DISTRIBUTED SYSTEMS (COMP9243)

Lecture 8c: Middleware

① Introduction
② Publish/Subscribe Middleware
③ Distributed Object Middleware
   • Remote Objects & CORBA
   • Distributed Shared Objects & Globe
Middleware

Machine A  Machine B  Machine C

Distributed applications

Middleware services

Network OS services

Kernel

Network

Network OS services

Kernel

Network OS services

Kernel
KINDS OF MIDDLEWARE

Distributed Object based:

→ Objects invoke each other's methods
Message-oriented:

- Messages are sent between processes
- Message queues
Coordination-based:

- Tuple space

A JavaSpace

Insert a copy of A

Insert a copy of B

Look for tuple that matches T

Return C (and optionally remove it)
Publish/Subscribe

Publisher

Subscriber

Subscriber

Data Item

Subscription

Match

Match

Publish/Subscribe Middleware

KINDS OF Middleware
Transaction Processing Monitors:
Web Services:

- **Stock Service**
  - query_stock
  - buy
  - sell

- **Bank Service**
  - balance
  - transfer

- **Auction Service**
  - search
  - get_auction
  - manage_auction
  - bid

- **Photo Service**
  - search
  - add_photo
  - delete_photo
  - update_photo

Communication Protocols:
- **HTTP**
- **XML-RPC**
- **SOAP**
**CHALLENGES**

**Transparency:**
- loose coupling → good transparency

**Scalability:**
- Potentially good due to loose coupling
- In practice hard to achieve
- Number of subscriptions
- Number of messages

**Flexibility:**
- Loose coupling gives good flexibility
- Language & platform independence
- Policy separate from mechanism

**Programmability:**
- Inherent distributed design
- Doesn’t use non-distributed concepts
EXAMPLES

Real-time Control Systems:
  ➜ External events (e.g. sensors)
  ➜ Event monitors

Stock Market Monitoring:
  ➜ Stock updates
  ➜ Traders subscribed to updates

Network Monitoring:
  ➜ Status logged by routers, servers
  ➜ Monitors screen for failures, intrusion attempts

Enterprise Application Integration:
  ➜ Independent applications
  ➜ Produce output as events
  ➜ Consume events as input
  ➜ Decoupled
MESSAGE FILTERING

Topic-based

Publisher

Data item: comp.os.distributed

Subscriber

Subscription: comp.os.*

Publish/Subscribe Middleware

Content-based

Publisher

Data item: name=john age=30

Subscriber

Subscription: name=john

Publish/Subscribe Middleware
ARCHITECTURE

Centralised:

![Centralised Diagram]

Peer-to-Peer:

![Peer-to-Peer Diagram]

Multicast-based:

![Multicast-based Diagram]
COMMUNICATION

➡ Point-to-point
➡ Multicast
  • hard part is building appropriate multicast tree
➡ Content-based routing
  • point-to-point based router network
  • make forwarding decisions based on message content
  • store subscription info at router nodes
Replication

Replicated Brokers:

➤ Copy subscription info on all nodes
➤ Keep nodes consistent
➤ What level of consistency is needed?
➤ Avoid sending redundant subscription update messages

Partitioned Brokers:

➤ Different subscription info on different nodes
➤ Events have to travel through all nodes
➤ Route events to nodes that contain their subscriptions
Fault Tolerance

Reliable Communication:
  ➜ Reliable multicast

Process Resilience (Broker):
  ➜ Process groups
  ➜ Active replication by subscribing to group messages

Routing:
  ➜ Stabilise routing if a broker crashes
  ➜ Lease entries in routing tables
EXAMPLE SYSTEMS

TIB/Rendezvous:
- Topic-based
- Multicast-based

Java Message Service (JMS):
- API for MOM
- Topic-based
- Centralised or peer-to-peer implementations possible

Scribe:
- Topic-based
- Peer-to-peer architecture, based on Pastry (DHT)
- Topics have unique IDs and map onto nodes
- Multicast for sending events
  - Tree is built up as nodes subscribe
Distributed Objects

The Essential Distributed Objects Survival Guide

Robert Orfali • Dan Harkey • Jeri Edwards
CHALLENGES

→ Transparency
  • Failure transparency
→ Reliability
  • Dealing with *partial failures*
→ Scalability
  • Number of clients of an object
  • Distance between client and object
→ Design
  • Must take distributed nature into account from beginning
→ Performance
→ Flexibility
OBJECT MODEL

→ Classes and Objects
  
  **Class:** defines a type
  
  **Object:** instance of a class

→ Interfaces

→ Object references

→ Active vs Passive objects

→ Persistent vs Transient objects

→ Static vs Dynamic method invocation
Remote Object Architectural Model

Remote Objects:
- Single copy of object state (at single object server)
- All methods executed at single object server
- All clients access object through proxy
- Object’s location is location of state
Client

Client Process:
→ Binds to distributed object
→ Invokes methods on object

Proxy:
→ Proxy: RPC stub + destination details
→ Binding causes a proxy to be created
→ Responsible for marshaling
→ Static vs dynamic proxies
→ Usually generated

Run-Time System:
→ Provides services (translating references, etc.)
→ Send and receive
**OBJECT SERVER**

**Object:**
- State & Methods
- Implements a particular interface

**Skeleton:**
- Server stub
- Static vs dynamic skeletons

**Run-Time System:**
- Dispatches to appropriate object
- Invocation policies

**Object Server:**
- Hosts object implementations
- Transient vs Persistent objects
- Concurrent access
- Support legacy code
OBJECT REFERENCE

Local Reference:

⇒ Language reference to proxy
**Remote Reference:**

- Server address + object ID
Reference to proxy code (e.g., URL) & init data
→ Object name (human friendly, object ID, etc.)

What are the drawbacks and/or benefits of each approach?
Remote Method Invocation (RMI)

Standard invocation (synchronous):

- Client invokes method on proxy
- Proxy performs RPC to object server
- Skeleton at object server invokes method on object
- Object server may be required to create object first

Other invocations:

- Asynchronous invocations
- Persistent invocations
- Notifications and Callbacks
CORBA

Features:

- Object Management Group (OMG) Standard (version 3.1)
- Range of language mappings
- Transparency: Location & some migration transparency
- Invocation semantics: at-most-once semantics by default; maybe semantics can be selected
- Services: include support for naming, security, events, persistent storage, transactions, etc.
CORBA Architecture

Client machine

Client application
- Static IDL proxy
- Dynamic Invocation Interface
- ORB interface
- Client ORB
- Local OS

Server machine

Object implementation
- Object adapter
- Skeleton
- Dynamic Skeleton Interface
- ORB interface
- Server ORB
- Local OS

Network
Example: A Simple File System:

module CorbaFS {
    interface File;       // forward declaration

    interface FileSystem {
        exception CantOpen {string reason;};
        enum OpenMode {Read, Write, ReadWrite};
        File open (in string fname, in OpenMode mode)
            raises (CantOpen);
    };

    interface File {     // an open file
        string read (in long nchars);
        void   write (in string data);
        void   close ();
    };
};
**Object Reference (OR)**

Object Reference (OR):

- Refers to exactly one object, but an object can have multiple, distinct ORs
- ORs are implementation specific

**Interoperable Object Reference (IOR)**

- Can be shared between different implementations

![Diagram of Interoperable Object Reference (IOR) components]

- Repository identifier
- Profile ID
- Profile
- Interoperable Object Reference (IOR)
- IIOP version
- Host
- Port
- Object key
- Components
- POA identifier
- Other server-specific information
**OBJECT REQUEST BROKER (ORB)**

- Provides run-time system
- Translate between remote and local references
- Send and receive messages
- Maintains interface repository
- Enables dynamic invocation (client and server side)
- Locates services
INTERCEPTORS

Client application

Client proxy

Invocation request

Request-level interceptor

Message-level interceptor

Client ORB

Local OS

To server

To server
**BINDING**

**Direct Binding:**
- ➜ Create proxy
- ➜ ORB connects to server (using info from IOR)
- ➜ Invocation requests are sent over connection

**Indirect Binding:**

![Diagram of Indirect Binding]

1. First invocation or binding request
2. Activate/start object
3. Ack object is active
4. Redirect message
5. Actual invocation

IOR refers to implementation repository
CORBA Services

Some of the standardised services are the following:

- Naming Service
- Event Service
- Transaction Service
- Security Service
- Fault Tolerance
CORBA BIBLIOGRAPHY


Play with CORBA. Many implementations available, including ORBit: http://www.gnome.org/projects/ORBit2/
Distributed Shared Objects:

- Object state can be replicated (at multiple object servers)
- Object state can be partitioned
- Methods executed at some or all replicas
- Object location no longer clearly defined
Client

- Client has local representative (LR) in its address space
- Stateless LR
  - Equivalent to proxy
  - Methods executed remotely
- Stateful LR
  - Full state
  - Partial state
  - Methods (possibly) executed locally
**Object Server**

- Server dedicated to hosting LRs
- Provides resources (network, disk, etc.)
- Static vs Dynamic LR support
- Transient vs Persistent LRs
- Security mechanisms

**Location of LRs:**

- LRs only hosted by clients
- Statefull LRs only hosted by object servers
- Statefull LRs on both clients and object servers
GLOBE (GLOBAL OBJECT BASED ENVIRONMENT)

Scalable wide-area distributed system:

- Wide-area scalability requires replication
- Wide-area scalability requires flexibility

Features:

- Per-object replication and consistency
- Per-object communication
- Mechanism not policy
- Transparency (replication, migration)
- Dynamic replication
HOMEWORK

→ Could you turn CORBA into a distributed shared object middleware using interceptors?

Hacker’s edition:

→ Implement the simple filesystem presented using a freely available version of CORBA (or other middleware if you prefer).
READING LIST

Globe: A Wide-Area Distributed System  An overview of Globe

CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments  An overview of CORBA

New Features for CORBA 3.0  More CORBA