What is Open System Interconnection?

Open System Interconnection (OSI) is an ISO (International Organization for Standardization) standard which was developed in the 1970s to "homogenize" the way different networks - and therefore computer systems communicating through them - work together.

This standard constitutes of a framework (i.e. a base to develop on) created to implement communication protocols in seven successive layers:

1. **Physical Layer** - forms the physical (i.e. hardware) base for OSI to work.
2. **Data Link Layer** - transfers the data between network [nodes].
3. **Network Layer** - directs the traffic (i.e. forwarding) between places.
4. **Transport Layer** - works to ensure reliability, data flow (i.e. rate) control and its stream.
5. **Session Layer** - responsible of managing the session between applications.
6. **Presentation (Syntax) Layer** - working to shape and present the data to be processed.
7. **Application Layer** - setting and ensuring common grounds - reaching the applications - for communication. (This is where AMQP lives!)

What is Application Layer?

Application layer - which is where AMQP lives - is one of the above pieces forming the Open System Interconnection standard. If we were to elaborate, application layer can be thought of as the [only] one that users interact with and as the one which defines how process-to-process (or application to application) communications take place.

Some [common] examples to application layer - other than AMQP - would be:

- IRC
- DNS
- FTP
- IMAP
- SSH
- and more.

What are Communication Protocols?

Each communication protocol is made of rules and regulations clearly defined to form a shared language spoken between different application with the end result of being able to communicate regardless how they might be originally set to work.

These protocols have elements such as data formats, definition of parties using the protocol, routings and flow (rate) control.

What is Advanced Message Queuing Protocol?

The Advanced Message Queuing Protocol (AMQP) creates interoperability between clients and brokers (i.e. messaging middleware). Its goal of creation was to enable a wide range of different applications and systems to be able to work together, regardless of their internal designs, standardizing enterprise messaging on industrial scale.

AMQP includes the definitions for both the way networking takes place and the way message broker applications work. This means the specifications for:

- Operations of routing and storage of messages with the message brokers and set of rules to define how components involved work
- And a wire protocol to implement how communications between the clients and the brokers performing the above operations work

Reasons for AMQP's Creation and Use

Before AMQP, there used to be different message brokering and transferring applications created and set in use by different vendors. However, they had one big problem and it was their lack of interoperability. There was simply not a way for one to work with another. The only method that could be used to get different systems using different protocols to work was by introducing an additional layer for converting messages called *messaging bridge*. These systems, using individual adapters to be able to receive messages like regular clients, would be used to connect multiple and different messaging systems (e.g. WebSphere MQ and another).

AMQP, by offering the clearly defined rules and instructions as we explained above, creates a common ground which can be used for all message queuing and brokering applications to work and interoperate.

What are AMQP Use Cases?

Whenever there is a need for high-quality and safe delivery of messages between applications and processes, AMQP implementing message brokering solutions can be considered for use.

AMQP ensures

- Reliability of message deliveries
- Rapid and ensured delivery of messages
- Message acknowledgements

These capabilities make it ideal for

- Monitoring and globally sharing updates
- Connecting different systems to talks to each other
- Allowing servers to respond to immediate requests quickly and delegate time consuming tasks for later processing

- Distributing a message to multiple recipients for consumption
- Enabling offline clients to fetch data at a later time
- Introducing fully asynchronous functionality for systems
- Increasing reliability and uptime of application deployments

How Do AMQP Message Brokers Work?

In AMQP, "message brokers" translate to applications which receive the actual messages and route (i.e. transfer) them to relevant parties.

```
APPLICATION     EXCHANGE      TASK LIST      WORKER
   [DATA] -------> [DATA] ---> [D]+[D][D][D] --->  [DATA]
 Publisher      EXCHANGE        Queue       Consumer
```

How does AMQP Exchanges Work?

After receiving messages from publishers (i.e. clients), the exchanges process them and route them to one or more queues. The type of routing performed depend on the type of the exchange and there are currently four of them.

Direct Exchange

Direct exchange type involves the delivery of messages to queues based on routing keys. Routing keys can be considered as additional data defined to set where a message will go.

Typical use case for direct exchange is load balancing tasks in a round-robin way between workers.

Fanout Exchange

Fanout exchange completely ignores the routing key and sends any message to all the queues bound to it.

Use cases for fanout exchanges usually involve distribution of a message to multiple clients for purposes similar to notifications:

- Sharing of messages (e.g. chat servers) and updates (e.g. news)
- Application states (e.g. configurations)

Topic Exchange

Topic exchange is mainly used for pub/sub (publish-subscribe) patterns. Using this type of transferring, a routing key alongside binding of queues to exchanges are used to match and send messages.

Whenever a specialized involvement of a consumer is necessary (such as a single working set to perform a certain type of actions), topic exchange comes in handy to distribute messages accordingly based on keys and patterns.

Headers Exchange

Headers exchange constitutes of using additional headers (i.e. message attributes) coupled with messages instead of depending on routing keys for routing to queues.

Being able to use types of data other than strings (which are what routing keys are), headers exchange allow differing routing mechanism with more possibilities but similar to direct exchange through keys.

Exchange to Exchange Bindings

Java Client Example

Use the Channel#exchangeBind method. The following example binds an exchange "destination"to "source" with routing key "routingKey".

```
Channel ch = conn.createChannel();
ch.exchangeBind("destination", "source", "routingKey");
```

Blocked Connection Notifications

It is sometimes desirable for clients to receive a notification when their connection gets blocked due to the broker running low on resources (memory or disk).

Using Blocked Connection Notifications with Java Client

With the official Java client, blocked connection notifications are handled by BlockedListenerinterface implementations. They can be registered on a Connection using the Connection.addBlockedListener method:

```
ConnectionFactory factory = new ConnectionFactory();
Connection connection = factory.newConnection();
connection.addBlockedListener(new BlockedListener() {
   public void handleBlocked(String reason) throws IOException {
      // Connection is now blocked
   }

   public void handleUnblocked() throws IOException {
      // Connection is now unblocked
   }
});
```