

DISTRIBUTED SYSTEMS (COMP9243)

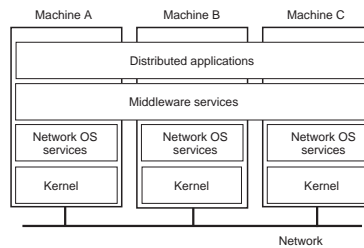
Lecture 4: Middleware

Slide 1

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

Slide 2

MIDDLEWARE



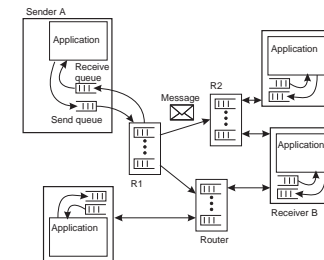
KINDS OF MIDDLEWARE

Slide 3

- Distributed Object based:
- Objects invoke each other's methods

Slide 4

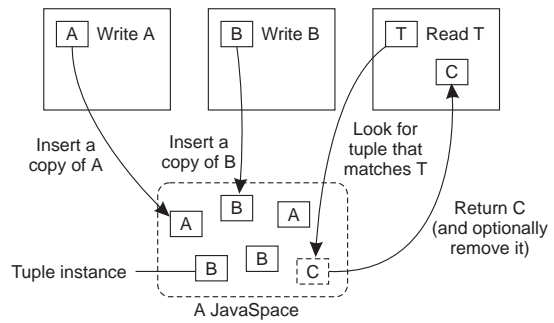
- Message-oriented:
- Messages are sent between processes
 - Message queues



Coordination-based:

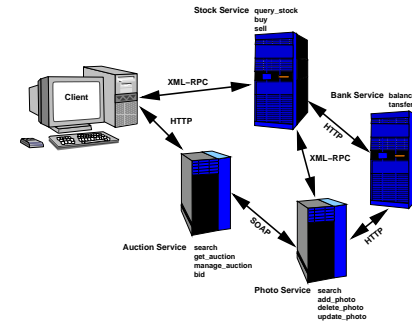
- Tuple space
- Publish/Subscribe

Slide 5



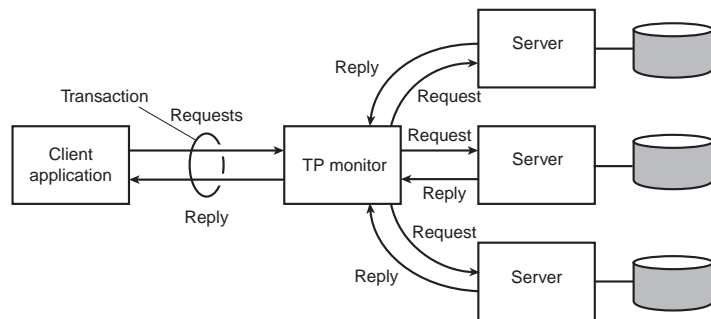
Web Services:

Slide 7



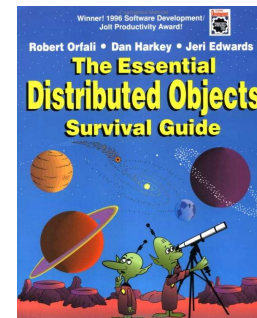
Transaction Processing Monitors:

Slide 6



DISTRIBUTED OBJECTS

Slide 8



CHALLENGES

Slide 9

- 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 10

- 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 11

Define an object's methods

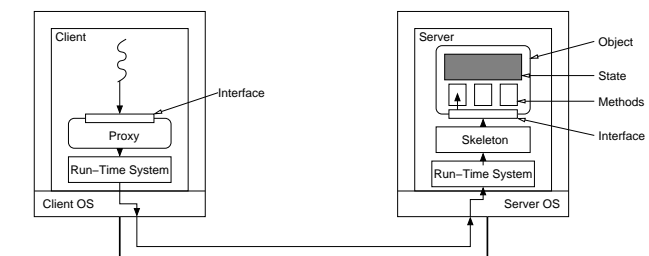
- Composition of interfaces
- Inheritance of interfaces
- Versions of interfaces

Interface Definition Language:

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

REMOTE OBJECT ARCHITECTURAL MODEL

Slide 12



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:

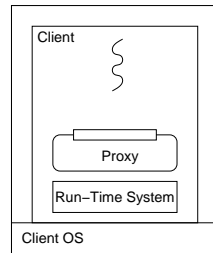
- Binds to distributed object
- Invokes methods on object

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



Slide 13

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 (where to get the proxy code?)
 - Connect proxy to object server

Slide 15

OBJECT SERVER

Object:

- State & Methods
- Implements a particular interface

Skeleton:

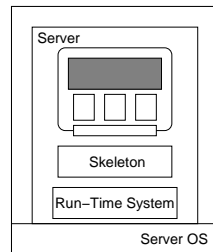
- Server stub
- Static vs dynamic skeletons

Run-Time System:

- Dispatches to appropriate object
- Invocation policies

Object Server:

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

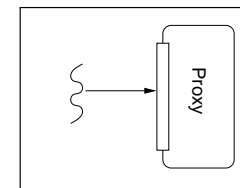


Slide 14

OBJECT REFERENCE

Local Reference:

- Language reference to proxy



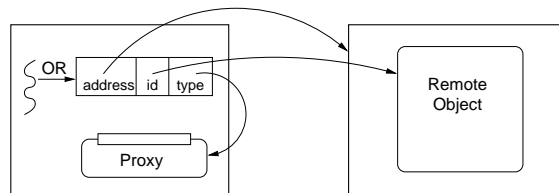
Slide 16

OBJECT REFERENCE

Remote Reference:

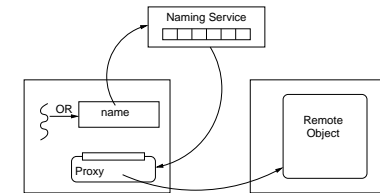
→ Server address + object ID

Slide 17



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

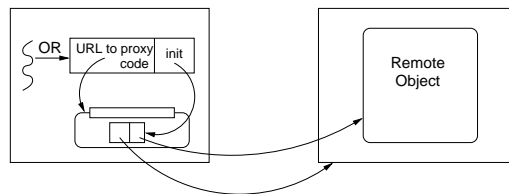
Slide 19



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

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

Slide 18



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

Slide 20

Other invocations:

- Asynchronous invocations
- Persistent invocations
- Events and Callbacks

INVOCATION SEMANTICS

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

Slide 21

Retransmit Request	Fault Tolerance Measure		Invocation Semantics
	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

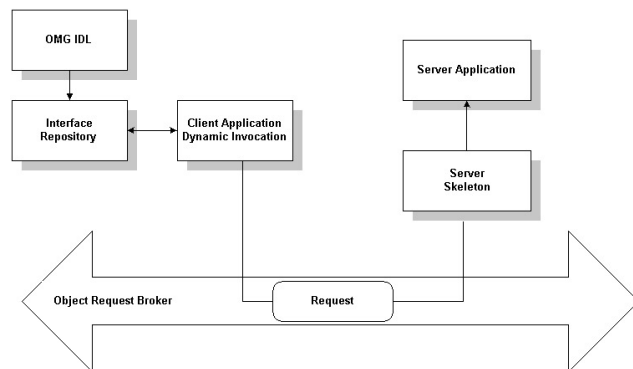
CORBA

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.

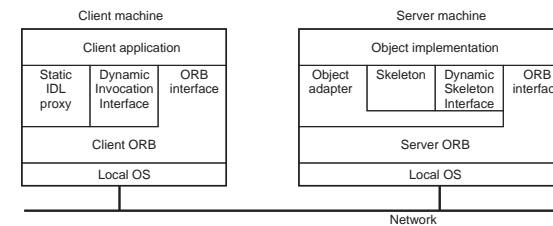
Slide 23

STATIC VS DYNAMIC INVOCATION



Slide 22

CORBA ARCHITECTURE



Slide 24

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 25

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

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 26

ELEMENTS OF IDL

Basic types::

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

Slide 27

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

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 {...};
};
```

Slide 28

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

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);
};
```

Slide 29

- 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.

Slide 30

- 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).
-

But C does not have:

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

Slide 31

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 `_`
-

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

⇓

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

Slide 32

```
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);
```

Slide 33

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

OBJECT REFERENCE (OR)

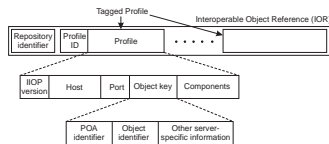
Object Reference (OR):

- 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 34

Interoperable Object Reference (IOR)

- Can be shared between different implementations



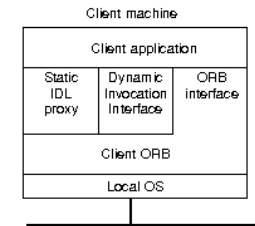
CLIENT

- 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

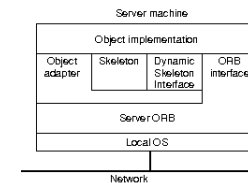
Slide 35



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)

Slide 36



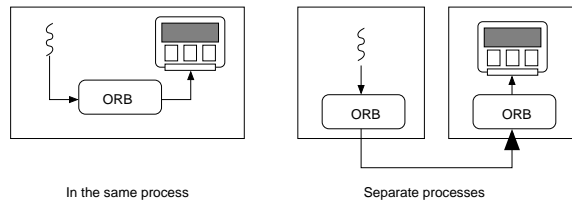
Object Adapter:

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

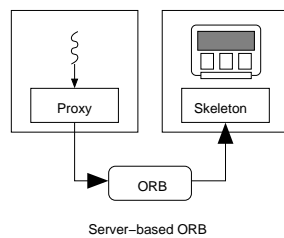
OBJECT REQUEST BROKER (ORB)

Slide 37

- 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 38



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 39

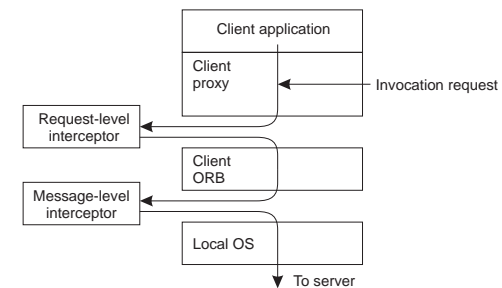
Disadvantages

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

This is in contrast to COM/DCOM.

INTERCEPTORS

Slide 40



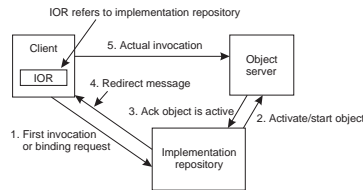
BINDING

Direct Binding:

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

Indirect Binding:

Slide 41



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 42

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 43

```
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 44

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
-

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 45

Slide 46

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 47

Slide 48

CORBA BIBLIOGRAPHY

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

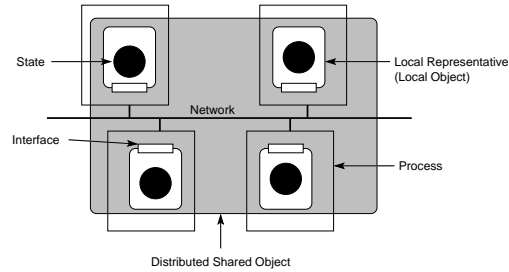
(2) *The Common Object Request Broker: Architecture and Specification (2.3.1)*, Object Management Group, 1999.

(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/>

DISTRIBUTED SHARED OBJECT (DSO) MODEL



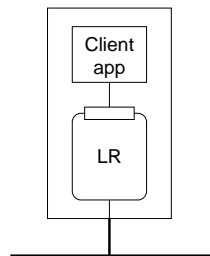
Slide 49

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

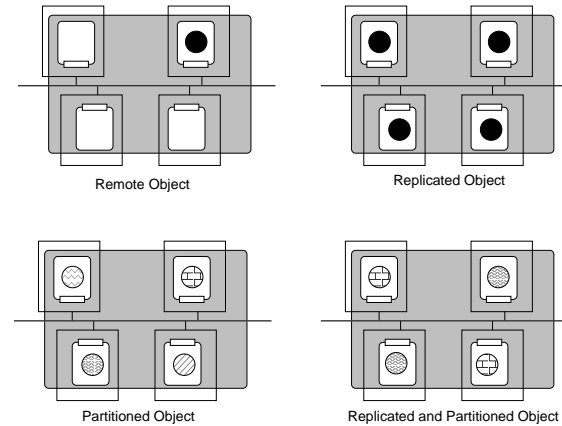
CLIENT

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



Slide 50

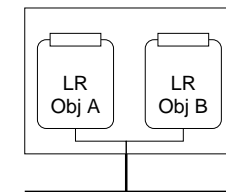
OBJECT



Slide 51

OBJECT SERVER

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



Slide 52

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)

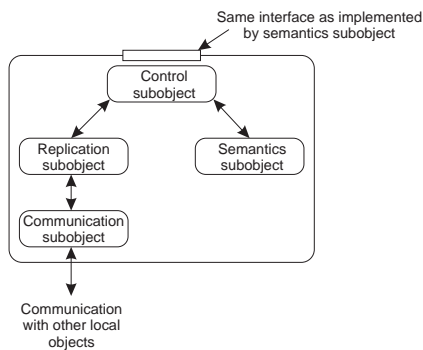
Scalable wide-area distributed system:

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

Slide 53 Features:

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

LOCAL REPRESENTATIVE



Slide 54

SEMANTICS SUBOBJECT

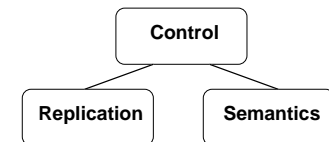
- Equivalent to CORBA Servant
- Implements object interface (Globe IDL)
- State stored in semantics subobject
- Written as though non-distributed (*in theory!*)
- Problems to look out for:
 - Concurrent access
 - No references to other Globe objects
 - No complex parameters (lists, other Globe objects,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

Slide 55

REPLICATION AND CONTROL SUBOBJECTS

Control Subobject:

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



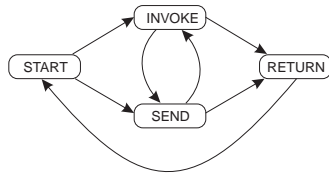
Slide 56

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:

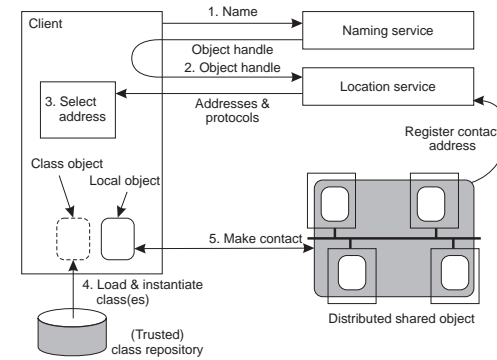


Slide 57

Replication Subobject:

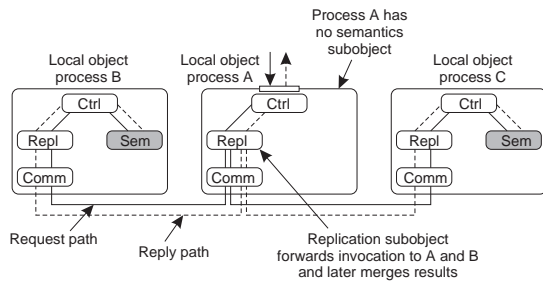
- Read methods
- Write methods
- Replication interface: `start()`, `send()`, `invoked()`

BINDING



Slide 59

REPLICATION EXAMPLE: ACTIVE REPLICATION



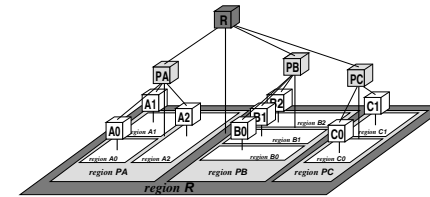
Slide 58

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 60



Slide 61

COORDINATION-BASED MIDDLEWARE

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

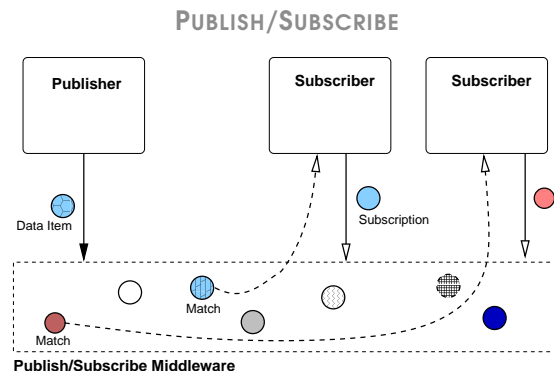
→ Policy separate from mechanism

Programmability:

→ Inherent distributed design

→ Doesn't use non-distributed concepts

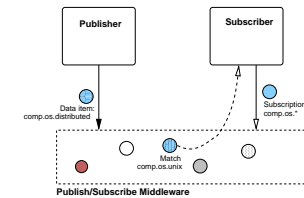
Slide 62



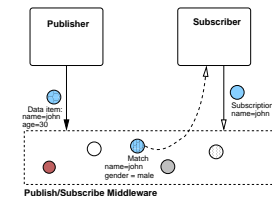
Slide 64

MESSAGE FILTERING

Topic-based



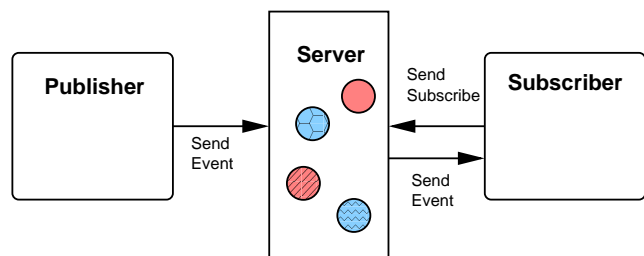
Content-based



ARCHITECTURE

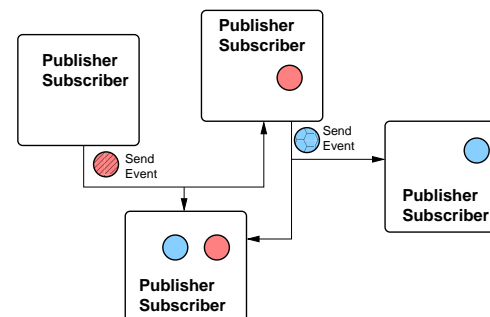
Centralised:

Slide 65



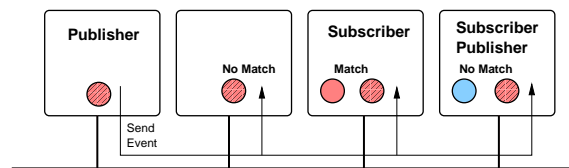
Peer-to-Peer:

Slide 67



Multicast-based:

Slide 66



COMMUNICATION

Slide 68

- 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

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 69