

Assignment 6

Roll no: 43144

Name: Ruchika Pande

Title: Any distributed application using the Messaging System in Publisher-Subscriber paradigm.

Problem Statement:

To develop a distributed application using the Messaging System in Publisher-Subscriber paradigm.

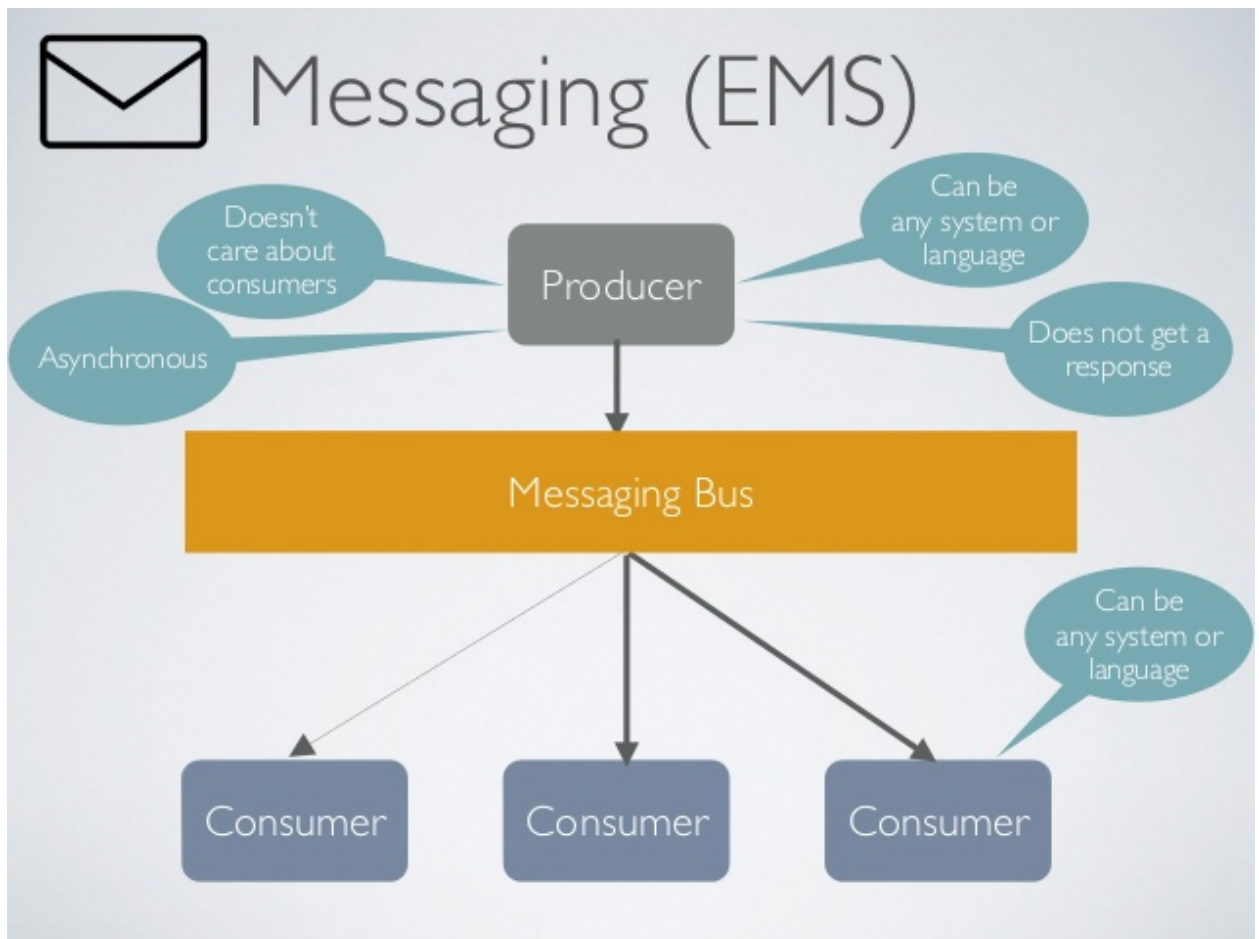
Objectives:

- To study about Publisher-Subscriber paradigm

Theory:

Enterprise messaging system :

Enterprise messaging system (EMS) is a messaging system allowing software applications and systems to communicate semantically. The semantics can be applied by sending precise messages to and fro throughout the enterprise. The messages are asynchronous data (messages not sent or processed in real time, meaning not like a chat room or telephone conversation) sent by one application or system to another application or system and stored in the queue of the receiving program until processed. The system is not dependent on a particular operating system or programming language.



Publish Subscribe paradigm:

Publish/Subscribe is a software design pattern that describes the flow of messages between applications, devices, or services in terms of a publisher-to-subscriber relationship.

Pub/Sub, as it is often called, works like this: a publisher (i.e. any source of data) pushes messages out to interested subscribers (i.e. receivers of data) via live-feed data streams known as channels (or topics). All subscribers to a specific publisher channel are immediately notified when new messages have been published on that channel, and the message data (or payload) is received together with the notification.

As a general messaging pattern that can be implemented across various programming languages and platforms, Publish/Subscribe has a few core principles that are universally useful. These are:

- Publishers do not need to know anything about their subscribers, and subscribers only have to know the name of the topic channel they are subscribed to. Publishers simply define what data goes in which channel and transmit the

channel data once only from one-to-many, so that it can be easily shared out amongst other apps for their own uses.

- Any publisher may also be a subscriber and data streams can be multiplexed, enabling the creation of interlinked systems that mesh together in an elegant, distributed, and internally-consistent manner.
- Since intercommunication is event-driven and based on notifications, there is no need for client-to-server polling techniques.
- Message data types can be anything from strings to complex objects representing text, sensor data, audio, video, or other digital content.

Publish/ Subscribe Pattern



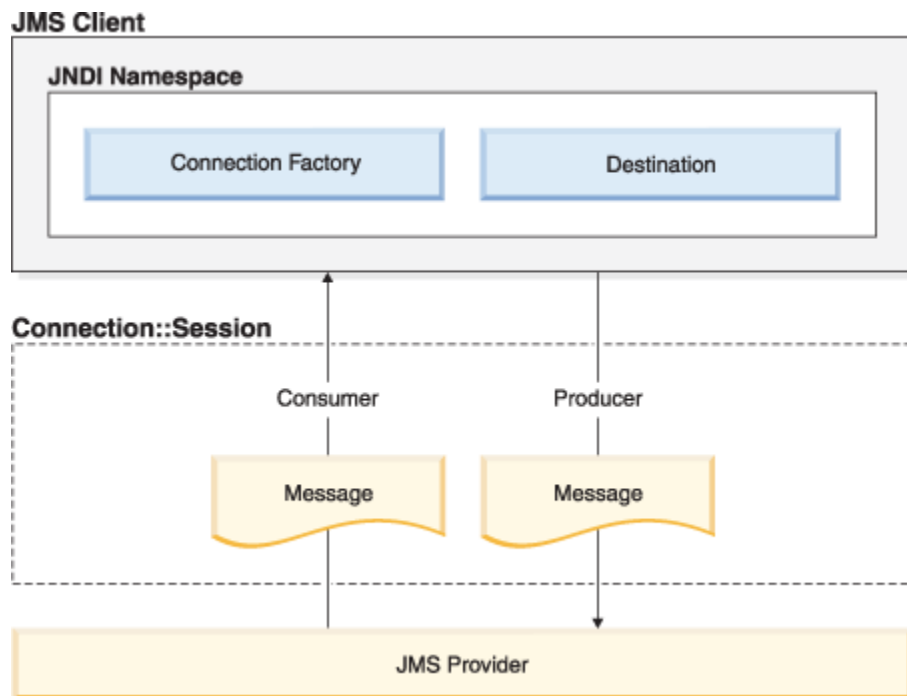
RealTimeEngine.io

JAVA Message Service:

JMS defines an API for accessing services from a messaging provider; that is, it is not a product itself. It is a set of interface classes that messaging provider vendors implement. Applications that use the JMS API can then communicate with the messaging provider the vendor supplies. The JMS API has become the industry standard interface for Java™ applications to create, send, receive and read messages using messaging providers that support the JMS API. The standard is associated with the Java Platform, Enterprise Edition (Java EE) platform. Java EE is a set of standards for a component-based way to develop, assemble and deploy enterprise applications. Java EE containers implement a standard runtime environment that provides quality of service items such as security, transaction support and thread pooling.

In the following diagram, the *JMS provider* implements the JMS API. It is the entity with which the *JMS client* interacts. The JMS client establishes a connection and a session through which the interaction takes place. The JMS client establishes the connection based on configuration information in the *connection factory* and identifies where a message is to be sent to or retrieved from based on the *destination*. Both the

connection factory and destination objects are listed in the Java Naming and Directory Interface (JNDI) namespace. JNDI is the Java industry standard API for accessing naming and directory services. Since the connection factory and destination are administered objects by JNDI, it means the JMS client can connect to different JMS providers without changing JMS client code; that is, it creates portability. It also means that attributes which often can and do change dynamically such as the destination can be changed independent of the JMS client code.



JMS has two messaging domains: point-to-point and publish-subscribe, which are the common ways of distributing messages. Note that JMS lets an application send or receive messages in either domain. The following table shows the interfaces used by these domains.

Implementing the solution:

1. To execute the pub-sub programs, you need the message queue environment.

The Java Message Service (JMS) API is a Java Message Oriented Middleware (MOM) API for sending messages between two or more clients. It is a Java API that allows applications to create, send, receive, and read messages. The JMS API enables communication that is loosely coupled, asynchronous and reliable.

To use JMS, we need to have a JMS provider that can manage the sessions, queues, and topics. Some examples of known JMS providers are Apache ActiveMQ,

WebSphere MQ from IBM or SonicMQ from Aurea Software. Starting from Java EE version 1.4, a JMS provider has to be contained in all Java EE application servers.

Here we are implementing the JMS concepts and illustrate them with a JMS Hello World example using ActiveMQ.

Interfaces extending core JMS interfaces for Topic help build publish-subscribe components.

2. The Publisher.java program to publish messages to the Publish-Subscribe topic. The code for which is shown in the below section.

3. While the preceding program helps publish messages to the Publish-Subscribe Topic, the Subscribe.java program is used to subscribe to the Publish-Subscribe Topic, which keeps receiving messages related to the Topic until the quit command is given.

Compilation and Executing the solution:

For Setting up an environment:

1. Download the 2 Jar files javax.jms.jar for JMS and apache-activemq-4.1.1.jar for Apache ActiveMQ. 2. Download Apache MQ and Install it using the Apache MQ Installation Link

Links for Download and installation instruction: a. Jms <http://www.java2s.com/Code/Jar/j/Downloadjavaxjmsjar.htm> [Jar file] b. Apache <http://www.java2s.com/Code/Jar/a/Downloadapacheactivemq411jar.htm> [Jar file] c. Download - <http://activemq.apache.org/activemq-5158-release.html>[ApacheMQ Download link] d. Install - <https://docs.wso2.com/display/BAM200/Installing+Apache+ActiveMQ+on+Linux> [Apache MQ Installation Instructions] e. Concept - <https://hackernoon.com/observer-vs-pub-sub-pattern-50d3b27f838c>

Steps to execute:

1. Create a Publisher.java file and copy paste the Publisher code 2. Create a Subscriber.java file and copy paste the Subscriber code 5. Add external jars a. Right Click on Project in eclipse package explorer b. Go to Build Path c. Select Configure Build Path d. Add external jars e. Select both the downloaded jars from the first step 6. Run activemq with the following command:

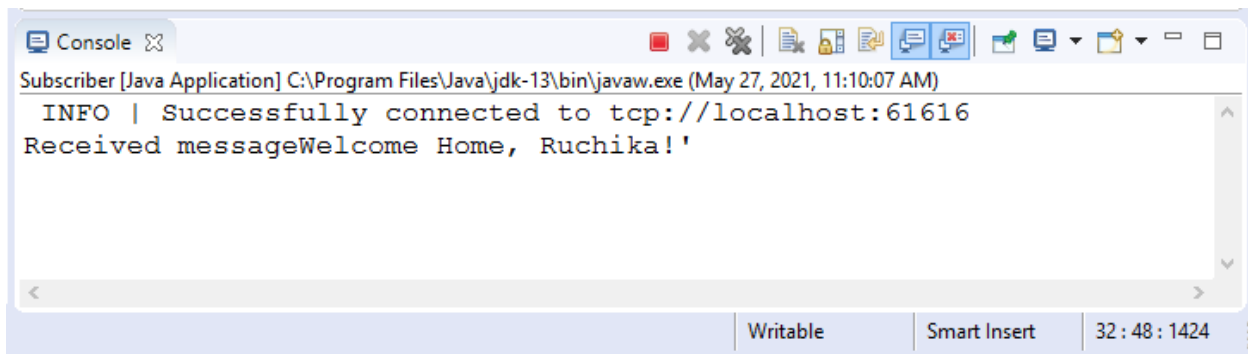
sudo sh active start

7. Run the publisher code and pin console for publisher 8. Run Subscriber

Output:



```
Subscriber [Java Application] C:\Program Files\Java\jdk-13\bin\javaw.exe (May 27, 2021, 11:05:59 AM)
INFO | Successfully connected to tcp://localhost:61616
```



```
Subscriber [Java Application] C:\Program Files\Java\jdk-13\bin\javaw.exe (May 27, 2021, 11:10:07 AM)
INFO | Successfully connected to tcp://localhost:61616
Received messageWelcome Home, Ruchika!'
```

Writable Smart Insert 32 : 48 : 1424

Conclusion:

In this assignment, we learned about the Publish-Subscribe model of Communication which is implemented using JMS and Apache ActiveMQ.