Slide 1

DISTRIBUTED SYSTEMS (COMP9243)

Lecture 9b: Middleware

① Introduction

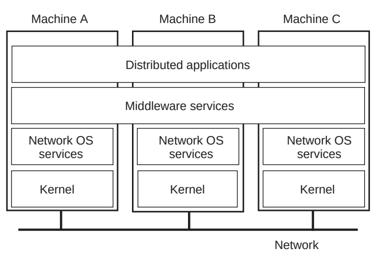
② Distributed Object Middleware

- Remote Objects & CORBA
- Distributed Shared Objects & Globe

③ Publish/Subscribe Middleware

Slide 2

MIDDLEWARE

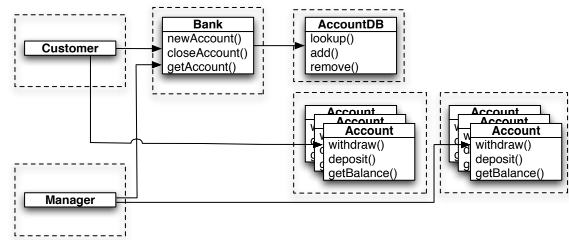


Slide 3

KINDS OF MIDDLEWARE

Distributed Object based:

➜ Objects invoke each other's methods

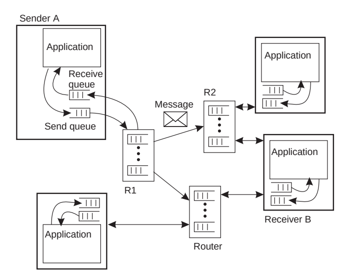


Slide 4

Message-oriented:

➜ Messages are sent between processes

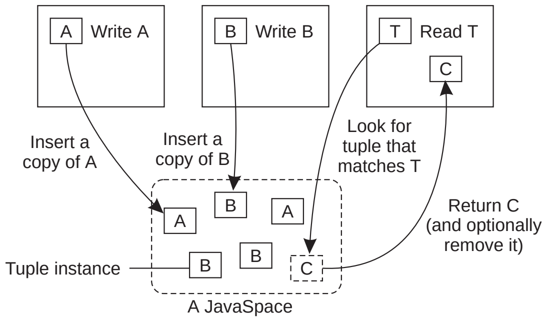
➜ Message queues



Slide 5

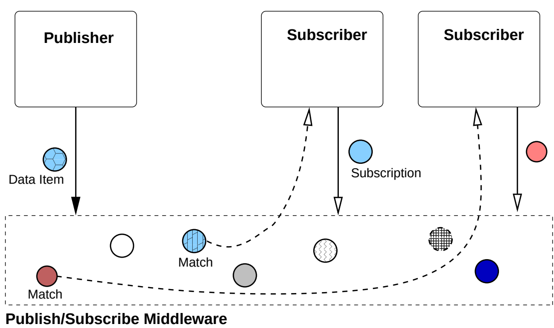
Coordination-based:

➜ Tuple space



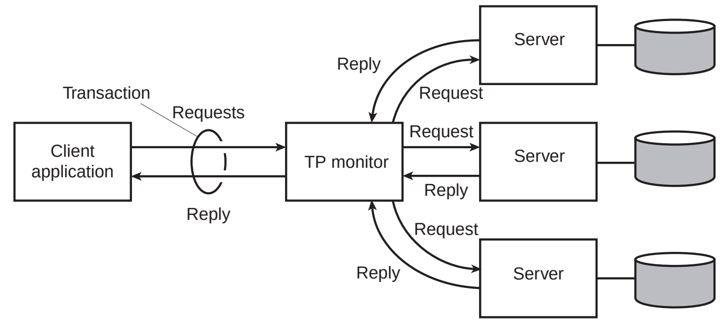
Slide 6

➜ Publish/Subscribe



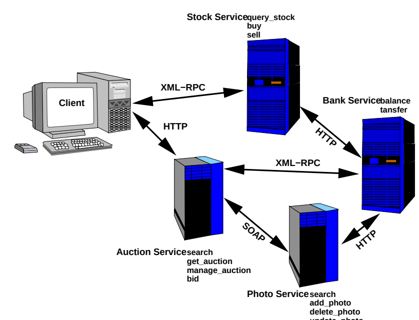
Slide 7

Transaction Processing Monitors:



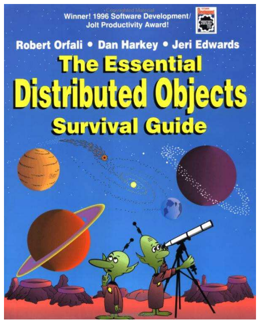
Slide 8

Web Services:



Slide 9

Distributed Objects



Slide 10

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

Slide 11

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

Slide 12

Interfaces

Define an object's methods

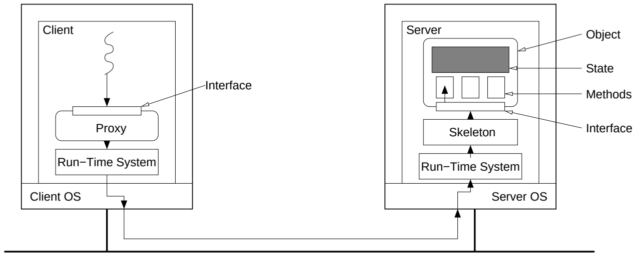
- ➜ Composition of interfaces
- ➜ Inheritance of interfaces
- ➜ Versions of interfaces

Interface Definition Language:

- ➜ Types of objects,
- ➜ Public methods,
- ➜ Exceptions that may occur
- ➜ etc.

Slide 13

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

Slide 14

Client

Client Process:

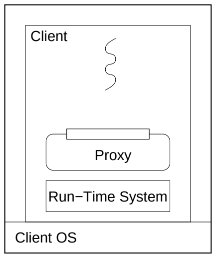
- ➜ Binds to distributed object
- ➜ Invokes methods on object

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



Slide 15

Object Server

Object:

- ➜ State & Methods
- ➜ Implements a particular interface

Skeleton:

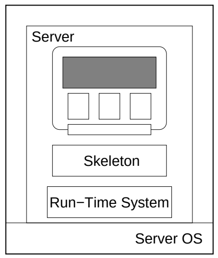
- ➜ Server stub
- ➜ Static vs dynamic skeletons

Run-Time System:

- ➜ Dispatches to appropriate object
- ➜ Invocation policies

Object Server:

- ➜ Hosts object implementations
- ➜ Transient vs Persistent objects
- ➜ Concurrent access
- ➜ Support legacy code

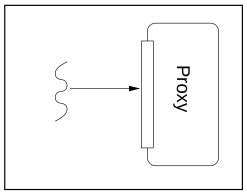


Slide 16

Object Reference

Local Reference:

- ➜ Language reference to proxy

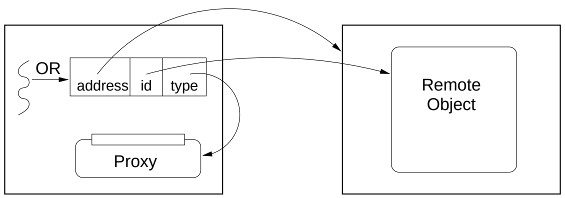


Slide 17

Object Reference

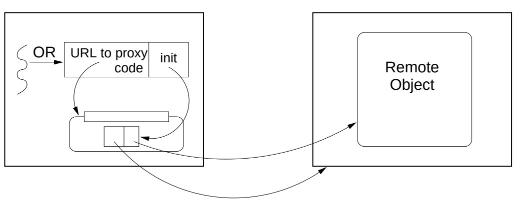
Remote Reference:

➜ Server address + object ID



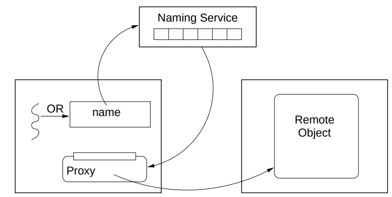
Slide 18

➜ Reference to proxy code (e.g., URL) & init data



Slide 19

➜ Object name (human friendly, object ID, etc.)



What are the drawbacks and/or benefits of each approach?

Slide 20

Binding and Name Resolution

Name Resolution:

➜ Name → remote reference

- Reference info contained in name (e.g., URL), or
- Naming service stores name to reference mappings

Binding:

➜ Remote reference → local reference

- Create a proxy
- Connect proxy to object server

Slide 21

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

Slide 22

Invocation Semantics

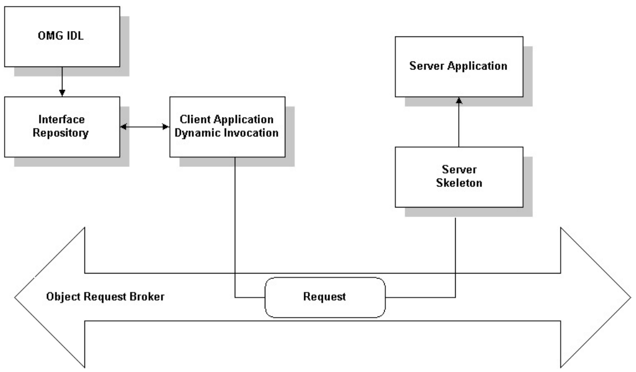
- ➜ Local method invocation: executed exactly once
- ➜ Problem: Cannot make such a guarantee for remote invocations!

Fault Tolerance Measure			
Retransmit Request	*Filter Duplicates*	*Re-execute, or Re-reply*	*Invocation Semantics*
No	n/a	n/a	Maybe
Yes	No	Re-execute procedure	At-least-once
Yes	Yes	Retransmit reply	At-most-once

What's the difference between Maybe and At-most-once?

Slide 23

Static vs Dynamic Invocation



Slide 24

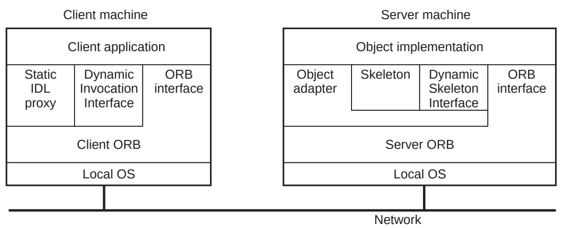
CORBA

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.

Slide 25

CORBA Architecture



Slide 26

Interfaces: OMG IDL

Components of an interface definition:

- ➜ Defines class attributes and methods
  (Classes are called interfaces and methods are called operations)
- ➜ Method arguments are annotated as `in`, `out`, and `inout`
- ➜ Errors/exceptions
- ➜ Interface inheritance

Characteristics of OMG IDL:

- ➜ Grammar is a subset of ANSI C++ plus additional constructs for object invocation
- ➜ No control structures, only declarations
- ➜ Primitive and complex data types, but no pointers
- ➜ No templates and no overloading
- ➜ Includes support for pre-processor

Slide 27

Example: A Simple File System:

```
module CorbaFS {
  interface File;       // forward declaration

  // a super-simple file system
  interface FileSystem {
    exception CantOpen {string reason;};
    enum OpenMode {Read, Write, ReadWrite};
    File open (in string fname, in OpenMode mode)
      raises (CantOpen);
  };

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

Slide 28

Object Reference (OR)

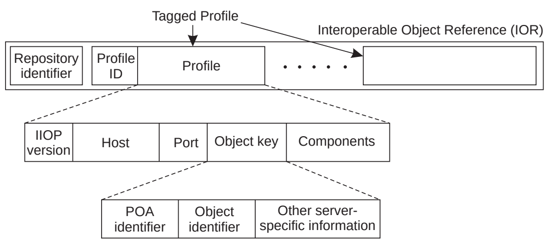
Object Reference (OR):

- ➜ Refers to exactly one object, but an object can have multiple, distinct handles
- ➜ A language mapping provides an opaque representation
- ➜ ORs are implementation specific

Slide 29

Interoperable Object Reference (IOR)

➜ Can be shared between different implementations



Slide 30

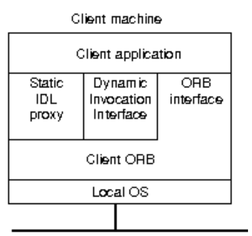
Client

➜ Binds to remote object
➜ Static proxy
➜ Dynamic invocation

Client-Side Dynamic Invocation:

➜ Dynamic Invocation interface (DII)
➜ Request is dynamically created
➜ Interface repository:

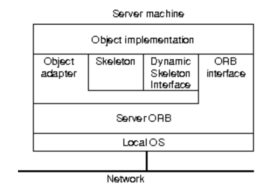
- Persistent storage of interface definitions
- Dynamic type checking and checking of inheritance graph
- Interface browser
- May be queried by language bindings



Slide 31

Object

➜ Also known as *Servant*
➜ Can be proper object (e.g., C++, Java)
➜ Can be state and procedures (e.g., C)
➜ Can be legacy code (with a wrapper)



Object Adapter:

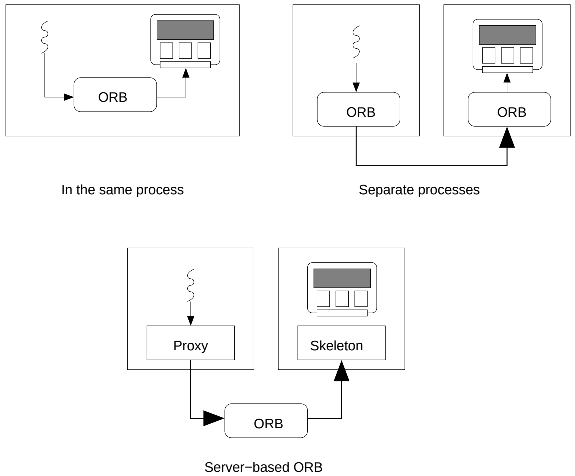
➜ Dispatcher (OR $\rightarrow$ Servant)
➜ Invokes skeleton (static or dynamic)
➜ Possibly creates objects
➜ Portable Object Adapter

Slide 32

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

Slide 33



Slide 34

CORBA Leaves Implementations a Lot of Freedom:

Advantages

- ➜ ORB implementations can more easily be optimised for size, speed, or functionality
- ➜ Wide applicability; allows extreme cases like use in real-time systems (TAOS)
- ➜ Future improvements in network & compiler technology can be exploited more easily
- ➜ Vendors can specialise

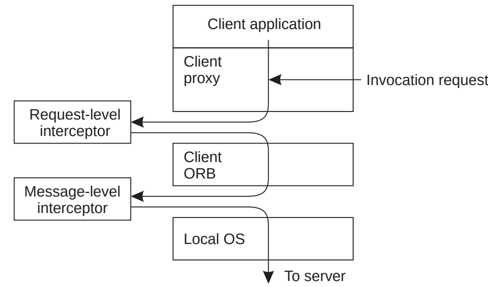
Disadvantages

- ➜ Complexity of the specification; more difficult to understand
- ➜ Incompatibility of different implementations

This is in contrast to COM/DCOM.

Slide 35

INTERCEPTORS



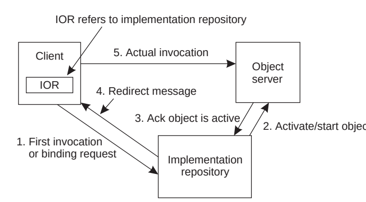
Slide 36

BINDING

Direct Binding:

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

Indirect Binding:



Slide 37

Communication: General Inter-ORB Protocol (GIOP)

- ➜ Protocol framework for communicating between ORBs
- ➜ Does not specify actual transport protocol

Components of the GIOP specification:

- ➜ Transfer syntax
- ➜ Message formats
- ➜ Transport layer assumptions

Goals of GIOP:

- ➜ Simplicity & ease of implementation
- ➜ Scalability
- ➜ Generality & architecture neutrality

Slide 38

Common Data Representation (CDR) ✧

- ➜ neutral, bi-canonical, on-the-wire representation
- ➜ variable byte addressing (receiver might swap)
- ➜ data aligned at natural boundaries (improves efficiency)
- ➜ covers all data types that can be expressed in OMG IDL

For example:

boolean: True; long: 132; short: 56; char: Q; long long: 234,663

```
Value   - 0x01 0x00 0x00 0x00 0x84 0x00 0x38 0x51 0x00 0x00
Address -    0    1    2    3    4    5    6    7    8    9
0x00 0x00 0x00 0x03 0x94 0xA7
  10   11   12   13   14   15

Value   - 0x01 0x20 0x20 0x20 0x00 0x00 0x00 0x84 0x00 0x38 0x51
Address -    0    1    2    3    4    5    6    7    8    9   10
0x20 0x20 0x20 0x20 0x20 0x00 0x00 0x00 0x00 0x00 0x03 0x94 0xA7
  11   12   13   14   15   16   17   18   19   20   21   22   23
```


Slide 39

Message Types

`Request` message::

- ➜ Invocation of an operation or use of attribute accessor
- ➜ Request header and request body

`Reply` message::

- ➜ Reply to a request (that requires a reply)
- ➜ Reply header and reply body

`CancelRequest` message::

- ➜ Request not needed anymore (advisory)
- ➜ Transfers the request ID

`LocateRequest` message::

- ➜ Check object validity and capabilities
- ➜ Object of interest

Slide 40

`LocateReply` message::

- ➜ Reply to locate request
- ➜ Object might be unknown, available, or forwarded

`CloseConnection` message::

- ➜ Server terminates connection

`MessageError` message::

- ➜ Error in message version or format

`Fragment` message::

- ➜ Follow up to incomplete message

Slide 41

Transport Layer (GIOP Requirements)

- ➜ Transport must be connection-oriented
- ➜ Transport must be reliable
- ➜ Unbounded stream of bytes
- ➜ Transport must notify in case of connection loss
- ➜ TCP's connection initiation model must be implementable

➠ TCP/IP fits very well

From GIOP to Internet Inter-Orb Protocol (IIOP):

A small step:

- ➜ IOR definition for TCP/IP
- ➜ Connection handling

Slide 42

CORBA Services

Some of the standardised services are the following:

- ➜ Naming Service
- ➜ Event Service
- ➜ Transaction Service
- ➜ Security Service
- ➜ Fault Tolerance

Slide 43

CORBA Bibliography

(1) *IIOP Complete*, W. Ruh, T. Herron, and P. Klinker, Addison Wesley, 1999.

(2) *The Common Object Request Broker: Architecture and Specification (2.3.1)*, Object Management Group, 1999.

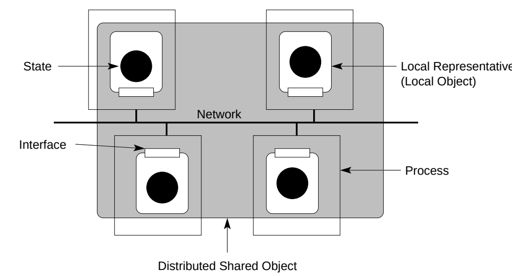
(3) *C Language Mapping Specification*, Object Management Group, 1999.

(4) *CORBAservices: Common Object Services Specification*, Object Management Group, 1998.

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

Slide 44

Distributed Shared Object (DSO) Model



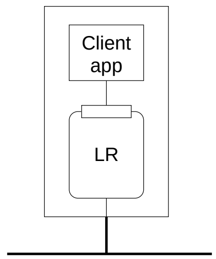
Distributed Shared Objects:

- ➜ 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

Slide 45

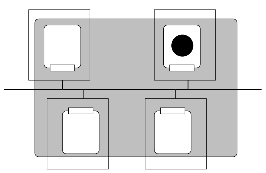
CLIENT

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

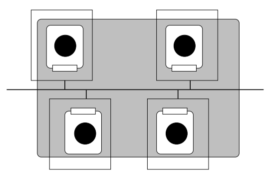


Slide 46

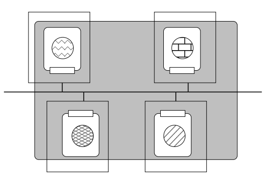
OBJECT



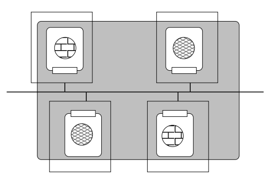
Remote Object



Replicated Object



Partitioned Object

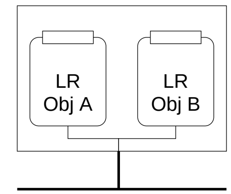


Replicated and Partitioned Object

Slide 47

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

Slide 48

GLOBE (GLOBAL OBJECT BASED ENVIRONMENT)

Scalable wide-area distributed system:

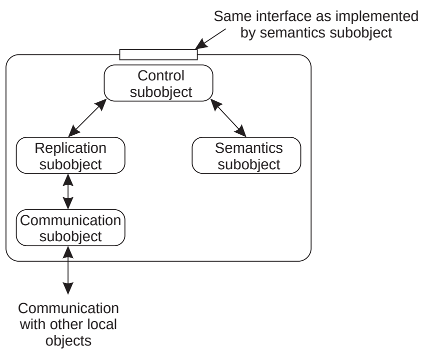
- ➜ Wide-area scalability requires replication
- ➜ Wide-area scalability requires flexibility

Features:

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

Slide 49

LOCAL REPRESENTATIVE



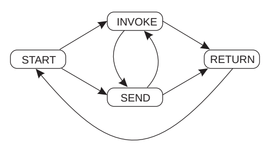
Slide 50

Semantics Subobject:

- ➜ Equivalent to CORBA Servant
- ➜ Implements object interface (Globe IDL)
- ➜ State stored in semantics subobject

Control Subobject:

- ➜ Implements standard *control interface*
- ➜ Mediates between replication and semantics subobjects
- ➜ Does marshaling/unmarshaling

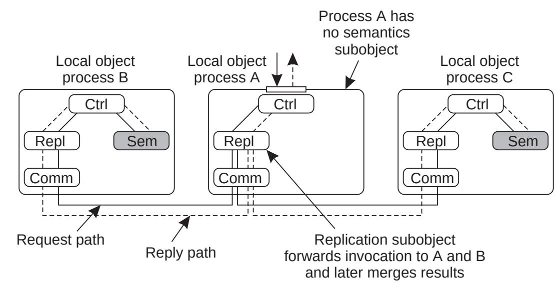


Replication Subobject:

- ➜ Implements standard *replication interface*: `start()`, `send()`, `invoked()`
- ➜ Responsible for replication and consistency
- ➜ Implementation independent of semantics subobject

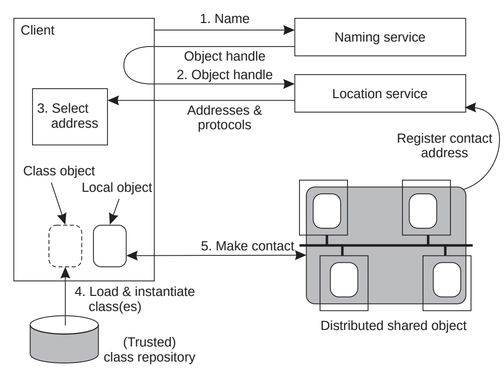
Slide 51

REPLICATION EXAMPLE: ACTIVE REPLICATION



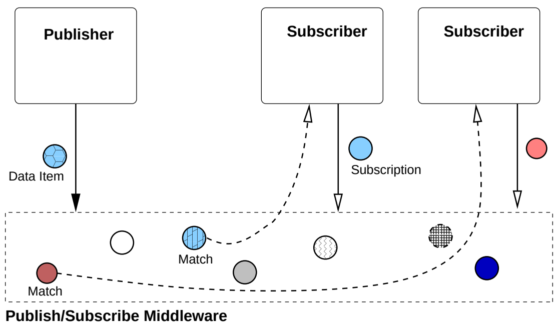
Slide 52

BINDING



Slide 53

Publish/Subscribe (Event-Based) Middleware



Slide 54

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

Slide 55

Examples

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

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

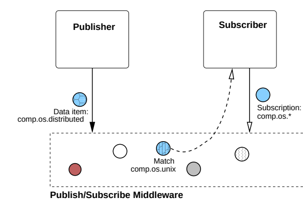
Network Monitoring:
- ➜ Status logged by routers, servers
- ➜ Monitors screen for failures, intrusion attempts

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

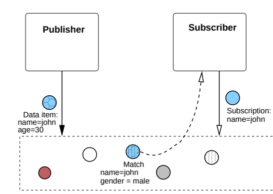
Slide 56

Message Filtering

Topic-based



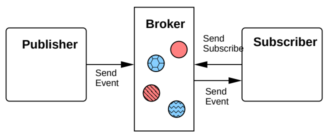
Content-based



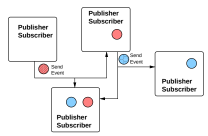
Slide 57

Architecture

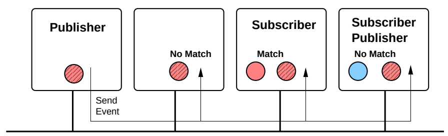
Centralised:



Peer-to-Peer:



Multicast-based:



Slide 58

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

Slide 59

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

Slide 60

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

Slide 61

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

Reading List

Slide 62

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