Report on the paper

Stefan Tai, Isabelle Rouvellou,
IBM T. J. Watson Research Center, NY, USA:
Strategies for Integrating Messaging and
Distributed Object Transactions,
Middleware 2000

Marek Prochazka
March 7, 2001
Synopsis

- Message classification framework
- Analysis of sample messaging middleware
- Different strategies for integrating messaging and distributed object transactions
Introduction

• **Transactions**
  – Well known from database systems
  – Guarantee stability of data
  – Transactions in distributed object environments
    • CORBA OTS
    • Enterprise JavaBeans

• **Messaging**
  – Communication model for notification and request processing
  – Different notions (multicast, asynchronous, MQ)

• **Distributed object middleware**
  – Supports both transactions and messaging
Message Classification Framework (MCF)

• For better understanding messaging

• For developing strategies for integration of messaging and transactions

• MCF models:
  – Message delivery model
  – Message processing model
  – Message failure model
MCF Message Delivery Model

• Essentially about event notification
• Model properties
  – **Representation**
    • Object / Data / Operation invocation
  – **Messaging API**
    • Application-independent / Application-dependent / Mixed
  – **Initiation**
    • Push / Pull / Both
  – **Intermediation**
    • None / Exactly one / Multiple
MCF Message Delivery Model (cont.)

- **Multiplicity of ultimate recipients**
  - Unicast / Multicast / Broadcast

- **Anonymity of ultimate recipients**
  - Known set / Unknown set / Mixed

- **Subscription**
  - Subscription / No subscription / Mixed

- **Synchronicity**
  - Synchronous / Asynchronous

- **Delivery guarantee**
  - Best effort / At-most-once /
    Implicit or explicit exactly once / Mixed
MCF Message Delivery Model (cont.)

- **Persistence**
  - Persistent / Transient / Both

- **Ordering**
  - FIFO / LIFO / Random / Priority-based / Mixed

- **Filtering**
  - None / Header(type) / Body(content) / Both
MCF Message Processing Model

- Is about request processing and sending results
- Model properties:
  - **Processing results**
    - Single return value / Single integrated return value / Set of individual return values
  - **Communication**
    - Separate reply message / Callback / Polling
MCF Message Failure Model

• Important for integrating messaging and transactions

• Model properties
  – **Failure level**
    • Delivery / Processing
  – **Failure scope**
    • Particular recipients / Number of any recipients / All recipients
  – **Failure detection**
    • Exceptions / Timeout of ack or reply / Both
Sample Messaging Middleware

• CORBA Messaging
• CORBA Events and CORBA Notification
• Java Messaging
• Message Queueing
CORBA Messaging

• Asynchronous method invocations (AMI)
  – mapped to standard IDL
  – Callback-based: ReplyHandler object
  – Poller-based: Poller object

• Asynchronous communication

• Push model

• No intermediator

• Targeted unicast

• At-most-once delivery
CORBA Messaging (cont.)

- Time-independent invocation (TII)
  - Can outlive client and server processes
  - Persistent
  - Exactly-once semantics

- FIFO ordering

- No filtering

- Failure level: Processing

- Failure scope: Particular servers

- Failure detection: System exceptions
CORBA Events and Notification

- A message
  - Generic (IDL type `any`)
  - Typed (operation invocation of an IDL interface)
- Application-independent API
- Intermediators: `event channels`
- Both push and pull models
- Multicast communication
- Subscribers are known to event channels, not to each other
- Asynchronous communication
- Type events are CORBA operations without return values
CORBA Events and Notification (cont.)

- At-most-once delivery
- Persistency: Depends on intermediator
- Message ordering
  - CORBA Events: Undefined
  - CORBA Notification: Per-consumer priority-specified
- Filtering
  - CORBA Events: Not supported
  - CORBA Notification: Header (type)-based, content-based
- No message processing
  - Messages are parameters of CORBA requests
  - Results are other messages
Java Messaging

- A message
  - JMSMessage object specialization (BytesMessage, TextMessage, MapMessage, StreamMessage, ObjectMessage)

- Application-independent API

- Point-to-point: Queue object as intermediary

- Publish/subscribe: Topic object for subscription

- Multicast communication

- Consumers anonymous to message producer

- Asynchronous communication
Java Messaging (cont.)

- Persistent messages: Exactly-once delivery
- Transient messages: At-most-once delivery
- FIFO ordering, priority ordering
- Filtering
- Results are separate reply messages with correlated ids
- Failure level: Processing
- Failure scope: Specific servers
- Failure detection: Exceptions, processing results, timeouts
MQ Messaging

- A message = data (header + body)
- Intermediators: Message queues (input, output, multiple)
- Both push and pull models
- Consumers anonymous to sender
- Multicast communication
- No subscription mechanism
- Asynchronous communication
MQ Messaging (cont.)

• Persistent queues: Exactly-once delivery
• FIFO or priority-based ordering
• Consumer-driven filtering
• Results are separate reply messages with correlated ids
• Failure level: Processing (if request/reply model selected)
• Failure detection: Exceptions, timeouts (detected by queue managers)
Samples Comparison

- CORBA Messaging: Pure asynchronous communication
- Initiation: Push and pull (CORBA Events: event channel can push)
- Multicast
- Anonymity of ultimate recipients
- At-most-once, exactly-once delivery
- FIFO or priority-based ordering
Samples Comparison (cont.)

- CORBA Notification: Filtering by event channel
- No single result of multicast message
- Failure level
  - CORBA Events/Notification: No request processing
  - Others: Processing
- Failure scope: Particular servers
- Failure detection: Exceptions, processing results, timeouts
Integration Strategies

- MQ-Integrating Transactions
- Message Delivery Transactions
- Message Processing Transactions
- Full Messaging Transactions
MQ-Integrating Transactions

- The only one strategy supported by current distributed object-oriented middleware

- Intent: Integrate message queues as resource managers

- Concept: Messages are transactional data managed by persistent message queues
try {
    tx.begin();
    data = inputQueue.getData();
    result = distributedObject.process(data);
    outputQueue.putData(result);
    tx.commit();
} catch (Exception e) {
    tx.rollback();
}
MQ-Integrating Transactions (cont.)

- Dequeueing and processing request and enqueueing results is one atomic action
- `put()` and `get()` are local calls
- Implementation: X/Open XA resources
Message Delivery Transactions

- **Intent:** Event notification within the scope of a transaction

- **Concept:**
  - Sending messages in addition to synchronous object invocations in the scope of transaction
  - Message delivery failure causes transaction abort
  - Transaction abort causes compensation of messages sent

- **Messages become part of sphere of atomicity**
  - Compensation: dequeueing from intermediator or sending an undo message
Message Delivery Transactions (cont.)

• Implementation:
  – Not supported
  – Usually: Messages visible after successful commit
  – Message failure (compensation) not addressed at all
  – JMS: Transacted sessions
    • Messages sent after commit
    • Use XA resource manager interface
    • MQ-integrating transactions
Message Processing Transactions

• Intent:
  – Asynchronous request processing
  – Transactional servers are not required to be available at the time of request is issued
  – ACID properties guaranteed

• Concept:
  – Transaction context is shared among all transaction participants
  – Processing server is nonymous to client, but need not be available
Message Processing Transactions (cont.)

• Implementation:
  – CORBA Messaging: *unshared transactions*
    • Transaction context is not propagated from client to server
    • May be implemented using three separate transactions:
      – sending of request by client
      – delivery of request and reception of result by middleware
      – propagation of result to client

• Requires
  – ordering guarantee for results
  – identification of ordered requests as a single entity
Full Messaging Transactions

- Complete distributed transaction model with message delivery transactions and message processing transactions

- In the scope of a single transaction
  - Event notification
  - Asynchronous request processing
  - Synchronous request processing
Related work


  – Three different kinds of messages with different failure recovery semantics

  – Implicit transaction demarcation
Future work

• Middleware support for full messaging transactions

• Advanced distributed transaction model being developed in two related projects in IBM Watson