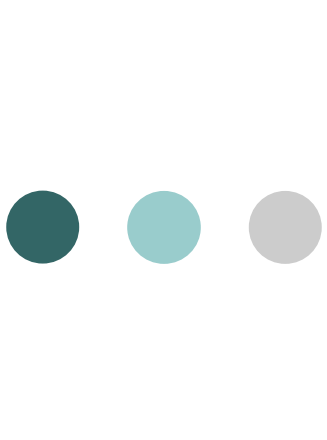
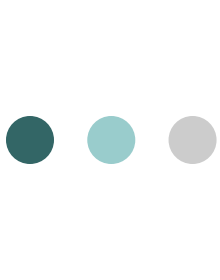


Lecture 12

SOA



Chapter 9

Web Services Architectures

Service Oriented Architectures

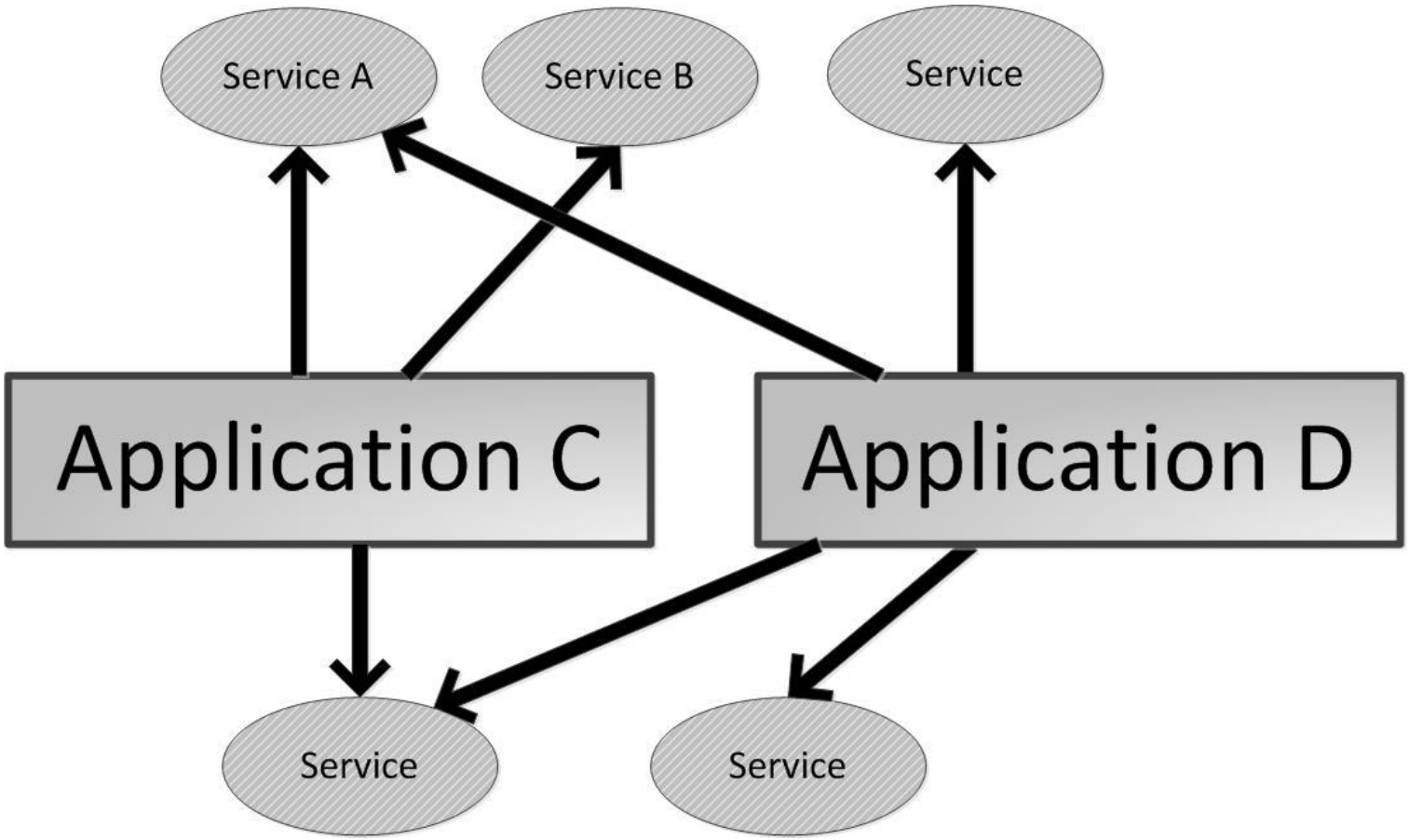
My own high level description, given as a sort of example:

- if application A does a particular task A, then application A can define an interface that allows access to the functionality provided by task A.
  - this interface would be called “service A” and would be described by a “service description.”
  - typically this service would be accessible through a particular web service technology (maybe WSDL/SOAP or perhaps JAX-RS)
- then if application B does a particular task B then application B can define an interface that allows access to the functionality provided by task B. T
  - this interface would be called “service B” and would be described by a “service description.”
  - again, service B would be accessible by a particular web service technology

Service Oriented Architectures

- Now application C calls service A and service B (as well as potentially other services) using the appropriate web service technology in each case, to perform part of the work that application C needs in order to do its own task
- We may also have application D that calls service A (as well as potentially other services) using the appropriate web service technology in each case, to perform part of the work that application D needs in order to do its own task

Service Oriented Architectures



Service Oriented Architectures

The main advantage of SOA is reusability

- Since each web service has a well-defined interface and is (typically) accessible using a well-known web service technology
  - any application can call that well-defined interface, and thus make use of that web service
- The individual web service applications are:
  - highly decoupled
    - and can vary independently of other web service applications
    - as long as they support their defined interface
    - This promotes reusability but also aids in improving maintainability and reliability

Service Oriented Architectures

- Another advantage is that since the interfaces (service A and service B in our example above) are defined using standard web-service technologies
  - internally the applications (application A and application B) can be written in different languages
  - can run on different operating systems
- New applications can easily access these web services, since they have well-defined interfaces
  - so integrating perhaps several of these web services together into newer, larger applications is relatively easy

Service Oriented Architectures

- However, there are many different technologies used to create web services
  - some web service technologies are able to call other web service technologies and some are not
- There are two main categories of web services, one category is called “non-RESTful” and the other is “RESTful”
  - within a category the web services can interact with each other, that is, for example, a RESTful technology can call another RESTful technology.
  - but between categories they are not easily interoperable, that is, a non-RESTful technology cannot easily call a RESTful technology

Service Oriented Architectures

Although typically SOAs are defined in terms of web services, and that's the common assumption

- it's also possible to do the same thing with distributed object/component technologies such as CORBA or EJBs or .NET Remoting
  - not usually on a cloud, however
    - because CORBA, EJBs, .NET Remoting work better in an intranet (inside a firewall)
    - or perhaps in some kind of large embedded system
      - (for a small embedded system a full up SOA might be overkill)

Service Oriented Architectures

As long as you have well-defined interfaces, and your new applications calls these interfaces in order to chain them together you have a pretty good argument that you have an SOA.

- perhaps with glue code, and perhaps some additional code in order to do new work

In Depth Look at Service Oriented Architectures

According to the Open Group (2016) (a global consortium that develops open, vendor-neutral information technology (IT) standards), a *Service Oriented Architecture (SOA)* is an architectural style that supports service orientation

- Service orientation is a way of thinking in terms of the outcomes of services, and how they can be developed and combined

In this definition, a *service* is a repeatable business activity that can be logically represented, the Open Group gives the examples: “check customer credit,” and “provide weather data”

- further, a service is self-contained, may be composed of other services, and consumers of the service treat the service as a black box

In Depth Look at Service Oriented Architectures

- According to the Open Group, when SOA was first defined
  - advocates of SOA claimed that SOA would replace all traditional IT architecture,
  - while traditionalists claimed it was a rehash of ideas about loose coupling and encapsulation

In Depth Look at Service Oriented Architectures

- The Open Group claims that neither position is correct
  - Rather, SOA is a distinct architectural style which is a major improvement over earlier ideas
    - although it includes some of the earlier ideas
  - Also, traditional architectural methods must be employed in order to obtain maximum benefit from using SOA

In Depth Look at Service Oriented Architectures

Another definition of Service Oriented Architecture comes from OASIS, Mackenzie et al. (2006):

> “A paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations.”

In Depth Look at Service Oriented Architectures

According to Mackenzie et al (2006), the focus of SOAs is to perform a task (business function)

- This is different from some other paradigms, such as object-oriented architectures, where the focus is more on structure of the solution
  - in the case of an object-oriented architecture, the focus is on how to package data inside an object.
- with an SOA, although you can think of methods as providing access to services, the focus is not on the methods
- SOAs map well into the normal ways that human activity behaves, because both humans and SOA work by delegation
  - Also, with both humans and SOA, ownership boundaries are an important consideration
  - SOAs address ownership boundaries through service descriptions and service interfaces
  - SOA provides reuse of externally developed frameworks by providing easy interoperability between systems

In Depth Look at Service Oriented Architectures

- According to Mackenzie et al (2006) (cont'd)
  - generally speaking, in order to perform a task, an SOA groups services on different systems, possibly running on different operating systems, possibly written using different programming languages
  - Most current SOA-based applications employ a synchronous client/server-type architectural style
    - A client issues requests to one or more servers, and the client blocks until the servers reply to these requests

In Depth Look at Service Oriented Architectures

- Chou (2008) argues
  - this works well when the application is centrally managed
  - even though the software components themselves are loosely coupled,
    - at a business process level this kind of central management can cause tight coupling between different business processes
  - Apparently, the argument here is that a different business process would have to interact with the central manager of the current business process in order to get its own work done

In Depth Look at Service Oriented Architectures

- Chou (2008) also argues that an asynchronous event driven architectural style is better for real time or proactive systems
  - since business processes are treated as a sequence of events
  - therefore different business processes that have little relationship with each other except for a few individual shared tasks do not have to obey the same kind of centralized management

In Depth Look at Service Oriented Architectures

In an asynchronous event driven architecture, an event message carries a state change to an event server

- The event server passes these events along to the servers, possibly with value added
- Servers may then generate messages for other event servers
- This is an example of a publish/subscribe architecture

In Depth Look at Service Oriented Architectures

SOAs don’t solve all your problems

- according to Mulesoft (2015), in the early 2000’s many companies thought SOAs were going to solve all their IT infrastructure problems
  - they spent millions of dollars on top down SOA initiatives
  - however, by 2009 many of these top down SOA initiatives had failed
  - Mulesoft (2015) says that some studies estimate only around 20% of these initiatives were fully completed

In Depth Look at Service Oriented Architectures

By 2009, this resulted in a backlash, for example, Maines (2009) stated:

> “It’s time to accept reality. SOA fatigue has turned into SOA disillusionment. Business people no longer believe that SOA will deliver spectacular benefits. ‘SOA’ has become a bad word. It must be removed from our vocabulary.”

In Depth Look at Service Oriented Architectures

Maine went on to say that instead of concentrating on services, which is what the SOA acronym stands for, people had gotten involved in technology debates

- including non-RESTful (WS-*) vs. RESTful

In Depth Look at Service Oriented Architectures

- Mulesoft (2015) claims that it wasn't SOA that was at fault,
  - it was the top down approach to SOA
  - a bottom up approach is needed instead

In Depth Look at Service Oriented Architectures

- According to Mulesoft (2015), the reasons why the top down approach didn't work:
  - Companies gave the responsibility of SOA adoption to a single team
  - This team was pressured to get SOA in place as quickly as possible
  - This pressure led to adoption of a proprietary SOA governance product (Oracle, IBM, etc.)
    - Developers were seldom consulted about which proprietary product to choose
    - These proprietary products were very expensive
  - Developers were tasked with moving to SOA in addition to their regular work
  - And now they had to employ a new mandated tool

In Depth Look at Service Oriented Architectures

- The basic idea behind a top down approach is
  - the group responsible for SOA identifies candidate services throughout the company
    - based on how the business itself works and interacts

In Depth Look at Service Oriented Architectures

- A bottom up approach is
  - the process of moving to SOA starts with developers
  - moving to SOA is done in a lightweight manner to begin with, primarily based on wrapping existing applications
    - This wrapping work is scheduled as part of developers' normal work tasks
    - One good way to work this is to begin by looking for problem areas,
    - if you look into wrapping an existing application with a web service, it's perhaps also a good time to fix any problems in the application
    - thus moving to an SOA becomes also a refactoring of your existing code, and can improve quality in other ways than just imposing the SOA architectural style

In Depth Look at Service Oriented Architectures

- Pomares (2010) quotes Rob Barry from SearchSOA as saying
  - that a bottom up approach has been criticized for requiring excessive updates and rework later on
  - the top down approach has been faulted for taking excessive time to produce results due to the extensive planning required

In Depth Look at Service Oriented Architectures

- ServiceOrientation.com (2015) says
  - the top down strategy requires a large initial investment because of the up-front analysis required
  - a bottom up strategy results in services that have a shorter lifespan and require more maintenance and refactoring

In Depth Look at Service Oriented Architectures

- One more term that often shows up when SOAs are discussed: "mashup."
- There are multiple definitions of mashup.

- One definition from Merrill (2009) says that:

  "Mashups are an exciting genre of interactive Web applications that draw upon content retrieved from external data sources to create entirely new and innovative services."

In Depth Look at Service Oriented Architectures

- According to Merrill, “mashup” was borrowed from popular music, where a mashup is a new song that is mixed from two different source songs
  - a mashup consists of the content providers, the mashup site, and a web browser
  - the work of combining the different sources can be done at the mashup site, or alternately in the web browser.

In Depth Look at Service Oriented Architectures

Chapman (2009) says that a mashup:

> “implies easy, fast integration, frequently using Open APIs and data sources to produce results that were not the original reason for producing the raw source data.”

In Depth Look at Service Oriented Architectures

- So what is the difference between a mashup and an SOA?

- McKendrick (2006) quotes Hendrick as saying that one difference is
  - a mashup must be web-based
  - an SOA need not be web-based (note that this goes against the more common assumptions about SOA)

In Depth Look at Service Oriented Architectures

- McKendrick says
  - mashups are developed in informal ways
  - SOAs are developed in a formal manner

In Depth Look at Service Oriented Architectures

One distinction between a mashup and an SOA comes from Hiroshige (2010):

> “Service-Oriented Architecture (SOA) is one way of architecting a Mashup or any other composite solution. SOA makes it easy to combine the existing applications because they are already exposing their functionality or data via their Services. You could (not that you’d want to) also architect your combined solution in a non-Service-Oriented way—it would still be a composite application even without the SOA.”

Relationship Between SOA and Cloud Computing

- There is some variation in opinion as to how heavily interrelated the two paradigms are:
  - Raines (2009) says:

    "Cloud computing and SOA can be pursued independently, or concurrently as complementary activities."
  - According to IBM (2009):

    "Some pundits have said that cloud is a flavor of SOA or a replacement for SOA. Neither of these statements positions the relationship between cloud and SOA accurately. They are actually complementary. Each supplies unique capability and functionality, and the two work together to provide an enterprise agility solution."

Relationship Between SOA and Cloud Computing

IBM (2009) also says:

> “SOA is an architectural approach that creates services that can be shared and reused”

but further:

> “on the other hand, cloud computing is about providing ease of access to and usage of services”

and finally:

> “Together SOA and cloud can provide a complete services-based solution.”

Relationship Between SOA and Cloud Computing

However, others find SOA and cloud computing more closely related.

- Krill (2009) quotes Cuomo, CTO of IBM WebSphere:

  "SOA is an architectural style for building applications, loosely coupled, allowing composition. Can we build a datacenter infrastructure on SOA principles? Yes, and that's the cloud, so it's a service oriented infrastructure. It's taking that architectural principle of SOA and applying it to an infrastructure."

- Mulholland et al. (2008) from HP said:

  "However, without Service-Oriented Architectures (SOA), organizations will find it almost impossible to reach the cloud."

Relationship Between SOA and Cloud Computing

Let’s examine this distinction.

- A cloud, loosely speaking, is a collection of computers that can be used to run applications and store data

- An SOA is a distributed architecture where individual applications are loosely coupled together through access of well-defined interfaces, typically implemented as web services.

Relationship Between SOA and Cloud Computing

So it's easy to see that you can get a lot of bang for the buck if you run your SOA on a cloud.

- each separate computer could potentially run a different web service application
- each web service application can potentially be moved to a different computer if the computer it's currently on gets too busy

Implementations of SOA

- The JAX-WS implementation of the Sailboat Marina project from chapter 18 is an example of an SOA
  - At least when using the GUI
  - The GUI accesses services that are located at different urls
  - The GUI uses those separate services to create a unified interface that successfully controls and monitors a Sailboat Marina

Implementations of SOA

- With the other Java-based Sailboat Marina projects, it's not as clear up front that they are examples of SOAs because:
  - They are loaded onto the same web server in the same war file
  - They use the same port number

Implementations of SOA

- With the other Java-based Sailboat Marina projects, it's not as clear up front that they are examples of SOAs because:
  - They are loaded onto the same web server in the same war file
  - They use the same port number

- Why might these be considered SOAs? What do you think?

RESTful Architectural Style

According to Fielding (2000), the RESTful architectural style focuses on:

> “the roles of components, the constraints upon their interaction with other components, and their interpretation of significant data elements”

RESTful Architectural Style

The RESTful architectural style consists of

- constraints on data
- constraints on the interpretation of data
- constraints on components
- constraints on connectors between components

RESTful Architectural Style: Resources

- Information is described as a “resource,”
  - “resource” is defined as the target of a hypertext reference
  - A resource can be empty, or a resource can be a collection of entities
  - A resource can be static, or it can change over time
  - One resource is distinguished from another resource by its mapping, that is, its hypertext reference

RESTful Architectural Style: Representation

- A “representation” consists of the data and metadata that is used to describe the data
  - Typically, a “representation” refers to an HTTP message plus its contents, or alternately a document or file
    - an HTTP message could include a document or file
  - There can be control data included in a representation (in this case an HTTP message) that specifies the purpose of the message
  - The format of the data in a representation is defined by an Internet Media-type (MIME type)
  - An example of a “representation” would be JPEG image or an HTML document

RESTful Architectural Style

- The simplest REST component is a “user agent ”
  - an example would be a web browser that accesses some kind of server using a client connector
  - the server that is accessed would use a server connector to reply

RESTful Architectural Style—Connectors

- The RESTful architectural style includes “connectors” that define
- how resources are accessed
- how the representations of the resources are transferred.
- the concept of connectors is used to hide underlying networking issues
  - so a connector is a communication abstraction

RESTful Architectural Style—Connectors

- Examples of RESTful connectors include:
  - “client”
  - “server”
  - “cache”
  - “resolver”

RESTful Architectural Style—Connectors

- “client” makes a request
- “server” listens for connections and returns a response
- “cache” can be used by either client or server to save information
  - so that it doesn’t have to be regenerated,or sent again
  - in cases where the information has been unchanged
- “resolver” is an address translator that converts a name to a full address

RESTful Architectural Style—Constraints

The RESTful architectural style possesses the following constraints:

- Client/Server
- Stateless
- Cacheable
- Uniform Interface:
- Layered System
- Code on Demand (optional)

RESTful Architectural Style—Constraints

- Client/Server: separation of concerns, exemplified by a client/server architecture
  - different components can evolve independently
- Stateless: the client-server interaction is stateless.
  - There is no stored context on the server.
  - Any session information must be kept by the client
- Cacheable: data in a response (a response to a previous request) is labeled as cacheable or non-cacheable
  - If cacheable, the client (or an intermediary) may reuse that for the same kind of request

RESTful Architectural Style—Constraints

Uniform Interface: there is a uniform interface between components. In practice, there are four interface constraints:

- resource identification—requests identify the resources they are operating on (by a URI, for example)
- resource manipulation through the representation of the resource
- messages are self-descriptive—the message contains enough information to allow a client or server to handle the message
  - normally done through the use of Internet Media Types (MIME types)
- use of hypermedia to change the state of the application

RESTful Architectural Style—Constraints

- Layered System: components are organized in hierarchical layers
  - the components are only aware of the layer within which the interaction is occurring.
    - intermediate filter components can change the message while it is in transit
- Code on Demand: (optional)
  - clients can download scripts that extend their functionality

RESTful Architectural Style—narrowing in

In industry, a RESTful web service is a web service

- that accesses URLs
- that uses HTTP messages (typically as part of the world wide web) to communicate
- may or may not always adhere to the “stateless” constraint
  - for example, cookies are often used

non-RESTful Web Services

How well do non-RESTful Web Services meet RESTful Expectations?

RESTful Expectation	Non-RESTful Web Services
Resources	Yes, uses urls
Uses representation to describe data	Yes, can use MIME types
Clients	Yes
Servers	Yes
Caches	Could be
Resolvers	Could be

non-RESTful Web Services

How well do non-RESTful Web Services meet RESTful Expectations?

RESTful Expectation (Constraint)	Non-RESTful Web Services
Client/Server	Yes (we already did that)
Stateless	Can do either with or without
Cacheable	Could be, either with or without
Uniform Interface	No, since HTTP is at a lower layer. Interfaces are specific to the task at hand.
Layered System	Yes
Code on Demand (optional)	Optional so let's ignore it

non-RESTful Web Services—narrowing in

As the term is commonly used, a “non-RESTful web service”

- defines a remote procedure call using a specifically defined interface
- this remote procedure call treats the HTTP protocol as a transparent lower networking layer
- it doesn’t have to know anything about HTTP, it just assume it’s there and it works
- uses the SOAP messaging protocol Sends data using an XML-type data format

non-RESTful Web Services—narrowing in

However, there are some confusing areas:

- A Web Services Description Language (WSDL) – based web service meets most of the previous non-RESTful definition
  - However, it is possible to configure a WSDL interface that runs directly using HTTP
- It's possible to use a Remote Procedure call and use SOAP but have the data sent directly over TCP
  - See .NET Remoting
  - However, .NET Remoting is distributed objects rather than a web service

non-RESTful Web Services

Advantages to using a WSDL-type interface:

- the interface is documented to a certain degree by the technology
  - you can read
    - name of the remote procedure
    - names of the parameters
    - types of the parameters
    - the return type
- since the interface is documented, clients can be automatically configured to work with the interface (see wsimport)