Assignment 1 Documentation

Alexandre Payumo

# Summary:

The Distributed Supply Management System (DSMS) is a sophisticated distributed application that manages inventory across three geographically distributed stores: Quebec (QC), Ontario (ON), and British Columbia (BC). The system demonstrates advanced distributed systems concepts including CORBA (Java IDL) for client-server communication, UDP for inter-server communication, comprehensive concurrency control, and atomic two-phase commit protocol for cross-store item exchanges.

# System Architecture:

**A computer screen shot of a diagram

AI-generated content may be incorrect.**

**Component Architecture**

**Core Components:**

- **StoreServer Interface**: Defines remote methods for RMI communication

- **StoreServerImpl**: Main server implementation with business logic

- **UDPCommunicator**: Handles marshalled inter-server communication

- **ManagerClient/CustomerClient**: User interface applications

- **DSMSLogger**: Comprehensive logging system

- **StoreServer.idl**: CORBA IDL interface definition (replaces Java RMI interface)

- **StoreServerPOA**: Portable Object Adapter skeleton (auto-generated from IDL)

- **StoreServerHelper/Holder**: CORBA helper classes for type conversion (auto-generated)

- **ORB (Object Request Broker)**: CORBA runtime that manages distributed communication

- **ExchangeTransaction**: Data structure tracking pending exchange operations for two-phase commit

**Data Models:**

- **Item**: Inventory item with ID, name, quantity, price

- **Purchase**: Purchase record with customer, item, date, price

- **UDPMessage/Request/Response**: Marshalled communication objects

- **ExchangeTransaction:** Tracks customer ID, new item ID, old item ID, and timestamp

  for pending exchange operations during two-phase commit protocol

**Communication Protocols**

**CORBA Communication (Client ↔ Server) :**

- Port: 1050 (CORBA Naming Service - orbd)

- Protocol: CORBA/IIOP (Internet Inter-ORB Protocol)

- Methods:

  \* Manager: addItem, removeItem, listItemAvailability

  \* Customer: purchaseItem, findItem, returnItem, exchangeItem (NEW)

  \* Helper: addToWaitlist, getStorePrefix

- Key Difference from RMI: Uses IDL-generated stubs/skeletons instead of RMI stubs,

  platform-independent interface definition, IIOP protocol instead of JRMP

**UDP Communication (Server ↔ Server):**

- Ports: 8001 (QC), 8002 (ON), 8003 (BC)

- Protocol: UDP with marshalled Java objects

- Operations: Cross-store purchases, inventory searches

# Technical Implementation:

**Data Structures**

A computer screen shot of a program

AI-generated content may be incorrect.

**Design Benefits:**

- **Scalability**: ConcurrentHashMap allows multiple concurrent readers

- **Consistency**: ReadWriteLocks ensure data integrity during updates

- **Performance**: Minimal lock contention with per-item locking strategy

# Test Scenarios:

**Test Case 1: Manager Operations**

**Description:** Testing manager functionality including add, remove, and list operations with security validation using CORBA interface.

**Test Scenarios:**

- Add new items to inventory

- Update existing item quantities (aggregation)

- Remove partial quantities from items

- Remove all quantity (set to 0, trigger waitlist)

- List inventory with proper formatting

- Invalid manager ID rejection

**Expected Output:** Successful item management with proper security enforcement through CORBA POA.

**Actual Results: ALL TESTS PASSED (6/6)**

**Analysis:** Perfect security implementation with role-based access control working correctly through CORBA. The POA-based servant properly validates manager credentials before executing operations.

**Additional Tests Verified:**

- Waitlist auto-assignment when manager adds quantity

- Complete item removal from inventory

- Remove non-existent item error handling

- Cross-store manager operation prevention

**Test Case 2: Customer Purchase Operations**

**Description:** Testing customer purchase functionality including local/cross-store purchases with quantity support, item searches, budget management, and waitlist handling via CORBA client-server communication.

**Test Scenarios:**

- Local purchase with quantity specification

- Remote purchase from another store (UDP coordination)

- Multi-store item search across all three stores

- Insufficient budget rejection

- Insufficient quantity handling

- Out of stock triggers waitlist prompt

- Add customer to waitlist

- Invalid quantity rejection (0 or negative)

**Expected Output:** Successful customer operations with proper business rule enforcement and CORBA-based remote method invocation.

**Actual Results: ALL TESTS PASSED (8/8)**

**Analysis:** Excellent business logic implementation with proper UDP cross-store communication. CORBA seamlessly handles remote method invocations while UDP manages inter-server coordination. The new quantity-based purchasing feature enhances flexibility.

**Additional Tests Verified:**

- Waitlist user choice handling ("Yes"/"No")

- Automatic purchase from waitlist when item becomes available

- Multiple customers in same waitlist (position tracking)

- Cross-store waitlist functionality

- Multiple same-name items across stores

**Test Case 3: Customer Return Operations**

**Description:** Testing item return functionality with 30-day policy enforcement and purchase validation.

**Test Scenarios:**

- Valid return within 30 days

- Reject expired return (>30 days)

- Reject return of non-purchased item

**Expected Output:** Successful returns within policy window with proper refund processing.

**Actual Results: ALL TESTS PASSED (3/3)**

**Analysis:** Robust return policy implementation with accurate date calculation and purchase history validation. CORBA interface properly handles return requests with atomic budget refunds.

**Additional Tests Verified:**

- Return wrong customer (customer can't return others' items)

- Cross-store return (return item purchased from different store)

- Budget correctly updated after return

**Test Case 4: Customer Exchange Operations**

**Description:** Testing the new exchangeItem operation with atomicity guarantees, including local and cross-store exchanges with 30-day policy enforcement.

**Test Scenarios:**

- Local exchange (same store)

- Cross-store exchange (different stores)

- Reject exchange of expired item (>30 days)

- Reject exchange of non-owned item

**Expected Output:** Atomic exchange operations where both return and purchase succeed or both fail, maintaining data consistency across stores.

**Actual Results: ALL TESTS PASSED (4/4)**

**Analysis:** Excellent implementation of atomic two-phase exchange protocol. Cross-store exchanges properly coordinate via UDP with PREPARE→COMMIT/ROLLBACK phases. Budget adjustments (price differences) handled correctly. No data corruption observed during failure scenarios.

**Additional Tests Verified:**

- Exchange with price increase (customer pays difference)

- Exchange with price decrease (customer receives refund)

- Rollback on remote store failure

- Cross-store purchase limit enforcement during exchange

- Exchange eligibility validation before execution

**Test Case 5: Edge Cases & Business Rules**

**Description:** Testing system constraints, security boundaries, and business rule enforcement.

**Test Scenarios:**

- Enforce remote store purchase limit (1 item per remote store)

- Invalid customer ID rejection

- Prevent customer from manager operations

- Prevent manager from customer operations

- Track customer budget correctly across operations

**Expected Output:** Proper enforcement of all business rules and security constraints through CORBA interface validation.

**Actual Results: ALL TESTS PASSED (5/5)**

**Analysis:** Comprehensive business logic validation. CORBA POA servant properly distinguishes between customer and manager roles. Purchase limits enforced across stores via centralized purchase history tracking.

**Test Case 6: Concurrency & Synchronization**

**Description:** Testing thread safety with multiple concurrent operations on shared resources.

**Test Scenarios:**

- Handle concurrent purchases of same item

- Handle concurrent add/remove operations

**Expected Output:** No race conditions, data corruption, or deadlocks. Proper synchronization ensures data integrity.

**Actual Results: ALL TESTS PASSED (2/2)**

**Analysis:** Outstanding concurrency control using ReentrantReadWriteLock for item-level locking. Multiple threads can safely operate on different items concurrently while preventing conflicts on same items. CORBA's multi-threaded ORB properly handles concurrent client requests without blocking.

**Additional Tests Verified:**

- Concurrent exchanges on different items

- Simultaneous purchases and returns

- Multiple managers modifying inventory simultaneously

- Waitlist processing during concurrent add operations

- No deadlocks when acquiring multiple locks (ordered locking prevents deadlock)

# Most Important/Difficult aspects:

**Problem:** Implementing Atomic Cross-Store Exchange with Two-Phase Commit

**Complexity:**

The exchange operation required implementing a distributed transaction protocol that guarantees atomicity across multiple autonomous servers, each managing their own state. This is significantly more complex than simple operations because:

- Coordinating distributed transaction across independent servers

- Ensuring true atomicity (both operations succeed or both fail)

- Handling network failures during multi-step process

- Preventing partial state visibility to clients

- Maintaining consistency across distributed servers under concurrent load

- Dealing with concurrent exchanges competing for same items

- Implementing proper rollback/compensation logic

- Preventing deadlocks with multiple resource locks

**Solution – Two-Phase Commit Protocol:**

A screen shot of a computer program

AI-generated content may be incorrect.

**Achievements:**

- Type-safe object serialization

- Comprehensive error handling with structured error codes

- Timeout management for reliability

- Backward compatibility with string-based methods