# DISTRIBUTED OBJECTS/COMPONENTS

The 'historical' approach to Distributed Client/Server Applications Software Development:

- Client - application logic: e.g. SQL client for data access, VC++ or VB for user interface.
- Server - based on relational databases, e.g. SQL server - Oracle, Ingres, Powerhouse, Informix, Sybase, etc.
- Add a transaction processing monitor on server to support multiple clients.
- Communications - e.g. RPC, TCP/IP sockets, etc. Too low a level of abstraction - poor encapsulation, ease of use, etc.
- Web specific - e.g. HTTP/CGI - very slow

Distributed Object approaches takes the client/server model further by decomposing both the client and server into multiple objects and increasing flexibility.

Distributed Objects need to be able to:

- migrate between nodes
- operate anywhere
- encapsulate "legacy" code

→ the complexity of distributed objects needs to be efficiently managed through the development of a standard architecture.

Object Management Group (OMG) - founded April 1989 to promote: "Interoperability for applications integration through co-operative creation and promulgation of object-orientated standards based on commercially available software".

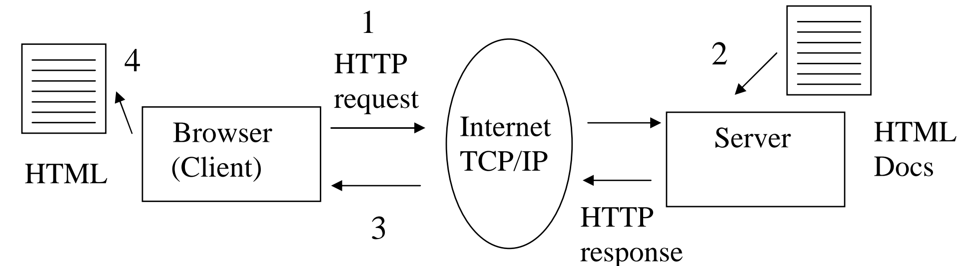
OMG sees application integration and distributed processing as the same problem → both need to support heterogeneous, networked, physically distributed and multi-vendor systems sharing information.

There are a number of ways the aims of OMG have been implemented, apart from the offically sponsored CORBA, e.g. Microsoft's DCOM, Java RMI -> need to look at key features of these approaches.



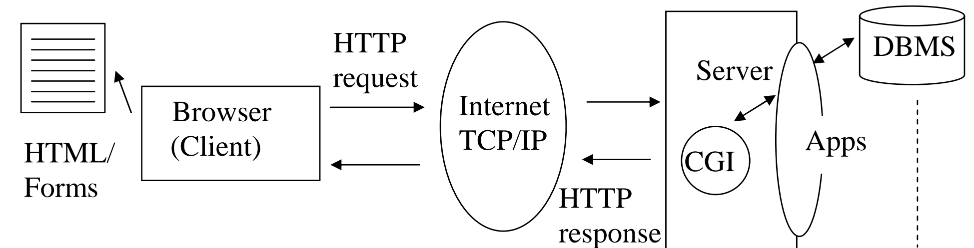
# HTTP/CGI

Before Java/CORBA on WWW there was the *Hypertext Transfer Protocol* (HTTP) which provides RPC-like operations on top of a Sockets layer. Pages in the *Hypertext Markup Language* (HTML) are located via *Universal Record Locators* (URLs) which uniquely identify all WWW resources. A typical transfer is shown here:

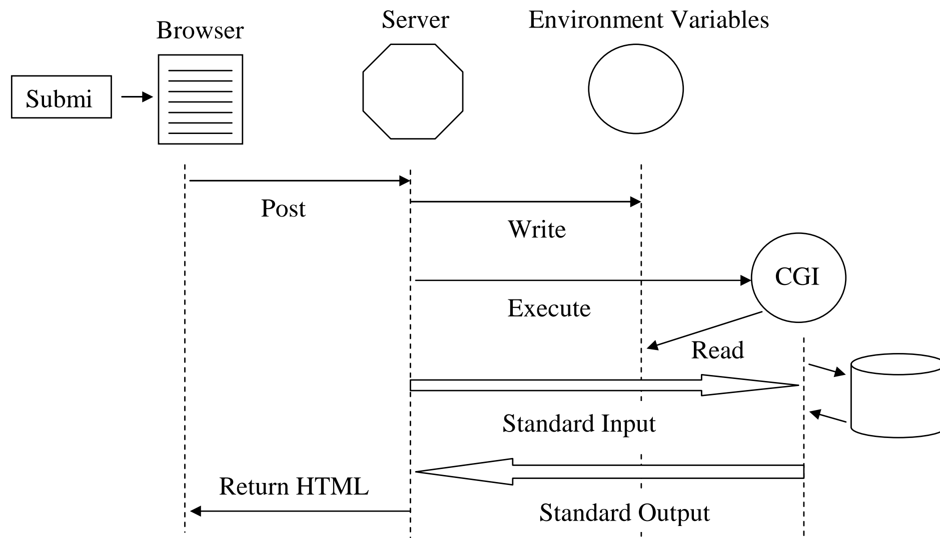



HTTP is very simple, it provides a stateless TCP/IP connection that is dropped at the end of request and it does not allow multiple parallel requests. It does support variable data representations which are negotiated between client and server every connection.

To provide a 3-tier client-server model HTTP requires data to be collected on the client and sent to the server - the *Common Gateway Interface* (CGI) provides for this. An HTML form on the client provides an interface to collect data contents to be sent to the server. The server requires a back-end program to interpret the data and it is passed via the CGI. Data is returned from this program in HTML format via the CGI:



## Example: HTTP Post - End-to-end Client/Server




## Performance Issues

- Stateless connection - impact on multiple forms. Requires hidden fields within a form to maintain the state on the client -> big overhead.
- The POST HTTP method is usually preferred over the GET method as GET appends the entire form contents after the URL (after a ?) whereas POST appends it to the body of the HTML message. For a GET, the HTTP server parses the URL and puts the form contents into a variable and the space for this may be very limited in the environment area -> it is truncated or OS problems.
- Ping - 600 msec (Pentium 120, NT4.0, 10Mb/s Ethernet) compared with appox. 2 msec for TCP/IP and 3 msec for CORBA.

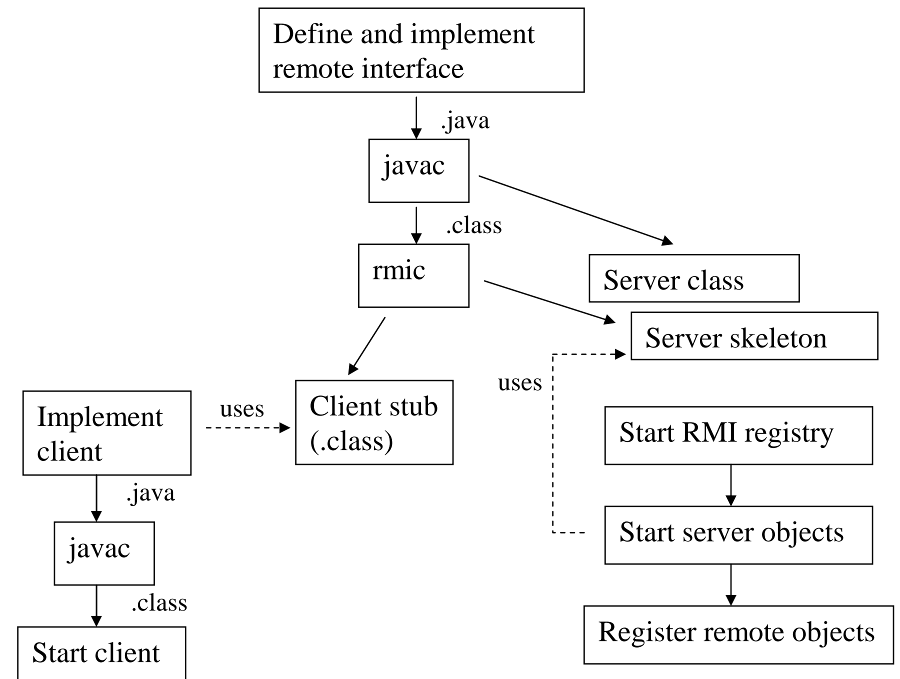


## Java RMI

*Remote Invocation Method* (RMI) supports remote method invocation for objects across Java virtual machines. It transparently allows remote RMI objects to be accessed as if they were local Java objects.

RMI invocation passes local objects by *value* rather than by *reference* and to achieve this an *object serialisation* service is used to 'flatten' a local Java object's state into a serial stream that can be passed as a parameter inside a message. RMI also extends the Java exception classes to support remote failures of objects.

## RMI development process




**Issues:** Provides an ORB, but no multilanguage support and limited scaling. Performance - ping (P120, NT4, 10Mbit/s) approx. 6 msec.

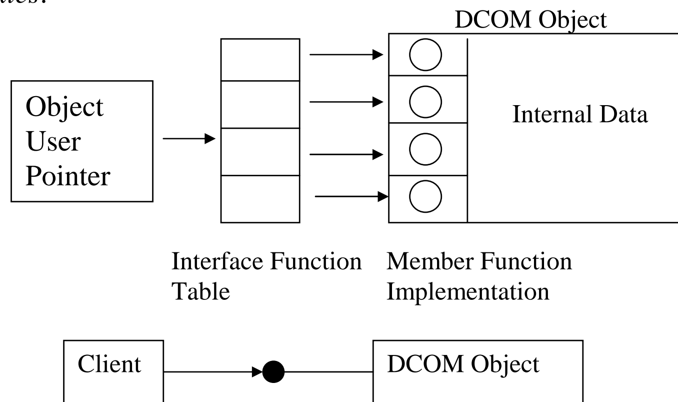
## Microsoft's DCOM

Microsoft's *Distributed Component Object Model* (DCOM) is the main competition to CORBA. DCOM is supplied with Windows NT4.0 and Windows 98. *ActiveX* is a DCOM object. Microsoft have integrated Java with DCOM by providing DCOM bindings in Visual J++ so that remote Java objects can be invoked along with ActiveX components.

DCOM separates object interfaces from implementation through an IDL based on Microsoft's DCE (Distributed Computing Environment). Importantly DCOM objects are not objects in the OO sense (with a state that can be maintained between connections). DCOM also doesn't support IDL-specified multiple inheritance (but it supports multiple interfaces which supports reuse).

## DCOM interfaces

DCOM interfaces are language independent 'contracts' between client and server - in effect it provides a binary interoperability specification for how clients are to access server interfaces via *pointers* and *remote proxies*:

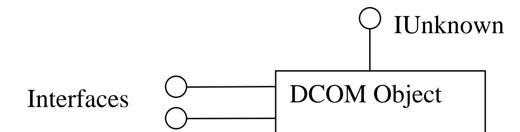



At run time, each interface has an *Interface Identifier* (IID) which is a DCOM-generated 128-bit *Globally Unique Identifier* (GUID). Clients use the GUID to query objects about interfaces (which are the smallest possible contracts between clients and servers).



## DCOM Objects

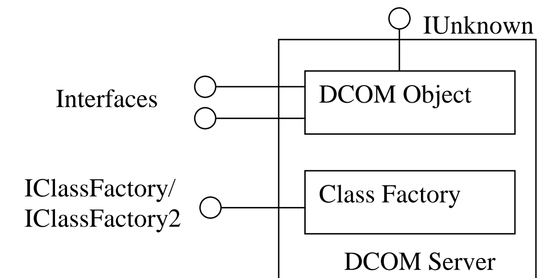
A DCOM (or *ActiveX*) object is a component that supports multiple interfaces defined by that object's class. A DCOM class implements one or more interfaces and is identified by a unique 128-bit *Class ID* (CLSID). A standard *IUnknown* interface is used to negotiate interfaces and allocate pointers to them.



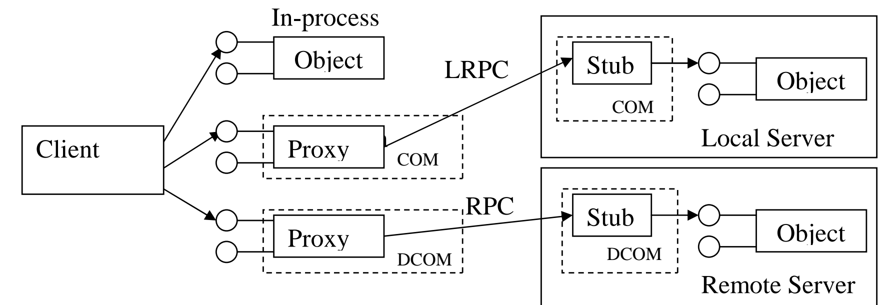

Note that DCOM does not have unique object IDs since objects are accessed by transient pointers to interfaces (consequently object instances are less important in DCOM than in more classical object models).

## DCOM Servers

DCOM servers house object classes (with CLSIDs) that are used to create objects via a *Class Factory*:




Clients can communicate with servers transparently through proxies for out-of-process remote servers or local servers:

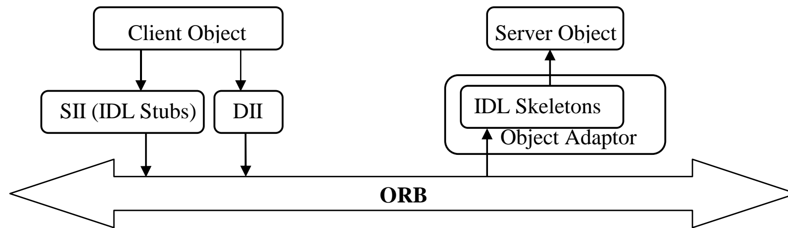



**Issues:** Provides an ORB, multilanguage support (Microsoft) and arguable scaling. Performance - ping (P120, NT4, 10Mbit/s) 3.9 msec.

# OMG CORBA

## Overview

The Common Object Request Broker Architecture (CORBA) - a software bus architecture - where the interface is separated from implementation and the bus is common to all nodes.




The key components in the CORBA specification are:

- The Object Request Broker (ORB) - the software bus, or infrastructure that allows objects to access each other.
- The Interface Definition Language (IDL) and associated interfaces to the ORB - supports object implementation that is totally independent of the interface.

To support the invocation of objects there are two interfaces provided in CORBA:

- The Static Invocation Interface (SII) - supports remote object invocation through a syntax that is natural for the implementation language of the invoking object (e.g. for C++ a remote object "jump" could be called with `myobject -> jump()`)
- The Dynamic Invocation Interface (DII) - where complete knowledge about the remote object is not known at compile time (which would allow static invocation to be used). The API for DII is more complex than the SII, but supports run-time binding.



The CORBA Standard:

- specification 2.0 released - late 1995.
- supports multiple language mappings - C, C++, Ada, SmallTalk, COBOL and Java.
- current services: naming, event management, transactions, lifecycle, security, relationships, persistent objects, concurrency externalization; with activity on extensions for queries, time management, change control, licensing, etc.
- current facilities: basic system management services; with activity in defining specific business objects .

Some major CORBA implementations:

- Orbix - IONA
- Joe/NEO - SUN
- Orb Plus - HP
- SOM - IBM
- ObjectBroker - DEC
- VisiBroker - Visigenic
- Java 2 - subset ORB

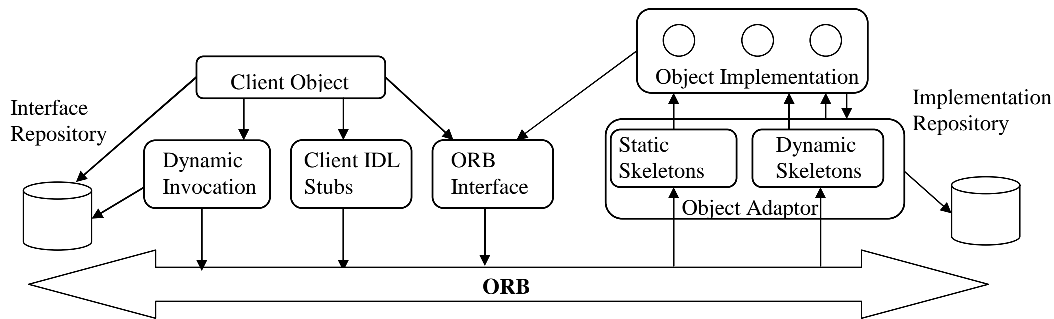
## CORBA ORB

The purpose of the ORB is to allow a client object to invoke a method on a server object anywhere on the ORB - note that the client/server relationship is limited to coordination of the interaction and is **not** a function of the objects.

The CORBA 2.0 ORB provides *global identifiers* or *Respository IDs* to globally and uniquely identify components - they are generated by IDL pragmas and can be various format strings (most commonly in a three level naming hierarchy IDL):

- object name
- major version
- minor version numbers

DCE and local user formats are also supported.



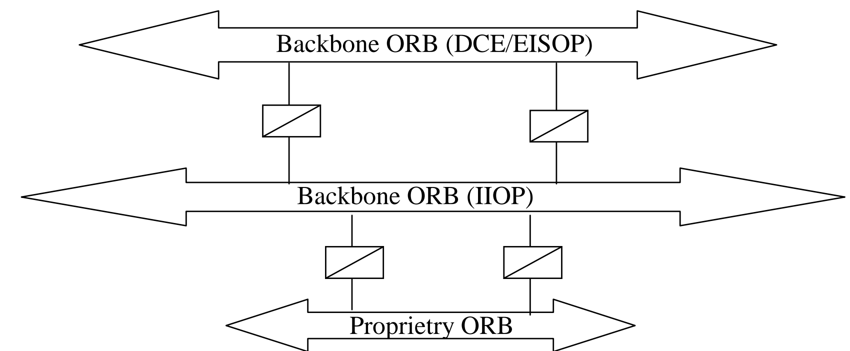

CORBA components:

- Client IDL stubs: precompiled stubs that define how clients invoke services on the servers. Services are defined by IDL, and these stubs are generated by the IDL compiler. One of the stub functions is *marshalling* - encodes and decodes operations and parameters into 'flattened' messages to be sent to the server.
- Dynamic Invocation Interface (DII): allows server interfaces to be looked up at run-time (via metadata) and interacted with.
- Interface Repository APIs: provides information on component interfaces, supported methods and required parameters.
- ORB Interface: utility function APIs for converting object references to/from strings.
- Server IDL stubs (or *skeletons*): static interfaces to services generated by the IDL compiler.
- Dynamic Skeleton Interface (DSI): run-time binding mechanism for components that do not have IDL-based compiled skeletons. It interprets incoming requests for target objects and methods.
- Object Adapters: provides a run-time environment for instantiating server objects, passing requests to them and assigning *object references* (or IDs). All ORBs must support a *Basic Object Adapter* (BOA).
- Implementation Repository: stores information about server classes (from which its objects are instantiated), administrative, audit and security information.



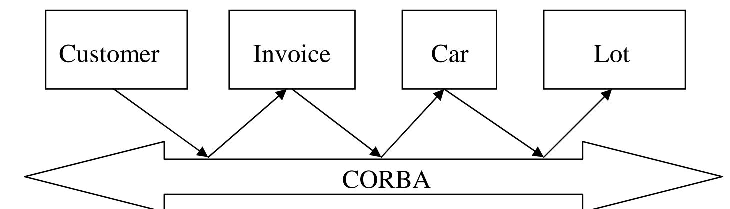
**Inter-ORB architecture**

One of the main strengths of CORBA 2.0 is the addition of interoperability specifications between ORBs through a mandatory *Internet Inter-ORB Protocol* (IIOP). It is essentially TCP/IP with some CORBA-defined messages to support a common backbone. All CORBA compliant ORBs must support IIOP or provide a *half-bridge* to it, i.e. some other *Environment-Specific Inter-ORB Protocol* (ESIOP) to IIOP. An example ESIOP specified by CORBA 2.0 is OSF's DCE for mission critical ORBs:



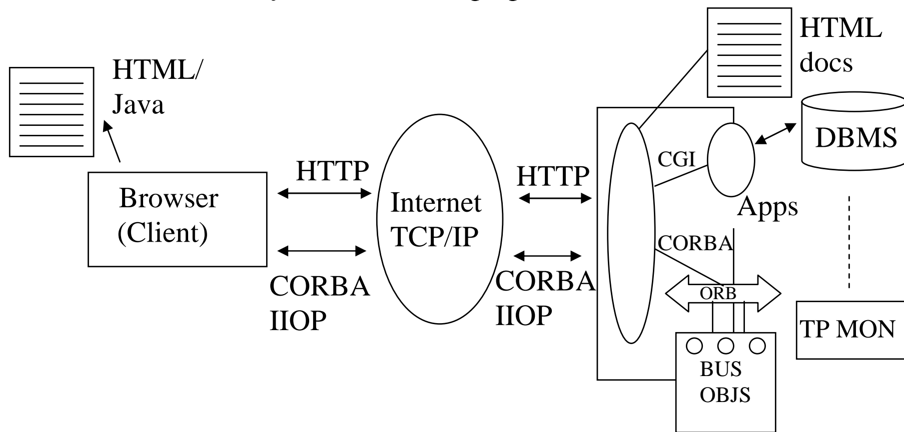

**CORBA Cooperating Business Objects**

Business objects are self-contained applications with a user interface, a state, and knowledge about cooperative processing with other business objects. They are top-level components that closely mimic business processes and are recognized by the end-user of the system.
Business objects have late and flexible binding and well-defined interfaces:



## The Object Web

A major limitation of the existing client/server platform for WWW, HTTP/CGI, is its speed of operation and lack of flexibility. CGI is a poor match for OO Java clients, and various attempts to extend CGI with proprietary servers has limited chance of success (e.g. Netscape's NSAPI, Microsoft's ISAPI, etc). CORBA provides a vital link between the Java portable application environment and a multitude of back-end services. A 3-tier *Object Web* is emerging:




Java clients communicate directly with CORBA objects as IIOP replaces HTTP/CGI, but both HTTP and IIOP can share the same network -> HTTP for HTML page download and IIOP for client-server interactions.

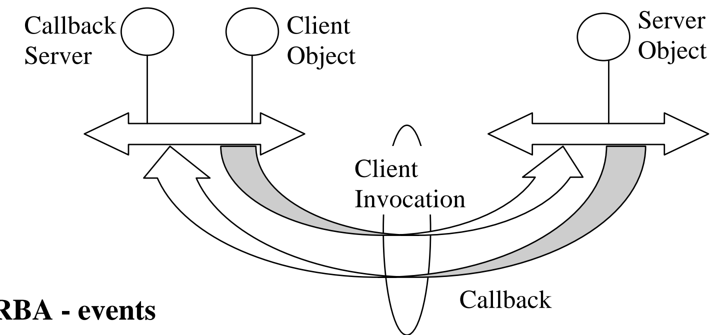
CORBA benefits:

- Avoids CGI bottleneck - direct invocation of methods on server with minimal client/server overhead.
- Scalable server-to-server structure - business objects on servers can communicate easily via the ORB -> multiple servers to handle the load
- Extension of Java applet communications across address boundaries, languages and networks -> augments Java with range of object services, e.g transactions, security, trading.



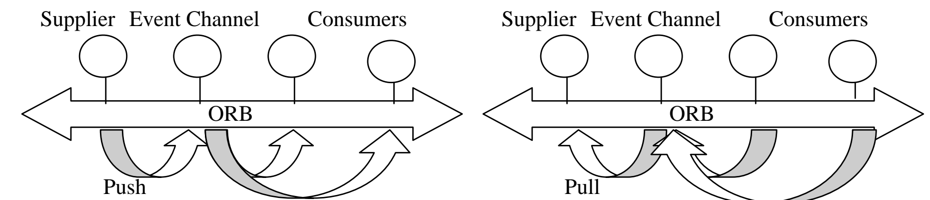
## CORBA - callbacks

To generalise role reversal for client/servers, servers can invoke callbacks on clients when some urgent event occurs or synchronisation is required:




## CORBA - events

An *event service* allows objects to dynamically register interest in specific events - objects can dynamically assume supplier or consumer roles. The two models supported for event communication are *push* and *pull*:

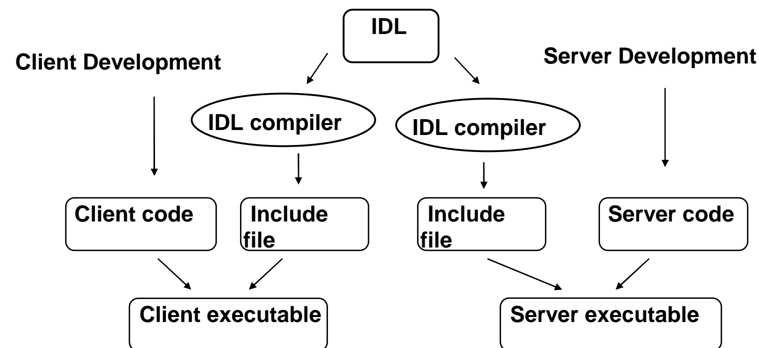



An *event channel* is a CORBA object that is a supplier and consumer of events:

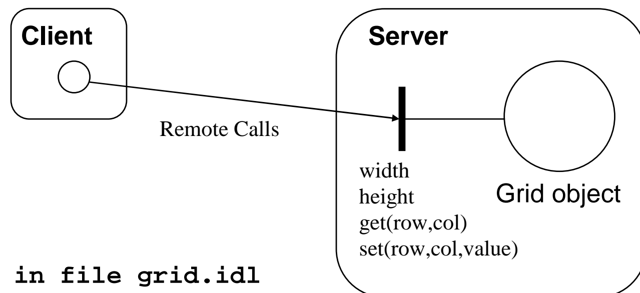
- push: consumers must register interest in an event channel with a *connect_push_consumer* method (and stop receiving events with *disconnect_push_consumer*).
- pull: consumers can use the *try_pull* method to poll for events. Suppliers can offer events on the channel with an *add_pull_supplier* method and suspend events with *disconnect_pull_supplier*. *Point-to-point* events are also supported (without intervening event channels) as are *event proxies* to further decouple suppliers and consumers of events.

## CORBA application example

- Define all objects and their IDL interfaces
- Write clients that will use the objects
- Write servers which provide the objects
- Registers all servers with the ORB




## Example: grid object




```
// IDL in file grid.idl
interface Grid {
  readonly attribute short height;
  readonly attribute short width;
  void set(in short x, in short y, in long value);
  long get(in short x, in short y);
}
```

The IDL compiler is run to produce the header file grid.hh



```
// Client.c
#include "grid.hh"
main() {
  Grid_var p;              // Just like a pointer
  // Connect to a remote Grid object
  p = Grid::_bind("myGrid:GridSrv", GridHost);
  // Can now use it like a local object
  cout << "height is " << p->height() << endl;
  cout << "width is " << p->width() << endl;
  p->set(2,4,123)          // do a remote call
  cout << "grid(2,4) is " << p->get(2,4) << endl;
}
```

```
// Client.c with error handling
. . .
try {
  p = Grid::_bind("myGrid:GridSrv", GridHost);
catch(CORBA::SystemException& ex) {
  cerr << "A CORBA error occurred: "<< &ex << endl;
  exit();
}
```

Implement the Server Object Interface:

```
. . .
// C++ in file grid_i.h
class Grid_i : public virtual GridBOAImpl {
. . .
public:virtual void set (CORBA::short n, CORBA::short m,
  CORBA::long value, CORBA::Environmental&)
```

Implement the Server:

```
// Server.c
#include "grid_i.h"
main() {
  Grid_i myGrid(100,100); // Create Grid object
  try {  // give control to the ORB
    CORBA::Orbix.impl_is_ready("myGrid:GridSrv");
  catch(CORBA::SystemException& ex) {
    cerr << "A CORBA error occurred: "<< &ex << endl;
    exit();
  }
}
```

Note that the IDL header file (grid.hh) must be available to the client with a static interface invocation (SII). CORBA also provides a dynamic interface invocation (DII), e.g:

```
// Client.c - build request dynamically
  . . .
  long result;
  short x = 2;
  short y = 4;

  CORBA::Object_var target =
    CORBA::Object::_bind ("myGrid:GridSrv", GridHost);
  CORBA::Request r(target, "get");

  // Stream in the arguments, and make call
  r << CORBA::inMode << x << y;
  r.invoke();
  r >> result; // get result
}
```

## CORBA 3

Started development in 1999 with OMG adoption in 2001 - it provides more comprehensive support for software components, with some notable features being:

- Internet: Corba firewall specification - allows IIOP's to more easily navigate through corporate firewalls.

- Messaging: Message orientated middleware (MOM) specification - cleaner asynchronous request protocol so clients don't need to be multithreaded or active during the transaction.

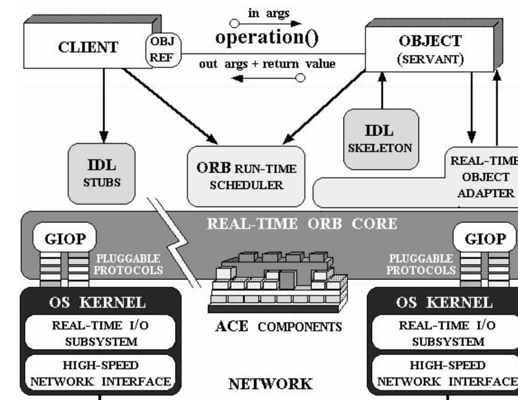
- Components: superset of Enterprise Java Beans (EJBs) for language neutral component containers.

- Portable Servers: more flexible BOA that better supports transient objects - requires an *instance manager* for each object type (which creates and destroys server objects, or *servants*).



**Real-time CORBA** (www.omg.org)

Specification V1.0 for Real-time CORBA was first released by OMG in 1999 – the key aim was to provide support for "end-to-end predictability" by supporting client and server priority assignments and bounding the latencies of all operation invocations. An early and current open-source implementation compliant with the specification is the ACE ORB (TAO) which is built on the Adaptive Communication Environment (ACE).




[TAO, 2000]

Key TAO Features:

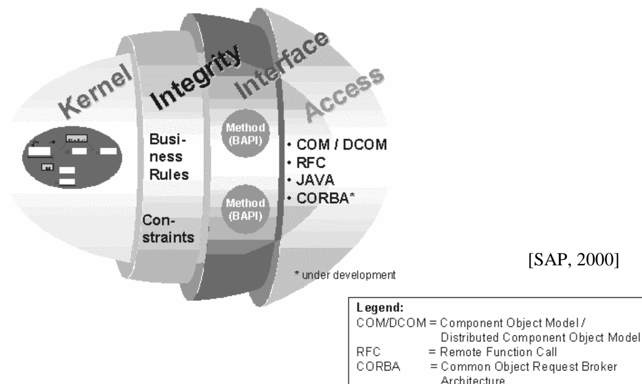
- Optimised IDL Stubs & Skeletons: High performance and flexible (compiled or interpreted) (de)marshalling
- RT Object Adaptor: Dispatches servant operations in $O(1)$ time regardless of the number of active connections
- Run-time Scheduler: Map application QoS requirements into hardware resource requirements
- RT ORB Core: Uses a multi-threaded, pre-emptive priority-based connection and concurrency architecture and allows other plug-in ORB protocols
- RT I/O Subsystem: Assigns priorities to RT I/O threads so the schedulability of applications can be met
- Built on ACE which provides a QoS framework: Support for a wide variety of OS's

**SAP R/3 System** (www.sap.com)

The SAP R/3 System is based on the concept of *business objects*. Real world objects, for example an employee or a sales order, are modelled as business objects in business application systems. It provides an integrated and scalable solution for business orientated client/server and open distributed systems.

SAP Business Objects can be seen as encapsulating data and business processes, thus hiding the details of the structure and implementation of the underlying data. To achieve this SAP Business Objects are constructed as entities with multiple layers:

- At the core of an SAP Business Object is the kernel, which represents the object's inherent data.
- The second layer, the integrity layer**,** represents the business logic of the object. It comprises the business rules and constraints that apply to the Business Object.
- The third layer, the interface layer, describes the implementation and structure of the SAP Business Object, and defines the object's interface to the outside world.
- The fourth and outermost layer of a Business is the access layer, which defines the technologies that can be used to obtain external access to the object's data.




[SAP, 2000]



## The Business Object Repository (BOR)

All SAP Business Object types and their methods are identified and described in the R/3 Business Object Repository (BOR).

The BOR contains two categories of object types:

- **Business object types**

  The SAP Business Object types are arranged within the BOR in a hierarchical structure based on the R/3 business application areas, such as sales, material management, etc.

- **Technical object types**

  These are items such as texts, work items, archived documents, as well as development and modelling objects.

The BOR is the central point of access to the SAP Business Objects through their access methods for external applications.

## Business Application Program Interfaces (BAPIs)

External access to the data and processes held in the BOR is only possible by means of specific methods: BAPIs

e.g. The functionality that is implemented with the SAP Business Object type "Material" includes a check for the material availability. Thus, the Business Object type "Material" offers a BAPI called "Material.CheckAvailability".

To invoke a BAPI from an application program only the interface information needs to be supplied. A BAPI interface is defined by:

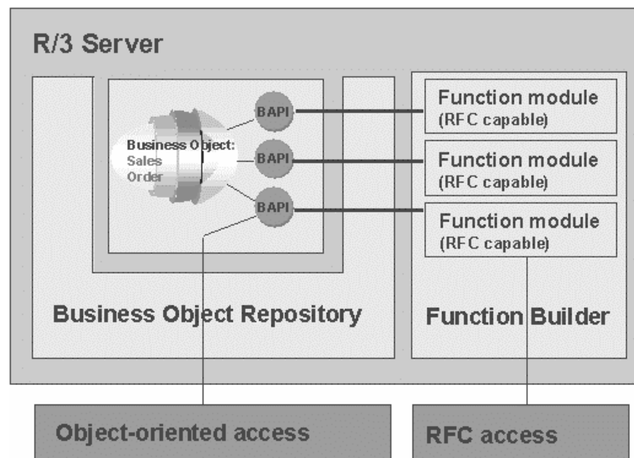
- Import parameters, which contain data to be transferred from the calling program to the BAPI
- Export parameters, which contain data to be transferred from the BAPI back to the calling program
- Import/export (table) parameters for both importing and exporting data

## Accessing BAPIs

BAPIs are defined as methods of SAP Business Objects in the Business Object Repository (BOR) and are implemented as function modules. The separation of a BAPI definition from its actual implementation allows a BAPI to be accessed in two ways:

- The BAPI can be called in the BOR, e.g. for the Windows 95/NT a BAPI ActiveX Control allows external client applications to access the SAP Business Objects in the BOR by invoking BAPIs through OLE Automation. BAPIs can also be invoked using SAP's BAPI C++ Class Library through member functions of C++ proxy classes. Access via a Java Class Library and Delphi is also supported.

- RFC calls can be made to the function module on which the BAPI is based, e.g. by using the C/C++ RFC class libraries.




[SAP, 2000]



## BAPI example

There are a number of BAPIs that provide services across the whole range of SAP Business Objects, exact functionality being determined by the SAP Business Object for which it is implemented.

**GetList BAPI**

This BAPI is used to search for object instances that fulfill certain selection criteria. An example is *CompanyCode.GetList*, which returns a collection of company codes, e.g:

A local object instance with an empty key field is created using the BAPI Control object:

*Set oCompanyCode = oBapiControl.GetSAPObject("CompanyCode")*

Then the BAPI *CompanyCode.GetList* with the parameters *Return* and *CompanyCodeList* is invoked and a table is provided for the company codes listing:

*oCompanyCode.GetList Return:=oReturn, CompanyCodeList:=otabCompanyCodes*

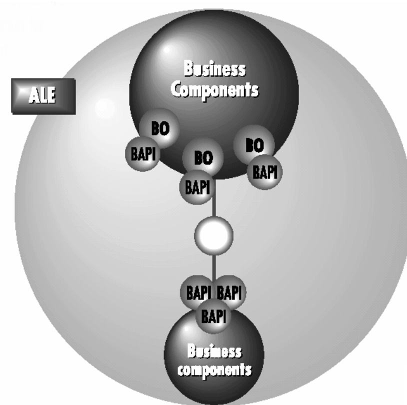
## Functional Modules

One of the strengths of the SAP R/3 systems is the large set of predefined Business Object Functional Modules that can be used stand alone or easily integrated:

- Logistics: Sales & Distribution, Production Planning & Control, Project, Materials and Quality Management, Plant Maintenance, Service and Product Management.
- Financial: Accounting, Control, Investment Management, Treasury, Enterprise Management.
- Human Resources: Personnel and Organisational Management, Payroll, Time Management and Personnel Development.

## Business Frameworks

SAP Functional modules can be encapsulated as Business Components with access provided by Business Objects through their corresponding BAPIs.  Business Objects are designed to change at a much slower rate than the underlying software technologies, so they provide a stable interface to Business Components.  SAP uses Application Link Enabling (ALE) to permit cross-component mapping of business processes.




[SAP, 2000]

## E-commerce Integration

MySAP.com was introduced in mid-2000 to provide a preconfigured enterprise portal to interface with SAP R/3 so as to better support web-based B2B transactions. In particular, mySAP preconfigured "Marketplaces" support the development of "e-business hubs" to allow suppliers and customers to interact through automated auction and bidding systems.



## SOAP (Simple Object Access Protocol)

Developed originally by Microsoft, but now submitted for W3C standardization as XP (XML protocol),  SOAP allows XML (eXtensible Markup Language) messages to be exchanged between enterprise portals particularly for B2B data exchange. SOAP allows an enterprise to structure business data as XML messages - the specification supports two formats:

- Self describing for EDI (Electronic Data Interchange) data exchange
- RPC (Remote Procedure Call) style interactions for object method invocation

SOAP message formats:

- Envelope - XML schema describes the message format and serialisation protocol
- Header - optional extension capability (e.g. security, transactions, etc)
- Body - application defined XML formatted data

Example Stock Quotation Full Request (from Spec V1.1):

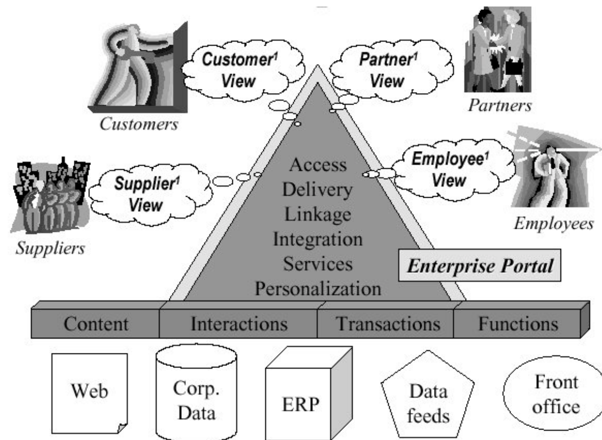
```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"

<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
     <m:GetLastTradePrice xmlns:m="Some-URI">
     <symbol>DIS</symbol>
     </m:GetLastTradePrice>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Example Response Body only (from Spec V1.1)::

```
  <SOAP-ENV:Body>
     <m:GetLastTradePriceResponse xmlns:m="Some-URI">
     <Price>34.5</Price>
     </m:GetLastTradePriceResponse>
  </SOAP-ENV:Body>
```

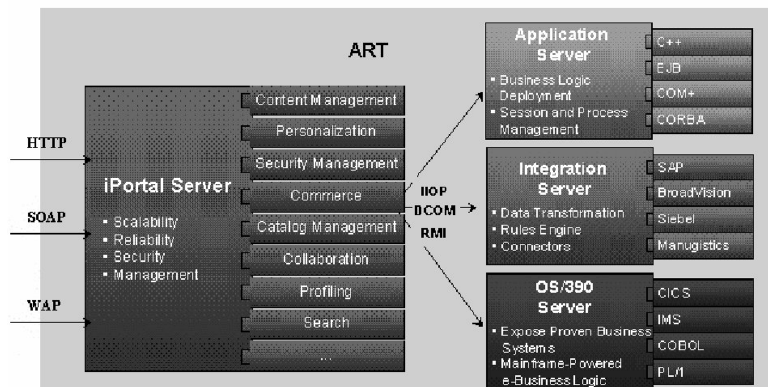
## Enterprise Portals

An immediate technology framework predecessor to the current "web services" technologies – they provide a single electronic point of presence for an enterprise that allows clients (other businesses and customers) to flexibly access an enterprise's data, transaction and processes:




[Iona Technologies, 2000]

Example: Iona's iPortal Suite architecture (www.iona.com) - combines all server platforms with application integration, legacy system connectivity and server administration:

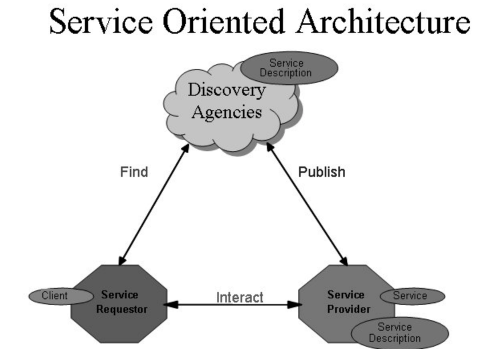



[Iona Technologies, 2000]



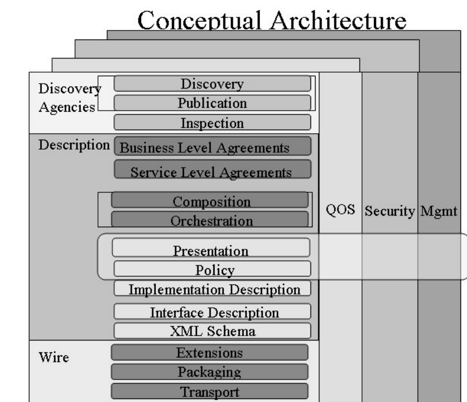
## Web Service Architectures *(www.w3.org/TR/ws-arch)*

"Web services" is a term used to describe a set of distributed software components that interact over the Internet to collectively provide the infrastructure for a web-enabled application. The distributed architectures used to provide these services follows a conventional client/server structure:




[W3C, 2002]

Web services group together a number of technologies supported by common Internet protocols (HTTP, & TCP) – the core or basic layers on that are XML and SOAP. Less well defined, but emerging is a common description language for web services (WSDL), and a range of basic services (Universal Description, Discovery and Integration or UDDI).

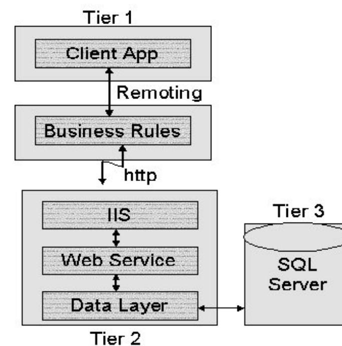



[W3C, 2002]

## Microsoft's DNA (COM, DCOM, COM+) → .NET Framework

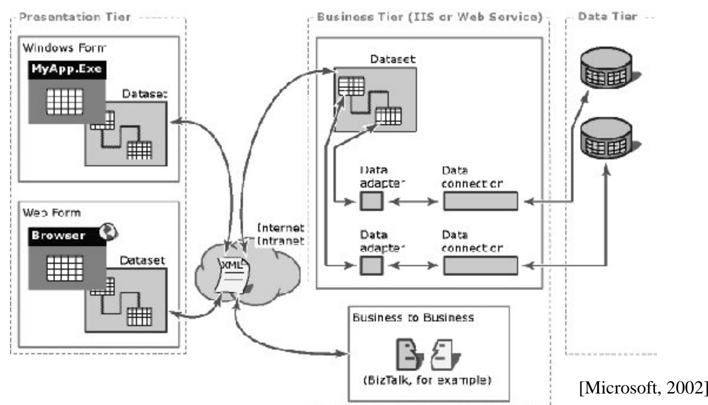
In DNA applications, business logic was usually implemented as COM or COM+ components which provided the middle layer between client ASP pages and back-end DBMS applications. Communications between components on different nodes was only possible via DCOM which provided security and authentication infrastructure.

The .NET framework allows business logic to be written in ASP.NET, and supports more options for distributed communication with .NET remoting (which supports both TCP and HTTP channels). DCOM for distributed communication among COM objects is also supported.
The .NET remoting TCP channel doesn't provide the strong security of DCOM, but the configuration overhead is much lower.




[Microsoft, 2002]

Data can be stored with ADO.NET objects – into *datasets* that act as caches – these datasets can be automatically serialized into XML and passed via SOAP.




[Microsoft, 2002]

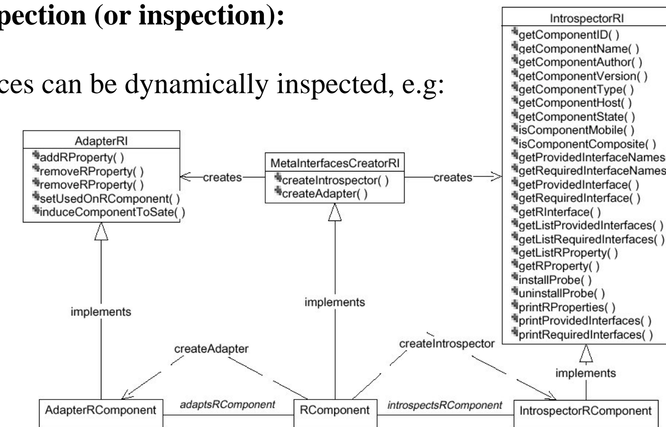


## Trends In Middleware Research

The primary focus currently is towards reflective middleware (i.e. an introspective and adaptive capable software bus to better support dynamic distributed mobile environments):

### Introspection (or inspection):

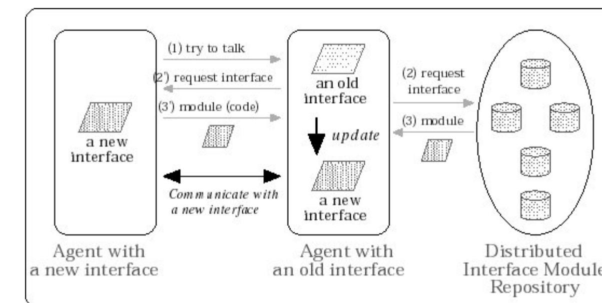
Interfaces can be dynamically inspected, e.g:




[Moreira et al, 2002]

### Adaptation:

Interfaces can be dynamically replaced, e.g:




[Soltysiak et al, 2002]

Combined with this trend is the development of Model Driven middleware architectures and Meta-Object Protocols → highly configurable 'technology neutral' middleware specified by design model (see OMG).