Prof. Dr.-Ing. Stefan Deßloch
AG Heterogene Informationssysteme
Geb. 36, Raum 329
Tel. 0631/205 3275
dessloch@informatik.uni-kl.de

Technische Universität Kaiserslautern

Chapter 9
Message-oriented Middleware (MOM)

Outline

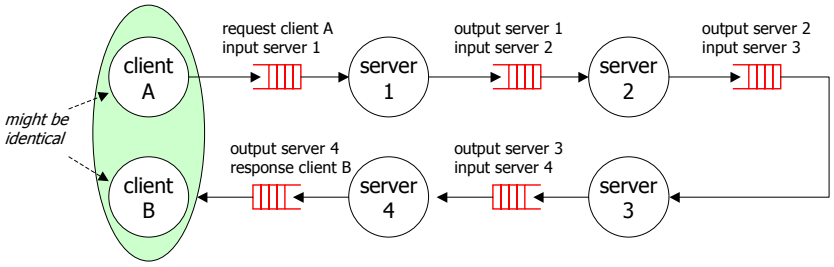
- Request-response queues in TP-monitors
  - asynchronous transaction processing
- Stratified transactions
- Message Queuing Systems
  - point-to-point, request-response
  - Java Messaging Service (JMS)
  - EJB Message-driven Beans
- Message Brokers
  - Enterprise Application Integration (EAI) – requirements
  - message routing
  - publish/subscribe, hub-and-spoke
  - message broker architecture components
- Summary

Request-Response Queues in TP-Monitors

- Load control (during direct transaction processing)
  - Handle temporary load peaks
  - Store request in (temporary) queue to avoid creating new processes
- End-user control
  - Delivering output (e.g., display information, print ticket, hand out money) is a critical step in asynchronous processing
  - Redelivery may be required until user explicitly acknowledges receipt
- Recoverable data entry
  - Some applications are driven by data entry at a high rate, without feedback to the data source
  - Optimize for high throughput (instead of short response times)
  - Input data are taken from queue by running application
  - Input data must not be lost, even during a crash
- Multi-transactional requests

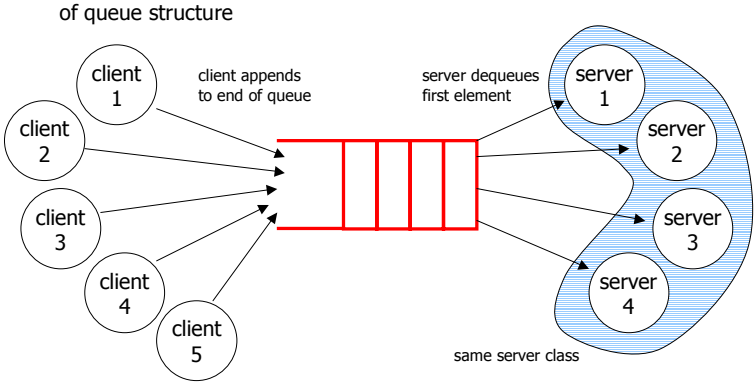
Multi-transactional Requests

- Single request processed in a sequence of multiple transactions
  - can be scheduled asynchronously for high throughput, as long as no intermediate user interactions are required
- Based on recoverable input data



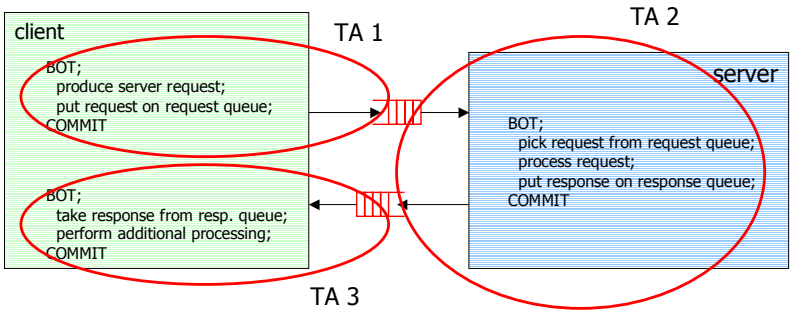
Short-term Queues

- Load control
  - Client-side model: direct, synchronous communication
    - client requests are temporarily stored in server-class-specific, volatile queues
  - "exactly-once" has to be guaranteed; concurrent access must preserve correctness of queue structure



Asynchronous Transaction Processing

- Decoupling Request Entry, Request Processing, and Response Delivery, use separate TAs for each task
  - optimize for throughput
  - avoid resource contention of single-transaction (TRPC) approach
  - can be generalized to multi-transaction requests

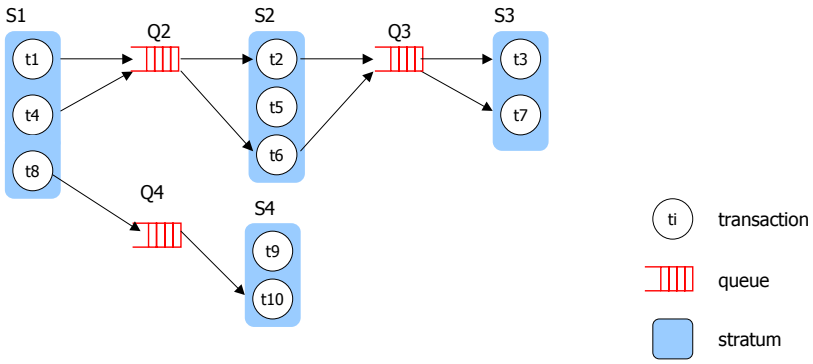


Queues for Asynchronous Transaction Processing

- Queues are durable
  - distinguishable, stable objects
  - can be manipulated through ACID transactions
    - send, receive operations are part of the respective transactions
      - queuing system is yet another transactional resource manager
    - request will become visible to other TAs only at commit of sending TA
    - if the receiving TA crashes, the request will be "put back" on the queue by the queuing system
      - server can re-process the request after recovery
    - ...
- Client view
  - Request-reply matching: for each request there is a reply
    - request-id for relating requests and responses, provided by the client
  - ACID request handling: request is executed exactly once
  - At-least once response handling: client sees response at least once
    - response may have to be presented repeatedly, e.g., after client failure/restart

Stratified Transactions

- Generalization of multi-transactional requests
  - Stratum: set of transactions to be coordinated under 2PC
    - connected through message queues
  - Connected strata form a tree structure



Stratified Transactions (2)

- Structure
  - some t_i should commit at the same time
  - disjoint, complete partitioning of T into sets of transactions $S_1, \ldots S_m$

    $$S_i \subseteq T \text{ mit } S_i \neq \emptyset \text{ und } S_i \cap S_j = \emptyset \text{ für } i \neq j \text{ und } \bigcup_{j=1}^{m} S_j = T$$

  - transactions in S_i are synchronized by 2PC
  - set of transactions S_i is called stratum
  - each S_i receives requests in a request queue Q_i
  - a queue Q_i does NOT associate more than 2 S_i
- Behavior
  - requests for stratum is only visible in input queue, if parent stratum commits
    - queues are transactional
  - all strata eventually commit if their respective parent stratum commits
    - stratified TA commits if root stratum commits
  - if stratum fails repeatedly, then this is an exception that requires manual intervention, compensation

Stratified Transactions (3)

- Advantages compared to single, global TA for T:
  - early commit of individual strata; implies less resource contention, higher throughput
  - reduced observed end user response time (commit of root stratum)
  - if all transactions in a stratum execute on the same node:
    - no network traffic for executing 2PC
    - TA-Manager coordinating global TA on respective nodes don't need to support external coordinator
- Requirements
  - all resources manipulated by transactions (including messages) need to be recoverable
  - resource managers need to be able to participate in 2PC

Message Queuing Systems (MQS)

- Have evolved from queuing systems in TP-monitors
- Message-oriented interoperability
  - programming model: message exchange
- Provide persistent message queues
  - reliable message buffer for asynchronous communication
- Loosely-coupled systems/components
  - "client" is not blocked during request processing
  - "server"
    - can flexibly chose processing time
    - can release resources/locks early
  - no propagation of security or transaction context
  - components don't need to be running/active at the same time
- Transactional MQS ("reliable MQS")
  - persistent MQS
  - guaranteed "exactly-one" semantics
  - transactional enqueue/dequeue operations

Interacting with MQS

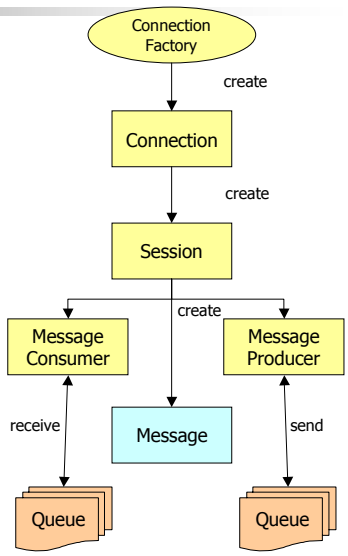
- Basic operations:
  - Connect/Disconnect to/from MQS
  - Send or Enqueue: appends a message to a MQ
    - usually non-blocking
  - Receive or Dequeue: reads and removes message from a MQ
    - usually blocking
- Variations
  - Shared Queues
    - multiple applications can access the same MQ
      - Example: load balancing by using multiple "server" components
  - Additional properties for messages
    - priority, time-out, …
  - Enhanced flexibility for "receive"
    - beyond FIFO

JMS – Standardized Interaction with MQS

- Java Messaging Service (JMS)
  - Standard-API for message queuing, message brokering (discussed later)
- Point-to-point messaging
  - components: producer, consumer, queue (destination)
  - support for multiple producers, multiple consumers per queue
    - but a message only has a single consumer
  - consumer has to acknowledge receipt of message
  - message delivery modes
    - PERSISTENT – exactly-once
    - NON_PERSISTENT – at-most-once
  - "session" concept preserving submission order of messages during transmission
  - support for transactional MQ

Architecture

- Connections
  - connect to JMS server
  - start/stop messaging service
- Session
  - execution context for sending and receiving messages by creating messages, producers, consumers
  - may be transactional
- Message
- Message producer
  - sends messages to queue
- Message consumer
  - receives messages from queue
    - synchronous receive( )
    - asynchronous using onMessage( ) method of Message Listener
- Message queues
  - administered objects, set up by administrative capabilities
  - registered/bound through JNDI



Messaging Model

- Transactions
  - MQ interactions may occur in context of a transactional session
  - session object provides commit/rollback methods with the obvious semantics on queues
  - distributed TA-support based on JTS/JTA
- Message Order
  - messages sent by a single session are received in the order in which they are sent
    - order is not defined across multiple queues or multiple session sending to the same queue
  - the sending order is affected by the following
    - message priority – messages with higher priority may jump ahead
    - non-persistent messages may be lost in case of a provider failure
    - order is only guaranteed within a delivery mode (persistent/non-persistent), if both are used
    - a transaction's order of messages

Messaging Model (2)

- Message Acknowledgement
  - messages need to be acknowledged after receiving them
    - are removed from the queue
  - queues can be recovered, resulting in redelivery of unacknowledged messages
    - messages are flagged as redelivered
- Transactional sessions
  - messages are automatically acknowledged at TA commit
  - queues are recovered automatically at rollback
- Non-transactional sessions
  - acknowledgement options
    - lazy acknowledgement – is likely to result in duplicate messages after a JMS failure
    - auto-acknowledge – automatically after a succesful receive
    - client acknowledge – explicit by calling Message.acknowledge()
      - automatically acknowledges all messages that have been delivered by its session
  - recover-method of a session will stop a session and restart it with its first unacknowledged message

Message Structure

- Header
  - standard message attributes set by JMS provider or message producer
  - message-id, correlation-id, delivery mode (persistent/not persistent), destination (queue), priority, redelivered, reply-to, timestamp
- Properties (optional)
  - application-specific, vendor-specific, and optional properties
- Body
  - actual message content
  - support for multiple content types (bytes, text, Java object, ...)
  - format of the method body is up to the applications
    - implicit agreement
    - no meta-data available

Message Selectors

- Message processing applications may implement components only interested in a subset of messages on a queue
- Queue receiver may specify a selector
  - messages that are not selected remain in the queue
  - message order is not guaranteed anymore
- Selector syntax
  - logical conditions based on a subset of SQL92 conditional expression syntax
    - literals, identifiers (field/property names)
    - logical connectors, comparison operators, arithmetic expressions
  - can reference message header fields and properties
    - no references to message body allowed

EJB Message-Driven Beans (MDB)

- Entity and session beans can use JMS to send asynchronous messages
  - receiving messages would be difficult, requires explicit client invocation to invoke a bean method "listening" on a queue
    - may block the thread until message becomes available
- Message-driven beans should be used to receive and process messages
  - stateless
    - no conversational state
    - can be pooled like stateless session beans
  - not invocable through RMI
    - don't have component interfaces (home, remote)
  - concurrent processing of messages
    - container can execute multiple instances, handles multi-threading

MDB Processing Model

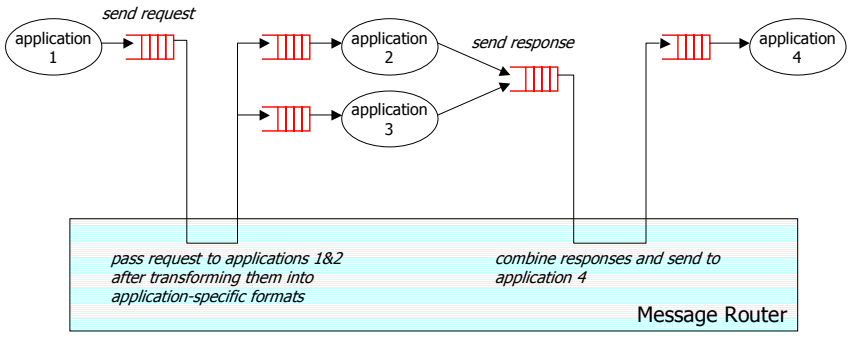
- MDB
  - implements MessageListener interface
    - onMessage(Message) method
      - receives and processes message
  - may use EJB services, interact with resource managers and other beans, send a reply message, …
- Transactional behavior
  - implemented in the same manner as for other beans
    - container-managed, bean-managed
- Deployment
  - descriptor includes additional attributes mapping to JMS processing properties
    - acknowledge-mode
    - message-selector
  - the queue from which a MDB should receive messages is fixed at deployment time

Message Queuing and Application Integration

- Message queuing characteristics
  - explicit definition, agreement regarding message destination
  - point-to-point, request-response
  - fixed message structure (content)
  - a message is always consumed by a single receiver
- Enterprise Application Integration (EAI)
  - Goal: bring together disparate application systems to exchange data and requests
  - Example: Supply Chain Automation
    - supplier/customer management, quotation, order processing, procurement, shipping, ...
  - Involves for each application
    - definition of a message set representing data/requests
    - developing an adapter that maps messages to invocation of application functions
      - front-end vs. back-end adapter
- Using plain message queuing for EAI
  - messaging application performs routing logic and required message transformations
  - hard to maintain, extend

Message Routing

- Idea: separate the routing and transformation logic from the applications
  - script defines sequences of application invocations and message transformation steps



Publish/Subscribe Paradigm

- Publish and Subscribe
  - further generalizes message routing aspects
  - applications may simply publish a message by submitting it to the message broker
  - interested applications subscribe to messages of a given type
  - message broker delivers copies of messages to all interested subscribers
- Subscription
  - can be static (fixed at deployment or configuration time) or dynamic (by application at run-time)
  - type-based subscription
    - based on defined message types
      - type namespace may be flat or hierarchical (e.g., SupplyChain.newPurchaseOrder)
    - also identified by the publisher
  - parameter-based subscription
    - boolean subscription condition identifying the messages a subscriber is interested in
      - example: type = "new PO" AND customer = "ACME" AND quantity > 1000
    - condition refers to message fields

Message Brokering

- Message Routing and Transport
  - employs queues as input/output infrastructure
    - asynchronous communication, store-and-forward
  - message flow control
- Rules-based processing and distribution of messages based on message fields
- Message broker as a neutral hub for message processing
  - hub-and-spoke interaction paradigm
  - establishes a neutral message format to reduce transformation complexity
- Message Repository
  - definition of message structure of all message sets
  - mapping rules
  - special transformation functions
  - routing scripts
  - subscription requests

Message Brokering (2)

- Message annotation
  - message can be combined with data from a database, from other messages, or both
  - annotations are defined in routing scripts or subscription requests
- Message warehouse
  - used to permanently store messages of predefined types
  - may be retrieved, annotated, projected on demand
  - basis for further analytical processing of messages
- JMS – Publish/Subscribe
  - Publishers send messaged to topics instead of queues
  - Subscribers create a special kind of receiver (topic subscriber) for a topic
    - non-durable subscription: published messages are not delivered if the subscriber is not active
    - durable subscription: messages are delivered until subscription expires

Summary

- Message Queuing
  - asynchronous interactions, communication
  - persistent and transactional MQs
  - asynchronous transaction processing
  - supported by
    - TP monitors
    - Workflow Management Systems
    - Message Queuing Middleware
- Message Broker
  - focus on application integration
  - message routing, pub/sub
  - neutral message hub
  - rule-based processing, routing, transformation of messages