Slide 1

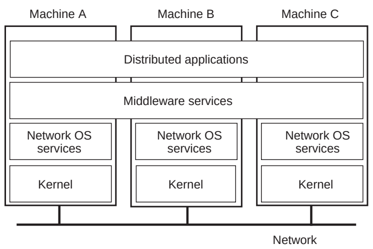
# DISTRIBUTED SYSTEMS (COMP9243)

## Lecture 9: Middleware

① Introduction
② Publish/Subscribe Middleware
③ Map-Reduce Middleware
④ Distributed Object Middleware
- Remote Objects & CORBA
- Distributed Shared Objects & Globe

Slide 2

## MIDDLEWARE

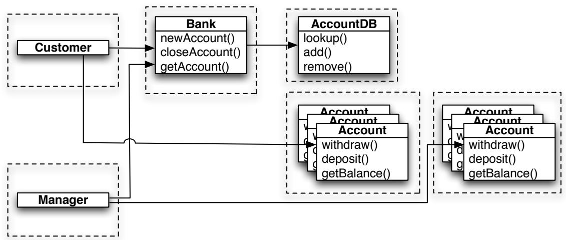



Slide 3

## KINDS OF MIDDLEWARE

Distributed Object based:

➜ Objects invoke each other's methods

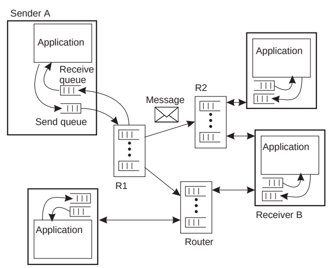



Slide 4

Message-oriented:

➜ Messages are sent between processes

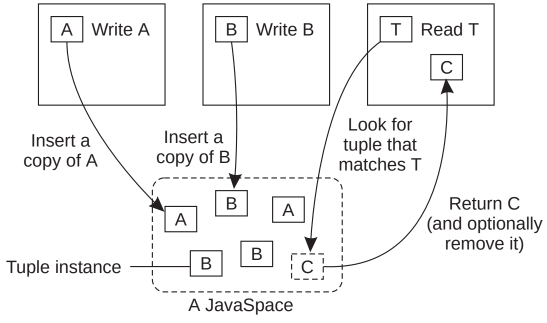
➜ Message queues



**Slide 5**

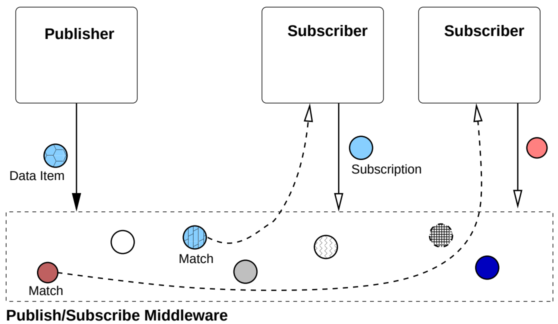
## Coordination-based:

- ➜ Tuple space




**Slide 6**

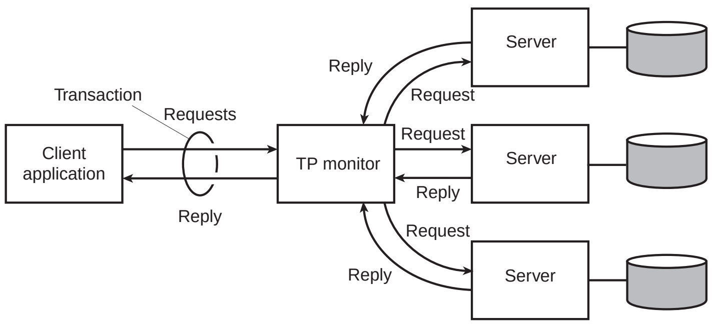
- ➜ Publish/Subscribe





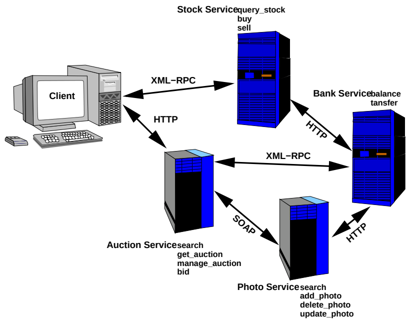

**Slide 7**

## Transaction Processing Monitors:



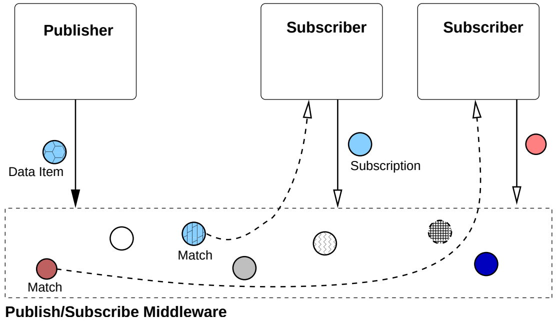

**Slide 8**

## Web Services:



Slide 9

## Publish/Subscribe (Event-Based) Middleware




Slide 10

## 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 11

## 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

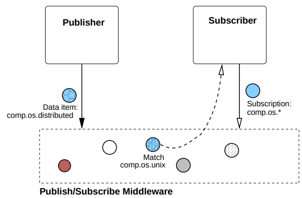
Enterprise Application Integration:

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

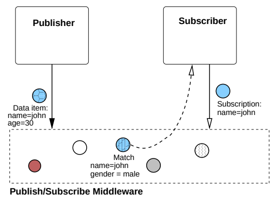
Slide 12

## Message Filtering

Topic-based



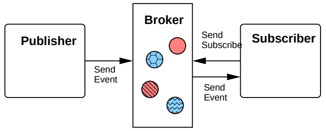

Content-based



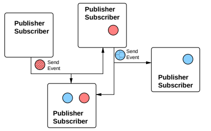
Slide 13

## Architecture

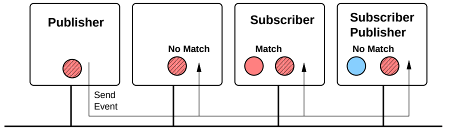
Centralised:




Peer-to-Peer:



Multicast-based:




Slide 14

## 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 15

## 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 16

## 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 17

## Example Systems

TIB/Rendezvous:

- ➜ Topic-based
- ➜ Multicast-based

Java Message Service (JMS):

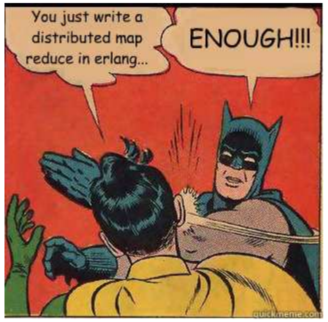
- ➜ API for MOM
- ➜ Topic-based
- ➜ centralised or peer-to-peer implementations possible

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 18

## Map-Reduce






Slide 19

## Context

Computations conceptually straightforward, but:

- ➜ Input data is usually large
- ➜ Need to finish in reasonable time
- ➜ Computations widely distributed (thousands of machines)

How to:

- ➜ Parallelize the computation?
- ➜ Distribute the data?
- ➜ Handle failures?
- ➜ Balance the load?

Slide 20

## Solution

Map-Reduce:

- ➜ New abstraction for simple computations.
- ➜ Hide dirty details.
- ➜ Based on *map* and *reduce* primitives from Lisp (functional language).

Basic computation:

- ➜ Takes set of input <key, value> pairs
- ➜ Produces set of output <key, value> pairs

Implementation:

- ➜ Google's version: *MapReduce*
- ➜ Open source version: *Hadoop*

Slide 21

User supplied functions:

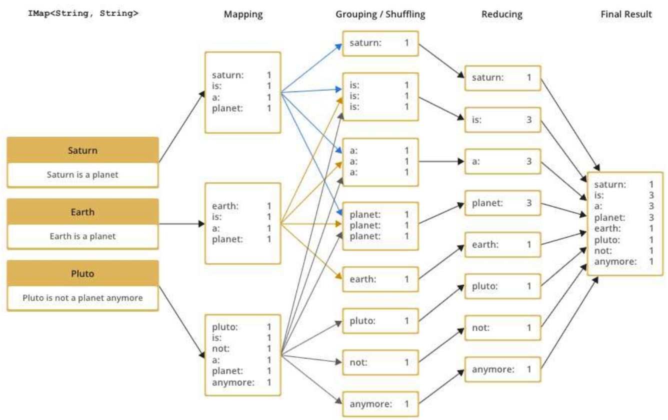
- ➜ **Map** Accepts: one input pair <key, value>
  Produces: a set of intermediate <key, value> pairs
- ➜ System groups intermediate values with same key together.
- ➜ **Reduce** Accepts: intermediate key, set of values for that key
  Produces: output list (typically small)

More formally:

- ➜ $map(k_1, v_1) \to list(k_2, v_2)$
- ➜ $reduce(k_2, list(v_2)) \to list(v2)$

Slide 22

## Example: Word Count




Slide 23

## Example: Word Count

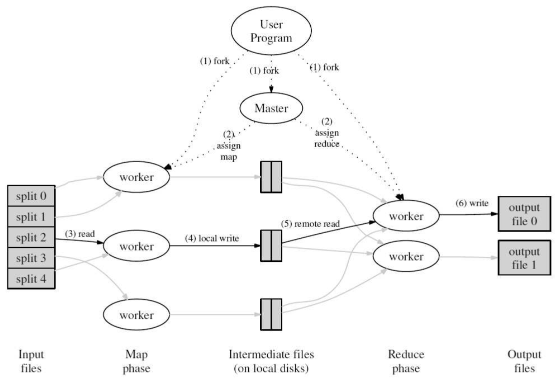
Count word occurances in in collection of documents:

```
map(String key, String value):
  // key:   document name
  // value: document contents
  for each word w in value:
    EmitIntermediate(w, "1");

reduce(String key, Iterator values):
  // key:    a word
  // values: a list of counts
  int result = 0;
  for each v in values:
    result += ParseInt(v);
  Emit(AsString(result));
```

Slide 24

## Execution Overview



Slide 25

## Master

Data structures:

- ➜ State of each map task and each reduce task
  (idle, in-progress, completed)
- ➜ Identity of worker machines
  (for non-idle tasks)
- ➜ Location of intermediate file regions
  (propagate from map to reduce tasks)

Fault tolerance:

- ➜ Data structures could be checkpointed to guard against failure
- ➜ In practice: Failure is unlikely
- ➜ On failure: Restart MapReduce

Slide 26

## Worker Fault Tolerance

Unreachable workers:

- ➜ Master pings workers periodically
- ➜ Unreachable workers are marked as failed.
- ➜ Tasks from failed workers reset to idle and rescheduled
  - ➜ Completed map tasks need restart too (results on local disks)
  - ➜ Completed reduce tasks not rescheduled (results on GFS)
- ➜ Map task first executes on A, then fails, then executed on B:
  Notify workers.
- ➜ Works well according to paper:
  Network upgrade disabled 80 machines at a time, but
  MapReduce continued to make progress.



Slide 27

Bad code:

- ➜ Sometimes user code crashes
- ➜ Ideally: Fix bug and re-run, but not always feasible
- ➜ Signal handler in worker catches crashes and sends *last gasp* packet to master, with sequence number of record
- ➜ If master records multiple failures on same record, the record is skipped on re-execution

Slide 28

## Locality

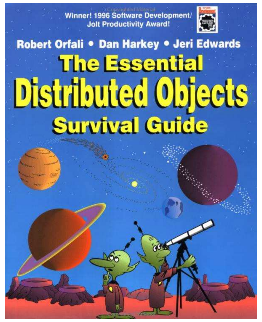
Network is scarce resource

- ➜ GFS divides files into blocks
- ➜ Each block is replicated (default: 3 replicas)
- ➜ MapReduce tries to schedule a map task on a machine that has a replica
- ➜ If that fails, schedule map task close to replica

Result: For large MapReduce operations, significant fraction of input data is read locally.

Slide 29

## Distributed Objects




Slide 30

## 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 31

## Object Model

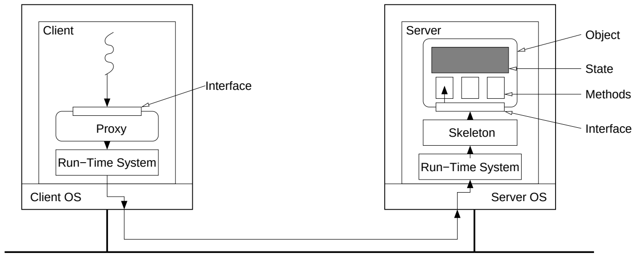
- ➜ Classes and Objects

  **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 32

## 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 33

## Client

Client Process:

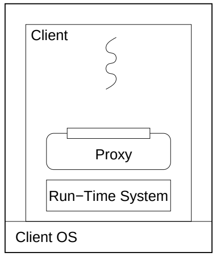
- Binds to distributed object
- Invokes methods on object

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 34

## Object Server

Object:

- State & Methods
- Implements a particular interface

Skeleton:

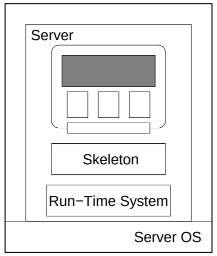
- Server stub
- Static vs dynamic skeletons

Run-Time System:

- Dispatches to appropriate object
- Invocation policies

Object Server:

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



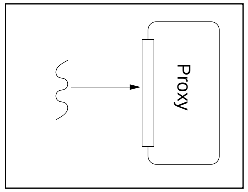



Slide 35

## Object Reference

Local Reference:

- Language reference to proxy

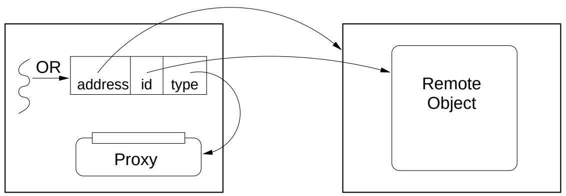



Slide 36

## Object Reference

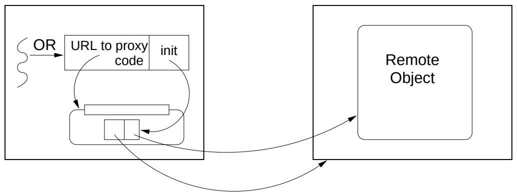
Remote Reference:

- Server address + object ID



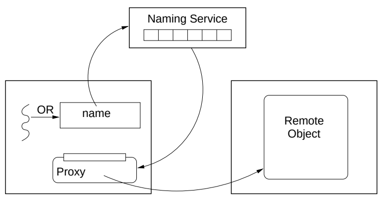
Slide 37

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




Slide 38

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




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

Slide 39

## 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 40

## CORBA

Features:

➜ Object Management Group (OMG) Standard (version 3.1)

➜ Range of language mappings

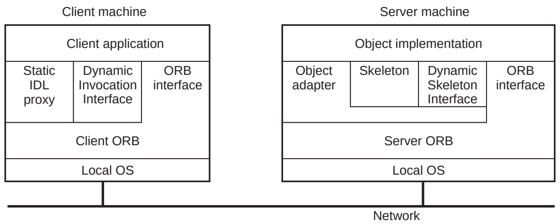
➜ Transparency: Location & some migration transparency

➜ 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 41

## CORBA Architecture




Slide 42

## Interfaces: OMG IDL

### 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 ();
  };
};
```

Slide 43

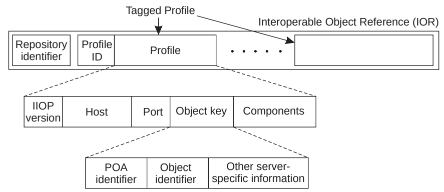
## Object Reference (OR)

### 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



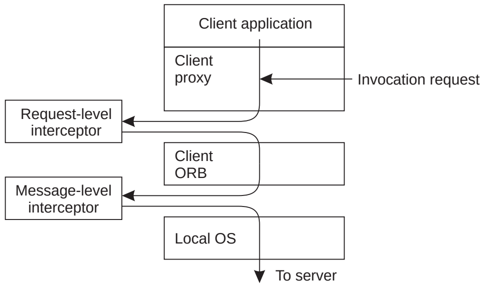

Slide 44

## 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 45

## Interceptors



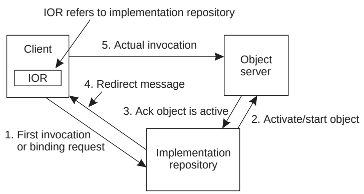

Slide 46

## Binding

Direct Binding:

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

Indirect Binding:




Slide 47

## CORBA Services

Some of the standardised services are the following:

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

Slide 48

## CORBA Bibliography


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

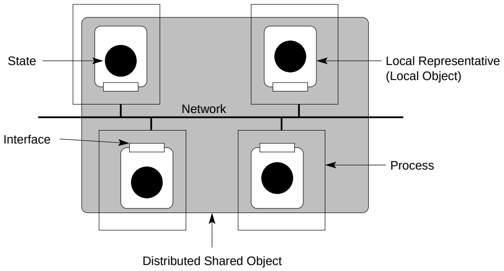
(2) *The Common Object Request Broker: Architecture and Specification (2.3.1)*, Object Management Group, 1999.

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

(4) *CORBAservices: 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



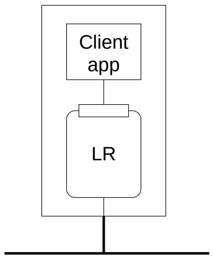

Slide 49

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

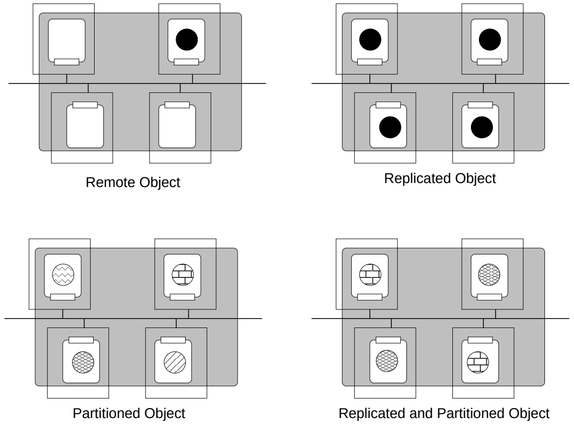
## Client

- ➜ 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

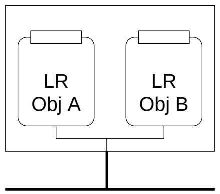
## Object




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




Location of LRs:

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

Slide 52

## Globe (Global Object Based Environment)

Slide 53

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

Slide 54

- 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

Slide 55

**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