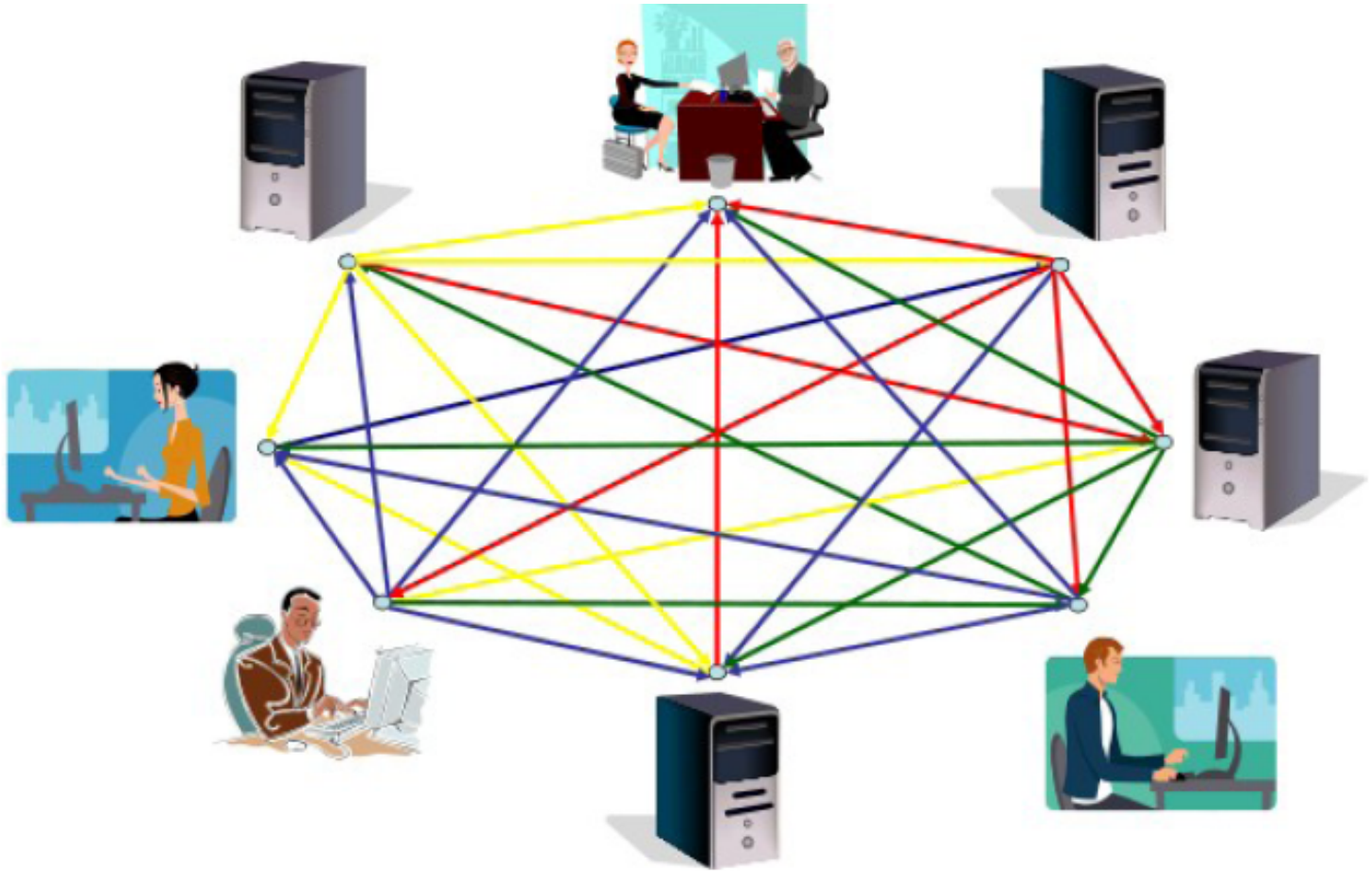
Enterprise Application Integration

Software Architecture
3rd Year– Semester 1
By Udara Samaratunge

Service Oriented Architecture - SOA



SOA Evolution

- Main Frames – Used Tapes to transfer files
- Later lower level socket based communication was used
- Then came, Network File System (NFS) and File Transfer Protocols (FTP)
- Remote Procedure Calls (RPCs) got matured along with the improved of server hardware
- CORBA came in but the advent of Java resulted the demise of CORBA
- DCOM came in but its proprietary nature resulted its demise
- SOAP relies on XML as the payload, which has got much higher degree of interoperability between programming languages.

Problems related to RPC

- **Tight coupling** between local and remote systems requires significant bandwidth demands

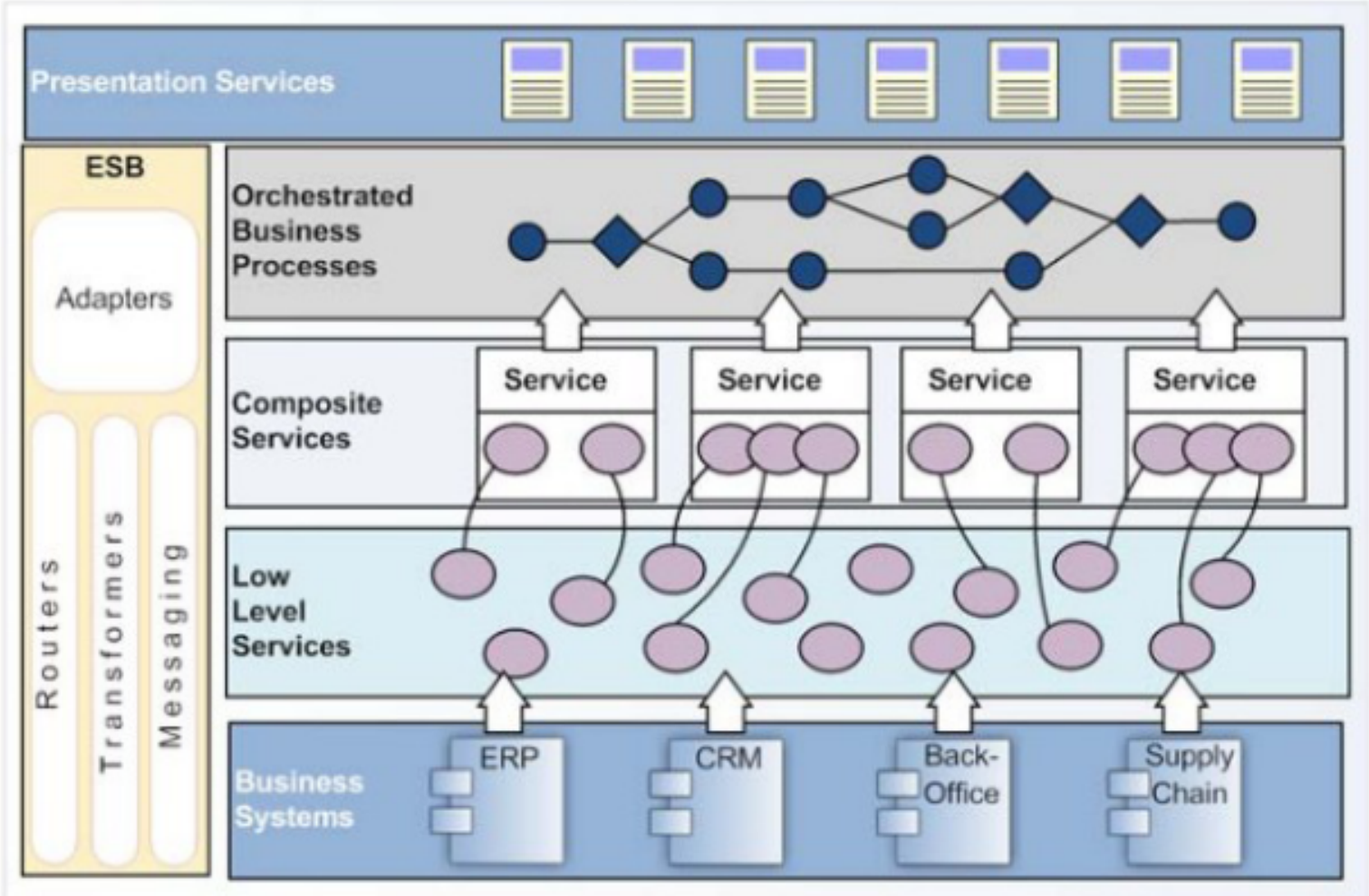
- **Interoperability issues** – Mainly due to incompatible data types in different languages

SOAP's message style can overcome above issues. SOA was developed keeping "loose coupling" and the "interoperability" in mind

Why SOA?

- CORBA, EJB, DCOM introduced a highly coupled RPC. [*Unlike SOA*]
- EJB and DCOM were tied to specific platforms and not at all inter-operable. [*Unlike SOA*]
- EJB, DCOM and CORBA were more relied on commercial oriented products. [*Unlike SOA*]
- **SOA** can be implemented using a completed "Open Source" Stack.
- **SOA** relies on XML as the *underlying data representation*, unlike the others, which used *proprietary binary-based objects*
- Unlike CORBA, EJB or DCOM, **SOA** is more than a RPC technology, It is a,
  - Governance
  - SLAs (Service Level Agreements)
  - Meta-data Definitions/ Registries

The SOA Environment

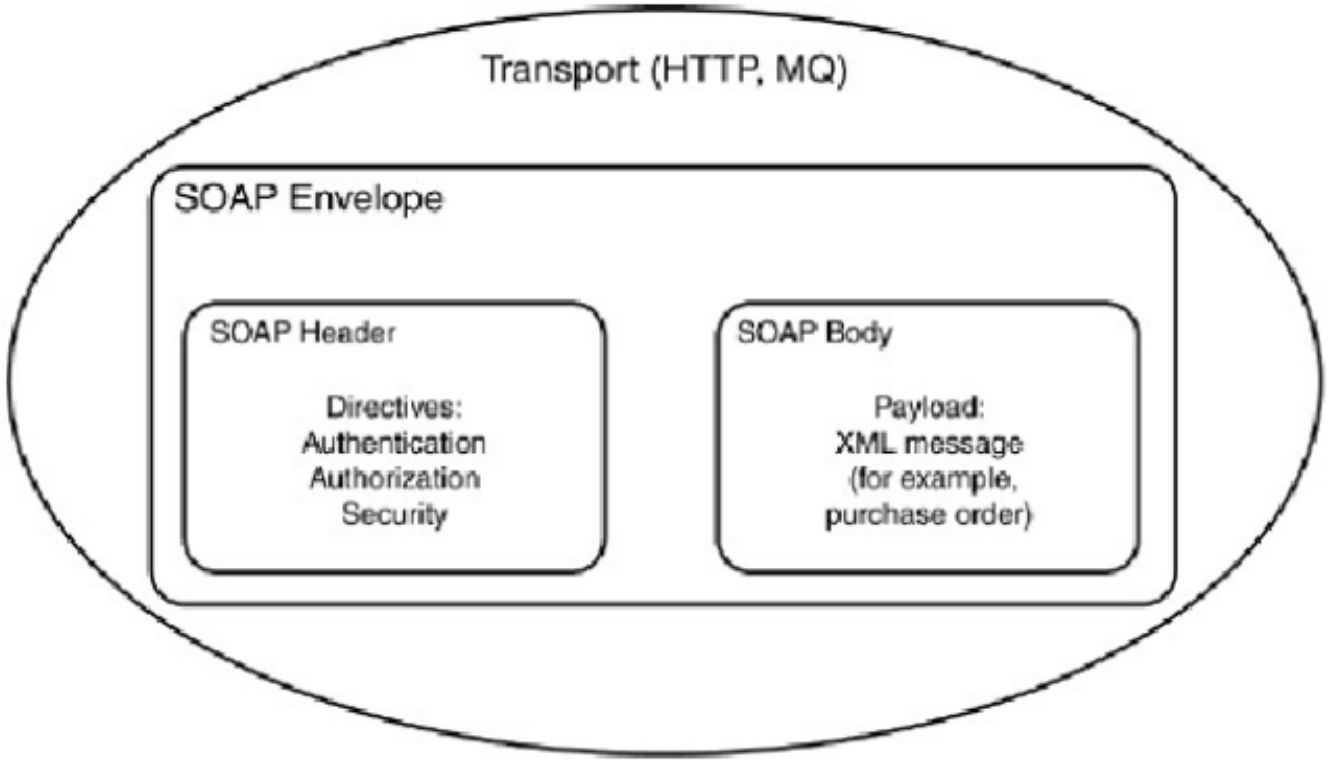


The characteristics of SOA

- The service Interface/Contract
- A service must have well defined interface contract.
- This contract should identify,
  - The "operations" that are available through the service
  - "Data Requirements" for any "Exchanged Information"
- **WSDL (Web Service Description Language)** is a good example for a **service contract**
- Web services – Related technologies = **XML / XML Schema**
- **Web service** use **SOAP** as **communication Protocol.** [**Simple Object Access Protocol**]
- **SOAP** runs on **HTTP protocol** & uses **default port 80**. **Message** structured as **XML**.

XML Messaging SOAP

- The SOAP envelop is just a container to hold XML data.
- **SOAP envelope**
  - **SOAP Header** – Contains information related to the message and its security
  - **SOAP Body** – Contains the **message pay load**
- Requests are encoded in XML and sent via HTTP POST
- Most **firewalls allow HTTP** traffic. This allows **XML-RPC** or **SOAP** messages to be used as **HTTP messages**.



WSDL

- **Three Sections**
  - **What Section** – Input and Output messages (<wsdl:types>, <wsdl:message>)
  - **How Section** – How messages should be packaged (bind) to different protocols in the SOAP envelop and how to transfer it (<wsdl:binding>)
  - **Where Section** – The endpoint details (<wsdl:service>)

```
−<wsdl:definitions targetNamespace="http://ead">
  <wsdl:documentation> Please Type your service description here </wsdl:documentation>
 +<wsdl:types></wsdl:types>
 +<wsdl:message name="farenhit2celciusRequest"></wsdl:message>
 +<wsdl:message name="farenhit2celciusResponse"></wsdl:message>
 +<wsdl:message name="celcius2farenhitRequest"></wsdl:message>
 +<wsdl:message name="celcius2farenhitResponse"></wsdl:message>
 −<wsdl:portType name="TempWSPortType">
  −<wsdl:operation name="farenhit2celcius">
     <wsdl:input message="ns:farenhit2celciusRequest" wsaw:Action="urn:farenhit2celcius"/>
     <wsdl:output message="ns:farenhit2celciusResponse" wsaw:Action="urn:farenhit2celciusResponse"/>
   </wsdl:operation>
  −<wsdl:operation name="celcius2farenhit">
     <wsdl:input message="ns:celcius2farenhitRequest" wsaw:Action="urn:celcius2farenhit"/>
     <wsdl:output message="ns:celcius2farenhitResponse" wsaw:Action="urn:celcius2farenhitResponse"/>
   </wsdl:operation>
 </wsdl:portType>
 +<wsdl:binding name="TempWSSoap11Binding" type="ns:TempWSPortType"></wsdl:binding>
 +<wsdl:binding name="TempWSSoap12Binding" type="ns:TempWSPortType"></wsdl:binding>
 +<wsdl:binding name="TempWSHttpBinding" type="ns:TempWSPortType"></wsdl:binding>
 −<wsdl:service name="TempWS">
  −<wsdl:port name="TempWSHttpSoap11Endpoint" binding="ns:TempWSSoap11Binding">
     <soap:address location="http://192.168.2.2:8080/axis2/services/TempWS.TempWSHttpSoap11Endpoint/"/>
   </wsdl:port>
  −<wsdl:port name="TempWSHttpSoap12Endpoint" binding="ns:TempWSSoap12Binding">
     <soap12:address location="http://192.168.2.2:8080/axis2/services/TempWS.TempWSHttpSoap12Endpoint/"/>
   </wsdl:port>
  −<wsdl:port name="TempWSHttpEndpoint" binding="ns:TempWSHttpBinding">
     <http:address location="http://192.168.2.2:8080/axis2/services/TempWS.TempWSHttpEndpoint/"/>
   </wsdl:port>
 </wsdl:service>
</wsdl:definitions>
```

What

How

Where

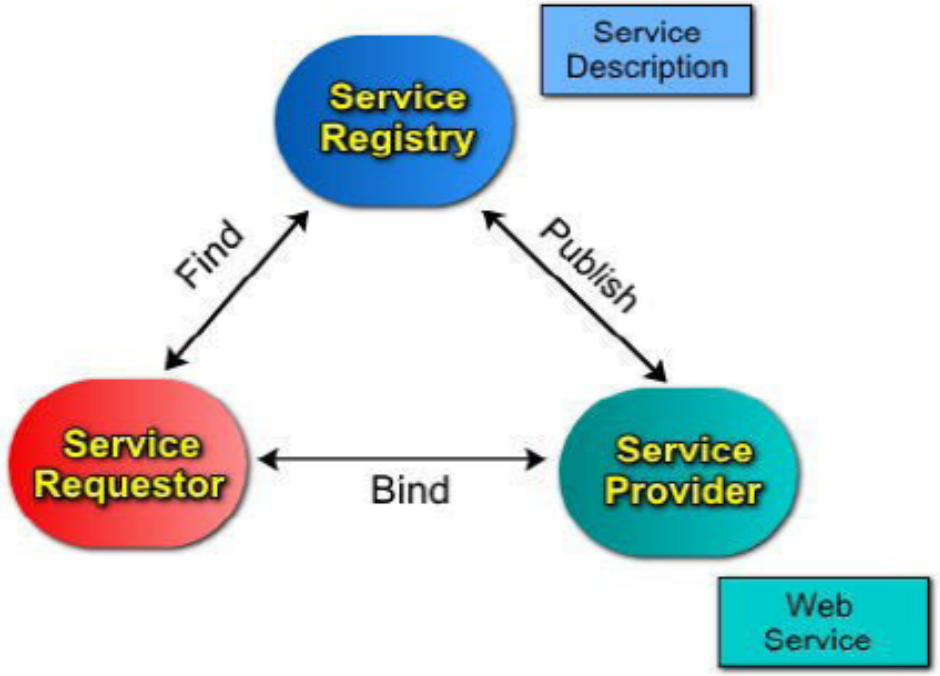
Web Services Model

- **UDDI** is an XMLbased standard for describing, publishing, and finding Web services
- UDDI = (**Universal Description, Discovery and Integration**)
- UDDI uses WSDL to describe interfaces to web services

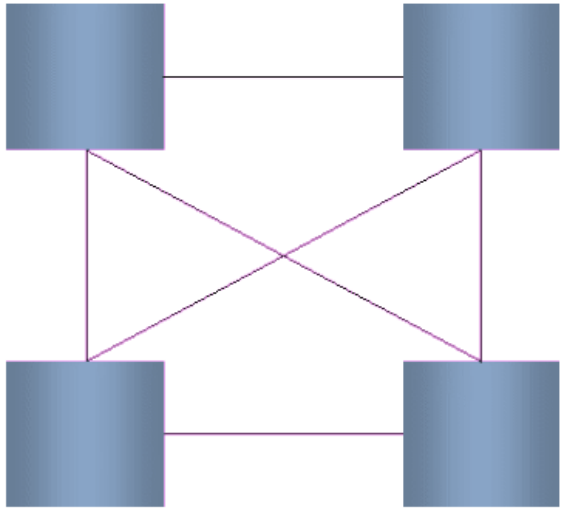
- **Service Provider**: The provider of web service
- **Service Requester**: The web service consumer
- **Service Registry**: The central directory of services

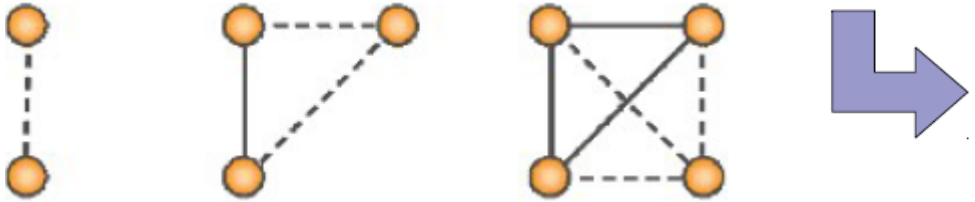
- **Approaches of writing web service**
  - **Bottom-Up/Code First Approach** [Implement web service method first]
  - **Top-Down/Contract First Approach** [Write WSDL first then generate Stub classes & Skeleton classes]

- If you generate web services for different plat-forms (Java or .NET) which method is most suitable?



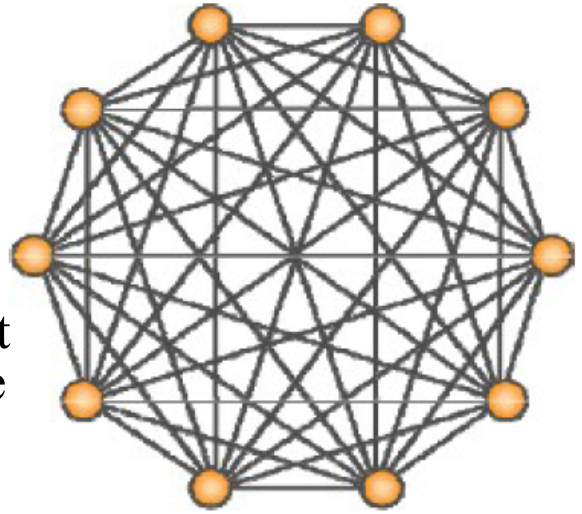
Point-to-Point Integration





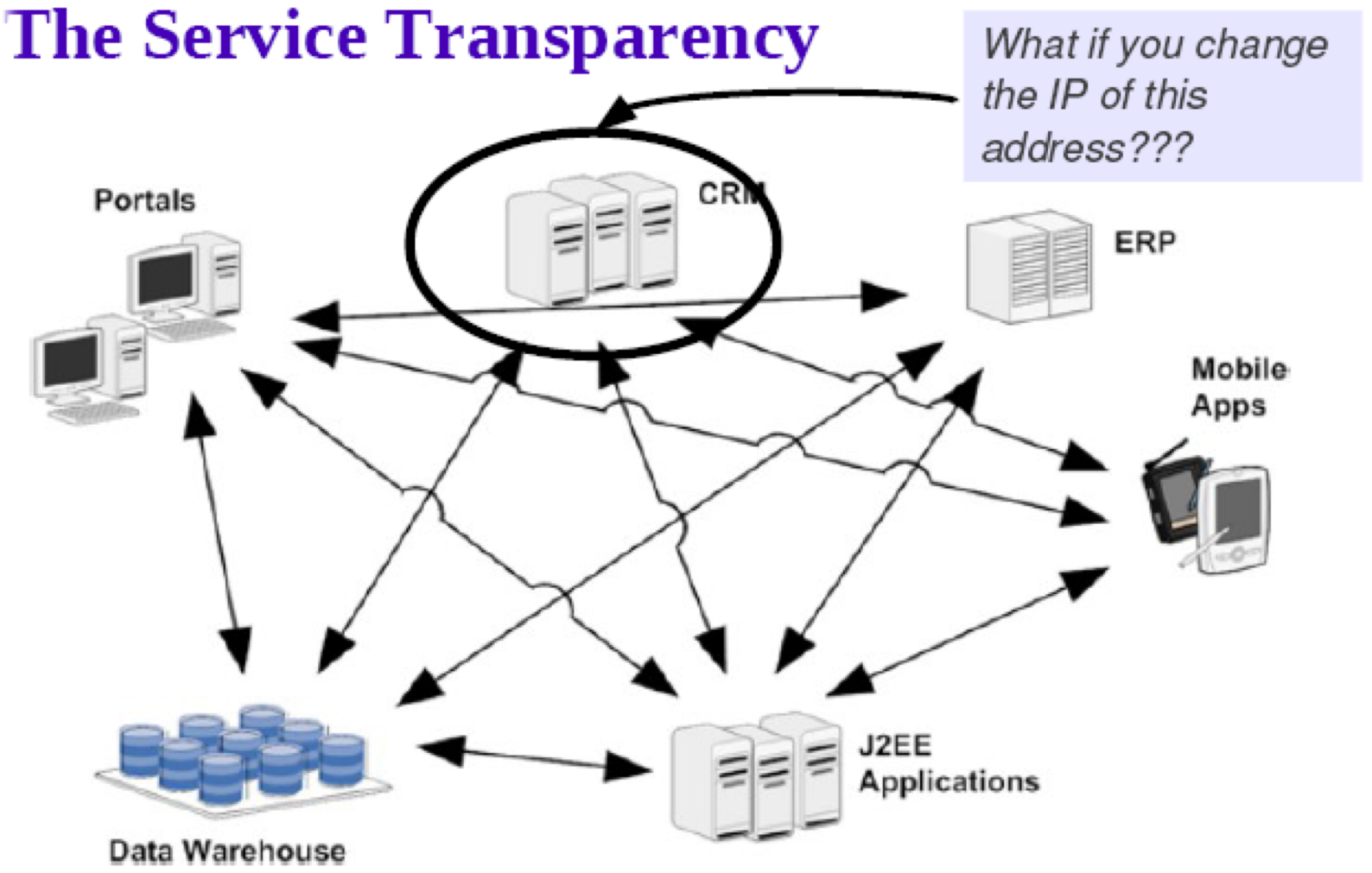
- **Benefits:**
  - Provides a way to connect each other
- **Drawbacks:**
  - Isolated without insufficient
  - Extremely “Spaghetti” like architecture, create headaches

- Specifically, linking every component to every other component will require **N(N-1)/2 physical connections**



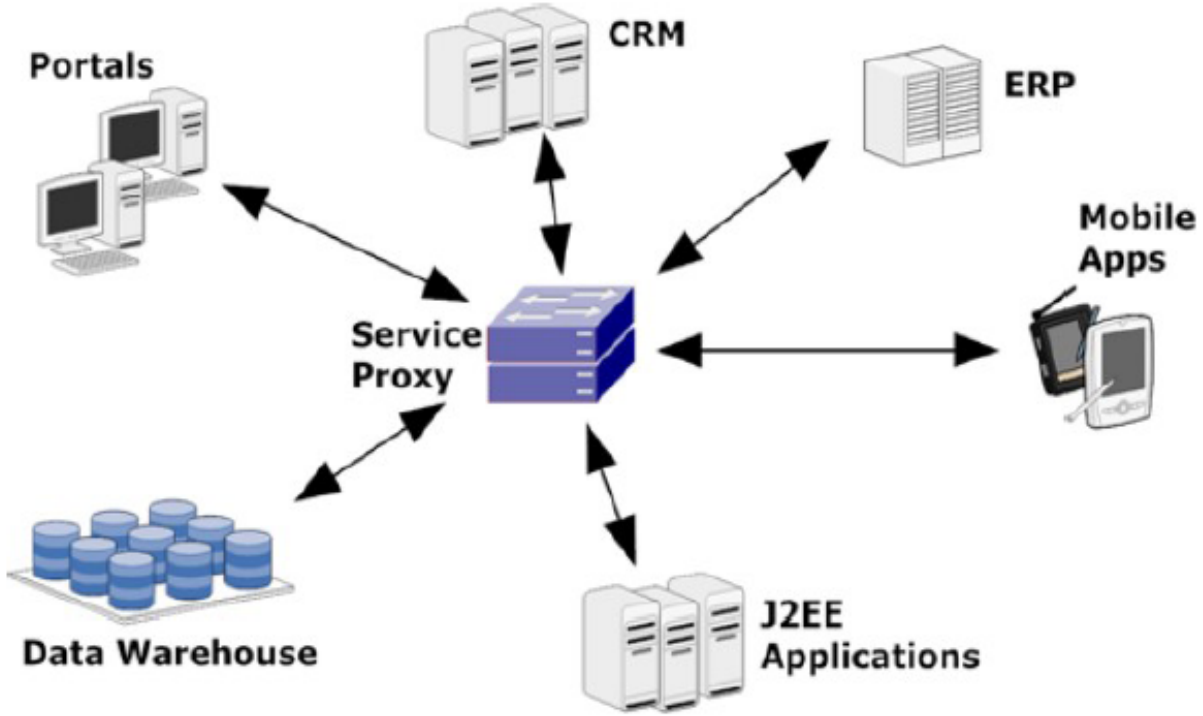
- N = Total Number of Components in the Network
- E.g: If there are 10 components in the network,
- Total number of physical connections = 10 (10-1)/2 = 45

Why ESB?


The Service Transparency
What if you change the IP of this address???
Portals
CRM
ERP
Mobile Apps
Data Warehouse
J2EE Applications

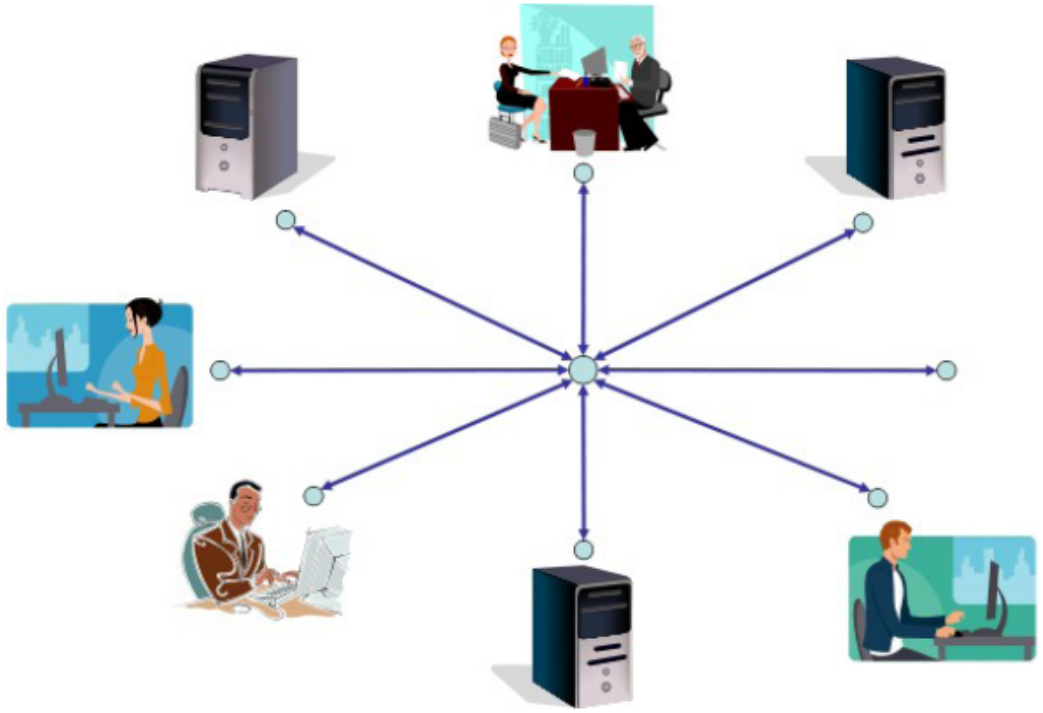
Solution for the Issue

The Service Transparency



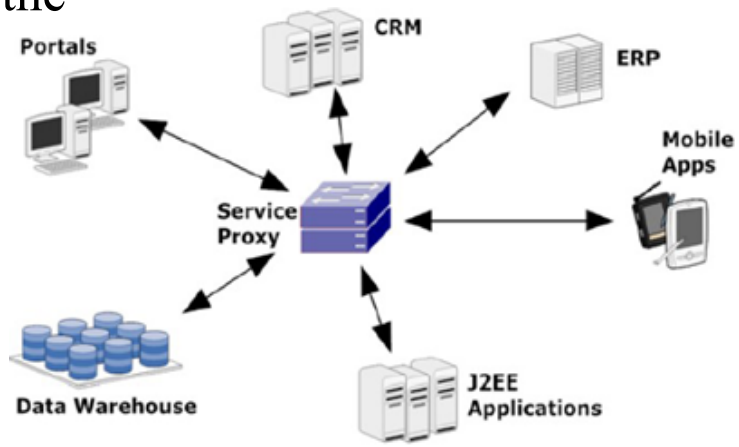
Hub-Spoke Model

A more centralized approached to the previous point-to-point approach



Solution from ESB

- An **ESB** or a **Service Proxy** can be the solution to the mess created by the point-to-point approach. **ESB** = Enterprise Service Bus
- All service calls are directed to the **proxy** or **gateway**, which in turn, forwards the message to the appropriate endpoint destination.
- If an endpoint is changed, only the **proxy configuration** will be required to be **changed**.
- Each **component** communicates with **Proxy**. **Proxy** should know how to send the **message for destination**.
- Now component free from maintaining IP addresses of destination. That **responsibility delegated to 3rd party module called ESB.**
- **ESB mediate the message for exact endpoint based on proxy details.**

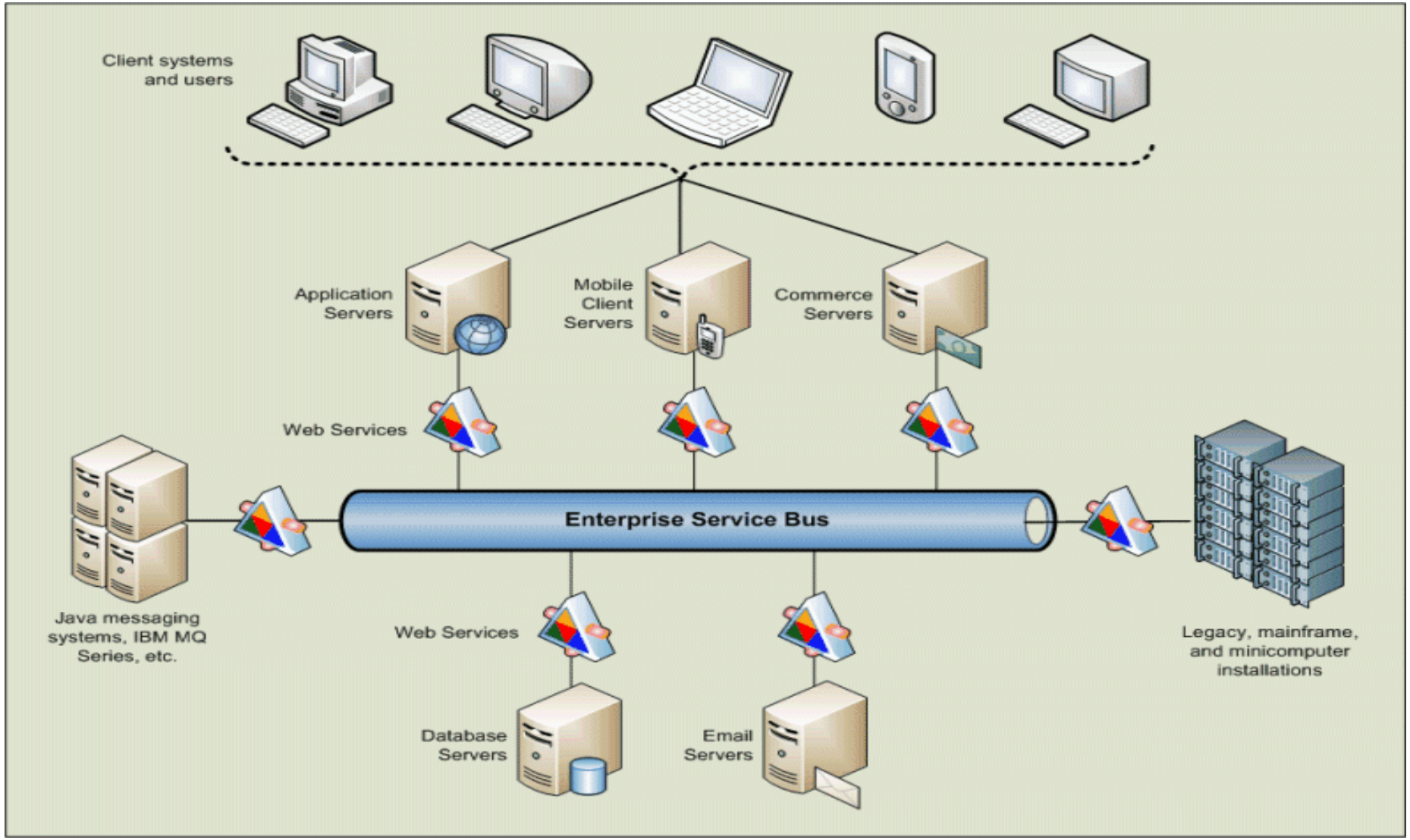


What is an ESB?

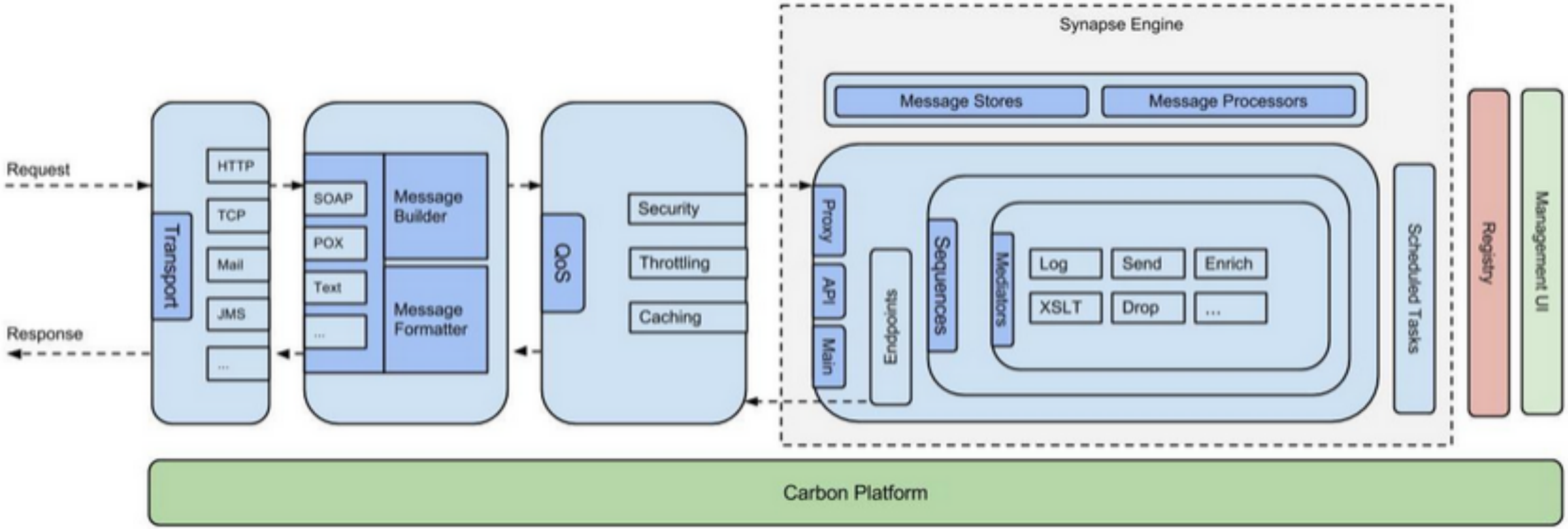
ESB "is a software architecture model used for designing and implementing the interaction and communication between **mutually interacting** software applications in **Service Oriented Architecture**"

- Promotes asynchronous message mediation
- Message identification and routing between applications and services.
- Allows messages to flow across different transport protocols
- Transforming of messages
- Allows secure, reliable communications
- Extensible architecture (based on pluggable components)

What is an ESB ? Cont.…

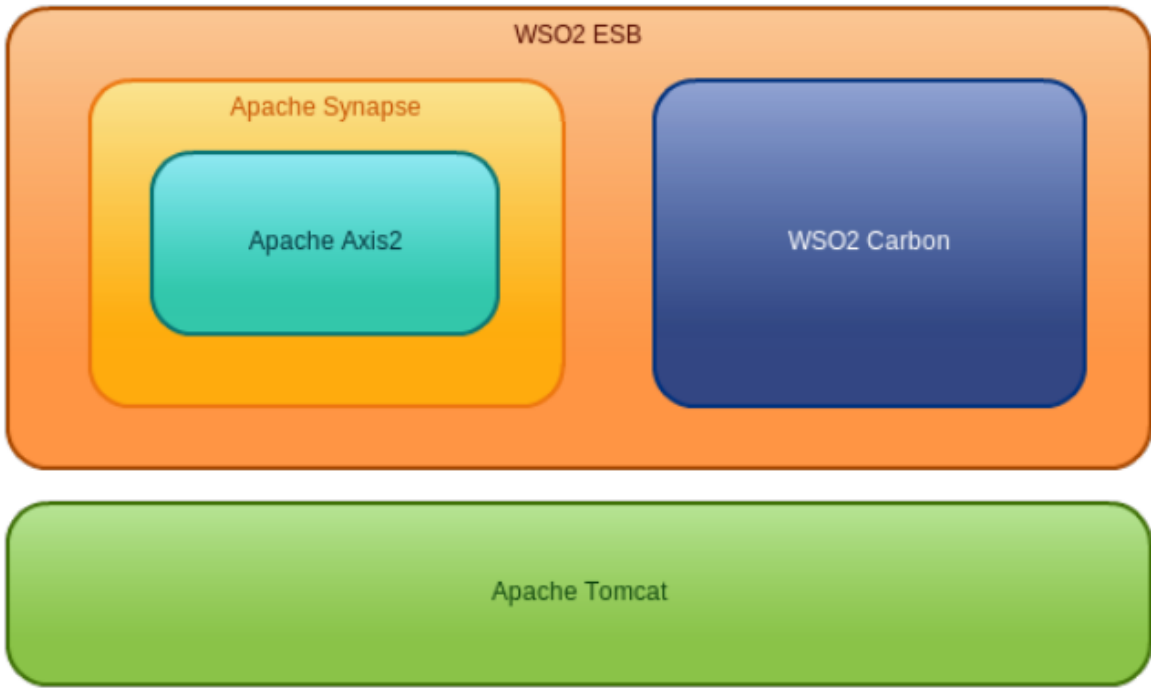


WSO2 ESB Architecture



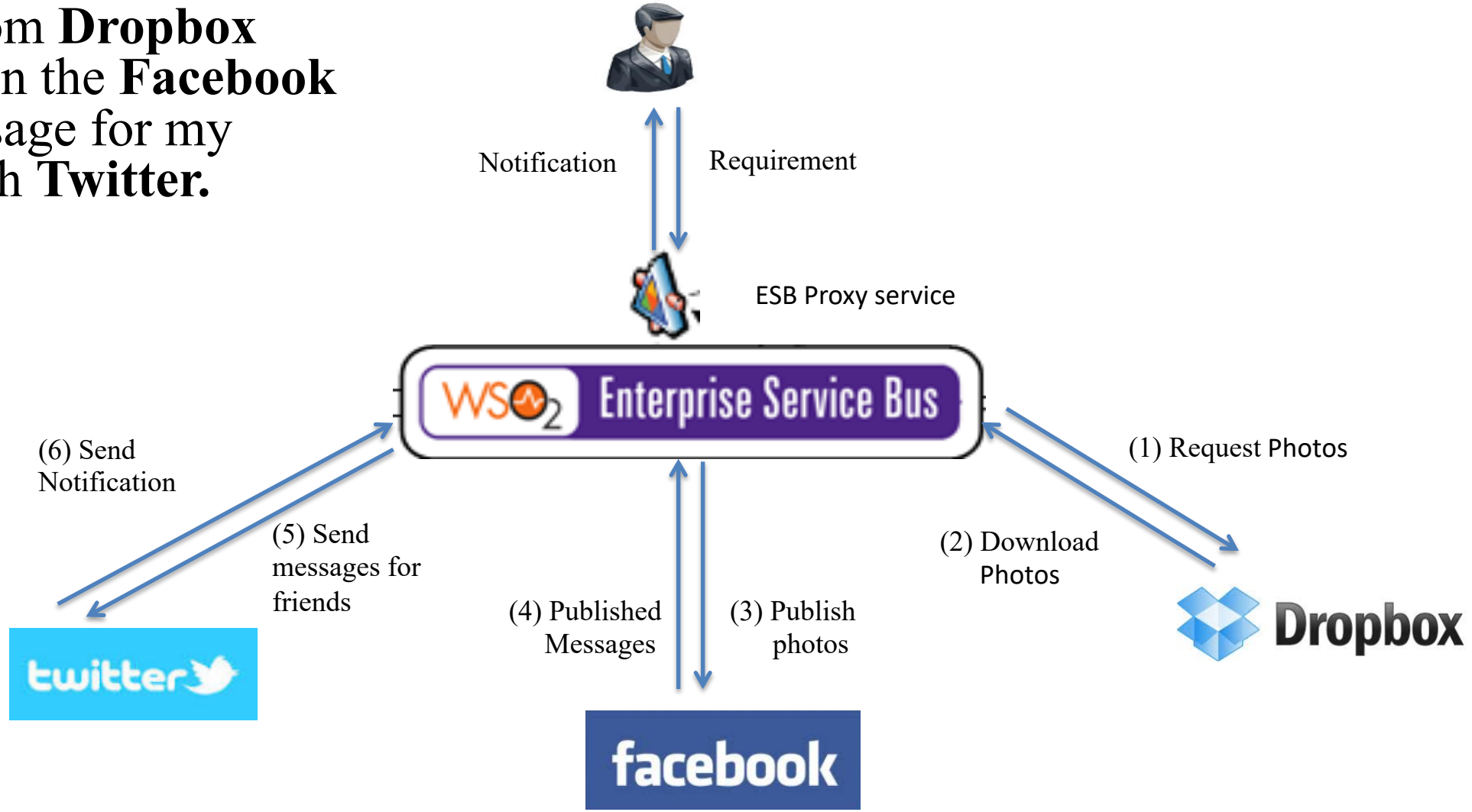
Technology Stack

- WSO2's core product, Middleware platform.
- A dynamic component model built for Java
- Components can be **started, stopped, installed, uninstalled** etc without reboot
- Built on **OSGi concepts**
- Powers SOA capabilities
- EVERYTHING that **WSO2 builds is based on Carbon**
- **Apache Synapse**:
  - Based on Axi2/Java
  - Acts a mediation library for different protocols
  - Supports different protocols through SOAP based Proxy Services



ESB Business Scenario.

- Get photos from **Dropbox** publish them in the **Facebook** and send message for my friends through **Twitter.**

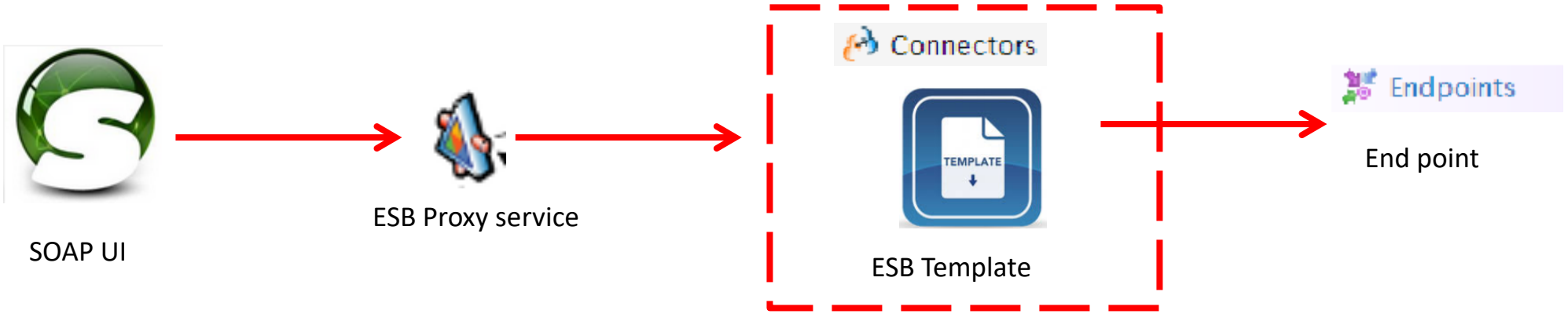


Apache Synapse

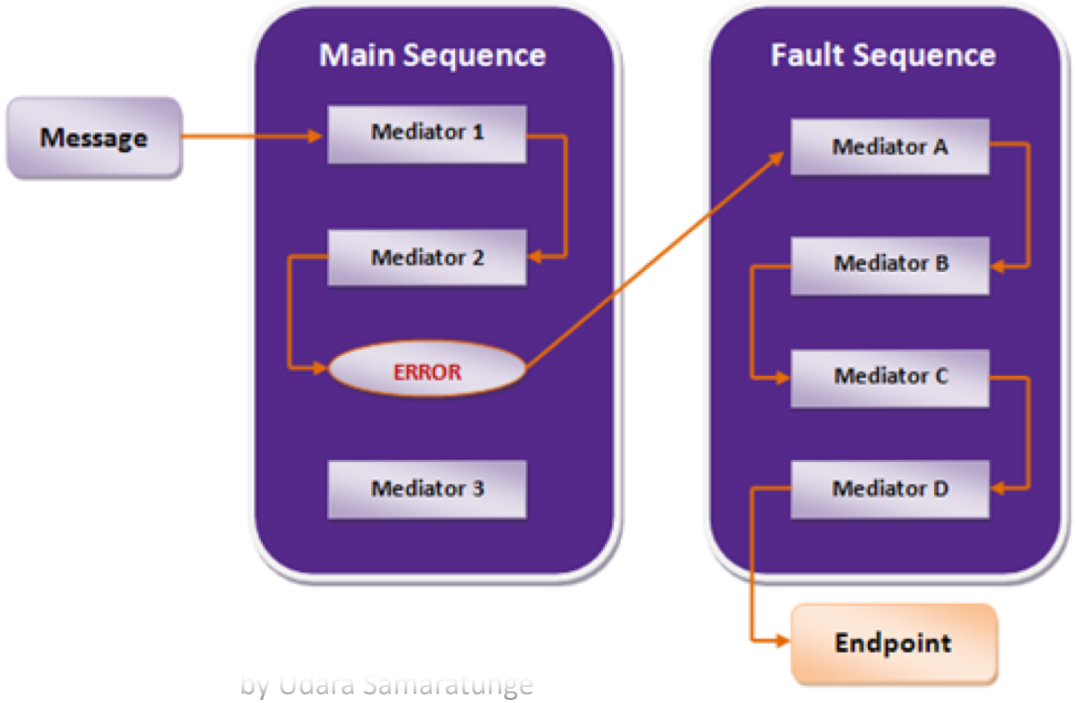
Apache Synapse "*is a lightweight and high-performance ESB with a powerful mediation engine.*"

- Default transport – HTTP-NIO (configurable pool of non-blocking worker threads)
- Support for any request types XML, SOAP, plain text, binary, JSON and etc.
- Supports any protocol HTTP, HTTPS, Mail (POP3, IMAP, SMTP), JMS, TCP, UDP, VFS, SMS, XMPP and FIX
- Non-blocking HTTP/HTTPS transports
- Support for WS-* standards (WS-Addressing, WS-Security and WS-Reliable Messaging)

Flow of request



In case an error occurs in the main sequence while processing, the message goes to the fault sequence.



Sample Request types

SOAP Request

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:urn="urn:wso2.connector.googledrive.getfile">
   <soapenv:Body>
   <root>
      <urn:fileId>1QL5LZOm9m4-h1lehXgU6gN4Kjovf8o3sqPKvw2RN40k</urn:fileId>
      <urn:updateViewedDate>false</urn:updateViewedDate>
      <urn:useServiceAccount>false</urn:useServiceAccount>
      <urn:clientId>521684679704.apps.googleusercontent.com</urn:clientId>
      <urn:clientSecret>AOZtcdakFwcnx1BuIavjtMEX</urn:clientSecret>
      <urn:accessToken></urn:accessToken>
      <urn:refreshToken>1/khM2ZQlpe_lPSp8WI</urn:refreshToken>
      <urn:serviceAccountEmail>
      757865184057@developer.gserviceaccount.com</urn:serviceAccountEmail>
      <urn:fields>alternateLink,labels</urn:fields>
   </root>
   </soapenv:Body>
</soapenv:Envelope>
```

POX Request

```
<createExpense>
   <arbitraryPassword></arbitraryPassword>
   <apiUrl>https://sansu.freshbooks.com</apiUrl>
   <authenticationToken>
   c361a63c7456519412fa8051ea605a6d</authenticationToken>
   <staffId>1</staffId>
   <status>1</status>
   <vendor>Sun Tzu Auto</vendor>
   <categoryId>5</categoryId>
   <projectId>19410</projectId>
   <date>2014-05-22</date>
   <clientId>99962</clientId>
   <compoundTax></compoundTax>
   <amount>2000</amount>
   <tax1Name>VAT</tax1Name>
   <tax1Amount>200</tax1Amount>
   <tax1Percent>10</tax1Percent>
   <tax2Name>GST</tax2Name>
   <tax2Amount>200</tax2Amount>
   <tax2Percent>10</tax2Percent>
   <notes>Expense Test</notes>
   <compoundTax>true</compoundTax>
</createExpense>
```

JSON Request

```
{
"accessToken":"AQV068KoSLBPT8U",
"apiUrl":"https://api.linkedin.com",
"publicUrl":"http://www.linkedin.com/pub/wso2connector-abdera/87/998/935",
"memberId":"12323"
}
```

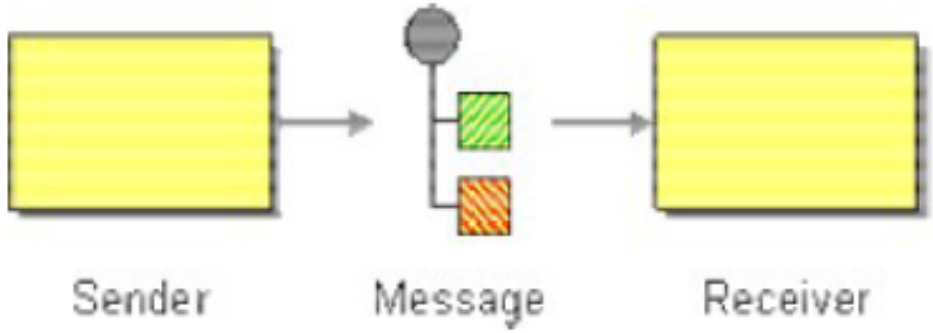
Message Mediation

- A **mediator** is **the basic, full-powered message processing unit** in the ESB.
- A mediator can take a message, carry out some predefined actions on it, and output the modified message.
- Usually, a **mediator** is configured using XML.
- Different mediators have their own XML configurations.
- At the run-time, a message is injected in to the mediator with the **ESB run-time information**.
- Then this mediator can do virtually anything with the message.
- A user can write a mediator and put it into the ESB.

Messaging

- **Messaging** is a form of ***loosely coupled*** distributed communication.
- **‘Communication'** can be understood as an exchange of messages between software components.
- **Message-oriented Middleware (MOM)** attempt to relax tightly coupled communication (such as TCP network sockets, CORBA or RMI) by the introduction of an “intermediary component”.
- **MOMs** allow software components to communicate **'indirectly'** with each other.
- **Benefits** of Messaging include message senders **not needing** to have precise **knowledge of their receivers**.
- Thus any data that is to be transmitted via a messaging system must be converted into one or more **messages**

Message Structure

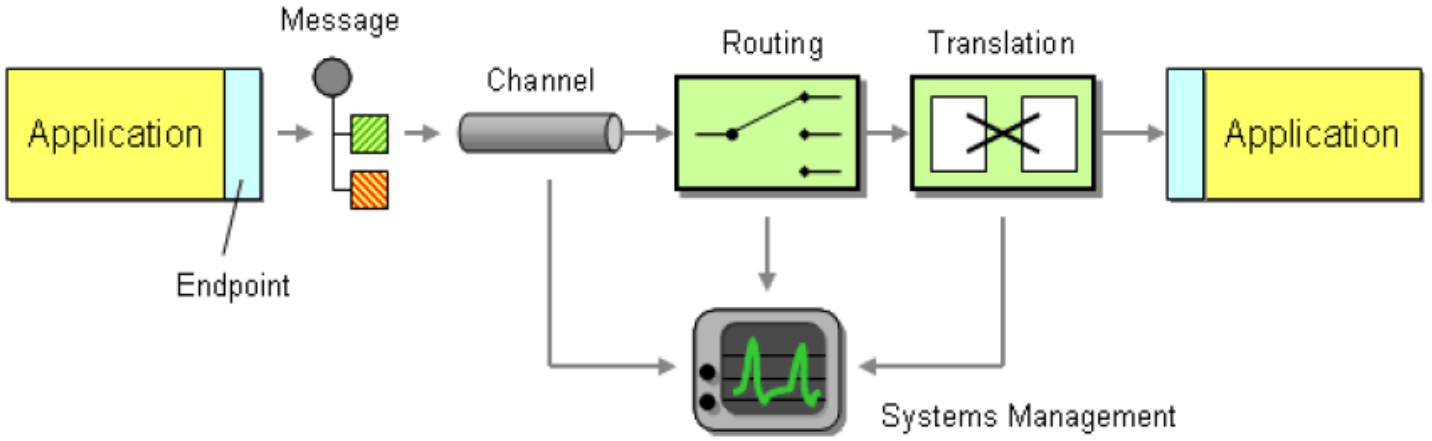


- Message consist of two basic parts.
- **Header**
  - Describes the data being transmitted, its origin, its destination, MessageID, Timestamp, Priority, Delivery mode, Message type and etc.
- **Body**
  - The data being transmitted; generally ignored by the messaging system and simply transmitted as-is.
- There are different kind of messages.
  - JMS Message
  - .NET Message
  - SOAP Message
- Messaging play major role in Enterprise Application Integration.

Enterprise Integration Patterns

There are 7 root patterns

1) Messaging
2) Message Channel
3) Message
4) Pipes and Filters
5) Message Router
6) Message Translator
7) Message Endpoint



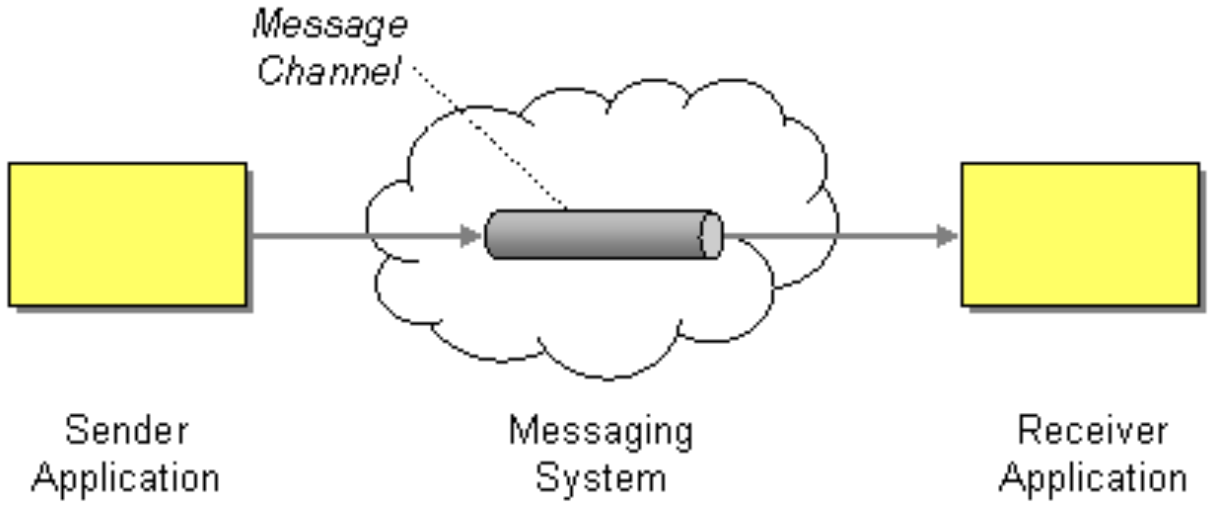
Enterprise Integration Patterns - Messaging

- **Messaging**
  - ✓ Is a technology that enables high speed, "**asynchronous**", program-to-program communication with reliable delivery.
  - ✓ Message itself is simply some sort of data structure such as a string, a byte array, a record, or an object

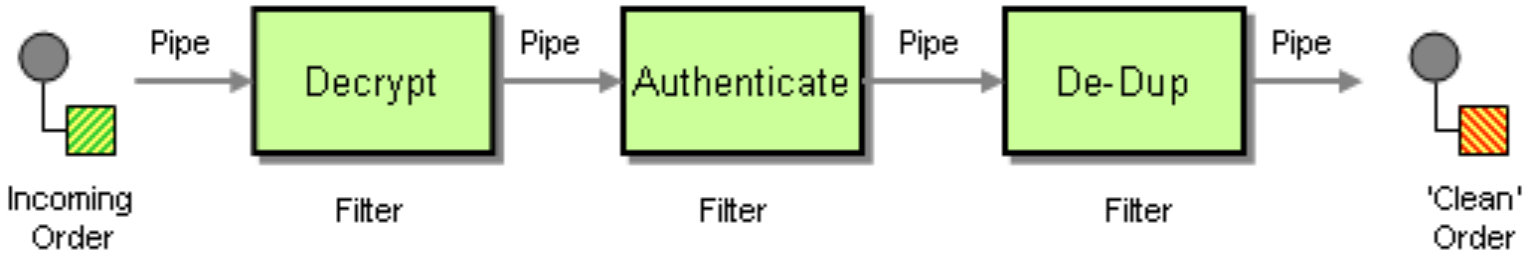
- **Messages**
  - ✓ Programs communicate by sending *packets of data* called "**messages**" to each other.
  - ✓ First process **marshals** the data into a byte stream and copy it from the first process to the second process.
  - ✓ The second process **unmarshal** the data back to its original form.

Enterprise Integration Patterns – Message channel

- **Message Channel**
  - ✓ Messaging applications transmit data through a Message Channel, a virtual pipe that connects a sender to a receiver
  - ✓ The media that can move data from one application to the other

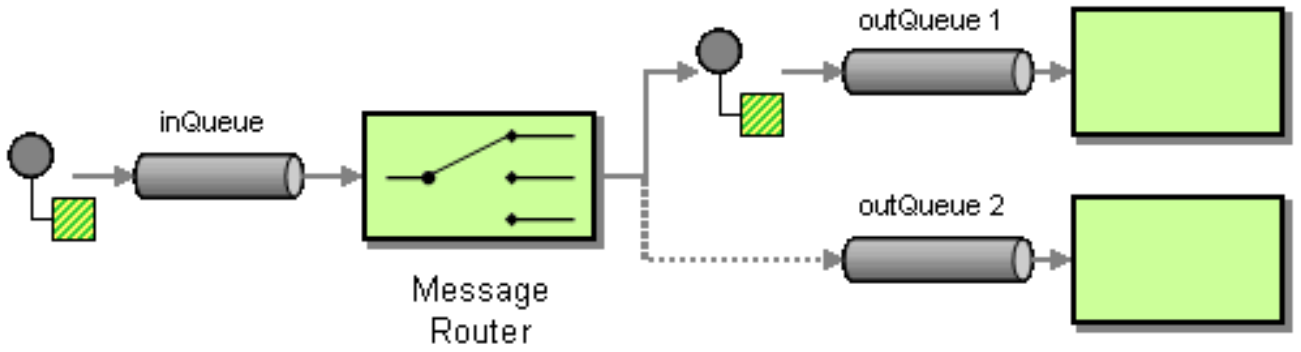


Enterprise Integration Patterns – Pipes and Filters



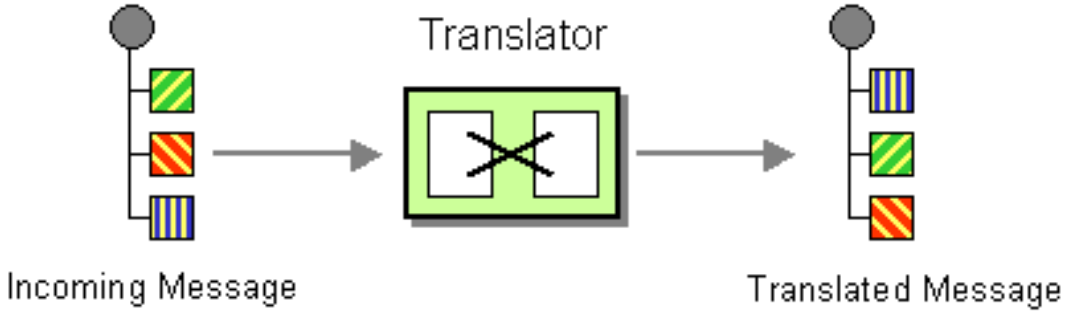
- Use the **Pipes and Filters** architectural style to **divide a larger processing task into a sequence of smaller, independent processing steps** (Filters)
- That are connected by channels (Pipes).
- Each filter exposes a very simple interface.
- The connection between filter and pipe is sometimes called **port**. In the basic form, each filter component has one input port and one output port.
- We can add new filters, omit existing ones or rearrange them into a new sequence.

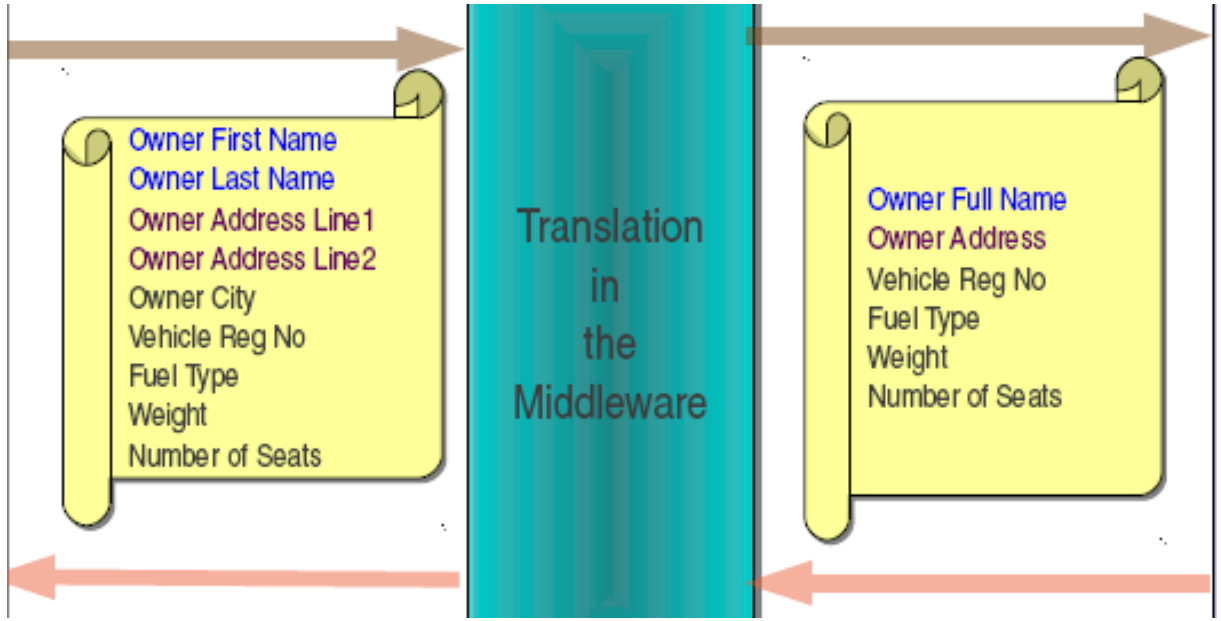
Message Router Pattern



- Consumes a Message from one Message Channel and republishes it to a different Message Channel depending on a set of conditions.
- A key property of the **Message Router** is that **it does not modify the message contents**.
- If new message types are defined,
  - new processing components are added,
  - or the routing rules change,
  - we need to change only the Message Router logic and all other components remain unaffected

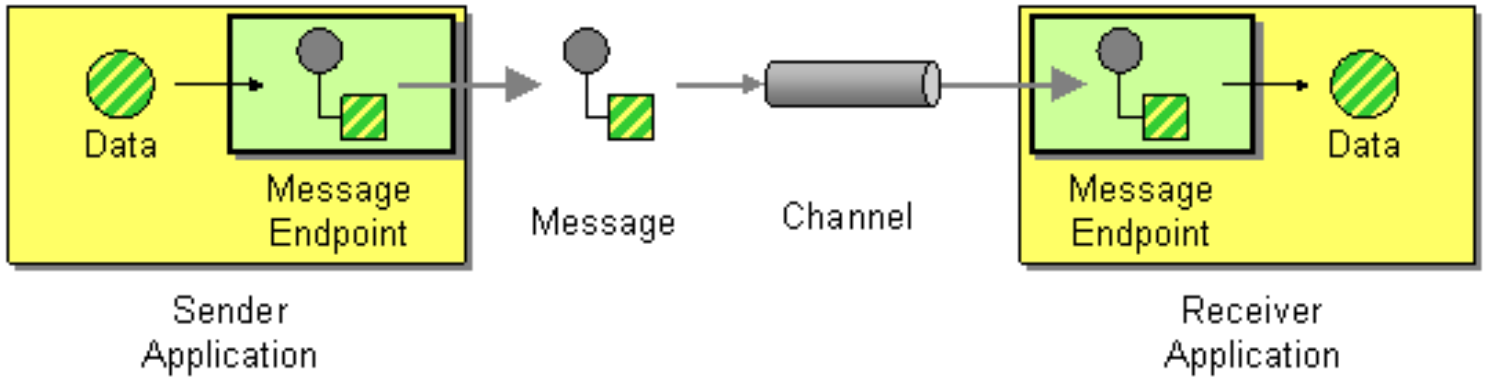
Message Translator Pattern





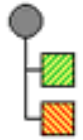
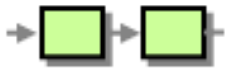
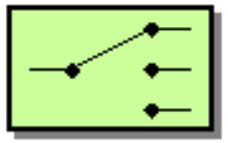
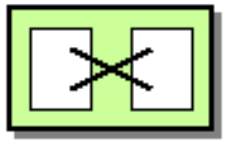
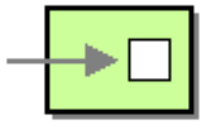
- The ***Message Translator*** is the messaging equivalent of the ***Adapter*** **pattern.**
- An adapter converts the interface of a component into a another interface so it can be used in a different context.
- **XSLT** Mediators can be used for message transformation.

Message Endpoint Pattern



- Connect an application to a messaging channel using a **Message Endpoint.**
- A client of the messaging system that the application can then use to send or receive messages.
- It is the **endpoint** that receives a message, **extracts the contents, and gives them to the application** in a meaningful way.

In a nutshell

	Message Channel	How does one application communicate with another using messaging?
	Message	How can two applications connected by a message channel exchange a piece of information?
	Pipes and Filters	How can we perform complex processing on a message while maintaining independence and flexibility?
	Message Router	How can you decouple individual processing steps so that messages can be passed to different filters depending on a set of conditions?
	Message Translator	How can systems using different data formats communicate with each other using messaging?
	Message Endpoint	How does an application connect to a messaging channel to send and receive messages?

Proxy Services

- **Pass Through Proxy** - Forwards messages to the endpoint without performing any processing on them. This proxy service is useful as a catch-all, so that messages that do not meet the criteria to be handled by other proxy services are simply forwarded to the endpoint.

- **Secure Proxy** - Uses WS-Security to process incoming requests and forward them to an unsecured backend service.

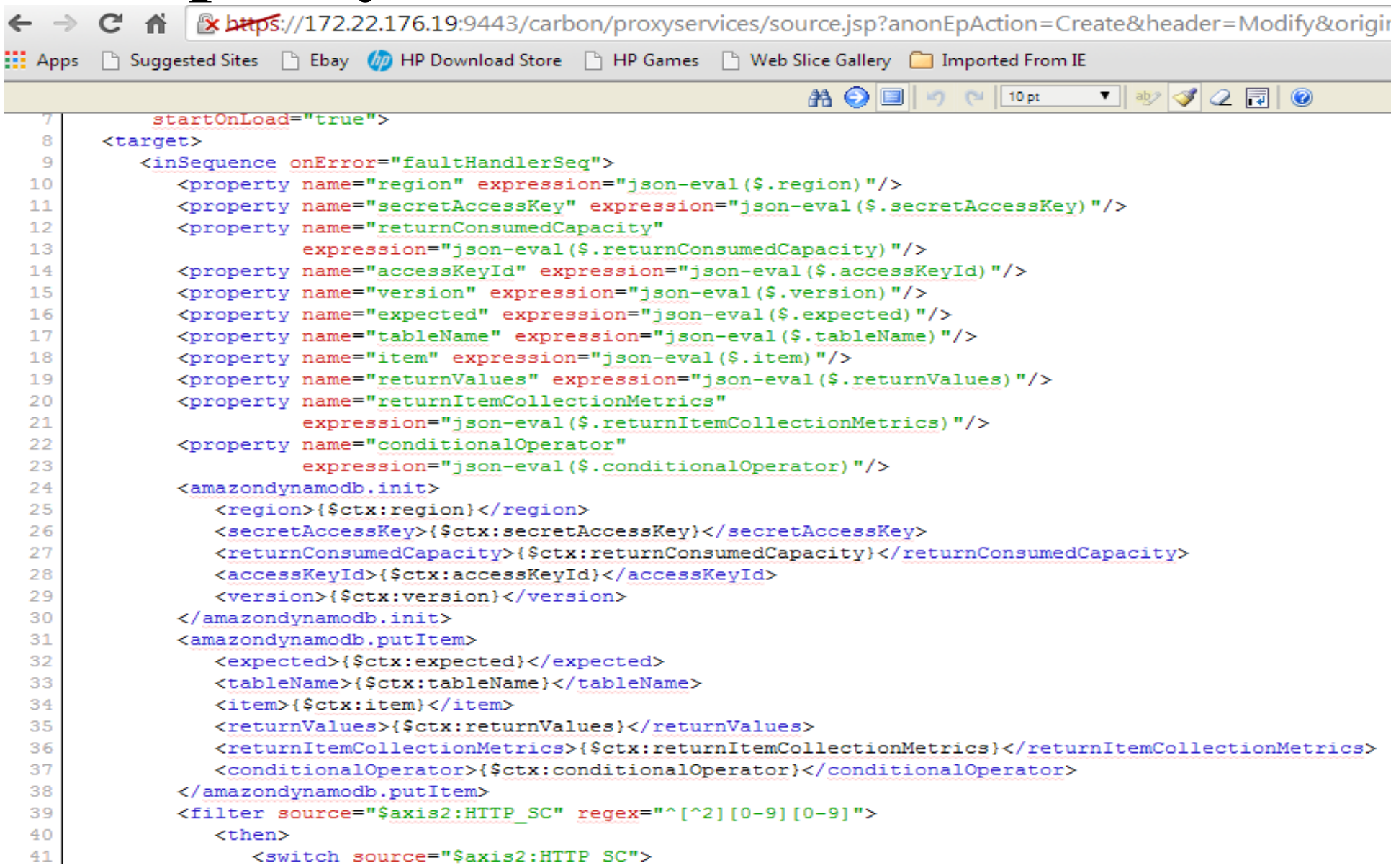
- **WSDL Based Proxy** - A proxy service that is created from the remotely hosted WSDL of an existing web service. The endpoint information is extracted from the WSDL.

- **Logging Proxy** - Logs all the incoming requests and forwards them to a given endpoint. It can also log responses from the backend service before routing them to the client.

- **Transformer Proxy** - Transforms all the incoming requests using XSLT and then forwards them to a given endpoint. It can also transform responses from the backend service.

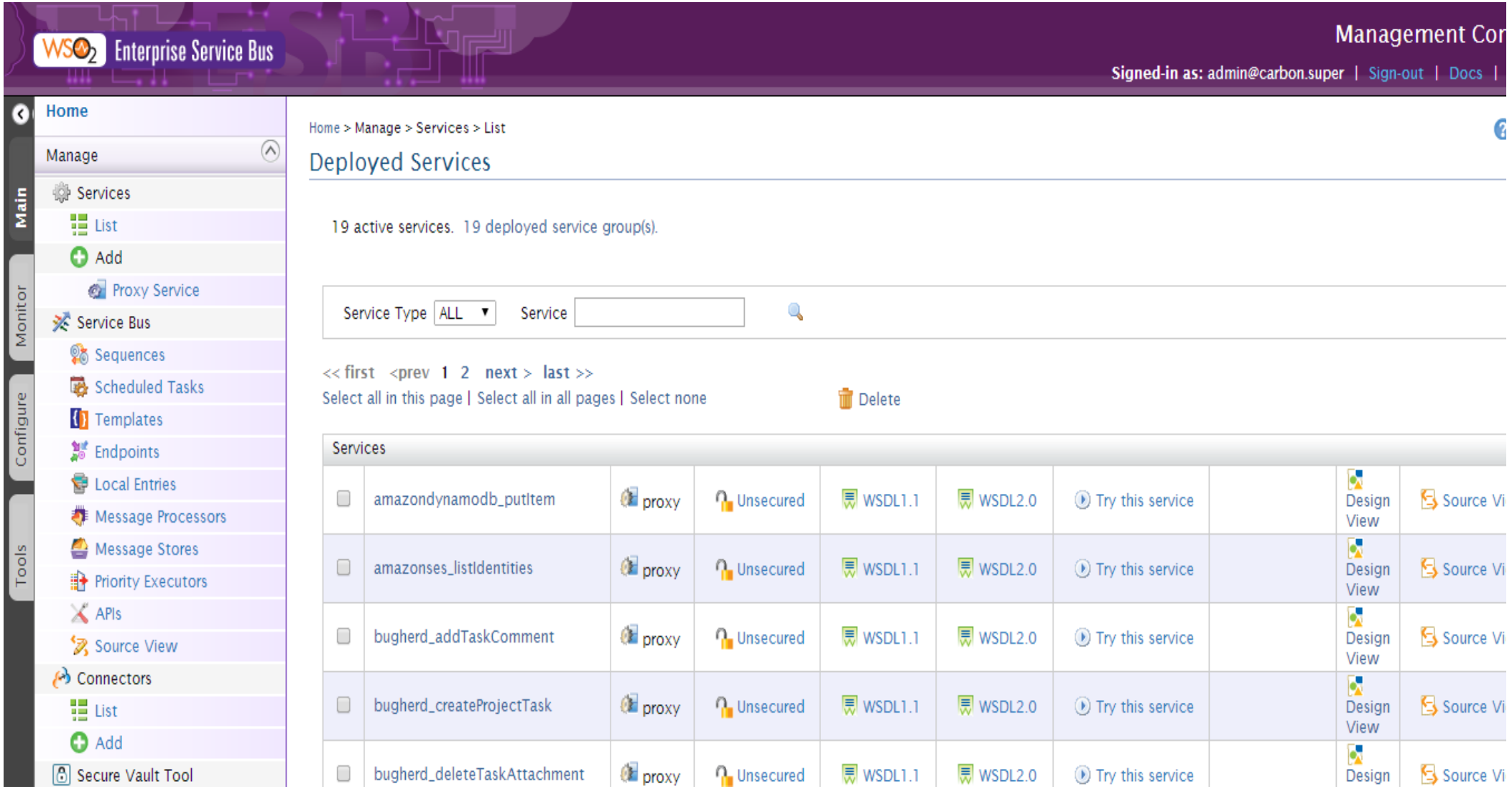
- **Custom Proxy** - A custom proxy service in which you customize the sequences, endpoints, transports, and other QoS settings.

Custom proxy service.




```
        startOnLoad="true">
   <target>
      <inSequence onError="faultHandlerSeq">
         <property name="region" expression="json-eval($.region)"/>
         <property name="secretAccessKey" expression="json-eval($.secretAccessKey)"/>
         <property name="returnConsumedCapacity"
                   expression="json-eval($.returnConsumedCapacity)"/>
         <property name="accessKeyId" expression="json-eval($.accessKeyId)"/>
         <property name="version" expression="json-eval($.version)"/>
         <property name="expected" expression="json-eval($.expected)"/>
         <property name="tableName" expression="json-eval($.tableName)"/>
         <property name="item" expression="json-eval($.item)"/>
         <property name="returnValues" expression="json-eval($.returnValues)"/>
         <property name="returnItemCollectionMetrics"
                   expression="json-eval($.returnItemCollectionMetrics)"/>
         <property name="conditionalOperator"
                   expression="json-eval($.conditionalOperator)"/>
         <amazondynamodb.init>
            <region>{$ctx:region}</region>
            <secretAccessKey>{$ctx:secretAccessKey}</secretAccessKey>
            <returnConsumedCapacity>{$ctx:returnConsumedCapacity}</returnConsumedCapacity>
            <accessKeyId>{$ctx:accessKeyId}</accessKeyId>
            <version>{$ctx:version}</version>
         </amazondynamodb.init>
         <amazondynamodb.putItem>
            <expected>{$ctx:expected}</expected>
            <tableName>{$ctx:tableName}</tableName>
            <item>{$ctx:item}</item>
            <returnValues>{$ctx:returnValues}</returnValues>
            <returnItemCollectionMetrics>{$ctx:returnItemCollectionMetrics}</returnItemCollectionMetrics>
            <conditionalOperator>{$ctx:conditionalOperator}</conditionalOperator>
         </amazondynamodb.putItem>
         <filter source="$axis2:HTTP_SC" regex="^[^2][0-9][0-9]">
            <then>
               <switch source="$axis2:HTTP_SC">
```

Sample proxy services

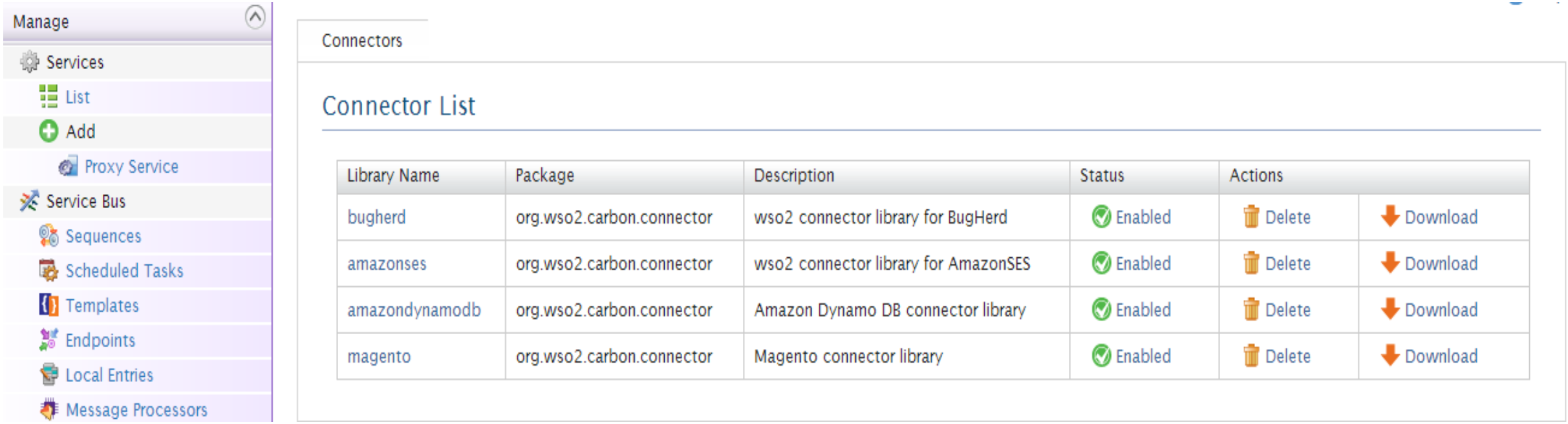


Connector Template Implementation

```
<template name="updateProjectTask" xmlns="http://ws.apache.org/ns/synapse">
  <parameter name="taskId" description="ID of the Project Task to be updated."/>
  <parameter name="task" description="The task JSON object to be sent to the API."/>
  <sequence>
    <property name="uri.var.task" expression="$func:task"/>
    <property name="uri.var.taskId" expression="$func:taskId"/>

    <payloadFactory media-type="json">
      <format>
        {"task":$1}
      </format>
      <args>
        <arg expression="get-property('uri.var.task')"/>
      </args>
    </payloadFactory>

    <property name="DISABLE_CHUNKING" value="true" scope="axis2"/>

    <call>
      <endpoint>
        <http method="put" uri-template="{uri.var.apiUrl}/api_v2/projects/{uri.var.projectId}/tasks/{uri.var.taskId}.json"/>
      </endpoint>
    </call>

    <!-- Remove custom Headers from the Response -->
    <header name="Via" scope="transport" action="remove"/>
    <header name="ETag" scope="transport" action="remove"/>
    <header name="X-Runtime" scope="transport" action="remove"/>
    <header name="X-Powered-By" scope="transport" action="remove"/>
    <header name="X-Rack-Cache" scope="transport" action="remove"/>
    <header name="X-Request-Id" scope="transport" action="remove"/>
    <header name="X-Frame-Options" scope="transport" action="remove"/>
    <header name="X-UA-Compatible" scope="transport" action="remove"/>
    <header name="X-XSS-Protection" scope="transport" action="remove"/>
```

ESB Connector

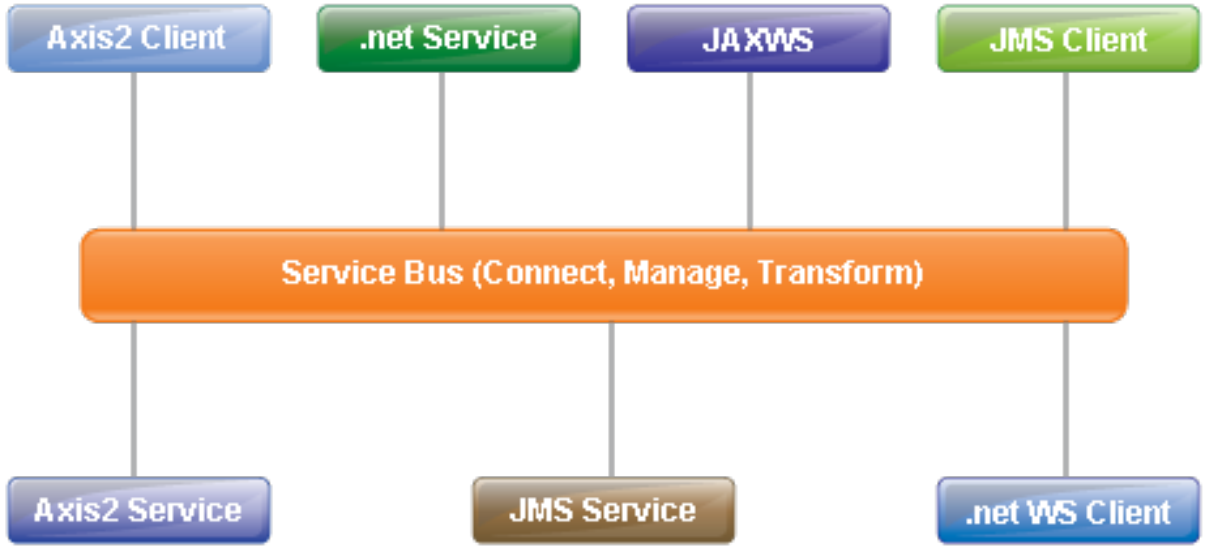
- Aggregation of **mediators** => is called **sequence**
- Aggregation of **sequences** => is called **template**
- Aggregation of **templates** => is called **connector**



Service Bus and Mediation

- Service Bus is the **common communication channel** that is used by all the parties in messaging.

- Mediation is the process of facilitated communication between parties by providing intermediary conflict resolutions.



References

- https://docs.wso2.com/display/ESB480/Using+a+Connector
- https://docs.wso2.com/display/ESB480/JIRA+Connector
- https://docs.wso2.com/display/ESB480/Managing+Connectors+in+Your+ESB+Instance
- https://docs.wso2.com/display/ESB480/Mediators