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

Technische Universität Kaiserslautern

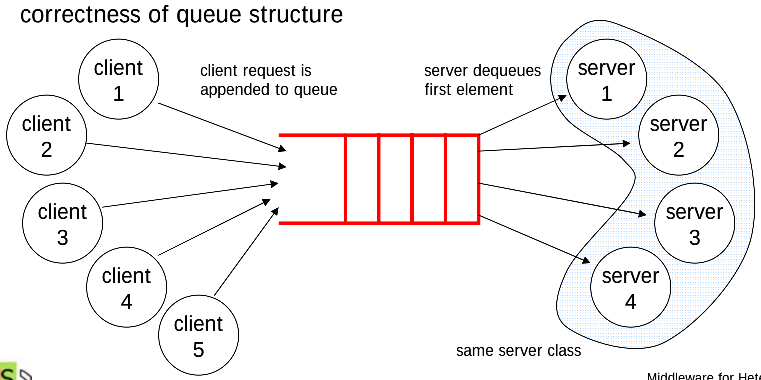
Chapter 11
Message-oriented Middleware (MOM)

Outline

- 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
  - message broker architecture components
  - hub-and-spoke topology
- Databases and Message Queuing Systems
  - roles
  - integration approaches
  - DBMS/MQS integration example

Short-term Queues for Load Control

- Load control (during direct transaction processing)
  - Handle temporary load peaks
  - Store request in (temporary) queue to avoid creating new processes
  - Client-side model: direct, synchronous communication
- "exactly-once" has to be guaranteed; concurrent access must preserve correctness of queue structure

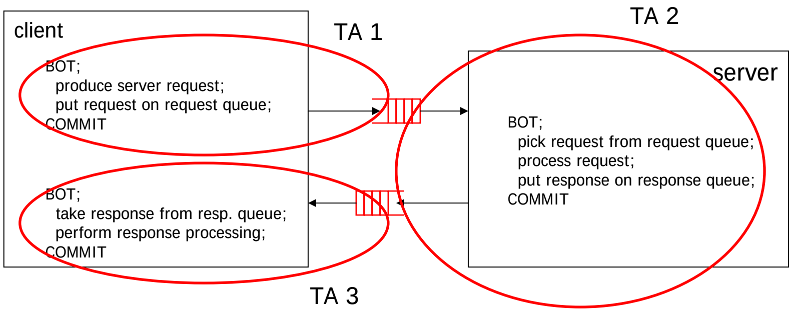


Persistent Queues in TP-Monitors

- 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
  - Single request is processed in multiple transactions
  - Transaction chaining

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

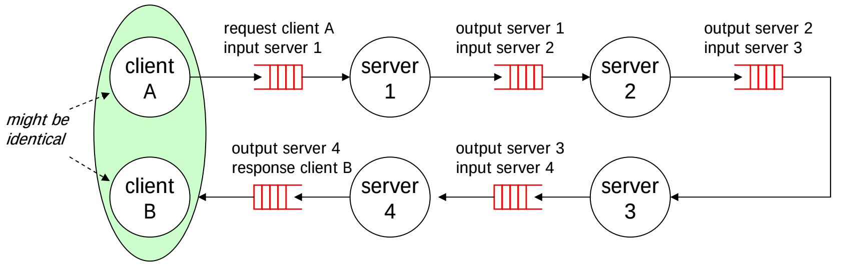


Queues for Asynchronous Transaction Processing

- Queues are persistent, transactional
  - 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
    - queue operations and operations on other RMs can happen within the same (distributed) transaction
    - 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
  - ACID request handling: request is executed exactly once
  - Request-reply matching: for each request there is a reply
    - request-id for relating requests and responses, provided by the client
  - At-least once response handling: client sees response at least once
    - response may have to be presented repeatedly, e.g., after client failure/restart

Multi-transactional Requests

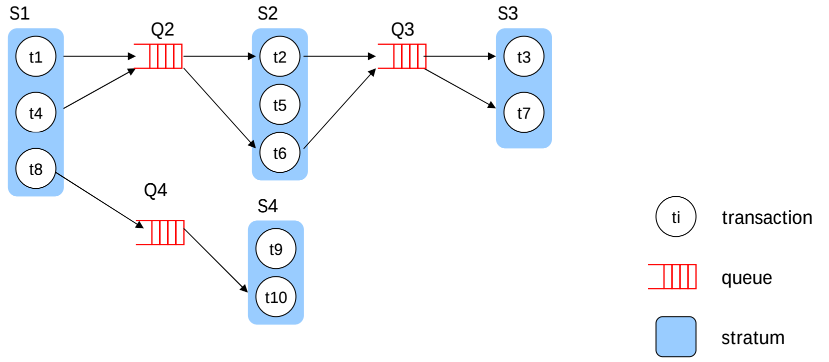
- Single request processed in a sequence of multiple transactions
  - can be scheduled asynchronously for high throughput, as long as no intermediate user interactions are required
- Based on recoverable input data (persistent queues)



- Assumption: each transaction in the sequence will finally commit
- Complete transaction sequence is no longer serializable

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, ... 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

Client Variations

- Non-transactional client
  - transaction support may not be available on the client
  - client still needs to be implemented in a fault-tolerant manner
    - make sure that the same request is not sent more than once
    - make sure that replies are delivered to the end user (at least) once
  - queuing infrastructure can help by
    - guaranteeing that message is stably stored when "enqueue message" operation returns to client
    - providing information (message-ids) about the last request submitted, last reply received when client reconnects after failure
    - allowing a client to
      - explicitly acknowledge receiving a reply
      - re-receive the unacknowledged replies
        - reply is deleted only when explicitly or implicitly acknowledged by the client
- One-way messaging
  - client requires no reply for a request
- Multiple clients submitting requests
  - one reply queue per client, identified as part of the request

Message Queuing Systems (MQS)

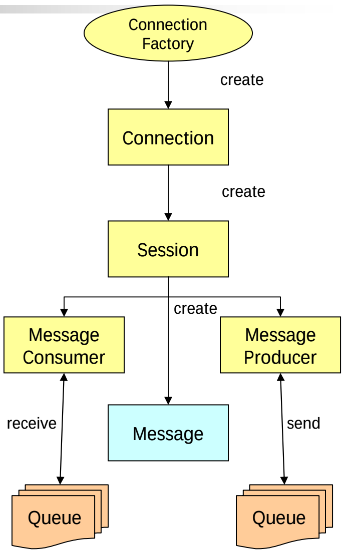
- Have evolved from queuing systems in TP-monitors
- Message-oriented interoperability
  - programming model: message exchange
- Loosely-coupled systems/components
  - "client" is not blocked during request processing
  - "server"
    - can flexibly chose processing time
    - can release resources/locks early
  - components don't need to be running/active at the same time
- Provide persistent message queues
  - reliable message buffer for asynchronous communication
  - "store and forward" behavior
- Transactional MQS ("reliable MQS")
  - persistent MQS
  - guaranteed "exactly-once" semantics
  - transactional enqueue/dequeue operations

Interacting with MQS

- Point-to-point messaging
  - Application explicitly interacts with message queues
  - Request/reply model needs to be built "on top"
- Basic operations:
  - Connect/Disconnect to/from MQS
  - Send or Enqueue: appends a message to a MQ
    - usually multiple producers can send/enqueue in the same queue
  - Receive or Dequeue: reads and removes message from a MQ
- Variations
  - Shared Queues
    - support for multiple consumers per queue
      - example: load balancing by using multiple "server" components
      - but a particular message only has a single consumer
  - Additional properties for messages
    - priority, time-out, ...
  - Enhanced flexibility for "receive"
    - beyond FIFO

JMS – Standardized Interaction with MQS

- 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

- Message delivery modes
  - PERSISTENT – exactly-once
  - NON_PERSISTENT – at-most-once
    - non-persistent messages may be lost in case of a provider failure
- 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
    - order is only guaranteed within a delivery mode (persistent/non-persistent), if both are used
    - a transaction's order of messages
  - the receiving order may further be influenced by the receiver (see subsequent chart)

Transactions and Message Acknowledgement

- Transactions
  - MQ interactions may occur in context of a transactional session
    - distributed TA-support based on JTS/JTA
  - session object provides commit/rollback methods with the obvious semantics on queues
- 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 successful 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)

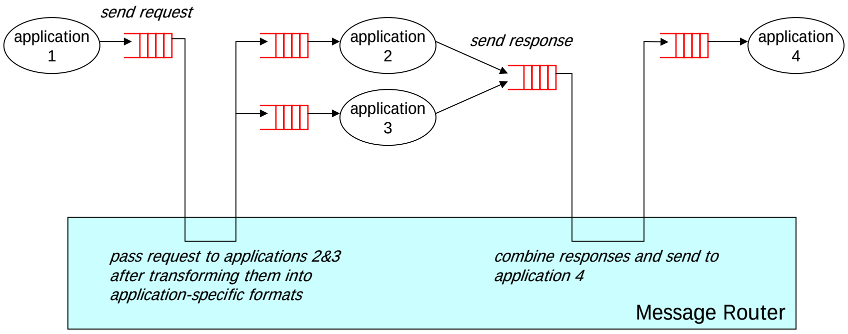
- Entity and session beans can use JMS to send asynchronous messages
  - receiving messages would be difficult, requires explicit client invocation to invoke a bean method "listening" on a queue
    - may block the thread until message becomes available
- Message-driven beans should be used to receive 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
- 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 particular 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/adapter has to perform complex routing logic and required message transformations for every application to be integrated
  - 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
    - transformations are program components invoked by the message router



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/topic
  - 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
  - non-durable subscription: published messages are not delivered if the subscriber is not active
  - durable subscription: messages are delivered until subscription expires
- JMS supports Publish/Subscribe
  - Publishers send messaged to topics instead of queues
  - Subscribers create a special kind of receiver (topic subscriber) for a topic

Message Brokering

- Message Transformations
  - restructuring (schema conversion)
  - data conversion, data cleaning (see data warehousing)
  - based on a neutral message format to reduce transformation complexity
- Message Routing and Transport
  - employs queues as input/output infrastructure
    - asynchronous communication, store-and-forward
  - performs message flow control (intelligent routing)
    - dynamic, based on message content
- Rules-based processing and distribution of messages based on message fields
- 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 Brokering (2)

- Message repository
  - definition of message structure (of all message sets)
  - mapping rules
  - special transformation functions
  - routing scripts
  - subscription requests
- Message warehouse
  - implements message persistence
  - can be used to permanently store messages of predefined types
    - may be retrieved, annotated, projected on demand
    - basis for further analytical processing of messages
  - message archiving, auditing

Message Broker Topologies

- Hub-and-spoke
  - message broker as a neutral hub for message processing
  - applications connected to broker in a "star" architecture
- Multi-hub
  - simple extension of hub-and-spoke for scalability
  - multiple message brokers are linked together
  - applications can be connected to any of the participating brokers
- Federation
  - generalizes multi-hub topology
  - heterogeneous message brokers
    - need to interact based on a common interchange format (e.g., XML)
  - applications are connected/bound to specific broker

Databases and Messaging Systems

- Roles of DBMS in a messaging world
  - persistence manager for messaging systems
    - store/retrieve messaging data and state information
    - reliable, transactional
  - provide advanced DBMS capabilities to achieve a DBMS/MQS synergy
    - querying messaging data

S. Doraiswamy, M. Altinel, L. Shrinivas, S.L. Palmer, F.N. Parr, B. Reinwald, C. Mohan: Reweaving the Tapestry: Integrating Database and Messaging Systems in the Wake of New Middleware Technologies, in T. Härder, W. Lehner (Eds.): Data Management in a Connected World, LNCS 3551, Springer 2005: 91-110

Database as a Message Store

- Database serves as a backing store
- Messaging systems can exploit integral database features, such as
  - storage definition, management, and underlying media/fabric exploitation
    - single DB table for storing similar messages of a single/few queues
    - administrator can configure the tables appropriately
  - buffer, cache, spill management
    - DB cache allows for quick access during timely message consumption
  - index creation, management, reorganization
    - on (unique) message ids, sequence numbers, subscription topics, ...
  - latching and lock management
    - avoid consumer/producers blocking on each other
    - row-level locking
    - lower isolation levels (skip over locked messages, etc.)
  - transaction management and coordination
    - synchronous or asynchronous message store/commit in local TAs, based on QoS requirements
    - global TA support
  - high-speed and scalable logging services

Improved Database and Messaging Synergy

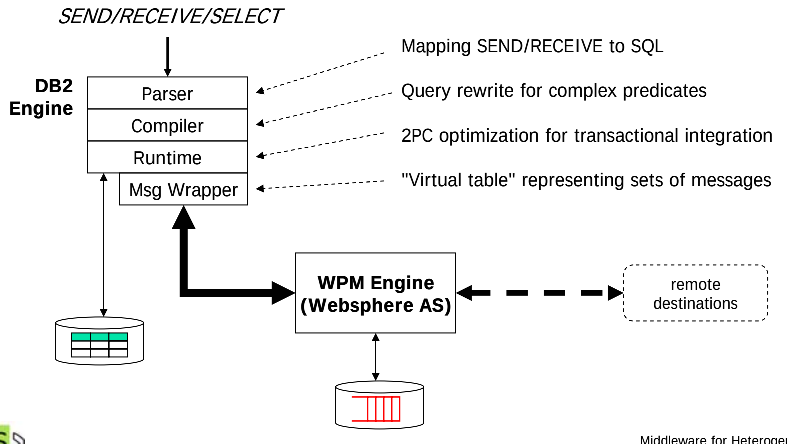
- DBMS helps accessing messaging data and destinations, possibly in combination with operational data
  - requires closer cooperation in terms of message schema and typing information
- Potential DBMS features
  - mapping message payloads structure to table structure
    - exploit object-relational and XML data capabilities of DBMS
  - message warehousing and replay functionality
    - tracking and analysis of message data
  - enabling the database for asynchronous operations
    - messaging triggers
  - use of SQL, SQL/XML, XQuery with MQS
  - publishing to message destinations as reaction to updates
    - triggers, messaging functions
    - replication
  - storing durable subscriptions
  - consume-with-wait support
    - instead of continued polling

Integration Strategies

- Database System using/integration messaging capabilities
  - database-specific messaging and queuing
    - queuing support added to the DBMS engine
  - interfacing with message engines
    - "light integration"
    - messaging data lives in DBMS, new built-in or user-defined routines to interface with a (co-located) messaging system
- Messaging system using/integrating DBMS
  - message-system-specific persistence, transactions logging
    - messaging engine implements all of the above by itself
  - database as a persistent message store
- Integrated Database Messaging
  - leverage the strengths of both DBMS and MQS, without reimplementation
  - potentially utilize additional middleware to achieve the integration
    - example: leverage (information integration) wrapper technology

Integrated Database Messaging – Example

- IBM research prototype based on DB2 Universal Database, WebSphere Platform Messaging (WPM)



SQL Language Extensions

- SEND statement
  - creates and puts a message into specific destination
  - example:

```
SEND TO stockdisplay ($body)
SELECT n.name || '#' || CHAR(q.price)
FROM quotes q, stocknames n
WHERE q.symbol = n.symbol
```

  - WPM initializes message properties
    - can be accessed by the sending application using additional syntax
  - internally represented as SELECT -> INSERT into virtual table
- RECEIVE statement
  - destructively reads a message from a destination
  - example:
    - RECEIVE $body
FROM stockdisplay
WHERE MINUTEDIFF (CURRENT TIMESTAMP – TIMESTAMP($timestamp)) < 60
  - internally represented as DELETE -> SELECT from virtual table

Message Wrapper

- Message wrapper provides a relational view of a JMS message destination
  - "virtual" table (see chapter on wrappers)
  - structure
    - each standard header field -> column
    - all application-defined properties -> single column
    - message body -> column
  - operations
    - maps DML operations and filter predicates to appropriate operations on message destinations
    - implements set-oriented semantics
- WPM does not support complex filter conditions
  - DB2 needs to compensate for lack of capabilities
  - requires two-step interaction to preserve semantics of message destination operations
    - step one
      - browse all messages that fulfill subset of search criteria supported by WPM
      - evaluate additional search conditions in DB2 engine
    - step two
      - destructively read only the qualifying messages from the destination

2PC Optimization

- DB2 and WPM are located on the same machine, can use the same DB for operational data and (local) message storage
- 2PC semantics may have to be enforced, but can be optimized
  - DB and WPM interactions with DB still occur through separate DB connections
  - tight coupling possible based on XA join/suspend behavior
    - transaction context passed along to messaging system
    - then back to DB during message interactions
    - DB2 TA-Mgr recognizes context, avoids full 2PC

Summary

- Message Queuing
  - asynchronous interactions, communication
  - persistent and transactional message queues
  - 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
- Databases and Messaging Systems
  - database as a message store
  - DBMS/MQS synergy
  - different integration strategies
    - DBMS-extension, MQS-extension, integration
  - integration example
    - SQL extensions for messaging
    - messaging wrapper
    - 2PC integration