

Lecture 8 : SOA and Web Services

This Week

- RMI, EJB and many other distributed computing frameworks suffer from many disadvantages.
- This week we'll look at Service Oriented Architecture which is an architecture proposed to solve many of these disadvantages
- Also we'll look at SOAP and REST as Web Service implementations

Issues with Traditional RPC

- The RPC frameworks we have discussed so far share a few common issues that tend to inter-relate
 - Tight coupling between client and server
 - Security problems:
 - Trust,
 - Firewalls
 - The Internet
 - Limited/non-existent interoperability between frameworks

Issue: Coupling

- Client and server in RPC are typically viewed as two parts in one (distributed) application
 - Stubs/Skeletons are generated from the same IDL file / interface
 - Marshalling/Serialization is technology dependent
 - Implicitly creates a coupling between client and server

Issue: Trust and Firewall Security

- Trust issues:
 - The server shares it's information with the client
 - The client can compromise the server
- Firewall security: The RPC frameworks advocate assigning each server component with its own port
 - Follows good network protocol design - each different service has its own port (e.g. ftp = port 21, http = port 80)
 - Firewalls are then configured to block access to dangerous/risky ports to minimise the risk of attacks

Issue: Internet Security

- When the RPC is only internal to a corporate network, setup and security is less of an issue
 - Physical + login security cuts out most attack vectors
- But what if we must communicate over the Internet?
 - Need to open a 'hole' in the firewall at the *gateway*, one hole for each server component inside the network
 - Network administrators are *very* reluctant to do this!

Issue: Interoperability

- Most RPC frameworks don't interoperate with other frameworks
 - Almost entirely down to incompatible communication protocols and message formats
 - E.g. IIOP for CORBA vs. MSRPC for DCOM,
 - Each protocol is tailored to the features of the framework it was designed for
 - Presently, RPC became well-understood enough to define stable protocol and message format standards

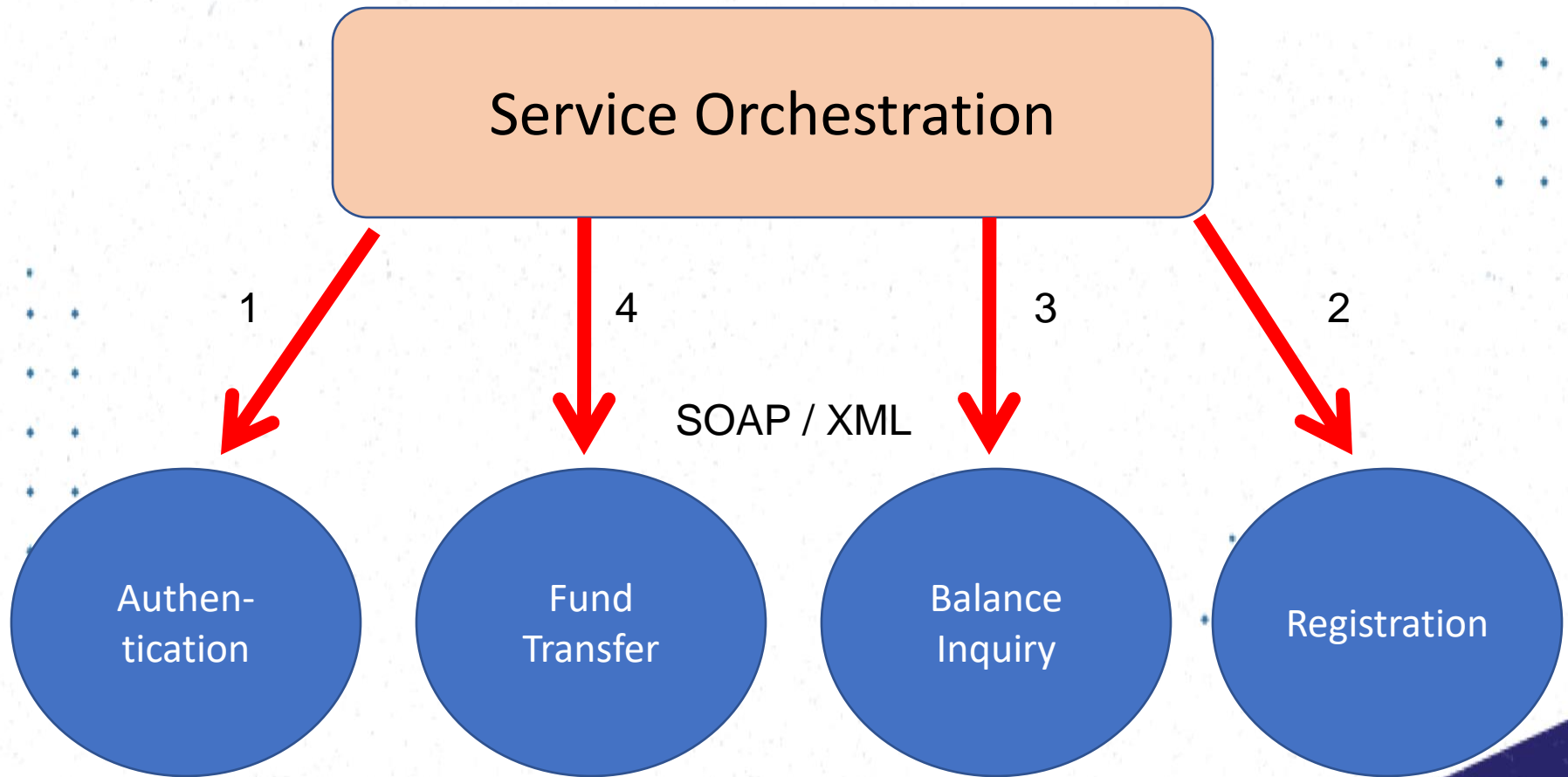
Software Architectures

- “The software architecture of a program or computing system is the structure or structures of the system, which include software components and the relationships among them.”
- In other words, software architecture describes the system’s components and the way they interact at a high level.
- Service-oriented architecture is a special kind of software architecture that has several unique characteristics

What is SOA?

- Service-oriented architecture (SOA) is an architectural style where existing or new functionalities are grouped into atomic services.
- SOA is commonly thought as an architecture that builds **loosely coupled, interoperable, Standard based** components called services.
- They typically implement functionalities most humans would recognize as a service
 - Filling out an online application for an account
 - Viewing an online bank statement
 - Placing an online book or airline ticket order.

What is SOA?



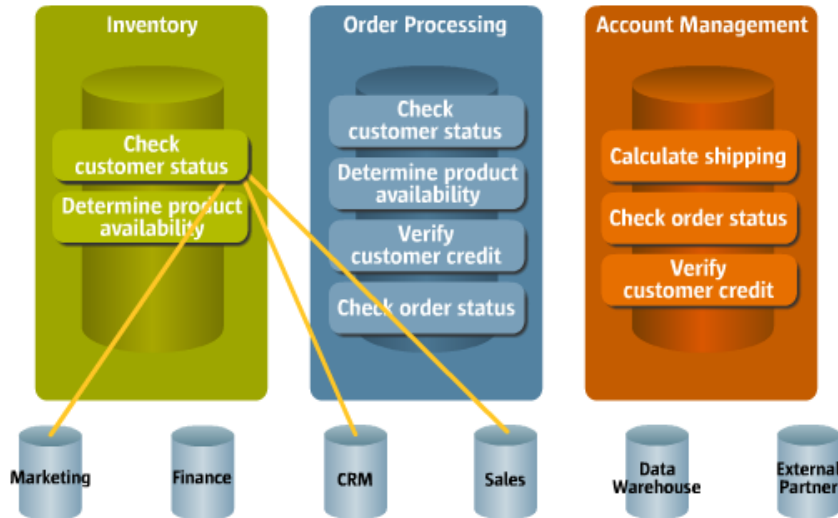
Services

What is SOA?

- They have no calls to each other embedded in them.
- Instead of services embedding calls to each other in their source code, protocols are defined which describe how one or more services can talk to each other.
- This architecture then relies on a business process expert to link and sequence services, in a process known as **orchestration**, to meet a new or existing business system requirement.

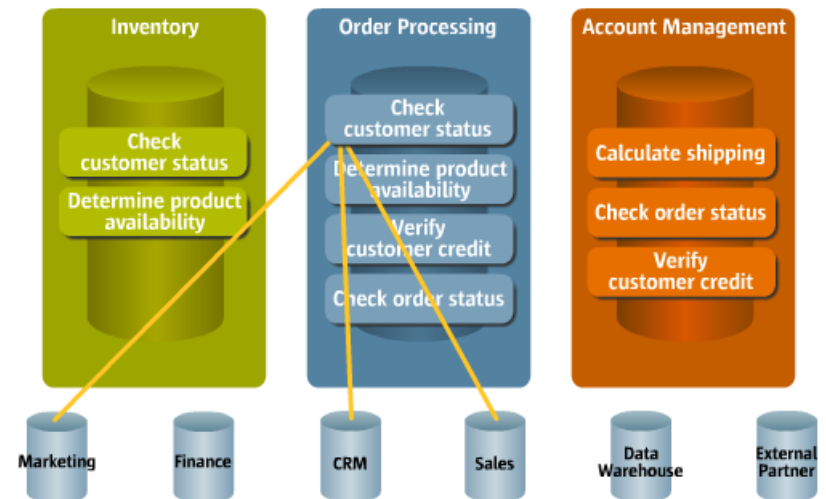
Traditional Distributed Systems

- Functions are duplicated



Order Processing also needs checking customer status

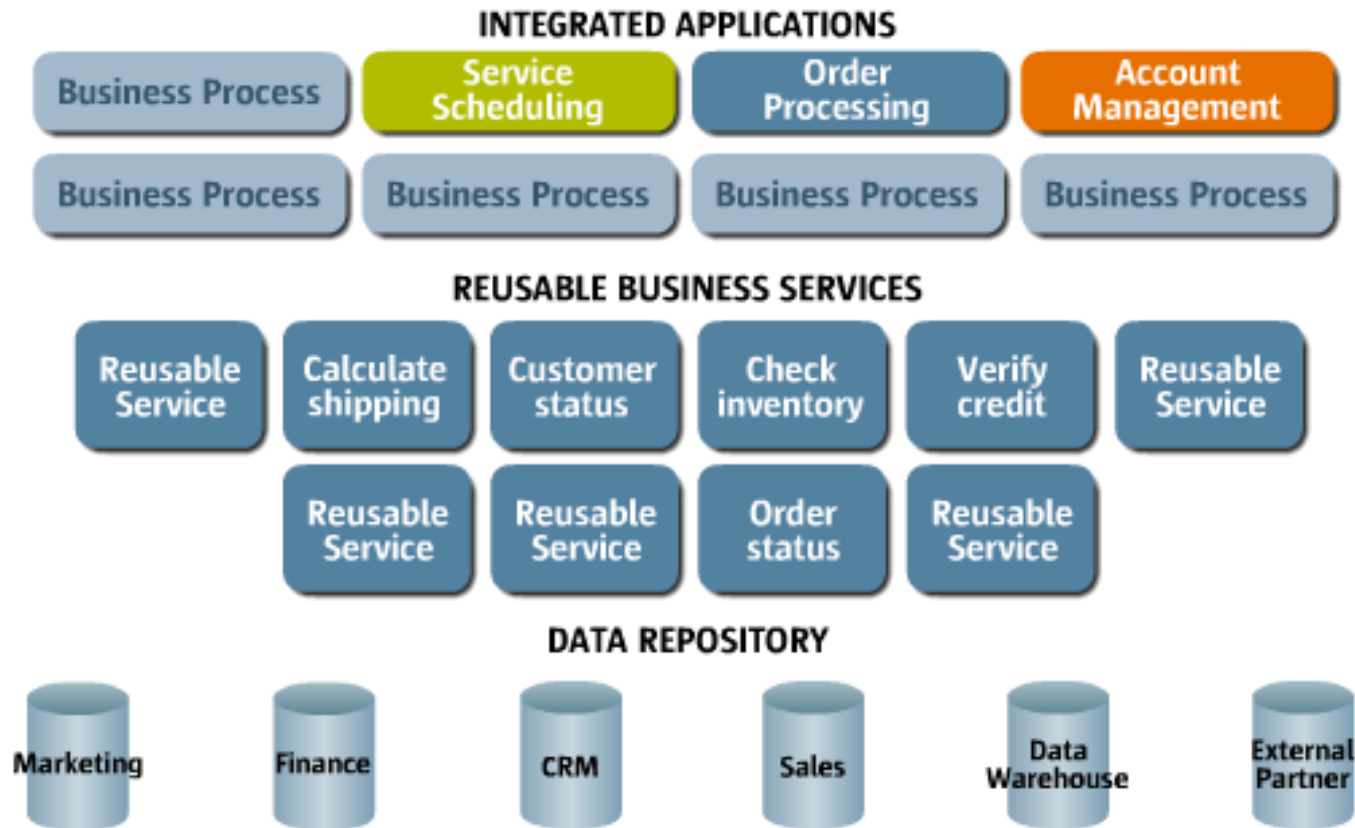
Inventory Processing needs checking customer status



Traditional Distributed Systems

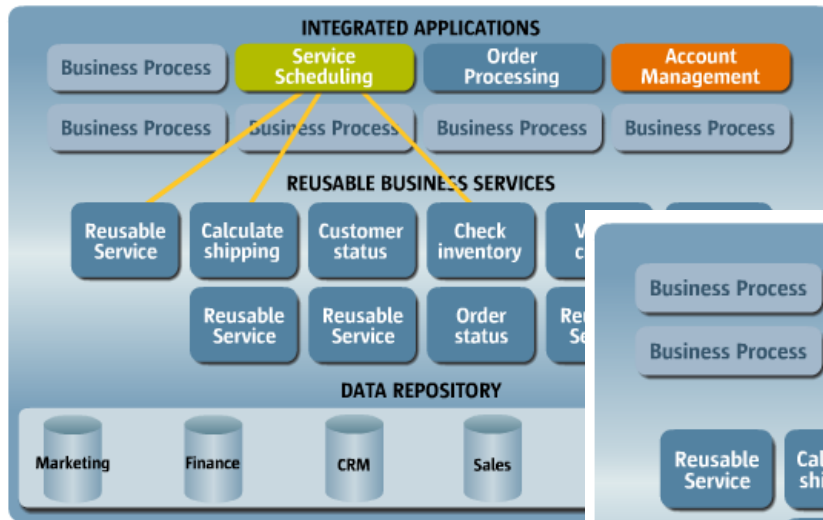
- Low reusability
 - If you try to reuse lot of cross references between sub-systems.
- Adding a new function is difficult
 - Develop everything from the beginning
- Function inconsistency
 - Development of “Checking customer status” can be different from sub-systems to sub-system.

SOA Style



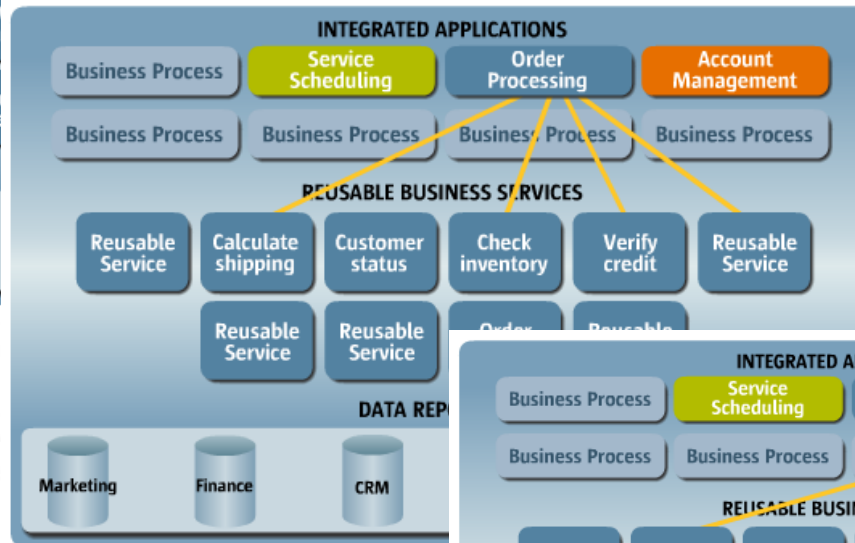
New ERP System is a reusable collection of services, that can be composed into Integrated Applications.

SOA Style



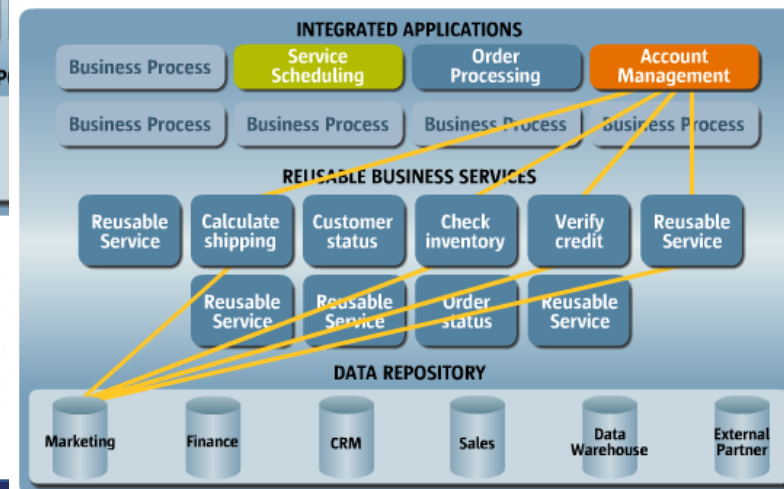
Service Scheduling Application

Order Processing Application



Management Report

Generation Application



SOA

- What distinguishes SOA from other architectures is loosely coupling.
- In loosely coupled systems the client of a service is essentially independent of the service.
- The way a client communicate with the service is not dependent of the service implementation.
- The client communicates with service according to a specified, well-defined interface.

What Happens in SOA?

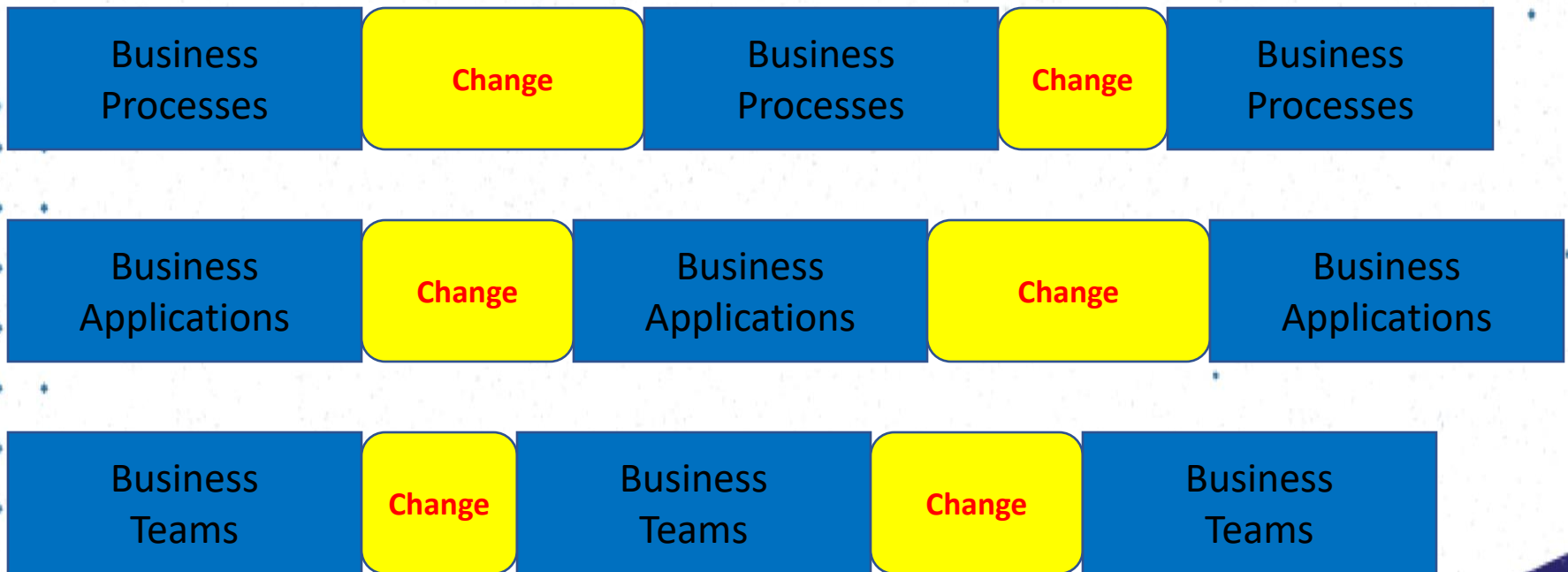
- Traditional
 - Services and service processing logic is mixed up in the code.
- SOA
 - Try to separate business process and services
 - Example : Data access layer detach the data management functionality out of application programs. Similarly we try to detach business process and the services.

Why SOA?

- Accommodate rapid changes to the IT landscape in relation to the changes in the Business environments.
- Promotes reuse of services across multiple business process automations.
- Simple dynamic interfacing of services.
- The services can be discovered and interfaces can be changed without major changes to applications.

Why SOA for Businesses

Manage **CHANGE**



Time

Characteristics of SOA

- **Loose Coupling** - client can discover server's supported protocols/formats and negotiate communication semantics
- **Reusable** – similar to objected-orientation
- **Autonomous** - runs independently of other systems
- **Stateless** - no ongoing commitment between client & service
- **Composable** - one service can contain another
- **Standards-based** - interoperability among SOA services
- **Contract-based** – i.e. uses interfaces
- **Fine-grained** - services should be small (higher cohesion)
 - Reusable, modular - another way of saying 'fine-grained'
- **Encapsulation** - information hiding
- **Heterogeneous** – technologies, platforms, applications, etc.
- **Location transparent**

Implementing SOA

- SOA can be implemented using many technologies:
 - Web services
 - RPC
 - CORBA
 - DCOM
 - SOAP
 - WCF (Windows Communication Foundations) Part of .NET framework.
 - REST (Web API)

Where to Use SOA?

- SOA is most useful for what it was designed for:
 - When crossing *platform* boundaries
 - When crossing *trust* boundaries
- Business logic that change frequently and highly reusable is more eligible for SOA.
 - E.g. Payment requests, Balance inquiries

Where *NOT* to Use SOA

- SOA isn't applicable everywhere. It's poor for:
 - **Non-distributed applications**
 - Applications with a **short deployed lifetime**
 - **Asynchronous communication** between servers
 - **Interactive GUI applications**
 - **A homogenous application environment**

A Web Service

- A service that is accessible over a web protocol
- Well defined interface - protocols define the interaction between the client and the server

Why Web Services?

- A Service accessible over a web URL!
 - Reusable functionality
 - Business to business integration
 - Information sharing
 - Business process automation
- Innovation - offer different services

Perform Web Service Invocation

What will you learn?

- Service Invocation

Hands-on

1 - Go to <http://openweathermap.org/>

2 - Read the documentation

3 - Signup and get a key

4 - Try to read weather by giving longitude and latitude

`http://api.openweathermap.org/data/2.5/weather?lat=35&
lon=139&APPID=your_key`

5 - Try to read weather in Colombo

Web Services Everywhere!

- Grid computing
 - [SETI@home](#)
- Cloud computing
 - IaaS - AWS
 - SaaS APIs - Salesforce APIs, Netsuite APIs, PeopleHR API
- Google Maps APIs

Web Services & SOAP

SOAP

What does it stand for?

- Simple Object Access Protocol

What is it?

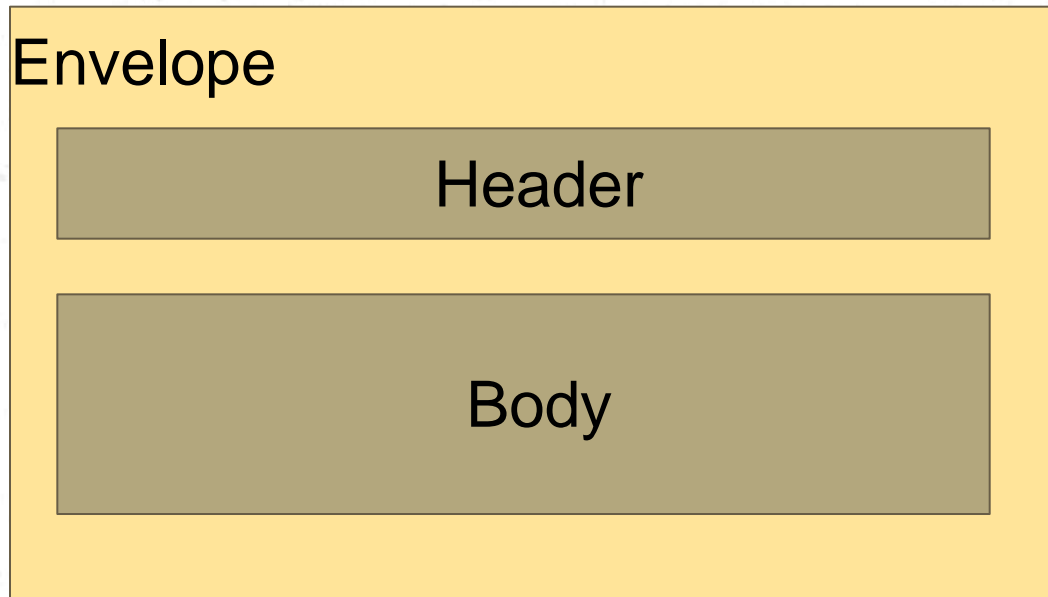
- Two versions - SOAP 1.1 and SOAP 1.2
- SOAP 1.2 became a W3C recommendation in 2003

Who/where/when?

- Initiated by IBM, Microsoft

SOAP Basics

- Relies on XML and defines a message structure
- Can run on any protocol HTTP, SMTP



Sample SOAP Message

```
<soapenv:Envelope xmlns:soapenv = "http://schemas.xmlsoap.org/soap/envelope/"  
  <soapenv:Body >  
    <ns:greetResponse xmlns:ns = "http://www.wso2.org/types" >  
      <return>Hello World, Dimuthu !!!</return>  
    </ns:greetResponse >  
  </soapenv:Body >  
</soapenv:Envelope >
```


SOAP Engine

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
    <ns:greetResponse xmlns:ns="http://www.wso2.org/types" >
      <return>Hello World, Dimuthu !!!</return>
    </ns:greetResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP Engine

Java Class
C# Class

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
    <ns:greetResponse xmlns:ns="http://www.wso2.org/types" >
      <return>Hello World, Dimuthu !!!</return>
    </ns:greetResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

WSDL

- Web Service Description Language - WSDL1.1 & 2.0
- Describes a web service using XML
 - Uses XML Schema
 - Input message
 - Output message
 - Transports
 - Versions

SOAP and WSDL - A strong marriage

```
- <wsdl:definitions targetNamespace="http://www.wso2.org/types">
  <wsdl:documentation>HelloService</wsdl:documentation>
  - <wsdl:types>
    - <xs:schema attributeFormDefault="qualified" elementFormDefault="unqualified" targetNam
      - <xs:element name="greet">
        - <xs:complexType>
          - <xs:sequence>
            <xs:element minOccurs="0" name="name" nillable="true" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      - <xs:element name="greetResponse">
        - <xs:complexType>
          - <xs:sequence>
            <xs:element minOccurs="0" name="return" nillable="true" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:schema>
  </wsdl:types>
  - <wsdl:message name="greetRequest">
    <wsdl:part name="parameters" element="ns:greet"/>
  </wsdl:message>
  - <wsdl:message name="greetResponse">
    <wsdl:part name="parameters" element="ns:greetResponse"/>
  </wsdl:message>
```

Perform SOAP Service Invocation

What will you learn?

- Service Invocation
- Self contained functionality
- Service interface

WS* Specifications

- WSDL 2.0
- WS Security
- WS Addressing
- WS Policy
- WS Trust

REST

REST

What does it stand for?

- REpresentational State Transfer

What is it?

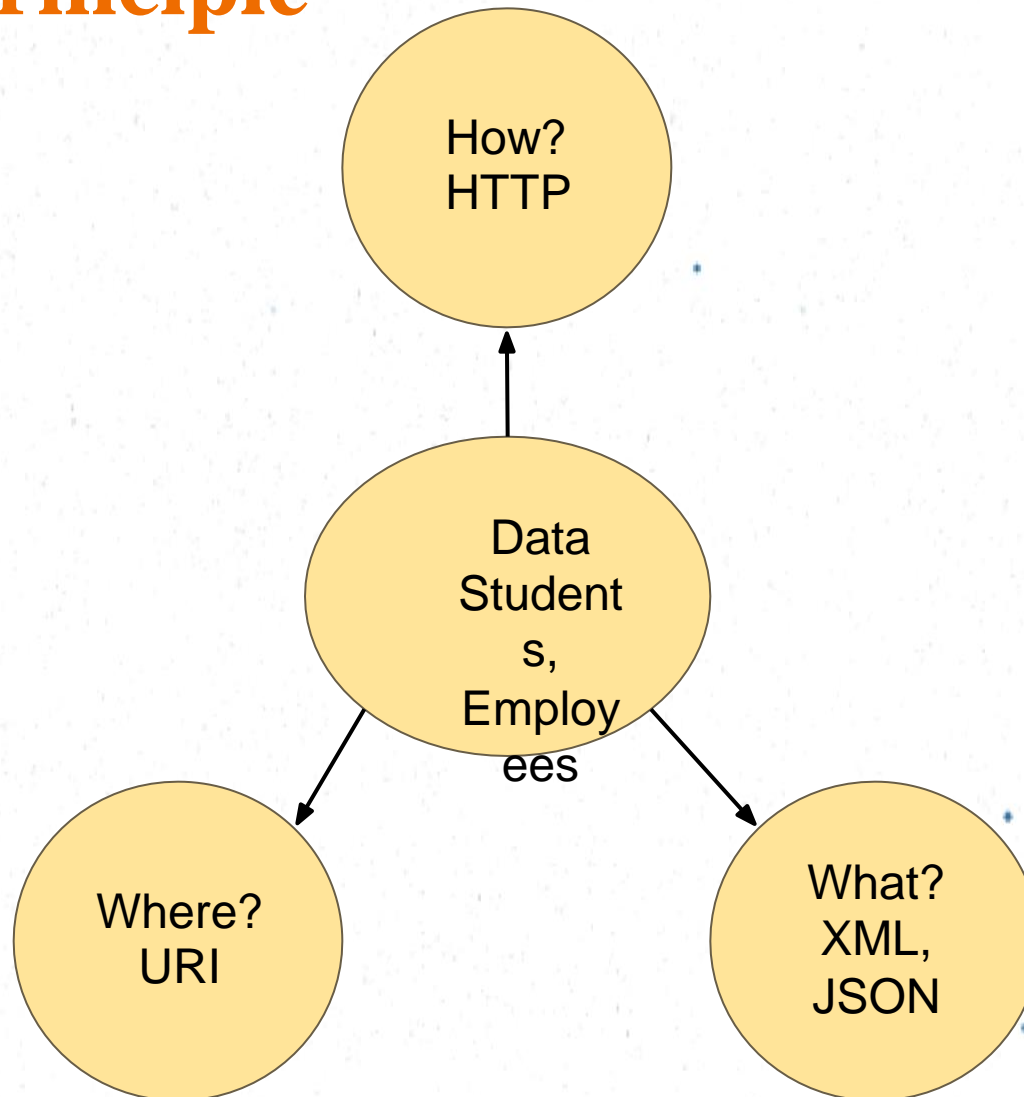
- Architectural pattern - not a standard

Who/where/when?

- Roy Fielding in 2001



REST Principle



Stateless

- No state stored on the server
- Every HTTP request executes in complete isolation
- Simpler to design and evolve
- Easier to scale

REST - Methods

- Defines the action taken with a URL
- Proper RESTful services expose all four

HTTP Method	Action	Example
POST	Create	http://wso2.com/general/dbusers/user/
GET	Read	http://wso2.com/general/dbusers/users http://wso2.com/general/dbusers/user/sam
PUT	Update or Create	http://wso2.com/general/dbusers/user/sam
DELETE	Delete	http://wso2.com/general/dbusers/user/sam

URIs - Addressability

- Name, address and version of resource
- Self-descriptive
- Unique URIs are exposed for every resource from RESTful system
 - URI per resource
- URIs are discoverable by clients

Data Representation

- Can be
 - XML, JSON, HTML
- Content negotiation based on HTTP headers
 - Accept or Content-Type
- Query parameters
 - GET /v1/employees/123?format=json
- URI extension
 - GET /v1/employees/123.xml

An Example REST

- In [http://ip-api.com/json/\[ip_address\]](http://ip-api.com/json/[ip_address]) service
 - Get the location of an Ip Address
 - Content type - Application/JSON
 - Send HTTP GET
-

REST Implementation

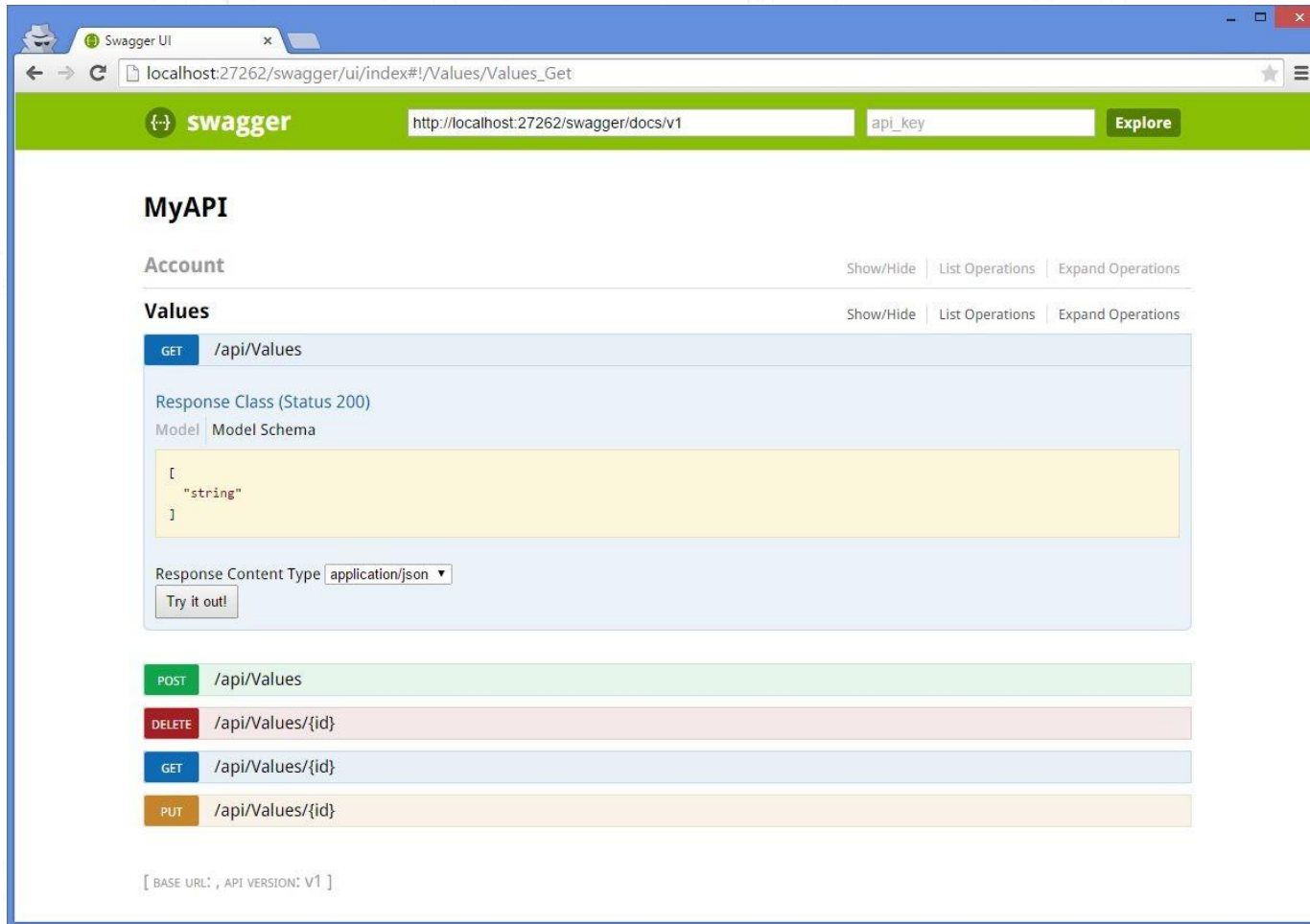
REST - Interface Description

- Swagger- Also known as OpenAPI specification
 - Interface description language for describing, producing, consuming and visualizing RESTful web services
- YAML based
- Allows both humans and machines to understand
- Goal - Update client and documentation at the same time as the server

Swagger

```
"paths": {
  "/": {
    "get": {
      "operationId": "listVersionsv2",
      "summary": "List API versions",
      "produces": [
        "application/json"
      ],
      "responses": {
        "200": {
          "description": "200 300 response",
          "examples": {
            "application/json": "{\n  \"versions\": [\n    {\n      \"name\": \"v1.0.0\", \"version\": \"1.0.0\", \"description\": \"Initial version\", \"status\": \"stable\", \"releaseDate\": \"2023-01-01\", \"isDefault\": true\n    },\n    {\n      \"name\": \"v1.1.0\", \"version\": \"1.1.0\", \"description\": \"Added new endpoint\", \"status\": \"stable\", \"releaseDate\": \"2023-03-15\", \"isDefault\": false\n    },\n    {\n      \"name\": \"v2.0.0-beta\", \"version\": \"2.0.0-beta\", \"description\": \"Beta version of v2\", \"status\": \"beta\", \"releaseDate\": \"2023-06-01\", \"isDefault\": false\n    }\n  ]\n}"
          }
        }
      }
    }
  }
}
```

Swagger



Web APIs

Web APIs

What is it?

- Not a standard. Not an architecture pattern. Just a “term”.
- Concentrating on the **accessibility** of services.
 - Secured (access controlled), open and monitored services
- A business capability delivered over the Internet to internal/external consumers

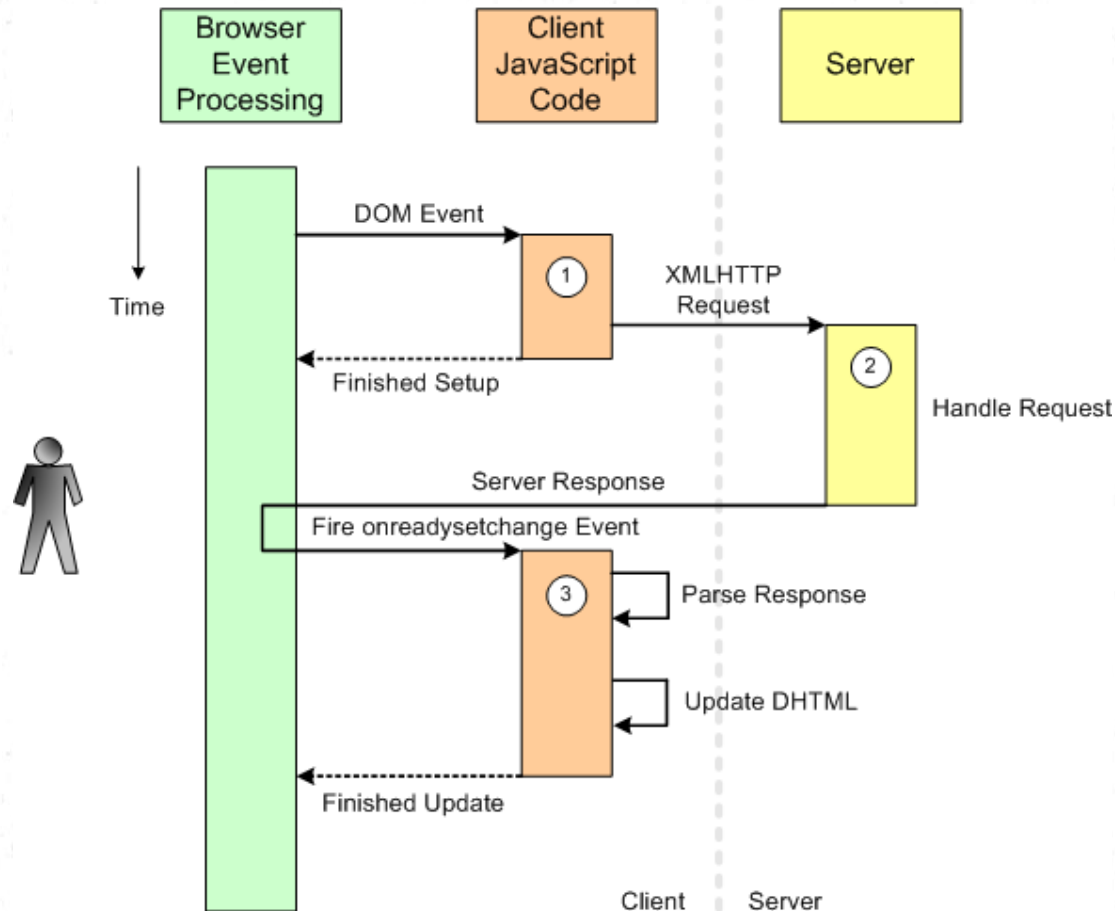
API = Service + Security + Documentation

Consuming REST Services - AJAX

Ajax

- **Ajax** stands for **A**synchronous **J**avaScript **A**nd **X**ML
 - Convergence of a few disparate technologies that together facilitate rich Web browser GUIs via client-side scripting
 - Term '**Ajax**' was coined to describe their use together
 - **JavaScript**: scripting language (ie: interpreted on the fly at run-time) for **client-side** processing in Web browsers
 - **Asynchronous**: Built-in browser support for sending *arbitrary* messages *asynchronously* to a server via JavaScript
 - **XML**: General-purpose data document format; Web browsers have built-in XML parsers for rendering HTML

Ajax Sequence Diagram



XMLHttpRequest Class

- Methods of XMLHttpRequest object:
 - **open** - sets the URL for submitting (sending) the request to
 - **setRequestHeader** - Add/set headers, usually just Content-Type
 - **send** - accepts the text of the message contents and sends it
- Properties of XMLHttpRequest object:
 - **onreadystatechange** - pointer to completion callback function.
Called every time the readyState changes. Note: lower case!
 - **readyState** - Callback state (see previous slide)
 - **status** - Call success/failure (200=success, others are error codes)
 - **responseText** - Raw message text from server
 - **responseXML** - XML parser object attached to responseText

```

function AddRPCAsync_SOAP12(onCompletionFn) {
    req = null;
    if (window.XMLHttpRequest != undefined)
        req = new XMLHttpRequest();           ← Firefox and compatible
    else
        req = new ActiveXObject("Microsoft.XMLHTTP");   ← Internet Explorer

    req.onreadystatechange = onCompletionFn;
    req.open("POST", "http://localhost/Webservices/Calculator.asmx", true); ← No .asmx/Add here!
    req.setRequestHeader("Content-Type", "application/soap+xml");           ← Set up header(s)

    req.send("<?xml version='1.0' encoding='utf-8'?'> \
        <soap12:Envelope xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
            xmlns:xsd='http://www.w3.org/2001/XMLSchema'
            xmlns:soap12='http://www.w3.org/2003/05/soap-envelope'> \
        <soap12:Body> \
            <Add xmlns='http://www.curtin.edu.au/SPD361/'> \
                <operand1>8</operand1> \
                <operand2>4</operand2> \
            </Add> \
        </soap12:Body> \
    </soap12:Envelope>");

}

function AddRPC_SOAP_OnCompletion() {           ← Same as SOAP 1.1
    if (req.readyState == 4) {
        if (req.status == 200) {
            var ndResult = req.responseXML.documentElement.getElementsByTagName("AddResult")[0];
            alert(ndResult.childNodes[0].nodeValue);           ← Access result (<AddResponse>) via DOM
        }
        else
            alert("Asynchronous call failed. ResponseText was:\n" + req.responseText);
    }
    req = null;
}

```

Ajax + Web Service Example (SOAP 1.2)

Calling a REST services with AJAX + JQUERY

```
$.ajax({  
  type: "GET",  
  dataType: "jsonp",  
  url: "http://localhost:8080/restws/json/product/get",  
  success: function(data){  
    alert(data);  
  }  
  error: function(data);  
    alert('error');  
});
```

AJAX in JQuery

- `$.get(url [, data] [, success(data,textStatus, jqXHR){})`

```
$.get( "ajax/test.html", function( data ) {  
    $( ".result" ).html( data );  
    alert( "Load was performed." );  
});
```

- `$.post(url [, data] [, success(data,textStatus, jqXHR){})`

```
$.post( "ajax/test.html", postdata, function( data ) {  
    $( ".result" ).html( data );  
});
```

- `$.getJSON(url [, data] [, success(data,textStatus, jqXHR){})`
 - Use an AJAX get request to get JSON data

REST and SOAP Implement SOA

Summary

- Service Oriented Architecture
- Web Services
- REST
- APIs & Microservices

Reuse



Many technologies for implementing a SOA.

None is perfect.

All achieve data sharing,
modularity, agility, reuse
and
innovation!

Questions?