

Introduction to Distributed Computing

Session 1

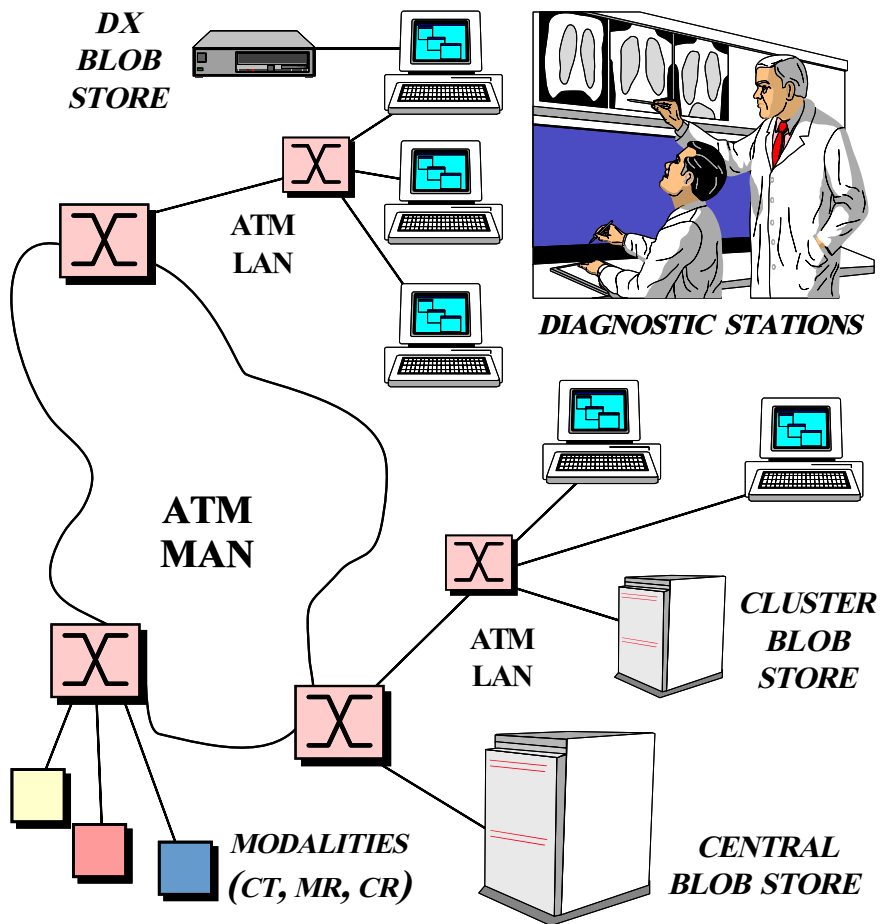
OOMWorks

Irfan Pyarali and Pradeep Gore

irfan@oomworks.com and pradeep@oomworks.com

May 3, 2001

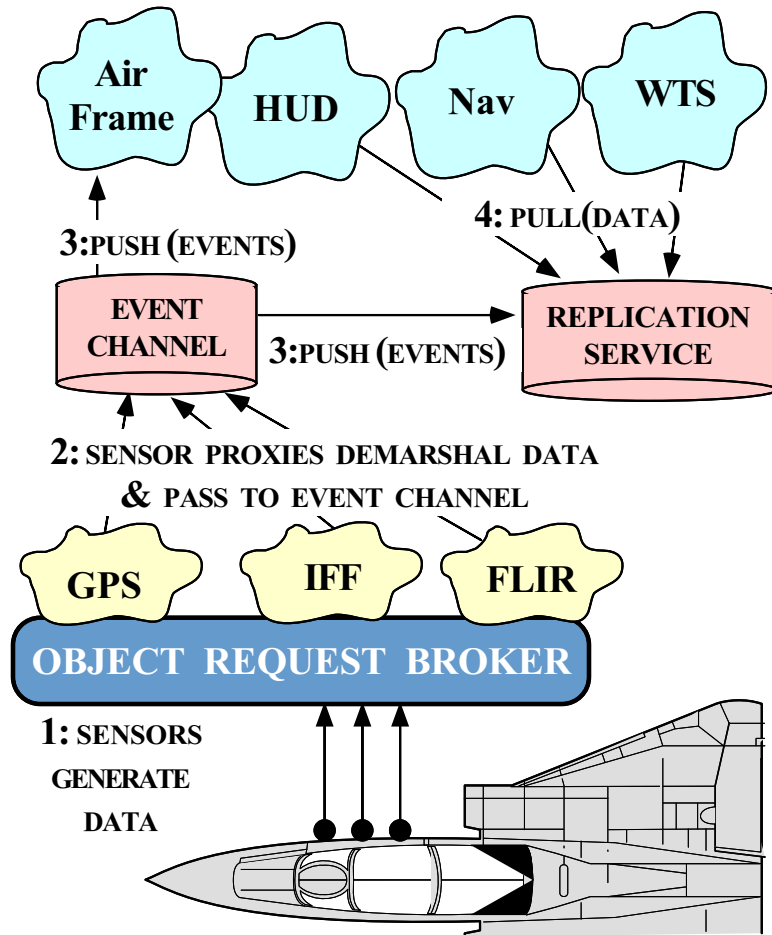
Example of distributed computing - Medical Imaging



- **Domain Challenges**

- Large volume of “Blob” data
 - * *e.g.*, 10 to 40 Mbps
- “Lossy compression” isn’t viable
- Prioritization of requests

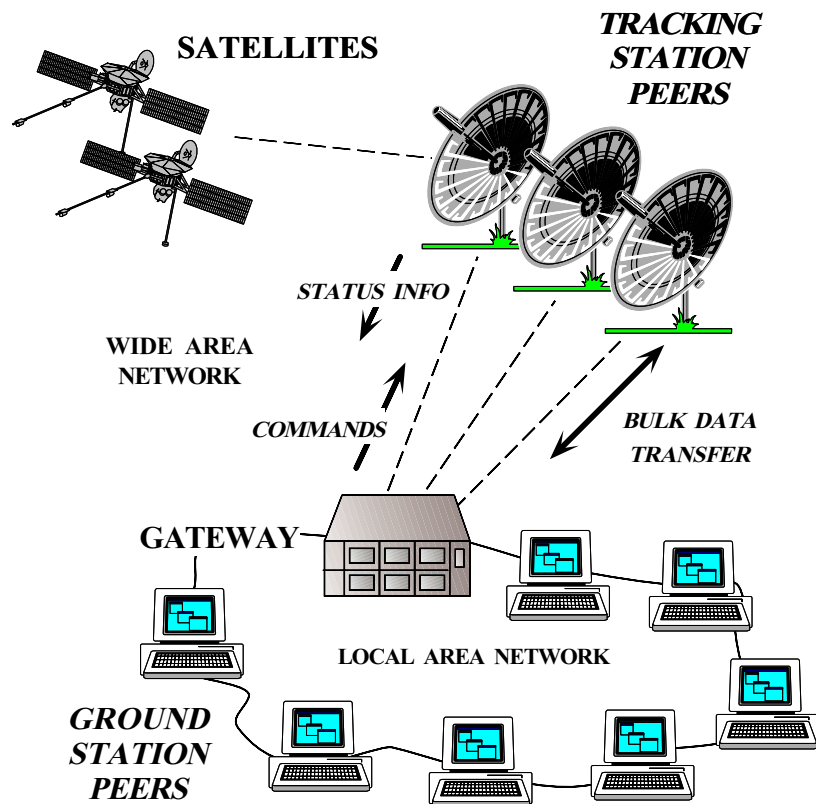
Real-time Avionics



- **Domain Challenges**

- Real-time periodic processing
- Complex dependencies
- Very low latency

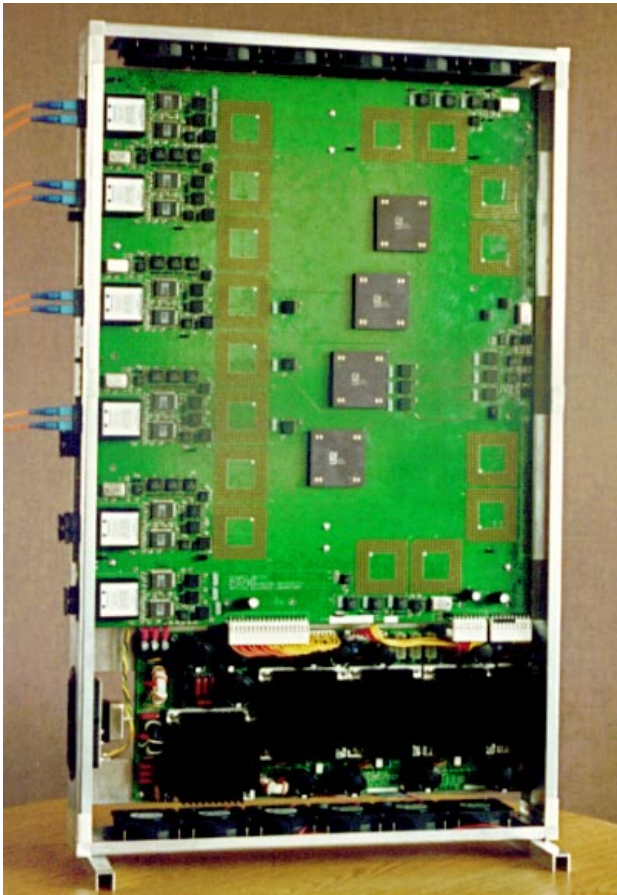
Global PCS



- **Domain Challenges**

- Long latency satellite links
- High reliability
- Prioritization

Motivation: the Distributed Software Crisis



Symptoms

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

Culprits

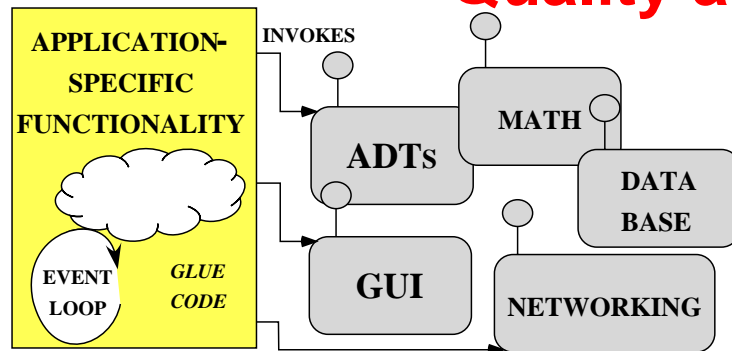
- **Inherent** and **accidental** complexity

Solution Approach

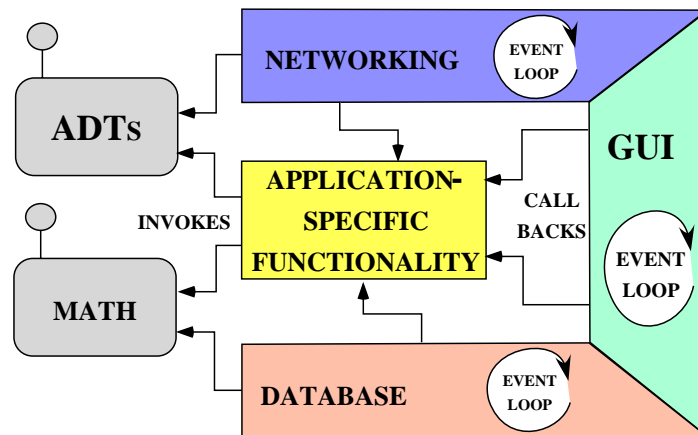
- **Components, Frameworks, Patterns, & Architecture**

Techniques for Improving Software Quality and Productivity

Proven solutions →



(A) CLASS LIBRARY ARCHITECTURE



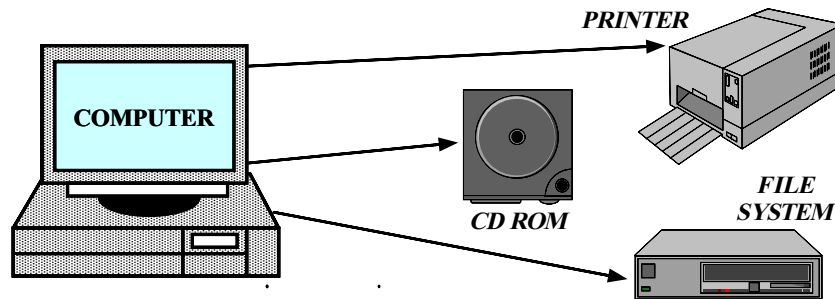
(B) FRAMEWORK ARCHITECTURE

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

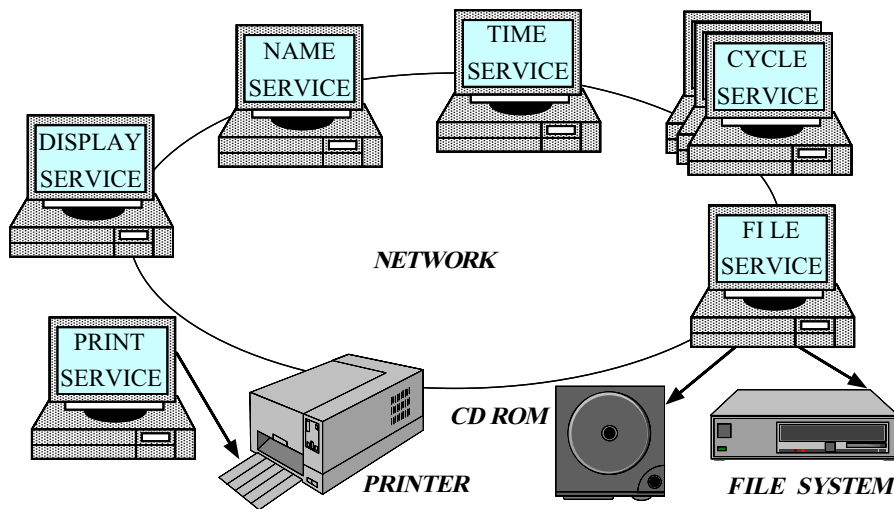
Motivation for COTS Middleware

- It is hard to develop distributed applications whose components collaborate *efficiently, reliably, transparently, and scalably*
- To help address this challenge, the Object Management Group (OMG) is specifying the *Common Object Request Broker Architecture* (CORBA)
- OMG is a consortium of $\sim 1,000$ computer companies
 - Sun, HP, DEC, IBM, IONA, Inprise, Cisco, Motorola, Boeing, etc.
- The latest version of the CORBA spec is now available
 - www.omg.org/technology/documents/formal/
- Other Distributed Computing Architectures
 - EJB (Enterprise Java Beans) by Sun Microsystems - uses CORBA as a complementary architecture.
 - .NET (pervasive computing et.al.) - Microsoft's previous focus was DCOM.

Sources of Complexity for Distributed Applications



(1) STAND-ALONE APPLICATION ARCHITECTURE



(2) DISTRIBUTED APPLICATION ARCHITECTURE

• Inherent complexity

- Latency
- Reliability
- Partitioning
- Ordering
- Security

• Accidental Complexity

- Low-level APIs
- Poor debugging tools
- Algorithmic decomposition
- Continuous re-invention

Sources of Inherent Complexity

- *Inherent complexity* results from fundamental challenges in the distributed application domain
- Key challenges include
 - Addressing the impact of latency
 - Detecting and recovering from partial failures of networks and hosts
 - Load balancing and service partitioning
 - Consistent ordering of distributed events

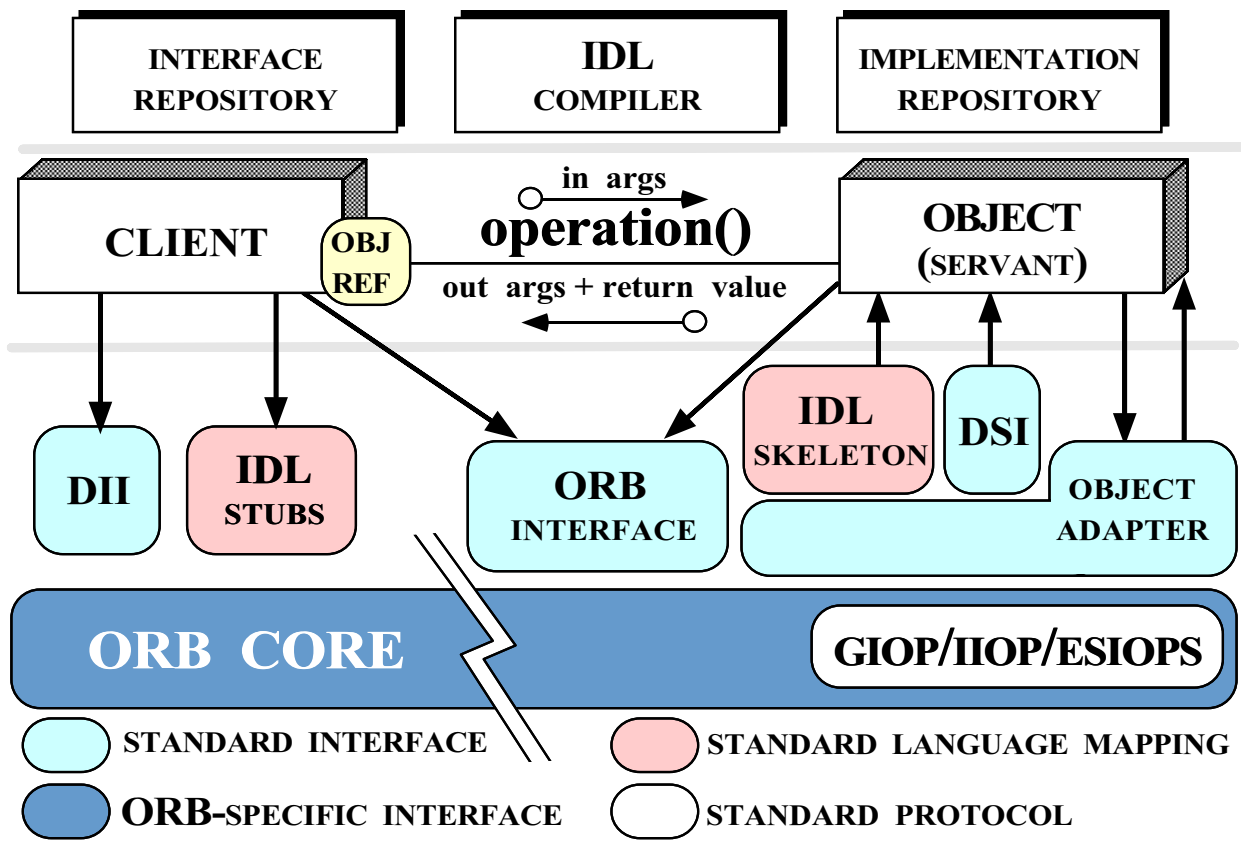
Sources of Accidental Complexity

- *Accidental complexity* results from limitations with tools and techniques used to develop distributed applications
- Key limitations include
 - Lack of type-safe, portable, re-entrant, and extensible system call interfaces and component libraries
 - Inadequate debugging support
 - Widespread use of *algorithmic* decomposition
 - Continuous rediscovery and reinvention of core concepts and components

Motivation for CORBA

- Simplifies application interworking
 - CORBA provides higher level integration than traditional *untyped TCP bytestreams*
- Benefits for distributed programming similar to OO languages for non-distributed programming
 - *e.g.*, encapsulation, interface inheritance, polymorphism, and exception handling
- Provides a foundation for higher-level distributed object collaboration
 - *e.g.*, ActiveX and the OMG Common Object Service Specification (COSS)

Overview of CORBA Middleware Architecture



Goals of CORBA

- Simplify distribution by automating
 - Object location & activation
 - Parameter marshaling
 - Demultiplexing
 - Error handling

- Provide foundation for higher-level services

www.cs.wustl.edu/~schmidt/corba.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

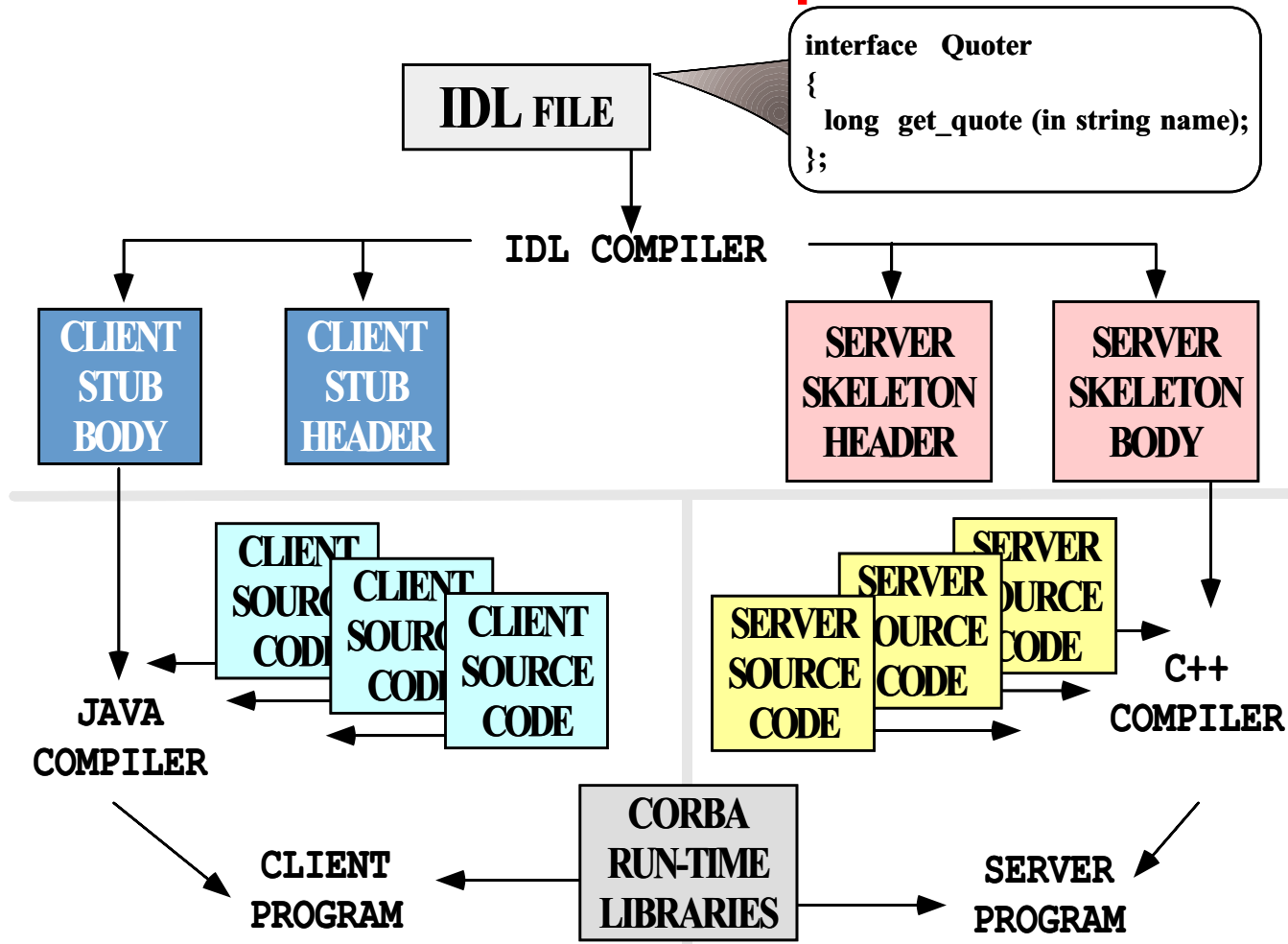
```
// IDL interface is like a C++
// class or Java interface.
interface Quoter
{
    exception Invalid_Stock {};

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

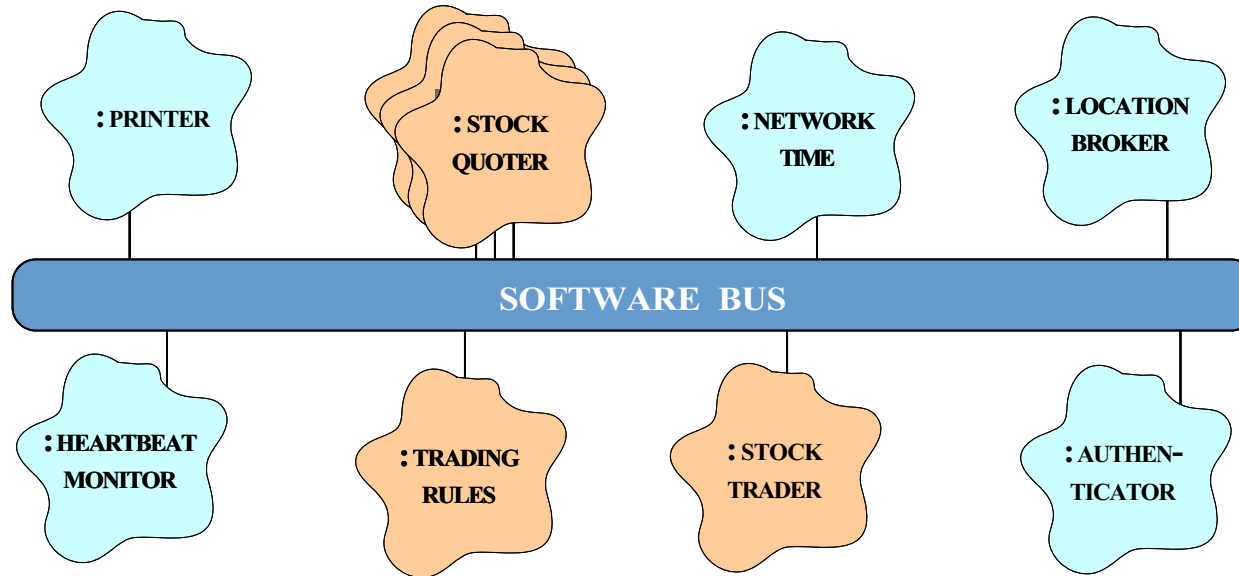
- We write an OMG IDL interface for our Quoter
 - Used by both clients and servers

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

OMG IDL Compiler

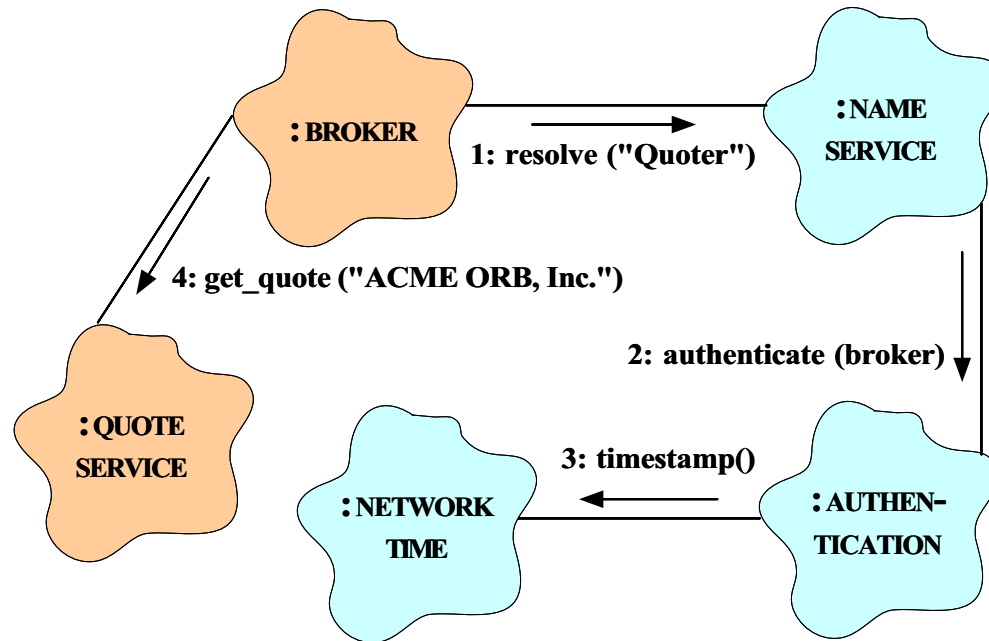


Software Bus



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

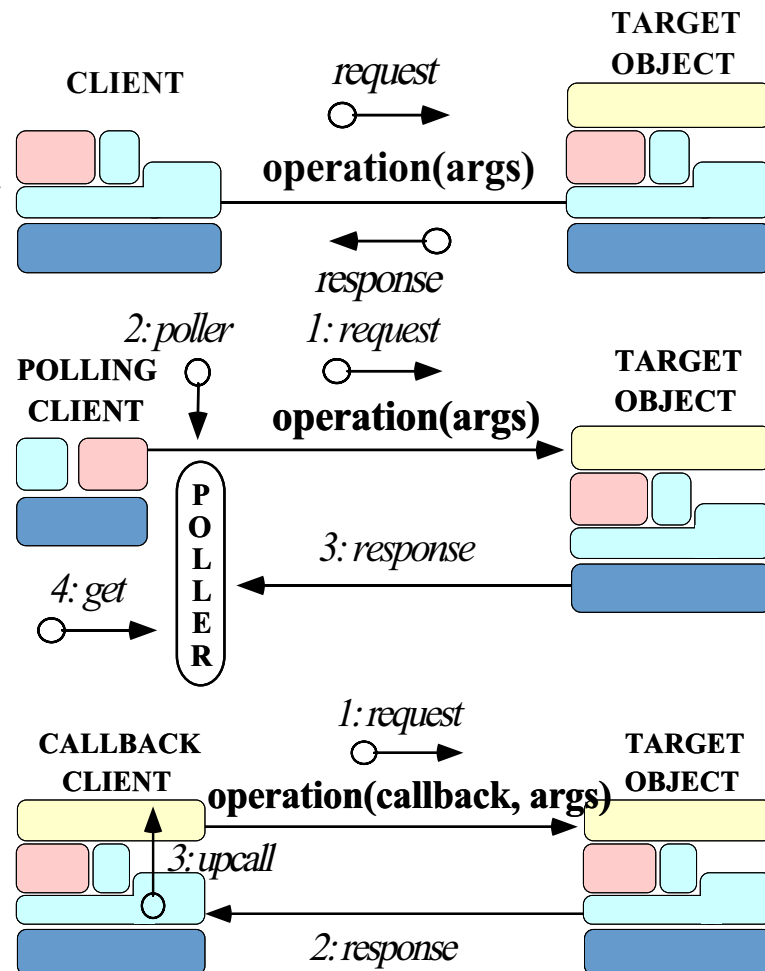
CORBA Object Collaboration



- 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*



Fundamental CORBA Design Principles

- Separation of interface and implementation
 - Clients depend on interfaces, not implementations
- Location transparency
 - Service use is orthogonal to service location
- Access transparency
 - Invoke operations on objects
- Typed interfaces
 - Object references are typed by interfaces
- Support of multiple inheritance of interfaces
 - Inheritance extends, evolves, and specializes behavior

Related Work

- Traditional RPC (*e.g.*, DCE)
 - Only supports “procedural” integration of application services
 - Doesn’t provide object abstractions, async message passing, or dynamic invocation
 - Doesn’t address inheritance of interfaces
- Windows COM/DCOM/COM+
 - 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, such as 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
 - * *e.g.*, can pass both object data and code by value
 - However, many challenging issues remain unresolved
 - * *e.g.*, security, robustness, and versioning

Interface Definition Language (IDL)

- Motivation
 - Developing flexible distributed applications on heterogeneous platforms requires a strict separation of *interface* from *implementation(s)*
- Benefits of using an IDL
 - Ensure platform independence → *e.g.*, Windows NT to UNIX
 - Enforce modularity → *e.g.*, separate concerns
 - Increase robustness → *e.g.*, eliminate common network programming errors
 - Enable language independence → *e.g.*, COBOL, C, C++, Java, etc.

Related IDLs

- Many IDLs are currently available, *e.g.*,
 - OSI ASN.1
 - OSI GDMO
 - SNMP SMI
 - DCE IDL
 - Microsoft's IDL (MIDL)
 - OMG IDL
 - ONC's XDR
- However, many of these are *procedural* IDLs
 - These are more complicated to extend and reuse since they don't support inheritance

CORBA Interface Definition Language (IDL)

- OMG IDL is an object-oriented interface definition language
 - Used to specify interfaces containing *operations* and *attributes*
 - OMG IDL support interface inheritance (both single and multiple inheritance)
- OMG IDL is designed to map onto multiple programming languages
 - *e.g.*, C, C++, Smalltalk, COBOL, Modula 3, DCE, Java, etc.
- OMG IDL is similar to Java `interfaces` and C++ abstract classes

Application Interfaces

- Interfaces described using OMG IDL may be application-specific, *e.g.*,
 - Databases
 - Spreadsheets
 - Spell checker
 - Network manager
 - Air traffic control
 - Documents
 - Medical imaging systems
- Objects may be defined at any level of granularity
 - *e.g.*, from fine-grained GUI objects to multi-megabyte multimedia “Blobs”

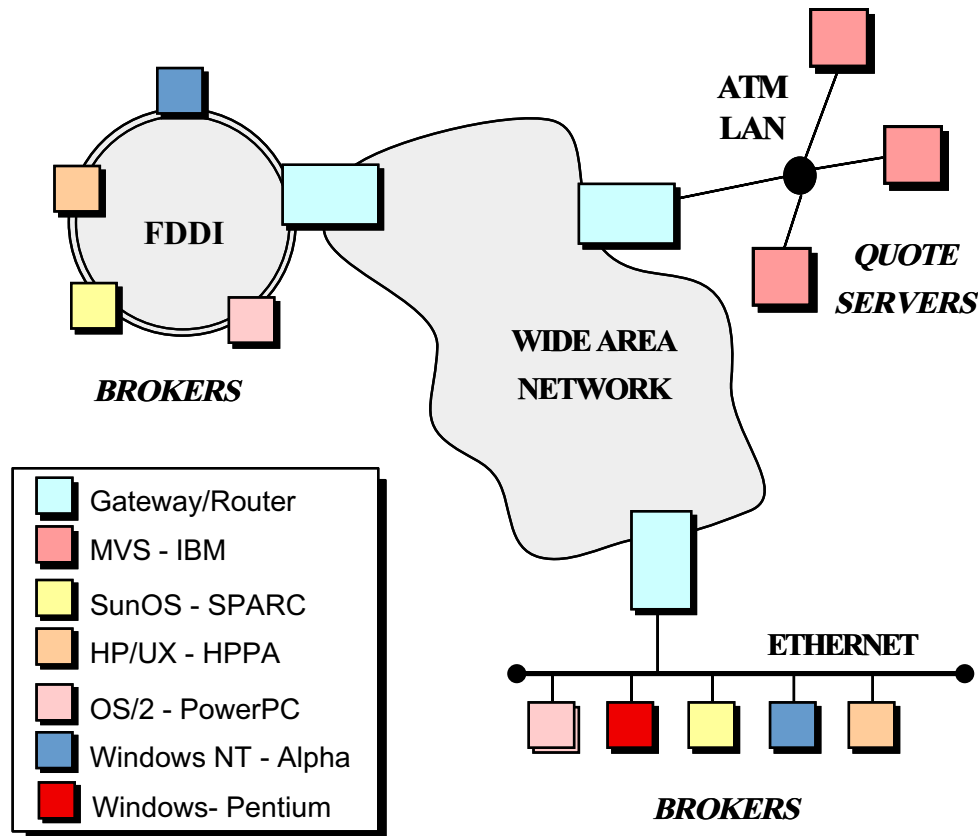
OMG IDL Features

- OMG IDL is similar to Java `interfaces` or C++ abstract classes
 - It is not a complete programming language, however, since it only defines *interfaces*
- OMG IDL supports the following features:
 - `modules` and `interfaces`
 - `Operations` and `Attributes`
 - `Single` and `multiple inheritance`
 - `Basic types` (*e.g.*, `double`, `long`, `char`, *etc.*).
 - `Arrays` and `sequence`
 - `struct`, `enum`, `union`, `typedef`
 - `consts`
 - `exceptions`

OMG IDL Differences from C++ and Java

- No control constructs
- No data members
- No pointers
- No constructors or destructors
- No overloaded operations
- No `int` data type
- Contains parameter passing modes
- Unions require a tag
- Different String type
- Different Sequence type
- Different exception interface
- No templates
- `oneway` call semantics
- `readonly` keyword

CORBA Stock Quoter Application Example



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