National College of Ireland

Mobile Architecture & Security

Service Layer

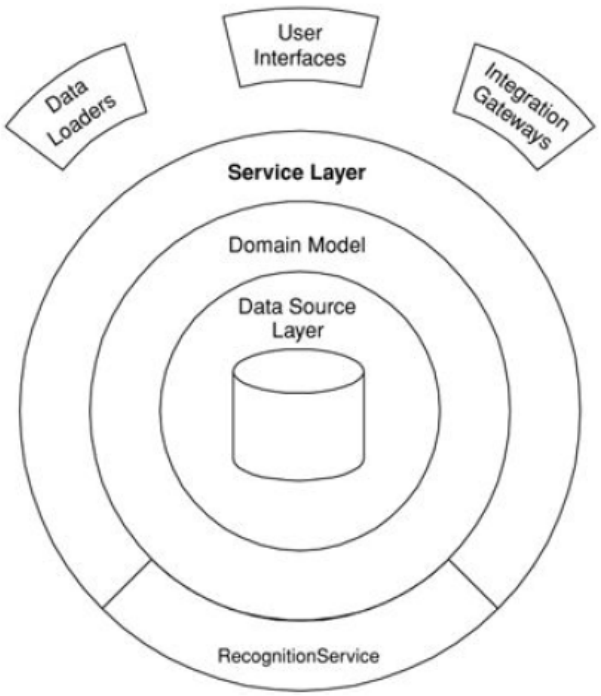
Mikhail.Timofeev@ncirl.ie

9. Data Access Layer

1. The Service Layer
2. Application Servers
3. Web of Services
4. Service Orientated Architecture
5. Web Service Protocols:
    1. XML
    2. RCP
    3. SOAP
    4. REST
    5. REST vs. SOAP

The Service Layer

– The Service Layer



- Defines an application's boundary with a layer of services that establishes a set of available operations and coordinates the application's response in each operation
- Organises business logic
  - Domain logic
  - Application logic / workflow

Are there any issues with putting application logic into pure domain object classes?

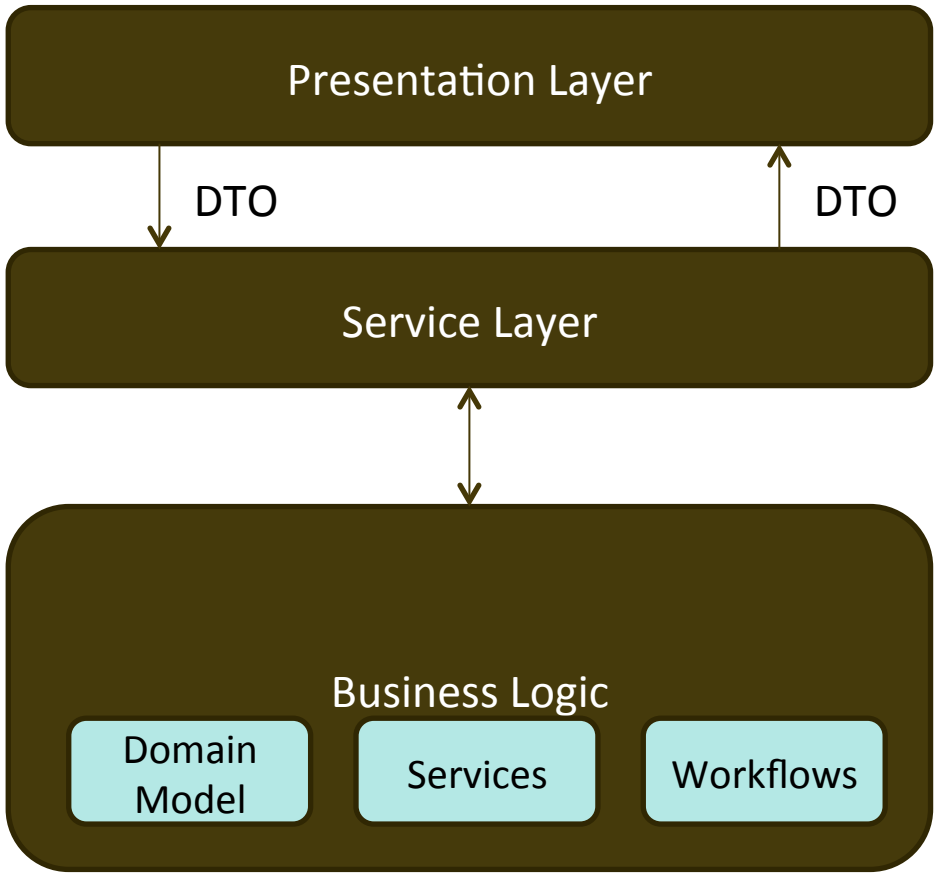
- Maybe less re-usable if implementing logic associated with a single application
- May inhibit separation of workflow responsibilities to a workflow engine etc.

The Service Layer

- What is the Service Layer?
  - Defines an interface for the Presentation Layer to trigger system actions.
  - Promotes Loose-Coupling between the Presentation Layer and the Business Logic Layer
  - Similar in idea to Transaction Script pattern
    - TS less structured
  - Service Layer orchestrates
    - The set of business objects used by an application
    - Application specific services
    - Workflow
    - Etc.
  - Should be used in applications where there is some complexity
    - Multi-tiered
    - Multiple front-ends etc.

The Service Layer

– Service Layer Responsibilities



- Service Layer can access the Data Access Layer as required also
- Orchestrates other services
  - Workflows
  - Database operations
- Service provide business related operations that application clients can repeatedly execute

The Service Layer

- Service Layer Responsibilities
  - Service can wrap a class's functionality and add extra capabilities
    - Cross-cutting concerns
    - Messaging
  - Services should be created with a fixed public interface
  - Services may be discoverable over the network
  - Applies principles of Low-Coupling and High-Cohesion with the aim of decoupling Presentation Layer and Business Logic Layer
  - May handle remote requests

The Service Layer

- Service Layer: Micro Services and Macro Services
  - Macro Services
    - Coarse-grained
    - Similar to Transaction Script ⇔ map to use-cases
    - Do not contain domain level logic
  - Micro Services
    - Offer specific services to the application and functions to the domain logic
      - Currency conversion service
  - Presentation Layer will typically make use of macro services, however, for simple requirements a micro service may be targeted

The Service Layer

- Service Layer Implementation

Advantages	Disadvantages
Loose-Coupling	Overhead for simple systems
Minimise Traffic between Presentation Layer and Business Layer	
Enables application remoting	

  - Services may be implemented as web-services, WCF services etc., however, it should also be noted that the Service Layer may be implemented as a set of standard classes.
  - Services can be grouped logically, e.g., CustomerService

The Service Layer

- Service Layer Implementation
  - Each service layer class should implement an interface

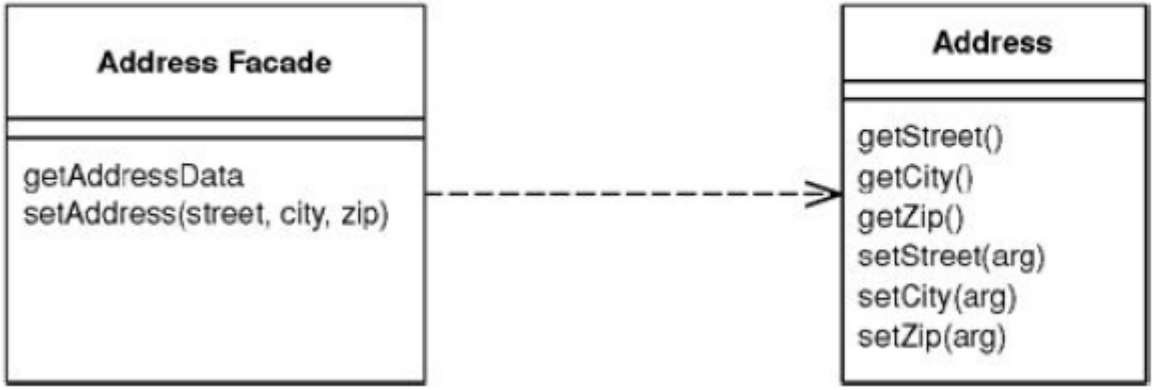
```
public interface ICustomerService
{
    void Create(Customer customer);
    List<Customer> FindAll();
    Customer FindByID(int customerID);
    // . . .
}

public CustomerService : ICustomerService
{
    // . . .
}
```

- What are the benefits of implementing an interface here?

The Service Layer

- Service Layer Implementation
  - The Remote Façade Pattern

Provides a coarse-grained facade on fine-grained objects to improve efficiency over a network.



- Use where you would like consistency between existing APIs
- OO leads towards creation of small objects with individual possibilities
  - We want to be able to execute a batch of operations over fine-grained objects
  - Create a remote façade to offer a coarse-grained interface

The Service Layer

- Service Layer DTOs
  - May be required as domain objects may not be easily serialised
  - Pragmatic decision required as to how extensive DTOs should be used in application
    - Number of domain entities
    - OK for Presentation Layer to receive domain objects
    - Presentation Layer and Service layer shared process space
  - Implementation of the Adapter design pattern
    - Bi-directional
      - Adapter from domain object to DTO
      - Adapter from DTO to domain object

The Service Layer

- Service Oriented Architecture (SOA)
  - Service Orientation envisages the software architecture as providing a set of interoperable services that can be composed together to support business processes.
    - SOA promotes a loosely-coupled architecture
    - SOA often incorporates open standards
    - SOA is a design paradigm for service design in general – not isolated to an application's Service Layer
  - SOA Principles
    - Boundaries are explicit
    - Services are autonomous
    - Use contracts, not classes
    - Compatibility is based on policy

The Service Layer

- Service Oriented Architecture (SOA)
  - Boundaries are explicit
    - Interaction with the service occurs through a public interface which acts as a clear explicit contract
    - Interface should be as simple as possible
    - Usage of message based semantics (rather than RPC)
  - Services are autonomous
    - Each service is deployed, managed, and versioned independent of the system in which it is deployed and consumed
    - Loosely-coupled
    - Communicates with other services via contract based messaging and policies

The Service Layer

- Service Oriented Architecture (SOA)
  - Use contracts, not classes
    - No platform specific details can cross the service boundary
    - Data exchange via XML strings
    - Service definition includes a service contract and data contracts
  - Compatibility is based on policies
    - Consumers of services should be able to determine a services capabilities
    - Compatibility should be exposed via publicly accessible and standard policy

Application Servers

- Also called an **appserver**

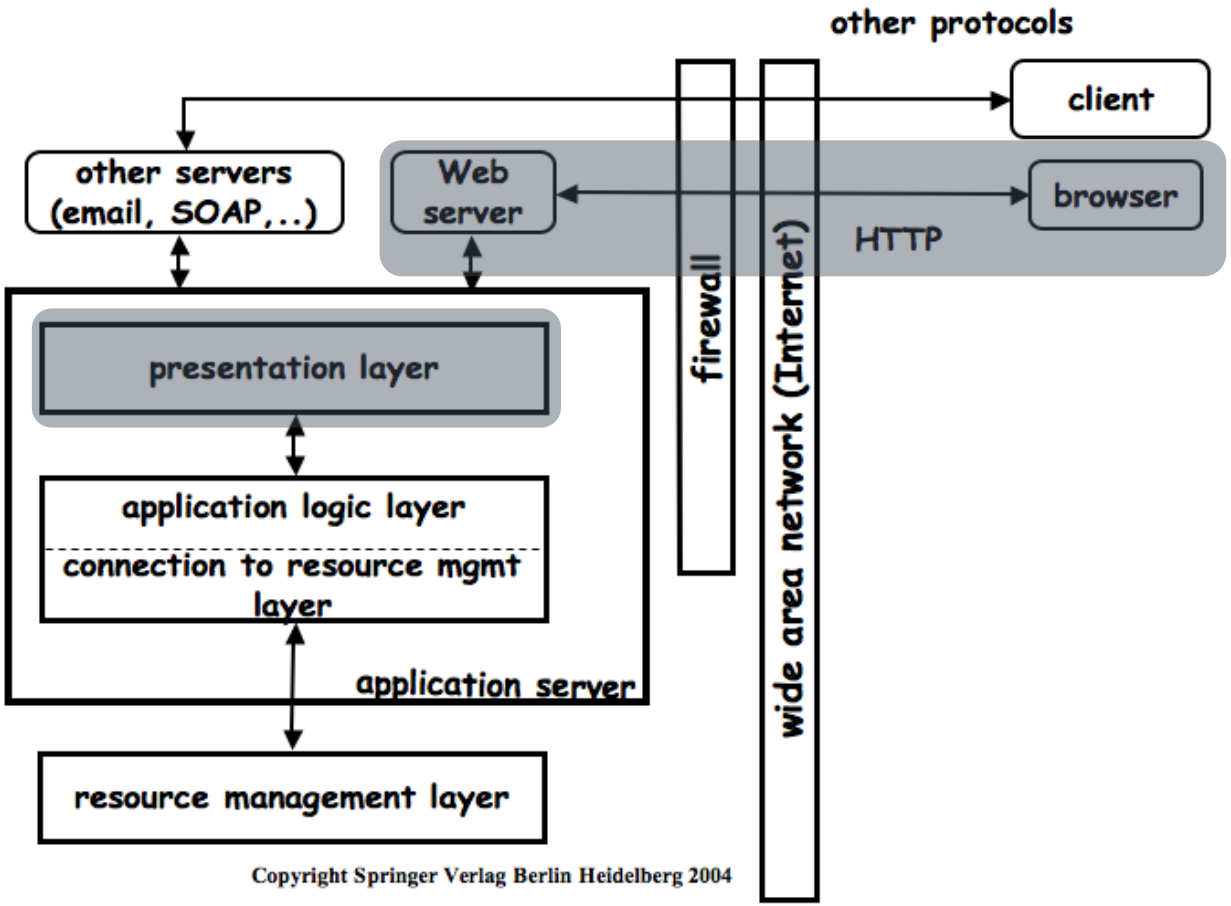
- Application server is a program that handles all application operations between users and an organization's backend business applications or databases

- Popular types of application servers, fall into three categories:
  - Java-based, including Oracle WebLogic and IBM WebSphere
  - Microsoft Windows Server-based
  - others, often built using open source technologies

Application Servers

- The purpose of an application server is to provide software abstractions for commonly used services
- Many application servers accept network requests from Web browsers and manage connections to large databases
- Typically found in business environments, application servers often run on the same network hardware as Web servers
- Application server support for the presentation layer is what differentiates app servers from other types of middleware

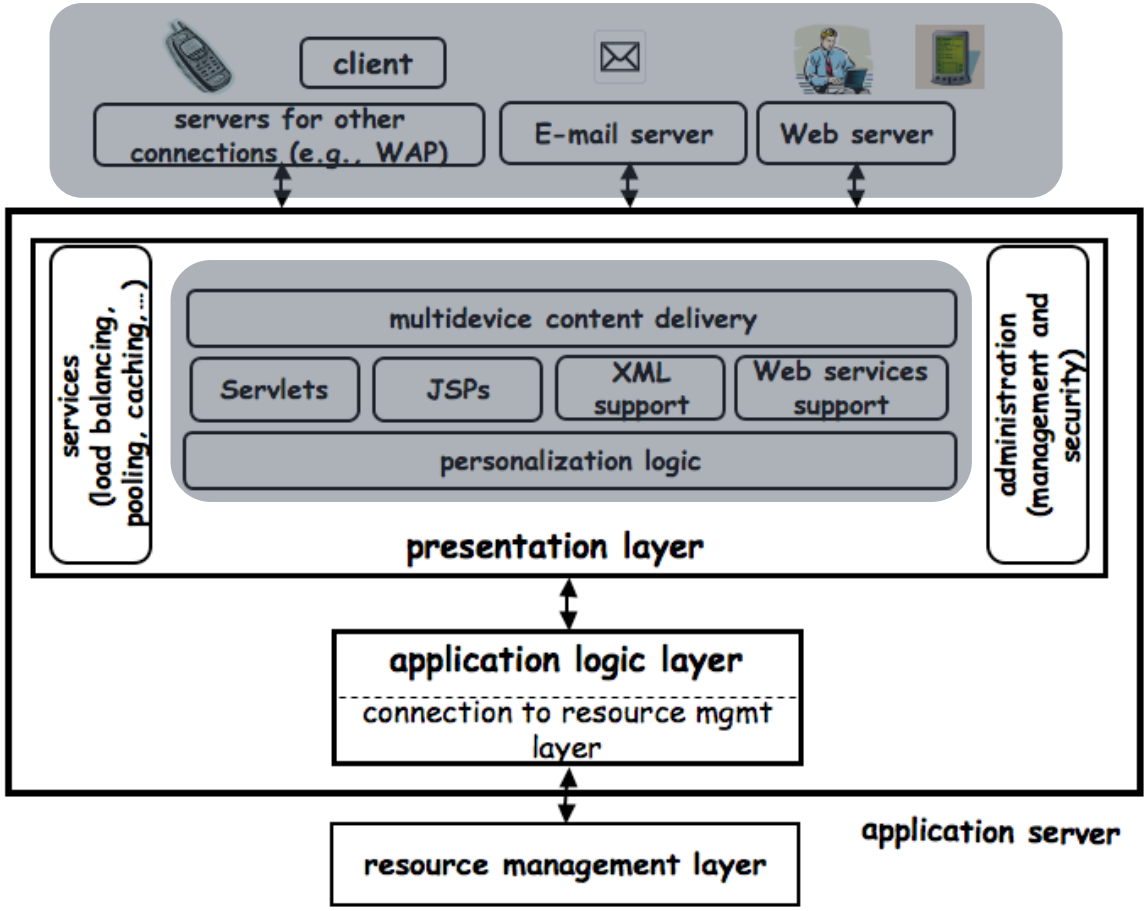
Application Servers



- Applications Servers are **middleware** for Web Applications
- The Web is the primary access channel
- The presentational layer has a wide range of technologies for delivery back to the client

Application Servers

- Application Servers can support:
  - Web browsers
  - Applications and devices
  - Email programs
  - Web services clients
- Presentational logic supports includes:
  - Multi-device content delivery
  - XML, JSPs, servlets
  - Personalization logic



Web Service Definitions

Generic:

- Any application accessible to other applications over the Web
  - Very open definition, almost any URL becomes a “Web service” under this definition

UDDI consortium:

- Web services are self-contained, modular business applications that have open, Internet-oriented, standards-based interfaces
  - Better, but still not specific enough; too much “business speak”

W3C:

- A Web service is a software application identified by a URI, whose interfaces and bindings are capable of being defined, described, and discovered as XML artifacts. A Web service supports direct interactions with other software agents using XML-based messages exchanged via Internet-based protocols
  - Slightly better, we now know that URIs and XML are involved (somehow) and that services are similar to components with interfaces that can be defined, described, and discovered

Web Service Definitions

Wikipedia:

- According to the W3C, a Web service is a software system designed to support interoperable machine-to-machine interaction over a network
- It has an interface that is described in a machine-friendly format such as WSDL
- Other systems interact with the Web service in a manner prescribed by its interface using messages, which may be enclosed in a SOAP envelope, or follow a REST approach
- These messages are typically conveyed using HTTP, and normally comprise XML in conjunction with other Web-related standards
- Software applications written in various programming languages and running on various platforms can use web services to exchange data over computer networks like the Internet in a manner similar to inter-process communication on a single computer
  - Much more specific, appears to reference a W3C definition

What is a Web Service?

- A service is software component at network-accessible endpoint. A Web service is a service accessible using common Web standards
  - Big services: SOAP, WSDL, UDDI
  - Light services: REST, HTTP, JSON

- A Web service is a software system designed to support interoperable machine-to-machine interaction over a network.
- It has an interface described in a machine-process able format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards

Web of Services

- Web of Services refers to message-based design frequently found on the Web and in enterprise software

- The Web of Services is based on technologies such as HTTP, XML, SOAP, WSDL, SPARQL and others
  - W3C

- Other systems interact with the Web service in a manner prescribed by its interface using messages, which may be enclosed in a SOAP envelope of follow a REST approach

Web of Services

- These messages are typically conveyed using HTTP, and normally comprise XML with other Web-standards
- Model of the web is that everything is a resource. Resources have names (URIs) and a location (URL)
- A Web client can use the information in a URL to locate and retrieve its associated resource

Web Services over the Network

RIA services can connect to services or external data:

- A Web service that can be discovered and consumed
- Talk to databases, or to a service with a business and data access layer which exposes the data
- UI/presentation logic typically runs on the client. Data management is usually run on the server. Application functionality may be on the client or on the server

Web Services over the Network

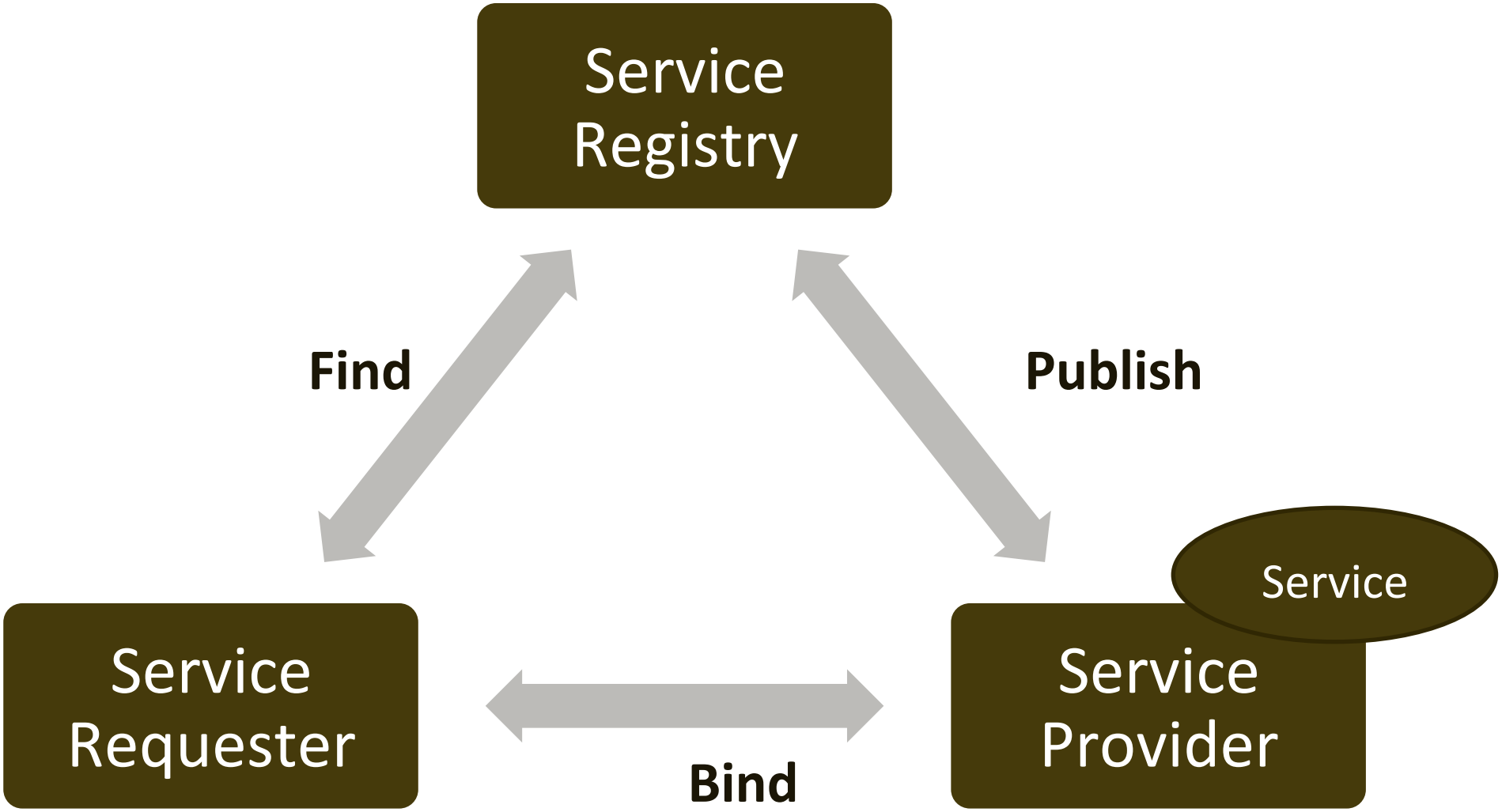
Networks:

- Connections over which computers can communicate
- A packet is a unit of transfer, sequence of bits carried over the network
- A protocol is an agreement as to how the data is transmitted. Depending on the application constraints for exchanging data across the Web, developers can choose among a series of protocols such as HTTP, SOAP and Web Services

Service-Oriented Architecture using Web Services standards

- SOA is based on the notion of externally provided services
- A web service is a standard approach to making a reusable component available and accessible across the web
- Service provision is independent of the application using the service
- A generic service:
  - An act or performance offered by one party to another. Although the process may be tied to a physical product, the performance is essentially intangible and does not normally result in the ownership of any of the factors of the product

Web Services SOA



Web Services SOA

- Based upon the interactions between three primary roles: service provider, service registry, and service requestor
- These roles interact using publish, find, and bind operations
- The service provider is the business that provides access to the Web service and publishes the service description in a service registry
- The service requestor finds the service description in a service registry and uses the information in the description to bind to a service
- In our image of the Web services architecture, the service registry provides a centralized location for storing service descriptions

SOA Characteristics

- Service provider defines how many users may be authorized to use it
- No shared objects or shared states
- Potentially, run-time binding
- Opportunistic construction of new services through composition
- Smaller, more compact applications
- Reactive and adaptive applications

SOA Benefits

- Services provide higher-level abstractions for organizing applications in large-scale, open environments – improves productivity and quality
- Standards enable the interoperation of software produced by different programmers
- Standards make it possible to develop general-purpose tools to manage the entire system lifecycle, including design, development, debugging, monitoring, and so on. These tools ensure that the components developed are interoperable, because vendors can validate their tools and shift part of the burden of validation from the application programmer
- The standards feed other standards. For example the above basic standards enable further standards, e.g., dealing with processes and transactions

Issues with Distributed Systems

- Resource Sharing
  - Managing the range of things that can be shared usefully in a distributed system
  - Sharing of hardware and software rescues
- Openness
  - A system can be extended in various ways: hardware or software extensions
  - Use of equipment and software from different vendors
- Concurrency
  - Concurrency processing to enhance processing
  - Synchronization between different users and server processes
- Scalability
  - Increased throughput by adding new resources
  - Many different size systems, not a question of scaling, but rather scaling well

Issues with Distributed Systems

- Fault Tolerance
  - The ability to continue in operation after a fault has occurred
- Complexity
  - Typically distributed systems are more complex than centralized systems
- Security
  - More susceptible to external attack
- Manageability
  - More effort required for system management
- Unpredictability
  - Unpredictable response depending on the system organization and network load

Web Service Protocols

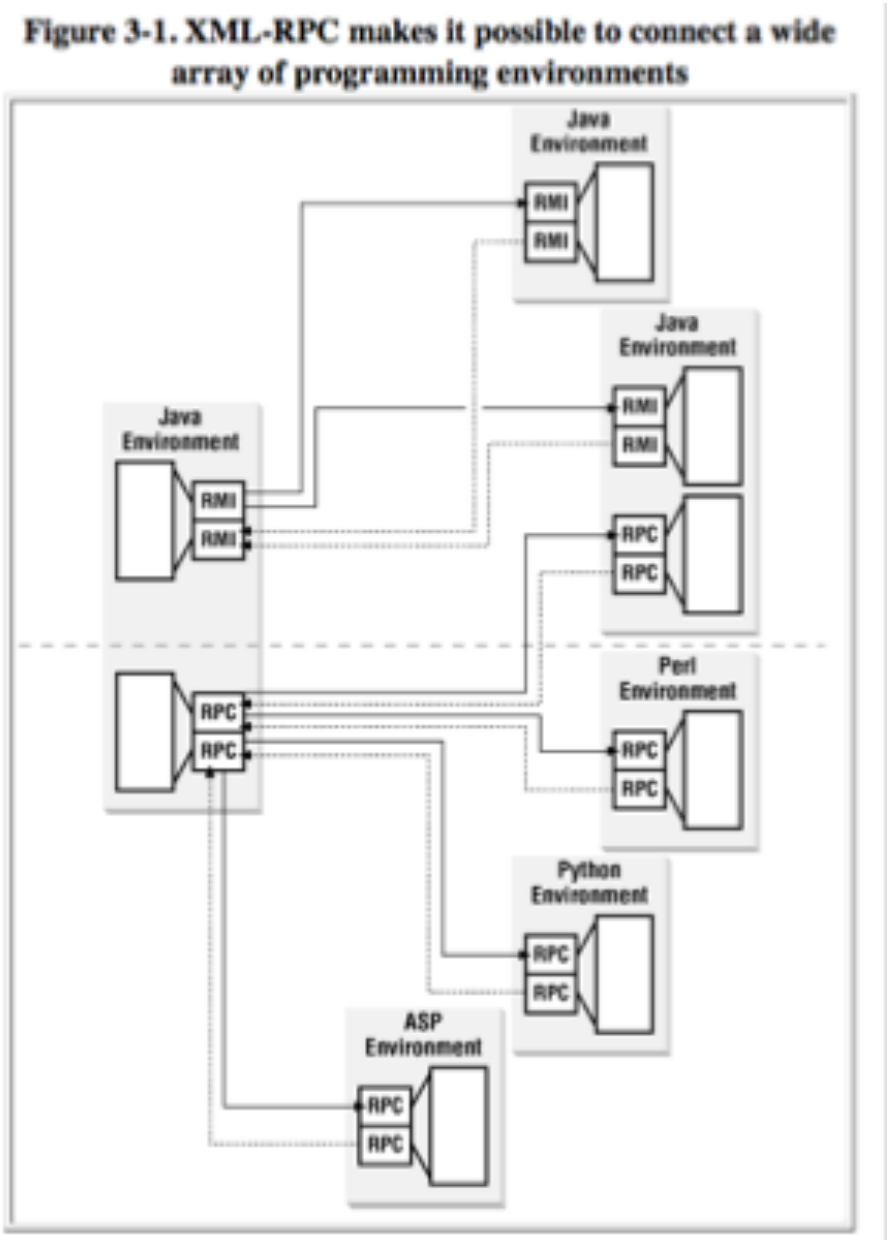
- Enable communication and consumption of services
  - Connect to self-describing services
  - A service that exposes information about its methods and data types
- Web Service Standards we use with RIAs:
  - Message transfer (SOAP, HTTP)
  - Data Structure (XML, XSD)
  - Interface description (WSDL/text)
  - Service Discovery (UDDI/hyperlinks)
  - Security (HTTP/WS-Security)
  - Reliability (WS-Reliability)
  - Transactions (WS-Transactions)

Plain Old XML

- Simple, well-formed and valid XML documents and XML fragments can be sent and received between services
- Gives the developer control over structure of the schema of the XML being sent
- Useful for scenario where a service being consumed does not fully support all Web standards
  - XML can be used to exchange data with the service

RPC

- Remote Procedure Call
  - Call a procedure that is running on another machine
  - Identify and access remote procedure
  - Parameters
  - Return Values
- Application-Oriented Design
  - Divide program into logical components and add communication protocols



SOAP

- Simple Object Access Protocol

- XML-based application-layer protocol for constructing and processing web service requests and responses
  - exchanging information between computers

- Today's applications communicate using Remote Procedure Calls (RPC) between objects like DCOM and CORBA, but HTTP was not designed for this

- A better way to communicate between applications is over HTTP, because HTTP is supported by all Internet browsers and servers
  - SOAP was created to accomplish this

SOAP

- Originally XML-RPC
- SOAP is a lightweight protocol

- Intended for exchanging structured information in a decentralized distributed environment
- Uses XML technologies to define an extensible messaging framework, this provides a message construct that can be exchanged over a variety of underlying protocols
- The framework has been designed to be independent of any particular programming model and implementation-specific semantics

SOAP

- RPC over HTTP via XML
  - Data encoded in XML for communication over HTTP (SMTP, TCP/IP)
  - Transport bindings over these protocols
  - Implements RPC interaction pattern
  - Defines how clients talk to remote server
  - Maintained by W3C XLM working group

- EAI (Enterprise App Integration) as a Web Service
  - XML + SOAP + WSDL = basic web service

SOAP as Messaging Protocol

- Stateless, one-way message exchange paradigm
- Provides a framework for application-specific information to be conveyed in an extensible manner
- Provides a full description of the required actions taken by SOAP node on receiving a SOAP message
- Three main components:
  - A generic XML messaging framework
  - A data encoding standard or set of encoding rules
  - An RPC (remote procedure call) framework
- Possible to use just the messaging framework or messaging framework/encoding standards without using RPC mechanism

SOAP messaging Framework

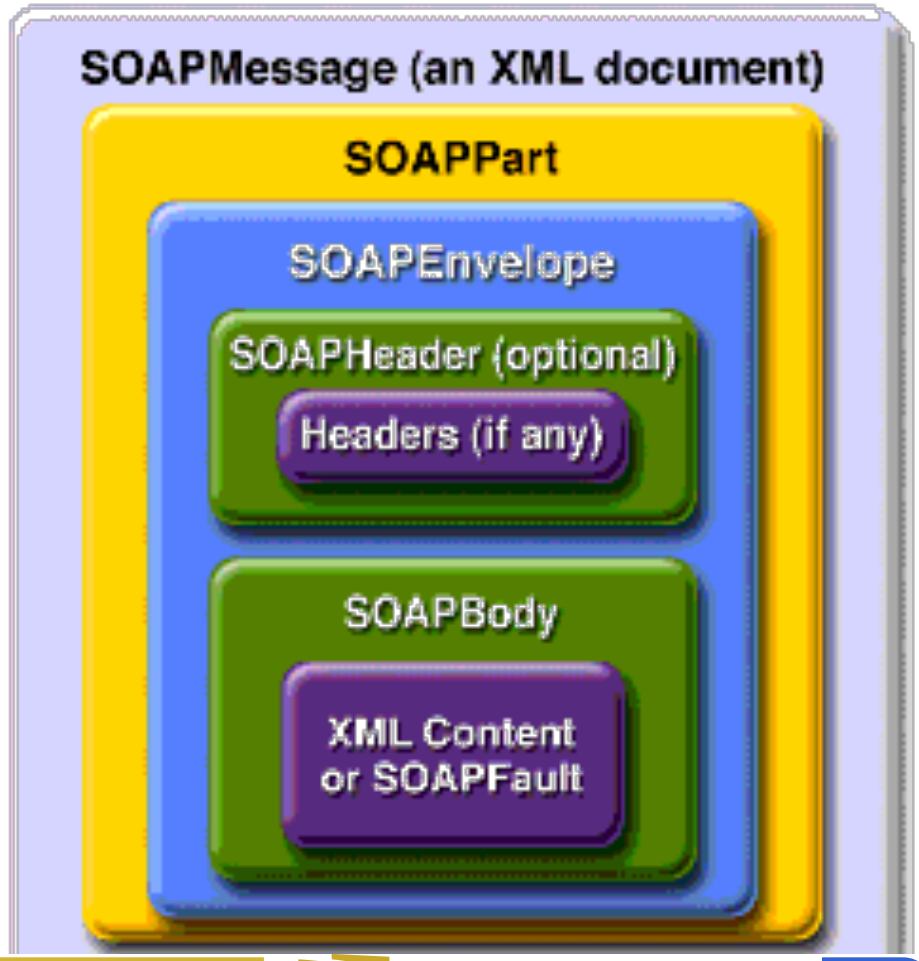
- Defines a generic message XML schema
- Package message as SOAP
- RPC mechanism
- Messages are transmitted using existing protocols
  - HTTP, HTTPS, SMTP, GSI, sockets
- SOAP defines bindings to different protocols that specify how SOAP is used with that protocol to send messages.

SOAP messaging Framework

- A SOAP message needs to be able to describe the data that it is carrying to the receiver
  - Refer to a schema from within the SOAP message and use regular xs types (“literal”)
  - Using soap encoding (or some other encoding standard) directly within the XML (“encoded”)

Structure of SOAP message

- Envelope
  - Defines message content
  - Identifies the XML document as a SOAP message
- Header (optional)
  - Destination information
  - Versioning
  - Extensions
  - Security
- Body
  - Payload document
  - Contains call and response information
  - Optional Fault element



SOAP encoding

- SOAP includes a set of rules for encoding data types. This enables the SOAP message to indicate specific data types, such as integers, floats, doubles, or arrays
- The serialization of data inside a SOAP message is referred to as encoding
- The encodingStyle attribute defined by the SOAP specification is used to identify the encoding rules used in a particular message
- SOAP does this in a language agnostic way
- If the encodingStyle attribute does not appear in the message, the receiver cannot make assumptions about how data will be represented within the message

SOAP encoding – XML schemas

- Encoding can make use of XML schemas
- The SOAP specification defines a single set of encoding rules called SOAP encoding
- Encoding rules are identified by a URI. This allows developers who do not need the capabilities of XML schemas to forego their use and start sending messages with encoding rules based on an accepted URI
  - http://schemas.xmlsoap.org/soap/encoding/
  - http://www.w3.org/2001/12/soap-encoding
- For examples, SOAP representation of an array of integers:

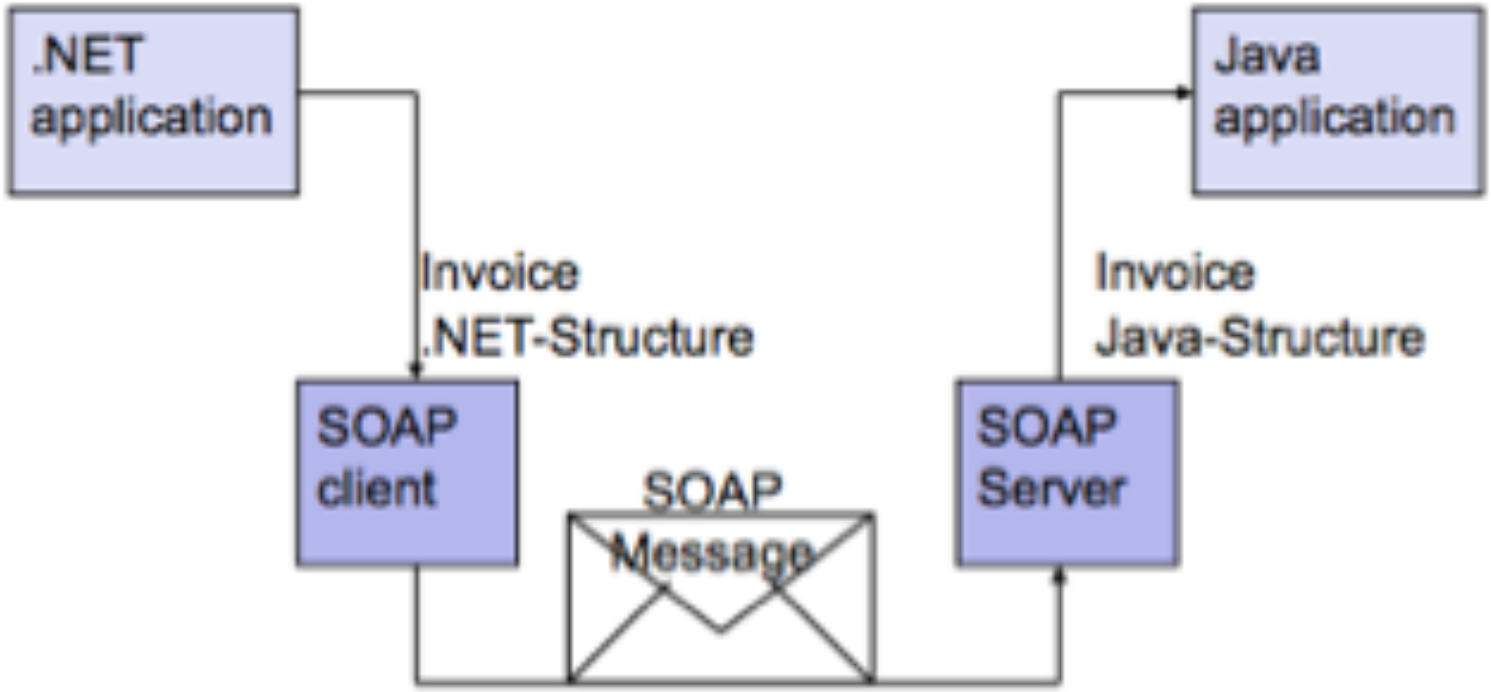
SOAP encoding – XML schemas

```
<SOAP-ENC:Array SOAP-ENC:arrayType="xsd:int[3]">
<SOAP-ENC:int>8</SOAP-ENC:int>
<SOAP-ENC:int>5</SOAP-ENC:int>
<SOAP-ENC:int>9</SOAP-ENC:int>
</SOAP-ENC:Array>
```

SOAP with RPC

- An RPC mechanism turns messages into method calls
- SOAP is just a generic message envelope
- Generic message structure + data – The message is turned into a function call automatically
- Augmented with encoding style and RPC rules – middleware layer for remote procedure calls

SOAP with RPC

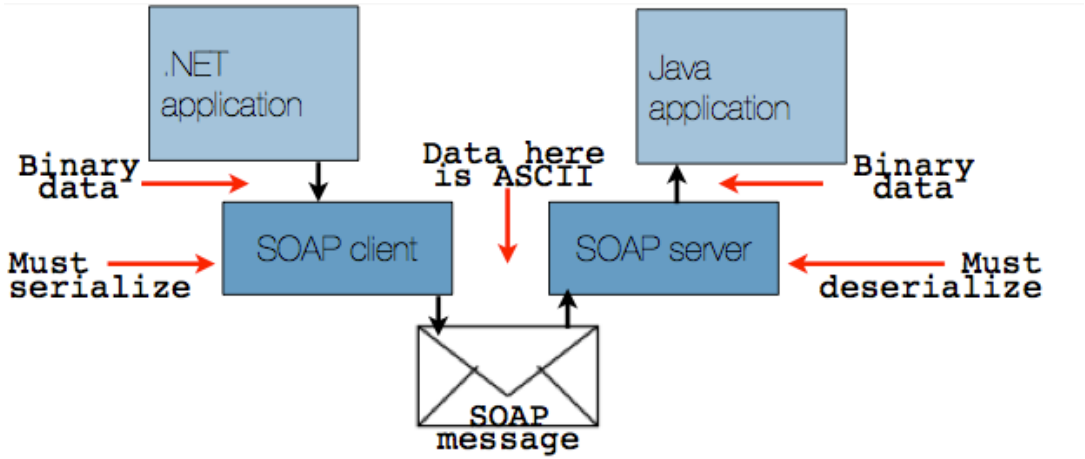


Serialization

- In traditional Web applications, communication between the clients and server is almost always initiated by the client. The messages sent are simple text. How do complex elements like binary data, images, graphs, animations, and multimedia get transported?
- The process of converting a complex item to a text representation is called serialization
- The process of converting a serialized, text representation back to the original object is called deserialization
- The most common formats for an item to be converted to for transportation over the Web are XML, SOAP, JSON. Many frameworks include facilities for serializing and deserializing data

SOAP Serialization

- To be interoperable, use XML
- XML is ASCII, not binary
- End points use binary
- Must serialize and deserialize (or marshaling/ unmarshaling)



SOAP Transport

SOAP messages are transmitted using existing protocols

- SOAP is transport independent
- HTTP, SMTP, GSI, HTTPS, pure sockets
- HTTP is the default binding
- Use these protocols security model

SOAP over HTTP

- A binding of SOAP to a transport protocol is a description of how a SOAP message is sent using the transport protocol
- The typical binding for SOAP is HTTP
- SOAP can use GET or POST
  - When using GET the request is not a SOAP message, but the response is
  - With POST both the request and response are SOAP messages
- SOAP uses the same error and status codes as those used in HTTP so that HTTP responses can be directly interpreted by a SOAP module

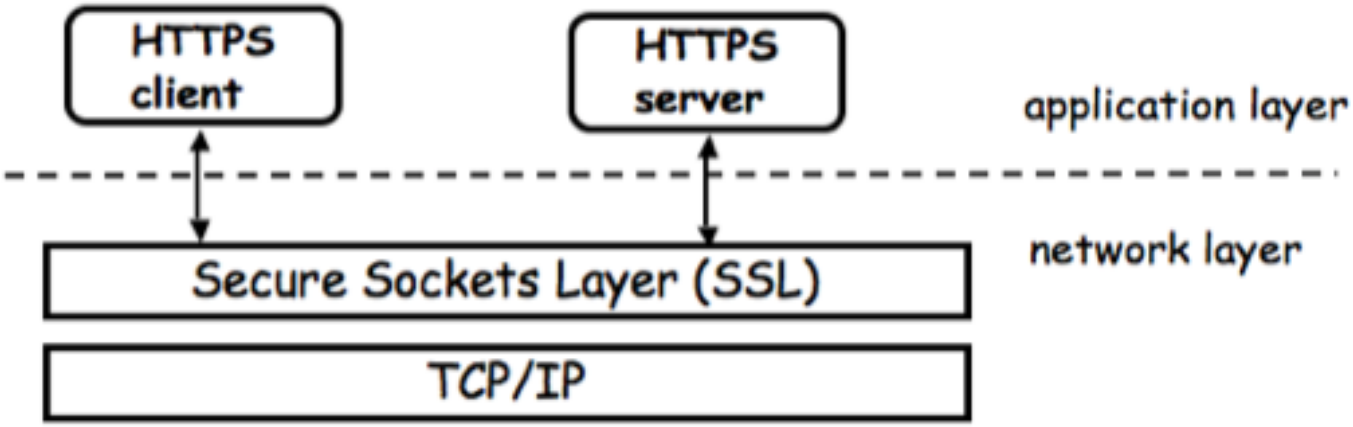
HTTP Refresher

- HTTP defines various methods that can be applied to a resource. These include:
  - GET – retrieve a resource
  - POST – create a resource
  - HEAD – retrieve metadata about a resource
  - PUT – add/create information to a resource
  - DELETE – delete an existing resource

- Web services/RESTful applications are founded on these methods

Transport over HTTP

- HTTP does not encrypt data before sending it.
- Secure sockets layer over TCP/IP to establish encryption and identity assurance
  - Digital document or passport to verify the security and authenticity of interaction
  - SSL certificate installed on web server to identify the business using it to encrypt sensitive data.

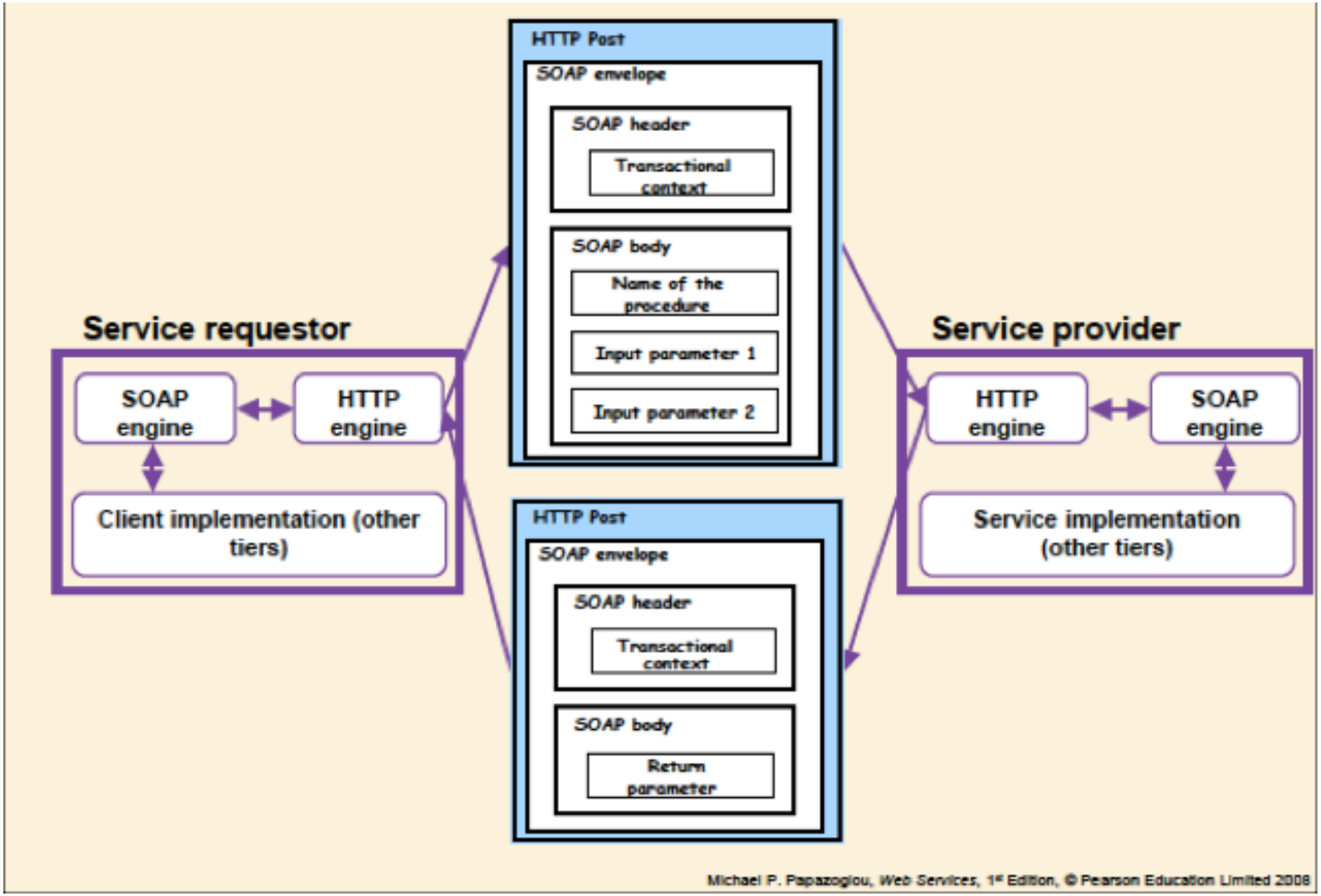


HTTP

```
GET / HTTP/1.1
Host: twitter.com
User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.6; en-US; rv:1.9.2.15) Gecko/20110303 Firefox/3.6.
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 115
Connection: keep-alive
Cookie: k=89.101.207.107.1299271270313890; guest_id=12992712703195709 3; __utma=43838368.2089094197.12992712
```

```
POST /1/statuses/update.json HTTP/1.1
Host: api.twitter.com
User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.6; en-US; rv:1.9.2.15) Gecko/20110303 Firefox/3.6.
Accept: application/json, text/javascript, */*
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 115
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
X-PHX: true
Referer: http://api.twitter.com/p_receiver.html
Content-Length: 183
Cookie: k=89.101.207.107.1299271270313890; __utma=43838368.2089094197.1299271272.1299616950.1299692588.16;
Pragma: no-cache
Cache-Control: no-cache
```

RPC call using SOAP over HTTP



Sample SOAP message for travel reservation (from W3C)

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
 <env:Header>
  <m:reservation xmlns:m="http://travelcompany.example.org/reservation"
          env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
           env:mustUnderstand="true">
   <m:reference>uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d</m:reference>
   <m:dateAndTime>2001-11-29T13:20:00.000-05:00</m:dateAndTime>
  </m:reservation>
  <n:passenger xmlns:n="http://mycompany.example.com/employees"
          env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
           env:mustUnderstand="true">
   <n:name>Åke Jógvan Øyvind</n:name>
  </n:passenger>
 </env:Header>
```

Sample SOAP message for travel reservation (from W3C)

```
<env:Body>
  <p:itinerary
    xmlns:p="http://travelcompany.example.org/reservation/travel">
   <p:departure>
     <p:departing>New York</p:departing>
     <p:arriving>Los Angeles</p:arriving>
     <p:departureDate>2001-12-14</p:departureDate>
     <p:departureTime>late afternoon</p:departureTime>
     <p:seatPreference>aisle</p:seatPreference>
   </p:departure>
   <p:return>
     <p:departing>Los Angeles</p:departing>
     <p:arriving>New York</p:arriving>
     <p:departureDate>2001-12-20</p:departureDate>
     <p:departureTime>mid-morning</p:departureTime>
     <p:seatPreference/>
   </p:return>
  </p:itinerary>
  <q:lodging
   xmlns:q="http://travelcompany.example.org/reservation/hotels">
   <q:preference>none</q:preference>
  </q:lodging>
 </env:Body>
</env:Envelope>
```

Sample SOAP reply

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
 <env:Header>
  <m:reservation xmlns:m="http://travelcompany.example.org/reservation"
      env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
           env:mustUnderstand="true">
   <m:reference>uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d</m:reference>
   <m:dateAndTime>2001-11-29T13:35:00.000-05:00</m:dateAndTime>
  </m:reservation>
  <n:passenger xmlns:n="http://mycompany.example.com/employees"
      env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
           env:mustUnderstand="true">
   <n:name>Åke Jógvan Øyvind</n:name>
  </n:passenger>
 </env:Header>
```

Sample SOAP reply

```
<env:Body>
 <p:itineraryClarification
   xmlns:p="http://travelcompany.example.org/reservation/travel">
  <p:departure>
    <p:departing>
      <p:airportChoices>
         JFK LGA EWR
      </p:airportChoices>
    </p:departing>
  </p:departure>
  <p:return>
    <p:arriving>
      <p:airportChoices>
        JFK LGA EWR
      </p:airportChoices>
    </p:arriving>
  </p:return>
 </p:itineraryClarification>
 </env:Body>
</env:Envelope>
```

REST

- Stages of RESTful web services
  - A network of web pages (virtual state-machine)
  - User progresses through the application by selecting links (state transition)
  - Resulting in the next page being transferred to the client (representing the next state of the application)
  - Rendered for use

- Five key principles:
  1. Give every “thing” an ID
  2. Link things together
  3. Use standard methods
  4. Resources with multiple representations
  5. Communicate statelessly

REST: Representational State Transfer

- The Web is a set of hyperlinked resources
  - Resources: items identified by URIs
- Accessing a link returns a resource (e.g. HTML, .jpg)
  - Resource puts client in some state
  - Traversing hyperlinks: accessing another resource
  - Traversing hyperlinks changes client's state
- Web application: state machine
  - States: resource representations
  - Transitions: hyperlink traversal
- REST makes use of the architectural style of the Web

REST Guidelines: Resource Identification through URI

- Every interesting resource has its own unique URI, which can be used to request an instance of the resource
  - E.g., http://example.org/users/
  - A URI that might return a list of users. The URI is the identifier. The resulting list of users is the resource

- A resource is not a 'storage object' but a conceptual entity - resources simply represent some identified item that can be accessed
- The identifier (URI) is passed from the client to the server where it is resolved into a resource that is returned from the server to the client

REST Guidelines - Uniform Interface for all resources

- GET (Query the state, idempotent, can be cached)
  - Used to retrieve a representation from a resource
  - Widely used today for downloading web pages but it might be used to GET any kind whatever representation a resource may provide, including XML data

- PUT (Transfer the state on existing/new resource)
  - Used to store a representation at a resource
  - Can be used to upload a file and have it stored at a particular URI at the web server or create a user account on a web forum
  - If you issue a PUT two times, then the second representation overwrites the first. This is fine if a file is uploaded two times but can cause problems if two different users issue a PUT to the same URI

REST Guidelines - Uniform Interface for all resources

- POST (Update a resource or create child resource)
  - Similar to PUT, POST also creates a resource. The key difference is that POST is used when the server is in control of storing information, not the client
  - This is usually the case when posting a blog entry, for example. If the client wishes to create a new blog entry, it is the server that is responsible for storing the information in a database and assigning a unique value and/or URI to the newly posted content
  - Also the appropriate tool to use when the operation results in an action other than the creation of a resource. E.g., a web-based form to transmit an email message

- DELETE
  - Deletes information that is stored
  - If there is a resource that you would like to remove from the server, DELETE is the method to use

REST Guidelines - Self-Descriptive Message Representations

- Client requests and server responses are messages and RESTful applications expect each message to the self-descriptive
- That means each message contains all the information necessary to complete the task e.g., which parser to use, whether the message can be cached
- Also called "state-less" or "context-free."
- Each message passed between client and server can have a body (or 'entity body') and metadata

REST Guidelines - Hyperlinks

- Use hyperlinks to define relationships between resources and valid state transitions of the service interaction

- Clients make state transitions only through actions that are dynamically identified within hypermedia by the server
  - (e.g., by hyperlinks within hypertext)

RESTful URIs

- RESTful applications do not implement protocols such as SOAP or XML-RPC
- There is no validation service to tell if the service is REST compliant
- RESTful applications are applications that follow the RESTful conventions
- REST defines the identity of a resource via URI
- Cool URIs do not change
  - http://www.w3.org/Provider/Style/URI

REST advocates ‘nice’ URIs

- Each resource has a unique address in URL form (i.e. using the http protocol).
  - E.g. A twitter status feed resource URI is https://twitter.com/stephenfry
- When designing the URI’s make them easy for users to work with. They should:
  - Be as short as possible
  - Let the user move up the tree e.g. /cars/toyota/yaris – if you move up the tree you should see all the Toyotas
  - Meaningful and predictable
  - Consistent
  - Use plural path for collections
  - Avoid extensions
  - Stateless

REST Principles

- Characteristics of RESTful applications
  - Client-server separation
  - Stateless
  - Cacheable
  - Layered System
  - Uniform interface between client and server
  - Code on demand
- The Web is an instance of REST
  - Stateless Protocol (HTTP)
  - Uniquely Addressable Resource (URIs)
  - Well-defined Operations (HTTP methods defined in HTTP spec)
  - User of hypermedia (HTML)
- Architectural Elements
  - Data Elements
  - Connectors
  - Components

REST Characteristics

- Client-Server
  - Emphasizes the separation of components
  - Separate the user interface (client) form the data storage (server)
  - Allows components to evolve independently

- Stateless Server
  - Each request from the client to server must contain all of the information necessary to understand the request
  - Cannot take advantage of any stored context on the server
  - Session state entirely on the client

REST Characteristics

- Caches
  - Improves network efficiency
  - Requires that the data within a response to a request be labeled as cacheable or non-cacheable
  - If cacheable, then client cache given right to reuse response data for later requests

- Uniform Interface
  - Principles of generality to the component interface. Systems architecture is simplified and visibility of interactions improved
  - Implementations are decoupled from the services they provide – independent, can evolve

REST Characteristics

- Layers
  - Constrain component behavior so that each component cannot see beyond the immediate layer with which they are interacting
  - Client cannot tell whether it is connected directly to the end server, or to an intermediary
  - Places bound on overall system complexity and promotes substrate independence as well as security

- Code on Demand (optional)
  - Client functionality extend by downloading and executing code in the form of applets, scripts (Javascript)
  - Simplifies clients by reducing features required to pre-implement
  - Improves system extensibility, but reduces visibility
  - Only optional constraint

REST Data Elements

- When a link is selected, information needs to be moved from the location where it is stored to the location where it will be used
- Handle the main design issues of moving information
- Separating concerns of client and server
- Ways of dealing with data
  - Render the data where it is located and send a fixed-format to the recipient
  - Encapsulate the data with rendering engine and send both
  - Send the raw data to the recipient along with metadata that describes the data type, so that the recipient can choose own rendering engine

REST Data Elements

Data Elements	Examples
resource	The intended conceptual target of a hypertext reference
resource identifier	URL, URN
representation	HTML document, JPEG image
representation metadata	Media type, last modified time
resource metadata	Source link, alternates, varies
control data	If-modified-since, cache-control

REST Connector Elements

- Present an abstract interface for component communication
- A connector manages a network communication for a component
- REST connectors support stateless interactions
- No need for connectors to retain application state between requests
- Interactions can be processed in parallel
- Intermediaries can understand requests in isolation
- Information that factors into cache behavior is present

REST Connector Elements

Connectors	Examples
client	Libwww, libwww-perl
server	Apache API, libwww
cache	Browser cache, Akamai cache network
resolver	Binding (DNS lookup library)
tunnel	SOCKS, SSL after HTTP CONNECT

REST Component Elements

- Origin Server
  - E.g. Apache HTTPD, MS IIS, Glassfish
- Client
  - Firefox and other Web browsers
- Proxy
  - Netscape proxy
  - Selected by client
- Gateway
  - CGI, squid, reverse proxy
  - Controlled by the server

REST examples

- Create RESTful applications
  - Web Monkey from Wired
    - Intro to REST
    - http://www.webmonkey.com/2010/02/get_started_with_rest/

- Examples of sites with REST APIs
  - Google Geocoder via HTTP
  - Yahoo HTTP Geocoder
  - Twitter
  - Delicious
  - http://www.programmableweb.com/apis/directory/1?protocol=REST&sort=mashups

Consuming a RESTful API

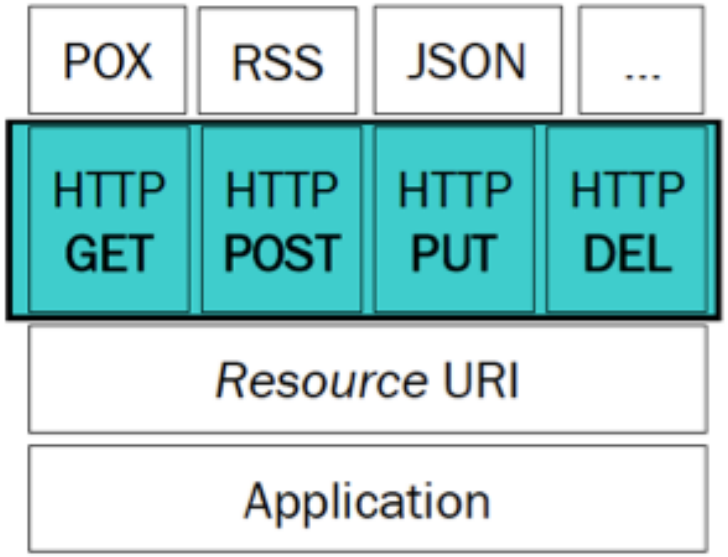
- Flickr – http://www.flickr.com/services/api/
- The REST Endpoint URL is
  - http://api.flickr.com/services/rest
- 1. Apply for your own API key
  - http://www.flickr.com/services/apps/create
- 2. Make a call to flickr.photos.getRecent
  - http://www.flickr.com/services/api/flickr.photos.getRecent.html
  - You should make a call to something like:
  - http://api.flickr.com/services/rest/?method=flickr.photos.getRecent&api_key=YOURKEY&format=rest
- 3. Change the returned format to JSON
- 4. How would you display these images?

REST v SOAP

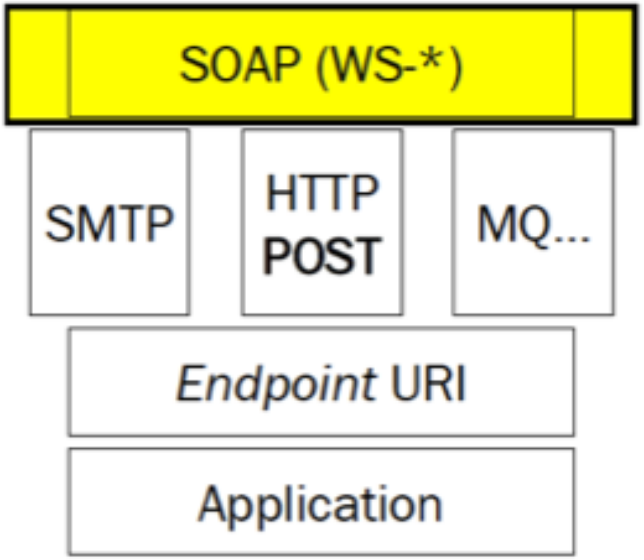
- Focus on whatever solution gets the job done rather than sticking with specific architectures or technologies
- WS-* and REST both have strengths and weaknesses and will be suitable to some applications and not for others
- The decision of which to use depends entirely on the application requirements and constraints

Protocol Layering

- "The Web is the universe of globally accessible information" (Tim Berners Lee)
  - Applications should publish their data on the Web (through URI)



- "The Web is the universal (tunneling) transport for messages"
  - Applications get a chance to interact but they remain "outside of the Web"



- Cesare Pautasso, Olaf Zimmermann, Frank Leymann, RESTful Web Services vs. Big Web Services: Making the Right Architectural Decision, Proc. of the 17th International World Wide Web Conference (WWW2008), Bejing, China, April 2008

National College of Ireland

Questions?

Mikhail Timofeev
Office 3.18
MTimofeev@ncirl.ie