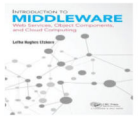
BITS Pilani
Pilani | Dubai | Goa | Hyderabad | Mumbai
WORK INTEGRATED LEARNING PROGRAMMES

IMP Note to Students

- It is important to know that just login to the session does not guarantee the attendance.
- Once you join the session, continue till the end to consider you as present in the class.
- IMPORTANTLY, you need to make the class more interactive by responding to Professors queries in the session.
- **Whenever Professor calls your number / name ,you need to respond, otherwise it will be considered as ABSENT**

Textbooks



T1: Letha Hughes Etzkorn - Introduction to middleware _ web services, object components, and cloud computing- Chapman and Hall_CRC (2017).

T2: William Grosso - Java RMI (Designing & Building Distributed Applications)

R1: Gregor Hohpe, Bobby Woolf - Enterprise Integration Patterns_ Designing, Building, and Deploying Messaging Solutions -Addison-Wesley Professional (2003)

R2: MongoDB in Action

Note: In order to broaden understanding of concepts as applied to Indian IT industry, students are advised to refer books of their choice and case-studies in their own organizations

BITS Pilani WILP

innovate achieve lead

BITS Pilani
Pilani | Dubai | Goa | Hyderabad | Mumbai
WORK INTEGRATED LEARNING PROGRAMMES

Modular Structure

No	Title of the Module
M1	Introduction and Evolution
M2	Enterprise Middleware
M3	Middleware Design and Patterns
M4	Middleware for Web-based Application and Cloud-based Applications
M5	Specialized Middleware

BITS Pilani
Pilani Campus

CS8: EAI: Message System Management

Agenda

- Challenges faced
  - Development
  - Testing
- Monitoring and Controlling
  - Control Bus
  - Detour
- Observing and Analyzing Message Traffic
  - Wire Tap
  - Message History
  - Message Store
  - Smart Proxy
- Testing and Debugging
  - Test Message Injection
  - Channel purger

Architect's dream, Developer's nightmare symptom.

- A message-based integration solution may produce, route and transform thousands or even millions of messages in a day.
- One has to deal with exceptions, performance bottlenecks and changes in the participating systems.
- Components are distributed across many platforms and machines that can reside at multiple locations.
- Architectural benefits of loose coupling actually make testing and debugging a system harder.
- Assumptions made about each other are less in such system and therefore provide flexibility.
- **Complexity for Testers:** Testing a system where a message producer is not aware of who the consumers of this message are, can be challenging.
- Asynchronous nature of messaging make things even more complicated.
  - For example, the messaging solution may not even be designed for the message producer to receive a reply message from the recipient(s).
  - Likewise, the messaging infrastructure typically guarantees the delivery of the message but not the delivery time.
  - This makes it hard to develop test cases that rely on the results of the message delivery.

BITS Pilani WILP

innovate achieve lead

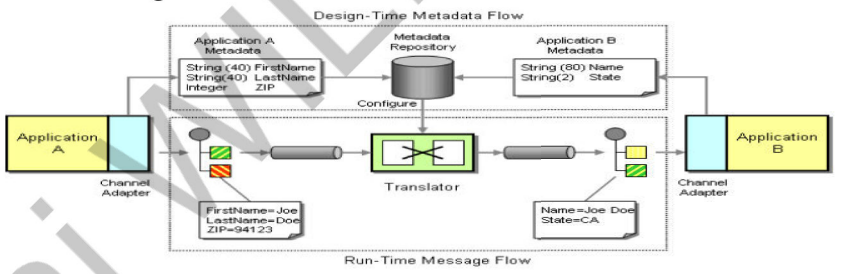
Message Transformations

Below are list of transformations types available

- Envelope Wrapper
- Content Enricher
- Content Filter
- Claim Check
- Normalizer
- Canonical Data Model

Message Transformations

- **Metadata Integration**

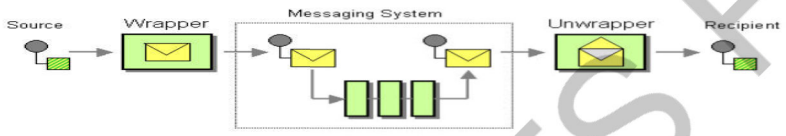


Metadata Integration

For example, the picture above depicts the integration between two applications that need to exchange customer information. Each system has a slightly different definition of customer data. Application A stores first and last name

Message Transformations

- **Wrapping and unwrapping**

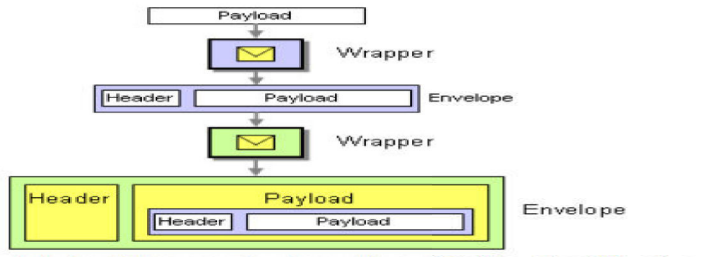


The process of wrapping and unwrapping a message consists of five steps:

- The Message Source publishes a message in a raw format. This format is typically determined by the nature of the application and does not comply with the requirements of the messaging infrastructure.
- The Wrapper takes the raw message and transforms it into a message format that complies with the messaging system. This may include adding message header fields, encrypting the message, adding security credentials etc.
- The Messaging System processes the compliant messages.
- A resulting message is delivered to the Unwrapper. The unwrapper reverses any modifications the wrapper made. This may include removing header fields, decrypting the message or verifying security credentials.
- The Message Recipient receives a 'clear text' message.

Message Transformations

- **Envelope structure**



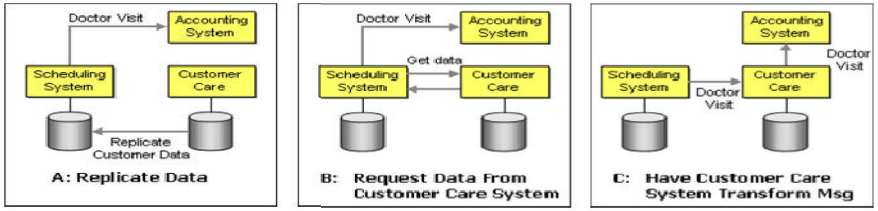
A Chain of Wrappers Creates a Hierarchical Envelope Structure

BITS Pilani WILP

innovate achieve lead

Message Transformations - Example

Let's consider the following example (see picture). A hospital scheduling system publishes a message announcing that the patient has completed a doctor's visit. The message contain's the patient's first name, his or her patient ID and the date of the visit. In order for the accounting system to log this visit and inform the insurance company, it requires the full patient name, the insurance carrier and the patient's social security number. However, the scheduling system does not store this information, it is contained in the customer care system. What are our options?



Message Transformations - Example

Option A: We could modify the scheduling system so it can store the additional information. When the customer's information changes in the customer care system (e.g. because the patient switches insurance carriers) the changes need to be replicated to the scheduling system. The scheduling system can now send a message that includes all required information. Unfortunately, this approach has two significant drawbacks. First, it requires a modification to the scheduling system's internal structure. In most cases, the scheduling system is going to be a packaged application and may not allow this type of modification. Second, even if the scheduling system is customizable, we need to consider that we are making a change to the system based on a the specific needs of another system. For example, if we also want to send a letter to the patient confirming the visit we would have to change the scheduling system again to accommodate the customer's mailing address. The integration solution would be much more maintainable if we decouple the scheduling system from the specifics of the applications that consume the "Doctor Visit" message.

Option B: Instead of storing the customer's information inside the scheduling system, the scheduling system could request the SSN and carrier data from the customer care system just 298 before it is sending the 'Doctor Visit' message. This solves the first problem – we no longer have to modify the storage of the scheduling system. However, the second problem remains: the scheduling system needs to know that the SSN and carrier information is required in order to notify the accounting system. Therefore, the semantics of the message is more similar to 'Notify Insurance' than 'Doctor Visit'. In a loosely coupled system we do not want one system to instruct the next one on what to do. We rather send an Event Message and let the other systems decide what to do. In addition, this solution couples the scheduling system more tightly to the customer care system because the scheduling system now needs to know where to get the missing data. This ties the scheduling system to both the accounting system and the customer care system. This type of coupling is undesirable because it leads to brittle integration solutions.

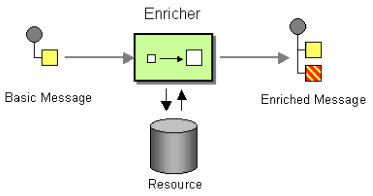
Message Transformations - Example

Option C: We can avoid some of these dependencies if we send the message to the customer care system first instead of the accounting system. The customer care system can then fetch all the required information and send a message with all required data to the accounting system. This decouples the scheduling system nicely from the subsequent flow of the message. However, now we implement the business rule that a the insurance company receives a bill after the patient visits the doctor inside the customer care system. This requires us to modify the logic inside the customer care system. If the customer care system is a packaged application, this modification may be difficult or impossible. Even if we can make this modification, we now make the customer care system indirectly responsible for sending bills messages. This may not be a problem if all the data items required by the accounting system are available inside the customer care system. If some of the fields have to be retrieved from other systems we are in a similar situation to where we started.

Option D (not shown): We could also modify the accounting system to only require the customer ID and retrieve the SSN and carrier information from the customer care system. This approach has two disadvantages. First, we now couple the accounting system to the customer care system. Second, this option again assumes that we have control over the accounting system. In most cases, the accounting system is going to be a packaged application with limited options for customization.

Message Transformations

- **Content Enricher**



- **Use a specialized transformer, a *Content Enricher*, to access an external data source in order to augment a message with missing information.**
- The *Content Enricher* uses information inside the incoming message (e.g. key fields) to retrieve data from an external source.
- After the *Content Enricher* retrieves the required data from the resource, it appends the data to the message.
- The original information from the incoming message may be carried over into the resulting message or may no longer be needed, depending on the specific needs of the receiving application.

BITS Pilani WILP

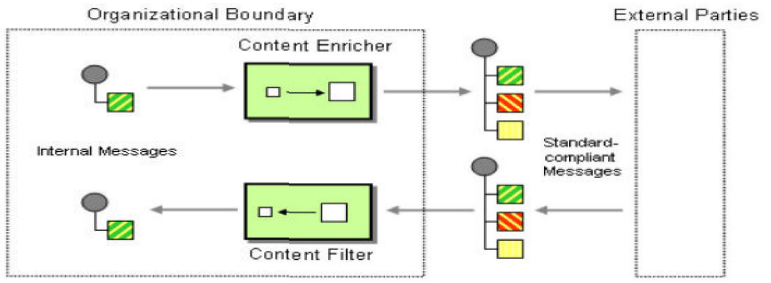
innovate achieve lead

BITS Pilani
Pilani | Dubai | Goa | Hyderabad | Mumbai
WORK INTEGRATED LEARNING PROGRAMMES

Message Transformations

innovate achieve lead

- **Content Enricher**

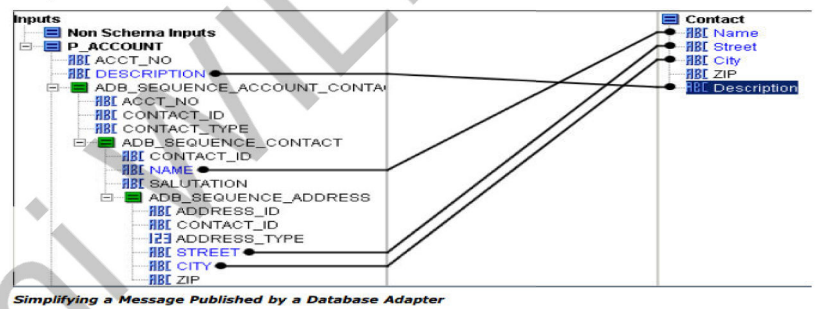


Using a Content Enricher / Content Filter Pair When Communicating with External Parties

Message Transformations

innovate achieve lead

- **Adapter**

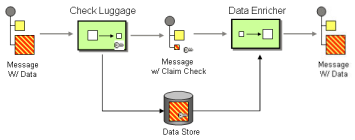


Simplifying a Message Published by a Database Adapter

Message Transformations

innovate achieve lead

- **Claim Check**



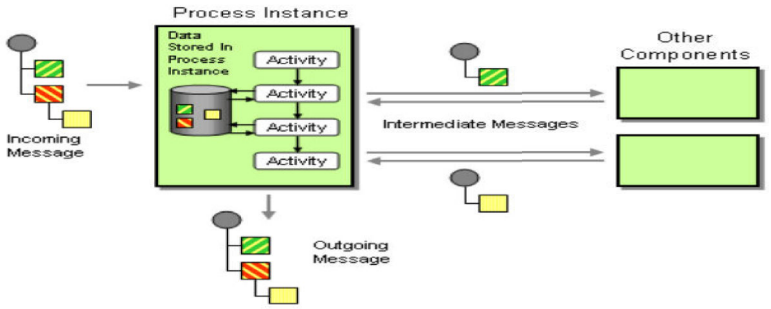
Store message data in a persistent store and pass a *Claim Check* to subsequent components. These components can use the *Claim Check* to retrieve the stored information.

The *Claim Check* pattern consists of the following steps:
1.A message with data arrives.
2.The "Check Luggage" component generates a unique key for the information. This key will be used later as the *Claim Check*
3.The Check Luggage component extracts the data from the message and stores it in a persistent store, e.g. a file or a database. It associates the stored data with the key .
4.It removes the persisted data from the message and adds the *Claim Check*.
5.Another component can use a *Content Enricher* to retrieve the data based on the *Claim Check*.
• This process is analogous to a luggage check at the airport. If you do not want to carry all your luggage with you, you simply check it with the airline counter.
• In return you receive a sticker on your ticket that has a reference number that uniquely identifies each piece of luggage you checked.
• Once you reach your final destination, you can retrieve your luggage.

Message Transformations

innovate achieve lead

- **Process Manager with Claim Check**



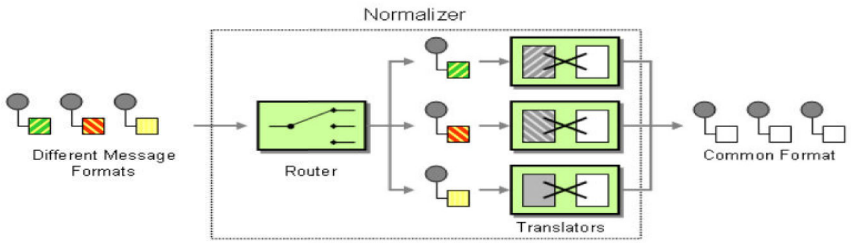
BITS Pilani WILP

innovate achieve lead

Message Transformations

- **Normalizer with Message Translator**

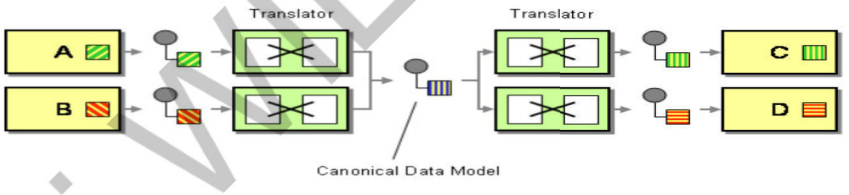
Use a *Normalizer* to route each message type through a custom *Message Translator* so that the resulting messages match a common format.



Message Transformations

- **Canonical Data Model**

Therefore, design a *Canonical Data Model* that is independent from any specific application. Require each application to produce and consume messages in this common format.



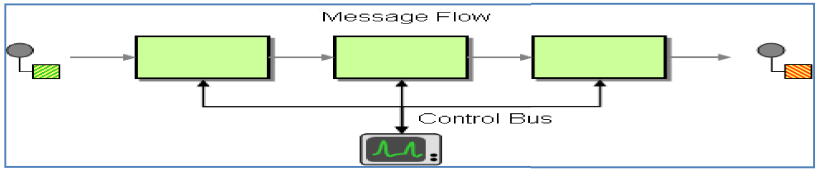
The *Canonical Data Model* provides an additional level of indirection between application's individual data formats. If a new application is added to the integration solution only transformation between the *Canonical Data Model* has to created, independent from the number of applications that already participate.

Monitoring and Controlling

- Messages can be tracked at two different levels of Abstraction.
  - System Management - Monitors how many messages are being sent or how long it took a message to be processed.
  - These monitoring solutions do not inspect the message data except maybe for some fields in the message header such as the message identifier or the Message History.
  - Business Activity Monitoring (BAM) - Focus on the payload data contained in the message, for example the Dollar value of all orders placed in the last hour.
- **Note** - Many of the patterns presented in this section are general enough that they can be used for either purpose.
- However, because business activity monitoring is a whole new field in itself and shares many complexities with data warehousing (something we have not touched on at all), we decided to discuss the patterns in the context of system management.

Control Bus

- Use a Control Bus to manage an enterprise integration system.
- The Control Bus uses the same messaging mechanism used by the application data, but uses separate channels to transmit data that is relevant to the management of components involved in the message flow.
- One can use additional steps, such as validation or logging for troubleshooting.

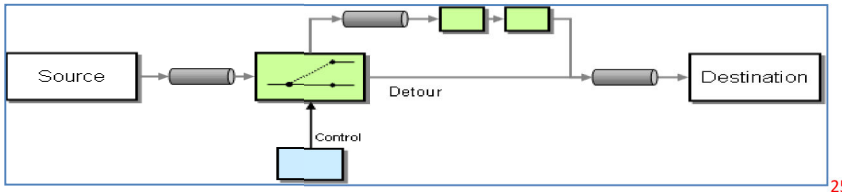


BITS Pilani WILP

innovate achieve lead

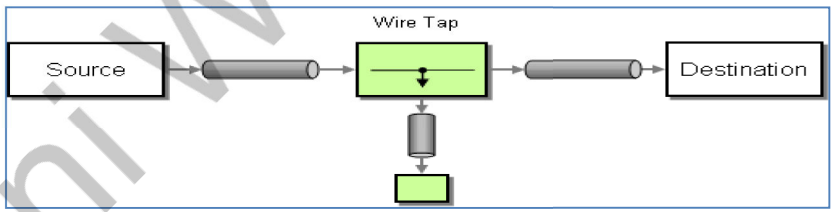
Detour (Validation, Testing or Debugging)

- The Detour uses a simple context-based router with two output channels. One output channel passes the unmodified message to the original destination.
- When instructed by the Control Bus, the Detour routes messages to a different channel. This channel sends the message to additional components that can inspect and/or modify the message. Ultimately, these components route the message to the same destination.



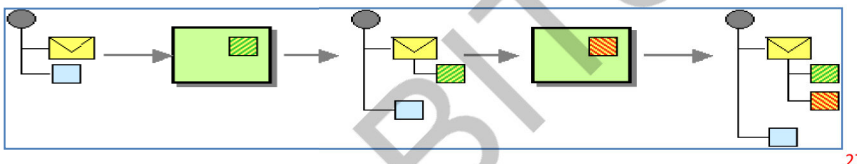
Wire Tap

- The Wire Tap is a fixed Recipient List with two output channels. It consumes messages off the input channel and publishes the unmodified message to both output channels.
- To insert the Wire Tap into a channel, you need to create an additional channel and change the destination receiver to consume of the second channel



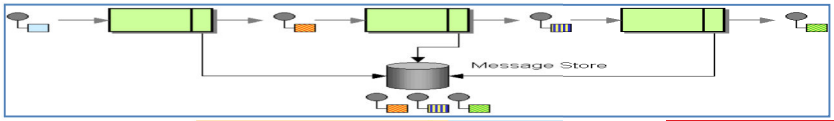
Message History

- Message History maintains a list of all components that the message passed through and its origination.
- Every component that processes the message (including the originator) adds one entry to the list.
- Message History should be part of the message header and not application data because it contains system-specific control information.



Message Store

- Message Store captures information about each message in a central location.
- Taking advantage of the asynchronous nature of a messaging infrastructure, a message when send to a channel, a duplicate of the same message is send to a special channel to be collected by the Message Store. This can be performed by the component itself or we can insert a Wire Tap into the channel.
- One can consider the secondary channel that carries a copy of the message as part of the Control Bus.
- Sending a second message in a 'fire-and-forget' mode will not slow down the flow of the main application messages. It does, however, increase network traffic.
- That's why we may not store the complete message, but just a few key fields that are required for later analysis, such as a message ID, or the channel on which the message was sent and a timestamp.

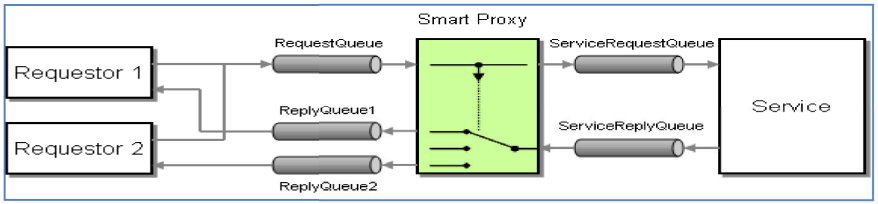


BITS Pilani WILP

innovate achieve lead

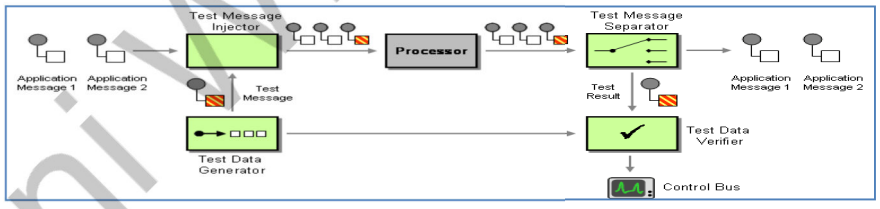
Smart Proxy

- In order to track messages sent to request-reply services, a Smart Proxy is inserted into the message stream.
- Smart Proxy stores the Return Address supplied by the original requestor and replace it with the address of the Smart Proxy. When the service sends the reply message route it to the original Return Address.



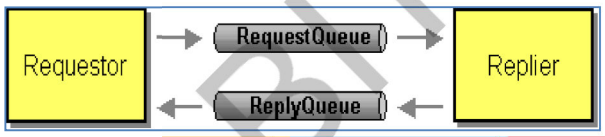
Testing and Debugging

- Testing a messaging system before deploying it into production is a good idea. But testing should not stop there. You should be actively verifying that the running messaging system is functioning properly.
- You can do this by periodically injecting a Test Message into the system and verifying the results.



Testing and Debugging

- **Test Message:** Troubleshooting message which a component is actively processing messages, but garbles outgoing messages due to an internal fault can be handled using a Test Message.
- **Channel Purger:** When a component fails or misbehaves, it is easy to end up with unwanted messages on a channel. During testing it is very useful to remove all remaining messages from a channel so that the components under test do not receive 'leftover' messages. A Channel Purger does that for us.Use a Channel Purger to remove unwanted messages from a channel.



Case Study: Online Order Processing System

- **Test Message:** Case Study: Online Order Processing System
- An e-commerce company, "OrderMax," processes thousands of orders daily.
- Their system integrates with multiple suppliers, shipping providers, and payment gateways.
- To ensure seamless communication between these systems, OrderMax uses various message transformation middleware technologies.
- 1. Envelope Wrapper
- Problem: Suppliers require different formats for order headers (e.g., order ID, date, and customer info).

BITS Pilani WILP

innovate achieve lead

BITS Pilani
Pilani | Dubai | Goa | Hyderabad | Mumbai
WORK INTEGRATED LEARNING PROGRAMMES

Case Study: Online Order Processing System

- Solution: OrderMax uses an Envelope Wrapper to wrap the order payload with the required headers for each supplier.
- **Original Message:**

```
{ "orderId": "123",
  "customerName": "John Doe",
  "items": [...] }
```

- **Wrapped Message (for Supplier A):**

```
{  "header": {
    "orderId": "123",
    "date": "2023-02-15"
  },
  "payload": {
    "customerName": "John Doe",
    "items": [...] } }
```


Case Study: Online Order Processing System

- **2. Content Enricher**
- Problem: Shipping providers require additional customer information (e.g., phone number, email) not present in the original order message.
- Solution: OrderMax uses a Content Enricher to fetch the required information from the customer database and add it to the message.
- Example:
- Original Message:

```
{
  "orderId": "123",
  "customerId": "456",
  "items": [...]
}
```


Case Study: Online Order Processing System

- Enriched Message:

```
{
  "orderId": "123",
  "customerId": "456",
  "customerPhone": "123-456-7890",
  "customerEmail": "johndoe@example.com",
  "items": [...]
}
```

- **3. Content Filter**
- Problem: Payment gateways only require a subset of order information (e.g., order ID, amount, and customer ID).
- Solution: OrderMax uses a Content Filter to remove unnecessary information from the message.

Case Study: Online Order Processing System

- Example: Original Message:

```
{
  "orderId": "123",
  "customerName": "John Doe",
  "items": [...],
  "amount": 100.00
}
```

- Filtered Message:

```
{
  "orderId": "123",
  "customerId": "456",
  "amount": 100.00
}
```


BITS Pilani WILP

innovate achieve lead

Case Study: Online Order Processing System

innovate achieve lead

- **4. Claim Check**

- Problem: Large order messages with many items exceed the message size limit for some suppliers.
- Solution: OrderMax uses a Claim Check to store the large message payload in a repository and include a claim check (reference ID) in the message.

- Example:

- Original Message:
- {
- "orderId": "123",
- "items": [...] // large payload
- }

Case Study: Online Order Processing System

innovate achieve lead

- **Transformed Message:**

- {
- "orderId": "123",
- "claimCheckId": "CCI-123"
- }

- The supplier can then retrieve the payload using the claim check ID.

- **5. Canonical Model**

- Problem: Different suppliers and shipping providers use varying formats for order messages.
- Solution: OrderMax uses a Canonical Model to define a standard order format, which is then transformed to match each external system's requirements.

Case Study: Online Order Processing System

innovate achieve lead

- **Example: - Canonical Order Model:**

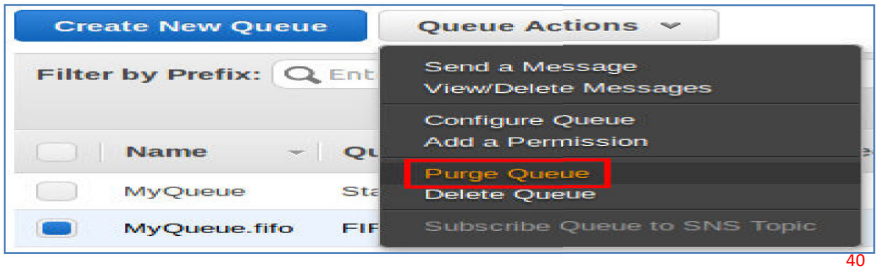
- {
- "orderId": "",
- "customer": {
- "name": "",
- "email": "",
- "phone": ""
- },
- "items": [...] }

- Transformed messages for each supplier/shipping provider would adhere to this canonical model, making it easier to integrate and maintain multiple systems.
- By applying these message transformation technologies, OrderMax ensures seamless integration with its suppliers, shipping providers, and payment gateways, reducing errors and improving overall system efficiency.

Chanel Purger Amazon-SQS.

innovate achieve lead

- Many hosted messaging systems have a built-in function to purge channels. For example, with Amazon's Simple Queuing Service (SQS), you can purge a queue simply from the web user interface.



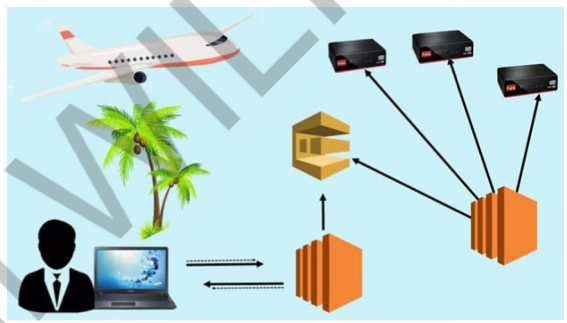
BITS Pilani WILP

innovate achieve lead

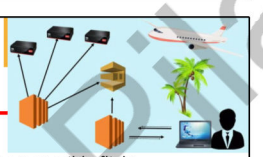
Chanel Purger Amazon-SQS.

- SQS stands for Simple Queue Service.
- SQS was the first service available in AWS.
- Amazon SQS is a web service that gives you access to a message queue that can be used to store messages while waiting for a computer to process them.
- Amazon SQS is a distributed queue system that enables web service applications to quickly and reliably queue messages that one component in the application generates to be consumed by another component where a queue is a temporary repository for messages that are awaiting processing.
- With the help of SQS, you can send, store and receive messages between software components at any volume without losing messages.
- Using Amazon sqs, you can separate the components of an application so that they can run independently, easing message management between components.

Chanel Purger Amazon-SQS.



Chanel Purger Amazon-SQS.



- Suppose the user wants to look for a package holiday and wants to look at the best possible flight.
- User types a query in a browser, it then hits the EC2 instance.
- An EC2 instance looks "What the user is looking for?", it then puts the message in a queue to the SQS.
- An EC2 instance pulls queue. An EC2 instance continuously pulling the queue and looking for the jobs to do.
- Once it gets the job, it then processes it.
- It interrogates the Airline service to get all the best possible flights.
- It sends the result to the web server, and the web server sends back the result to the user.
- User then selects the best flight according to his or her budget.

Chanel Purger Amazon-SQS.

- **Here's a real-life case study on using a Chained Purger with Amazon SQS:**
- Case Study: Order Processing System with Amazon SQS and Chained Purger
- A large e-commerce company, "ShopZone," uses Amazon SQS to handle their order processing workflow.
- ShopZone's system consists of multiple microservices, each responsible for a specific task, such as order validation, inventory management, and payment processing.
- Problem:
- ShopZone's order processing system was experiencing issues with:
- Message duplication: Duplicate order messages were being processed, causing inconsistencies in their inventory and payment systems.

BITS PilaniWILP

innovate achieve lead

BITS Pilani
Pilani | Dubai | Goa | Hyderabad | Mumbai
WORK INTEGRATED LEARNING PROGRAMMES

Chanel Purger Amazon-SQS.

- Message ordering: Orders were not being processed in the correct sequence, leading to errors and retries.
- Solution:
- ShopZone implemented a Chained Purger pattern using Amazon SQS to address these issues.
- Here's how they did it:
- Architecture:
- Order Queue: ShopZone created an SQS queue (Order Queue) where orders are sent by their web application.
- Deduplication Queue: A second SQS queue (Deduplication Queue) is used to store unique order messages.
- Chained Purger: A Lambda function (Chained Purger) is triggered by the Order Queue. It checks for duplicate orders and purges them, ensuring only unique orders are sent to the Deduplication Queue.

Chanel Purger Amazon-SQS.

- Chained Purger Logic:
- Message Receipt: The Chained Purger Lambda function receives order messages from the Order Queue.
- Duplicate Check: It checks if the order message is a duplicate by verifying the order ID against a cache (e.g., Redis) or a database.
- Purge Duplicates: If a duplicate order is found, the Chained Purger deletes the message from the Order Queue.
- Send Unique Messages: If the order is unique, the Chained Purger sends the message to the Deduplication Queue.

Chanel Purger Amazon-SQS.

- Benefits:
- Eliminated Duplicate Processing: By purging duplicate orders, ShopZone eliminated inconsistencies in their inventory and payment systems.
- Ensured Correct Order Processing: The Chained Purger ensured that orders are processed in the correct sequence, reducing errors and retries.
- Improved System Efficiency: By removing duplicates and ensuring correct order processing, ShopZone's system efficiency improved, and they experienced fewer errors

Chanel Purger Amazon-SQS.

```
const AWS = require('aws-sdk');
const redis = require('redis');

exports.handler = async (event) => {
  const sqs = new AWS.SQS({ region: 'your-region' });
  const redisClient = redis.createClient({ host: 'your-redis-host' });

  for (const record of event.Records) {
    const orderId = JSON.parse(record.body).orderId;

    // Check for duplicate orders in Redis
    const isDuplicate = await redisClient.exists(orderId);
```


BITS Pilani WILP

innovate achieve lead

Chanel Purger Amazon-SQS.

```
if (isDuplicate) {
  // Delete duplicate message from Order Queue
  await sqs.deleteMessage({

QueueUrl: 'https://sqs.your-region.amazonaws.com/your-account-id/OrderQueue',
    ReceiptHandle: record.receiptHandle,
  }).promise();
```

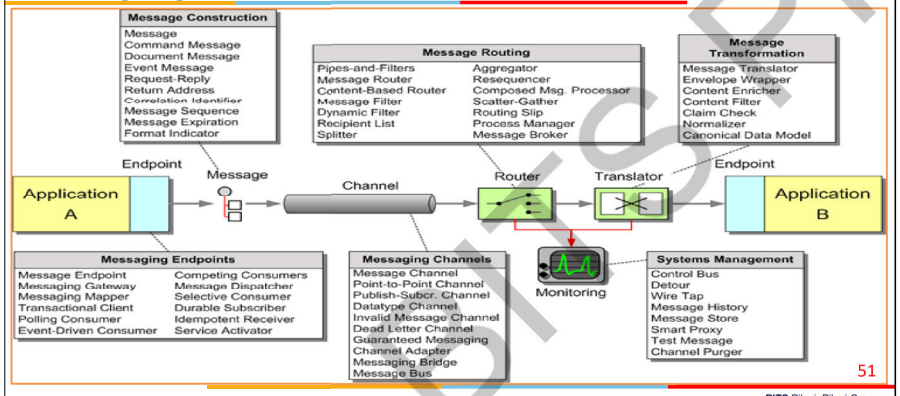

Chanel Purger Amazon-SQS.

```
} else {
    // Send unique message to Deduplication Queue
    await sqs.sendMessage({
      QueueUrl: 'https://sqs.your-region.amazonaws.com/your-account-id/DeduplicationQueue',
      MessageBody: record.body,
    }).promise();

    // Set order ID in Redis with a TTL (e.g., 1 hour)
    await redisClient.set(orderId, 'processed', 'EX', 3600);
  }
}

return { statusCode: 200 };
};
```


Messaging Overview



Thank You

BITS Pilani WILP

BITS Pilani WILP

BITS Pilani
Pilani | Dubai | Goa | Hyderabad | Mumbai
WORK INTEGRATED LEARNING PROGRAMMES

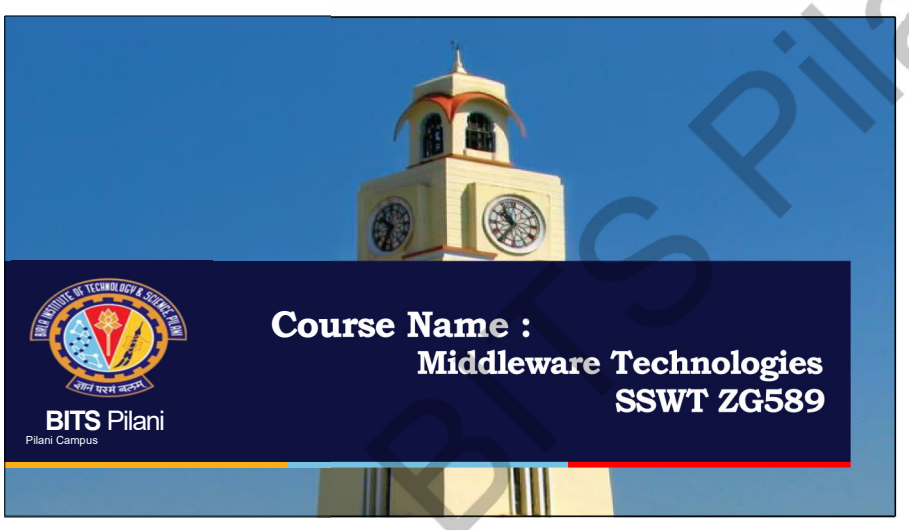
innovate achieve lead

Start Recording

innovate achieve lead

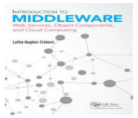
IMP Note to Students

- It is important to know that just login to the session does not guarantee the attendance.
- Once you join the session, continue till the end to consider you as present in the class.
- IMPORTANTLY, you need to make the class more interactive by responding to Professors queries in the session.
- **Whenever Professor calls your number / name ,you need to respond, otherwise it will be considered as ABSENT**



innovate achieve lead

Textbooks



T1: Letha Hughes Etzkorn - Introduction to middleware _ web services, object components, and cloud computing- Chapman and Hall_CRC (2017).

T2: William Grosso - Java RMI (Designing & Building Distributed Applications)

R1: Gregor Hohpe, Bobby Woolf - Enterprise Integration Patterns_ Designing, Building, and Deploying Messaging Solutions -Addison-Wesley Professional (2003)

R2: MongoDB in Action

Note: In order to broaden understanding of concepts as applied to Indian IT industry, students are advised to refer books of their choice and case-studies in their own organizations

BITS Pilani WILP

innovate achieve lead

BITS Pilani
Pilani | Dubai | Goa | Hyderabad | Mumbai
WORK INTEGRATED LEARNING PROGRAMMES

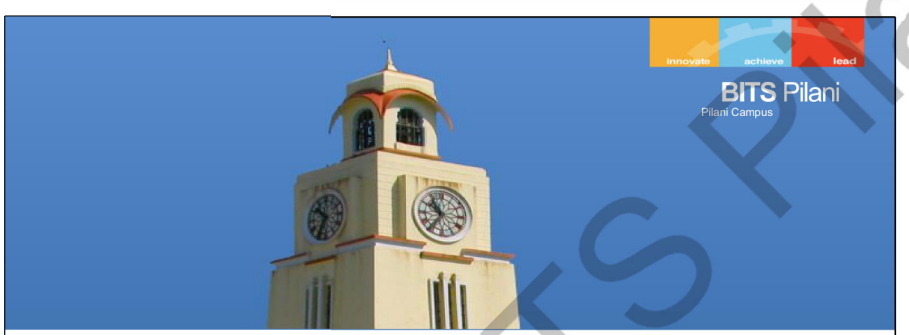
Evaluation Components

Evaluation Component	Name	Type	Weight	Duration	Schedule
EC – 1	Quiz I & II Assignment/ Laboratory Exercises	Individual / Take-home Practical	10% 10%		Pre/Post Mid-Sem TBA
EC – 2	Mid-Semester Examination	Closed Book	30%	2 Hrs.	TBA
EC – 3	End-Semester Examination	Open Book	50%	3 Hrs.	TBA

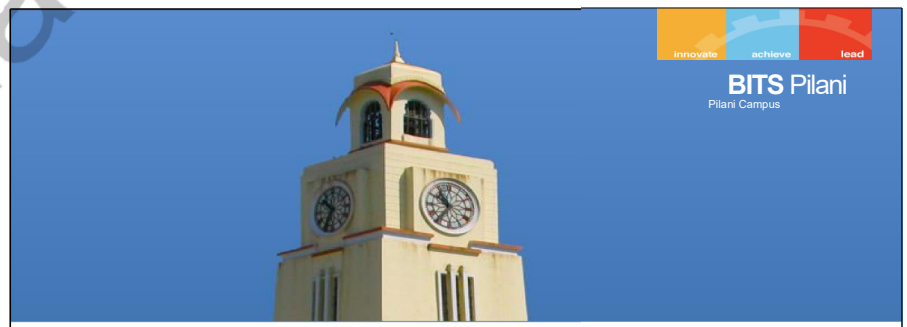
Modular Structure

innovate achieve lead

No	Title of the Module
M1	Introduction and Evolution
M2	Enterprise Middleware
M3	Middleware Design and Patterns
M4	Middleware for Web-based Application and Cloud-based Applications
M5	Specialized Middleware



CS 9 – CS15 : REVISIT



CS 9 : Web Services and Service-Oriented Architectures

BITS Pilani WILP

innovate achieve lead

BITS Pilani
Pilani | Dubai | Goa | Hyderabad | Mumbai
WORK INTEGRATED LEARNING PROGRAMMES

Agenda

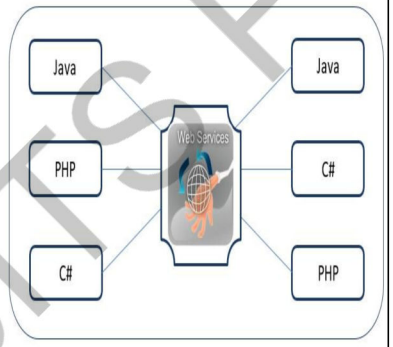
- Web Services Architectures
- Service-Oriented Architectures
- Architectures Styles, Relationship and Implementation

Introduction

- Web service is an appropriate medium to propagate communication between the client and server applications on the World Wide Web. It is a software module which is designed to perform a certain set of tasks as follows:
  - The web services can be searched for over the network and can also be invoked accordingly.
  - When invoked the web service would be able to provide the functionality to the client which invokes that web service.
- A way to expose the functionality of an information system and making it available through standard web technologies.
- "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" [W3C]

Web Services

- Any service that :
  - Is available over the Internet or private (intranet) networks
  - Uses a standardized XML messaging system
  - Is not tied to any one operating system or programming language
  - Is self-describing via a common XML grammar
  - Is discoverable via a simple find mechanism



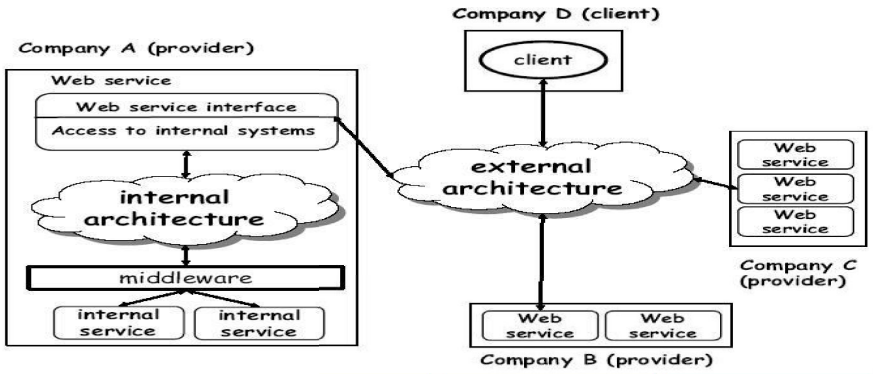
Web Service Architecture

- Web service architecture can be viewed using the either of the way:
  - The first is to examine the individual roles of each Web service actor.
  - The second is to examine the emerging web service protocol stack.
- Internal architecture
  - Web services expose internal operations to be invoked through the web
  - Receive requests through the web
  - Pass the requests to the underlying IT system
- External architecture
  - A middleware architecture which integrates different web services

BITS Pilani WILP

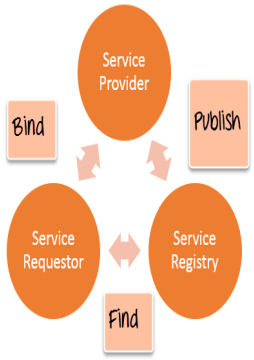
innovate achieve lead

Web service architecture comprises Internal and external architecture

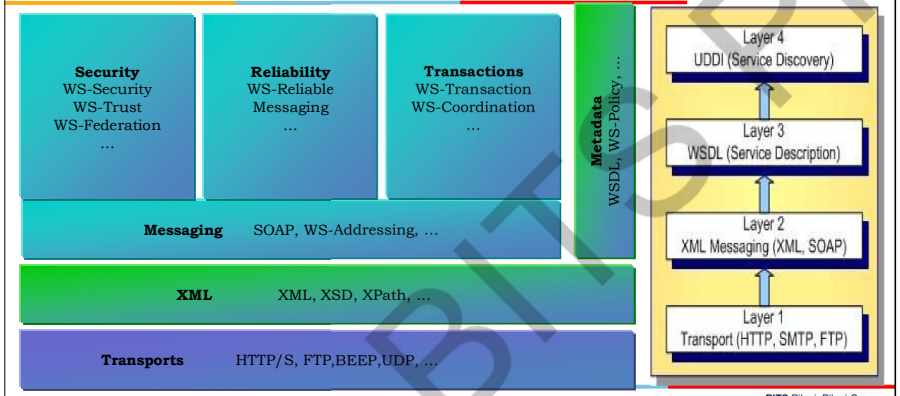


Web Service Roles

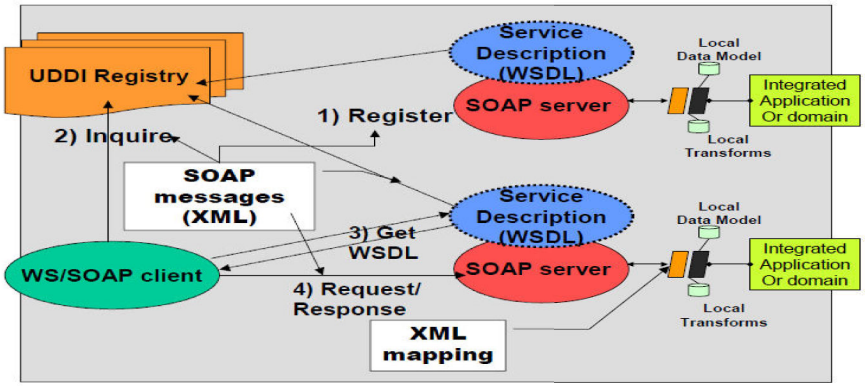
- **Service Provider:** This is the provider of the web service. The service provider implements the service and makes it available on the Internet.
- **Service Requestor:** This is any consumer of the web service. The requestor utilizes an existing web service by opening a network connection and sending an XML request.
- **Service Registry:** This is a logically centralized directory of services. The registry provides a central place where developers can publish new services or find existing ones. It therefore serves as a centralized clearing house for companies and their services.



Web Service Protocol Stack



Web Services In Action



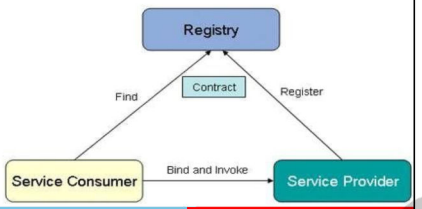
BITS Pilani WILP

BITS Pilani
Pilani | Dubai | Goa | Hyderabad | Mumbai
WORK INTEGRATED LEARNING PROGRAMMES

Web service components and Technologies

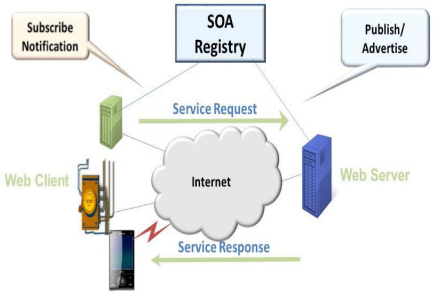
- **SOAP** – "Simple Object Access Protocol"
  - A simple way to send documents (some people have called it "email for documents")
  - How to format XML documents for transmission
- **WSDL** - "Web Services Description Language"
  - Defines all details about a service
- **UDDI** - "Universal Description, Discovery and Integration"
  - One way to advertise and discover services

- **Web Service implementation:**
  - **Build and Publish**
  - **Find**
  - **Bind and Invoke**



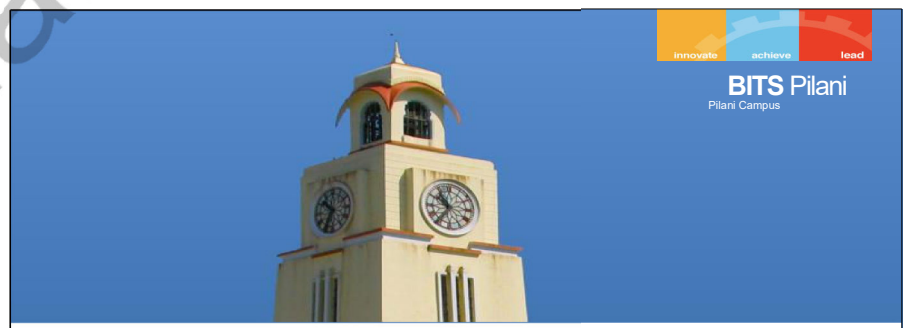
Service Oriented Architecture - Overview

- A Service Oriented Architecture (SOA) is a form of distributed systems architecture that is typically characterized by the following properties
  - Logical view
  - Message Orientation
  - Description Orientation
  - Granularity
  - Network Orientation
  - Platform Neutrality
- Implementation of SOA
  - Arbitrary Web services (SOAP/XML)
  - RESTful Web services
  - Apache Thrift



SOA Maturity Levels

Maturity level	SOA view	Benefits and metrics	Business involvement	Methodology	Service sourcing	Governance
Initial	Fine-grained software components	Promise of reuse; no metrics	Hidden from business	Minor adaptation of current software methods	Largely derived from existing application programming interfaces	Basic service definition policies at project level
Managed	Emerged software architecture	Standardization of data and resources	Services exposed as part of project cost	Project-level service definition methods	In-house development of new fine-grained software components	Service policies managed by registry monitors
Defined	Business process support	Business process redesign	Services part of requirement capture	Business service definition methods	External sourcing possible by defined function	Full set of service policies and metrics in place
Quantitatively managed	Enterprise service architecture	Agility and flexibility	Organizational "service thinking"	Intra- and inter-organizational service definition	Enterprise service bus to coordinate services	Business and IT governance metrics aligned
Optimized	Adaptive architecture	Autonomic systems	Value-stream "service thinking"	Value-chain service optimization	Full integration up and down the stack	Policy feedback used to adjust delivery



CS 10: Non-Rest Web Services

BITS Pilani WILP

innovate achieve lead

BITS Pilani
Pilani | Dubai | Goa | Hyderabad | Mumbai
WORK INTEGRATED LEARNING PROGRAMMES

Agenda

- Web Services Types
- SOAP Messaging Protocol
- Web Services Description Language (WSDL)
- Java API for XML Web Services (JAX-WS)

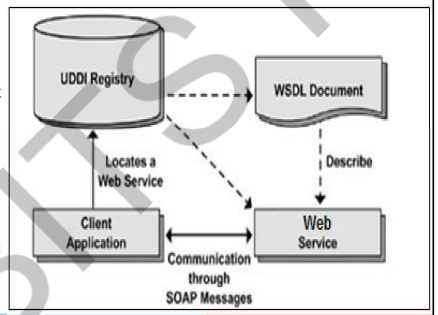
Web Services Types

- Two types of web services as follows:
- Non-REST (SOAP) Web Services (CS 10)
- RESTful Web Services (CS 11)

- Non-REST Web Services
  - First version of web services – released in late '90s
  - Implemented as arbitrary web services grouped together based on business context
  - Analogous to functional services – focus on functionality
  - Use SOAP (Simple Object Access Protocol) for data transfer over HTTP/S
  - Has embedded security via WS-Security header
  - Supports XML (SOAP is a restricted version of XML)

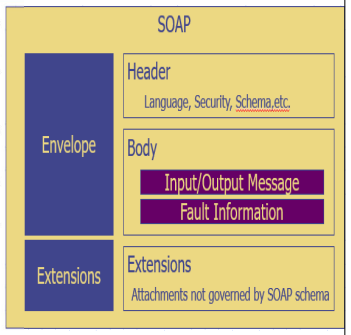
SOAP

- SOAP is an XML-based protocol to let applications exchange information over HTTP. Or more simple: SOAP is a protocol for accessing a Web Service.
- SOAP is a communication protocol
- SOAP is a format for sending messages
- SOAP is designed to communicate via Internet
- SOAP is platform independent
- SOAP is language independent
- SOAP is based on XML
- SOAP is simple and extensible
- SOAP allows you to get around firewalls
- SOAP is a W3C standard



SOAP Message Protocol

- Transfers structured information in the form of XML documents
- SOAP message is bound to the underlying protocol (mostly HTTP)
  - SOAP 1.1 – HTTP POST only
  - SOAP 1.2 (via web methods) – HTTP GET/POST/DELETE
- SOAP message goes from sender to receiver via intermediary nodes (SOAP nodes)
  - SOAP nodes may process SOAP headers and body or may add some additional headers for QoS / Security
- SOAP Message exchange patterns
  - Request-Reply
  - Response-only



SOAP Message

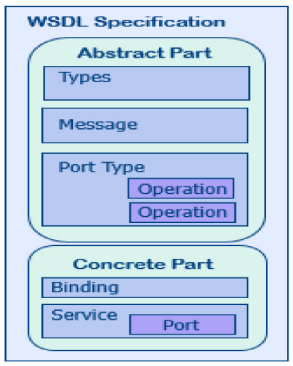
BITS Pilani WILP

innovate achieve lead

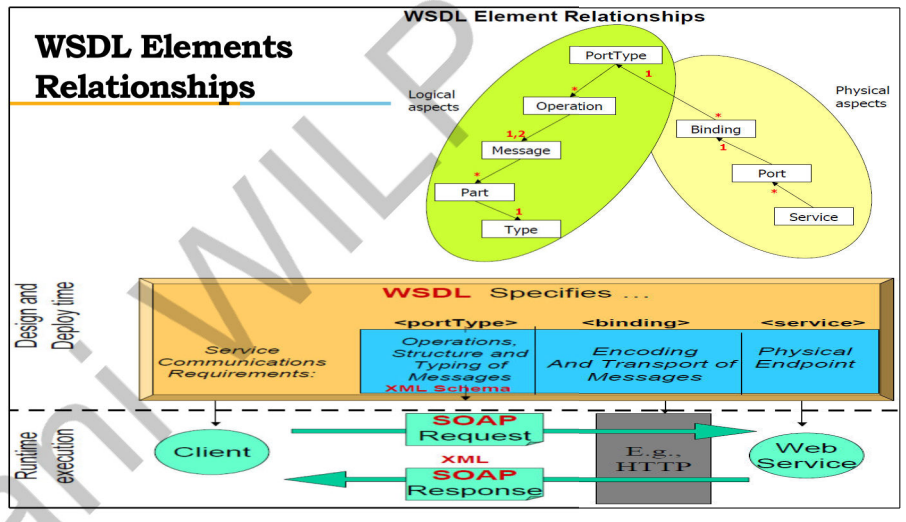
BITS Pilani
Pilani | Dubai | Goa | Hyderabad | Mumbai
WORK INTEGRATED LEARNING PROGRAMMES

Web Services Description Language

- WSDL stands for Web Services Description Language
- WSDL is based on XML
- WSDL is used to describe Web services
- WSDL is used to locate Web services
- WSDL is a W3C standard
- Defines the interface to a web service
  - Types
    - XML schema describing used data types
  - Messages
    - Necessary to invoke an operation of the service
  - Operations
    - Reference to input/output message
  - Port type
    - Set of operations that conform an instance of a service
  - Binding
    - Actual protocol to be used to invoke the operations
  - Services and ports
    - References to actual location of service



WSDL Elements Relationships



JAVA API FOR XML WEB SERVICES (JAX-WS)

- Java API for creating SOAP web services, part of Java EE
- Server side code
- Service publisher
- Creating client with wsimport
  - **wsimport -keep http://localhost:8080/hello?wsdl -d .**

```
package myHelloWorld;
import javax.jws.WebMethod;
import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;
import javax.jws.soap.SOAPBinding.Style;

//Service Endpoint Interface
@WebService
@SOAPBinding(style = Style.RPC)
public interface HelloWorld{

    @WebMethod String HelloWorld(String name);
}
```

```
package myHelloWorld;
import javax.jws.WebService;

//Service Implementation
@WebService(endpointInterface = "myHelloWorld.HelloWorld")
public class HelloWorldImpl implements HelloWorld{

    @Override
    public String HelloWorld(String name) {
        System.out.println(name+" says hello");
        return "Hello World " + name;
    }
}
```

```
package myHelloWorld;
import javax.xml.ws.Endpoint;
import myHelloWorld.HelloWorldImpl;

//Endpoint publisher
public class HelloWorldPublisher{

    public static void main(String[] args) {
        Endpoint.publish("http://localhost:8080/hello", new HelloWorldImpl());
    }
}
```

JAVA API FOR XML WEB SERVICES (JAX-WS)

- Client Stub

```
import myhelloworld.HelloWorld;
import myhelloworld.HelloWorldImplService;
public class HelloWorldClient {
    public static void main(String[] args) {
        HelloWorldImplService myHelloWorld = new HelloWorldImplService();
        HelloWorld myinterface = myHelloWorld.getHelloWorldImplPort();
        //Note the format of the operation call "helloWorld".
        //This matches the format in the wsimport-generated HelloWorld.java file.
        String response = myinterface.helloWorld(args[0]);

    System.out.println(response);
    }
}
```

- Dynamic Proxy Client

```
package myHelloWorld;

import java.net.URL;
import javax.xml.namespace.QName;
import javax.xml.ws.Service;
import myHelloWorld.HelloWorld;
/* This example does not use wsimport. Instead, it
creates a service instance manually. */

public class HelloWorldClient{
    public static void main(String[] args) throws Exception {
        URL location_of_wsdl = new URL("http://localhost:8080/hello?wsdl");
        QName name_of_service = new QName("http://myHelloWorld/", "Hello
            WorldImplService");
        Service service = Service.create(location_of_wsdl, name_of_service);
        HelloWorld hello = service.getPort(HelloWorld.class);
        String response = hello.HelloWorld(args[0]);
        System.out.println(response);
    }
}
```


BITS Pilani WILP

innovate achieve lead

BITS Pilani
Pilani | Dubai | Goa | Hyderabad | Mumbai
WORK INTEGRATED LEARNING PROGRAMMES

UDDI

- UDDI is a directory service where companies can register and search for Web services.
- UDDI stands for Universal Description, Discovery and Integration
- UDDI is a directory for storing information about web services
- UDDI is a directory of web service interfaces described by WSDL
- UDDI communicates via SOAP
- A specification that defines
  - how to interact with a registry
  - What the entries on the registry look like
- Interaction with UDDI
  - Registration
    - Adding new service descriptions to the registry
  - Lookup
    - Queries to search for right services
- Types of UDDI registry
  - Public
    - Open search-engines for web services
  - Private
    - Created by companies for their own use

BITS Pilani
Pilani Campus

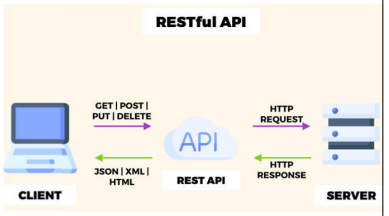
CS 11: Rest Services, JAX-RS Basics, Annotations and implementation

Agenda

- Rest Services
- JAX-RS Basics and Annotations
- JAX-RS implementations

RESTful Web Services

- REST – Architectural Constraints
- JAVA API FOR RESTFUL WEB SERVICES (JAX-RS)
  - Annotations
  - Server deployment
  - Client API
  - JSON API



- What Is a REST API?
  - An API (application programming interface) is a set of functions and protocols that enables communications between different software applications or systems.
  - The API establishes a collection of commands that users can invoke to retrieve the information that they're looking for in an organized, programmatic manner.

BITS Pilani WILP

innovate achieve lead

BITS Pilani
Pilani | Dubai | Goa | Hyderabad | Mumbai
WORK INTEGRATED LEARNING PROGRAMMES

RESTful API

- REST (representational state transfer) is a style of software architecture that defines constraints on how different systems communicate.
- An API that complies with these REST principles is known as a "REST API."
- What Is a REST API (RESTful API)?
- REST suggests to create an object of the data requested by the client and send the values of the object in response to the user.
- For example, if the user is requesting for a movie in City at a certain place and time, then you can create an object on the server side.
- REST stands for Representational State Transfer.
- It is a software architecture style that relies on a stateless communications protocol, most commonly, HTTP.
- REST structures data in XML, YAML, or any other format that is machine-readable, but usually JSON is most widely used.

RESTful Web Services - Types

- There are four main types of APIs:
- **Open APIs:** Also known as Public APIs, there are no restrictions to access these types of APIs because they are publicly available.
- **Internal APIs:** Also known as Private APIs, only internal systems expose this type of API, which is, therefore, less known and often meant to be used inside the company. The company uses this type of API among the different internal teams to be able to improve its products and services.
- **Partner APIs:** One needs specific rights or licenses in order to access this type of APIs because they are not available to the public.
- **Composite APIs:** This type of API combines different data and service APIs.
- It is a sequence of tasks that run synchronously as a result of the execution and not at the request of a task.
- Its main uses are to speed up the process of execution and improve the performance of the listeners in the web interfaces.
- Examples of REST APIs:
  - Instagram API permits your applications to retrieve user tags, photos, account and much more.
  - Twitter also provides a REST API which a developer can query to source the latest tweets, or provide a search query that will return the results in JSON format.
  - GitHub also offers super REST API that you can utilize to perform actions such as following GitHub issues, tracking user activity, and create repositories from your app.

RESTful Web Services - principles

- The six principles of REST (five required, and one optional) are:
- **Client-server architecture:** REST APIs separate the client (the one requesting information) from the server (the one possessing information). The client does not have to worry about how the server stores and retrieves data.
- **Uniform interface:** Whether the client is a software application, a browser, a mobile app, or something else entirely, it can access and use the REST API in the same way.
- **Statelessness:** The server does not have to remember the client's state. All the client's requests must be "stateless," so each request must include all necessary information (such as the client's authentication details).
- **Cacheability:** REST servers can cache data and reuse it for other requests in the future.
- **Layered system:** REST APIs may have multiple intermediary layers between the client and the server. However, the client does not have to know these implementation details.
- **Code on demand (optional):** Clients may download code (e.g. Java applets or JavaScript scripts) to access additional functionality at runtime.

RESTful Web Services

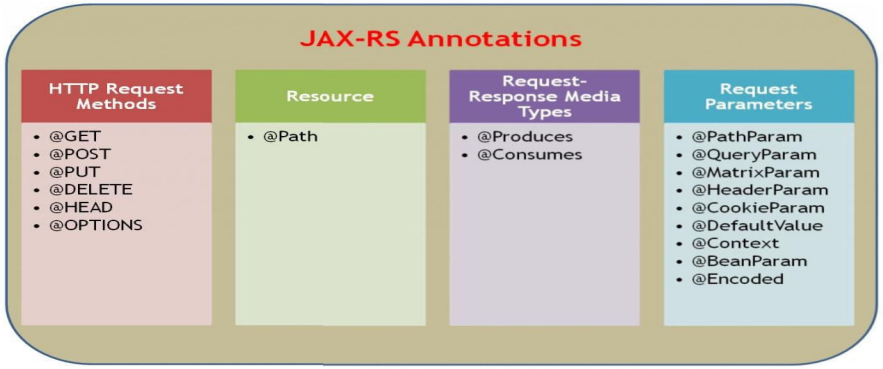
- The acronym REST, stands for REpresentational State Transfer.
- REST is an architectural style, meaning each unique URL represents an individual object of some sort.
- A REST web service uses HTTP and supports several HTTP methods: GET, POST, PUT or DELETE.
- It also offers simple CRUD-oriented services.
- So, as you can see REST relies on a simple URL to make a request, instead of using XML.
- But in some situations, you must provide additional information, but most web services using REST rely exclusively on using the URL approach.
- Let's see an example using basic HTTP requests.
- In this case we going to use "Swagger Pet Store API" from Swagger.io (https://swagger.io/).
- Sending a GET request to /pet/{petId} would retrieve pets with a specified ID from the database.
- Sending a POST request to /pet/{petId}/uploadImage would add a new image of the pet.
- Sending a PUT request to /pet/{petId} would update the attributes of an existing pet, identified by a specified id.
- Sending a DELETE request to /pet/{petId} would delete a specified pet.

BITS Pilani WILP

innovate achieve lead

BITS Pilani
Pilani | Dubai | Goa | Hyderabad | Mumbai
WORK INTEGRATED LEARNING PROGRAMMES

JAX – RS Annotations



JAX – RS Examples

- **RESTful JAX-RS File Download Example**
- We can download text files, image files, pdf files, excel files in java by JAX-RS API.
- To do so we need to write few lines of code only.
- Here, we are using jersey implementation for developing JAX-RS file download examples.
- You need to specify different content type to download different files. The @Produces annotation is used to specify the type of file content.

- **@Produces("text/plain"): for downloading text file.**
- **@Produces("image/png"): for downloading png image file.**
- **@Produces("application/pdf"): for downloading PDF file.**
- **@Produces("application/vnd.ms-excel"): for downloading excel file.**
- **@Produces("application/msword"): for downloading ms word file.**

JAX – RS Examples

File: FileDownloadService.java

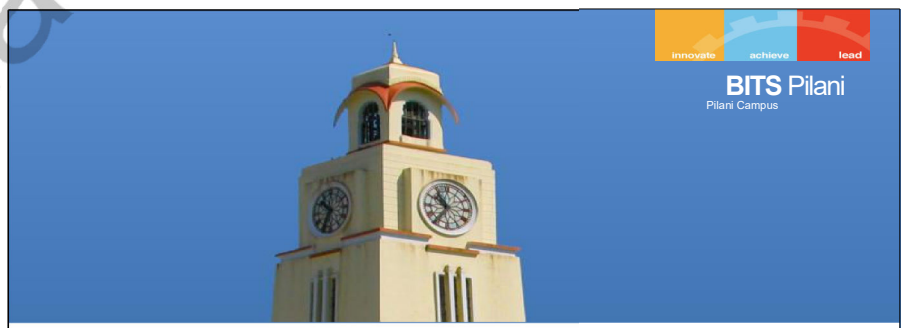
```
package com.javatpoint.rest;
import java.io.File;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.Response;
import javax.ws.rs.core.Response.ResponseBuilder;
@Path("/files")
public class FileDownloadService {
    private static final String FILE_PATH = "c:\\myfile.txt";
    @GET
    @Path("/txt")
    @Produces("text/plain")
    public Response getFile() {
        File file = new File(FILE_PATH);
        ResponseBuilder response = Response.ok((Object) file);
response.header("Content-
Disposition","attachment; filename=\"bits_file.txt\"");
        return response.build();
      }
}
```

File: index.html

```
<a href="rest/files/txt">Download Text File</a>
```

File: web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi=http://www.w3.org/2001/XMLSchema-
instance xmlns=http://java.sun.com/xml/ns/javaee
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://j
ava.sun.com/xml/ns/javaee/web-
app_3_0.xsd" id="WebApp_ID" version="3.0">
<servlet>
<servlet-name>Jersey REST Service</servlet-name>
<servlet-class>
org.glassfish.jersey.servlet.ServletContainer</servlet-class>
<init-param>
<param-name>jersey.config.server.provider.packages</param-
name>
<param-value>com.javatpoint.rest</param-value>
</init-param>
<load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
<servlet-name>Jersey REST Service</servlet-name>
<url-pattern>/rest/*</url-pattern>
</servlet-mapping>
</web-app>
```

CS 12: Middleware Technologies –Services of Cloud and Cloud Providers

innovate achieve lead

BITS Pilani WILP

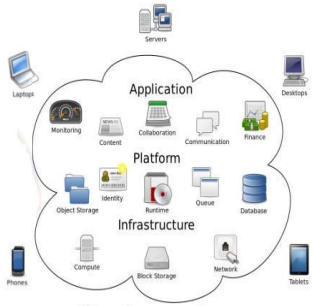
BITS Pilani
Pilani | Dubai | Goa | Hyderabad | Mumbai
WORK INTEGRATED LEARNING PROGRAMMES

Agenda

- Cloud Computing Overview
- AWS Cloud Components
- Launching EC2 for Linux / Windows
- AWS Lambda
- Hybrid Clouds
- Load Balancing

Middleware for Cloud Applications - AWS

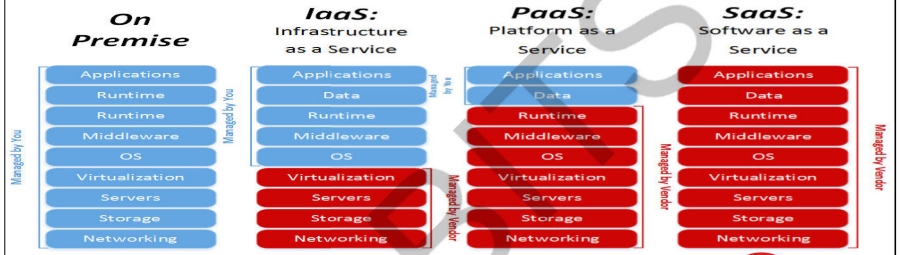
- Cloud Computing Overview
- Cloud computing is the on-demand availability of computer system resources, especially data storage and computing power, without direct active management by the user
- Cloud computing may be available:
  - Exclusively to one organization (enterprise / private cloud)
  - Multiple organizations (public cloud)
  - Combination (hybrid)
- Cloud computing relies on sharing of resources to achieve coherence and economies of scale
- Major cloud providers:
- Amazon – AWS
- Microsoft – Azure
- Google – Google Cloud



Cloud computing

What is Cloud Computing ?

- A number of characteristics define cloud data, applications services and infrastructure:
  - **Remotely hosted:** Services or data are hosted on remote infrastructure.
  - **Ubiquitous:** Services or data are available from anywhere.
  - **Commodified:** The result is a utility computing model similar to traditional that of traditional utilities, like gas and electricity - you pay for what you would want!



Amazon Web Services

- Amazon Web Services Cloud
  - Provides highly reliable and scalable infrastructure for deploying web-scale solutions
  - With minimal support and administration costs
  - More flexibility than own infrastructure, either on premise or at a data center facility
- AWS – Cloud Offerings
  - Amazon Simple Storage Service (S3), which provides a highly reliable and highly available object store over HTTP
  - Amazon SimpleDB, a key-value store and
  - Amazon Relational Database Service (RDS), which provides a MySQL instance in the cloud.
  - Amazon Elastic Compute Cloud (EC2), which is Amazon's IaaS offering

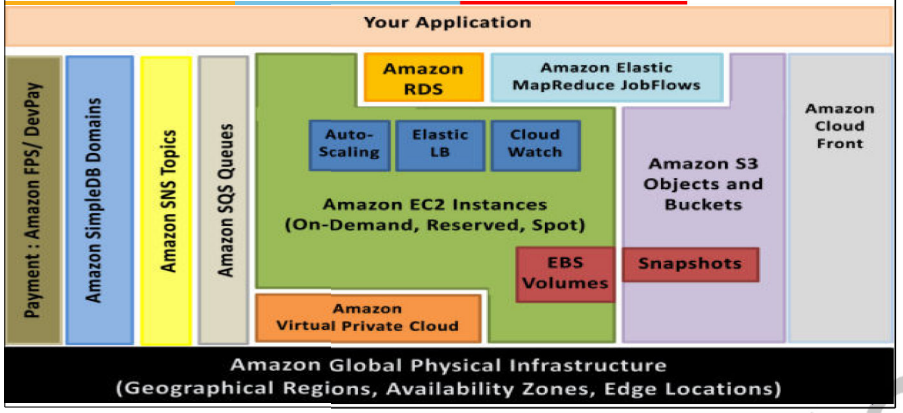
BITS Pilani WILP

innovate achieve lead

BITS Pilani
Pilani | Dubai | Goa | Hyderabad | Mumbai
WORK INTEGRATED LEARNING PROGRAMMES

innovate achieve lead

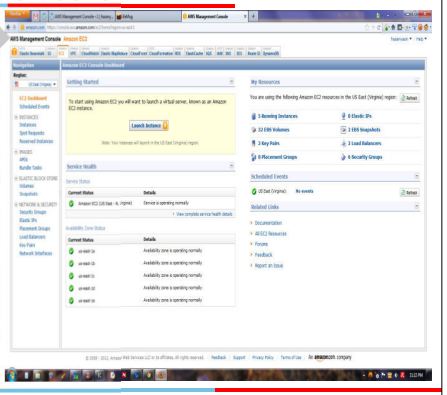
AWS Infrastructure Services



innovate achieve lead

Accessing EC2 Using AWS Console

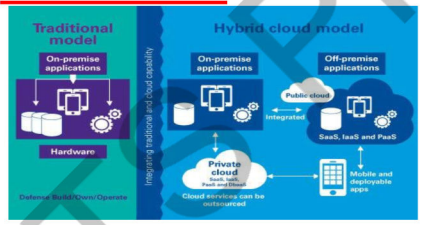
- Launching the instance gives a public DNS name that the user can use to login remotely and use as if the cloud server was on the same network as the client machine.
- Get Windows Password" button on the AWS Instance page.
- The console returns the administrator password that can be used to connect to the instance using a
- Remote Desktop application (usually available at Start-> All Programs -> Accessories -> Remote Desktop Connection).
- A description of how to use the AWS EC2 Console to request the computational, storage and networking resources needed to set up and launch a web server is described in the Simple EC2 example:
- Setting up a Web Server



innovate achieve lead

Hybrid Cloud & Multi Cloud Architectures

- **Hybrid Cloud:**
- Hybrid cloud has part of an enterprise's computing is provided by a separate cloud provider and part is done in house
- **Multi cloud** setup has an enterprise utilizing services of multiple cloud service providers in tandem (optionally with their on-prem traditional infra or private cloud)
- Advantages
  - Cost effective
  - Independence from a single provider
  - Flexibility through choice
- Disadvantages
  - Complexity of managing affairs
  - Security is a challenge
  - Redundancy and High availability across providers



innovate achieve lead

Middleware for Cloud Applications -AWS

- **AWS Lambda**
  - AWS Lambda is an event-driven, serverless computing platform provided by Amazon as a part
  - of AWS
  - The purpose of Lambda, as compared to AWS EC2, is to simplify building smaller, on-demand
  - applications that are responsive to events and new information
  - AWS Lambda supports securely running native Linux executables

BITS Pilani WILP

innovate achieve lead

BITS Pilani
Pilani | Dubai | Goa | Hyderabad | Mumbai
WORK INTEGRATED LEARNING PROGRAMMES

CS 13: Comparison of Services provided by Various Cloud providers

Agenda

- Key Advantage of Cloud
- Various cloud provider
- Case study

Key Advantage of Cloud

- Whether it's infrastructure, software, applications, services, products, or even an operating system, everything is making its way to the cloud.
  - Cloud computing allows a business to cut their operational and fixed monthly costs of hardware, databases, servers, software licenses. Eventually, it will reduce the need for IT resources, including people. All hardware, database servers, web servers, software, products, and services are hosted in the cloud and added to an account as needed.
  - Cloud computing offers 24/7 uptime (99.99% uptime). Cloud servers and data centers are managed by the cloud service provided. Therefore, there is no need for employee management.
  - Cloud computing is scalable and reliable. There is no limit to the number of users or resources. Furthermore, the cloud increases processing and resources as needed. If you do not need resources, you can always scale down.
  - Cloud computing provides maintainability and automatic updates of new software, OS, databases, and third-party software. It also reduces IT labor cost for a business.
  - Cloud service providers have data centers in various locations, which makes them faster and more reliable. Larger companies such as Microsoft and AWS even have data centers around the world.

Categories of Cloud Computing

- Software as a Service (SaaS)
  - SaaS is a software developed and hosted by someone else. Businesses or individuals are able to use them as needed.
- Platform as a Service (PaaS)
  - The PaaS component of cloud computing offers a full development and deployment environment in the cloud, including dev, test, QA, debugging, and deployment tools and services.
- Infrastructure as a Service (IaaS)
  - IaaS offers entire IT computing infrastructure, provisioned and managed over the internet.
  - The key components of IaaS are used to replace existing development and test environments, virtual machines, website hosting, storage, backup, networking, servers, operating systems, middleware, data, and applications, and high-performance computing (HPC).

BITS Pilani WILP

innovate achieve lead

BITS Pilani
Pilani | Dubai | Goa | Hyderabad | Mumbai
WORK INTEGRATED LEARNING PROGRAMMES

Top Cloud Service Provider

- Amazon Web Services (AWS)
- Microsoft Azure
- Google Cloud
- Alibaba Cloud
- IBM Cloud
- Oracle
- Hp
- AT&T
- Salesforce
- Rackspace

Case Study – Cloud Computing in a Chosen Public Sector Subject

- In order to illustrate the utilisation of cloud computing in the public sector, an in-depth interview was conducted with a secondary school's IT administrator.
- Profile of the School
- The chosen subject is a secondary school. Its official name is Střední škola a mateřska škola, o.p.s. Litoměřice. It consists of a kindergarten for 40 pre-school children and a secondary school, where students can study at a grammar school (an academically oriented secondary school) or several vocational education courses including those for beauticians, hairdressers, cabinetmakers, shopfitters, graphical designers. There is also an academy of the third age and a university of the third age. The school uses three buildings and employs more than 30 people. The information for the case study was provided by the school's (organization's) IT administrator of the school, Ing. Karel Klatovský.

Reasons for Implementing Cloud Technology

- The main motivations were cost and human resources savings in relation to streamlined administration.
- As the organization uses three separate buildings several kilometers from one another, it is necessary to connect them in a simple way.
- Previous IT solution did not solve difficult data synchronization among the three buildings.
- The organization considered whether to implement a stand-alone cloud solution to be able to implement such a solution only for data storage and data backup. Nevertheless, it was also necessary to address e-mail accounts, document sharing and register system.
- The organization used all these services before, but they did not want to administer them on their own anymore.

Case Study – Airbnb

- Airbnb is a community marketplace that allows property owners and travelers to connect with each other for the purpose of renting unique vacation spaces around the world. The Airbnb community users' activities are conducted on the company's Website and through its iPhone and Android applications. The San Francisco-based Airbnb began operation in 2008 and currently has hundreds of employees across the globe supporting property rentals in nearly 25,000 cities in 192 countries.
- According to Nathan Blecharczyk, Co-founder & CTO of Airbnb, due to service administration challenges experience with Airbnb original provider, Airbnb migrated to cloud computing functions of AWS (Amazon Web Services) after a year of its creation to gain the ease of managing and customizing its stack. From this, the company has continued to grow upon the reliance on AWS cloud. Thus, AWS is the easy answer for any internet business that wants to scale to the next level.

BITS Pilani WILP

innovate achieve lead

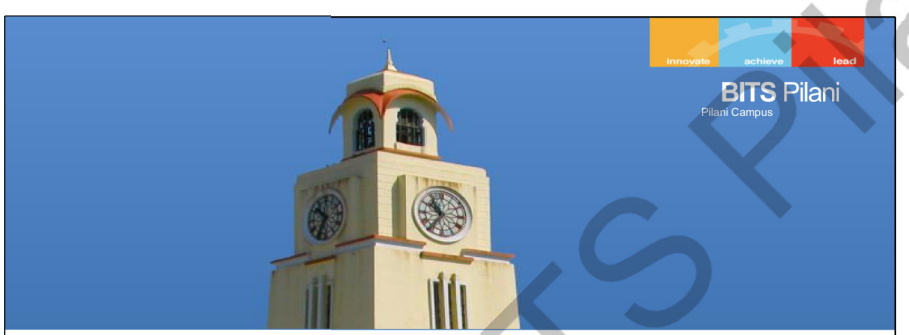
BITS Pilani
Pilani | Dubai | Goa | Hyderabad | Mumbai
WORK INTEGRATED LEARNING PROGRAMMES

Challenges experienced by Airbnb before migrating to AWS cloud.

- Small 5-person operations team
- Infrastructure scalability problem
- Huge traffic load during peak period like; festivals, public holidays etc.

Case Study – in Business: Cloud Strategies

- **Problem / Motivation**
- Identify special causes that relate to bad outcomes for the quality-related parameters of the products and visually inspected defects
- Complex upstream process conditions and dependencies making the problem difficult to solve using traditional statistical / analytical methods
- Determine the optimal process settings that can increase the yield and reduce defects through predictive quality assurancePotential savings huge as the cost of rework and rejects are very high
- **Solution**
- Use ontology to model the complex manufacturing processes and utilize semantic technologies to provide key insights into how outcomes and causes are related
- Develop a rich internet application that allows the user to evaluate process outcomes and conditions at a high level and drill down to specific areas of interest to address performance issues



CS 14: P2P Overlay Middleware

Agenda

- Introduction
- Napster and its legacy
- Peer-to-peer middleware
- Routing overlays
- Overlay case studies
- Application case studies

BITS Pilani WILP

innovate achieve lead

BITS Pilani
Pilani | Dubai | Goa | Hyderabad | Mumbai
WORK INTEGRATED LEARNING PROGRAMMES

Introduction - Traditional client/server systems

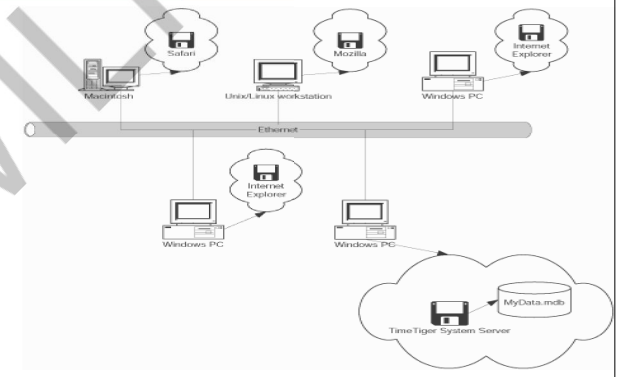
innovate achieve lead

- Manage and provide access to resource on a single server or cluster of tightly-couples servers.
- Centralized design.
- Limitations
  - Server hardware resource
  - Network connectivity
  - Requires administration
- **What is P2P system ?**
- P2P systems aim to support useful distributed services and applications using data and computing resources available in the personal computers and workstations that are present on the Internet and other networks in ever-increasing numbers.
- The main goal:
- To allow users to share data without putting them on central servers.

Architecture and Characteristics of P2P System

innovate achieve lead

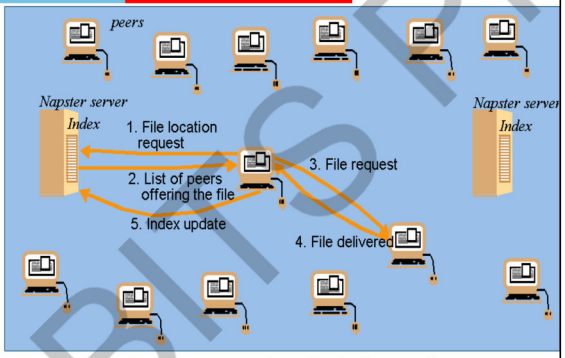
- Their design to ensure that each user contributes resource to the system.
- All nodes have the same functional capabilities and responsibilities.
- Nodes are autonomous.
- Nodes collaborate directly with each other.



Napster and its legacy - Introduction

innovate achieve lead

- A very popular for music exchange soon after its launch in 1999.
- At its peak:
  - Several million users were registered
  - Thousands users were swapping music files simultaneously.



Architecture and method of operation

Napster and its legacy – Lesson Learnt

innovate achieve lead

- The feasibility of building a useful large-scale service which depends almost wholly on data and computers owned by ordinary internet users.
- The service to scale to meet the needs of large numbers of users.
- Limitations
  - Object discovery and addressing is likely to become a bottle-neck.
  - No guarantees are required concerning the availability of individual files.

BITS Pilani WILP

innovate achieve lead

BITS Pilani
Pilani | Dubai | Goa | Hyderabad | Mumbai
WORK INTEGRATED LEARNING PROGRAMMES

P2P Middleware

- Design to meet the need for automatic placement and subsequent location of the distributed objects managed by peer-to-peer system and application.
- Functional Requirements
  - Simplify the construction of services that are implemented across many hosts in a widely distributed network.
  - Add new resources and remove them.
  - Add hosts to service and remove them.
  - Offer a simple programming interface that is independent of the type of distributed resource that application manipulates.

P2P Middleware

- Non-Functional Requirements
  - Global scalability: support application access millions of object on very large numbers of hosts.
  - Load balancing: Exploit a large number of computers depends upon the balanced distributed of workload across them
  - Optimization for local interactions between neighboring peers: The middleware should aim to place resources close to the nodes that access them the most.
  - Accommodating to highly dynamic host availability: hosts are free to join or leave the system at any time.
  - Security of data in an environment with heterogeneous trust: using authentication and encryption mechanisms to ensure the integrity and privacy of information.

Routing Overlay

- Routing overlay is a distributed algorithm takes responsibility for locating nodes and objects.
- The middleware takes the form of a layer that is responsibility for routing request from any client to the host that the object to which the request is addressed.
- The routing overlay ensures that any node can access any object by routing request through a sequence of node.
- Every nodes and objects has a identifier call GUID.
- The GUID is generated by a hash function using the object's value.
- Overlay routing systems are sometimes described as distributed hash tables (DHT) and this is reflected by the simplest form of API used to access them.

Overlay Case Study - Pastry

- Pastry is a routing overlay
- All the nodes and objects are assigned 128-bit GUIDs.
- In a network with N nodes, the Pastry routing algorithm will correctly route a message addressed to any GUID in O(logN) steps.
  - If a node is currently active, the message is delivered to that node
  - Otherwise, the message is delivered to the active node whose GUID is numerically closest to it.
- Active nodes take responsibility for processing requests addressed to all objects in their numerical neighborhood.
- Routing steps involve the use of an underlying transport protocol (normally UDP) to transfer the message to a Pastry node that is 'closer' to its destination.
- The real transport of a message across the Internet between two Pastry nodes may require a substantial number of IP hops.

BITS Pilani WILP

innovate achieve lead

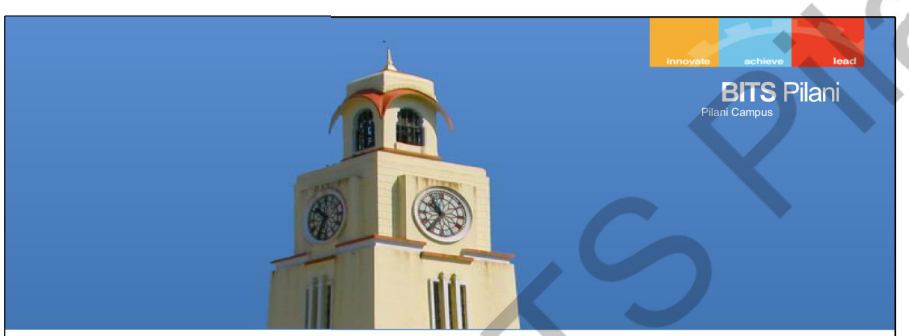
BITS Pilani
Pilani | Dubai | Goa | Hyderabad | Mumbai
WORK INTEGRATED LEARNING PROGRAMMES

Overlay Case Study - Tapestry

- A Tapestry is a P2P overlay network provides:
  - high-performance
  - scalable
  - fault-tolerant
  - location-independent routing
- Tapestry use adaptive algorithms with soft state to maintain fault tolerance.
- Tapestry nodes are assigned nodeIDs uniformly at random from a large identifier space
- More than one node may be hosted by one physical host.
- Application-specific endpoints are assigned globally unique identifiers (GUIDs) selected from the same identifier space
- Tapestry currently uses an identifier space of 160-bit values.

Application Case Study – Squirrel Web Cache

- Web browsers generate HTTP GET requests for Internet objects like HTML pages, images, …
  - a browser cache on the client machine,
  - a proxy web cache: a service running on another computer in the same local network or on a nearby node in the Internet
  - the origin web server : the server whose domain name is included in the parameters of the GET request.
- Some objects are uncacheable
- When a browser cache or proxy web cache receives a GET request:
  - the requested object is uncacheable
  - there is a cache miss
  - the object is found in the cache.



CS 15: NoSQL and Caching

Agenda

- NoSQL Databases – Introduction
- RDBMS Vs NoSQL DB
- MongoDB – Overview Introduction

BITS Pilani WILP

innovate achieve lead

BITS Pilani
Pilani | Dubai | Goa | Hyderabad | Mumbai
WORK INTEGRATED LEARNING PROGRAMMES

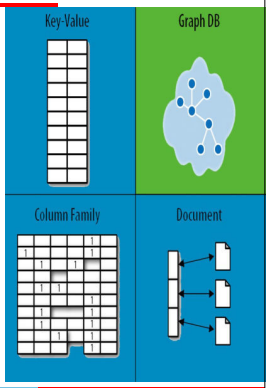
Specialized Middleware - NoSQL Databases

- RDBMS vs NoSQL

	SQL	NoSQL
Type	Relational	Non-Relational
Data	Structured Data stored in Tables	Un-structured stored in JSON files but the graph database does supports relationship
Schema	Static	Dynamic
Scalability	Vertical	Horizantal
Language	Structured Query Language	Un-structured Query Language
Joins	Helpful to design complex queries	No joins, Don't have the powerful interface to prepare complex query
OLTP	Recommended and best suited for OLTP systems	Less likely to be considered for OLTP system
Support	Great support	community depedent, they are expanding the support model
Integrated Caching	Supports In-line memory(SQL2014 and SQL 2016)	Supports integrated caching
flexible	rigid schema bound to relationship	Non-rigid schema and flexible
Transaction	ACID	CAP theorem
Auto elasticity	Requires downtime in most cases	Automatic, No outage required

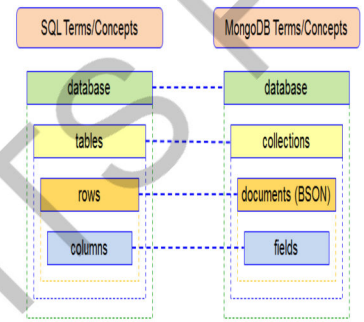
NoSQL – Database types

- Discussing NoSQL databases is complicated because there are a variety of types:
- **Sorted ordered Column Store**
  - Optimized for queries over large datasets, and store columns of data together, instead of rows
- **Document databases:**
  - pair each key with a complex data structure known as a document.
- **Key-Value Store :**
  - are the simplest NoSQL databases. Every single item in the database is stored as an attribute name (or 'key'), together with its value.
- **Graph Databases :**
  - are used to store information about networks of data, such as social connections.

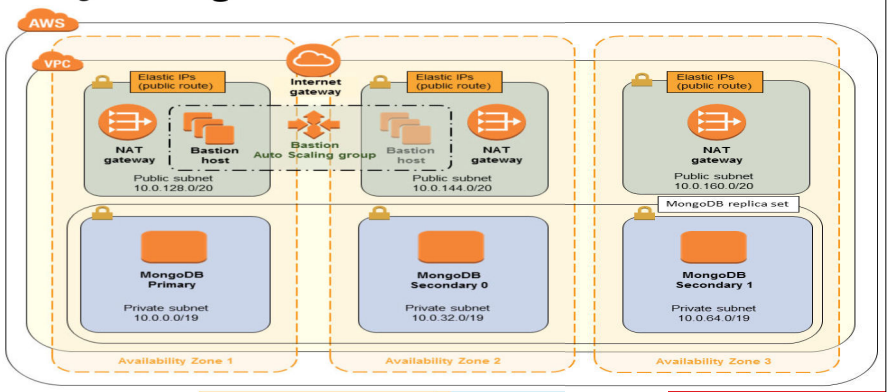


NoSQL – Mongo DB

- MongoDB is a document oriented DB following NoSQL paradigm
- Released in 2009, latest version 4.0
- Written in C++, Go, Python and Javascript
- Offers a JS shell for all operations
- Available as a standalone server and Cloud offering
- Features
  - Adhoc Querying
  - Indexes
  - Replication
  - Speed and durability
  - Scaling



NoSQL : Mongo DB – AWS



BITS Pilani WILP

innovate achieve lead

NoSQL – Mongo DB

- **Performance:**
- There is no perfect NoSQL database
- Every database has its advantages and disadvantages
- Depending on the type of tasks (and preferences) to accomplish
- NoSQL is a set of concepts, ideas, technologies, and software dealing with
- Big data
- Sparse un/semi-structured data
- High horizontal scalability
- Massive parallel processing
- Different applications, goals, targets, approaches need different NoSQL solutions

Specialized Middleware - Caching

- Cache is a high-speed data storage layer which stores a subset of data, typically transient in nature, so that future requests for that data are served up faster
- Distributed Cache an extension of the traditional concept of cache, spanning multiple servers
- Examples – Redis, Memcached, Hazelcast, Spark, Couchbase, Apache Ignite
- Use cases
  - Database Caching
  - Content Delivery Networks
  - Session Management
  - APIs
  - Web caching

Thank You

BITS Pilani WILP

BITS Pilani
Pilani | Dubai | Goa | Hyderabad | Mumbai
WORK INTEGRATED LEARNING PROGRAMMES

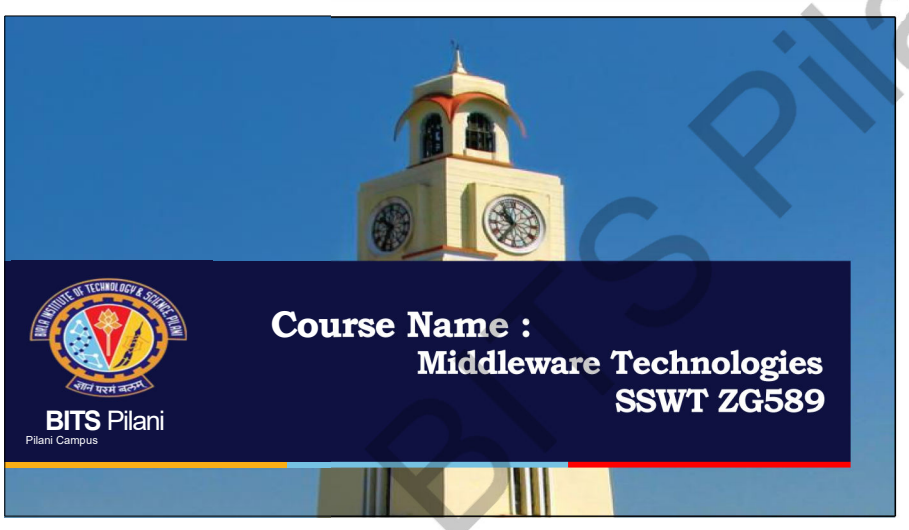
innovate achieve lead

Start Recording

innovate achieve lead

IMP Note to Students

- It is important to know that just login to the session does not guarantee the attendance.
- Once you join the session, continue till the end to consider you as present in the class.
- IMPORTANTLY, you need to make the class more interactive by responding to Professors queries in the session.
- **Whenever Professor calls your number / name ,you need to respond, otherwise it will be considered as ABSENT**



innovate achieve lead

Modular Structure

No	Title of the Module
M1	Introduction and Evolution
M2	Enterprise Middleware
M3	Middleware Design and Patterns
M4	Middleware for Web-based Application and Cloud-based Applications
M5	Specialized Middleware

BITS Pilani WILP

innovate achieve lead

BITS Pilani
Pilani | Dubai | Goa | Hyderabad | Mumbai
WORK INTEGRATED LEARNING PROGRAMMES

CS 9 : Web Services and Service-Oriented Architectures

Agenda

- Web Services Architectures
- Service-Oriented Architectures
- Architectures Styles, Relationship and Implementation

Introduction

- Web service is an appropriate medium to propagate communication between the client and server applications on the World Wide Web. It is a software module which is designed to perform a certain set of tasks as follows:
  - The web services can be searched for over the network and can also be invoked accordingly.
  - When invoked the web service would be able to provide the functionality to the client which invokes that web service.
- A way to expose the functionality of an information system and making it available through standard web technologies.
- "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" [W3C]

Need of Web Services

- Modern day business applications use variety of programming platforms to develop web-based applications. Some applications may be developed in Java, others in .Net, while some other in Angular JS, Node.js, etc.
- Most often than not, these heterogeneous applications need some sort of communication to happen between them. Since they are built using different development languages, it becomes really difficult to ensure accurate communication between applications.
- Here is where web services come in. Web services provide a common platform that allows multiple applications built on various programming languages to have the ability to communicate with each other.

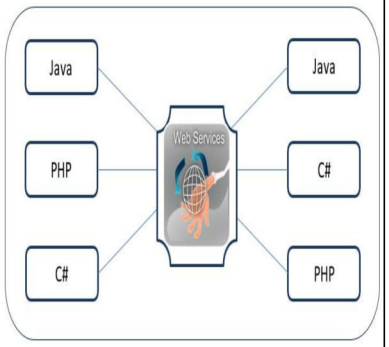
BITS Pilani WILP

innovate achieve lead

BITS Pilani
Pilani | Dubai | Goa | Hyderabad | Mumbai
WORK INTEGRATED LEARNING PROGRAMMES

Web Services

- Any service that :
  - Is available over the Internet or private (intranet) networks
  - Uses a standardized XML messaging system
  - Is not tied to any one operating system or programming language
  - Is self-describing via a common XML grammar
  - Is discoverable via a simple find mechanism



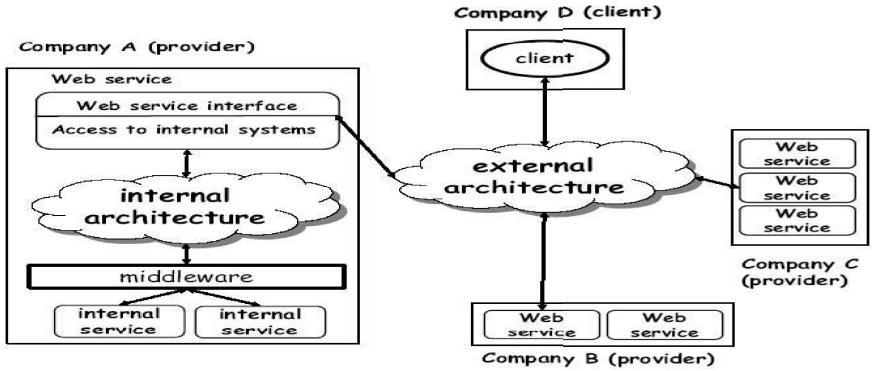
Benefits of Web Services

- **Exposing Business Functionality on the network** – A web service is a managed code unit that provides some sort of functionality to client applications or end-users. This functionality can be invoked over the HTTP protocol which means that it can also be invoked over the internet. Web services can be used anywhere on the internet and provide the necessary functionality as required.
- **Interoperability amongst applications** – Web services allow various applications to communicate with each other and share data and services among themselves.
- **A standardized Protocol which everybody understands** – Web services use standardized industry protocol for the communication. All four layers (Service Transport, XML Messaging, Service Description, and Service Discovery layers) uses well-defined protocols in the web services protocol stack.
- **Reduction in cost of communication** – Web services make use of SOAP over HTTP protocol, which lets you use your existing low-cost internet for implementing web services.

Web Service Architecture

- Web service architecture can be viewed using the either of the way:
  - The first is to examine the individual roles of each Web service actor.
  - The second is to examine the emerging web service protocol stack.
- Internal architecture
  - Web services expose internal operations to be invoked through the web
  - Receive requests through the web
  - Pass the requests to the underlying IT system
- External architecture
  - A middleware architecture which integrates different web services

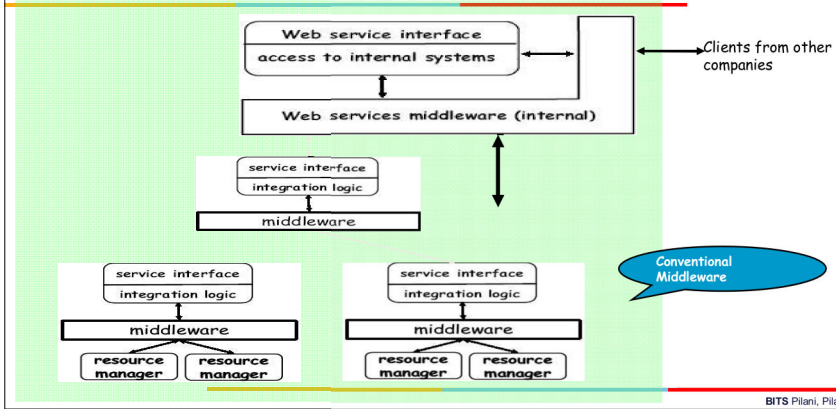
Web service architecture comprises Internal and external architecture



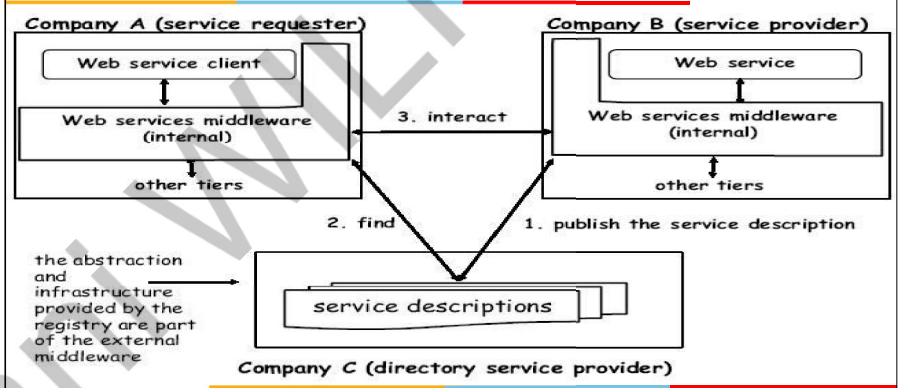
BITS Pilani WILP

innovate achieve lead

Basic architecture of a Web service implemented tiered architecture

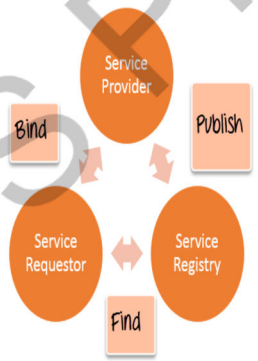


External architecture of Web services



Web Service Roles

- **Service Provider:** This is the provider of the web service. The service provider implements the service and makes it available on the Internet.
- **Service Requestor:** This is any consumer of the web service. The requestor utilizes an existing web service by opening a network connection and sending an XML request.
- **Service Registry:** This is a logically centralized directory of services. The registry provides a central place where developers can publish new services or find existing ones. It therefore serves as a centralized clearing house for companies and their services.

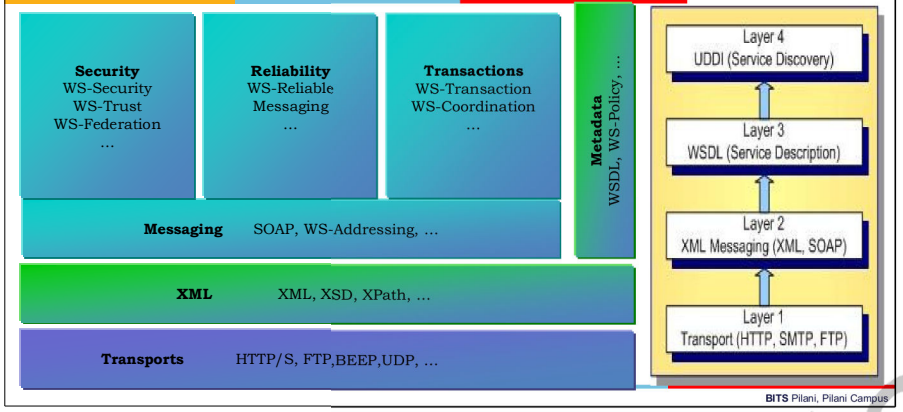


Web Service Tasks

- **Publish** - A provider informs the broker (service registry) about the existence of the web service by using the broker's publish interface to make the service accessible to clients
- **Find** - The requestor consults the broker to locate a published web service
- **Bind/Interact** - With the information it gained from the broker(service registry) about the web service, the requestor is able to bind, or invoke, the web service.

BITS Pilani WILP

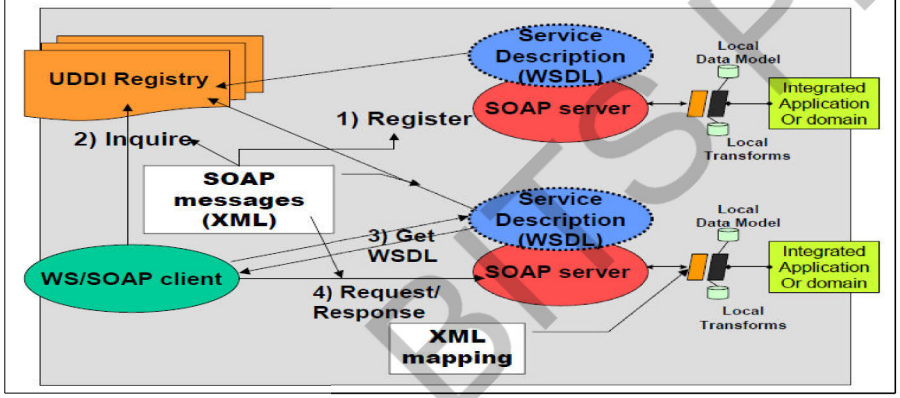
Web Service Protocol Stack



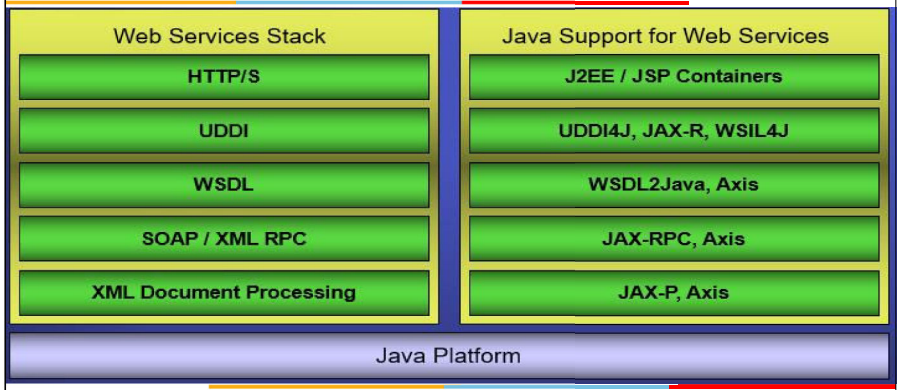
Web Services Protocol Stack Layer

- **Service Transport:** This layer is responsible for transporting messages between applications. Currently, this layer includes Hyper Text Transport Protocol (HTTP), Simple Mail Transfer Protocol (SMTP), File Transfer Protocol (FTP), and newer protocols such as Blocks Extensible Exchange Protocol (BEEP).
- **XML Messaging:** This layer is responsible for encoding messages in a common XML format so that messages can be understood at either end. Currently, this layer includes XML-RPC and SOAP.
- **Service Description:** This layer is responsible for describing the public interface to a specific web service. Currently, service description is handled via the Web Service Description Language (WSDL).
- **Service Discovery:** This layer is responsible for centralizing services into a common registry and providing easy publish/find functionality. Currently, service discovery is handled via Universal Description, Discovery, and Integration (UDDI).

Web Services In Action



Web Services Stack - Java



BITS Pilani WILP

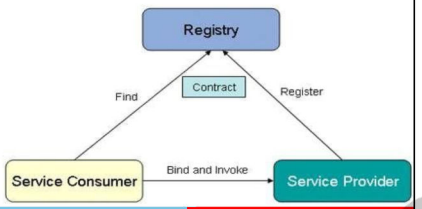
innovate achieve lead

BITS Pilani
Pilani | Dubai | Goa | Hyderabad | Mumbai
WORK INTEGRATED LEARNING PROGRAMMES

Web service components and Technologies

- **SOAP** – "Simple Object Access Protocol"
  - A simple way to send documents (some people have called it "email for documents")
  - How to format XML documents for transmission
- **WSDL** - "Web Services Description Language"
  - Defines all details about a service
- **UDDI** - "Universal Description, Discovery and Integration"
  - One way to advertise and discover services

- **Web Service implementation:**
  - **Build and Publish**
  - **Find**
  - **Bind and Invoke**



Web Services Implementation

- **Pre Conditions**
  - No one will be able to use your service
- **Step 1 :Build and Publish**
  - Build
    - Create your application
    - Create your contract file (WSDL)
  - Publish
    - Register your application as a web service onto any registry
    - This process happens on UDDI using a separate SOAP request
    - This process is useful only if your web service should be accessible using Internet
- **Post conditions**
  - Your web service will be available to the Public (If published to any registry) or will be accessible to intranet users

Web Services Implementation

- **Pre Conditions**
  - The user needs to access a service but is not aware of the service details
- **Step 2 :Find**
  - Find
    - Search in the registry for a service which provides your needs
    - Obtain the necessary details about the service
- **Post conditions**
  - The user will have all the details about the service and gets ready to contact the service

- **Web Services Samples**
  - Eclipse_tutorial_link: http://www.eclipse.org/webtools/jst/components/ws/1.5/tutorials/index.html
  - Bottom to Top approach
    - JAVA to WSDL
  - Top to bottom approach
    - WSDL to JAVA

Web Services Implementation

- **Pre Conditions**
  - The user will have the contract necessary to identify and call the service
- **Step 3 :Bind**
  - Bind
    - Use the contract file to build the request message.
  - Invoke
    - Send a request to the service and request for the necessary operation available from the service.
    - The request should be sent in the protocol which is required by the service
- **Post conditions**
  - The user will receive the response from the service in a format specified in the contract file.

BITS Pilani WILP

innovate achieve lead