Messaging notes.

General.

Synchronous communication.

Synchronous communication represents a *tight coupling* between *a sender and receiver* :

- the sender and receiver must both be available – i.e. real-time communication
- the sender will block until the response is available
- communication is one-to-one
- sender / receiver have some knowledge of each other – e.g. location, protocol, methods, etc.
- the sender is responsible for dealing with failures - e.g. retries, etc.

"request/response" is an example of a synchronous communication paradigm.

Example : a browser sends a request to a web server and "waits" for the response. The "wait" can be indefinite or until a timeout occurs.

Asynchronous communication.

Asynchronous communication represents a *loose coupling* between *senders / receivers* :

- sender and receiver communicate via a midde-man / broker / mediator
- no requirement for sender or receiver to both be available – i.e. deferred communication
- sender doesn't wait for a response (if any)
- communication can be one-to-one or one-to-many
- sender and receiver have no direct knowledge of each other so promotes reuse / extension – e.g. new senders / receivers can be added without impact; senders / receivers can be replaced as required
- the broker is responsible for dealing with failures – e.g. retries, etc.
- messages may be delayed and/or arrive out of sequence

"queues" and "publisher/subscriber" are examples of asynchronous communication paradigms.

Example : the sender mail client sends an email via a mail server. The sender's mail server attempts delivery to the recipient mail server and repeats if unsuccessful. The recipient mail client picks up the mail from the recipient server next time they're online.

Messaging.

Message Oriented Middleware (MOM) is available from several vendors.
The MOM facilitates a loose coupling between *producers* and *consumers* and ensures that message delivery is *reliable* (some include support for ACID properties, transactions, etc.)

Messaging products implement a combination of the *Mediator* and *Observer* patterns.
The MOM acts as a Mediator between *Colleagues*, which in this case are the *Subject(s)* and *Observer(s)*

Messaging is asynchronous in nature but can also operate synchronously.
Producers always operate asynchronously – i.e. complete after message production.
Consumers can operate either :

- synchronously (pull).
  Wait for message OR wait with timeout OR get next message if available. ***N.B. the MOM client talks to the MOM server component via a network connection. Polling the MOM server frequently could result in excessive network traffic and/or MOM server load.***

- asynchronously (push).
  Register a message listener with the MOM, MOM pushes messages as they arrive leaving the application free to do other things. ***N.B. not all clients can support push, e.g. applet behind firewall***

The MOM can be used to implement a request/response mechanism (either synchronously or asynchronously) but with the added benefits of scalability, reliability, maintainability, etc.

JMS.

Introduction.

JMS (Java Message Service) is an API from Sun which allows generic access to MOM services – in a similar way that JDBC allows generic access to databases.
As JMS allows access to existing MOM products, JMS can be used for legacy integration.

A JMS implementation is required in J2EE 1.3+ platforms.

Concepts.

- *Provider* – vendor implementation of the JMS API
- *Messaging domains* – two types of domain, *Point To Point (PTP)* or *publisher/subscriber*
- *Message consumption* – two types of consumption, *synchronous* and *asynchronous* (see below).
- *Administered objects* – JMS bootstrap objects (e.g. connection factories, destinations) bound in the JNDI namespace by an administrator
- *ConnectionFactory* – *Queue/TopicConnectionFactory*, used to create a connection to the Provider
- *Destination* – *Queue/Topic*, target of message production/consumption
- *Connection* – *Queue/TopicConnection*, encapsulates connection to the Provider; *start ()* indicates that message delivery should begin
- *Session* – *Queue/TopicSession*, used to create producers, consumers and message; also provides a context for transactions
- *MessageProducer* – *QueueSender/TopicPublisher*
- *MessageConsumer* – *QueueReceiver/TopicSubscriber*; consume messages synchronously using the *receive ()* methods or asynchronously by passing a *MessageListener* to *setMessageListener ()*
- *Message* – object produced or consumed; composed of header, properties and body. Body types are : text, name/value pairs, bytes, stream of primitives, serialized object or none.

PTP messaging.

- based on *Queue*s, *Sender*s and *Receiver*s
- Senders and Receivers address the Queue by name
- one or more Senders but only one Receiver
- messages stay in the Queue until consumed or past expiry time
- receiver acknowledges successful processing of a message

Publisher / subscriber.

- based on *Topic*s, *Publisher*s and *Subscriber*s
- Publishers and Subscribers address the Topic by name
- one or more Publishers publish to the Topic subscribed to by one or more Subscribers
- messages only stay in the Topic for as long as it takes to send them to the active Subscribers
- a subscriber can only consume "active" messages – i.e. those published since the subscriber became active (unless durable subscriptions are in use)

Message consumption.

JMS producers/consumers can be Java applications, EJBs, servlets, JSPs, applets (unlikely), etc.
Applications can consume messages synchronously or asynchronously.
EJBs, servlets and JSPs can consume messages synchronously (can't do asynchronously as request/response based).
EJB2.0 supports MessageDrivenBeans which consume messages asynchronously (with the help of the application server)

SCEA Messaging Objectives.

List benefits of synchronous and asynchronous communication* interpreted objective

Type	*Benefits*
Synchronous (e.g. RMI)	• *real-time response* • *lock-step* – order of execution guaranteed • *self-contained* - *i*t's a single-shot "transaction", no ongoing management over time so suitable in pooled environments like EJB
Asynchronous (e.g. JMS)	• *loose coupling* producers/consumers delegate responsibility of reliable message delivery to MOM • *deferred communication* – consumers can be offline, servers down, etc. • *promotes reuse* – components can be added/removed/modified without impact • *supports broadcast of messages*

Identify scenarios that are appropriate to implementation using messaging, EJB or both.

Type	*Scenario*
Messaging (asynch. comms)	• *distributed events with Q.o.S.* – e.g. news feeds, stock prices, user profiling, etc. • *workflow* – order processing, problem tracking, etc. A pipeline of tasks that follow on from one another; time taken to complete each task may differ (e.g. minutes, days); pipeline may be fully automated or may required human intervention; tasks may branch off into subtasks and rejoin main workflow at a later point • *"threads"* – messaging can be used to simulate threading (not allowed in EJB) • *integration* – use MOM to talk to legacy apps, apps in other technologies (e.g. C++)
EJB (synch. comms)	• *need an immediate response* – e.g. credit card check • *transactional / secure operations* – producer and consumer are separated so can't be subjected to the same transaction / security context • *simple access to business logic required* – e.g. User, Account, etc.
Both (combined)	• *logging* – applications produce log messages; EJB consumer stores them • *merging branches in workflow* Each branch stores it's result in an entity bean. When all branches have completed, use entity bean for the next stage in the workflow. • *large scale messaging applications* MessageDriveBean is managed by the EJB container; multiple MDBs can concurrently consume messages asynchronously so higher throughput with lower resource utilisation • *session facade as JMS producer* - hides details of JMS from user of session bean • *standalone JMS client consumer invoking session bean facade to multiple entity beans* Legacy app publishes message for order request; standalone client picks up request and invokes session facade; session facade manipulates multiple entity beans (all within a transaction)

List benefits of synchronous and asynchronous messaging.

Type	***Benefits***
Synchronous (client pull)	• *faster message acquisition* - asynchronous may buffer messages • *client control* - messages processed when client wants • *works with all MOM clients* - some clients don't support push
Asynchronous (MOM push)	• *efficiency* MOM server pushes messages to client rather than the "timed-poll" of synchronous. Timed-poll doesn't scale as it can lead to increased network traffic and/or MOM server utilisation. • *management* MOM automatically invokes register handler so the client is free to do other things

Identify scenarios that are more appropriate to using implementation using synchronous messaging, rather than asynchronous.

- *applications that don't support push* - e.g. applet behind a firewall, JSPs, etc.
- *applications that require fine control* - e.g. application wants to control message processing rate

Identify scenarios that are more appropriate to using implementation using asynchronous messaging, rather than synchronous.

- *large scale messaging applications* - "timed-poll" doesn't scale
- *standalone MOM clients* - application is free to do other things
- *multi-tier decoupled clusters* - provides "natural" load-balancing (multiple instances of "hungry" consumers; also can partition queues/topics by usage) and resilience (e.g. if the back end is down, the front end can continue to produce messages; when the back end comes back up, messages are waiting to be consumed)