

Web Services: Navigating the Alphabet Soup

Scott McManus

CS 6210, Advanced OS

November 19, 2008

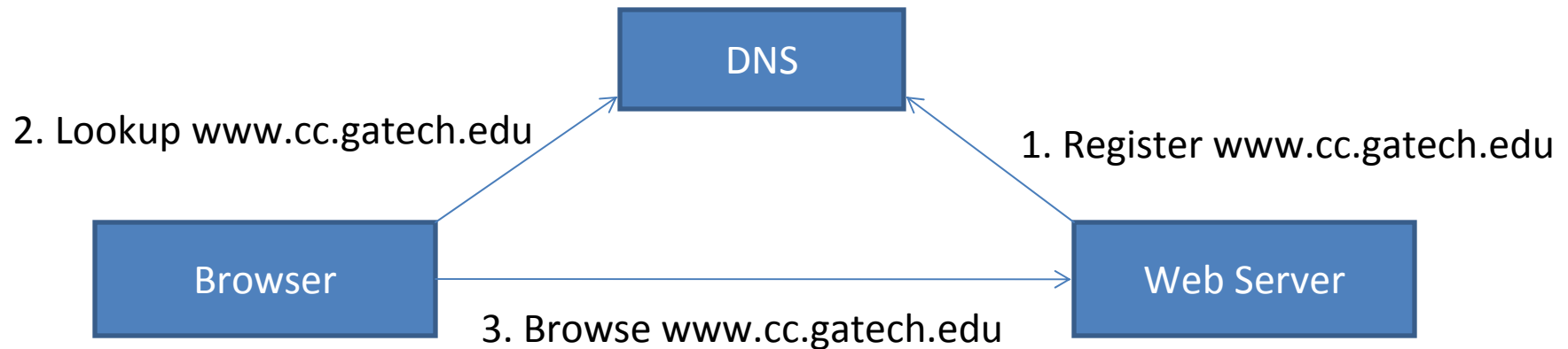
Overview

- Old School: CORBA & RMI
- J2EE Basics
- Web Services: XML/WSDL/UDDI/SOAP
- Service-Oriented Architectures (SOA)
- SOA Policies

Register/Find/Invoke

- Most of the technologies you'll see have a register/find/invoke paradigm:
 - The server ***registers*** itself with a directory
 - The client ***finds*** the service in the directory
 - The client connects to the service & ***invokes*** operations

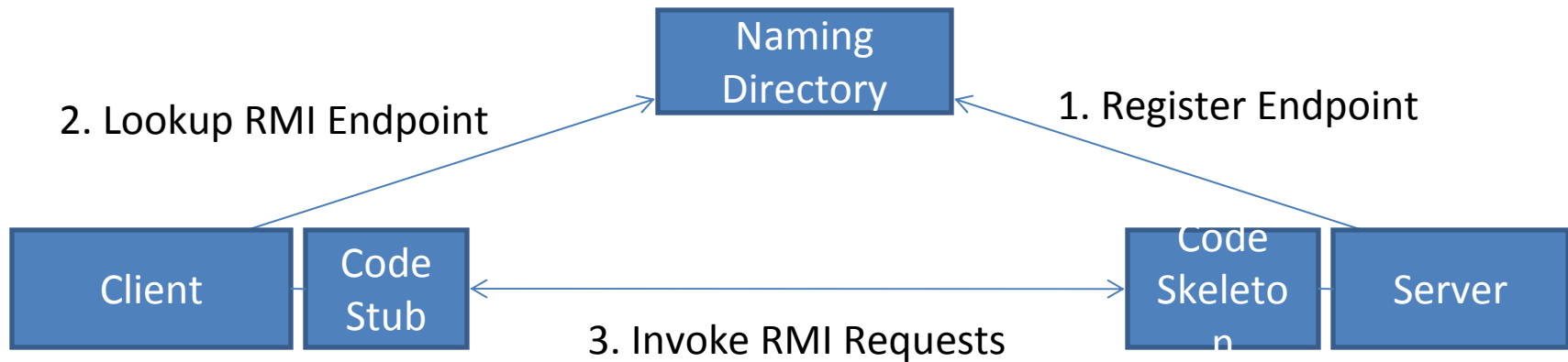
Register/Find/Invoke with DNS



RMI & CORBA

- RMI: Remote Method Invocation
 - Remote Procedure Calls were automated by compiling Java code into a ***stub*** executed by the client and a ***skeleton*** executed by the server.
 - Servers registered with a naming directory
 - Clients looked up endpoints at the naming directory.

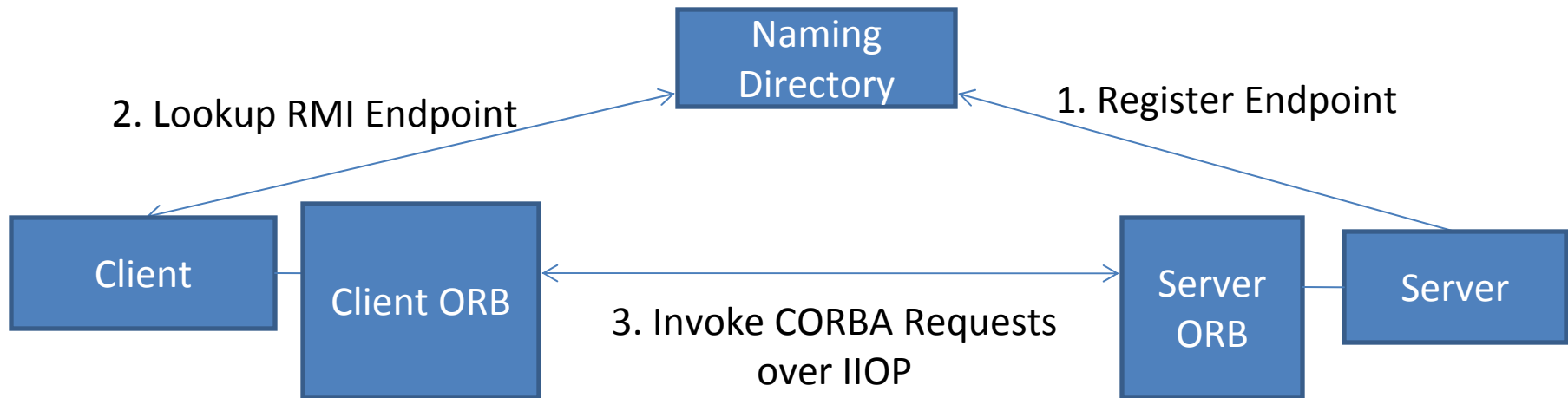
RMI - Example



CORBA

- CORBA – Common Object Request Broker Architecture
- Similar to RMI but with some key differences:
 - Object Request Broker (ORB) handled communication and session management
 - Generalized protocol, Internet Inter-Orb Protocol (IIOP) for communication
 - Client and Server specified actions with Interface Definition Language (IDL)

CORBA – Example



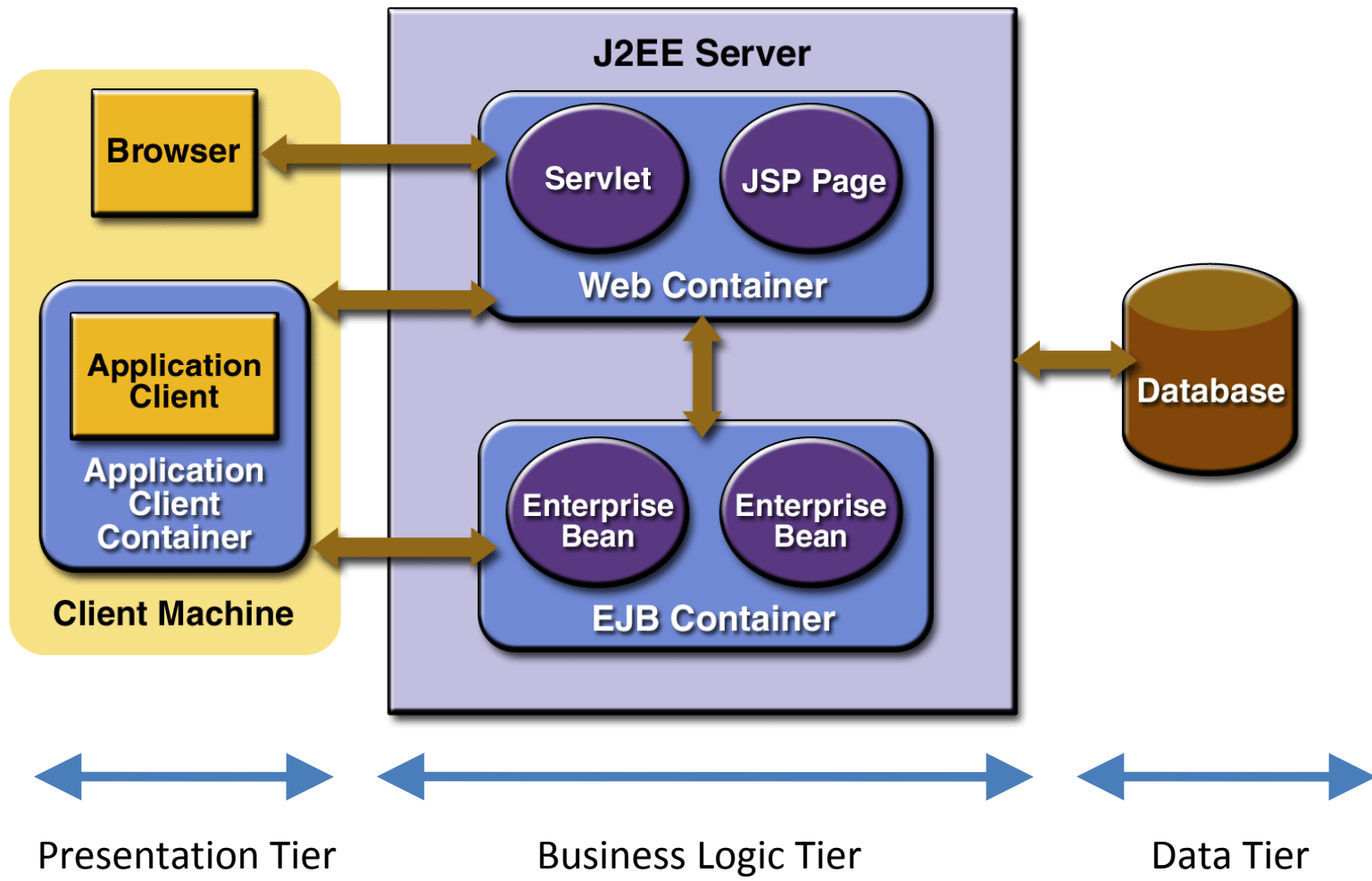
What Happened to CORBA?

- Complicated specification
 - Means interoperability was tough to impossible
- All objects were accessed as if remote
 - Slow response times (think of “Lightweight Remote Procedure Calls” and other papers)

Java 2, Enterprise Edition

- J2EE wanted to learn from RMI and CORBA issues, among others
 - How is interoperability between services **within** a single company handled?
 - How is interoperability between services **between** companies handled?
 - A growing number of legacy systems need to be accessed without rewriting code for a new protocol (which no one wants to do).
 - Want to present a single “face” for a service.

J2EE Architecture



J2EE Web Container

- Web Containers:
 - Handle generic HTTP processing
 - Route requests to appropriate service (JSP or Servlet)
 - Route replies to browsers
 - Maintain sessions, threads, and object life cycle (i.e., common things in a Web app that no one wants to recode)

J2EE Servlets & JSP

- A servlet is a Java class that is run on the server.
- A Java Server Page (JSP) has code embedded within an HTML page

J2EE: Why use anything beyond servlets and JSPs?

- If servlets & JSPs can do anything, then why is anything else needed?
 - It could be helpful to automatically do some tasks, like transaction processing.
 - Some applications need to be aware at some level of session semantics.

J2EE Enterprise Java Beans (EJBs)

- There are three major kinds of beans:
 - Entity Beans
 - Session Beans
 - Message Driven Beans

J2EE Entity Beans

- An Entity Bean represents a persistent object (i.e., one that's stored in a database)
 - They have primary keys and can communicate with other entity beans
- Two kinds of entity bean persistence exist:
 - Container Managed Persistence (CMP) - the EJB Container handles retrieval from the database
 - Bean Managed Persistence (BMP) – the EJB itself has calls to the database

EJB – Session Beans and Message-Driven Beans

- Session Beans:
 - Represent a single client
 - Can be:
 - stateless (so each request is handled independently)
 - stateful (the details of the conversation are maintained)
- Message-Driven Beans:
 - Pretty much the same as Stateless Session Beans
 - Basically an interface to Java Messaging Service

J2EE Naming

- The Java Naming and Directory Interface (JNDI) has to be accessed to find services.

J2EE Standard

- J2EE is a standard, not a product
- There are many J2EE-compatible products out there.
- A short list (by no means exhaustive or the most important ones) includes:
 - IBM Websphere
 - BEA WebLogic
 - Apache Tomcat (web container only)
 - Apache Geronimo
 - JBoss

Modern Web Services

- J2EE isn't used for everything
- Web services often consist of the same pieces but with different technologies:
 - XML: structure and encode data
 - SOAP: logical messages
 - WSDL: define web service interfaces
 - UDDI: discovery & naming service

Extensible Markup Language (XML)

- XML gives the ability to define tags
 - Schemas and Document Type Definitions (DTDs) define the structure of an XML document
 - XML documents can be transmitted and understood if the schema or DTD is available.

XML Example

XML Document TypeDefinition (DTD):

```
<?xml version="1.0"?>
<!DOCTYPE envelope [
  <!ELEMENT envelope (header, body, signature)>
  <!ELEMENT header (#PCDATA)>
  <!ELEMENT body (#PCDATA)>
  <!ELEMENT signature (#PCDATA)>
]>
```

XML Document:

```
<?xml version="1.0"?>
<envelope>
  <header>Scott McManus, 266 Ferst Dr</header>
  <body>Scott - here's that check for $0.02.</body>
  <signature>Ada Gavrilovska</signature>
</envelope>
```

Simple Object Access Protocol (SOAP)

- SOAP defines a common message structure to use in Web Services
- Each SOAP message is an envelope with a header and body.
- The message can be structured however the service wants to structure it.
- There are some SOAP DTDs already defined for common header elements.
- SOAP is transported over a protocol, which will be HTTP for web services.

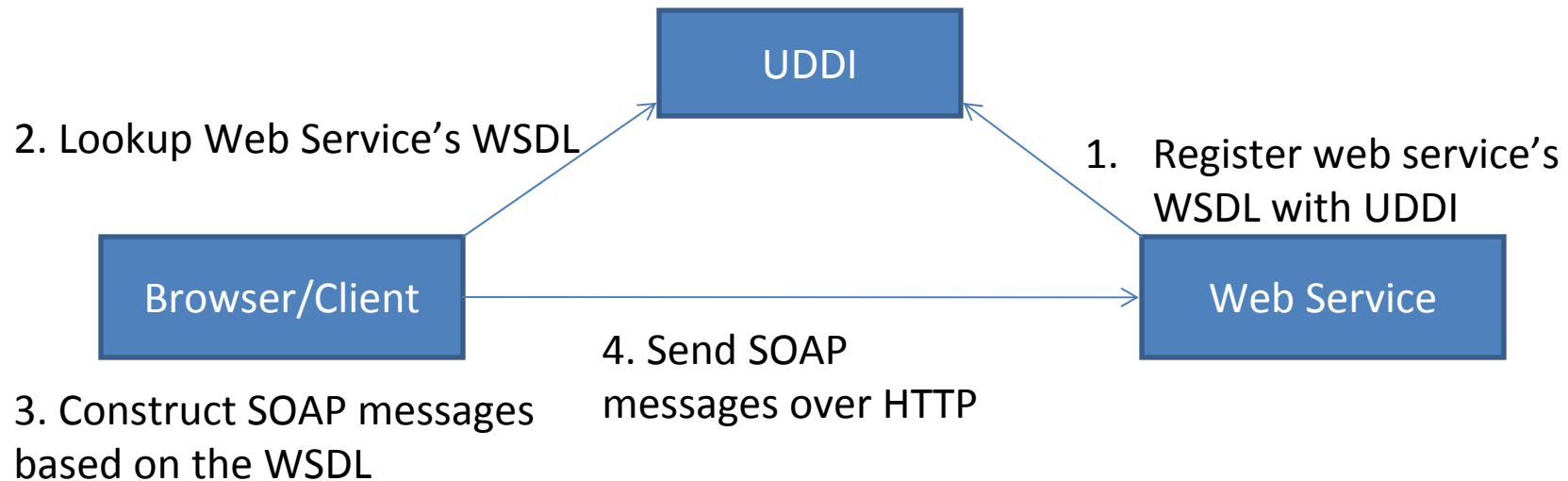
UDDI: Universal Description, Discovery, and Integration

- In our register/find/invoke cycle, we still need something to do the registration and finding.
- That's where UDDI comes in.
- Services registered with UDDI may be:
 - Private: only the company can use it;
 - Semiprivate: the service is only visible to certain people;
 - Public: the service is visible to everyone.
- UDDI uses XML for its messages, too.

WSDL: Web Services Definition Language

- We may know how to lookup the service, but how do we know what the service does or where it lives?
- Don't want to hardcode service's definition
- WSDL defines the service's:
 - Message types
 - Operations (or “interfaces”)
 - Communication protocols (or “bindings”)
 - Location

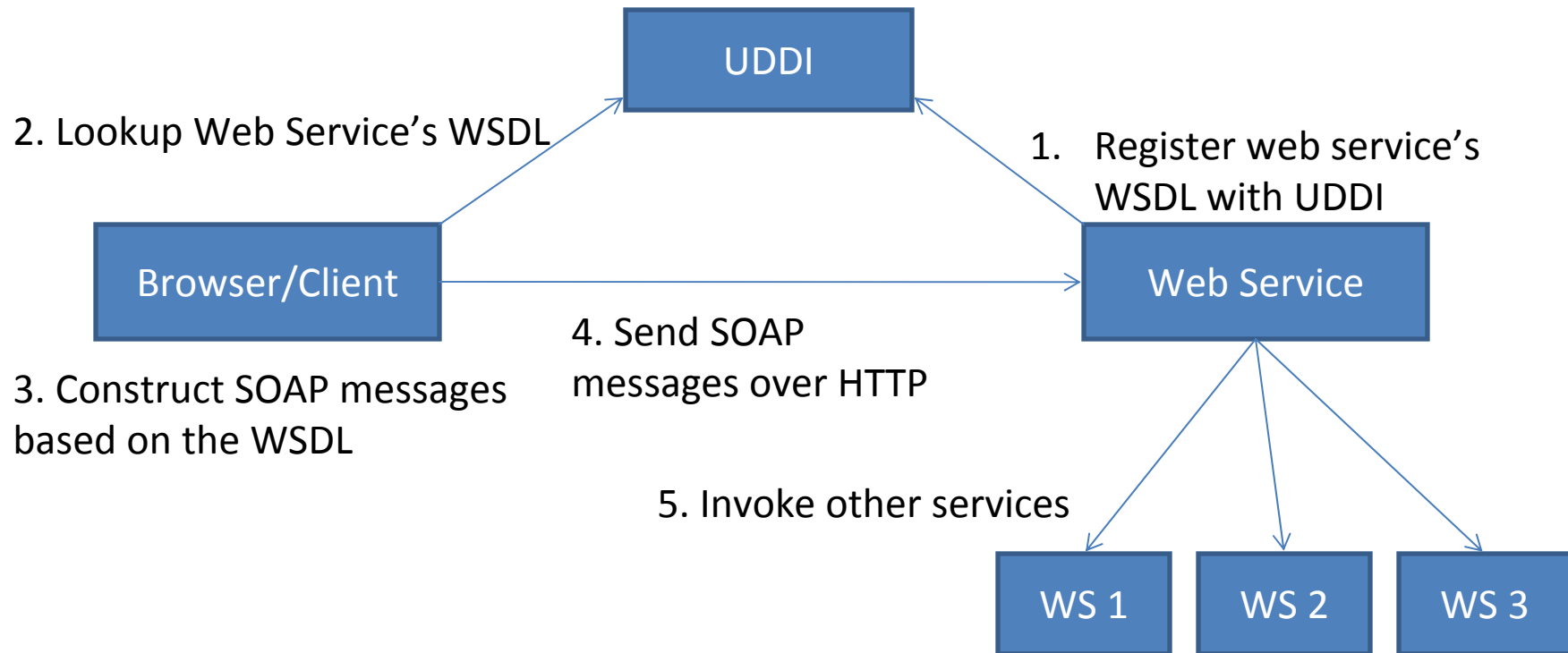
Web Services: Putting it all together



Service Oriented Architecture (SOA)

- Emerged in current form around 2004
- Like J2EE, there needs to be some way to invoke multiple web services across and between companies.
- Individual Web Services are broken down into component services.
- A complete service is a composite of multiple services.

SOA Example



Bundling Services Together

- What if invoked services call other services?
- Service Component Architecture (SCA) defines the relationships between services.
 - Process-oriented: Execute services in a certain order, as in a workflow.
 - Structure-oriented: Define the graph of services that are invoked.

SOA Contracts & Quality of Service (QoS)

- Contracts define what one of these aggregate services will do. This is based on WSDL, just as with a single service.
- **Jargon Alert: Quality of Service means something very different in networking**
 - Networking: QoS defines latency, bandwidth, and behavior when there's too much data
 - SOA: QoS refers to qualities of the service, such as semantics about the service (e.g., security, transactional guarantees (ACID properties), and reliability)

SOA Policies

- These are relatively new – first defined in 2007
- Service description includes a policy.
- Policy = Requirements + Capabilities + Behaviors
 - Requirements: specify what service users must do
 - Capabilities: specify what service can do
 - Behavior: the operations that a service performs

Policy Scope

- Policies can be interpreted in a few places:
 - All messages to an endpoint, as in reliability guarantees;
 - Execution of a single operation, as in a transaction;
 - Handling of a single message, as in security.

SOA Policies for Bundled Services

- An application or service may be invoking several other services.
- What if one of these other services can't make the same guarantees about reliability?

Methods to handle SOA policies for bundled services

– Bottom-up:

- Starting with qualities of the underlying services, determine what the composite of those qualities will be.
- Problem is that there are no set rules for how to do this composition and calculate the qualities of the resulting service.

– Top-down:

- Start with intended service qualities and determine other bundled services (or component) configurations.
- Simple models are difficult to configure this way.
- Qualities of the service will have to extend to components that can't handle them.

Questions?

Thanks!