Chapter 5
Application Server Middleware

Outline

- Types of application server middleware
  - tasks
- TP monitors
- CORBA
- Server-side components and EJB
- Summary
Types of Application Server Middleware

- RPC/RMI middleware infrastructure
  - basic development and execution support
  - additional services
- TP monitor
  - transaction management, TRPC
  - process management
  - broad set of capabilities
- Object broker (e.g., CORBA)
  - distributed object computing, RMI
  - additional services
- Object transaction monitor
  - ... = TP monitor + object broker
  - most often: TP monitor extended with object-oriented (object broker) interfaces
- Component Transaction Monitor
  - ... = TP monitor + distributed objects + server-side component model

Middleware Tasks

- Distributed computing infrastructure (RPC, RMI)
- Transactional capabilities
  - programming abstractions (demarcation)
  - distributed transaction management
- Security services
  - authentication, authorization, secure transmission, ...
- Unified access to heterogeneous information sources and application systems
- Scalable and efficient application processing
  - large number of client applications or end users
- Reliability, high availability

*Programming model abstractions that allow the developer to focus on application logic (i.e., ignore infrastructure as much as possible)*
Java RMI

- Location-transparency
- Platform-independence
- Java only
- Additional drawbacks
  - no standardized RMI format/protocol
  - missing support for important information systems services
    - transactions, security, ...
  - no support for remaining middleware tasks

TP Monitor

- Provides functionality to develop, run, manage, and maintain transactional distributed IS
  - transaction management
  - process management
- Additional capabilities (beyond TRPC)
  - high number of connected clients/terminals \((10^2 - 10^4)\)
  - concurrent execution of functions
  - access shared data
    - most current, consistent, secure
  - high availability
    - short response times
    - fault tolerance
  - flexible load balancing
  - administrative functions
    - installation, management, performance monitoring and tuning
- One of the oldest form of middleware
  - proven, mature technology
Scalable and Efficient Application Processing

- Managing large workloads
  - one process per client is not feasible
  - TP monitor manages server pools
    - groups of processes or threads, pre-started, waiting for work
  - client requests are dynamically directed to servers
  - extends to pooling of resource connections
- Load balancing
  - distribute work evenly among members of pool
  - TP monitor can dynamically extend/shrink size of server pools based on actual workload
  - management of priorities for incoming requests

Basic Components of a TP Monitor

- Interface
  - programs and terminals
- Program flow
  - store, load, execute procedures
- Router
  - maps logical resource operations to physical resources (e.g., DBMS)
- Communication manager
  - infrastructure for communicating with resources
- Transaction manager
- Wrappers
  - hide heterogeneity of resources
- Services
  - security, performance management, high availability, robustness to failures, ...
Transactional Services

- Need to strictly distinguish TP monitor and TA manager functionality
  - many users/applications don't need a TP monitor: batch applications, ad-hoc query processing
  - special application systems (e.g., CAD) have their own (terminal) environment
  - but all need transactional support
- Separation of components for
  - transactional control (TA manager)
  - transaction-oriented scheduling and management of resources (TP monitor)

CORBA - Introduction

- CORBA: **Common Object Request Broker Architecture**
- Object-oriented, universal middleware platform
  - object bus architecture based on RMI concept
  - language-independent
  - platform-independent
- OMG
  - industry consortium (founded in 1989, 11 members)
  - today over 1000 members
  - creates specifications (no standard/reference implementations)
- First CORBA products appeared in the 90's
  - e.g., IONA's Orbix in 1993 (for C and C++)
CORBA – Reference Model

- Object Management Architecture (OMA)
  - Interfaces in different categories
    - Application Interfaces
    - Object Services (horizontal)
    - Domain Interfaces (vertical)
      - Telecommunication, Finance, E-Commerce, Medicine, ...

CORBA – Interface Definition Language

- IDL defines:
  - Types
  - Constants
  - Object-Interfaces (Attributes, Methods and Exceptions)
- Independent of programming language
  - language-specific IDL bindings and compilers
CORBA IDL - Example

Module PizzaService {
    interface OrderService {
        void newOrder  (in long custNo, out long orderNo);
        void addItem  (in long orderNo,
                        in long pizzaNo,
                        in long count);
    }
    interface DeliveryService {
        long delivery(in long custNo);
    }
}

interface Order {
    readonly attribute long id; // only get-method
    attribute Date deliveryDate; // Date is an IDL interface
    void addItem(in long pizzaId, in long pizzaCount);
}

CORBA – Core Components

- Object References (Interoperable Object References, IOR)
- Object Request Broker (ORB)
- Object Adapter
- Stubs and Skeletons
- Dynamic Invocation/Skeleton Interface (DII/DSI)

- Service-specific: Stub, Skeleton
- Identical for all applications: ORB Interface, DII, DSI
CORBA – ORB and Object Adapter

- ORB
  - provides network communication and connection management
  - manages stubs (client-side)
  - maps RMI to object adapter (server side)
  - provides helper functions (e.g., converting object references)

- Object adapter: Portable Object Adapter (POA)
  - generates object references
  - maps RMI to server objects
  - activates/deactivates/registers server objects
  - may perform multi-threading, ...

- ORB + object adapter = request broker

CORBA – Static and Dynamic Invocation

- Static invocation
  - stub and skeleton generated by IDL compiler
  - IDL interface is mapped to specific programming language
    - static type checking (at compile time)

- Dynamic invocation
  - object interfaces (meta data) can be discovered/selected at run-time using interface repository
  - generic DII (dynamic invocation interface) operations are used to construct and perform a request
  - dynamic type checking (at run-time)
  - more flexible, but less efficient than static invocation
**CORBA – “On the wire”**

- **Data format:**
  - defines encoding of data types
  - defines responsibilities for required conversions
  - Common Data Representation (CDR)

- **Communication protocol**
  - defines client/server interactions
  - message format
  - message sequence
  - CORBA 2.0: General Inter-ORB Protocol (GIOP)
  - Internet-Inter-ORB-Protocol (IIOP)
    - maps GIOP to TCP/IP
    - internet as “Backbone-ORB”
  - optional: Environment-Specific Inter-ORB Protocols (ESIOP)
    - example: DCE Common Inter-ORB Protocol (DCE-CIOP)

**CORBA Object Services**

- **Goal:** extend basic ORB capabilities to provide additional OTM system services
  - Naming, Life Cycle, Events, Persistence, Concurrency Control, Transaction, Relationship, Externalization, Query, Licensing, Properties, Time, Security, Trading, Collections

- **Service usage**
  - functionality defined using CORBA-IDL
  - CORBA object invokes method of service object
    - Example: NameService
  - CORBA object implements interface provided as part of a service (may not need to provide any code)
    - Example: TransactionalObject
CORBA – Object Transaction Service

- Based on X/OPEN DTP model and capabilities
  - (flat) ACID transactions
  - optional: nested transactions
  - TAs may span across ORBs
  - X/OPEN DTP
  - interoperability with "procedural" TA-Managers
- Roles and interfaces
  - transactional client
    - demarcation (begin, commit, rollback)
    - uses OTS Interface \textit{Current}
  - transactional server
    - participates in TA, does not manage any recoverable resources
    - "implements" OTS Interface \textit{TransactionalObject}
      - only serves as a "flag" to have the ORB propagate the transaction context
    - optionally uses OTS Interface \textit{Current}
  - recoverable server
    - participates in TA, manages recoverable resources
    - implements OTS Interface \textit{TransactionalObject} and \textit{Resource}, uses \textit{Current} and \textit{Coordinator}
      - participates in 2PC

OTS – Elements and Interaction
Server-side Component Models

- Problems with CORBA (up to 2.0)
  - complex, non-standard programming of server objects
  - service usage (transactions, security, ...)
    - behavior fixed at development time
  - resource management, load balancing
    - proprietary programming model and interfaces, is supported by object adapter
- Standardized Server-side component model
  - defines a set of "contracts" between component and component server for developing and packaging the component
  - developer focuses on application logic
    - service use can be defined at deployment time by configuring the application component
      - code generation as part of deployment step
    - resource management, load balancing realized by application server
      - component only has to fulfill certain implementation restrictions
  - server components are portable

Enterprise JavaBeans (EJBs)

- Standard server-side components in Java
  - encapsulates application logic
    - business object components
    - can be combined with presentation logic component models
      - servlets, JSPs
  - EJB container
    - run-time environment for EJB
      - provides services and execution context
      - Bean-container-contract
        - EJB implements call-back methods
- Interoperability with CORBA
  - invocation: RMI/IIOP
  - services
EJB – Types Of Objects

- **Session Object**
  - realizes business activity or process
  - often remotely accessible, "course-grained"
  - relatively short-lived (transient)

- **Entity Object** *(see next chapter)*
  - represent persistent, transactional business object
  - usually locally accessible, "fine-grained"
  - can be long-lived

- **Message-driven Object**
  - asynchronous, message-oriented invocation *(see subsequent chapter)*
  - facilitates integration with existing applications

EJB - Concepts

- **Enterprise Bean (EB)** consists of *(ejb-jar file)*:
  - class implementing business logic *(Bean, e.g., CartBean)*
  - bean business interface, defining methods *(e.g., Cart)*
    - remote and/or local access
  - deployment descriptor/meta-data

- **Client** interacts with bean using **business interface object**
  - generated at deployment time
  - contains infrastructure code *(transaction & security support, ...)*
  - client obtains reference to interface object using JNDI *(or dependency injection)*
Session Beans

- Realization of session-oriented activities and processes
  - isolates client from entity details
  - reduces communication between client and server components
- Session beans are transient
  - bean instance exists (logically) only for duration of a "session"
- stateless session bean
  - state available only for single method invocation
- stateful session bean
  - state is preserved across method invocation
    - session context
  - association of bean instance with client necessary
- singleton session bean
  - a single bean instance is shared across applications with concurrent access support
  - not persistent, but can manipulate persistent data
    - example: use JDBC, SQLJ to access RDBMS

Example

- look up Cart interface
  ```java
  @Resource SessionContext ctx;  //use dependency injection to obtain JNDI context
  Cart cart = (Cart) ctx.lookup("cart"); //perform lookup, autom. creates EB object
  ```
- call method to initialize bean
  ```java
  cart.startShopping("John", "7506");
  ```
- invoke bean methods
  ```java
  cart.addItem(66);
  cart.addItem(22);
  ...
  ```
- remove session bean
  ```java
  cart.close() // the "close" method was annotated/declared as a "RemoveMethod"
  ```
Deployment

- EB is portable, server-independent
- Component properties
  - mapping of bean attributes to DB structures
  - configuration regarding transactional behavior
  - configuration of security aspects
- Specified using
  - source code annotations (specified at development time)
  - an XML deployment descriptor (customization at deployment time)
- What happens during deployment
  - generation of glue-code based on component properties
  - make classes and interfaces known
  - setting environment/context variables

Demarcation of Transactions
Transactional Object Behavior

Transaction Management Approaches

- Explicit (programmatic) management
  - method interacts with TA manager using demarcation API
    - begin, commit, rollback
    - suspend, resume
  - management of transaction context
    - direct: passed along as explicit method parameter
    - indirect (preferred!): a "current" TA context is propagated automatically
- Implicit (declarative) management
  - separate specification of transactional properties
    - can be realized/modified independent of application logic
    - may be deferred to deployment phase
  - supported through container services
- Combination of both approaches in distributed IS
Explicit Demarcation with JTA

- Can be used by EJB Session Beans and EJB client, web components
  - EJB: in descriptor transaction-type = Bean
  - not supported for EntityBeans
- Demarcation uses JTA UserTransaction
  - `begin()` – creates new TA, associated with current thread
  - `commit()` – ends TA, current thread no longer associated with a TA
  - `rollback()` – aborts TA
  - `setRollbackOnly()` – marks TA for later rollback
    - beans with implicit TA-mgmt can use method on `EJBContext`
  - `setTransactionTimeout(int seconds)` – sets timeout limit for TA
  - `getStatus()` – returns TA status information
    - active, marked rollback, no transaction, ...
- Stateless SessionBeans
  - `begin()` and `commit()` have to be issued in the same method
- Stateful SessionBeans
  - `commit()` and `begin()` can be issued in different methods
  - TA can remain active across method invocations of the same bean

Implicit (Declarative) Demarcation in EJB
EJBs – Transactional Properties

- Transaction attributes for methods specified in deployment descriptor:

<table>
<thead>
<tr>
<th>TA-Attribute</th>
<th>Client-TA</th>
<th>TA in method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not Supported</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>Supports</td>
<td>none</td>
<td>T1</td>
</tr>
<tr>
<td>Required</td>
<td>none</td>
<td>T1</td>
</tr>
<tr>
<td>RequiresNew</td>
<td>none</td>
<td>T2</td>
</tr>
<tr>
<td>Mandatory</td>
<td>none</td>
<td>error!</td>
</tr>
<tr>
<td>Never</td>
<td>none</td>
<td>T1</td>
</tr>
</tbody>
</table>

Recommended for CMP entity beans

Transactions in Java EE

- Application component may use Java Transaction APIs (JTA)
- UserTransaction object provided via JNDI (or EJB-context)
JDBC - Distributed Transaction Support

- Requires interaction with a transaction manager
  - X/Open DTP, Java Transaction Service (JTS)
- Demarcation of transaction boundaries
  - Java Transaction API (JTA)
    - UserTransaction Object
  - NOT using methods of Connection interface
- JDBC defines additional interfaces to be supported by a driver implementation to interact with transaction manager
  - XADataSource, XAConnection, ...
- DataSource interface helps to make distributed transaction processing transparent to the application

Connection Pooling

- Improves performance, scalability
  - establishing a connection is expensive
    - communication/storage resources
    - authentication, creation of security context
- Server-side application components
  - DB access often in the context of few (shared) user ids
  - connection is often held only for short duration (i.e., short processing step)
- Reuse of physical DB connection desirable
  - open -> "get connection from pool"
  - close -> "return connection to pool"
- Connection pooling can be "hidden" by DataSource, Connection interfaces
  - transparent to the application
Distributed Transaction Processing with JDBC

JTS Architecture

(EJB-) application server  resource adapter  resource manager

javax.transaction.TransactionManager

javax.transaction.xa-XAResource

Org.com.CosTransactions

JTS TM

JTA

Java-Mapping of CORBA

JTS or OTS transaction manager (TM)
EJB Resource Management

- Traditional task of a (component) TP monitor
  - pooling of resources, load management and balancing
- EJB specification
  - Instance Pooling and Instance Swapping
    - EJB server manages (small) number of Enterprise Beans
    - reuse, dynamic selection for processing incoming requests
    - made possible by ‘indirect’ bean access via EJB object
    - usually only applicable for stateless session beans and for entity beans
  - Passivation and Activation
    - bean state can be stored separately from bean (passivation)
    - allows freeing up resources (storage), if bean is not used for a while (e.g., end user think time)
    - if needed, bean can be reactivated (activation)
    - uses Java Serialization
    - can also be used for stateful session beans
- "Disallowed" for EJB developers:
  - creating threads, using synchronization primitives
  - I/O, GUI operation
  - network communication
  - JNI

CORBA Component Model

- Motivated by success of EJB component model
- New CORBA Component Model (CCM) as middle-tier infrastructure
  - adds successful EJB concepts
  - separates implementation from deployment
  - provides container capabilities (transactions, persistence, security, events)
  - interoperability with EJBs
- Advantage: CORBA components can be implemented in various programming languages
Summary

- Distributed computing infrastructure and transactional capabilities are core application server middleware features
- Different types of application server middleware
  - TP monitors, object brokers, object transaction monitors, component transaction monitors
- Additional tasks addressed by middleware
  - security, uniform access to heterogeneous resources, scalable and efficient application processing, reliability, high availability, ...
  - server-side component model
    - high-level abstractions
    - portability of server components
    - deployment phase
- Broad variance of support for these tasks
- Convergence of different types of middleware