Service-Oriented Architecture

COURSE NOTES

Version 0.1.0

Table of Contents

Module 2: Web Service

Upon completion of this module, you will be able to:

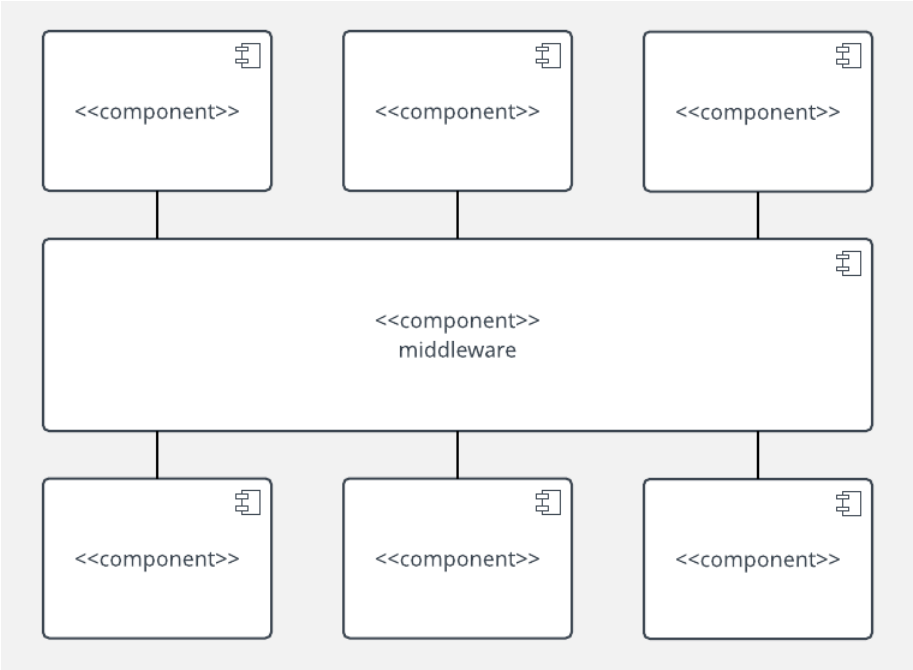
(a) Define a "first generation" web service
(b) Define what a web service is
(c) Explain how to use a web service
(d) Describe the standards for how web services are invoked, described, published, discovered, and composed

Introduction to Web Services

In order for services to be able to be used by other processes, there must be some way of "exposing" the service. In other words, services have interfaces that can be used by some service requester. It follows then that a web service is exposed and accessible using web technologies.

Before web services became commonly used, building integrated systems was a difficult task. Many different pieces of software might be involved, with even more combinations of connections between those software, which is work-intensive to implement, maintain, and extend.

Enterprise Application Integration (EAI) is an enterprise-level solution for integration problems in integrated systems. The exact architecture for EAI varies, but it does use middleware. In the previous module, we learned that middleware is software that is located between other software and facilitates interactions between them. Middleware helps manage complexity in an organization.



Rather than deal with multiple combinations of connections, middleware ensures that a system has fewer interfaces to implement and maintain. However, implementation in business to business (B2B) interactions, where interactions are between business instead of within them, is not always clear. For example, identifying which business implements the middleware, managing security, and protecting data from outside influence can complicate B2B implementation of EAI. Consequently, EAI is not used for B2B interactions.

To solve these issues, web services are implemented for interactions between businesses. Web services are usually implemented with a specific set of standards and protocols for implementing services over the web. Web services are defined by the World Wide Web Consortium (W3C) as "a software system designed to support interoperable machine-to-machine interaction over a network".

The table below summarizes the technologies and standards that make up web services.

Technology or Standard	Description

Web Infrastructure	Web services are built on top of web infrastructures. They start with TCP, the networking protocol responsible for reliable, ordered, connection-oriented communication. On top of that is HTTP, which web services use to send information and interact with their clients. Other transfer protocols exist as well as HTTP. HTTP however is compatible with nearly every machine and provides a strong foundation for platform independence and interoperability. HTTP is the basis for web services such as RESTful web services, which will be discussed later in this course.
Invoking (SOAP)	In order for a service requester to use a particular service, it must invoke it. Invoking is like a method call in an object-oriented language, except that it must be done through a request in XML. Invocation will include which operation is requested along with the parameters and data. In web services, invocation is done with SOAP, a protocol specification that is based on XML and allows services to send information to one another. Service requesters and service providers use these SOAP messages to send each other information. Since this is done through XML, systems coded in different languages and on different platforms can easily communicate.

Describing (WSDL)	Services must know how to interact in order for interaction to take place. **Web Service Description Language (WSDL)** is the standard protocol for describing the interface of a service. WSDL are written in XML. A WSDL description will describe the interface of a service in a machine-readable fashion, so that a service requester can bind itself to this interface. Binding is the act of generating the necessary code to interact with a service so that a service requester can begin invoking it. If a service interface is described unambiguously with WSDL, binding can be done by generating the necessary code automatically.
Publishing and Discovery (UDDI)	Service requesters and service providers need ways to come into contact. **Universal Description, Discovery and Integration (UDDI)** is used by service providers to publish descriptions of their services. Service requesters looking for a service can search by the WSDL descriptions or other aspects of the service. This is called **discovery**.
Composition (WS-BPEL)	Various standards can be built on the foundational standards of SOAP, WSDL, and UDDI for web services. These standards usually have the prefix WS, such as WS-Security for adding security functions, or WS-Coordination for coordinating the activities of many services. WS-BPEL is one of these standards, for Business Process Execution Language (BPEL). BPEL facilitates **service composition** as it allows developers to combine existing services into new, composite services. Composite services are services built with other services, which can be basic services which do not rely on other services, or can be other composite services.

Together, SOAP, WSDL, and UDDI are the three foundational standards of web services. They allow for web services to be invoked by service

requesters, to describe themselves, and to be published to registries where they can be discovered. All of them rely on web infrastructure.

> Instructor's Note: In this module, web services will interact through the use of XML-formatted documents, usually in request-response messaging design.

The standardization of how web services invoke, describe, and publish means that their internal implementation does not matter. Service requesters and services can effectively interact despite being on different platforms and in different languages. However, the commands and parameters of the standards must be supported by the service provider.

The use of web services makes building interfaces easier, because communication between services and service requesters is standardized. Although interactions are pair-wise, they are implemented with web service standards, so applications can be developed in any language and on any platform that supports standard web technology.

Web services can take on the role of middleware, and facilitate interactions between processes for business-to-business interactions. Businesses can control how they interact by choosing how to expose their services. They also do not need to provide a different interface for each outside company.

Service Invocation (SOAP)

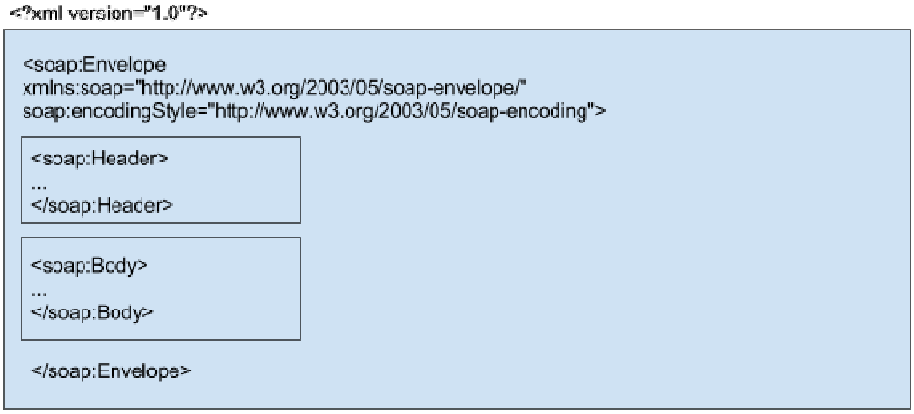
Web services use a form of XML messages for communication between service requesters and service providers. The request-response messaging pattern is the basis of all interactions between web services and the software that uses those services.

These XML messages conform to SOAP, a standard developed at Microsoft. SOAP allows a service requester to invoke services. It provides standardization to interactions between service requesters and service providers, or clients and servers. The way that XML defines and organizes requests and responses must be standard so that both parties can understand it.

Standardization therefore allows for interoperability between platforms and languages. SOAP also provides instructions for how the message is bound to the underlying transport protocol.

An SOAP message is meant to solicit an operation from a remote web service, and is in an XML-formatted document. Consider the example below to understand the structure of a SOAP message for a web service

that determines the exchange rate between two currencies.

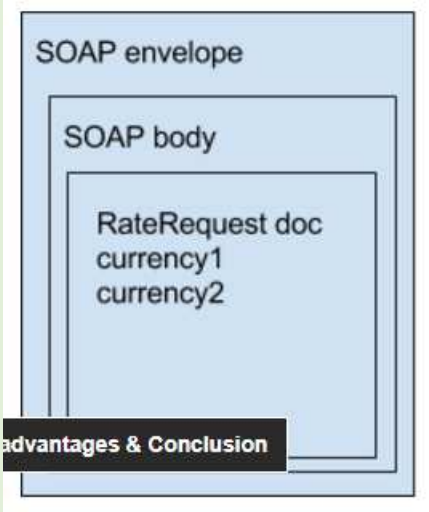


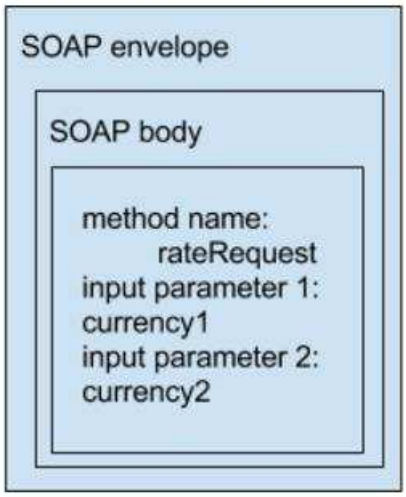
The XML document is identified as a SOAP message by the envelope. In this example, a header is used to also provide contextual information like information about the client or routing information. An envelope is required for SOAP messages, while headers are not.

Additionally, SOAP messages have a body. The body contains the information that the service provider needs to determine which service to provide and the service's input. The body is required.

Two "styles" of SOAP messaging exist. SOAP does not dictate which of the two styles to use. Both styles are commonly used.

Style of SOAP Messaging	Description

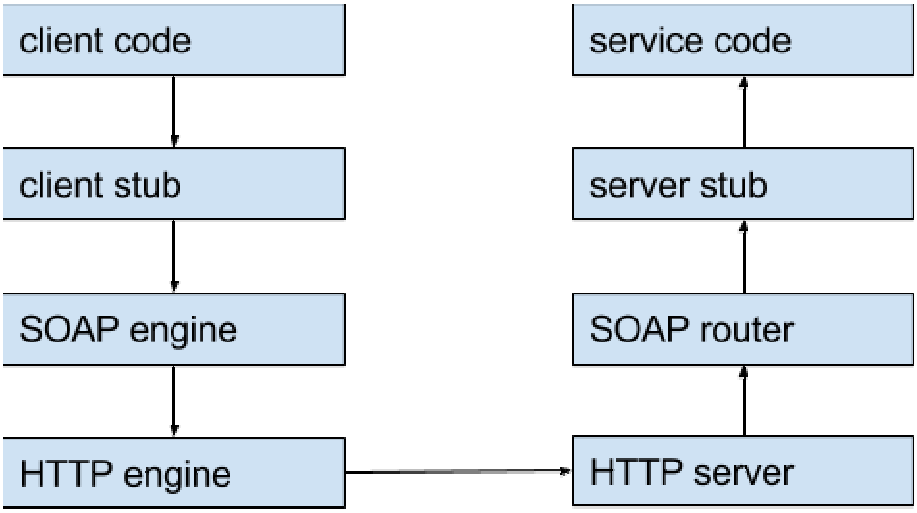
Document Style SOAP	In document style SOAP, a SOAP message is a structured document that contains a request that will be understood by both parties. Below is an example of a document style SOAP message for a currency exchange rate web service. The type of request is determined by the type of document. The two currencies that the service requester wants to compare are fields in the document. 

RPC Style SOAP	In RPC style SOAP, the body of the message is similar to a method-call, that consists of an operation and input parameters. Below is an example of an RPC style SOAP message for a currency exchange rate web service. In this example, there are two input parameters, the two currencies to compare. 

SOAP messages are sent over a transport protocol. For example, Simple Mail Transfer Protocol (SMTP) is used for email. This lesson will focus on using HTTP. SOAP messages are sent using HTTP POST. HTTP determines where to send the request, since this information is not directly included in the SOAP message itself. Since HTTP must acknowledge a POST request, it could return the response, or a simple acknowledgement that the request has been received.

Messaging is **synchronous** if the service request waits for a response before continuing. A program might be left doing nothing while waiting for a response, particularly if the availability or response time of a web service is an issue.

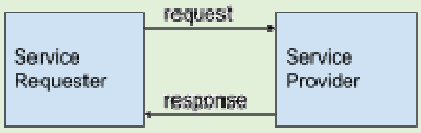
Messaging is **asynchronous** if interactions allow the code to keep executing. This means that when a message returns from the service provider, the code can process it.



A simple implementation of an SOAP request starts when the client code makes a local call to a stub. The stub then converts the request into a SOAP message. This SOAP message is packaged into HTTP and sent to the service provider. When it arrives, the HTTP server passes the content to a SOAP router. The SOAP router determines the appropriate server stub and delivers the message. The server stub uses the information of the SOAP message to make an appropriate method call. Once the service code handles the request, the process works in reverse to send this response.

Messaging Patterns

Four basic messaging patterns exist for SOAP. More complicated messaging patterns exist and are required for meaningful interactions. Since SOAP messages are stateless, these interactions are implemented by relating messages another way, like storing the interaction state on the client and/or the server, or by using extensions to web services like WS-Coordination.

Messaging Pattern	Description
Request-response	In a request-response pattern, the server requester first sends a message, then receives a reply from the service provider.  This process is synchronous and can be implemented over HTTP.

Solicit-response	In a solicit-response pattern, the service provider makes a request to the service requester.  This process is synchronous, and is often a confirmation.
One-way	In a one-way communication pattern, the service requester sends a request to the service provider, but no response is expected. This could be a simple notification that the service requester is up and running.  This process is asynchronous.
Notification	In the notification messaging pattern, the service provider sends a notification to the requester without expecting a response. This model is well-suited to event-based systems where there are publishers and subscribers.  This process is asynchronous.

SOAP messages have certain disadvantages, which fall beyond the scope of this lesson. Some of these disadvantages include the fact that XML encoding and decoding adds overhead and does not easily accommodate some data types. These disadvantages have resulted in SOAP being superseded in many applications by methods that use HTTP more directly, such as RESTful web services.

SOAP and its related web service infrastructure were the basis of the first major consensus on web services. SOAP's neutrality allowed systems on different platforms and in different languages to interact and pass data.

The XML-based structure allowed for machine-readable data that could be manipulated by any machine connected to the internet.

Service Description (WSDL)

In web services, **Web Service Description Language (WSDL)** is a standard used to describe the interface of a web service. This helps SOAP messages find services, and understand how to interact with services, including parameters. WSDL descriptions can be read by potential service requesters, either programmatically or by developers. WSDL was created by Microsoft, IBM, and Ariba by combining various attempts at standardization. WSDL descriptions are written in XML, and can be compared to method signatures in object-oriented programming.

WSDL descriptions include how to structure a request, the input parameters required, the data the service will output, the location where the service requester will send SOAP messages, the transport protocol it will be sent on, and more.

The WSDL description helps the service requester determine how to request the service. WSDL descriptions are machine-readable, which allows the service requester to generate necessary code to interface with a service provider automatically. This process is known as **binding**, whether or not it is done automatically or by the developer. Only after binding can the service requester invoke the service using SOAP messages that they structured with the help of WSDL descriptions.

WSDL Description

Let us examine the general form of WSDL descriptions. WSDL is modular, which means that it is made up of different components that can be combined to define the desired interface or multiple interfaces. WSDL descriptions can even import other WSDL specifications by importing their descriptions and using them to combine into new interface descriptions.

There have been several versions of WSDL. The previous versions used different terminology and structure, although concepts remain largely the same. Some of the most important part of a WSDL 2.0 description are:

- **Types,** which describe the data types that are used. Developers can define abstract data types in XML. If only basic data types already available in XML are used by interactions, then this part is not needed.
- **Interfaces** describe interfaces to the services provided in terms of what operations can be performed and in what order. The order of operations can be described by the message exchange patters of

request-response, solicit-response, one-way, and notification. Interfaces were formally called portTypes in WSDL 1.2.

Interfaces are abstract. They cannot be used until they are bound to the concrete implementations that are needed to use web services.

The categories used to bind interfaces to concrete implementations are:

- **Bindings**, which determine how the SOAP message is translated into XML, and dictate the form of the messages. This includes specifying between document-style and RPC-style interactions, as well as specifying the transport protocol on top of which the SOAP messages are sent.
- **Services**, which bring together interfaces and bindings, and assign them to **endpoints** or **ports**. These are located with URIs.

WSDL provides a robust, modular, and extensible service description language. WSDL description enables reuse because WSDL descriptions are broken into very fine descriptions which allow for the reuse of parts of WSDL specifications in different ways. WSDL documents can also import other WSDL documents, gaining them access to data types in the imported WSDL description or to interfaces.

WSDL descriptions, as an XML-based approach, like other aspects of web services, allow for interoperability between different platforms over the Internet. WSDL can also describe non-XML based services. However, like SOAP messages, encoding and parsing XML imposes computational costs. This is the major disadvantage of XML-based standards.

Service Publication and Discovery (UDDI)

Web services need to be **published** and **discovered**. Developers need to be able to find services to build apps, and there needs to be a way to get people to use these created services.

The advent of the Internet helped customers find services through search engines. However, it is important to advertise web services. This is known as publishing. The first framework for publishing was **Universal Description, Discovery, and Integration (UDDI)**.

UDDI was created in 2000 by Ariba, Microsoft, and IBM, but it is now managed by the Organization for the Advancement of Structured Information Standards (OASIS), which is a non-profit organization that also manages a number of other open standards. UDDI was intended to be used to specify a universal registry and broker of web services, using XML and

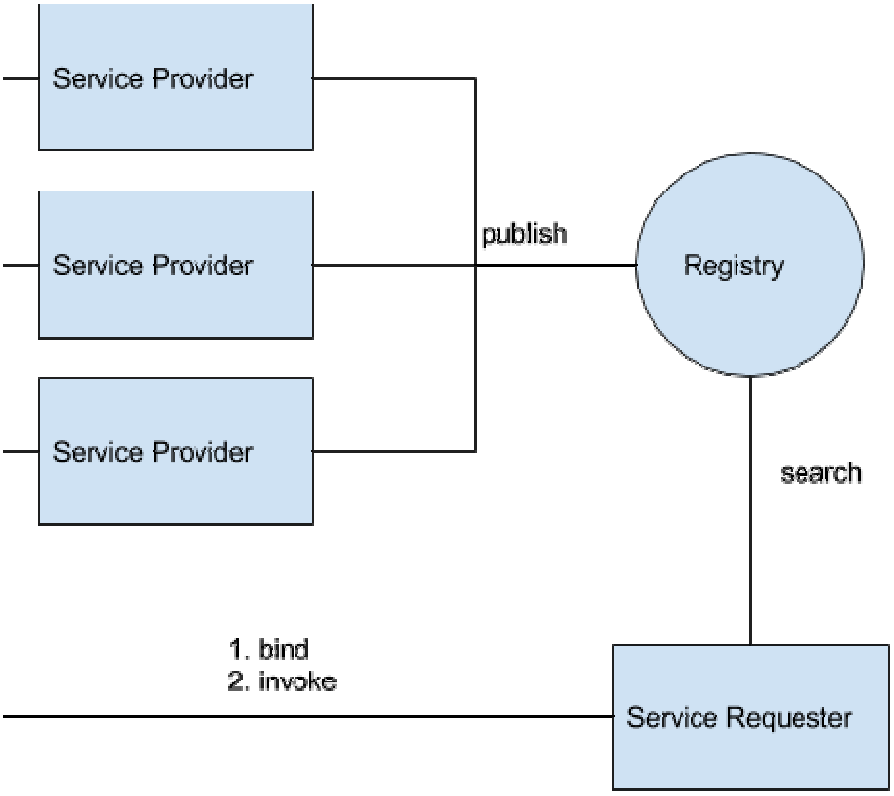
WSDL to structure data about the web services and how they were provided.

UDDI is a protocol for publishing and discovery, and not a necessary component of implementing web services, it is not tied to a specific registry. If the web services to use and the binding is known, then a discovery mechanism is not needed. Conversely, if a service is developed for use with a few specific applications, and it can be bound to those applications directly, then the web service does not need to be published. However, UDDI, is a useful standard for bringing service requester and service providers together.

Overview of UDDI

UDDI allows service providers to publish their services to a UDDI registry. Once they are published, potential service requesters can search the registry and discover the service they need. This is done by searching elements of the WSDL description, or other descriptions of the service or service providers. Bringing service providers and service requesters together is an essential part of creating an effective service ecosystem.

Once the service that the service requester wants to use is found, the service requester can bind to it using the WSDL descriptions. After the service requester is bound to the service by generating the code for the necessary messaging patterns, the service requester can **invoke** the service.



Publishing

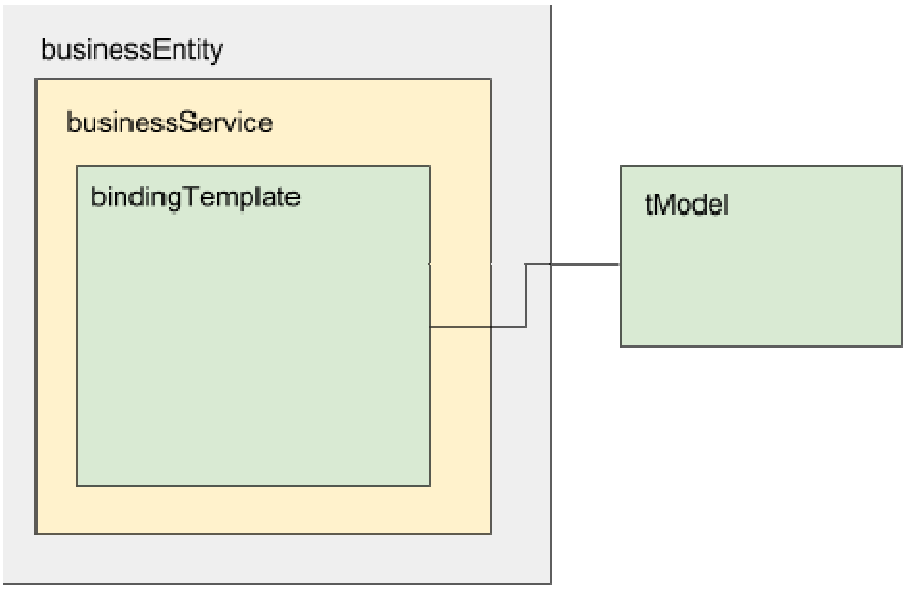
Publishing registers information about the service with a UDDI registry, which includes information about the service provider, the service itself, and various technical descriptions of the service. A **uniform resource identifier (URI)** is assigned to the service by the UDDI registry. The URI is a unique reference used to invoke the service.

Information encapsulated in the UDDI standard is contained in three categories. These are:

1. White pages, where information such as the business name, a short description, contact information, and unique business identification numbers is stored.
2. Yellow pages, which contain information about the service or industry that the business is in, including hierarchical information about the business. For example, exchange rates are a subset of currency services.
3. Green pages, which contain the technical details of how to use the service.

Information encapsulated in UDDI also falls into four data structures. These are:

1. businessEntity, which contains information about the business. This roughly corresponds to the "white pages".
2. businessService, which describes the services that the business is offering. A business can have many services, but a service can only have one business that owns it. This roughly corresponds to the "yellow pages".
3. bindingTemplate, which gives the necessary information needed to invoke the service in question. It is a description of the interface needed for communication. A service may have more than one bindingTemplate. This, along with the tModel, roughly corresponds to the "green pages".
4. tModel, which gives more detailed information about the service. This may include references to the protocols used by the service or detailed technical specifications of the service. The tModel is a flexible data structure; it can represent any abstract concept, and tModels can be nested to provide various meanings. For web services, one of these tModels will reference a WSDL description of the service. This with the bindingTemplate, roughly corresponds to the "green pages".



Publishing and Discovery as Services

UDDI specifies web services for publishing. Service providers publish, including adding, deleting, and modifying entries to a registry through SOAP messages. Operations could be *save_business*, *save_service*, *save_binding*, *save_tModel*, or delete commands for these same elements.

UDDI also specifies web services for discovery. These are accessed by SOAP messages. Commands to search for services include *find_business*, *find_service*, *find_binding*, and *find_tModel*. Information can be requested with commands such as *get_businessDetail*, *get_serviceDetail*, etc.

Service requesters use these commands to find and get information about the service interface. Once the service requester has information about the service interface, it can generate the necessary code to access the interface for the service. In other words, it can dynamically bind to it.

Publishing and discovery are often hidden from the developer. Instead, developers often use web portals to search for services.

Dynamic Binding

Binding can be a highly dynamic, run-time activity, although this has repercussions for a developer. For example, it may prevent a developer from knowing what errors might occur or what exceptions may be generated, and thus from developing robust code. Or a web service provided by a business may require contracts or agreements, which are managed by people and not programs. Consequently, service discovery is usually a design-time activity. Binding can still be automated, although this may be challenging as it would require the interface description to be completely unambiguous.

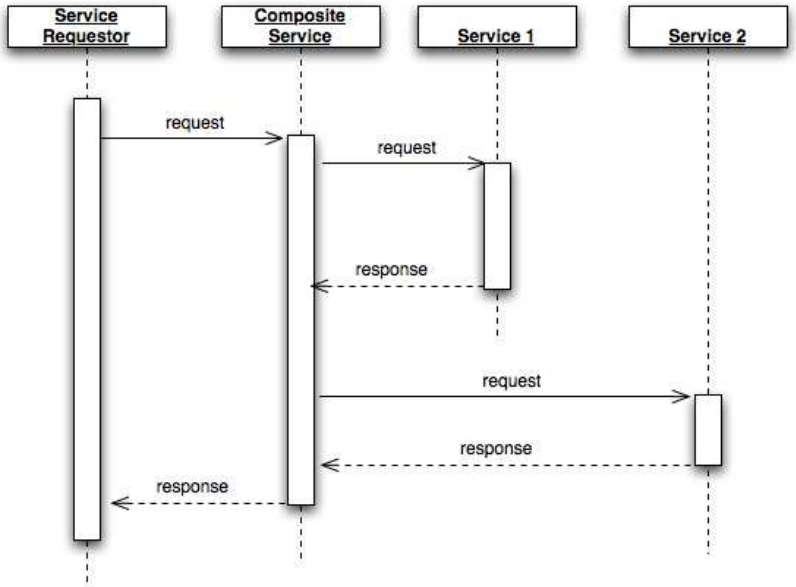
Discovery is usually performed by a human developer, while binding is at least monitored and tested by a human developer.

Service Composition (BPEL)

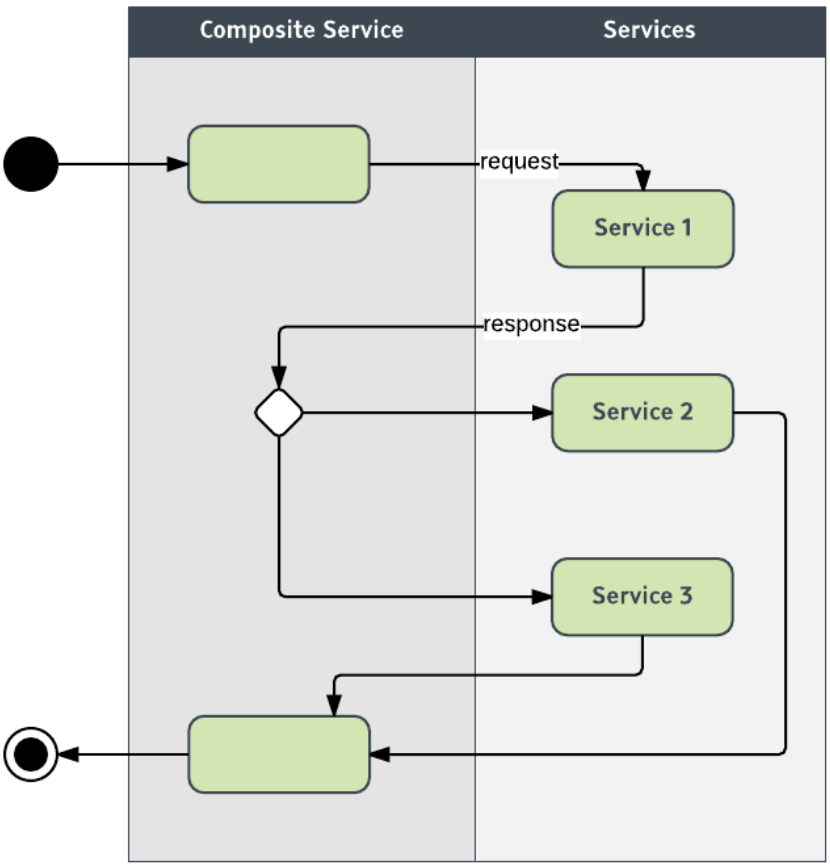
Composition is the action of coordinating several services and providing an interface. A service is composed when it is made up of other services. Further, a service that is composed of other, lower-level services may be used to compose higher-level services. This is comparable to the composing objects principle in object-oriented design. New functionality can be created by combining existing functionality, and encapsulating the new functionality as a service. However, web services are not usually compiled and run in the same physical location like objects.

Composing services involves invoking services in a certain order and handling exceptions that may arise. Services that comprise a composite service still remain separate from that service. A composite service could be like a "script" that uses other services according to some pre-determined order.

The UML sequence diagram below illustrates this:



Service composition can also be captured through a UML activity diagram, as requests made to a service or the service called may depend on response from another service.



Middleware services can be difficult to compose, because their interfaces are not standardized. By extension, wrappers had to be developed as needed, so that components could interface with each other or with the middleware.

Web services, on the other hand, can be easily composed. This is because web services are accessed in similar ways. However, the problems that come with combining services do not go away with web services, they are just easier to manage. This module has examined services invoked with SOAP messages, described by WSDL, and catalogued by UDDI.

Composing a service from other services is similar to a business process. Business processes are easily mapped to activity diagrams. The beginning and end of a process are the points where the composite service is exposed to service requesters.

Basic services often have low-level, basic functionality. This functionality can be combined into higher level functionality with Business Process Execution Language (BPEL), the standard high-level composition language for web services, also known as WS-BPEL. WS-BPEL can then expose this

higher-level functionality as a service which can also, in turn, be composed into higher-level services. Lower-level details of inter-service communication are dealt with by protocols such as SOAP and WSDL.

BPEL allows developers to compose compatible services into a business process that is itself exposed as a service. These services can be from external sources, on the Internet, or internal to a company or organization. Essentially, BPEL allows developers to create new services by composing them with existing services.

BPEL supports basic operations like "if-then-else" decisions, or other logic from various program languages and wrappers.

In addition to composition, web services are also associated with **coordination**. Coordination is when a process coordinates the activities of two or more services. Composition is distinguished from coordination because it exposes the collection of actions as another service.

This concludes the module on web services for this Specialization. The next, and final module will focus on REST architecture.