JTA Y JMS

Diana Benavides

José Abásolo

TRANSACCIONES DISTRIBUIDAS

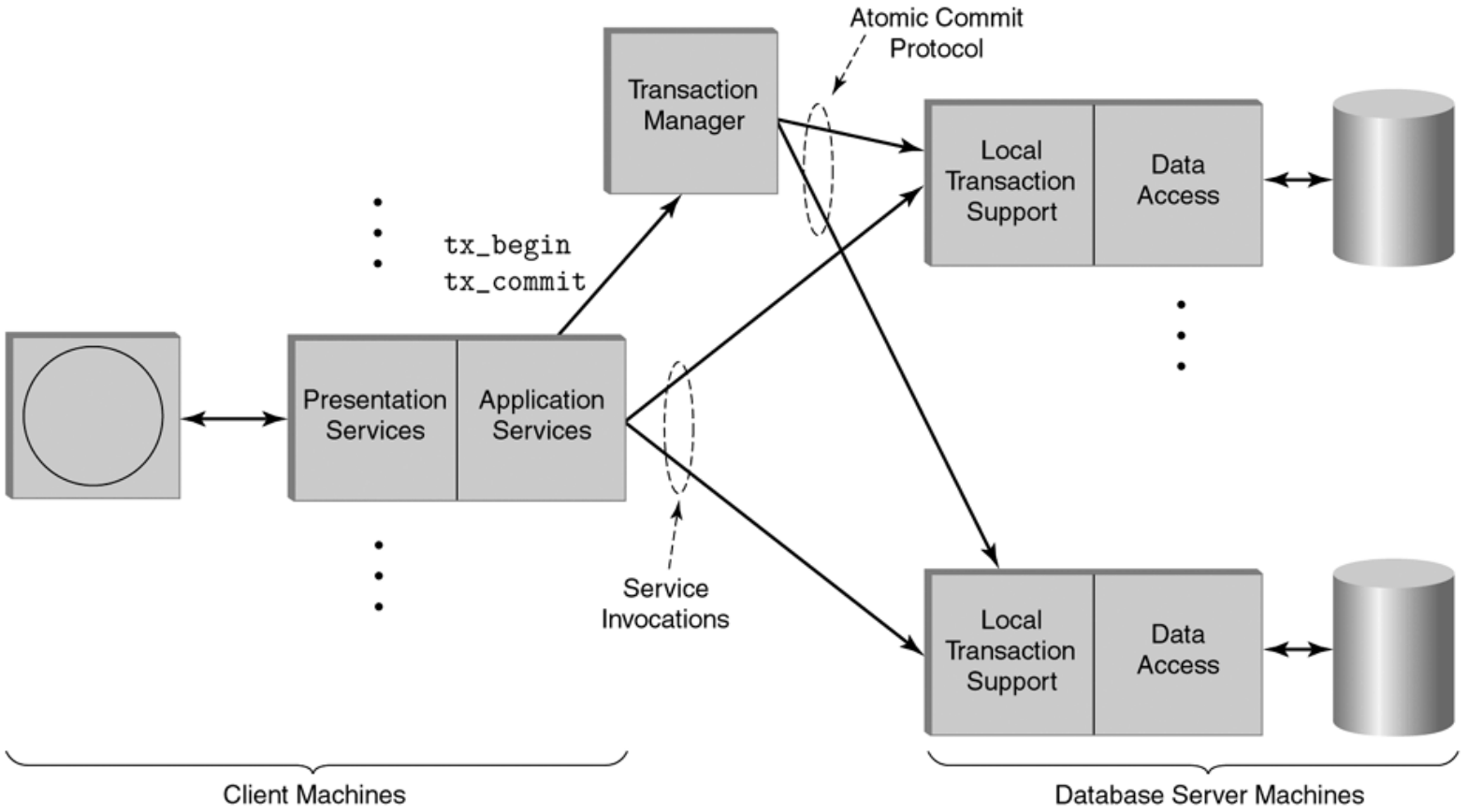


FIGURE 23.10 Two-tiered multidatabase transaction processing system in which a transaction can access several database servers.

TWO PHASE COMMIT

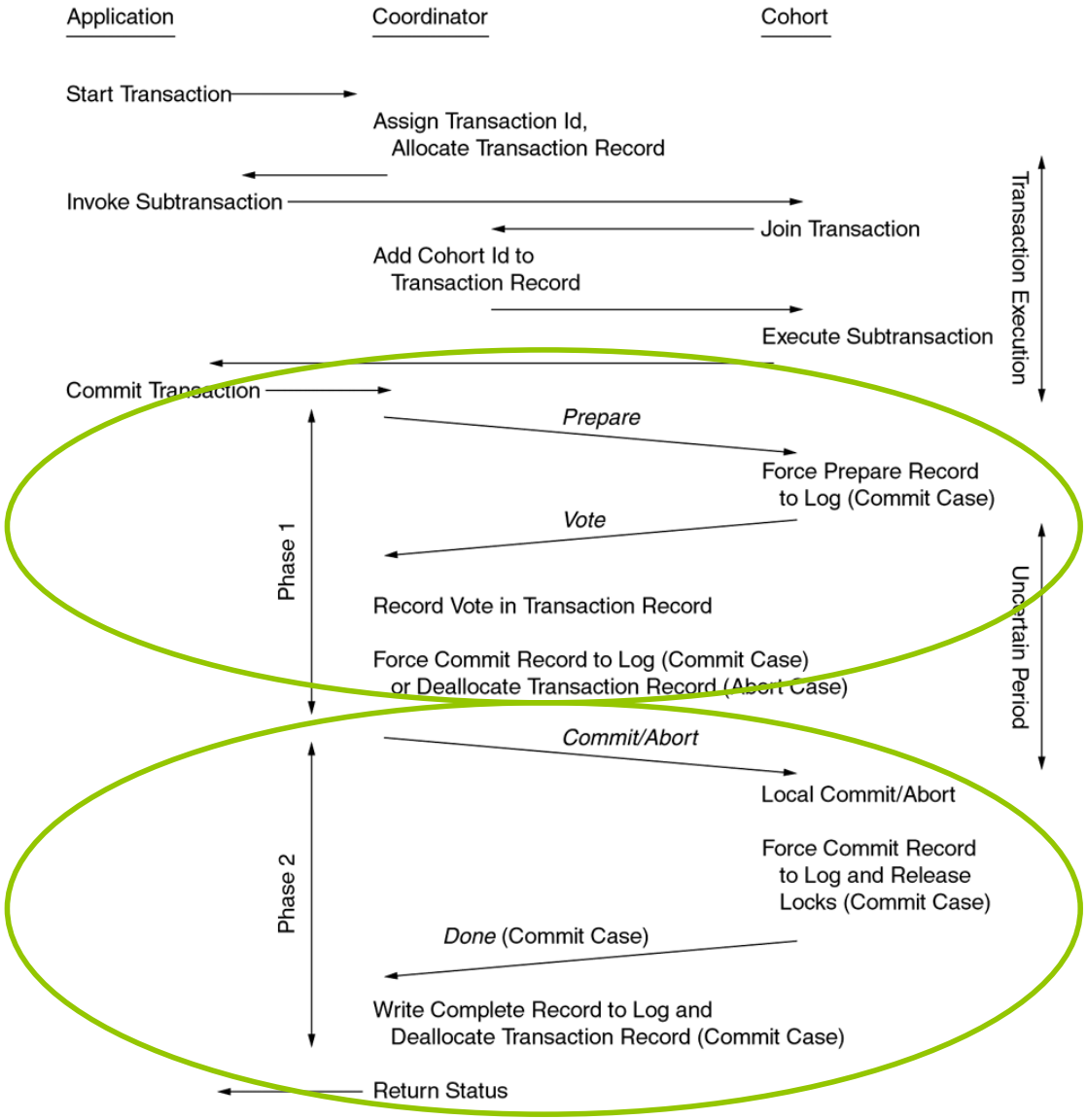


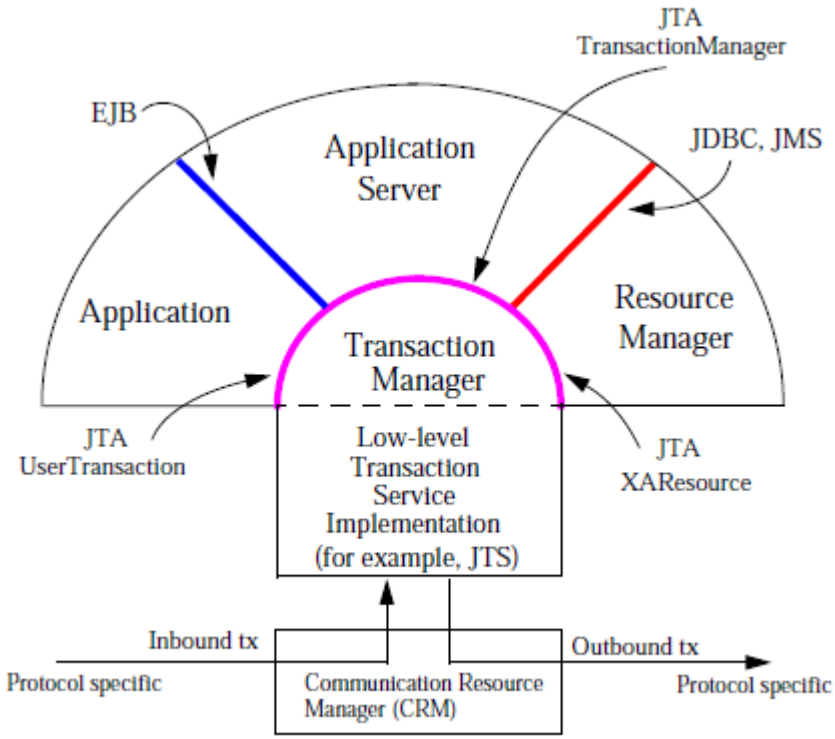
FIGURE 24.3 Exchange of messages in a two-phase commit protocol.

JTA

- JTA especifica interfaces locales de Java entre un **transaction manager (gestor de transacciones)** y las partes involucradas en un sistema de transacciones distribuidas.

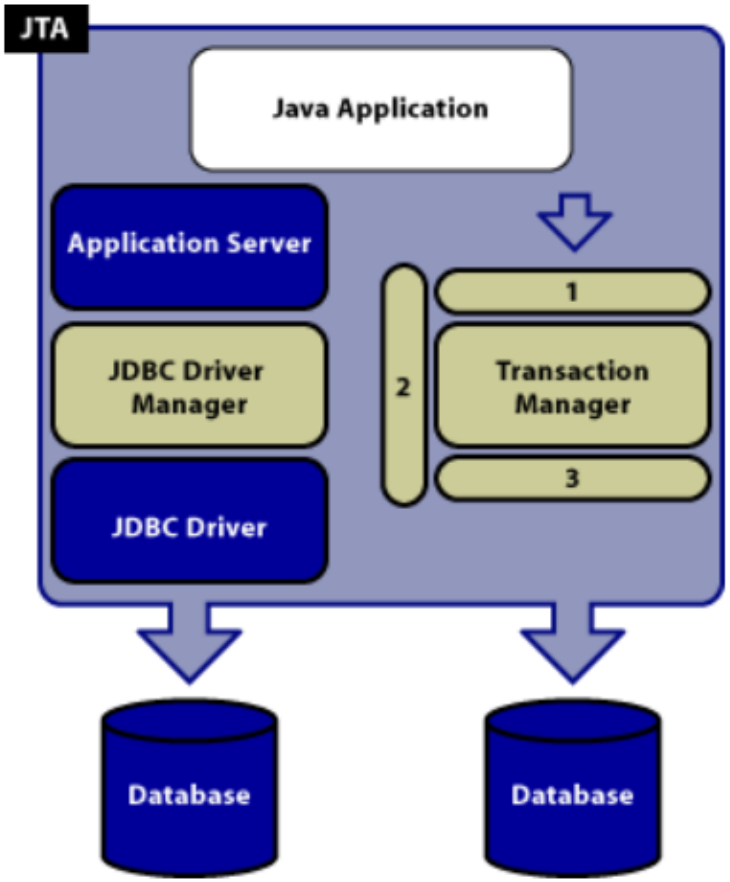
JTA: Arquitectura

- Partes involucradas:
  - Transaction manager
  - Servidor de aplicaciones (TP Monitor)
  - Gestor de recursos
  - Aplicación cliente
  - Gestor de comunicación de recursos (CRM)



JTA: API

- El API se apoya en tres partes:
  - Una interfaz de aplicación de alto nivel, que controla los límites de las transacciones → javax.transaction.UserTransaction.
  - Una interfaz de alto nivel para el transaction manager → javax.transaction.TransactionManager.
  - Una implementación Java del estándar X/Open que permite a un gestor de recursos participar en una transacción global controlada por un gestor de transacciones → javax.transaction.xa.XAResource.



JTA

- ¿Cómo funciona?

1. Una aplicación envía una solicitud de transacción al Transaction Manager.

2. El Transaction Manager crea una ramificación de la transacción, por cada gestor de recursos involucrado.

3. Cada ramificación debe ser confirmada/revertida por el gestor de recursos respectivo.

javax.transaction.UserTransaction

Método	Descripción
void begin()	Crear una nueva transacción y la asocia con el proceso actual.
void commit()	Completar la transacción asociada al proceso actual.
int getStatus()	Obtener el estado de la transacción asociada al proceso actual.
void rollback()	Deshacer la operación asociada con el proceso actual.
void setRollbackOnly()	Modificar la operación asociada con el proceso actual de tal manera que el único resultado posible de la transacción sea deshacer la transacción.
void setTransactionTimeout(int sec)	Modificar el valor de tiempo de espera que está asociado con las transacciones iniciadas por las siguientes invocaciones del método Begin.

javax.transaction.TransactionManager

Método	Descripción
Transaction getTransaction()	Obtener el objeto de transacción que representa el contexto de la transacción del proceso de llamada.
Transaction suspend()	Suspender la operación actualmente asociada al proceso de llamada y devolver un objeto de transacción que representa el contexto de la transacción que esta siendo suspendida.

javax.transaction.xa.XAResource

Método	Descripción
void commit(Xid xid, boolean onePhase)	Confirma la transacción global especificada por el XID.
void end(Xid xid, int flags)	Finaliza el trabajo realizado en nombre de una ramificación de la transacción.
void forget(Xid xid)	Le dice al gestor de recursos que se olvide de una rama de transacción completada heurísticamente
int getTransactionTimeout()	Obtiene el valor de tiempo de espera de la transacción actual establecido para esta instancia XAResource.
boolean isSameRM(XAResource xares)	Este método se llama para determinar si la instancia del administrador de recursos representada por el objeto de destino es igual a la instancia del administrador de recursos representada por el parámetro xares.
int prepare(Xid xid)	Se le dice al administrador de recursos que se prepare para una confirmación de la transacción especificada en XID.
Xid[] recover(int flag)	Obtiene una lista de las ramas de la transacción a partir de un administrador de recursos.
void rollback(Xid xid)	Informa al administrador de recursos para hacer retroceder el trabajo realizado en nombre de una rama de la transacción.
boolean setTransactionTimeout(int seconds)	Establece el valor de transacción de tiempo de espera actual de esta instancia XAResource.
void start(Xid xid, int flags)	Empezar a trabajar desde una rama de la transacción especificada en XID.

COLAS

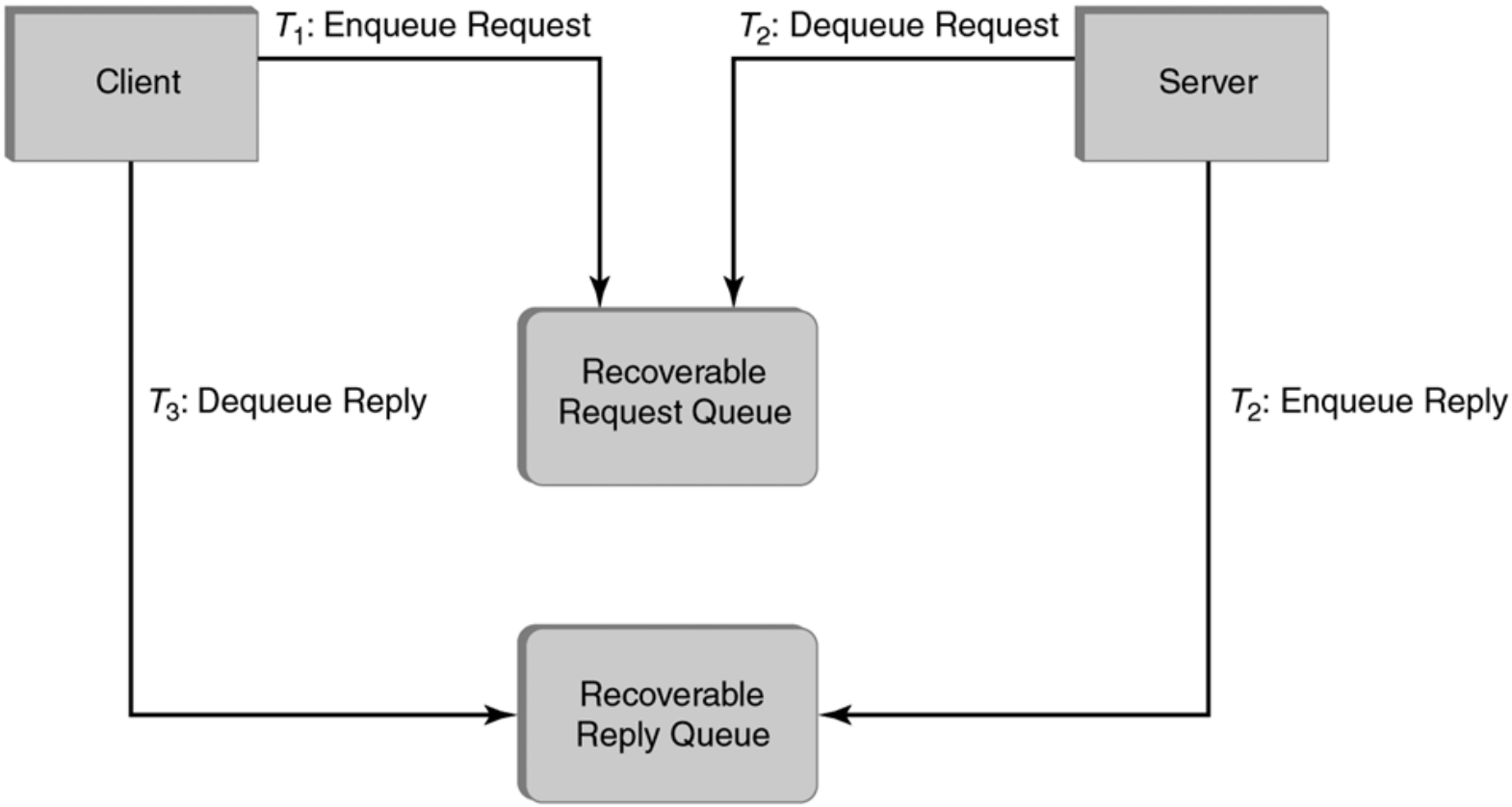


FIGURE 23.8 Queued transaction processing involves two queues and three transactions.

JMS

- Basado en el concepto de mensajería → hay al menos un emisor y un receptor.

- JMS es un API que permite a las aplicaciones crear, enviar, recibir y leer mensajes.

JMS: Arquitectura

- **Proveedor JMS**, sistema de mensajería que implementa las interfaces JMS, con funciones administrativas y de control.
- **Clientes JMS**, programas o componentes Java que envían o reciben mensajes.
- **Mensajes**.
- **Objetos administrados**.

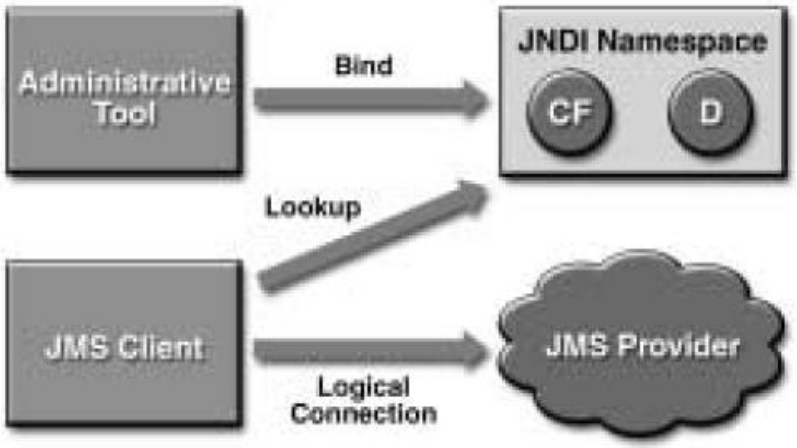


Figure 2.1 JMS API Architecture

JMS: API

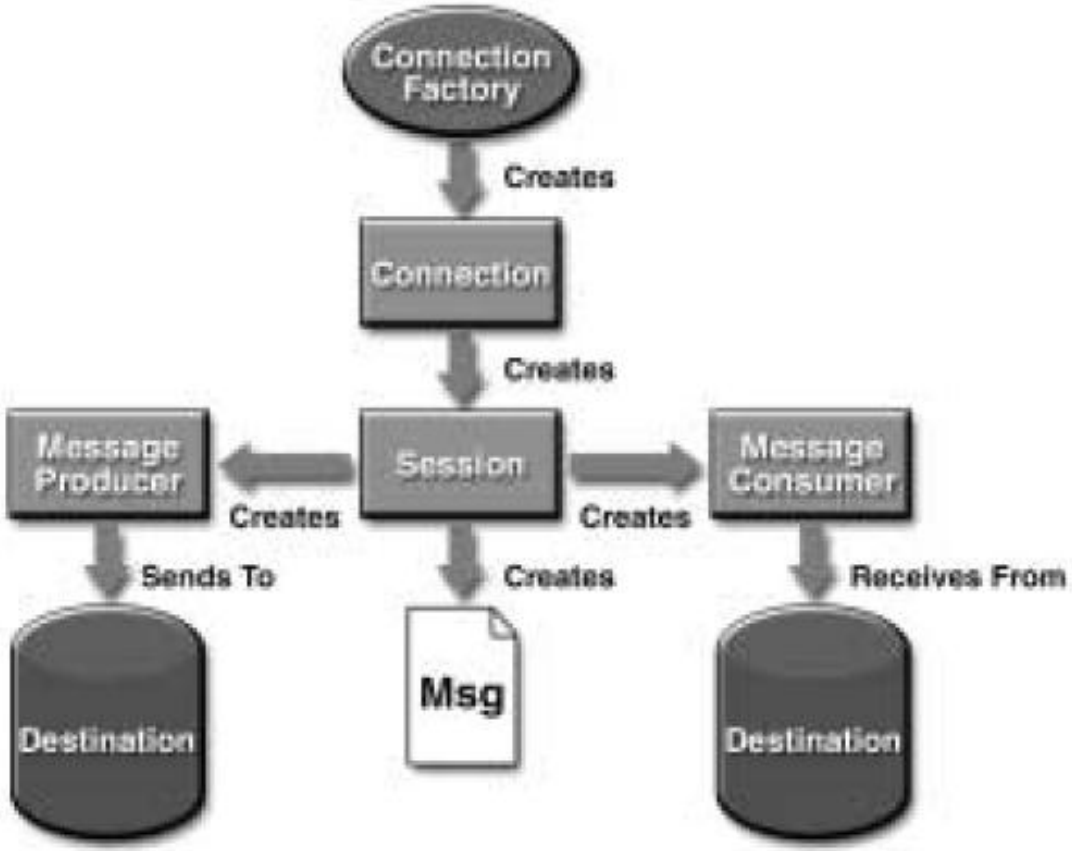


Figure 3.1 The JMS API Programming Model

JMS: Enfoque de manejo de mensajes

- **Punto a punto**.

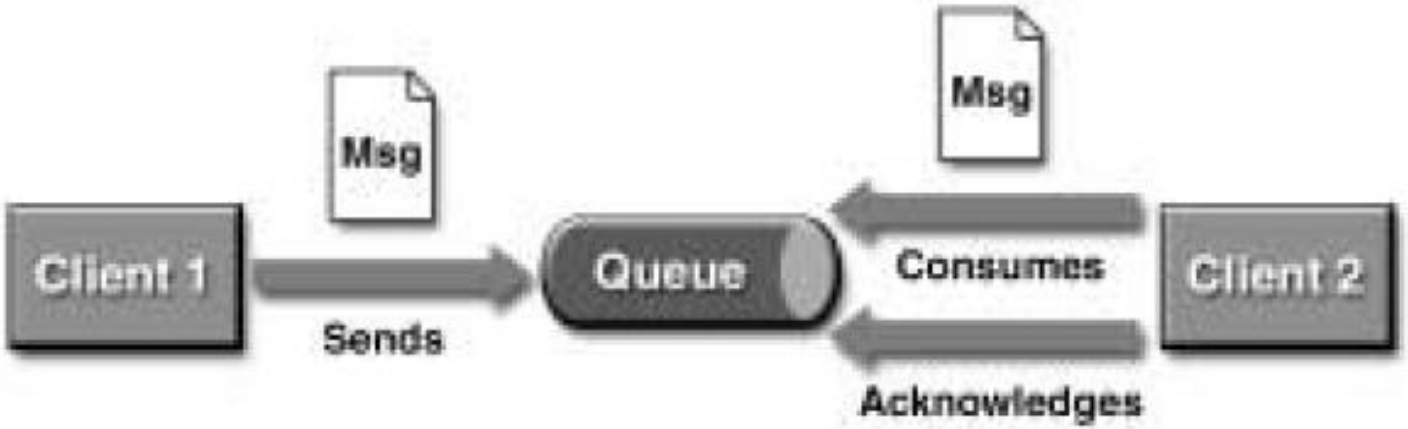


Figure 2.2 Point-to-Point Messaging

JMS: Enfoque de manejo de mensajes

- **Publicación/Suscripción**.

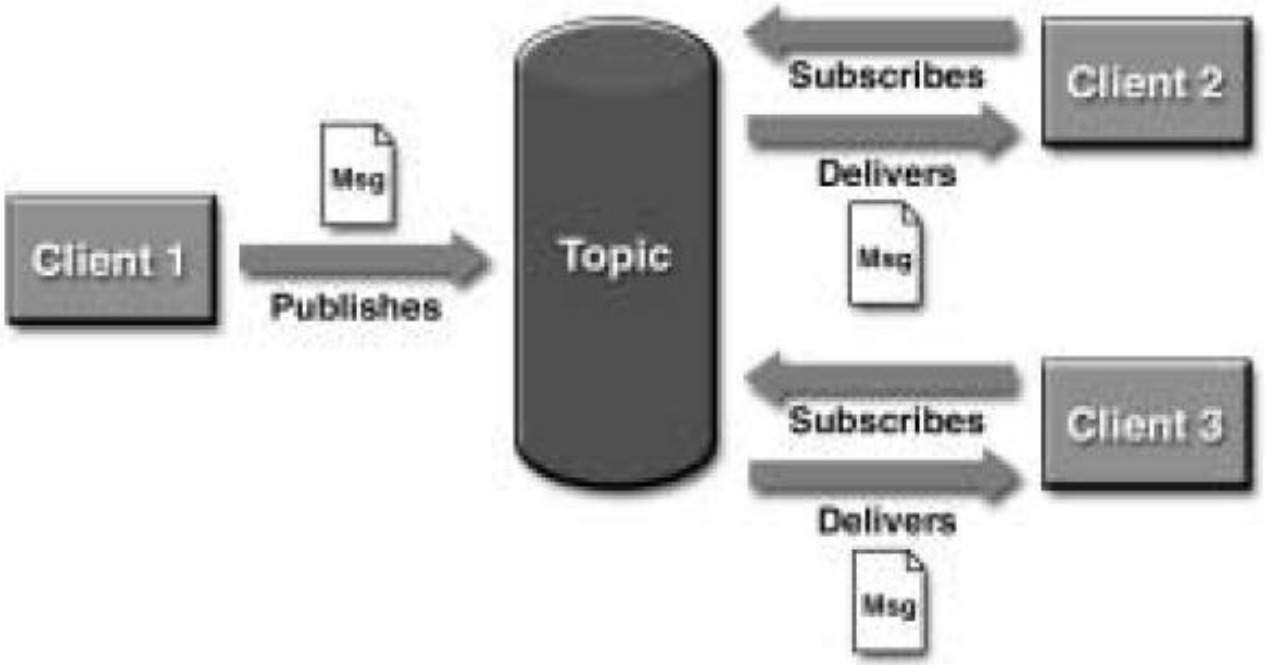


Figure 2.3 Publish/Subscribe Messaging

JMS: Mensajes

- Componentes:
  - Encabezado
  - Propiedades (opcional)
  - Cuerpo (opcional)

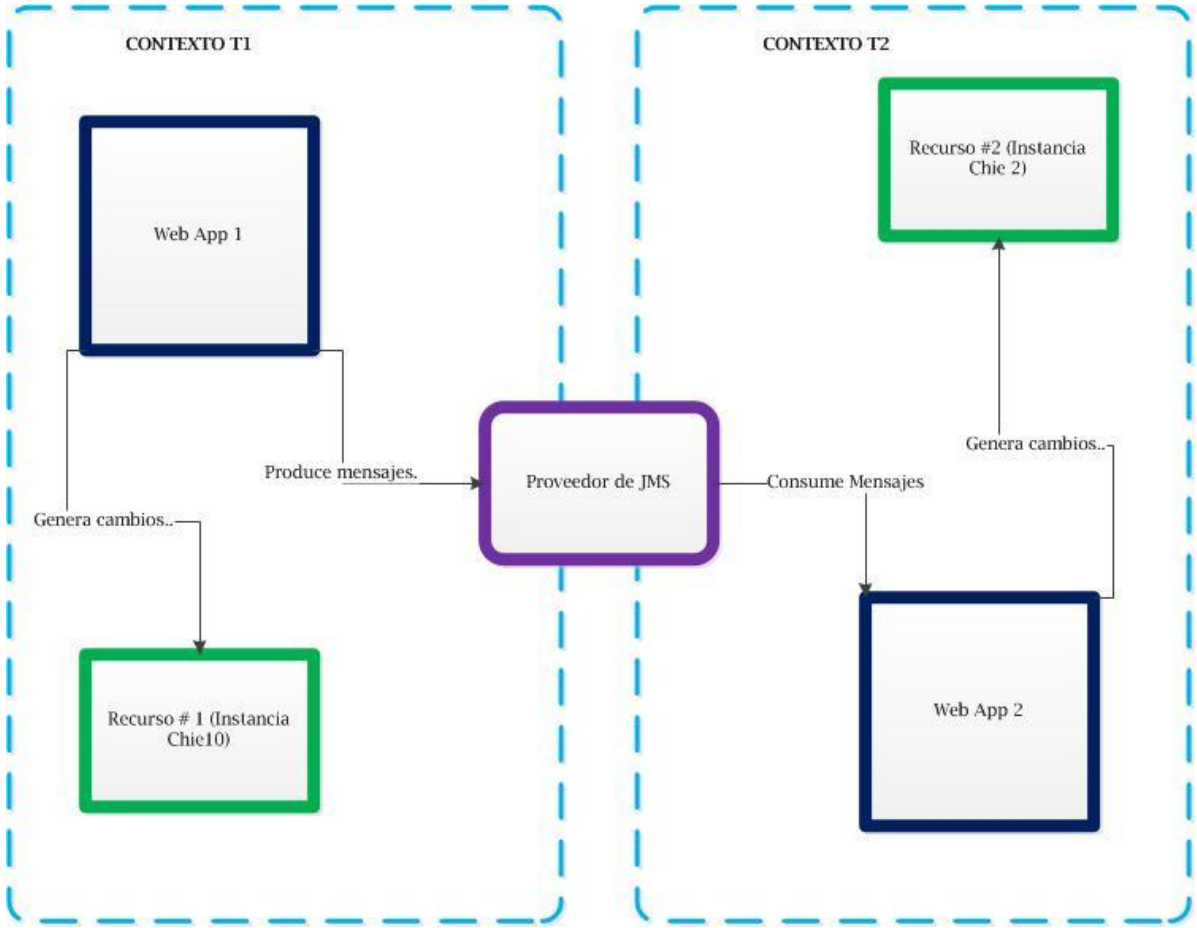
Table 3.1: How JMS Message Header Field Values Are Set

Header Field	Set By
JMSDestination	send or publish method
JMSDeliveryMode	send or publish method
JMSExpiration	send or publish method
JMSPriority	send or publish method
JMSMessageID	send or publish method
JMSTimestamp	send or publish method
JMSCorrelationID	Client
JMSReplyTo	Client
JMSType	Client
JMSRedelivered	JMS provider

Table 3.2: JMS Message Types

Message Type	Body Contains
TextMessage	A java.lang.String object (for example, the contents of an Extensible Markup Language file).
MapMessage	A set of name/value pairs, with names as String objects and values as primitive types in the Java programming language. The entries can be accessed sequentially by enumerator or randomly by name. The order of the entries is undefined.
BytesMessage	A stream of uninterpreted bytes. This message type is for literally encoding a body to match an existing message format.
StreamMessage	A stream of primitive values in the Java programming language, filled and read sequentially.
ObjectMessage	A Serializable object in the Java programming language.
Message	Nothing. Composed of header fields and properties only. This message type is useful when a message body is not required.

JMS: Ejemplo uso en transacciones distribuidas



JMS: Ejemplo uso en transacciones distribuidas

La aplicación Web App 2 crea una cola de mensajes utilizando JMS. A través de esta cola **recibe** mensajes que inician cierta transacción.

El Proveedor de JMS maneja la cola, permite enviar y recibir mensajes a los productores y receptores.

La aplicación Web App 1 inicia su transacción y **produce** un mensaje hacia la cola creada en Web App 2.

JMS: Ejemplo uso en transacciones distribuidas

Los recursos de cada aplicación representan su base de datos.

¿Cuál es el funcionamiento básico?

1. Web App 1 inicia la transacción T1, genera unos cambios en su base de datos y envía un mensaje a la cola de mensajes de Web App 2. En este momento termina la transacción T1.
2. La transacción T2 inicia cuando Web App 2 recibe el mensaje. Al recibirlo, genera unos cambios en su base de datos y termina su transacción.