Architecture of Software Systems

Distributed Components, CORBA

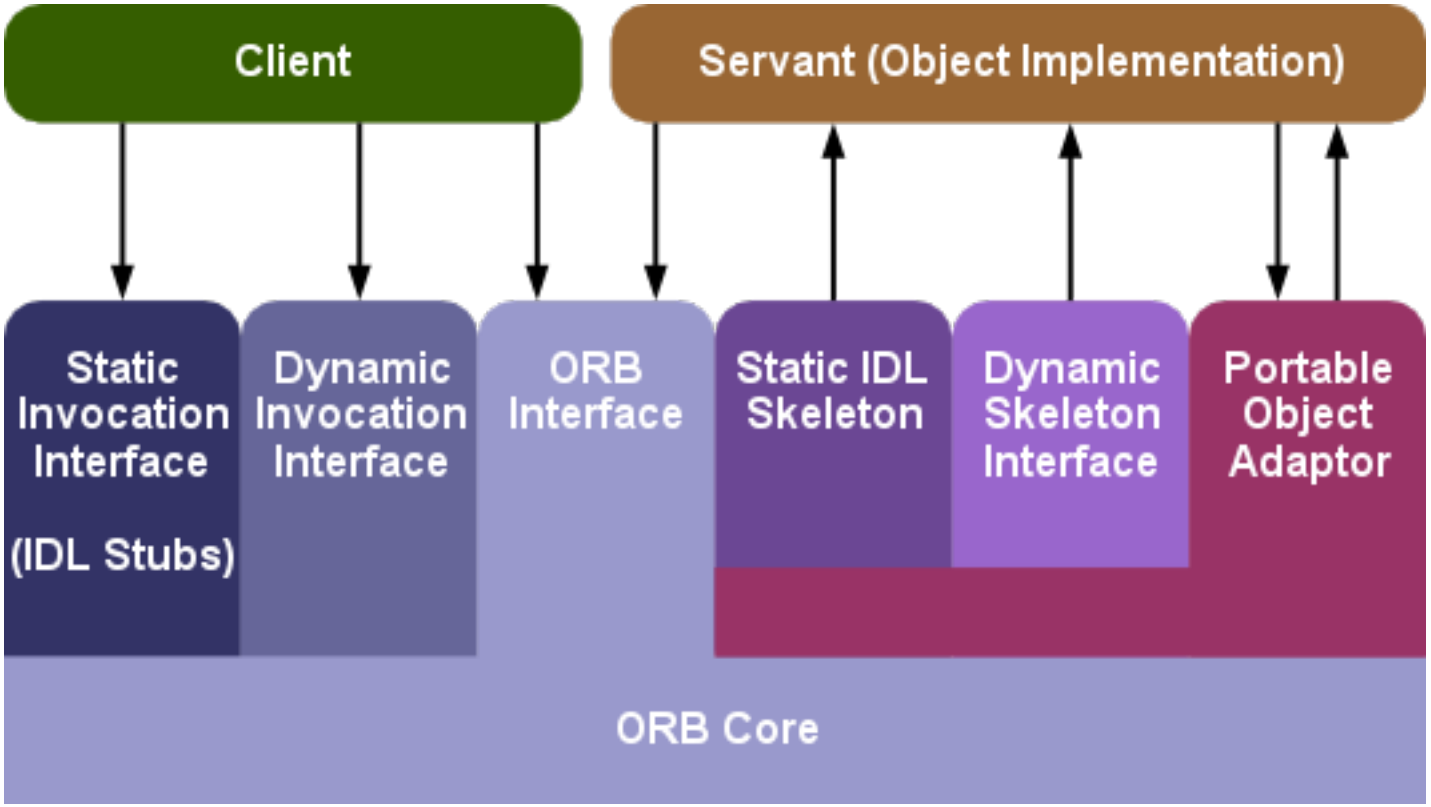
Martin Rehák

CORBA is OMG's open, vendor-independent specification for an architecture and infrastructure that computer applications use to work together over networks. Interoperability results from two key parts of the specification: OMG Interface Definition Language (OMG IDL), and the standardized protocols GIOP and IIOP®. These allow a CORBA-based program from any vendor, on almost any computer, operating system, programming language, and network, to interoperate with a CORBA-based program from the same or another vendor, on almost any other computer, operating system, programming language, and network. [OMG webpage]

CORBA – Distributed Objects

- **Common Object Request Broker Architecture**
- Platform and language independent standard for distributed computing: object management, method invocation, error handling,…
- Deeply influenced subsequent distributed computing frameworks
- Currently rarely used in new projects, but supports/influences many current technologies:
  - IDL
  - IIOP (+ HTTP/SSL version)
  - CORBA Component Model

Overview


Client
Servant (Object Implementation)
Static Invocation Interface
(IDL Stubs)
Dynamic Invocation Interface
ORB Interface
Static IDL Skeleton
Dynamic Skeleton Interface
Portable Object Adaptor
ORB Core

Historical Perspective

- Three versions of standard:
  - CORBA 1: IDL
  - CORBA 2: Interoperability and IIOP
  - CORBA 3: CORBA Component Model

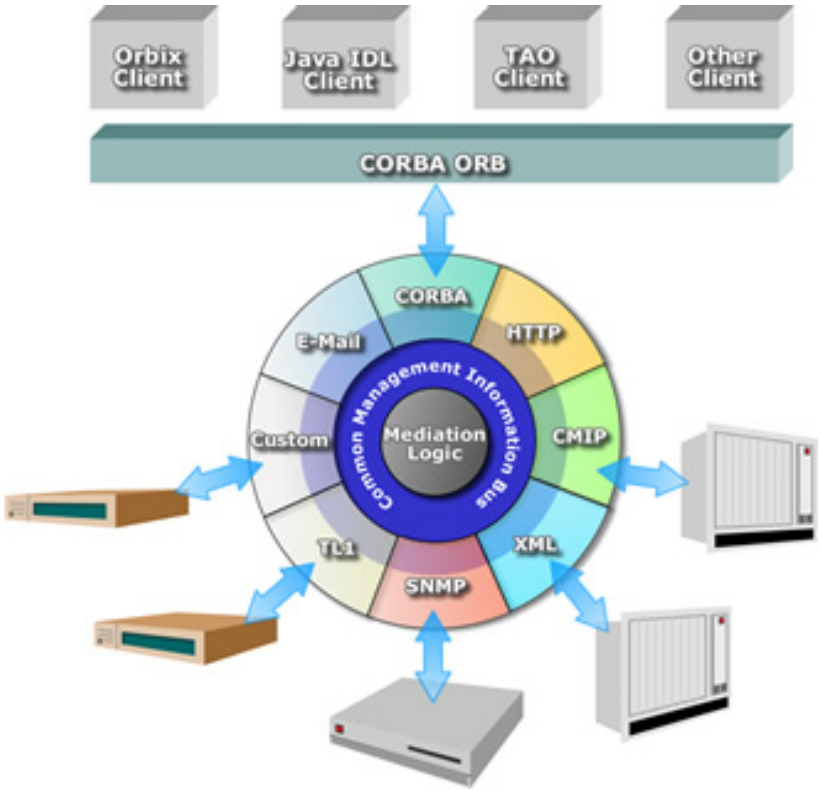
Historical Perspective

- Three versions of standard:
  - CORBA 1: IDL [90ies]
    - *”false start with CORBA 1.0, which was not interoperable and provided only a C mapping”*
  - CORBA 2: Interoperability and IIOP [1997]
    - *“standardized protocol and a C++ language mapping, with a Java language mapping following in 1998.”*
  - CORBA 3: CORBA Component Model [1999]
    - *“The failure of CCM did little to boost the confidence of CORBA customers, who were still stuck with their complex technology.”*

Michi Henning : The Rise and Fall of CORBA, 2006

Design Goals

- Objects and methods can be transparently invoked over networks
- Middleware simplifies development of distributed applications
- Middleware provides Fail-Over, Robustness, Scalability, …
- *"Designing complex applications with simple programmers"* fallacy



Principles

- IDL describes the operations provided by the object both locally and on the server
- Client invokes operations on local proxy
- ORB transmits the request to server location and invokes the operation on remote object
- Servant handles the request on the server side
- ORB manages servant invocation, lifecycle, load-balancing
- ORB then transmits the response back to the client

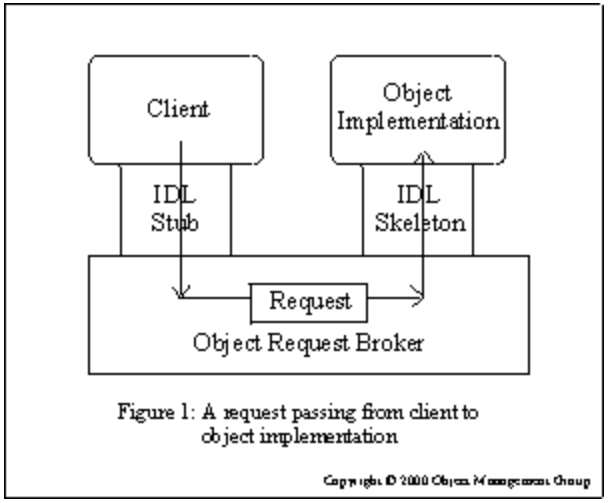


Figure 1: A request passing from client to object implementation

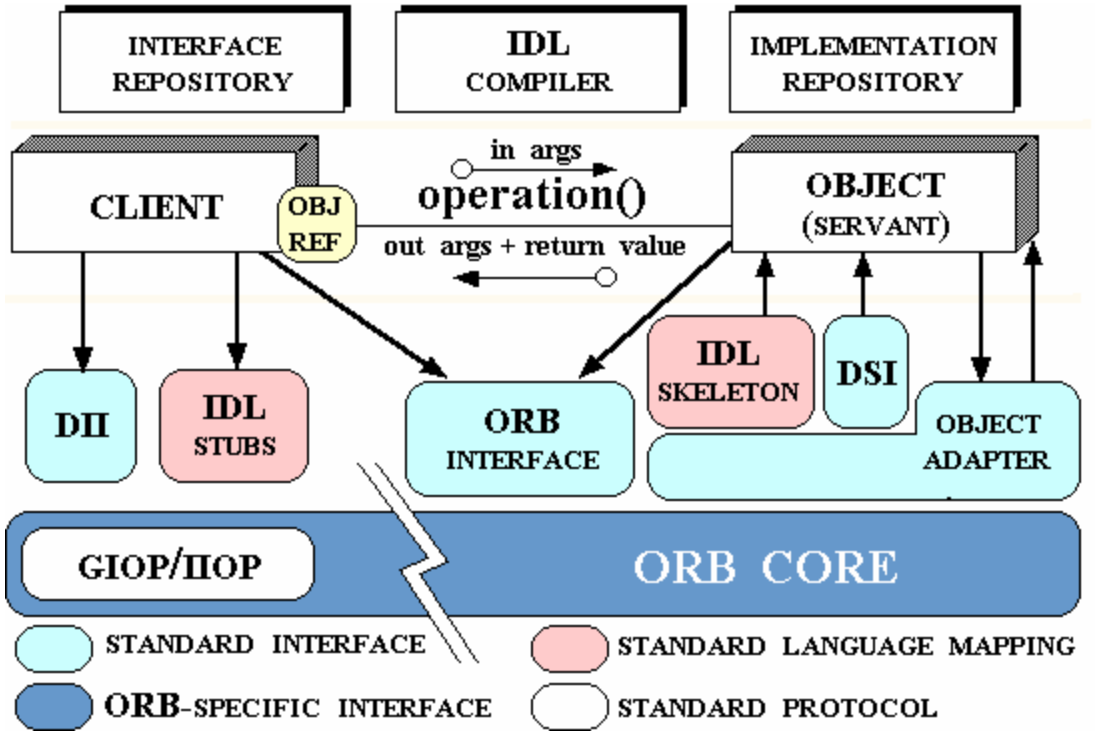
CORBA Services

- Proxy: Location, Platform and Language transparency
  - Request and Response routing
  - Parameter handling
  - Reference management
- Wrapper Façade: Resource allocation transparency
  - Object-Servant Mapping
  - Load Balancing and Fail-Over
- Service/Object/Resource Location
  - Naming service (object registration)
  - Trading service (service/capability registration)
- Dynamic Invocation (Not discussed)

Services - detailed

- **Object life cycle:** Defines how CORBA objects are created, removed, moved, and copied
- **Naming:** Defines how CORBA objects can have friendly symbolic names
- **Events:** Decouples the communication between distributed objects
  Relationships: Provides arbitrary typed n-ary relationships between CORBA objects
- **Externalization:** Coordinates the transformation of CORBA objects to and from external media
- **Transactions:** Coordinates atomic access to CORBA objects
- **Concurrency Control:** Provides a locking service for CORBA objects in order to ensure serializable access
- **Property:** Supports the association of name-value pairs with CORBA objects
- **Trader:** Supports the finding of CORBA objects based on properties describing the service offered by the object
- **Query:** Supports queries on objects

Overview



D. Schmidt (with two next slides as well)

Terminology

- **Object** -- This is a CORBA programming entity that consists of an *identity*, an *interface*, and an *implementation*, which is known as a *Servant*.
- **Servant** -- This is an implementation programming language entity that defines the operations that support a CORBA IDL interface. Servants can be written in a variety of languages, including C, C++, Java, Smalltalk, and Ada.
- **Client** -- This is the program entity that invokes an operation on an object implementation. Accessing the services of a remote object should be transparent to the caller. Ideally, it should be as simple as calling a method on an object, i.e., obj->op(args). The remaining components in Figure 2 help to support this level of transparency.
- **Object Request Broker (ORB)** -- The ORB provides a mechanism for transparently communicating client requests to target object implementations. The ORB simplifies distributed programming by decoupling the client from the details of the method invocations. This makes client requests appear to be local procedure calls. When a client invokes an operation, the ORB is responsible for finding the object implementation, transparently activating it if necessary, delivering the request to the object, and returning any response to the caller.
- **ORB Interface** -- An ORB is a logical entity that may be implemented in various ways (such as one or more processes or a set of libraries). To decouple applications from implementation details, the CORBA specification defines an abstract interface for an ORB. This interface provides various helper functions such as converting object references to strings and vice versa, and creating argument lists for requests made through the dynamic invocation interface described below.

Terminology (2)

- **CORBA IDL stubs and skeletons** -- CORBA IDL stubs and skeletons serve as the ``glue'' between the client and server applications, respectively, and the ORB. The transformation between CORBA IDL definitions and the target programming language is automated by a CORBA IDL compiler. The use of a compiler reduces the potential for inconsistencies between client stubs and server skeletons and increases opportunities for automated compiler optimizations.
- **Dynamic Invocation Interface (DII)** -- This interface allows a client to directly access the underlying request mechanisms provided by an ORB. Applications use the DII to dynamically issue requests to objects without requiring IDL interface-specific stubs to be linked in. Unlike IDL stubs (which only allow RPC-style requests), the DII also allows clients to make non-blocking *deferred synchronous* (separate send and receive operations) and *oneway* (send-only) calls.
- **Dynamic Skeleton Interface (DSI)** -- This is the server side's analogue to the client side's DII. The DSI allows an ORB to deliver requests to an object implementation that does not have compile-time knowledge of the type of the object it is implementing. The client making the request has no idea whether the implementation is using the type-specific IDL skeletons or is using the dynamic skeletons.
- **Object Adapter** -- This assists the ORB with delivering requests to the object and with activating the object. More importantly, an object adapter associates object implementations with the ORB. Object adapters can be specialized to provide support for certain object implementation styles (such as OODB object adapters for persistence and library object adapters for non-remote objects).

IDL: Interface Definition Language

- Platform and language independent
- Specifies only the business level services performed by an object
- Used to generate wrappers and implementation code stubs
- Object-oriented and structured
  - Modules
  - Objects
  - Methods
  - Exceptions
  - Primitive types
  - …

IDL example

```
module StockObjects {
        struct Quote {
                string symbol;
                long at_time;
                double price;
                long volume;
        };
        exception Unknown{};
        interface Stock {
                 // Returns the current stock quote.
                Quote get_quote() raises(Unknown);
                // Sets the current stock quote.
                void set_quote(in Quote stock_quote);
                // Provides the stock description,
                readonly attribute string description;
        };
        interface StockFactory {
                Stock create_stock( in string symbol, in string description );
        };
};
```

IDL-Java Mapping

IDL

- Module
- Interface
- Operation
- Attribute
- Exception

Java

- Package
- Interface
- Method
- Pair of methods (property)
- Exception

Operations on IDL Objects

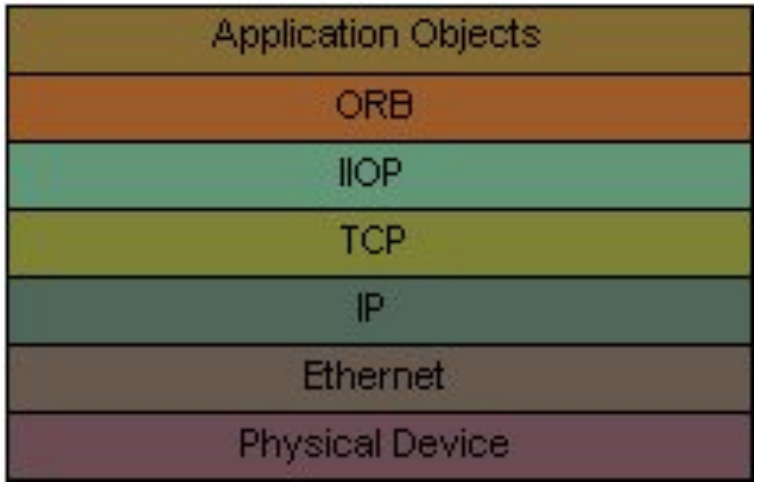
- Inheritance
  Interface Martin: Mother {…};
- Multiple Inheritance
  Interface Martin: Mother, Father {…};
- All CORBA objects inherit from Object
- narrow: cast to a more specific type
  org.omg.CORBA.Object obj = ...
  Stock theStock = StockHelper.narrow(obj);
- is_a: type-checking (interface)
  if (obj.is_a(StockHelper.id())) …
- id(): Interface id valid in a repository
  IDL:StockObjects/Stock:1.0

Object Reference

- Conceptually a mix of URL and pointer features
  - …
- IOR format – opaque for developer
- URI format
  - Reference is locally matched with one or more IDL-defined interfaces
  - Operations on the interfaces are invoked on reference
  - Synchronous invocation of operations
  - Operations are invoked on local stub
- CORBA location:
  - corbaloc::160.45.110.41:38693/StandardNS/NameServer-POA/_root

IIOP Protocol

- Ensures binary compatibility between ORB instances
- Used by other frameworks
  - RMI
  - EJB
  - JADE
  - …
  - Distributed control
- Binary coding
- Language-specific mappings



Writing an Application

- Define the IDL files
- Generate client and server stubs and skeletons
- Implement server objects
  - Ideally *independently* of generated code
    - (Because you will re-generate it again and again…)
- Implement/select Object Adapters
  - Basic object adapters and portable object adapters
- Implement client-side code
- Build
- Start the server and client
- Based on Java/CORBA Tutorial
  - http://java.sun.com/developer/onlineTraining/corba/corba.html#c2

IDL example

```
module StockObjects {
        struct Quote {
                string symbol;
                long at_time;
                double price;
                long volume;
        };
        exception Unknown{};
        interface Stock {
                 // Returns the current stock quote.
                Quote get_quote() raises(Unknown);
                // Sets the current stock quote.
                void set_quote(in Quote stock_quote);
                // Provides the stock description,
                readonly attribute string description;
        };
        interface StockFactory {
                Stock create_stock( in string symbol, in string description );
        };
};
```

Skeleton for server object implementation

```
public class StockImpl extends StockObjects._StockImplBase {
    private Quote _quote=null;
    private String _description=null;
    public StockImpl( String name, String description) {
        super();
        _description = description;
     }
     public Quote get_quote() throws Unknown {
        if (_quote==null)
            throw new Unknown();
        return _quote;
    }
    public void set_quote(Quote quote)  { _quote = quote; }
    public String description() { return _description; }
}
```

Server (simplified !)

```
public class theServer {
        public static void main(String[] args) {
                try {
                        // Initialize the ORB.
                        org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args,null);
                        // Create a stock object.
                        StockImpl theStock = new StockImpl("GII", "Global Industries Inc.");
                         // Let the ORB know about the object
                        orb.connect(theStock);
                        PrintWriter out = new PrintWriter(new BufferedWriter( new FileWriter(args[0])));
                        out.println( orb.object_to_string(theStock) );
                        out.close();
                        // wait for invocations from clients
                        java.lang.Object sync = new java.lang.Object();
                        synchronized (sync) { sync.wait(); }
                }
                catch (Exception e) {
                        System.err.println( "Stock server error: " + e);
                        e.printStackTrace(System.out);
                }
        }
}
```

Patterns: Server Side (CCM)

- Service
  - Stateless, used once, very efficient, single method invocation, temporary object reference
- Session
  - Lasts several calls from the same client. Temporary reference, not registered. (example: Iterators). No persistent storage.
- Process
  - Persistent object reference, registration with naming or trading service. Medium-term process, typically maps to business process.
- Entity
  - Like fully-fledged Entity bean from EJB 2.0. Primary function is persistent entity representation and database mapping.

Discussion: Standardization process

- Vendor Centric process
- Market forces
- Inconsistent vendor implementations/lack of reference implementation
- Complexity
  - *"For example, CORBA's object adapter requires more than 200 lines of interface definitions, even though the same functionality can be provided in about 30 lines—the other 170 lines contribute nothing to functionality, but severely complicate program interactions with the CORBA runtime."*
- Lack of Maturity (Security, Versioning, late specifications)
- Language independence/type inflexibility

- Michi Henning: *The Rise and Fall of CORBA, ACM Queue, June 2006*
  - http://queue.acm.org/detail.cfm?id=1142044