Effective Architectures for Distributed Object Computing

Douglas C. Schmidt

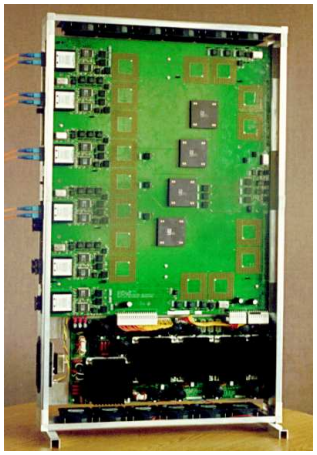
Associate Professor
www.ece.uci.edu/~schmidt/

Elec. Comp. Eng. Dept.
University of California, Irvine

Sponsors

NSF, DARPA, ATD, BBN, Boeing, Cisco, Comverse, GDIS, Experian, Global MT, Hughes, Kodak, Krones, Lockheed, Lucent, Microsoft, Mitre, Motorola, Nokia, Nortel, OCI, Oresis, OTI, QNX, Raytheon, SAIC, Siemens SCR, Siemens MED, Siemens ZT, Sprint, Telcordia, USENIX

Motivation: the Distributed Software Crisis



Symptoms

- ***Hardware*** gets smaller, faster, cheaper
- ***Software*** gets larger, slower, more expensive

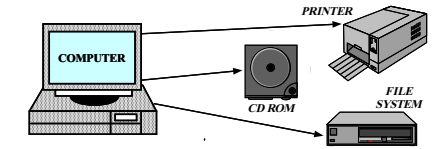
Culprits

- ***Inherent*** and ***accidental*** complexity

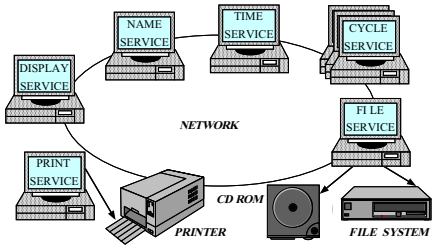
Solution Approach

- ***Components, Frameworks, Patterns, & Architecture***

Sources of Complexity for Distributed Applications



(1) STAND-ALONE APPLICATION ARCHITECTURE



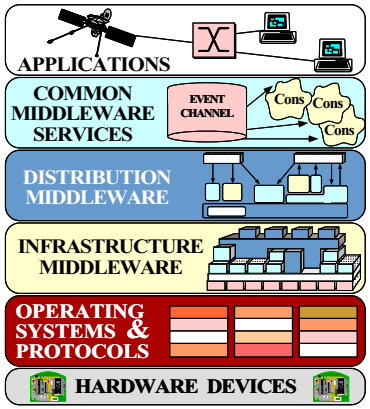
(2) DISTRIBUTED APPLICATION ARCHITECTURE

- **Inherent complexity**
  - Latency
  - Reliability
  - Partitioning
  - Ordering
- **Accidental Complexity**
  - Low-level APIs
  - Poor debugging tools
  - Algorithmic decomposition
  - Continuous re-invention

Motivation for COTS Middleware

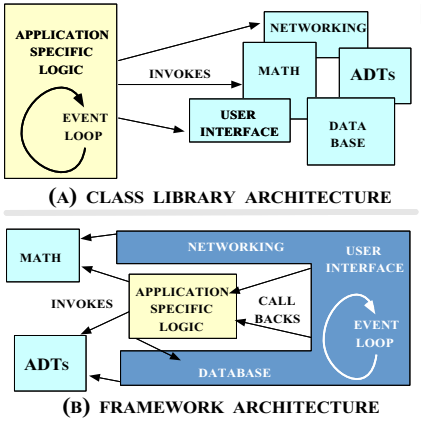
- Middleware makes it easier to build *efficient*, *reliably*, *predictable*, and *scalable* distributed applications, as follows:
  - Simplifying application interworking
    - Provides higher level integration than traditional *untyped TCP bytestreams*
  - Providing foundation for higher-level distributed object collaboration
    - *e.g.*, Windows COM+/OLE and the OMG Common Object Service Specification (COSS)
  - Programming distributed apps using same OO features used for non-distributed apps
    - *e.g.*, encapsulation, interface inheritance, polymorphism, and exception handling

Levels of Abstraction in Distributed Systems



- **Observations**
  - Historically, distributed apps built directly atop OS
  - Today, more and more apps built atop *middleware*
  - Middleware has several layers
- **Challenges**
  - Buy vs. build
  - Identify reuse boundaries
  - Determine where to add value

Techniques for Developing Middleware



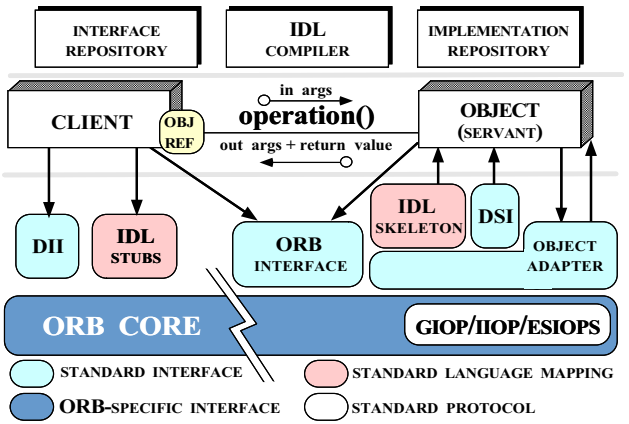
Proven solutions $\rightarrow$

- *Components*
  - Self-contained, "pluggable" ADTs
- *Frameworks*
  - Reusable, "semi-complete" applications
- *Patterns*
  - Problem/solution/context
- *Architecture*
  - Families of related patterns and components

Tutorial Outline

- Overview of CORBA middleware and applications
- Overview of advanced CORBA features
- Summary and recommendations

Overview of CORBA Middleware Architecture

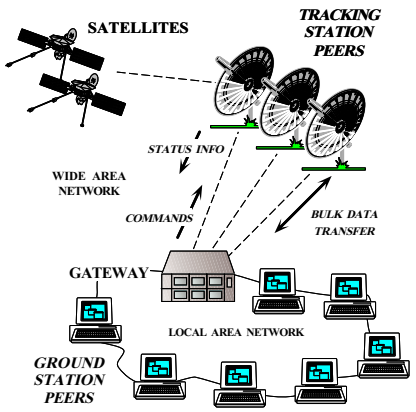


www.cs.wustl.edu/∼schmidt/corba.html

Goals of CORBA

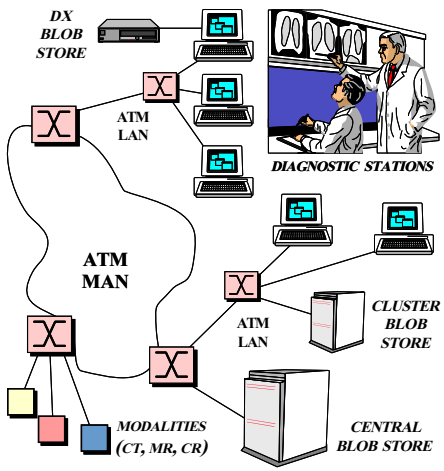
- Simplify distribution by automating
  - Object location & activation
  - Parameter marshaling
  - Demultiplexing
  - Error handling
- Provide foundation for higher-level services

Applying CORBA to Global PCS



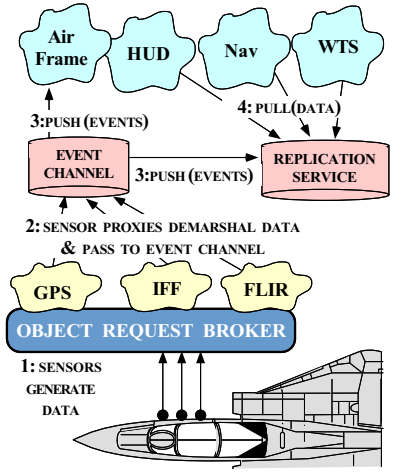
- **Domain Challenges**
  - Long latency satellite links
  - High reliability
  - Prioritization
- **URL**
  - ~schmidt/TAPOS-95.ps.gz

Applying CORBA to Medical Imaging



- **Domain Challenges**
  - Large volume of "Blob" data
    - *e.g.*, 10 to 40 Mbps
  - "Lossy compression" isn't viable
  - Prioritization of requests
- **URLs**
  - ~schmidt/COOTS-96.ps.gz
  - ~schmidt/av.ps.gz
  - ~schmidt/NMVC.html

Applying CORBA to Real-time Avionics



- **Domain Challenges**
  - Real-time periodic processing
  - Complex dependencies
  - Very low latency
- **URLs**
  - ~schmidt/JSAC-98.ps.gz
  - ~schmidt/TAO-boeing.html

CORBA Quoter Example

```
int main (void)
{
  // Use a factory to bind
  // to a Quoter.
  Quoter_var quoter =
    bind_quoter_service ();

  const char *name =
    "ACME ORB Inc.";

  CORBA::Long value =
    quoter->get_quote (name);
  cout << name << " = "
       << value << endl;
}
```

- Ideally, a distributed service should look just like a non-distributed service
- Unfortunately, life is harder when errors occur...

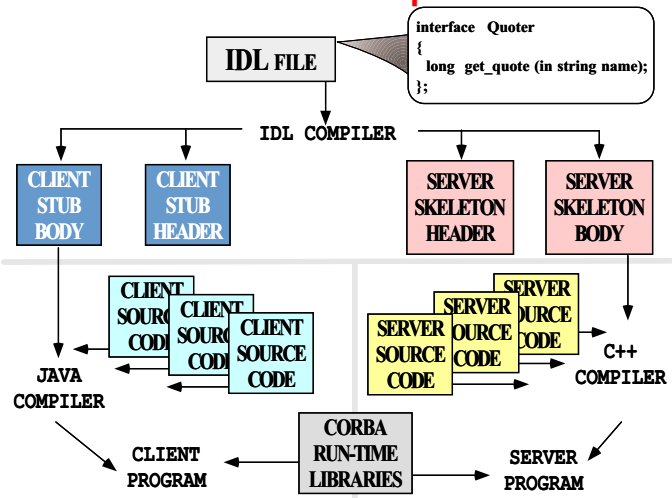
CORBA Quoter Interface

```
// IDL interface is like a C++
// class or Java interface.
interface Quoter
{
  exception Invalid_Stock {};

  long get_quote
    (in string stock_name)
    raises (Invalid_Stock);
};
```

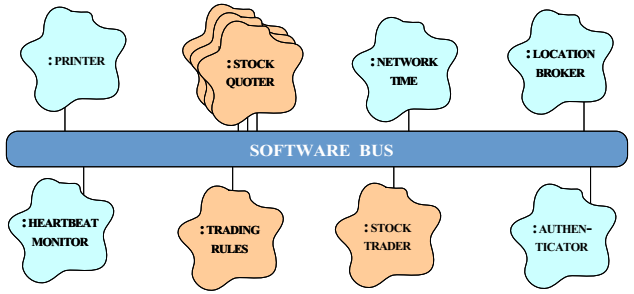
- We write an OMG IDL `interface` for our `Quoter`
  - Used by both clients and servers

Using OMG IDL promotes *language/platform independence*, *location transparency*, *modularity*, and *robustness*

OMG IDL Compiler

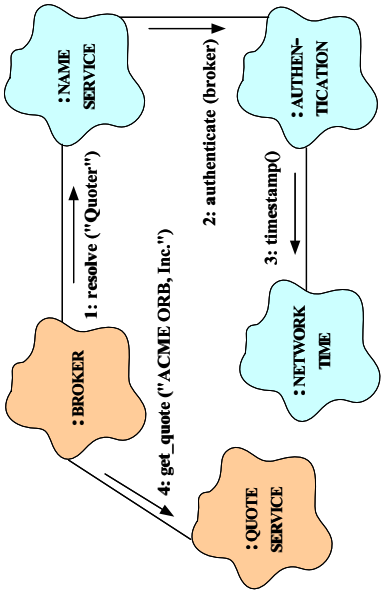


Software Bus



- CORBA provides a communication infrastructure for a heterogeneous, distributed collection of collaborating objects
- Analogous to “hardware bus”

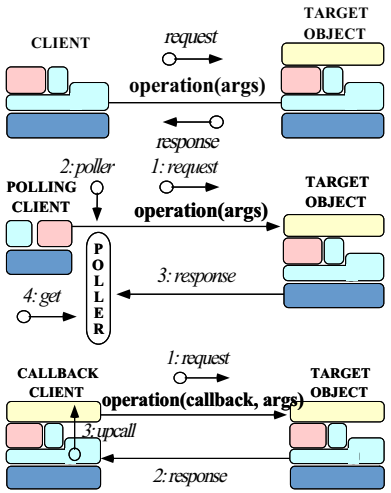
CORBA Object Collaboration



- Collaborating objects can be either remote or local
  - *i.e.*, distributed or collocated
- For this to work transparently the ORB should support *nested upcalls* and *collocation optimizations*

Communication Features of CORBA

- CORBA supports reliable, uni-cast communication
  - *i.e.*, *oneway*, *twoway*, *deferred synchronous*, and *asynchronous*
- CORBA objects can also collaborate in a *client/server*, *peer-to-peer*, or *publish/subscribe* manner
  - *e.g.*, COS Events & Notification Services define a publish & subscribe communication paradigm



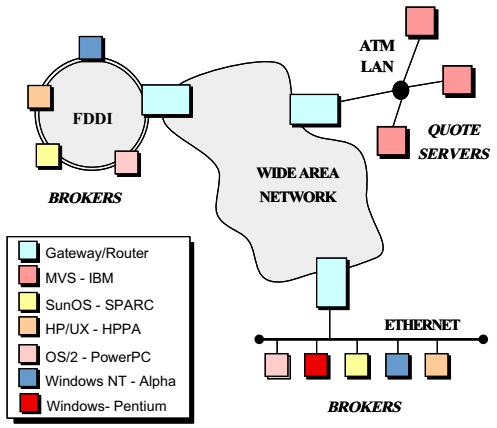
Related Work

- Traditional RPC (*e.g.*, DCE)
  - Only supports "procedural" integration of application services
  - Doesn't provide object abstractions, message passing, or dynamic invocation
  - Doesn't address inheritance of interfaces
- Windows COM/DCOM
  - Traditionally limited to desktop applications
  - Does not address heterogeneous distributed computing

Related Work (cont'd)

- Java RMI
  - Limited to Java only
  - Can be extended into other languages (*e.g.*, C or C++) by using a bridge across the Java Native Interface (JNI)
  - Well-suited for all-Java applications because of its tight integration with the Java virtual machine

CORBA Stock Quoter Application Example



- The quote server(s) maintains the current stock prices
- Brokers access the quote server(s) via CORBA

OMG IDL Quoter Definition

```
module Stock {
  exception Invalid_Stock {}; // Similar to structs.

  // Interface is similar to a C++ class.
  interface Quoter : CosLifeCycle::LifeCycleObject
  {
    long get_quote (in string stock_name)
      raises (Invalid_Stock);
    // Inherits:
    // void remove () raises (NotRemovable);
  };

  // Manage the lifecycle of a Quoter object.
  interface Quoter_Factory :
    CosLifeCycle::GenericFactory
  {
    // Returns a new Quoter selected by name
    // e.g., "Dow Jones," "Reuters,", etc.
    // Inherits:
    // Object create_object (in Key k,
    //                       in Criteria criteria)
    // raises (NoFactory, InvalidCriteria,
    //         CannotMeetCriteria);
    //
  };
};
```


RPC-style vs. Object-style Communication

Object References

- An "object reference" is an opaque handle to an object
  - It identifies the object's location
- Object references may be passed among processes on separate hosts
  - The underlying CORBA ORB will correctly convert object references into a form that can be transmitted over the network
  - The ORB provides the receiver with a pointer to a proxy in its own address space
    - This proxy refers to remote object implementation
- Object references are a powerful feature of CORBA
  - *e.g.*, supports *peer-to-peer* interactions and *distributed callbacks*

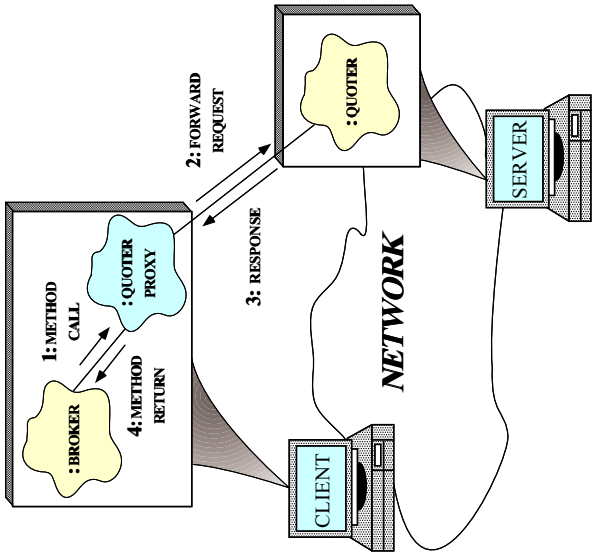
Object Reference in URL Format

- Transient object reference

Compiling the Interface Definition

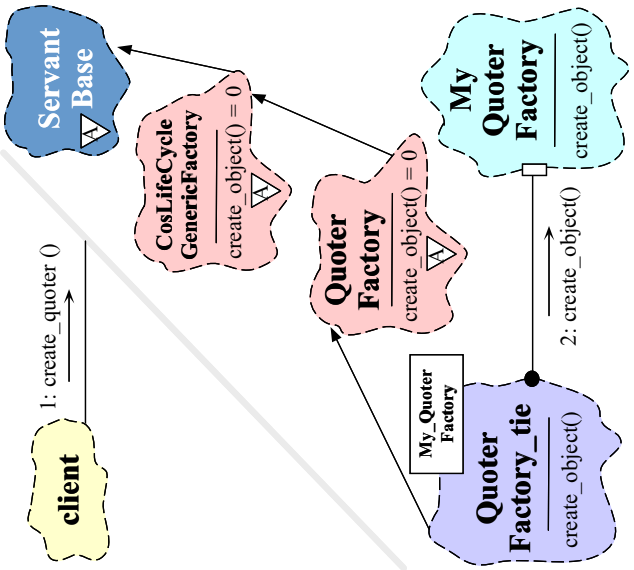
- Running the `Stock` module definition through the IDL compiler generates stubs and skeletons
  - The stub is a *proxy* that marshals parameters on the client
  - The skeleton is an *adapter* that demarshals parameters on the server
- CORBA associates a servant to a generated IDL skeleton using either
  1. The Class form of the Adapter pattern (inheritance)
     `POA_Stock::Quoter`
  2. The Object form of the Adapter pattern (object composition, *i.e.*, TIE)
     `template <class Impl> class POA_Stock::Quoter_tie`

Applying the Proxy Pattern for CORBA Stubs



Intent: provide a surrogate for another object that controls access to it

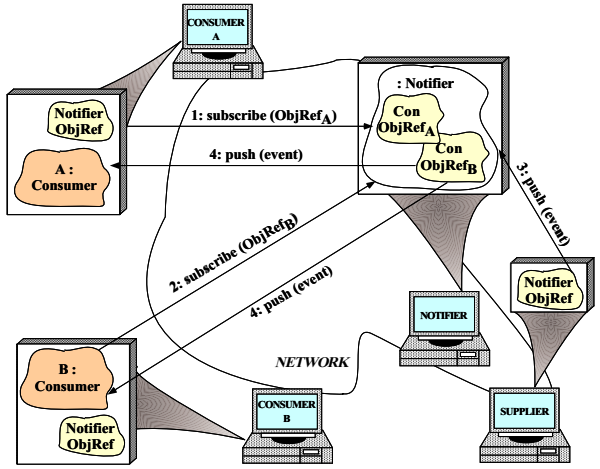
The Object Form of the Adapter Pattern



Binding a Client to a CORBA Object

- Several steps:
  1. Client uses `resolve_initial_references` and "Interoperable Naming Service" to obtain a `NamingContext`
     - This is the standard ORB "bootstrapping" mechanism
  2. Client then uses `NamingContext` to obtain desired object reference
  3. The client then invokes operations via object reference
- Object references can be passed as parameters to other remote objects
  - This supports various types of "factory" patterns

Using Object References for Push Architectures



Event Receiver Interface

```
struct Event {
  string topic_; // Used for filtering.
  any value_; // Event contents.
};

interface Consumer
{
  // Inform the Consumer
  // event has occurred.
  void push (in Event event);

  // Disconnect the Consumer
  // from the Notifier.
  void disconnect (in string reason);
};
```

A `Consumer` is called back by the Notifier

Notifier Interface

```
interface Notifier {
  // = For Consumers.
  // Subscribe the Consumer to receive
  // events that match filtering_criteria
  // applied by the Notifier.
  void subscribe
    (in Consumer consumer,
     in string filtering_criteria);
  // Unsubscribe the Consumer.
  void unsubscribe (in Consumer consumer);

  // = For Suppliers.
  // Push the Event to all the consumers
  // who have subscribed and who match
  // the filtering criteria.
  void push (in Event event);
};
```

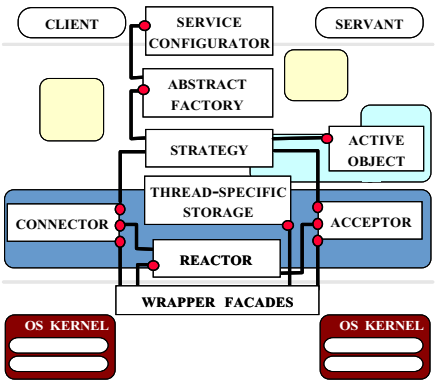
A Notifier publishes `Events` to subscribed Consumers

Notifier Implementation

```
class My_Notifier { // C++ pseudo-code
public:
  void subscribe (Consumer_ptr consumer,
                  const char *fc) {
    insert <consumer> into
    <consumer_set_> with <fc>.
  }
  void unsubscribe (Consumer_ptr consumer) {
    remove <consumer> from <consumer_set_>.
  }
  void push (const Event &event) {
    foreach <consumer> in <consumer_set_>
      if (event.topic_ matches <consumer>.filter_criteria)
        <consumer>.push (event);
  }
private: // e.g., use an STL map.
  map <string, Consumer_ptr> consumer_set_;
};
```

- The Notifier maintains a table of object references to `Consumers`

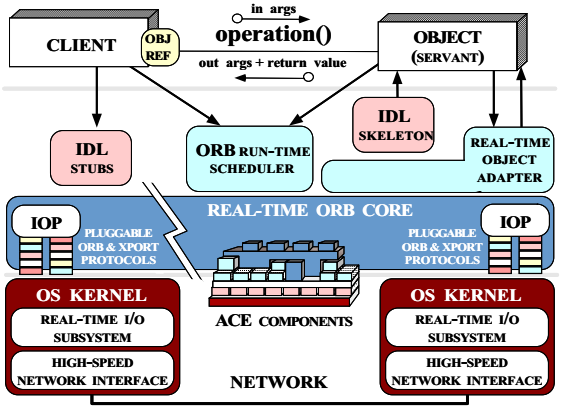
Applying Patterns to Develop Extensible ORBs



www.cs.wustl.edu/∼schmidt/ORB-patterns.ps.gz

- *Factories* produce *Strategies*
- *Strategies* implement interchangeable policies
- Concurrency strategies use *Reactor* and *Active Object*
- *Acceptor-Connector* decouple transport from GIOP operations
- *Service Configurator* permits dynamic configuration

The ACE ORB (TAO)

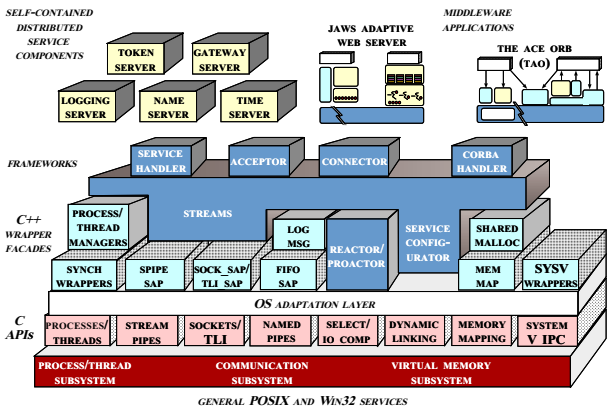


www.cs.wustl.edu/∼schmidt/TAO.html

TAO Overview →

- An open-source, standards-based, real-time, high-performance CORBA ORB
- Runs on POSIX, Win32, & embedded RT platforms
  - *e.g.*, VxWorks, Chorus, LynxOS
- Leverages ACE

The ADAPTIVE Communication Environment (ACE)



www.cs.wustl.edu/∼schmidt/ACE.html

ACE Overview →

- A concurrent OO networking framework
- Available in C++ and Java
- Ported to POSIX, Win32, and RTOSs

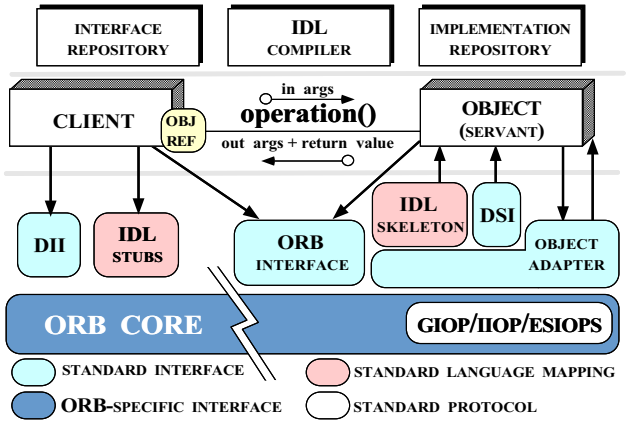
Related work →

- x-Kernel
- SysV STREAMS

ACE and TAO Statistics

- Over 35 person-years of effort
  - ACE > 200,000 LOC
  - TAO > 125,000 LOC
  - TAO IDL compiler > 100,000 LOC
  - TAO CORBA Object Services > 150,000 LOC
- Ported to UNIX, Win32, MVS, and RTOS platforms
- Large user community
  - www.cs.wustl.edu/∼schmidt/ACE-users.html
- Currently used by dozens of companies
  - Bellcore, Boeing, Ericsson, Kodak, Lockheed, Lucent, Motorola, Nokia, Nortel, Raytheon, SAIC, Siemens, etc.
- Supported commercially
  - ACE → www.riverace.com
  - TAO → www.ociweb.com

Advanced CORBA Features

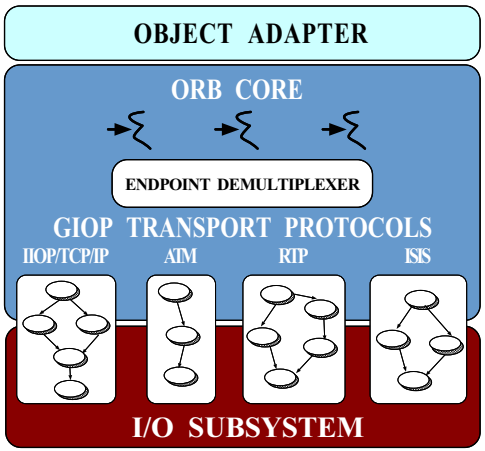


www.cs.wustl.edu/∼schmidt/corba.html

Features

- ORB Core + ORB protocols
- Real-time CORBA
- Portable Object Adapter
- Asynchronous Messaging

Overview of the ORB Core

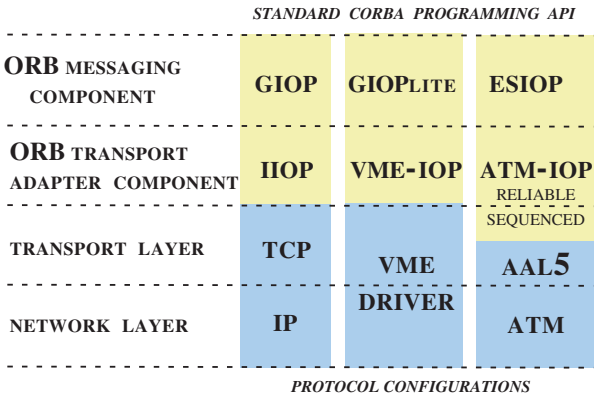


Features

- Connection/memory management
- Request transfer
- Endpoint demuxing
- Concurrency control

CORBA Interoperability Protocol Architecture

Features →



- Presentation layer
  - *e.g.*, CDR
- Message formats
  - *e.g.*, GIOP, DCE
- Transport assumptions
  - *e.g.*, TCP, ATM
- Object addressing
  - *e.g.*, IIOP IOR

GIOP/IIOP Overview

- Common Data Representation (CDR)
  - Transfer syntax mapping OMG-IDL data types into a bi-canonical low-level representation
    - Supports variable byte ordering and aligned primitive types
- Message transfer
  - Request multiplexing, *i.e.*, shared connections
  - Ordering constraints are minimal, *i.e.*, can be asynchronous
- Message formats
  - Client: `Request`, `CancelRequest`, `LocateRequest`
  - Server: `Reply`, `LocateReply`, `CloseConnection`
  - Both: `MessageError`

Example GIOP Format

```
module GIOP {
  enum MsgType {
    Request, Reply, CancelRequest,
    LocateRequest, LocateReply,
    CloseConnection, MessageError
  };

  struct MessageHeader {
    char magic[4];
    Version GIOP_version;
    octet byte_order; // fragment bit in 1.1.
    octet message_type;
    unsigned long message_size;
  };

  struct RequestHeader {
    IOP::ServiceContextList service_context;
    unsigned long request_id;
    boolean response_requested;
    sequence<octet> object_key;
    string operation;
    Principal requesting_principal;
  };

    // ...
```


Overview of the Real-time CORBA Specification

Features

1. Portable priorities
2. End-to-end priority propagation
3. Protocol properties
4. Thread pools
5. Explicit binding
6. Standard synchronizers

www.cs.wustl.edu/∼schmidt/oorc.ps.gz

Portable Priorities

Features

- Designed to support heterogeneous real-time platforms
- *CORBA priorities* range from $0 \rightarrow 32767$
- Users can map CORBA priorities to *native OS priorities*
- No silver bullet, but rather an "enabling technique"

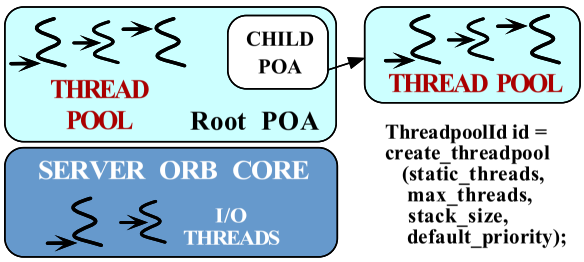
End-to-End Priority Propagation

www.cs.wustl.edu/∼schmidt/RT-ORB-std-new.pdf.gz

Features

- Priorities can propagate end-to-end

Thread Pools

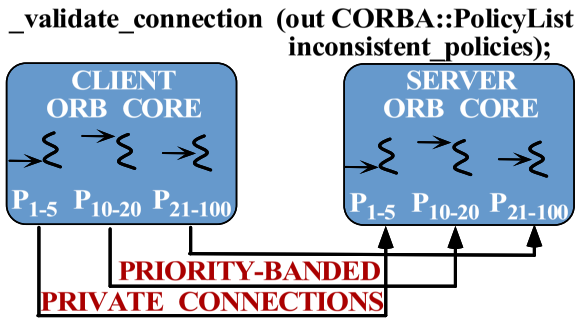


Features

- Pre-allocate threads and thread attributes
  - Stacksize
  - Static threads
  - Maximum threads
  - Default priority
- Applicable at both the ORB and POA level

- Supports heterogeneous RTOS priority mappings
- Supports priority inheritance
- Servers can also declare priority

Explicit Binding



Features

- Enables pre-establishment of connections
  - Priority-banded connections
  - Private connections
  - Protocol policies

Configurable Protocol Properties

```
interface ProtocolProperties {};

typedef struct {
  IOP::ProfileId protocol_type;
  ProtocolProperties
    orb_protocol_properties;
  ProtocolProperties
    transport_protocol_properties;
} Protocol;
typedef sequence <Protocol> ProtocolList;

interface TCPProtocolProperties
  : ProtocolProperties
{
  attribute long send_buffer_size;
  attribute long recv_buffer_size;
  attribute boolean keep_alive;
  attribute boolean dont_route;
  attribute boolean no_delay;
};
```

Features

- Select and configure communication protocols
  - *e.g.*, TCP socket options
- Supports *ORB protocol* and *transport protocol* configuration
- Ordering in `ProtocolList` indicates preferences

Standard Synchronizers

Features

- A portable Mutex API
  - *e.g.*, `lock`, `unlock`, `try_lock`
- Necessary to ensure consistency between ORB and application synchronizers
- Locality constrained

Overview of the Portable Object Adapter (POA)

POA Features

- Creates object refs
- Activates and deactivates objects
- Etherealizes and incarnates servants
- Maps requests to servants

The POA is very important for certain applications

- *e.g.*, telecom MIBs, enterprise servers

Design Goals of the Portable Object Adapter

- Servants that are portable between ORBs
- Objects with persistent & transient identities
- Transient objects with minimal programming effort and overhead
- Transparent activation & deactivation of servants
- Implicit and explicit servant activation
- A single servant can support multiple object identities
- Multiple (nested) instances of the POA in a server process
- POA behavior is dictated by creation policies
- Servants can inherit from skeletons or use DSI

The POA Architecture

POA Components

- Client: Makes requests on an object through one of its references
- Server: Computational context for servants
  - Generally, a server corresponds to a process
  - Client and server are "roles" - a program can play both roles
- Object: A CORBA programming entity with an identity, an interface, and an implementation
- Servant: A programming language entity that implements requests on one or more objects
- Policy: Specifics the characteristics of a POA or child POA

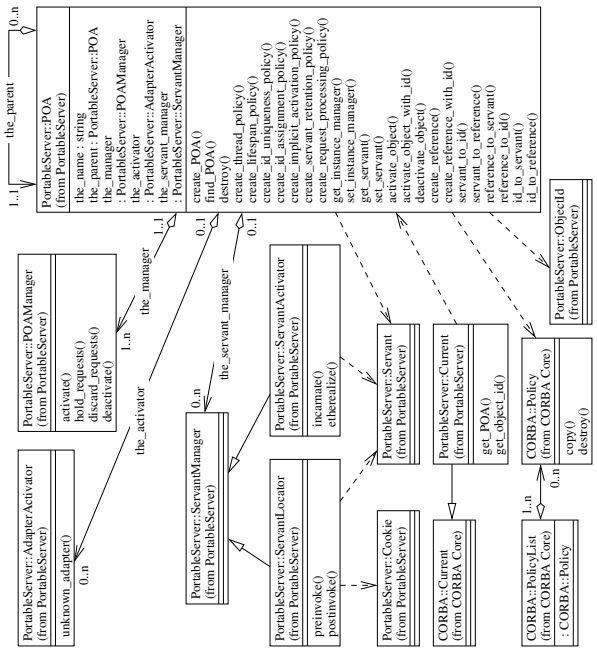
POA Components (cont'd)

- Object Id: A value that is used by the POA and by the implementation to identify a particular CORBA object
  - Object Id values may be assigned by the POA, or by the user implementation
  - Object Id values are hidden from clients, encapsulated by references
  - Object Ids have no standard form; they are managed by the POA as uninterpreted octet sequences
- Object Reference: Encapsulates an Object Id, a POA identity, and transport profiles
- POA: A namespace for Object Ids and a namespace for child POAs
  - Nested POAs form a hierarchical name space for objects in servers

POA Components (cont'd)

- POA Manager: Encapsulates the processing state of associated POAs
  - Can hold, discard requests for the associated POAs and deactivate POA(s)
- Servant Manager: Two kinds of callback objects used to incarnate and etherealize servants on demand
  - `ServantActivator` → first time
  - `ServantLocator` → one time
- Adapter Activator: Callback object used when a request is received for a child POA that does not exist currently
  - The adapter activator can then create the required POA on demand

POA Architecture in UML



POA Manager Processing States

Getting the Root POA

```
// ORB is ``locality constrained''
CORBA::ORB_var orb = CORBA::ORB_init (argc, argv);

// Root POA is the default POA (locality constrained)
CORBA::Object_var obj =
  orb->resolve_initial_references ("RootPOA");

// Type-safe downcast.
PortableServer::POA_var root_POA
  = PortableServer::POA::_narrow (obj.in ());

// Activate the POA.
PortableServer::POA_Manager_var poa_manager =
  root_POA->the_POAManager ();
poa_manager->activate ();
```


Creating a Child POA

```
CORBA::PolicyList policies (2);

policies[0] = root_POA->create_thread_policy
  (PortableServer::ThreadPolicy::ORB_CTRL_MODEL);

policies[1] = root_POA->create_lifespan_policy
  (PortableServer::LifespanPolicy::PERSISTENT);

PortableServer::POA_ptr child_poa =
  root_POA->create_POA
             ("child_poa",
              PortableServer::POAManager::_nil (),
              policies);
```


Explicit Activation with POA-assigned Object Ids

```
// IDL
interface Quoter /* ... */
{
  long get_quote
    (in string stock_name)
      raises (Invalid_Stock);
};

// Auto-generated for use by servants.
class My_Quoter : public virtual POA_Stock::Quoter
{
public:
  // ...
  CORBA::Long get_quote (const char *stock_name);
};

My_Quoter *quoter = new My_Quoter;
PortableServer::ObjectId_var oid =
  poa->activate_object (quoter);

poa->the_POAManager ()->activate ();
orb->run ();
```

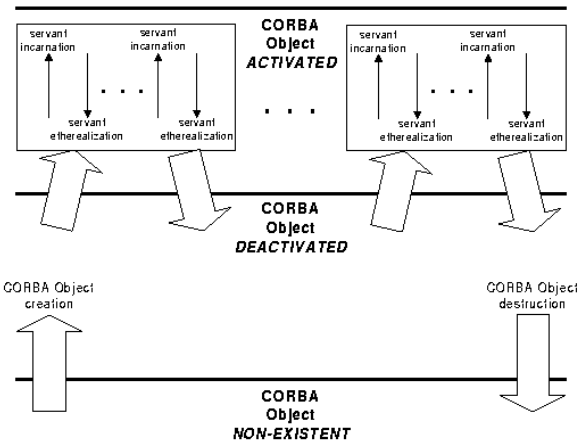
Explicit Activation With User-assigned Object Ids

```
// Create a new servant object.
My_Quoter *quoter = new My_Quoter;

// Create a new Object ID for the object.
PortableServer::ObjectId_var oid =
  PortableServer::string_to_ObjectId ("my quoter");

// Activate the object with the new Object ID.
poa->activate_object_with_id (oid.in (),
                              quoter);
// Run the ORB's event loop.
poa->the_POAManager ()->activate ();
orb->run ();
```


Creating References Before Activation

```
PortableServer::ObjectId_var oid =
  PortableServer::string_to_ObjectId ("my quoter");
CORBA::Object_var obj =
  poa->create_reference_with_id (oid.in (),
                                 "IDL:Quoter:1.0");
// Insert into a name context.
name_context->bind (svc_name, obj.in ());

// ...later...

My_Quoter *quoter = new My_Quoter;
poa->activate_object_with_id (oid.in (), quoter);
```


Request Lifecycle for POA



Servant Activator Definition

```
typedef ServantBase *Servant;

// Skeleton class
namespace POA_PortableServer
{
  class ServantActivator :
    public virtual ServantManager
    {
      // Destructor.
      virtual ~ServantActivator (void);

      // Create a new servant for <id>.
      virtual Servant incarnate
        (const ObjectId &id,
         POA_ptr poa) = 0;

      // <servant> is no longer active in <poa>.
      virtual void etherealize
        (const ObjectId &,
         POA_ptr poa,
         Servant servant,
         Boolean remaining_activations) = 0;
    };
}
```

Custom ServantLocator Definition and Creation

```
// Implementation class.
class My_Quoter_Servant_Locator :
  public POA_PortableServer::ServantLocator {
  Servant preinvoke
    (const PortableServer::ObjectId &id,
     PortableServer::POA_ptr poa,
     const char *operation,
     PortableServer::Cookie &cookie) {
    CORBA::String_var str =
      PortableServer::ObjectId_to_string (id);
    CORBA::Long val =
      database_lookup (str);
    if (val == -1)
      throw CORBA::OBJECT_NOT_EXIST ();
    return new My_Quoter (val);
  }

  void postinvoke
    (const PortableServer::ObjectId &id,
     PortableServer::POA_ptr poa,
     const char *operation,
     PortableServer::Cookie cookie,
     PortableServer::Servant servant)
   { delete servant; }
};
```

Custom ServantActivator Definition and Creation

```
// Implementation class.
class My_Quoter_Servant_Activator :
  public POA_PortableServer::ServantActivator
{
  Servant incarnate (const ObjectId &oid,
                     POA_ptr poa) {
    String_var s =
      PortableServer::ObjectId_to_string (oid);

    if (strcmp (s, "my quoter") == 0)
      return new My_Quoter;
    else
      throw CORBA::OBJECT_NOT_EXIST ();
  }

  void etherealize
    (const ObjectId &oid,
     POA_ptr poa,
     Servant servant,
     Boolean remaining_activations) {
    if (remaining_activations == 0)
      delete servant;
  }
};
```

Registering Servant Activators

```
PortableServer::ObjectId_var oid =
  PortableServer::string_to_ObjectId ("my quoter");
CORBA::Object_var obj =
  poa->create_reference_with_id (oid.in (),
                                 "IDL:Quoter:1.0");
// Insert into a name context.
name_context->bind (svc_name, obj.in ());

My_Quoter_Servant_Activator *quoter_activator =
  new My_Quoter_Servant_Activator;

// Locality constrained.
ServantActivator_var activator = quoter_activator->_this ();
poa->set_servant_manager (activator);
poa->the_POAmanager ()->activate ();

orb->run ();
```

Servant Locator Definition

```
typedef ServantBase *Servant;

// Skeleton class
namespace POA_PortableServer
{
  class ServantLocator :
    public virtual ServantManager
  {
    // Destructor.
    virtual ~ServantLocator (void);

    // Create a new servant for <id>.
    PortableServer::Servant preinvoke
      (const PortableServer::ObjectId &id,
       PortableServer::POA_ptr poa,
       const char *operation,
       PortableServer::Cookie &cookie) = 0;

    // <servant> is no longer active in <poa>.
    void postinvoke
      (const PortableServer::ObjectId &id,
       PortableServer::POA_ptr poa,
       const char *operation,
       PortableServer::Cookie cookie,
       PortableServer::Servant servant) = 0;
  };
}
```

Additional Information on the POA

- See POA specification for some examples:
  - One Servant for all Objects
  - Single Servant, many objects and types, using DSI
- See Vinoski/Henning book for even more examples
  - See also www.cs.wustl.edu/∼schmidt/report-doc.html
- See TAO release to experiment with working POA examples
  - `$TAO_ROOT/examples/POA/`

Motivation for Asynchronous Method Invocations (AMI)

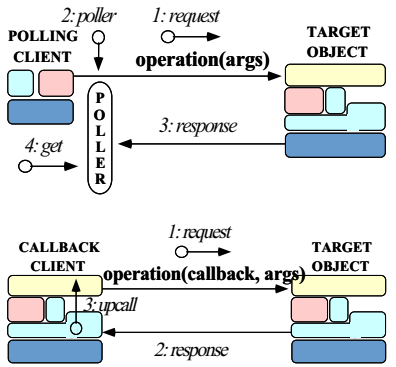
- Early versions of CORBA lacked support for asynchronous two-way invocations
- This omission yielded the following drawbacks
  1. Increase the number of client threads
     - *e.g.*, due to synchronous two-way communication
  2. Increase the end-to-end latency for multiple requests
     - *e.g.*, due to blocking on certain long-delay operations
  3. Decrease OS/network resource utilization
     - *e.g.*, inefficient support for bulk data transfers

Limitations with Workarounds for CORBA's Lack of Asynchrony

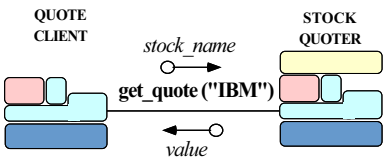
- *Synchronous method invocation* (SMI) multi-threading
  - Often non-portable, non-scalable, and inefficient
- Oneway operations
  - Best-effort semantics are unreliable
  - Requires *callback* objects
  - Applications must match callbacks with requests
- Deferred synchronous
  - Uses DII, thus *very* hard to program
  - Not type-safe

OMG Solution → CORBA Messaging Specification

- Defines *QoS Policies* for the ORB
  - Timeouts
  - Priority
  - Reliable one-ways
- Specifies two *asynchronous method invocation* (AMI) models
  1. Poller model
  2. Callback model
- Standardizes *time-independent invocation* (TII) model
  - Used for store/forward routers



AMI Callback Overview



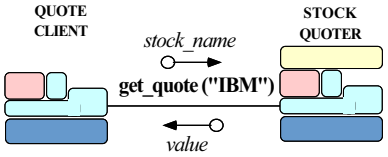
Quoter IDL Interface:

```
module Stock {
 interface Quoter {
   // Two-way operation to
   // get current stock value.
   long get_quote
     (in string stock_name);
 };
 // ...
}
```

Implied-IDL:

```
module Stock {
 interface Quoter {
   // Two-way synchronous operation.
   long get_quote (in string stock_name);

   // Two-way asynchronous operation.
   void sendc_get_quote
      (Stock::AMI_QuoterHandler
       in string stock);
  };
};

// ReplyHandler.
module Stock {
  interface AMI_QuoterHandler
    : Messaging::ReplyHandler {
   public:
     // Callback method
     void get_quote (in long l);
   };
};
```


Example: Synchronous Client



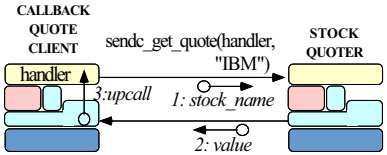
IDL-generated stub:

```
CORBA::ULong
Stock::Quoter::get_quote
  (const char *name)
{
  // setup connection
  // marshal
  // send request
  // get reply
  // demarshal
};
```

Application:

```
// NASDAQ abbreviations for ORB vendors.
static const char *stocks[] =
{
  "IONAY" // IONA Orbix
  "INPR"  // Inprise VisiBroker
  "IBM"   // IBM Component Broker
}
// Set the max number of ORB stocks.
static const int MAX_STOCKS = 3;

// Make synchronous two-way calls.
  for (int i = 0; i < MAX_STOCKS; i++) {
    CORBA::Long value =
      quoter_ref->get_quote (stocks[i]);
    cout << "Current value of "
         << stocks[i] << " stock: "
         << value << endl;
  }
```


Example: AMI Callback Client



Asynchronous stub:

```
void
Stock::Quoter::sendc_get_quote
       (AMI_QuoterHandler_ptr,
        const char *name)
{
 // setup connection
 // marshal
 // send request
 // return
};
```

Reply Handler Servant:

```
class My_Async_Stock_Handler
 : public POA_Stock::AMI_QuoterHandler {
public:
 My_Async_Stock_Handler (const char *s)
   : stock_ (CORBA::string_dup (s))
 { }
 ~My_Async_Stock_Handler (void) { }

 virtual void get_quote (CORBA::Long q)
 {
   cout << stock_ << " stock: "
        << q << endl;
   // Decrement global reply count.
   reply_count--;
 }

private:
  CORBA::String_var stock_;
};
```


Example: AMI Callback Client (cont'd)

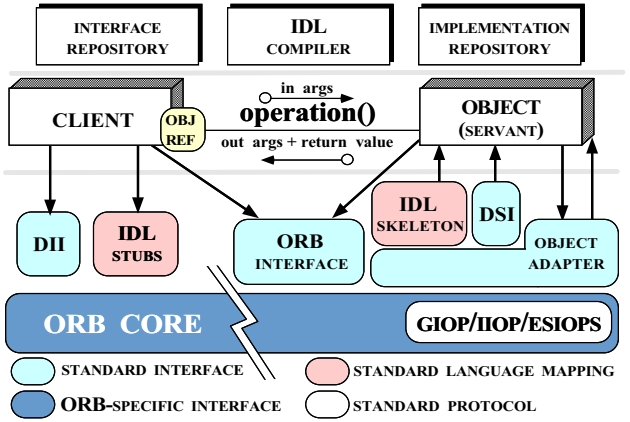
```
// Global reply count
int reply_count = MAX_STOCKS;

// Servants.
My_Async_Stock_Handler *
  handlers[MAX_STOCKS];

// Objrefs.
Stock::AMI_QuoterHandler_var
  handler_refs[MAX_STOCKS];

int i;

// Initialize ReplyHandler
// servants.
for (i = 0; i < MAX_STOCKS; i++)
 handlers[i] = new
 My_Async_Stock_Handler (stocks[i]);
```

```
// Initialize ReplyHandler object refs.
for (i = 0; i < MAX_STOCKS; i++)
  handler_refs[i] =
    handlers[i]->_this ();

// Make asynchronous two-way calls
// using the callback model.
for (i = 0; i < MAX_STOCKS; i++)
  quoter_ref->sendc_get_quote
    (handler_refs[i],
     stocks[i]);

// Event loop to receive all replies.
while (reply_count > 0)
  if (orb->work_pending ())
    orb->perform_work ();
```

Additional Information on AMI

- See Asynchronous Messaging specification
  - www.cs.wustl.edu/∼schmidt/CORBA-docs/98-05-05.pdf.gz
- See Vinoski's CACM article on CORBA 3.0 for more info.
  - www.cs.wustl.edu/∼schmidt/vinoski-98.pdf.gz
- See our papers on AMI
  - www.cs.wustl.edu/∼schmidt/report-doc.html
  - www.cs.wustl.edu/∼schmidt/ami1.ps.gz
  - www.cs.wustl.edu/∼schmidt/ami2.ps.gz
- See TAO release to experiment with working AMI examples
  - `$TAO_ROOT/examples/AMI/`

Criteria for Evaluating Middleware



Criteria

- Learning curve
- Interoperability
- Portability
- Feature Limitations
- Performance

www.cs.wustl.edu/∼schmidt/corba.html

Learning Curve

- Middleware introduces the following:
  1. New concepts
     - *e.g.*, object references, proxies, and object adapters
  2. New components and tools
     - *e.g.*, interface definition languages, IDL compilers, and object-request brokers
  3. New features
     - *e.g.*, exception handling and interface inheritance
- Time spent learning this must be amortized over many projects

Interoperability

- The first CORBA 1 spec was woefully incomplete with respect to interoperability
  - The solution was to use ORBs provided by a single supplier
- CORBA 2.x defines a useful interoperability specification
  - Later extensions deal with portability issues for server-side
    - *i.e.*, the POA spec
- Most ORB implementations now support IIOP or GIOP robustly...
  - However, higher-level CORBA services aren't covered by ORB interoperability spec...

Portability

- To improve portability, the latest CORBA specification standardizes
  - IDL-to-C++ language mapping
  - Naming service, event service, lifecycle service
  - ORB initialization service
  - Portable Object Adapter API
  - Servant mapping
- Porting applications from ORB-to-ORB will be limited, however, until conformance tests become common-place
  - www.opengroup.org/testing/testsuites/vsorb.htm

Feature Limitations

- Standard middleware doesn't (yet) address key "inherent" complexities of distributed computing, *e.g.*,
  - *Latency*
  - *Fault tolerance* (note, RFP process is underway on this)
  - *Causal ordering*
  - *Deadlock*
- To some extent, it does address *service partitioning*
  - But you must be very careful in practice...

Feature Limitations (cont'd)

- Many ORBs do not support passing objects-by-value (yet)
  - However, CORBA 2.3 specifies a solution for this
- Most ORBs still support only the following semantics:
  - Object references are passed-by-reference
    - However, all operation invocations on the remote host are routed back to the originator host
  - C-style structures and discriminated unions may be passed-by-value
    - However, these structures and unions do *not* contain any methods
- Until OBV spec is ubiquitous, objects by value can be hand-crafted using "factories"

Feature Limitations (cont'd)

- Most ORBs do not yet support AMI and/or standard timeouts
  - These capabilities are defined in the OMG CORBA Messaging spec Specification
- Fault tolerance is not yet standardized
  - There's an RFP in the works, however
  - www.omg.org/techprocess/meetings/schedule/Fault_Tolerance_RFP.html
- Versioning is supported in IDL via `pragmas`
  - Unlike Sun RPC or DCE, which include in language

Performance Limitations

- Performance may not be as good as hand-crafted code for some applications due to
  - Additional remote invocations for naming
  - Marshaling/demarshaling overhead
  - Data copying and memory management
  - Endpoint and request demultiplexing
  - Context switching and synchronization overhead
- Typical trade-off between extensibility, robustness, maintainability → *micro-level efficiency*
- Note that a well-crafted ORB may be able to automatically optimize *macro-level efficiency*

Lessons Learned Developing QoS-enabled ORBs

- Avoid dynamic connection management
- Minimize dynamic memory management and data copying
- Avoid multiplexing connections for different priority threads
- Avoid complex concurrency models
- Integrate ORB with OS and I/O subsystem and avoid reimplementing OS mechanisms
- Guide ORB design by empirical benchmarks and patterns

Concluding Remarks

- Researchers and developers of distributed, real-time applications confront many common challenges
  - *e.g.*, service initialization and distribution, error handling, flow control, scheduling, event demultiplexing, concurrency control, persistence, fault tolerance
- Successful researchers and developers apply *patterns*, *frameworks*, and *components* to resolve these challenges
- Careful application of patterns can yield efficient, predictable, scalable, *and* flexible middleware
  - *i.e.*, middleware performance is largely an "implementation detail"
- Next-generation ORBs will be highly QoS-enabled, though many research challenges remain

Web URLs for Additional Information

- Additional information about middleware and CORBA is available on-line at the following WWW URLs
  - Doug Schmidt's CORBA page
    - www.cs.wustl.edu/∼schmidt/corba.html
  - OMG's WWW Page
    - www.omg.org/
    - www.omg.org/library/c2indx.html
  - CETUS CORBA Page
    - www.cetus-links.org/oo_corba.html
  - LANL's OMG Page
    - www.acl.lanl.gov/CORBA