefore, failing the entire SOA effort. Similarly, for services to be composable, they need to be able to effectively communicate with each other without creating some sort of special communication mechanism for every different composition. If this happens, again consumers will stay away from composition, and again, the entire value of SOA is negated. So in order to tackle these and many other concerns, SOA architects need to apply design principles with proper tradeoffs. Design principles and other implementation considerations are discussed in detail in the next course. For now, just keep in mind that the business advantages of reuse and composition need the balanced application of design principles and implementation practices.

Service Functional Context

A functional context specifies the purpose of a service. In other words, the functionality that this service intends to deliver the results of its existence, if you will. Now each service should have a distinct functional context that is made up of the set of capabilities that the service exposes. So why is it important to define the functional context of a service? Well for two very important reasons. First, is to group related capabilities together. Capabilities that serve the same functional context are grouped together into the same service. This allows for consistent and manageable service design and discovery. So if you're looking for a capability that is part of functional context A, you consume service A. If, however, you are looking for a capability that is part of functional context B, then you consume service B, and so on. The second very important reason is to prevent redundant functionalities across services. Recall that one of the advantages of SOA design is a single point of access for functionality and data. By uniquely defining a functional context of a service, you guarantee that no other service offers the same functionality and no other service is a point of access to the same set of data. Let's consider a couple of examples. The functional context of a student entity service is the student business entity. All capabilities that are required to retrieve or update data related to a student profile are centralized in this service. No other service holds the same capabilities and the student profile is only accessed by this service. Similarly, the functional context of an ERPInventory integration service is the inventory data. No other service retrieves or updates ERPInventory data and all related capabilities are grouped in this service. Finally, the functional context of an approved task service is defined by the specific business task that is required to

automate, in which case, approving a vacation request. Most of the times, the functional context of task services depends on composing other services.

Side Note: Is SOA Still Relevant as Cloud Adoption Accelerates?

With the cloud booming and more and more organizations and companies moving to the cloud, you might ask why still bother and layer SOA. Well, SOA is still important and will continue to be for a long time to come. It's true that cloud movement is accelerating and cloud advantages are numerous, but do not expect on-premise setups to go away anytime soon. There are a couple of reasons for this. Organizations have put a lot of investment in existing setups. These organizations will be either hesitant to move to the cloud, or at best, they will adopt gradual movement, which will take years to materialize. Other organizations might have security concerns that just prevent them from moving their data to the cloud. The issue has been well-debated and cloud providers are providing more security assurances than ever, but the fact is some organizations, mostly government organizations, will just not move their data. For these types of organizations, there are two possible setups. They either dismiss the possibility of cloud altogether or they adopt a hybrid model where part of their applications and non-sensitive data sit in the cloud while other applications and sensitive data remain on-premise. In all of these cases, organizations will still require on-premise setups and SOA, if the business case calls for it for sure, will continue to play an important role in creating a giant enterprise. But SOA can also play an important role in delivering cloud services. Software as a service is a well-known business model for offering services to customers. As a consumer, you only see services, but software as a service providers have an entire setup running to deliver these services. So software as a service providers can use SOA to create reusable, composable, and interoperable services in order to create a giant business processes. This allows software as a service providers to better respond to new opportunities and market demands while lowering the development costs.

Case Study: Early Look at Reference Architecture Applied

Given what you know about how a typical SOA reference architecture might look like, let's see a quick illustration about how such architecture might fit as a solution in our business case. So recall that we have a set of business processes crosscutting our business domains that will be part of our SOA scope. This business processes are backed up by various existing systems, platforms, APIs, and so on. We want to enhance the flexibility of these processes and we decided to do so by applying SOA. Now applying the reference architecture that we saw in this module, the existing information systems are part of the existing resources layer. We want to reuse these systems without modification in order to limit the impact of SOA. We will create a set of services that are technical in nature that will access the information systems. These services will loosely couple the business services from technology concerns. Both set of services form our SOA inventory, which is populated via our SOA analysis and design methodology. The

input of this methodology is of course the business processes that we are trying to get the flexibility enhanced. Each one of the processes is itself exposed as a business process service, which is consumed by one of the front-end channels that applicants and students use. Business services exchange messages based on a canonical semantic information model. This is yet another step towards creating flexible business processes as we design our business entities regardless of how physically they are stored in the information systems. This means that we must have transformation capabilities between this semantic model and the actual data models used by each of the organization resources. Transformation capabilities and other integration services are offered by an integration layer. A sort of governance model must be in place in order to set policies, guidelines, and rules to govern service creation and the evolution of these services. And of course, services will be monitored and managed to confirm services operate within the expected quality measures and provide the business value they're supposed to deliver.

Summary

In this module, we started to look at SOA overall design represented by the SOA reference architecture. A SOA reference architecture indicates how SOA building blocks are layered and how they interact with each other. The service is identified as part of our analysis and design methodology from the source service inventory. As the inventory grows, reuse and composition speeds up business processes changes and creations. Each one of the service inventory services must have a unique functional context such that functionality and data access are centralized in services, which means there is no redundant functionality and data consistency is improved.

Service Layers

Introduction

Welcome to module, Service Layers. You are watching Real World SOA: Analysis and Design. In this module, we continue what we started in the previous module and that is discussing SOA reference architecture. Now we're going to take a detailed look at the services layer, the core layer of the SOA reference architecture. In the previous module, you saw a typical SOA reference architecture. In this section, let's zoom in on a core layer, the services layer. There are various ways how services are classified and actually there are no different right and wrong ways since each organization might have different characteristics, which might result in specialized service layering. That being said; however, there are two fundamental characteristics that must be honored in any SOA service layering approach. First, there must be a layer for business services. These business-oriented reusable services are the essence of SOA, which helps organizations to achieve agility. Second, there must be a service layer, which loosely couple these business services from the existing organization assets. Abstracting this organization resources behind

this layer allows the business services to evolve independent of existing systems, platforms, technologies, and APIs. This is essential to achieving the required business flexibility. Now given these two architecture constraints, let's now examine the service types in our SOA reference architecture.

Process Services

Those services are not exactly part of the services layer the way it's represented in the reference architecture, but process services are an important part of SOA since they represent the output of SOA to business consumers. Now let's examine the characteristics of process services. Process services encapsulate an entire business process. As such, they are mostly long-running services that can spend hours or even days to complete. For example, a major service is a process service that encapsulates the change major business process. When, for example, the change capability of the major service is called, the change major business process is triggered. The process itself can take hours or days to complete. Upon completion, the consumer of the major service might get a completion notification via a separate endpoint in the push mechanism or the consumer might pull a status check endpoint in a periodic manner. As such, process services typically contain a mix of automated and human-based business tasks. Technically, a business process is typically implemented via an orchestration engine, such as a business process management engine. Therefore, a process service is typically the published endpoint of the business process directly from the business process management engine.

Task Services

Task services fall into the category as business services inside the services layer. As explained, business services mean that these services are driven by business goals and are derived using top-down analysis using business architecture. Do not worry for now. Analysis will be explained in detail later. For now, let's examine the characteristics of task services. A task service implements a business task within a business process where this business task is a non-manual step in the process. For example, in a change major business process, we might have a task which checks the eligibility of a student to change the major. Such task might require the execution of multiple services and the application of a certain business logic. This kind of processing is automated via task service. Therefore, task services are typically rich in composition as they compose different other services. Task services are mostly real-time services that are executed in a single atomic transaction where the service executes and returns the result in a requested response manner. However, less commonly, task services can also be long running if the business task is itself long running requiring asynchronous callbacks. You will see both types of task services later in the case study. Task services are typically not widely reused since they are essentially meant to automate a step in a specific business process meaning that the functional context is typically tied to a specific business process. A reusable task service is usually the result of automating a sub-process, which is itself reused across different parent processes. And finally,

task services can be implemented using an orchestration engine, such as one from an enterprise service bus. This engine helps and compositions data transformation, data aggregation, atomic transactions, and service and location. This is a detail design and implementation consolidation, so it will be discussed in the next course. The question typically arises, why create a task service, instead of creating the composition logic directly inside the consumer applications. So in other words, if we were to visit the portion of the reference architecture, why make a business task and a business process call a task service, which then performs the required automation? Why not try the automation logic inside the business task itself, especially that the business process is most probably being implemented in a powerful business process management orchestration engine. Well there are various important reasons. The first is reusability. By writing the correlated task service, functionality is available in one place, and therefore, can be reused by multiple consumers without having to code the functionality for each separate consumer. Another important related reason is that changes are done in one place. So if the task service logic has changed, changes are done in the centralized service and all consumers pick up the change. Of course, this brings up versioning considerations where some consumers might not want to use the update just yet, in which case, we must have a versioning strategy of the service and versioning is discussed in the next course. We find an important reason is hiding implementation details. By encapsulating the logic in a task service, implementation details are hidden from consumers and consumers only care about the business functionality. This alone is a very important reason to use a separate task service even if the service is itself not reusable because the composition logic can change without affecting the business process as long as the input and the output of this service remain the same.

Entity Services

Just like task services, entity services are also business services driven by business needs and identified using top-down analysis starting with business architecture. However, entity services have a different set of characteristics than task services. So let's examine those. Entity services have a functional context around business entities, such as a student, employee, invoice, and so on. These are highly reusable services. Because the functional context is independent of any parent business process. What I mean by this is that while the task service is built mostly specifically to serve a certain business functionality within a business process, entity services are business process independent and are reused across various processes. So for example, a student entity service, which offers capabilities to retrieve and update student information is clearly independent of any specific process and will be required by almost any process that deals with a student profile. So the capabilities exposed by entity services revolve around accessing and manipulating the corresponding business entities, which the service functional context covers. In other words, a student entity service would not contain a GetInvoice capability, for example, because such capability falls into the invoice functional context, not the student functional context. Entity services typically play a supporting role to task services by being reused in various compositions. In other words, task services compose entity service in order to get and manipulate data of business entities. Entity services contain capabilities

that resemble, create, read, update, and delete operations, for example, AddStudent, GetStudentInfo, UpdateStudent, and DeleteStudent. But they also contain domain-specific capabilities that are specific to the business domain being tackled, for example, SuspendStudent and GetStudentChangeMajorHistory are such domain-specific capabilities. It is best if we prevent an entity service from invoking or composing other entity services. The reason is that entity services are the most reusable business services, and therefore, they are best kept autonomous by not depending on the functional context of other entity services.

Task vs. Entity Functional Context

Before explaining the other types of services, let me elaborate more on the difference between the functional context of task and entity services. An entity service has a functional context associated with the business entities it corresponds to. For example, consider a class entity service. This service has a functional context of the class business entity. The service might have a domain-specific capability called IncrementRegistration, which increments the number of reserved seats. The service would also contain CRUD-like operations in the form of Insert, Update, Delete, and RetrieveClass. In general, the service has a functional context associated only with the class business entity and all capabilities operate within this functional context. Now consider a task service called RegisterStudent. This task service has a functional context, which is relate to a parent business process, for example, Student Reservation. RegisterStudent might have a capability called register, which composes multiple services, one of which is the class entity service. For example, the register capability might compose an entity service to collect student information and extend their service to check the credit balance of the supplied credit card, an integration service to talk to an internally RP system, and then perform some business rules, and finally cause the increment registration of the class entity service. So as you can see, the functional context of the entity service is autonomous. It lives alone and it can support multiple business processes and task services. While task services are business process specific and typically have a functional context, which is the summation of the functional context of the services they compose.

Utility Services

While both task and entity services they present business logic, and therefore, are classified as business services, utility services are typically non-business centric and are sometimes referred to as technical services. Utility services provide crosscutting functionality, such as logging, notification, exception handling, message formatting services, security services, such as token is showing and token transformation, encryption, and decryption services, and so on. Because they're to present crosscutting concerns, utility services are highly reusable.

Integration Services

Just like utility services, integration services are also non-business centric. Over integration services are really special, so let's examine what are the special considerations. Integration services are derived using a bottom-up analysis method meaning that unlike business services, they are not derived based on business needs and business architecture. Instead, they are derived based on the need to expose existing system capabilities as services. The purpose of integration services is to decouple business services from existing systems by providing an abstraction layer. So business services talk to integration services, which in turn communicate with existing systems. The integration services abstraction layer is essential to achieve business flexibility. There are three reasons for this. First, because it decouples business services from existing systems. This allows business services to exist and evolve independent of existing systems. So existing systems can be upgraded, replaced, merged, have their APIs changed all without affecting business services. Integration services hide these details and provide a consistent interface for business services. Although the internal implementation of integration services will change, as long as the interface with the business services remains unchanged, then business services will be unaffected. Integration services hide technical implementation details from business services. So for example, say a business service wants to update a student profile. The student profile data might be physically spread over multiple data stores. So for the update to take place, a transaction manager, such as DTC should be used. This kind of technical detail should be hidden from business services in order to preserve flexibility because maybe later, sources are merged and the DTC is not required anymore or a different transaction manager should be used. So these kinds of concerns are hidden in integration service, which exposes an interface that communicates with the business service using a student profile document. Now how exactly does student profile document is spread over data stores is an integration service concern and not a business service concern. Finally, business services define their interfaces based on a semantic information model independent of how data is physically presented or formatted in existing systems. This is essential for semantic interoperability. So again, in our example of the student profile, business services define the student profile in business terms in a way that makes sense from business perspective. Physically, the student profile data might be presented using different data models across different systems. Once again, this is hidden from business services and the required transformation between the semantic model used by the business services and the physical data models used by the existing systems is done at the integration services layer. You will learn about semantic interoperability and the information model later in this course. There is more to learn about integration services, but since they are a detailed design consideration, they are discussed in more detail in the next course, including the design tradeoffs between the flexibility gained by integration services versus the performance and manageability hits caused by introducing a new service layer.

What About External Services?

There is still one kind of service that we haven't discussed yet and that is External Services. All the services that are offered to our organization by external providers. For example, a credit card company offers a service to pay via

credit card, a shipment company offers a service to track shipments, and so on, but these external services are not limited to commercial providers. They might also be services offered by other partners. For example, in our case, there might be services offered by other government agencies as part of the e-government interaction. So how to work with external services? Well, people handle this differently, so let's examine two common approaches and discuss why I prefer one over the other. The first approach I've seen people taking is considering external services part of their own service classification and inventory. In other words, following our classification, our external service would classify as an entity or a task service and become part of the service inventory alongside the internally developed services. I do not like this approach for three major reasons. First of all, external services are designed by the external provider standards, not our standards. As I explained before, one of the advantages of service classification is standardizing design principles for each layer. If we include external services as part of the layer, then we're breaching the standardization. Second major reason is that I'm forming a dependency on the external provider technology. As we saw, I've already went into the trouble of establishing an entire integration services layer just to decouple business services from any technology concerns. So introducing this dependency back is not a very smart thing to do. Final major consideration is that of course external services will not be designed following our common semantic information model, which is breaching our semantic interoperability, since all our internally developed business services respect this model. The approach I take is to treat external services the same way I treat existing systems. So from the perspective of business services, external services are just systems providing access to certain resources. Business services communicate to these services using the integration services layer. So say for example, during my service analysis, I identify the need of a credit card payment functionality. I identify a task service regardless of the physical resource representation. Later, during integration services identification, I then create an integration service which talks to the credit card provider. So in this case, the only difference between external providers and internal systems becomes that external provider services live outside my organization boundaries, and therefore, integration services will have to connect to these services over an internet or VPN, for example, and consequently, we'll have to attack security concerns for these scenarios. But again, the beauty is that these concerns are not part of our business services layer.

Why Classify Services?

So why do we classify services? What is the benefit of associating types with services? Why at the end they are identified using the same SOA methodology? Well, classification of service types is actually important and has several advantages. By creating a common classification, different stakeholders can talk common language when discussing the type and nature of services. This allows stakeholders to communicate their ideas more effectively. For example, by talking about an entity service, everyone understands the functional context of the service being a business entity and everyone understands that it's reusable across different business processes. Similarly, when I'm talking about task service, everyone understands that it's composing other services that it automates a business task, and

that it has lower autonomy and performance measures. On the other hand, when we talk about integration services, system owners understand that we're talking about accessing their systems, so establishing a common language is a key aspect of any architecture, not just SOA. In fact, effective communication is just as important as having a solid architecture. By classifying services into known service types, similar design characteristics are applied to these services. For example, we know that business process services must be consumed by various channels, so we might need to support multiple communication protocols. We also know that entity services are highly reused, so the performance quality attribute has a higher importance in tradeoff analysis. Similarly, we know that task services are rich in composition, so we need to have an orchestration engine in place and integration services need to access multiple existing resources, so we need to have a DTC in place. Of course, these are just examples. Designed aspects will be explained in detail in the next course. Service classification also helps in project management. Where service is falling into the same service type, typically have similar implementation and maintenance costs, require similar skills, and have a similar set of stakeholders. All this allows a project manager to better estimate costs, and time plans, and resource allocation. As I explained, by separating business and technical services, you enhance loose coupling and flexibility. This in turn supports change management where the effect of a change is more easily started and the amount of work is more easily estimated. When we need to compose a certain service, we need first to discover the set of existing services to decide what and how a service can be used. By establishing service layers, it becomes easier to search for services based on the layers since the characteristics of each layer is standardized. And of course, governance is a prime beneficiary from having service layers because it becomes easier to follow policies, principles, and standards.

Still Skeptical?

At the start of the course, I promised you that I will teach you everything in the context of real world challenges and I'm keeping my promise. What you have learned so far about service types is essential so that you know the concepts. But later in this course, I will take you on a long journey on how to identify these services, how to classify them, and what kind of design decisions you have to take. All this in the context of the real world business case we're covering in this course.

Case Study: Applying Service Layers to Reference Architecture

In the previous module, you saw an illustration of how a SOA reference architecture would fit as a solution to our business case. Now having a better understanding about service layers, let's see a further detailed illustration about the SOA reference architecture. The consumer application presents a business service to a student to change current major. The student logs into the application and consumes the business service. The business service is actually a business process service, which the application calls, for example, a business process service called major

exposing a capability called change. A business process service wraps a business process, which it starts execution when the change capability is called. First, a manual task is performed and then an automated business task must be executed to check if the student is eligible to change major. The business task calls a capability called a check of a task service called eligibility. So again, why invoke a task service instead of writing the logic directly inside the business task? As explained, there are various reasons. First, because a service can be reused across different processes, instead of duplicating functionality. Second, because this provides a central access point to this business logic. If it changes, you change it once. And third, to loosely couple the business task from the composition logic. A business task just calls the task service and gets back a business meaningful data. If the composition logic changes, the business process itself is not affected. Now the task service needs to compose multiple services to perform its logic. For example, it calls an entity service to load student information. It also calls an entity service to get major history and it calls a utility service to log the action. Remember that entity services provide a central access point to business entities, which means we do not have redundant functionality and we have a single source of data access. While utility services provide usable crosscutting concerns, so we write security and other functions just once. Entity services, the newest integration services to talk to existing systems in order to load their required information. Doing so means that entity services and business services in general are loosely coupled from technology concerns and loosely coupled from how data is physically stored in systems. Once again, because business processes are being able to automate business tasks from a pool of reusable services in the service inventory and because multiple business processes are being able to reuse services without having to ask for new functions, this works towards an agile enterprise.

Summary

The service layer is at the heart of SOA and that is why we dedicated a complete module talking about it. Inside the service layer, services are classified based on their functional context and while there are different ways to classify services, however as owned, you must have business services that present the business functionalities required and technical services that loosely couple business services from existing systems. Process services are business services. A process service encapsulates an entire business process and is consumed by channels, such as web applications and mobile apps. Task services are also business services. A task service automates a certain business task and typically composes other services and includes whatever business logic required for task automation. A task service is usually not highly used because its functional scope is typically typed to a specific business process. Entity services are business services that have functional scope around business entities. As such, they are the most reusable services since they are independent of any specific business process. Entity services are mostly composed by task services. Utility services are technical in nature. They provide the crosscutting concerns such as security and logging, and as such can be composed by both entity and task services. And finally, integration services will write an abstraction layer on top of existing systems. Business services then use this layer instead of directly accessing these

systems, although adding integration services increases the number of services to manage, but this is a penalty that must be paid in order to preserve the loose coupling of business services, which is critical to achieving the business objectives of SOA.

Service Granularity

Introduction

Welcome to the module, Service Granularity. You are watching Real World SOA: Analysis and Design. So far we have discussed in detail SOA building blocks in terms of reference architecture and we zoomed in into the service layers and examined the different types of services. Before wrapping up the what aspect, we need to examine an important topic and that is service granularity. Granularity is one of the topics that causes confusion for service developers, but as we will see in this module it is a very important topic to be understood.

What Is Granularity?

Service and granularity is not a SOA-specific concern. It is a service design concern in general; however, as you will see, it has special importance within a SOA solution. Service and granularity can be divided into two parts, functional granularity and data granularity. Functional granularity indicates the scope of functionality exposed by a service. The wider the scope is, the more granular it is and the service is typically referred to as a coarse-grained. On the opposite end, the service with another function and scope is typically referred to as a fine-grained service. Data granularity indicates the amount of information exchanged by service capabilities, a capability that exchanges a lot of information is set to have coarse data granularity, while a capability that exchanges small amount of information is set to have fine data granularity. Now let's examine a simple example to understand granularity. Consider a student service with a GetStudentInfo capability. This is an entity service with a fine-grained functionality that is reusable across multiple business processes. For example, a business process which enrolls a student in a major, a business process which allows a student to change major, and a business process, which allows a student to ask for bill compensation. All these processes require at a certain stage the functionality of floating the student profile, so this makes the GetStudentInfo capability highly reusable because it is independent of any specific process. On the other hand, consider a coarse-grained business process service called enrollment, which encapsulates the enrollment business process. This service exposes a capability called enroll. This capability encapsulates big chunk of functionality fulfilling the entire business process, so this is a very coarse-grained service. Now one of the tasks inside the business process wants to check for student's eligibility to change major. This is automated via an eligibility task service with a check capability. When this capability is called, it composes the GetStudentInfo capability to load data,

a major service to get major information, and a utility service to check a central business rules store for certain rules. This makes the task service also coarse-grained, but less than the business process service. So as you can see, granularity is the measure rather than an absolute value. Now let's see data granularity. Let's zoom in into the GetStudentInfo capability. If we make this capability return the entire student document including say the profile in addition to the academic information, this causes the capability to have coarse-grained data granularity. This increases the reuse potential of the capability because it can serve multiple compositions; however, because it exchanges a lot of information, performance will suffer. On the other hand, if we separate the GetStudentInfo capability into two, GetProfileInfo and GetAcademicInfo, returning profile information and academic information respectively, then we have created two capabilities with fine data granularity with less reuse potential for each. In addition, a consumer that needs both, the student and the academic information will now have to make two calls instead of one. So as you can see, the topic of granularity is not straightforward as it involves a lot of tradeoffs. In the remaining sections, we explore these tradeoffs in greater detail.

The Confusion Around Granularity

The number one source of confusion regarding granularity and you will find this a lot in SOA discussions is that services must be coarse-grained and that fine-grained services should not be part of SOA. People that say this say that coarse-grained services are high-level business services, something like change major, for example. While fine-grained services are chatty services that perform CRUD-like operations, such as insert or delete a profile. Well in fact, this is a massive over-simplification of the granularity design issue and this is not accurate for various reasons. Let us start with the most basic issue. SOA service inventory will actually contain services of all granularity levels. So for example, a business process service is typically coarse-grained because it traps an entire process, and therefore, offers a wide scope of functionality. A task service is less granular than the business process service, but it is also typically coarse-grained as it automates a usually complex business task. An entity service; however, with a functional scope of a class entity, for example, can be considered fine-grained, while most of the technical services are fine-grained as they perform specific functions of existing systems. So the point is that it's simply not true that SOA is all about coarse-grained services. SOA is composed of services of varying granularity levels and all of these services are important. Second, the more coarse-grained a service is, the less flexible it is. A service with coarse functional granularity is more subjective to changes and its wide business logic, yet because it is so large, it is not flexible to change. A very important consideration is reusability. Services with more functional granularity are less reusable. The more functionality a service provides, the more it is tied to a specific business process. For example, a business process service encapsulates an entire process, which makes it effectively not reusable. While a task service automates a business task within a business process, it is still coarse-grained, less than the business process service, but also mostly not reusable. On the other hand, the more fine-grained entity service, which deals with a specific entity is more reusable because it encapsulates less amount business logic. While a fine-grained logging service,

which simply writes a log record into the database is extremely reusable. On the other hand, the logic works in an opposite way when it comes to data granularity. The more data a service exchanges, the wider the range of consumers it can serve, so it becomes more reusable. And the more fine-grained the amount of data it changes, the more limited set of consumer base. In this section, we have discussed the concept of granularity. In the upcoming sections, you will see examples, which will make things more clear, but the key takeaway is that granularity is a design decision that is subjective to design tradeoffs and it's not simply a matter of coarse-grained being good and fine-grained being bad. In the following sections, you will understand these tradeoffs and you will understand how to design the granularity of your service according to these tradeoffs.

Functional Granularity Design Tradeoffs

Let's now consider a set of design tradeoffs related to functional granularity. The more fine-grained services are, the better they were before because simply they have less work to do, so why just not create a lot of fine-grained services? Well because each of the fine-grained services by itself do just a little bit of work, so this means that a consumer will need to make a lot of calls to achieve a certain functionality. And as you can imagine, a lot of calls negatively affects performance, not to mention, of course, increasing the possibility of connectivity issues. Also, chances are that fine-grained services will find themselves as part of a distributed transaction, in which case, a distributed transaction manager, such as DTC is likely to be used, which also negatively affects performance. And what about manageability. It's clear that as the number of services grow that manageability and maintainability become harder. However, on the positive side, as explained before, because fine-grained services encapsulate limited functionality, they tend to be business process or business logic independent, and therefore, they have higher reuse potentially across different business processes. Also because they encapsulate less functionality, this means they are less likely to change and are actually more flexible to change. So for example, a student entity service is highly reusable since it's not dependent on specific business logic or business process, and at the same time, it is much less likely to change than the business logic, which uses it, such as a checking for eligibility to change major, for example, which would actually be a coarse-grained task service. Now let's look at coarse-grained services. The more coarse-grained a service is means that it encapsulates more functionality, which means it's less reusable across different business processes because it's likely bounded to a specific business process. Also, because the encapsulated functionality is large, then the service is more prone for changes, yet it is less flexible to change. On the other hand, because coarse-grained services encapsulate more functionality, this means consumers need less invocation calls to achieve the required functionality, which results in better performance. So as you can see once again, design is all about tradeoffs. Beside your expert judgment, there is no other way to tell what is the best granularity for a service. The service layer is part of the reference architecture, give you starting points and guidelines, but how you classify services, and consequently, want the granularity you ___ to them is driven only by experience.

Functional Granularity and Functional Context

As I explained in the fundamentals module, functional context specifies the purpose of a service, in other words, the functionality that this service intends to deliver. Recall that each service should have a distinct functional context that is made up of the set of capabilities that the service exposes. Now there is a direct correlation between a functional context and a service functional granularity. A service granularity refers to the scope of functional context exposed by a service. The higher the granularity is, the larger the functional scope of these capabilities combined. The lower the granularity is, the smaller the functional scope of these capabilities combined.

Data Granularity Design Tradeoffs

As explained before, data granularity is the measure of the amount of data exchanged in a single service interaction. Like functional granularity, design tradeoffs must also be applied. For example, back to the student enrollment example, which we discussed in the functional granularity section, the eligibility task service has one capability called check. Typically, such capability would accept a StudentId and a MajorId and then performs the required composition and business logic to check for students eligibility to change major. The return is most of the times some kind of acknowledgement that the business logic has been executed. So its functional granularity is coarse-grained, but it's data granularity is fine-grained. However, the student entity service, which is composed by the task service, typically has a lower functional granularity since its functional scope is around the student business entity only. However, the data granularity varies and must be balanced. For example, shall we return the entire student profile including the personal information and TrackHistory? Doing so leads to a coarse-grained data granularity and the advantage is that we're loading all the data required once, so we reduce the number of calls, but its advantages are that we're utilizing more bandwidth by increasing the message size and we are putting on the composing service the burden of managing the extra information until they become useful for its processing. The other design option is to divide the capability into two different capabilities. The first one returns only the student call profile information and thus lowering data granularity of the capability and then exposing another capability with fine-grained data granularity that returns track history. This form of lazy loading would mean more calls required from the composing task service, but on the other hand, it requires less bandwidth than a call and relieves the composing service from the burden of managing data until it really needs it. So again, as you can see, there is no such thing as an absolute good or bad granularity. Tradeoffs must be applied and your expert judgment plays the key role in deciding the most suitable granularity given each situation.

Granularity as a Relative Measure

As I explained before, the granularity is a relative measure, so I do not believe it's best to ask if a service is fine or coarse-grained, although, you'll find this question commonly asked in SOA discussions. The better question is the

level of granularity of a service. In that sense, granularity can be seen as a horizontal access with less granular services positioned to the left and more granular services are positioned more to the right. The more fine-grained services we have, services are more reusable across different processes because they have a limited functional scope and the services naturally are more flexible to change. On the other hand, we need more services to cover a certain functional scope, and therefore, we have a higher number of services to manage. Also, since functionality is spread across various services, this means that we need more composition calls to fulfill a certain function. The more coarse-grained services are, the more functional scope they encapsulate, so we have a lower number of services to manage and compositions become faster since we require a less number of calls to achieve a certain function. On the other hand, as the functional scope is larger, services become less reusable and less flexible to change.

Case Study: What Role Does Granularity Play?

The value of understanding granularity becomes apparent when we start identifying services and continues to play a major role as we implement design principles and tradeoffs later in the service design phase. During service identification, we use our understanding of granularity to decide how we are going to distribute our functional scope across services and how we are going to group capabilities into services. We saw various examples in this module that show you how granularity considerations shape the way we classify services, so please make sure to understand these examples because they resemble the real-world challenges that you will face in your SOA implementations. In addition, later during the service design phase, granularity consideration is a guiding factor and the tradeoff analysis of design principles. Service design principles, their application, and tradeoffs are discussed in detail in the next course.

Summary

Granularity is a design topic that is often misunderstood or misjudged. Functional granularity refers to the amount of functionality encapsulated in a service, while data granularity, the first of the amount of data exchanged in any service invocation. In both cases, think of granularity as a measure, rather than an absolute value. Services with high granularity are typically called coarse-grained services, while services with low granularity are typically called fine-grained. Our service inventory will hold services of all ranges of granularity. Design tradeoffs should be applied using expert judgment to decide what is the most suitable granularity for any given situation.

Pause and Recap - Part 2

Pause and Recap

Just as we did at the end of the why aspect, let's again take a breather and have a quick recap of what we've covered and a look at what lies ahead. You're watching Real World SOA: Analysis and Design. Just like any architecture, SOA is made up of a set of building blocks. Before you take any step forward, you must have a solid understanding of these building blocks, how they are structured, and how they interact with each other. SOA building blocks cover more than just services and you need to understand the role of each of these blocks. Services are of course the core building block and services are classified into different layers. Each layer plays a different role. In the previous three modules, we covered all about SOA building blocks, how they interact with each other, and the different classifications of services. In addition, because it's an important design topic that often causes confusion, we allocated dedicated space to discuss service granularity. Having reached so far, you now have the required knowledge to set up a SOA reference architecture that fits your organization needs. Next, we start a discussion about the how aspect of SOA armed with our understanding of the why and what aspects. Next, we learn a step by step methodology of how to apply SOA to our business case. We start with business architecture and explain why it plays such an important role in SOA, we then discuss a methodology to identify and classify services, and then we see how to design an information model to guarantee the semantic interoperability of the identified services.

Creating the Business Architecture

Introduction

Welcome to module, Creating the Business Architecture. You are watching Real World SOA: Analysis and Design. In this module, we start our discussion about the how aspect. As explained in the previous modules, creating the business aligned SOA services requires the understanding of the business. Business architecture is an extremely important discipline that plays a critical role in business IT alignment. In this module, we have an overview about business architecture and we understand its importance to SOA.

Why Understanding the Business Is a Must?

As explained before, SOA is an IT architecture that is driven by strategic objectives. Business IT alignment is a big field of study and you will get to know more about it in this module. But in the simplest of terms, for SOA business IT alignment means that the services that we identify and design are driven by business needs and play an important role in helping organizations achieve their business objectives. In our case, the organization we are studying is looking to achieve agility, cost reduction, and customer centricity. For our SOA effort to be successful, services must help our

organization achieve these goals. In other words, our SOA must be a step towards the business IT alignment of the organization. So how is such alignment achieved and how do we establish a correct path for identifying business aligned services. Here comes the role of the business architecture discipline. In the next sections, you'll understand what role this discipline plays and how it helps creating business align services.

Basics: From Goals to Strategy to Implementation

So understanding business is mandatory to SOA, therefore, this section gives a brief overview of business concepts. It is by no means a comprehensive discussion. Just enough and know that to understand where SOA fits in within the big picture. So organizations set goals that they want to achieve, goals are achieved by following a business strategy. Strategy is defined as the creation of a unique and valuable position involving a different set of activities. Now part of defining a strategy is creating a business model, which describes the rational of how an organization creates, delivers, and captures value. So a business model is basically a layout of the activities required as part of the strategy. Now let's zoom in into the business model. A business model is made up of interconnected elements. These elements should define the value proposition that is what unique value the enterprise provides to customers, how the enterprise makes profits, in other words, how it generates its revenue stream. What are the key resources required to deliver value and the generic profits, and what are the key processes that should be in place to deliver value and generic profit. These elements are the core building blocks of the business and together they create and deliver value, so a business model is defined at high level during strategy creation. Now in order to operationalize this business model, in other words, to make it executable, you need a framework for execution. This framework is called an Operating Model. This operating model specifies what the organization must possess in order to execute this strategy. That is the processes, the integration and standardization of this processes, IT systems supporting the processes, governance, people, and the entire business ecosystem in terms of partners and suppliers. So this should be starting to ring bells now. Can you start seeing the resemblance to the SOA building blocks we talked about? Well if you have been following closely, that should be the case. However, we're not there just yet, so let's keep going. The next critical question is how to move from the strategy and its business model to the implementation of the operating model without risking investing in projects and services that are not aligned with this strategy, or projects and services that are in conflict with each other, or their outputs are in conflict or redundant with existing assets. We need some discipline that guides this transition. We simply cannot afford to leave this transition without a discipline that provides a holistic view of every aspect of the enterprise and a discipline that makes sure that whatever we're building is a value to the business and is in coherence with all of the organization's building blocks. This discipline is the enterprise architecture and it's a key discipline for business IT alignment. So where does SOA fit within this structure? Let's examine this in the next section.

Enterprise and Business Architecture: Where SOA Fits In?

So enterprise architecture is the discipline that operationalize the business model. It guides the transition from the strategy and business model to the operating model. Therefore, enterprise architecture plays a critical role in ensuring business IT alignment. Now enterprise architecture in turn is divided into different domains, typically, business architecture, application or solution architecture, information architecture, and technology architecture. Collectively, application architecture, information architecture, and technology architectures are called IT architectures. I will talk more about business architecture in a moment, but in a nutshell, it includes all the business building blocks, their structure, and their dynamics, while the IT architecture includes all the information systems and their supporting infrastructure structure. So it is clear now why enterprise architecture is the discipline we're seeking to help us transition from strategy to operations. It is clearly one that provides a disciplined approach to transition from business to IT, more specifically, from business architecture to IT architecture. So again, where does SOA fit in? Well remember that SOA is an IT architecture, which includes services, information, and technology. Therefore, we each know the result that we started with at the very beginning of this course that understanding an organization's business architecture is a prerequisite for creating IT architectures that are aligned with the strategy. SOA is one of these IT architectures. So since business architecture is that much important, in the next section, let's take an extended look at it.

Extended Look at Business Architecture

So we established that business architecture is critical to the alignment of business and IT. Within the context of enterprise architecture, business architecture is the discipline that guides the transition to IT solutions. Without proper understanding of the business building blocks, their structure, and their dynamics, we cannot build IT solutions that act as enablers for the business. So what is the definition of business architecture? IEEE has a widely adopted definition of software architecture as the fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution. Now by applying the same definition on business architecture, by replacing system with business, we find that business architecture refers to a set of core business building blocks, their structure, their dynamics, and the governing principles. So if the well-known enterprise architecture framework defines business architecture as a description of the structure and interaction between the business strategy, the organization, the functions, the business processes, and the information needs where strategy, organization, function, business processes, and information needs all refer to the business building blocks I was talking about. So in summary, a business architecture is a blueprint of the business building blocks clarifying the structure and the dynamics of these blocks. The strategy, objectives, internal organization, business principles, value streams, business capabilities, business processes, business entities, business services, business rules, and the business ecosystem are all common core building blocks of the business modeled as part of the business architecture blueprint views. Of course, business architecture is a huge field of study worthy of books by itself, but for the sake of our SOA discussion, we will examine briefly value

streams, capabilities, and business processes all required to derive SOA services, while business entities drive the semantic information model used to design documents exchanged by SOA business services. And business services essentially map to the business process services with SOA in the SOA reference architecture. So now that you have a high-level understanding of business architecture, the next essential question becomes how to derive SOA services from business architecture building blocks, especially using value streams, capabilities, and business processes. Let's find out in the next section.

From Business Architecture to SOA: Overview

So how can we use business architecture to derive the required SOA services? One way, which is gaining momentum is capability analysis. In this section, let's have an overview of the entire process before we go into further details in the next sections. So our end target is to derive our SOA services. As we made it clear in the SOA reference architecture, for this we need to have the business processes that SOA services attempt to automate and increase their flexibility. Yet, at this stage, we're still at the point very high level that we cannot simply identify the business processes. We will use capability analysis and provide a capability map. This map is at the high abstraction level that dissects the business in terms of business capabilities, regardless of how these capabilities are fulfilled, which is the job of the business processes. Now to derive the capability map itself, we need to know the values that our business delivers, after all, without knowing what values we are providing, how can we tell what capabilities we need to possess in order to deliver these values. So value stream analysis will be used as the highest business abstraction level. From the value streams we will derive the capabilities.

Introduction to Value Streams

Value streams are one of the viewpoints of business architecture. They represent an end-to-end high level process that generates value for certain stakeholder. Now, the phrase end-to-end means that value streams do not focus on how work is actually done. This is left to business processes. Rather, they focus on the high level stages that deliver value. Stages in a value stream are arranged in a timely fashion resembling the way a scenario takes place, but without showing how work is actually implemented. As an example, let's consider the insurance business domain. A claim value stream would contain three stages, a customer registers, submits a claim, and collects money. This value stream shows the high level end-to-end process, which provides value to customers in terms of collecting money. Now each stage within this value stream is implemented with business processes and possibly sub-processes detailing the actual activities that take place. This is mainly the difference between value streams and business processes. Value streams focus on the value delivering stages, not the actual activities. So as you can see, there is a mapping between value streams and business processes, yet doing this mapping directly is not easy because value streams are at the high abstraction level, while business processes are at the execution level. So after modeling the

value stream, you ask a key question, what capabilities the organization must possess in order to deliver the value promised for each of the value stream stages. In other words, I'm the insurance company and I want to deliver the claim value stream. Now what should I own in terms of people, systems, IT, processes, and so on, in order to make this happen. This brings us to the capability map, which we will examine in the next section.

Introduction to Capability Maps

The capability is anything that an organization may possess that helps this organization achieve a certain outcome. This capability describes what the organization owns that help this organization achieve outcomes, not how these outcomes are achieved. A capability abstracts things such as people, processes, and procedures, technology, and information. For example, marketing is a capability that encapsulates things such as marketing people, marketing processes, IT communication systems, and marketing information. All this helps the organization reduce marketing campaigns. Similarly, customer management is a capability that encapsulates processes for customer management, IT systems, and customer information. All this allows an organization to work on customer satisfaction. Now the most important thing about capabilities is that they allow us to model the business using its most stable elements. The how of any business changes frequently in response to business requirements, events, and opportunities. The what aspect of the business, however, rarely changes. So for example, how we do marketing campaigns, the skills required, and the supporting IT systems frequently change in response to new methods and technologies available, but the fact that a marketing capability should exist is almost always true. Capabilities are modeled in a capability map, which is a hierarchical description of what the business does. We usually create a three-level hierarchy of capabilities, but that could change based on the complexity of the business. Level 1 starts at the highest level of abstraction and then decomposed into lower levels until we reach a stage where the next level is the start of business process modeling. For example, consider again the insurance domain. Claim processing is a capability, which is called to value creation because our insurance company needs to possess the ability to issue claims, yet claim processing is at the high abstraction level such that it is difficult to derive the business processes from it directly. In capability modeling, claim processing is modeled as a level 1 capability. Now further analysis shows that this capability can be broken into claim validation and payment capabilities at level 2. Validation can further be broken into level 3 capabilities, account validation and coverage validation. At this stage, we find out that we're at enough granular level to start deriving business processes that fulfill these capabilities. Now having this business process defined, deriving SOA services then becomes possible. This way we can trace each service back to the business process that uses it, and consequently, to the set of business capabilities that this service helps enable, and therefore, achieving the business IT alignment.

Value Streams to Capabilities

So now that you understand value streams and capabilities, let's see how they work together. What you are seeing here is the result of the case study capability analysis that we will perform in the next module. For now, let's just have a quick look. So we start with a set of value streams showing the business value creation at the highest abstraction level. Notice that the value streams themselves are classified in the hierarchy. So for example, this value stream here is a top level stream while this stream is derived from the status stage of the top level stream, while this stream is derived from the gain experience stage of the top level stream. Next, for each of the value stream stages, we ask a fundamental question. What capabilities do we, as an organization, must own in order to deliver the values classified by this stage. Since the identifying Level 1 capabilities, for example, 3 Level 1 capabilities are required to deliver the values in the top level value stream. We then start to compose a Level 1 capabilities into Level 2 capabilities using the other value streams. So for example, these 3 Level 2 capabilities are required to deliver the values of this value stream, while these 3 Level 2 capabilities are required to deliver the value of this value stream, and we continue this to composition until we reach typically Level 3 capabilities. Now after Level 3, we ask another fundamental question. How will each capability be fulfilled? This derives the set of business processes that implement the capabilities. It's these business processes that will form the basis of the service identification.

The Essence of SOA: Business Traceability

This entire traceability that we've established provides the essential business IT alignment, or in this case, the business SOA alignment. This traceability allows you to trace SOA services back to the capabilities that this service has helped enable. And consequently, to the values that these services help achieve. It's only then that you will be sure that your SOA services are business aligned simply because each service plays a role in delivering business value and this is the most fundamental difference between a SOA-based approach for building services versus a non-SOA approach. At this stage, the entire talk about the role of business in SOA becomes apparent.

Prioritize and Iterate? Capability Heat Map

Recall that working on our creations based on priorities is an essential element of SOA planning. Without it, you fail to deliver the most important value in the fastest possible time. Capability maps helps you set priorities. By looking at the map, you now have an entire representation of your business capabilities. By proper analysis, you identify the most important business capabilities that you need to start with. This will allow you to achieve the most important value in the fastest time possible based on your business objectives. So you end up with something called a heat map where the most important business capabilities are marked and these become the basis of planning for your SOA work. In other words, service is derived from these capabilities and business processes are the highest priority services that you will start with.

But Business Architecture Is Not My Job!

So you have come so far, and hopefully, now you understand the role of business and the value of understanding business for SOA. But there is a big chance you still have a question in your mind. You might ask, what if I work in an organization that has all this business stuff worked out and the business processes, which eventually are the input to SOA, are properly identified and designed. Then in this case, why would I care about business architecture and why would I care about understanding concepts, such as value streams and capability maps. Well to answer this, let's take a simplistic look at what are the possible types of organizations you could be working for. You could be working in an organization that has no business architecture defined and has no enterprise architecture or business architecture disciplines. In this case, as a SOA architecture, you have to help derive the business architecture as an input to SOA. So you need to have a solid understanding of the topic. This is essentially the approach I followed in this course. Or your organization could have business architecture already defined, or at least, recently established an enterprise architecture or business architecture discipline who is assigned to create the business architecture. In this case, as a SOA architect, you have to talk the same language of the business architecture and you must understand what portion of the entire organization's business architecture you will use depending on the strategic objectives you are trying to fulfill with SOA. So the key takeaway is that regardless of whether business architecture is defined at your organization or not, you must trace your SOA services back to strategic objectives and business values. Unless you do that, you're not doing SOA. And the only way you can establish this traceability is using business architecture. So even if business architecture is already defined, you must know how to use and interpret it in order to build services that is aligned with it and that is why understanding business architecture is important.

Summary: Overall Traceability View

Because we've covered a lot of ground in this module, let's wrap things with this summarized illustration. So business goals drive the business strategies selection. Part of this strategy selection is the definition of the business model, but a business model is set at a high-level. An operating model provides a framework of execution for the business model. It defines processes, IT systems, governance, people, and the business ecosystem that is required to execute the business model. Enterprise architecture is the discipline that provides a controlled transition from the business model to the operating model. Enterprise architecture is made up for business architecture domain and an IT architecture domain. Business architecture always precedes IT architecture and is essential in achieving business IT alignment. Various business architecture building blocks guide this IT transformation. Value streams allow us to look at the business from the highest abstraction point in terms of value creation regardless of what are the needs to provide this value. These needs are made clear using capabilities, which specify what an organization needs to own in order to achieve the value stream values. Finally, the how aspect is covered in the business processes, which specify how the capabilities are fulfilled. SOA is an IT architecture that is guided by these business architecture building

blocks. So by effectively following this framework, SOA is guaranteed to provide the business value that it is supposed to deliver.

Case Study: Creating the Business Architecture

Introduction

Welcome to the module, Case Study: Creating the Business Architecture. You are watching Real World SOA: Analysis and Design. In the previous module, you learned about the important concepts of business architecture. You learned that business architecture is the only way you can link SOA to business objectives and show how SOA provides business value. In this module, let's apply these concepts in our case study.

Step 1: Articulating the Business Goals

As we have discussed in detail, every IT architecture we do should be to help an organization implement its strategy and achieve its goals. So the start of any IT effort is to write down the business goals that you want to achieve. These goals have been discussed in detail at the case study module along with the motivations behind them. So before moving on, please make sure to review the business case if you need to refresh your information. Here, let's just restate these calls. The agency wants to be agile in response to new demands. It wants to improve the flexibility of business processes so that changes and the new requirements are applied in a timely manner and the time to market is decreased. The agency also wants to decrease the IT development and operation costs. It wants less development time and effort to be put to respond to changes and less time and effort to operate existing systems and data in response to changes. The agency wants to create a customer centric enterprise so that customers must not feel as if they are dealing with different independent entities. The customer must have a single unified experience throughout the interaction with the agency regardless of which unit is offering the service. And finally, a goal driven by new emersion business need is to extend the ecosystem of the agency by allowing companies, which offer internships and jobs to be able to post and edit information about vacancies. In addition, these companies can ask for a certain set of skills and decide to offer graduates that fit their needs an opportunity for internship and possibly later a job.

Step 2: Modeling the Value Streams

So as explained in the previous module, our starting point is the value stream analysis in order to model the business at the highest abstraction level. In this top-down analysis, we start with the fundamental question, what values do we want to provide to our customers? The first essential top-level value is to provide our customers the ability to study overseas. This results in a value stream with stages study, gain experience, and inquire. This becomes our top-level value stream. Now let's briefly examine the values generated by each one of the value stream stages. First, we want to manage the scholarship of a student. We want to provide students with the value of helping them achieve and get a successful study experience. We also want to help graduates find a training or a job, so the value we want to provide here is to continue supporting graduates and help them start their practical experience. During this entire time, we want to continuously support student requests or might have inquiries or complaints about the services we provide. One level below in our value analysis, under the study stage, we identify another value stream with the three stages. Students apply to gain a scholarship. Their application must be processed and any inquiries while the application is in the pending state must be managed. Scholarship students then must be able to submit various requests related to the scholarship throughout their study lifecycle, for example, requests related to changing a major or stopping a scholarship. Scholarship students also require to have the financial issues handled for them. Compensations and payments are examples. And finally, on the same level, under the gain experience stage, we identify another value stream with three stages. Graduates need to apply for training and job opportunities in companies. They need to work as interns in these companies and then they need to find jobs in the market. So this concludes the simplified value stream model. Next, we start deriving the capabilities that are required by each of the value stream stages.

Step 3: Mapping Value Streams to Capabilities

Having created the value streams, we now ask the question, what capabilities must the organization possess in order to provide the values as created with each of the values stream stages. To answer this question, we start our exercise, which maps value streams to capabilities. The top-level value stream requires that the organization possess a set of level one capabilities. A scholarship capability, which provides the ability of managing a student entire study lifecycle. A graduation capability, which provides the ability of helping a graduate land a training or a job. The marketing capability, which allows the organization to reach the students and graduates and promote new and improved services. A customer management capability, which allows students to inquire about services, ask for new services, and complain about existing services. And just like that, we continue our mapping to find level one capabilities. For example, we ended up with a document management capability as we found out that we need to provide students and graduates the ability to upload, review, combine, and edit personal day-to-day documents. We also decided to include enterprise architecture itself as a level one capability. These types of capabilities do not provide direct value to customers, rather they are enablers for other capabilities. IT management is also such type of capability. Next, we start the composing level one capabilities into a level two capabilities. In this exercise, we will only to compose the scholarship and graduation capabilities as they are the ones relevant to our business case. So we again we use value

stream mapping to help us decide on the level two capabilities required. The apply value stream stage requires having an application capability, which allows the students to apply for scholarship. The manage study stage requires having a study capability, which manages the study lifecycle of a student. And the manage finance stage requires having a finance capability, which manages all the financial aspects of a scholarship. And then we do the same one to composing the graduation capability into level two capabilities. In summary, we need the agencies capability to manage the registering, new hiring, and the training institutes. We need to have an internship capability to manage how graduates become interns. And similarly, we need to have recruitment capability to manage how graduates find a job. At this stage, we have a capability map of two levels. In the next section, we complete the map by decomposing into a third level.

Step 4: Complete the Capability Map

So we now need to find a level of capability to composition before discovering the business processes that fulfill these capabilities. Recall that capability to composition stops when we discover capabilities at the right granular level that is suitable for deriving business processes. The application capability is to compose into a submission capability to manage scholarship applications, submission, and inspection. The processing capability to manage processing the application for approval and the candidate capability to manage applicant requests pending application approval. The study capability is the composed into an annexation capability to allow students to add companions to their scholarships. And then academic life capability to manage all core aspects of the scholarship. And finally, the finance capability is the compose into a rewards capability to manage financial compensation of rewards. And then invoices capability to manage financial compensation of invoices. Now upon analysis, we find that the level two decomposition of the graduation capability is enough for deriving business processes. Therefore, there is no need for further decomposition. So as you can see, we can actually end up with different levels of the composition in the same map. At the end, the important thing is that for each of the final level capabilities, we'd add the suitable granularity level to derive business processes. This then concludes our entire capability map, but SOA work must be scoped based on business priorities. And now that we have the entire business capabilities modeled in front of us, scoping becomes easier. Let's see how in the next section.

Step 5: Set Your Scope by Creating the Heat Map

Now that we have the capability map or at least the portion that is relevant to this course, we need to identify the topmost priority capabilities that we need to start with based on business objectives and impact. So as we start analyzing the capability map, we conclude that the submission capability is of high importance since many students apply each day. The rest of the capabilities under the application capability are less used and no frequent changes are asked for. On the other hand, the study capabilities are frequently used and most the changes and additions

come in this area. Therefore, the flexibility of the study-related business processes is of highest priority. Also, we found out that the finance capability is also widely used as students ask for compensations and it's frequent now that new business processes are being asked for because new types of compensations are being encountered. The IT systems supporting the study and finance capabilities suffer from redundant functionalities and data, inconsistent data, and data of different models, All of which result in high development and operation costs and severely impacting flexibility and agility. As a result of this analysis, we decide that the submission, study, and finance capabilities are the ones we're going to include in our first service identification iteration. But wait a minute. What about the graduation capability? We call that extending the ecosystem by allowing companies to post vacancies and ask for skills as a new business objective. So unlike the other capabilities, here we're not talking about something that already existed that we want to enhance, rather we're talking about introducing a brand new capability. This capability is decided to be part of iteration two and I will talk about it in the next course as I will use it as a case for my S base service. For the remainder of this course, let's focus on iteration one, which includes the submission, study, and finance capabilities.

Step 6: Map Capabilities to Business Processes

Next we move to the question, what business processes are required to fulfill the capabilities? In other words, how are capabilities implemented by business processes? For example, the submission capability is implemented by an apply and inspect processes. Similarly, the processing capability is implemented by an interview and complain processes. While study and finance capabilities are implemented by various processes, such as add companion, initiate study, change major, compensate reward, and so on. Based on our heat map, which we identified in the previous step, our scope of work for iteration one is submission, study, and finance, so we will select three processes to base our analysis and design work on and these are the apply, change major, and compensate reward processes. There is no particular reason I selected these processes. It's just to limit this scope of discussion.

A Word about Business Processes, BPM, and SOA

Business Process Management is a discipline that includes an approach or methodology and a set of supporting tools that collectively allows an organization to identify, enhance, model, develop, automate, and manage the business processes in support of strategic goals. BPM is a big discipline and includes topics such as business analysis, strategy alignment, business process improvement, business modelling, and business process monitoring and management. So on this study, we care only about the relation between BPM and SOA. Any business has business processes no matter what the business domain is. In the banking domain, for example, issuing credit cards, approving loans, and transferring money are business processes required to run the bank. And the insurance domain, processing a claim, and issuing a new insurance are processes required to run the business. And as we have seen here, the government

sector, business processes are required to provide public services to target audience. Even a business as simple as a grocery store has business processes to order and ship goods. The existence of a BPM discipline means that these processes are analyzed, improved, modeled, automated, monitored, and managed to make the business run in a more effective and efficient manner. So the existence of BPM is a great aid for SOA because then you know that the business processes you are using to identify services are themselves driven by a business strategy properly modeled highlighting automation possibilities and are orchestrated using a BPM engine. However, not all organizations have a BPM discipline. So what if you find yourself working on a SOA project in an organization that does not have a BPM discipline in place and in which the business processes are not already analyzed and modeled. Well in this case, you have to make this activity part of your project. At the end, without business processes, just like the rest of the business architecture, you cannot derive business online services.

Step 7: Start Service Identification

At this stage, we have completed our business architecture, and now we're ready to start identifying services and their interactions. This is discussed in detail in the next modules.

Identify Services

Introduction

Welcome to the module, Identify Services. You are watching Real World SOA: Analysis and Design. In the previous two modules, we understood what is business architecture and we derived the business architecture for our case study that is required to start service identification. In this brief module, we cover essentials for service identification, and in the next module, we start the actual process.

Service Identification

So now we have the business processes and we have an empty service inventory. Service identification is actually composed of a set of interrelated activities. We identify services, we then classify them based on their functional context and granularity considerations, topics which were covered in detail in the previous modules, and then we decide on the capabilities that they will hold, in other words, we group capabilities into services. These services will then be part of our service inventory. That being said, there are two important points to keep in mind regarding service identification. First of all, service inventory will be refined over time. During analysis, services are identified in iterations, which means a service identified, say in iteration one, could be merged with another service in a later

iteration. And of course, the new capabilities could be added to an existing service. All capabilities could be merged in later iterations. In summary, SOA, just like any other architecture, becomes more clear with time and refinement via iterations is inevitable. The other important point is that the services you identify during analysis are at the business level without any regard to implementation mediums. What I mean here is that at this stage, we do not talk about SOAP-based services or web APIs. For example, the SOAP-based approach models services in the more traditional way of a service and the capabilities as a set of operations, while the API-based approach models services in terms of resources and capabilities in terms of actions on these resources. This topic will be examined in the next course as we dive into service design aspects. For now, during analysis, this is not our concern. One final thing I want to point out before moving on is that in this course, we're going to identify business process services, task services, entity services, and some of the utility services. In the next course, we identify the integration services because they are considered more into the detail design site since they deal with systems and technologies.

Top-down vs. Bottom-up Analysis

There are two service identification methods, top-down and bottom-up. Top-down is the method that we've been actually promoting and using this entire course. In this method, we start with the business objectives, strategy, business architecture, and then identify services. As explained in detail, it's the approach that results in the business align services. The bottom-up analysis method is about identifying services based on the analysis of existing systems. Following this method, you identify a need for a certain functionality, you then analyze existing systems, and you build a service that encapsulates this functionality. Integration services are derived using this method. So let's say that we identify a need for a business service to access a certain system. We analyze the existing systems, their functions, how these functions are distributed, the data models of these systems, and how data is distributed, and then based on this analysis, we create our integration services. You will see this method in detail in the next course, and you will see what kind of challenges you will face when analyzing your existing systems. For the remainder of this course, our focus will be the top-down approach.

Case Study: Identify Services

Introduction

Welcome to module, Case Study: Identify Services. You're watching Real World SOA: Analysis and Design. After all the preparation, let's finally get our hands dirty by identifying our SOA services. In this module, we'll examine three processes I selected out from the processes item defined as part of our capability map. Each process will be analyzed in a separate section and each section will show you a set of real life challenges typically faced during service

identification. Please note that the business processes are simplified to remove business-specific complexities, not relevant to our discussion. At the end, the purpose is not to let you master the business domain of the business case, rather of course, to learn SOA.

A Note About Capability Parameters

Before we start service identification, I want to highlight that as we identify services and their capabilities, the parameters we will use for these capabilities are at conceptual business level. At this stage of analysis and high-level design, we do not know yet what the actual parameters will be. We just have an idea from a high-level business perspective about what a capability needs in order to perform its functionality. In the next module when we talk about the information model, we will refine our capabilities and use the parameters as defined in the semantic information model. These will be the actual parameters for our capabilities.

'Apply' Business Process

We start with the apply business process, which is one of the processes required to fulfill the submission capability that we saw on the capability map. This entire process is orchestrated via business process management engine and it is exposed as a business process service to applicants via channels such as web and mobile applications. From the high-level business perspective, this business process service is the business service providing value to customers who are the scholarship applicants. In the first step, applicants use a web form to enter the information for a new scholarship. Of the requested information, the scholarship country and major must be selected, therefore, the form needs to preload these lookup values. So this drives our first entity services. A country entity service will get countries capability and a major entity service with a GetMajorsByCountry capability. This capability requires the country and it returns the majors available for this country. (Working) Next, the check eligibility business task is one that can be entirely automated. This task requires two business rules. First, that the applicant has not previously completed any scholarships, in other words, the applicant wasn't a previous student, and the other rule is checking that an applicant has no other existing running processes or requests, for example, the applicant doesn't have an existing application request. Now since we're talking about applicants, then we identify a new entity service called applicant. This service will provide a single point of access to applicant information. Before deciding how to implement the business rules, we have an interesting design decision to make. Shall we include a capability that checks eligibility inside the applicant entity service and then call this capability from the business task or shall we create a separate task service that checks eligibility. To take a decision, recall what we explained before that we want our entity services to be independent of any specific business process, so we ask ourselves, isn't checking eligibility a business process specific functionality or is it reusable across a business domain? In this case, the answer was that its business process specific and since we want out entity services to be independent of any specific business process, then we decided

to create a task service called eligibility. This service has then a capability called a check and it only needs the national Id of a student. Now to implement the first business rule of checking that the applicant has not completed previous scholarships. We decide to create a capability called has completed scholarships inside the applicant entity service. This is an example of a domain-specific capability, rather than a CRUD-like capability. (Typing) Now what about the second business rule checking if the applicant has no other running processes. Here we have to make another design decision. Should we include this logic as part of the eligibility task service or shall we create a dedicated service to perform this check and then we can call the service from the task service. Well the decision here depends on whether this piece of functionality is reusable or not. We have identified no need for reuse and we have no reason to think a reuse opportunity would rise in the future. So we decide to place this logic inside the check capability of the task service. As a hint, when we analyze the next process, a new criteria will make us revise this decision. So what we end up with is a service composition. When the business task is executed, the check capability of the eligibility task service is called. It has the logic to check if the applicant has running processes and it also calls the applicant entity service to check for completed scholarships. The result of the composition is then returned back to the business task. Next, the review application is a manual business task that is executed by employees. There is no automation here, so we're not interested in this task from SOA perspective. Next, the process a new student record task creates the scholarship record for the student. Creating the scholarship involves two actions, creating a scholarship record and adding the student's bank account information to be used later for compensations. So our first as a decision is to create a new entity service called scholarship with a capability that adds the scholarship record. Similarly, we create an entity service called AccountInfo with a capability that adds the account information for a specific scholarship. Now a typical design decision is to decide where to host the composition logic. The process new student record task is, remember, being automated via a business process management engine. So one option is to create the composition logic directly inside this task and make this task compose the entity services. The other option is to create a new task service, which performs the composition and calls the entity services. Well as explained before, coding the composition logic directly inside the business task means that we cannot reuse the logic, and more crucially, if the composition logic changes, then the business process itself has to be changed and recompiled. Therefore, a task service is the best decision. So we create a task service called StudentRecord and it holds a capability called Process. This capability then composes the two entity services. (Working) The final business task is issue travel insurance. After a student gets accepted in a scholarship, he or she must be issued a travel insurance before travelling to the destination country. Again, we want to automate this business task. The business logic of this business task is made up of the following steps. First, we will broadcast messages to a list of predefined travel insurance providers asking these providers to supply us with their prices. After getting back their responses, we will compare prices and select the best one. Then we will send a purchase order for the insurance provider whose price was selected. The insurance provider will then process our purchase order and reply to us at a later time, maybe after hours or even days. This means that, in this case, our entire business logic is long running and instead of the real-time request response model. And finally, after we get the response back from the provider, we attach the travel insurance

to the scholarship. So to create this logic, we identify the need of an entity service called Provider. This service contains two capabilities, one to get the price of an insurance provider as we supply the travel insurance information (Typing) and the other is to send a purchase order to the selected provider in order to issue a travel insurance for the students. Once there, the provider service will be talking to a third party, but we do not care because we leave that concern to the integration services layer. Now to add the travel insurance to the scholarship, there is no need to create a new service because the call that we already identified is scholarship service. So instead, we add another capability to this scholarship entity service to attach the travel insurance. This is also an example of a domain specific capability, rather than a CRUD-like capability. After we compose the whole logic together, we create a task service called travel insurance-- with a capability called issue-- which then calls the capabilities of the two entity services.

'Change Scholarship' Business Process

Now let's identify services for the change scholarship process. As we see in this section, as we analyze more processes, we get to expand our service inventory. We find the existing services and capabilities and refine previous design decisions. So once again, we have the entire business process exposed as a business process service, which is presented by front-end channels, such as web applications and mobile apps. Now here we have an early design decision to make. Why create a new business process service? Why not create a capability to change the scholarship inside the application business process service, which we identified in the previous process. Well the decision here is based on governance more than anything else. Recall from our capability map that the applied business process implements the application business capability while the change scholarship business process implements the study business capability. Each business service is owned by a different line of business, so to ease our governance, the decision is to create two different business process services. The first business task is supplying the new or the updated scholarship information. To implement this task, the user interface form will have to load the existing information so that users can enter them. Therefore, we need to load countries, load majors for countries, load existing scholarship information, and load the account information for the scholarship. To implement these actions, we get to reuse the country entity service and the major entity service that were identified during the analysis of the previous process. So here as you can see as we analyze more processes, the service inventory gets bigger and reuse opportunities get higher. Next, we need to load the scholarship information and here we have a design decision to make. Recall from our discussion about data granularity that here we basically have two options. We either create a capability, which returns all scholarship information. This makes the capability more reusable, but returns more information than we actually want or we create a capability, which returns only the required portion of the scholarship information. This makes the capability less reusable, but returns only the required information, which saves bandwidth and processing headache on the consumers. We go the second route and we identify a GetMajorInfoFromScholarship capability, which returns only the portion of the scholarship we're interested in. (Typing) And then to get the account information, we reuse the AccountInfo entity service, which was identified in

the previous process and we create a new capability to get the account information. All these capabilities will then be called during the form loading. In the next business task, we want to check if the new major selected is actually valid and the student is eligible to select this major. There are two business rules for this verification. First, we have to check if the student has already changed major once because the students are allowed a maximum of one change in major and we also have to check if the student has already any other process running. Now to check if the student has already changed major once, we add a new capability as part of the already existing scholarship entity service. (Working) Now, however, we have a more interesting design decision between our hands. Let's go back to the applied business process we discussed before. And recall that back then, we also needed the functionality to check for existing running processes. Back then, we had no reuse need, so we ended up adding this function as part of the composition logic of the task service itself. Now, however, after analyzing another process, we identify the need for the same logic again. So clearly we have a reuse opportunity. Therefore, we need to refine our work. We need to extract this logic and centralize it into a separate service. Hold on, however. We still have a decision to make. What kind of service shall we use for this centralized logic? Basically, we have two options. We either create a capability inside the applicant entity service or we create a new utility service to perform this check. I decided to create a utility service and here's why. See check existing processes practically means checking if the business process management engine has running a processes for the applicant. The way I see it, checking the running business processes and the BPM is not a business-oriented service and especially not one related to the applicant business entity. It's more of a crosscutting function of a technical nature that might be reused by various processes. Therefore, I decided to create a dedicated utility service, which will host all operations related to business process management ProcessInquiry. This can later include functions, such as composing a process, terminating a process, and so on. This means that now we need to refine our previous work and make the eligibility service compose the ProcessInquiry utility service since it now includes the extracted reusable logic. Now back to the change of scholarship process. We reuse the utility service, and to automate the entire thing, we create a task service called Validity. (Typing) This service then composes the scholarship entity service and the utility service. The next step, review request is a manual step where an employee manually reviews the data, so this is outside our SOA scope. Next, we want to actually update the scholarship to point to the new major. Two actions must be done here. The first is to update the actual scholarship record to hold the new major and the second is to update the account information for the student. Both of these actions can be implemented by reusing the existing scholarship entity service and adding an update capability. Also we reuse the AccountInfo entity service and we also add a new update capability. (Typing) And then we create a new task service called StudentRecord and this service composes the two entity services. The final step, we want to notify the student that his or her major change has been successfully completed. They call that the business processes long running because we have a manual step called review request. So when a student invokes this business process, she is not waiting for the response in real-time. Instead, she will be notified via this notification step. So we create a utility service that encapsulates the notification logic. This logic sends notification via email and SMS. Now as a hint, in the next process we will revise this decision based on new criteria.

'Reward Compensation' Business Process

The final process we're going to examine is the reward compensation process. As always, the entire process is exposed as a business process service called RewardCompensation. We submit reward info task is similar to the tasks that we saw in the previous processes as it requires loading the same information, so it's simply reuse services or any part of our service inventory. Review input data is a manual task, so it's outside our SOA scope. The task that we care about is check eligibility. Here we need to check if the student asking for the financial compensation is actually eligible for this compensation. To make it simple, let's assume that a student is eligible for compensation. In case the amount is under a certain value and the compensation type is an accepted one, such as a ticket compensation, course compensation, and so on, so we decide to create an entity service called reward and a capability called IsEligible. Not that this is yet another example of a business domain-specific capability, I have not simply accrued capability. Okay, here we have a couple of interesting design decisions to make. First, I decide to use a business rules engine to evaluate my business rules. Business rule engine will be explained in the next course, but in a nutshell, it gives us the ability to evaluate business rules at runtime, rather than at design time. A business rule engine allows us to change business rules at runtime without needing to recompile and redeploy our application. So using a business rules engine greatly increases flexibility because without it, every time the rules change, a separate compilation of the service is required. Now the next design decision shouldn't we call the business rules engine directly from the IsEligible capability of the entity service or shall we create a dedicated utility service and call the business rules engine from it. To decide, remember that the idea is keeping the entity service as business and technology agnostic as possible. Now in order to call the business rules engine, we need to identify the name of the policy and we need to access the API of the business rules engine all of which are concerns that should not be part of an entity service. Therefore, we create a utility service and call the business rules engine from this service. The entity service then calls or composes the utility service. Next, in the pay amount business task, we need to do two actions. First, we need to call the bank service passing in the bank account information of the student and the compensation amount to be paid. The bank then internally processes the request and notifies us back. After receiving the bank notification, we then need to update the financial record of the student. Once again, we have a couple of design decisions between our hands. So first, we need to call the bank. So we create an entity service called Bank with a pay capability. Note that this is similar to the case of the travel insurance provider in the apply business process. Because I know that we will be calling an external entity, but in terms of SOA design, I do not really care. I will simply leave the exact data access and technical details to the integration services layer. Next, our analysis shows us that the compensation amount as we have it is not in a suitable format for the bank and we cannot pass it as-is because the bank schema requires a different format. So the first idea that we have is to create a utility service, (Typing) which performs schema transformation, and then call the service from the bank service. Well this is a 100% good option and in fact utility services are perfect for these kind of functions. However, in this specific case, I already took a decision to use an enterprise service bus to manage the entire composition because it's long running. As we'll discuss in the next model,

ESBs are suitable for this kind of long running transactions and ESB has transformational capabilities built in. So we will use these capabilities instead of creating a separate utility service. This is a specific example of thinking ahead in terms of design. Now if we haven't done this kind of thinking, you would have created a utility service, and later in the more detail design phase, we would refine our work and use the ESB transformational capabilities instead. This is a normal thing to happen due to the iterative nature of SOA work. Next, to update the financial record, we create an entity service called FinancialRecord (Typing) with a capability update. (Typing) And finally, to compose and automate the entire logic, we create a task service called Payment with a pay capability. In the final task, we need to notify the student. Now this might look simple, but as always, we've got a design decision to make. In this specific process, we identify the need to send only email notification, no SMS notification is required. However, if we go back to the Change Scholarship process, recall that notification was done via email and SMS, so at that time, we ended up creating a single utility notification capability. However, given that now we must send email notification separately, then we need to refine our work and split notify into two capabilities, one, to send email, and the other to send SMS. (Typing) In addition, we create a capability called Notify, (Typing) which composes the other capabilities. Now in this change scholarship process, because we want both email and SMS notifications, the business task calls the notify capability, which then calls the other two capabilities. While back in our reward compensation process, the notification task only has to call the send email capability. This practice of having utility capabilities composing other capabilities is not that common, but this is yet another example that design is not an exact science and various situations might require different solutions.

A Word About Utility and Integration Services

The focus of this module was the identification of SOA business services, in particular, business process service, task services, and entity services. We also identify some utility services that were needed at this stage, such as the utility services, which checks running processes inside the BPM engine and the notification service. There are other types of utility services that will always exist and that provide crosscutting functionalities, such as logging and exception handling. These, however, are more detailed design aspects and we do not technically concern ourselves with them during this stage. In the next course, we will examine such topics and how they are implemented in SOA. As for integration services, as you know by now, they are into the detail, design, and implementation site as they decouple business services from existing systems. The identification and implementation of these services are explained in the next course.

Summary of Design Decisions

In this module, we identified the services from three selected business processes. During identification, we have seen some of the real world challenges and decisions that you have to make. We saw how we preserved the business

agnostic nature of antique services and kept any business process-specific logic inside task services. Task services also enable us to encapsulate composition logic away from the business process. This not only means that the task service itself can be reused, but more importantly, means that any change in the composition logic is abstracted away from the business process, which increases the flexibility of this process. At this stage, we had no special treatment when dealing with external providers. From business perspective, an external provider is just another source of information and the concern of how we communicate with the provider is left to the integration services layer. We saw that we have two types of compositions that are atomic or real time compositions and we also have long-running compositions. Task services will be encapsulating both types of compositions. This type of information helps us take design decisions, such as using an Enterprise Service Bus for some of the long-running compositions and utilize some of the enterprise service bus capabilities, such as transformation capabilities. Another example of early design decisions was to use a business rules engine to increase the flexibility of our services. These types of early design decisions save us time that otherwise will be required to refine our work later in detail design. We also saw how our understanding of granularity and functional context shapes our service identification decisions. We always look to have a unique functional context per service and we always try to balance reusability with the flexibility and the amount of information exchanged by a service. Governance play an important role in the grouping capabilities. We had the chance to include two business process services in the same service, but we decided not to do so in order to ease up governance as each business process service serves different capability. We saw how reuse opportunities grow over time. As more and more services are identified, our service inventor gets bigger and there are more chances that a functionality that we need either exists in the inventory or that can be assembled by composing other existing services. Refinement is a key factor in any analysis work. We saw how a design decision that we took in on process was changed as new criteria became available and this led us to extract logic and to introduce a new service. Expect such kind of refinements in your SOA projects. We saw how we use utility services to abstract technical concerns away from business services. For example, we decided that talking to a business process management engine is not something that should be part of the entity service logic, even if this means introducing a new service.

Create the Information Model

Introduction

Welcome to module, Create the Information Model. You are watching Real World SOA: Analysis and Design. The final piece of the puzzle is to create an information model that guarantees semantic interoperability between our services.

So what is semantic interoperability, why do we need it, and what exactly is an information model? All these topics will be explained in this module.

What Is Semantic Interoperability?

Often, when service developers talk about interoperability, they talk about standard communication protocols. For example, in web services, interoperability is achieved by using standards, such as XML, SOAP, and WSDL. Similarly, when using web APIs, interoperability is achieved by using HTML verbs and media types. These types of interoperability are called syntactic interoperability. It's about syntax, not data. The other type of interoperability is called semantic interoperability. While syntactic interoperability is about the syntax of data, semantic interoperability is about the meaning of this data. Services that are semantically interoperable have common understanding and representation of data. Semantic interoperability is achieved by defining a common information model, which defines the organization's standard business entities, such as student, major, scholarship, and so on. Services and service consumers then exchange messages based on this information model. Let's now examine the important advantages of the semantic information model. First, it provides common and clear definitions that are used consistently across the organization. This common business dictionary is an important communication tool during service analysis and design. Without the common understanding of what entities mean, it is impossible for stakeholders to agree on service definitions, especially as business app technical stakeholders might have different understanding of these entities. The core services and service consumers have common understanding of messages being exchanged. This improves the maintainability of services because changes are more easily and quickly applied. So let's say for example, that we have an application entity service with an apply capability that takes scholarship document as a parameter. After the application gets approved, we have a scholarship entity service with an add capability that also takes scholarship document as a parameter. Because both services use the same scholarship document and a later change on the structure or the meaning of the scholarship document is understood commonly by the two services. Also, because services have common understanding of the messages being exchanged, this improves performance because the need of transformation between business services is greatly reduced. So let's consider an example of the travel insurance task service with an issue capability. This capability composes the scholarship entity service with attached travel insurance capability and the provider entity service with a GetPrice capability. Because all services have the same understanding and representation of the travel insurance business document, the travel insurance service does not have to perform transformations when composing the other services. The same document is passed around unchanged. By having a common semantic model at the business services layer, these services understand each other regardless of how data is physically modeled in the existing systems. As explained before, physical data can have different data models, can be redundant, fragmented, and inconsistent, but all this doesn't matter from the perspective of the business services who only see the semantic information model. This greatly increases the flexibility and maintainability of business

services because the design of their information exchange model is based on optimal business needs, not based on the physical data model. As we will see in detail in the next course and touch on briefly in the next section, it is then the job of the integration service layer to do the required transformation from the semantic model to the physical data models.

Semantic Information Model vs. Operational Data Model

It's important to differentiate between the information model and the operational data model. The information model is at the logical high-level of business architecture and defines the business entities of the organization independent of how these entities are represented physically as data within data stores. The operation data model that presents the physical data persisted within data stores. For example, a product business entity is defined at the semantic level based on business service interaction needs. This entity is required to carry product attributes, such as let's say name, weight, and price, as well as a count of recent orders grouped by month, and the operational level this entity may be stored physically part in the ERP system and part in another custom relational database holding all the statistics. Integration services transform the product from the semantic level format to the format required for accessing the operational data. Note that because data physically might be spread across various systems, data access concerns might not only be about aggregating data from multiple sources, but might also require transaction management for right operations and you do not want such kind of concerns to be a part of your business services. This separation of concerns greatly enhances flexibility and loose coupling by decoupling business services from the physical data access concerns.

Creating the Semantic Model

Recall this diagram of the business architecture building blocks from the business architecture module. Back then, I said that business entities drive the creation of the information model. This is because business entities are defined at the organization level regardless of the actual physical data entities. This is essentially what we want for our semantic information model. Now during SOA analysis and service identification, more entities are discovered and existing ones are refined as we identify business services interaction requirements. In other words, SOA analysis helps the business architecture because in SOA we are thinking more about the interaction requirements of SOA services, so we get to spend more time analyzing the structure and interactions of business entities. This entire model is then called the semantic information model. To create the semantic model, we can set the following generic process. First, we understand the business domains and their information needs. We then model this information as a set of business entities and their attributes. Next, we model their relationships between these entities. And then finally, we're ready to derive business documents or messages from this information model. In the next sections, let's examine this process.

Step 1: Understand Business Domain Information Needs

The very first thing that we need to do is to understand the information requirements of our business domain. When you have a large business domain, it becomes easier to partition this domain into a set of subdomains. This way, you get to study the information needs of each subdomain separately with the corresponding domain expert as these experts have a clear understanding of their information needs and their relations between information. After you study each domain, you then work out the relations between these domains. Doing this, manages complexity and eases up management, maintenance, and communication. As an example, recall that when we explain the case study, we did actually partition our business domain into a set of subdomains. These subdomains are then used as the basis of information modeling. For example, we start asking ourselves what business entities we need to manage the application domain and we come up, for example, with an application business entity and an applicant business entity. Similarly, we ask ourselves what business entities we need to manage the study domain and we come up with a scholarship entity and a student entity and so on, and so forth until we cover all the domains. We're going to see that in more detail as we cover the case study later in this module. In summary, the whole idea is that by partitioning a big problem into a set of smaller ones, it becomes easier to work out this problem and this applies to information modeling. By looking at the organization as a set of interrelated business domains, it becomes easier to manage the information needs for this organization.

Step 2: Model Business Entities and Their Attributes

Having identified the business information needs, we then start working out the attributes required to describe each entity. So for example, start date and destination country are two attributes of an application, while age, gender, and name are three attributes required to describe a student, and status, start date, and finish date are attributes of a scholarship. Sometimes it might be helpful to indicate if attributes are mandatory or optional. This is especially done if the existence or lack of existence for an attribute holds special meaning for a business. For example, by specifying that a scholarship finish date is optional means that students are not required to know ahead of time when they will finish the scholarship. The exact date might change due to various reasons. However, by specifying that the scholarship start date is mandatory, means that students must specify when they intend to start the scholarship right from the beginning. This, for example, might be required by the agency for planning purposes. So this analysis will continue until we identify most of the attributes required to describe our business entities.

Step 3: Model Entity Relationships

Next, we need to work out how business entities relate to each other. As I said before, business domain experts play a crucial role in figuring out these relations. Relationships are mainly modeled as associations with the multiplicities between entities. For example, here we see relationships between student, compensation, and payment business

entities. The association relationship between student and compensation means that a student can have 0 or infinite number of compensations. In business terms, this means that there is no limit to the number of compensations that a student can get, while the compensation, of course, cannot exist without being associated to a student. The relationship between compensation and payment is interesting as it shows that we can have two types of associations between the same entities. So on the first relationship, the compensation can be associated with a single payment where the payment succeeds, while the other relationship means that a compensation can have multiple failed payment attempts. So again, in business terms, if will you recall the work compensation business process, this means that when we send the bank a payment order for a certain compensation, the payment might fail for whatever reason, but we still need to log such attempt and link it to the compensation. Therefore, we might end up with multiple attempts, but clearly, we need to have one succeeded payment attempt linked to the compensation. So again, it's the domain experts that have this knowledge and they are the best people to work with for information modeling.

Step 4: Derive Business Documents (Messages)

Finally, when you have enough of the information model ready, you can start deriving the business documents or messages as they are more commonly known. These documents are the ones that will be exchanged between SOA business services. Documents can be thought of as a hierarchy of data structures representing a particular view of the entire semantic information model. The best way to understand this is to see an example. So here you see a portion of the semantic model of our case study and this is an example of a scholarship business document being exchanged by SOA business services. So the first thing to notice is that the message contains a hierarchy of entities modeled in the information model. So the scholarship message includes information from the scholarship business entity, from the major business entity, from the travel insurance business entity, from the student business entity, which itself contains necessary information about the local address of the student and the overseas address. Remember from our previous discussion that there are two associations between a student and an address and these are modeled as two different address objects nested in the student object. Also notice that the relationship between a message and the business entity and the information model is not one to one. So a scholarship and the information model as a business entity with a set of attributes, when a scholarship message is a hierarchy of information extracted from different business entities, including the scholarship entity. Now what information the scholarship message requires is decided by the information Exchange requirements of SOA services, but the important thing is that this information is always a subset of the entire information model. Of course, it's not always the case that a message will contain a hierarchy of entities. For example, a SOA service that needs to retrieve a single student would, for example, define a student message simply with the same elements as that of a student business entity. While another service which needs to retrieve a student with the address information would define a student message with nested address information. Note that should this happen, you will end up with two versions of a student message and that's fine because the semantic of each message is different. So in summary, the messages

that you would define are based on information exchange requirements by SOA services. But whatever message is you define, they are always a subset of the semantic information model.

Case Study: Semantic Information Model

What you see here is the information model for the relevant part of our case study. Here I'm using numeric less diagrams, which is a widely-used notation information modeling. I won't go into the details for each business entity and each relationship you see here because that will really be just digging too deep into the business domain of the agency. Instead, I will talk about some entities and their relationships just to give you a sense of how modeling takes place. So we have already seen that a student has two associations with address, with role name Local, while the other with role name Abroad. This indicates that a student must have a local address and an overseas address. This is indicated by the association multiplicity numbers. An applicant, however, only has one address so that is one association and there is no need to specify a role name. A scholarship must be linked to one approved application. So we have an association with role name approved. An application, however, can have a maximum of one associated scholarship if it is approved. If it's not approved, however, then it will have no associated scholarship. The student must have one scholarship, else he won't be part of our domain and that scholarship can, of course, belong to only one student. If an applicant gets their application rejected, then it remains an applicant, and therefore, belongs to the applicant entity and doesn't belong to the student entity. Now recalling from our discussion of the change major business process that a student is allowed to change major only one time. This means that we have two associations between scholarship and major entities. The first one with role name active indicates the current active major, while the second one with role abandoned represent the major that was changed and I have already touched on that, but again, notice that a compensation must be associated with the payment. However, a payment might fail multiple times until we get it right. Therefore, we have two associations, one with role name succeeded, which represents the eventually completed payment and the multiplicity indicates that we must have one completed payment, while the other association with role name failed represents any failed payment, and the multiplicity indicates that there can be any number of failed payment attempts. So the same way you can continue reading the entire information model as it represents our business domain, but information modeling actually helps us not only derive messages, but also get early ideas about our service design. Let's see a couple of examples in the next section.

Case Study: Learning from the Information Model

Besides the advantages that we discussed about the semantic information model, notice that by understanding more about this model, we also get to shape up some of our service design decisions. So for example, as explained before, there is a failed and succeeded associations between real world and payment. We know that when a payment is made, it must eventually succeed even if it fails multiple times. Now in terms of service design, this means

that whatever service is invoked to make the payment, vault handling must be in place to ensure eventual success of the payment. This also means that a failed payment attempt is an important business entity to preserve, which means that our service must be able to record failed attempts and correlate these attempts to the eventual succeeded payment. As another example, we can see that a payment cannot exist without a reward. This means that the services creating these entities must make sure that adding a reward is successful before adding the payment. This creates a dependency on how services are invoked and might also result in a design that involves atomic transactions. Also, another example within the compensation, the due amount attribute is a derived attribute. Where its value is derived from the values of the requested amount and the paid amount attributes of the payment entity. This means that a student might get paid a different amount than what she initially claimed. The exact rules behind the calculation of this derived attribute is outside the scope of the information model, but this is just telling us that somewhere the business services must provide these rules. So in summary, what I want to say is that a semantic information model gives us some indications and requirements for our business services design. So the more we understand about the information model, the more our chance of creating a solid service design model.

Case Study: Messages

Finally, after having the information model ready, you can start identifying the messages required by services. Recall that we have already identified the services and their capabilities and we also identified at a high business level what a capability needs in order to perform its job. So one way to do this is to list down all service capabilities and then for each of the capability parameters we design the corresponding message from the information model. So for example, for the apply capability of the application service, we study the requirements of the capability processing and the information exchange requirements and we come up with an application message. Similarly, we study, for example, the requirements of the Get capability of the scholarship service and we come up with a scholarship message. I won't spend further time on this because that would be just going too deep into the business domain and the structure of the messages. Please refer back to the section of deriving business documents for further discussion on some of the design aspects.

Summary

There are two types of interoperability that are required for SOA. Many implementations focus only on syntactic interoperability, but while this interoperability is, of course, important, it's about syntax, not data. Semantic interoperability guarantees that business services have common understanding and representation of data. This has various important advantages as it provides a common and consistent vocabulary among stakeholders. Services become easier to maintain as they deal with the same set of entities. Performance is enhanced as we eliminate the need of transformation between interacting services. And services are loosely coupled from the physical data

models. To create the information model, we start by understanding the information needs of the business domain and domain experts play an important role in this. We then model these information needs as a set of business entities and attributes that describe these entities. Next, we model the relationships between entities, and again, domain experts play an important role in this. And finally, we're ready to derive messages out of this model. These messages are the parameters of the service capabilities.

Recap and a Look Ahead

Introduction

Welcome to the final module, Recap and a Look Ahead. You're watching Real World SOA: Analysis and Design. We have covered a lot of ground in this course. In this module, let's recap what we've learned. Let's also take one final look at how we have managed to achieve the business objectives through what we've learned and let's take a look at where to go from here.

What You've Learned?

To make it easier to grasp all the concepts of this course, I divided content into three areas. In the first area, we talked about the why aspect. We understood why SOA would be a suitable solution to a given problem. In other words, what are the goals that SOA helps achieve. We also understood what the challenges are that you typically face when adopting SOA. In your real projects, the importance of understanding these concepts is that you get to evaluate the benefits of SOA versus the challenges and the cost that you will pay to tackle these challenges. This kind of tradeoff analysis is more important than you might think because it's only after studying the tradeoffs that you can reach an informed decision if SOA is a suitable solution to your needs and this is the first essential step into not falling into the trap of failed SOA implementations. In the next area, we talked about the what aspect. Here we understood what are the building blocks of SOA, how these building blocks are structured, and how they interact with each other. Although SOA is more than just services, we did however allocate a considerable time discussing the services building block. We examined the different types of service classifications and the characteristics of these services. In your real projects, understanding these concepts is critical for designing the overall architecture and for service identification as you get to layer services and group capabilities into services in such a way that preserves loose coupling between business services and technical concerns and that provides the required balance between attributes, such as performance, reusability, and maintainability. In the final area, we covered the how aspect as we used our knowledge from the why and the what aspects to create the overall architecture driven by the organization's business architecture and then to perform the actual service identification, their capabilities, and their

information exchange needs in terms of a semantic information model. In your real projects, you get to have a sense of the real world challenges that are typically faced in SOA implementations. While it's true that each organization is different and each has its unique challenges, what you've learned here is definitely a major step ahead.

So How Did You Achieve the Business Goals?

The entire course was to show you when and how SOA helps organizations solve certain problems. This was done in the context of a real-world driven case study. In this section, let's briefly examine one final time, how did we manage to use SOA to achieve the business case objectives. Business-IT alignment is a core strategic goal for all organizations, our agents included. We want to make sure that the services that were built support our goals so that the effort and cost we put into building these services actually helps us move towards business-IT alignments. Following our top-down analysis approach, we can trace services back to business processes showing how work is done, and back to capabilities showing what is needed for work to be done, and back to value streams showing what values our work must deliver. This traceability guarantees that our services support business objectives, and therefore, are a real step towards the overall organization's business-IT alignment. Agility is a goal that most organizations seek, yet very few achieve. Agility means that organizations can take advantage of new opportunities and can react in response to changing requirements. Things that when achieved mean that organizations are always in the front seat when it comes to competition and providing sustainable value. For our organization, by adopting SOA, we create an inventory of reusable business aligned services. As the inventory grows, most the new demands are either satisfied from an existing service or by composing a set of existing services. This means that our business processes become easier to change and the new business processes are faster to create. Once this flexibility and business processes is achieved, the overall agility of the organization is improved. A direct impact of reusability of services is the fact that the development time is greatly reduced. As we start composing business processes and applications out of existing services, the time map cost required to create these systems is then reduced, but probably even more important is the reduction of the operation costs. Remember that SOA services remote single access of information and functionality. This means no redundant functionality and no duplicate and inconsistent data. This greatly reduces operation and maintenance costs. However, keep in mind that SOA does not come cheap. SOA requires up front analysis mainly due to the application of design principles to make services reusable and the existence of governance practices, so SOA must be seen as a long-term investment, rather than a quick fix for our cost problem. The cost gains of SOA are mostly apparent over time as reuse opportunities grow, but initially, we need to be ready to invest until we start seeing the advantages. Lack of customer centricity is a common problem to organizations, especially ones that grew out of acquisitions or ones that were built in silos, which is the case for our organization. In these types of organizations, customers feel as if they are dealing with different organizations when they are internally dealing with different units or domains. Through SOA, services are reused across different units, functionality is accessed in one place by different units, and data is consistent across units. This means that customers become unaware what

internal unit they are dealing with. For them, the organization is just one big source of services. The final goal was to introduce a new capability that allows the organization to extend its ecosystem to companies. The capability map helps us by allowing us to study the entire organization and set priorities based on business objectives. It's through these priorities that we then decide to leave achieving this goal to the next iteration. Without such holistic view provided to us by the business architecture, it would be difficult to envision where the new capability fits in and what is its priority related to the other capabilities.

A Look Ahead

In this course, we covered the overall architecture of SOA and the analysis method to identify services. Next, we need to start the detailed design phase. This phase includes things such as the design of individual services by applying a set of SOA designed principles. Service quality attributes, deciding the actual type of services will they be SOAP-based or web API-based services, and what criteria we use to take the decision, how services will access physical systems, and what are the challenges and their solutions, how services are secured, what role does an ESB play in SOA, and many service design considerations, such as exception handling, logging, and versioning. This and other design and implementation topics are examined in the next course.

Course author

Mohamad Halabi

Mohamad is a Solution Architect currently working in the e-government sector. He started his professional career back in 2003. He is a firm believer that architects are much more than just...

Course info

Level	Intermediate
Rating	★★★★★ (60)
My rating	★★★★★
Duration	3h 52m

Released 29 Jul 2016

Share course