

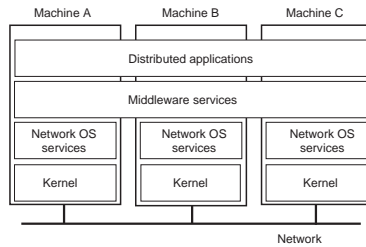
DISTRIBUTED SYSTEMS (COMP9243)

Lecture 8: Middleware

Slide 1

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

Slide 2

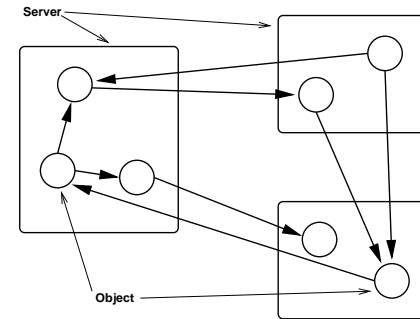


KINDS OF MIDDLEWARE

Distributed Object based:

→ Objects invoke each other's methods

Slide 3

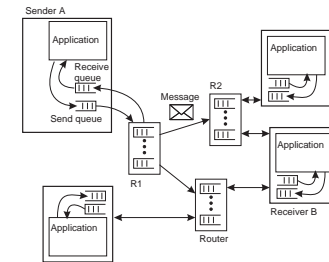


Message-oriented:

→ Messages are sent between processes

→ Message queues

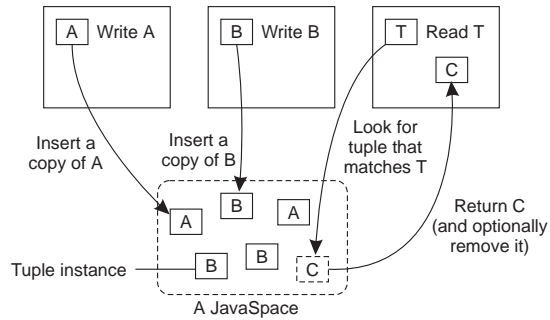
Slide 4



Coordination-based:

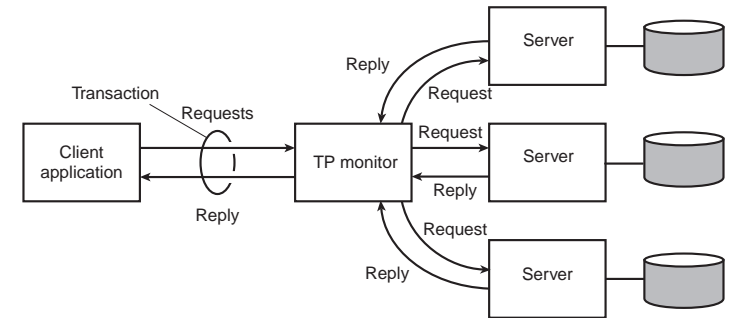
→ Tuple space

Slide 5



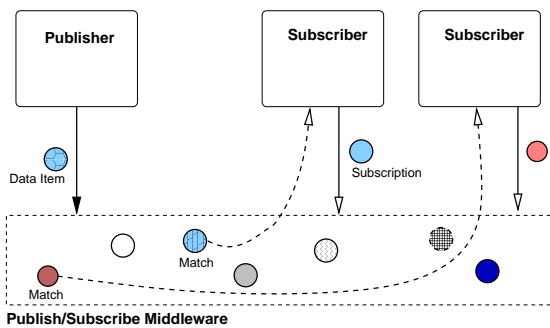
Transaction Processing Monitors:

Slide 7



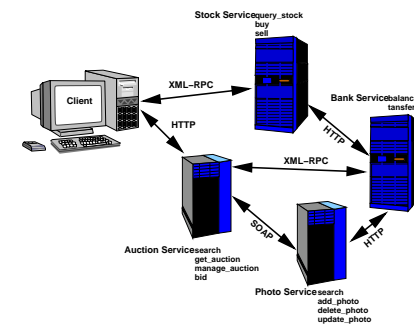
→ Publish/Subscribe

Slide 6

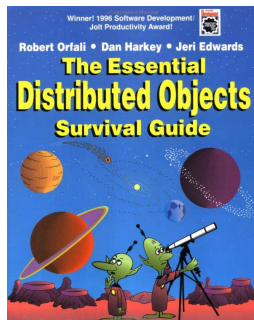


Web Services:

Slide 8



DISTRIBUTED OBJECTS



Slide 9

CHALLENGES

Slide 10

- 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

Slide 11

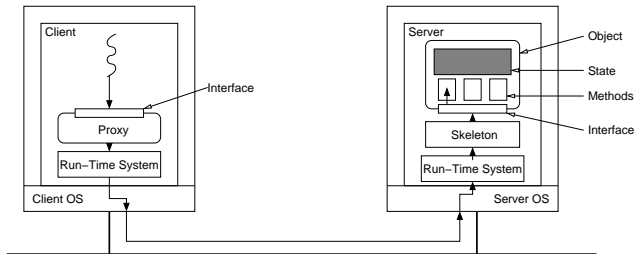
- Classes and Objects
 - Class:** defines a type
 - Object:** instance of a class
- Interfaces
- Object references
- Active vs Passive objects
- Persistent vs Transient objects
- Static vs Dynamic method invocation

INTERFACES

Slide 12

- 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.

REMOTE OBJECT ARCHITECTURAL MODEL



Slide 13

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

OBJECT SERVER ◇

Object:

- State & Methods
- Implements a particular interface

Skeleton:

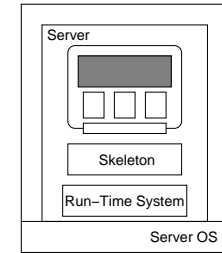
- Server stub
- Static vs dynamic skeletons

Run-Time System:

- Dispatches to appropriate object
- Invocation policies

Object Server:

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



Slide 15

CLIENT ◇

Client Process:

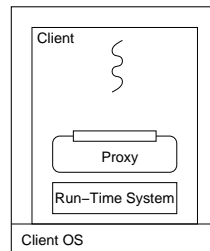
- Binds to distributed object
- Invokes methods on object

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



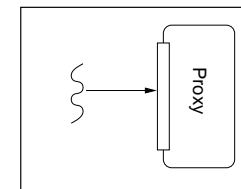
Slide 14

OBJECT REFERENCE

Local Reference:

- Language reference to proxy

Slide 16

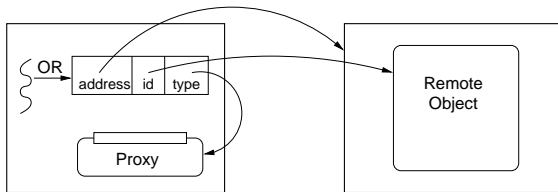


OBJECT REFERENCE

Remote Reference:

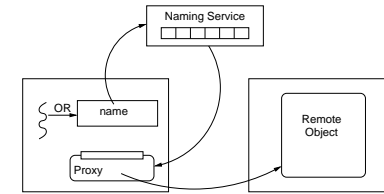
→ Server address + object ID

Slide 17



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

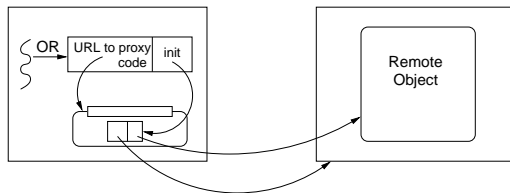
Slide 19



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

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

Slide 18



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

Slide 20

Binding:

→ Remote reference → local reference

- Create a proxy
- Connect proxy to object server

REMOTE METHOD INVOCATION (RMI)

Slide 21

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

INVOCATION SEMANTICS

Slide 22

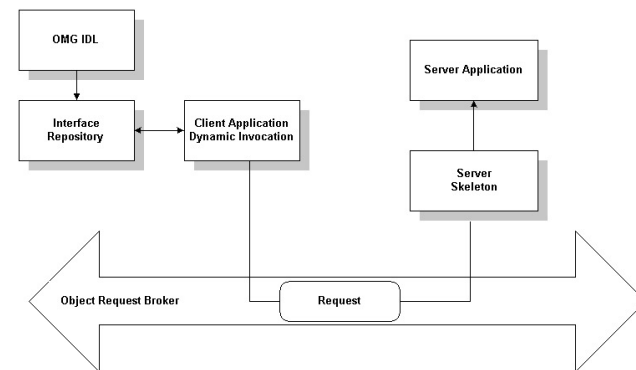
- Local method invocation: executed **exactly once**
- **Problem:** Cannot make such a guarantee for remote invocations!

	Fault Tolerance Measure			Invocation Semantics
	Retransmit Request	Filter Duplicates	Re-execute, or Re-reply	
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**?

STATIC VS DYNAMIC INVOCATION

Slide 23



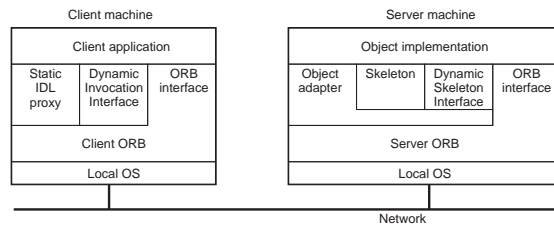
CORBA

Slide 24

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



Slide 25

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 27

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

Slide 26

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

ELEMENTS OF IDL ✧

Basic types::

- The usual short to long long types (size fixed)
- Latin 1 & wide characters
- boolean & octet
- any type

Constructed types::

- C/C++-style enumerations & structures

Other types::

- Sequence types—e.g., `sequence<octet>`
- Strings, Arrays

Exceptions:

- Standard exceptions
- User-defined exceptions: struct-like data type

Slide 28

Slide 29

Modules & Interfaces: ◇

- An interface defines the **public interface** of a CORBA object
- Interfaces can be defined using (multiple) inheritance
- A module bundles related declarations and interfaces

```
module Foo {
    typedef long bar;
    interface A {
        enum baz {...};
        ...
    };
    interface B {...};
    interface C: A, B {...};
};
```

- Valid names: `Foo::bar`, `Foo::A::baz`, and `Foo::C::baz`

Slide 30

Operations & Attributes:

- All **operations** contained in the interface of an object can be invoked on that object

```
interface Echo {
    void doEcho (in string msg) raises(EchoException);
};
```

- oneway operations
- **Attributes** specify publicly accessible data items

```
interface Counter {
    readonly attribute long value;
    void inc ();
};
```

- **readonly** attributes

LANGUAGE MAPPINGS

Accessing interfaces from "real" programming languages?:

- Standardised language mappings define
 - Host language representation for basic and constructed IDL types,
 - References to constants and objects,
 - Mechanisms for invoking operations,
 - Behaviour in case of an exceptions, and
 - Signatures for standard operations like those of the object adapters.
- Ideally, client and (to a lesser extent) object implementation code should be portable across different IDL compilers and ORBs.
- **Note:** Defines only how an IDL interface appears in a given language, but not how it is implemented (e.g., OR).

Slide 31

But C does not have: ◇

- objects,
- inheritance,
- exceptions, and
- nested name spaces.

Slide 32

The C language mapping:

- Objects: each object has type `CORBA_Object`
- Operations: function gets object as additional first argument
- Inheritance: method vectors and object adapter
- Exceptions: each operation gets a reference to `CORBA_Environment` as additional last argument
- Poor man's name spaces: `::` is replaced by `_`

Slide 33

```
interface Echo {  
    void doEcho (in string msg);  
};
```

↓

```
typedef CORBA_Object Echo;  
void Echo_doEcho (Echo _obj,  
                 CORBA_char *msg,  
                 CORBA_Environment *ev);
```

Slide 34

```
interface Counter {  
    readonly attribute long value;  
    void inc ();  
};
```

↓

```
typedef CORBA_Object Counter;  
CORBA_long Counter__get_value (Counter _obj,  
                               CORBA_Environment *ev);  
void Counter_inc (Counter _obj,  
                 CORBA_Environment *ev);
```

→ Note: No definition for the attribute itself, only the _get_value operation.

OBJECT REFERENCE (OR)

Object Reference (OR):

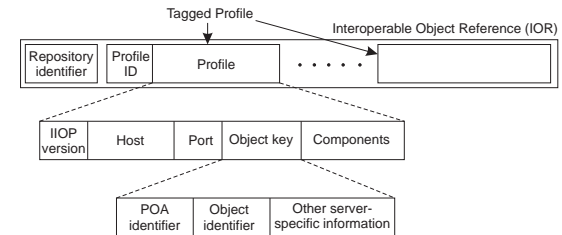
Slide 35

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

Interoperable Object Reference (IOR)

- Can be shared between different implementations

Slide 36

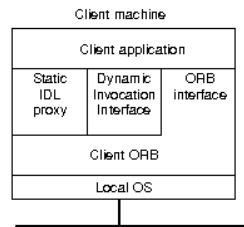


CLIENT

- 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



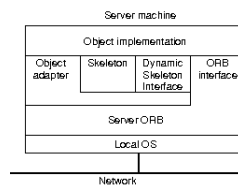
Slide 37

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 → Servant)
- Invokes skeleton (static or dynamic)
- Possibly creates objects
- Portable Object Adapter

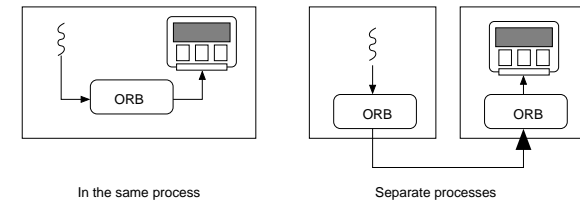


Slide 38

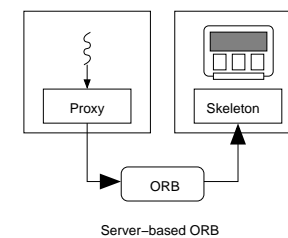
OBJECT REQUEST BROKER (ORB)

- 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 39



Slide 40



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

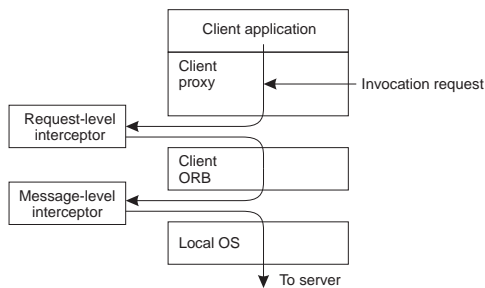
Slide 41

Disadvantages

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

This is in contrast to COM/DCOM.

INTERCEPTORS



Slide 42

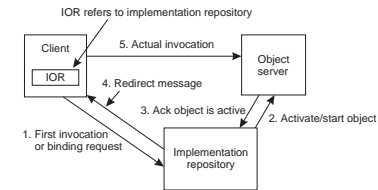
BINDING

Direct Binding:

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

Indirect Binding:

Slide 43



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

Slide 44

Goals of GIOP:

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

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

Slide 45

```
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 0x03 0x94 0xA7
11 12 13 14 15 16 17 18 19 20 21 22 23
```

GIOP MESSAGES

GIOP message header (1.1):

```
module GIOP {
    struct Version {octet major; octet minor;};
    enum MsgType_1_1 {
        Request, Reply, CancelRequest, LocateRequest, LocateReply,
        CloseConnection, MessageError, Fragment
    };
    struct MessageHeader_1_1 {
        char magic[4];
        Version GIOP_version;
        octet flags; // byte order, last fragment
        octet message_type;
        unsigned long message_size;
    };
};
```

Slide 46

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

Slide 47

CancelRequest message::

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

LocateRequest message::

- Check object validity and capabilities
 - Object of interest
-

LocateReply message::

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

CloseConnection message::

- Server terminates connection

Slide 48

MessageError message::

- Error in message version or format

Fragment message::

- Follow up to incomplete message
-

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

Slide 49

- TCP/IP fits very well

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

A small step:

- IOR definition for TCP/IP
- Connection handling

CORBA SERVICES

Some of the standardised services are the following:

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

Slide 50

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.

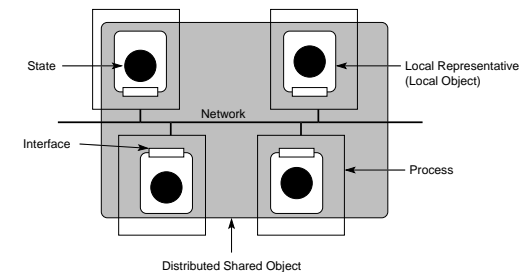
(3) *C Language Mapping Specification*, Object Management Group, 1999.

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

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

Slide 51

DISTRIBUTED SHARED OBJECT (DSO) MODEL



Slide 52

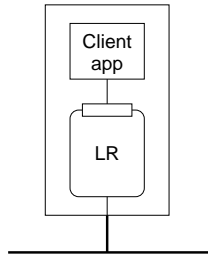
Distributed Shared Objects:

- 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 53

CLIENT

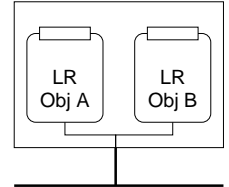
- Client has local representative (LR) in its address space
- Stateless LR
 - Equivalent to proxy
 - Methods executed remotely
- Statefull LR
 - Full state
 - Partial state
 - Methods (possibly) executed locally



Slide 55

OBJECT SERVER

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

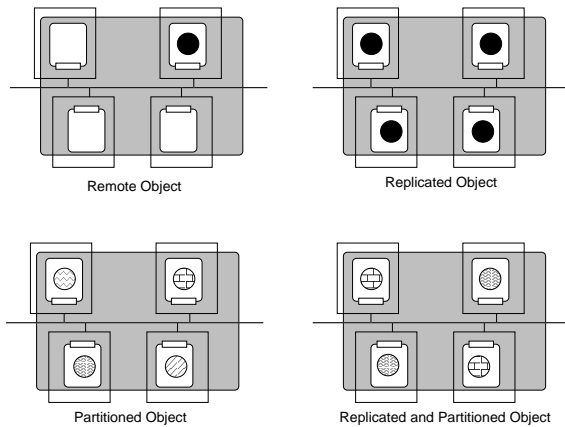


Location of LRs:

- LRs only hosted by clients
- Statefull LRs only hosted by object servers
- Statefull LRs on both clients and object servers

Slide 54

OBJECT



Slide 56

GLOBE (GLOBAL OBJECT BASED ENVIRONMENT)

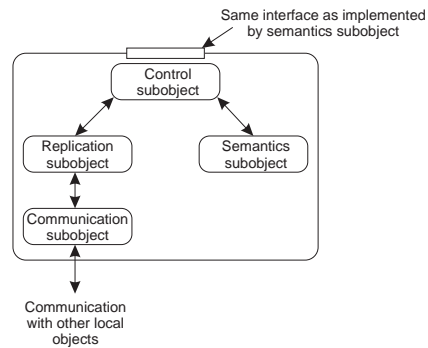
Scalable wide-area distributed system:

- 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

LOCAL REPRESENTATIVE



Slide 57

SEMANTICS SUBOBJECT

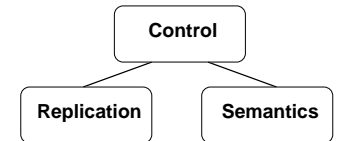
- Equivalent to CORBA Servant
- Implements object interface (Globe IDL)
- State stored in semantics subobject
- Written as though non-distributed (in theory!)
- Problems to look out for:
 - Concurrent access
 - No references to other Globe objects
 - No complex parameters (lists, other Globe objects, etc.)
 - No assumptions about local environment

Slide 58

REPLICATION AND CONTROL SUBOBJECTS

Control Subobject:

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



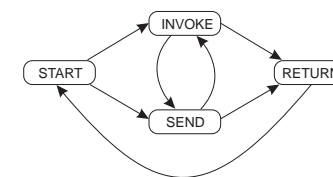
Slide 59

Replication Subobject:

- Implements standard *replication interface*
- Responsible for replication and consistency
- Decides how to handle local invocations and incoming requests
- Implementation independent of semantics subobject

CONTROL-REPLICATION SUBOBJECT INTERACTION

Control Subobject:



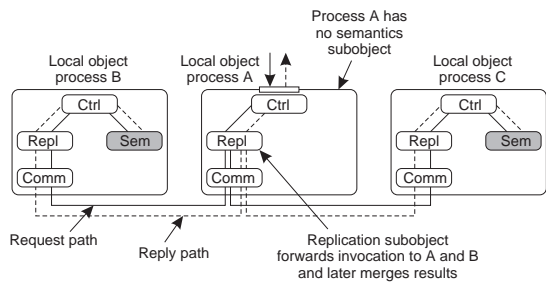
Slide 60

Replication Subobject:

- Read methods
- Write methods
- Replication interface: `start()`, `send()`, `invoked()`

REPLICATION EXAMPLE: ACTIVE REPLICATION

Slide 61

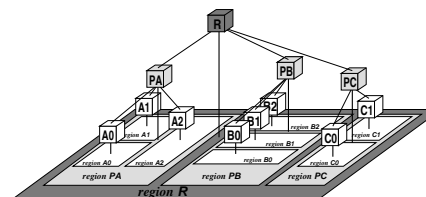


NAMING AND LOCATION SERVICES

Location Service:

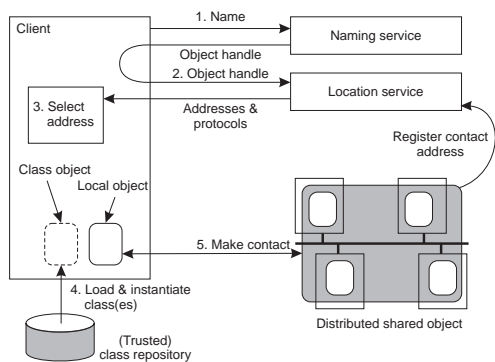
- Object handle → Contact Address
- Contact Address: Contact Point + LR implementation info
- Contact Point: Address
- Hierarchy of regions
- Pointer caching

Slide 63



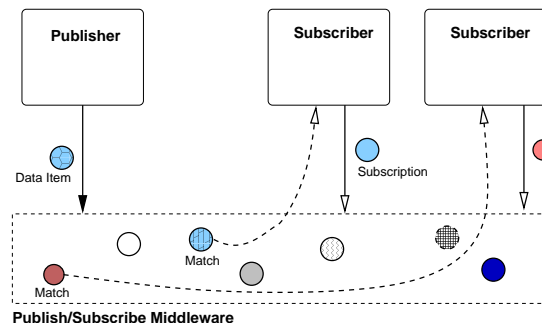
BINDING

Slide 62



PUBLISH/SUBSCRIBE (EVENT-BASED) MIDDLEWARE

Slide 64



CHALLENGES

Transparency:

- loose coupling → good transparency

Scalability:

- Potentially good due to loose coupling
- ✗ In practice hard to achieve
- Number of subscriptions
- Number of messages

Slide 65

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

Slide 66

Network Monitoring:

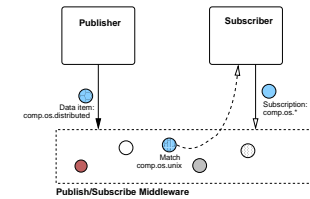
- Status logged by routers, servers
- Monitors screen for failures, intrusion attempts

Enterprise Application Integration:

- Independent applications
- Produce output as events
- Consume events as input
- Decoupled

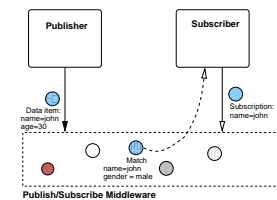
MESSAGE FILTERING

Topic-based



Slide 67

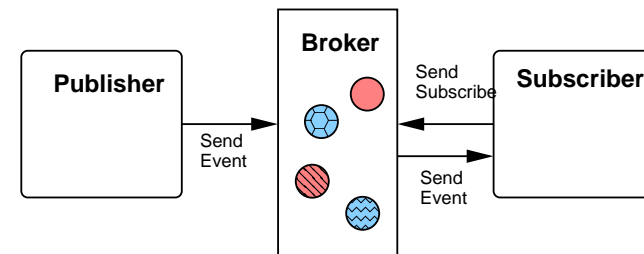
Content-based



ARCHITECTURE

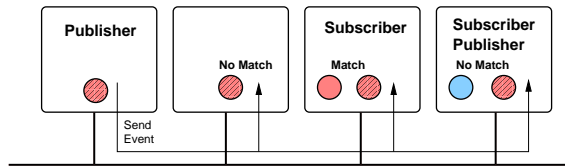
Centralised:

Slide 68



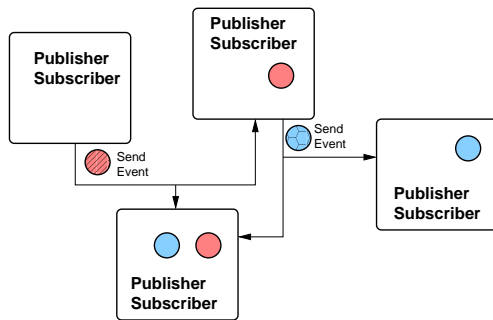
Slide 69

Multicast-based:



Slide 70

Peer-to-Peer:



COMMUNICATION

- Point-to-point
- Multicast

Slide 71

- 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

Slide 72

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
-

Slide 73

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
-

Slide 74

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

Slide 75