**Middleware**

Distributed Objects and Components

Interoperability

Dynamic Linking

Middleware

NAS

OLE/COM

CORBA

---

**Distributed Objects and Components**

- **Stovepipe systems**
  - A stovepipe system is a set of legacy applications that resists adaptation to user and organizational needs to create client/server solutions
  - Monolithic, vertically integrated applications include in one tool all the services (e.g., for a desktop publishing, include word processing, spreadsheet, database, etc.)
    - Ever-growing tools
  - Closed system and custom proprietary solution intolerant to outside services
  - Little discernible software architecture system poorly understood by developers and maintainers
    - Not easily reusable or extensible
    - Slow development and deployment
    - Expensive maintenance and evolution

- **Architectural focus!**
  - Leave out details
  - Understand the system: components, interfaces, constraints, patterns, & rationale
  - Keep the size of components manageable and focus

- **Interoperable distributed components**
Distributed Objects and Components

"An object is a living, breathing blob of intelligence that knows how to act in a given situation."
- Steve Jobs

- Client-server can be viewed as the precursor technology to distributed objects
- Objects work together across machine and network boundaries to create client/server solutions
- Distributed objects = independent software components
- A component is not bound to any particular OSs, HWs, network, tools, PLs, Applications, Vendors

Plug&Play

Portability

Coexistence (through object wrappers)

Interoperability
(communication (channel & protocol), (interpretable) request for ins & data, data interchange format, data semantics)

Middleware

Unix OS/2 MAC

Interoperability between N vendor databases ends up being an N*N problem

True database independence will not be possible without 3-tiered architectures
(Meta Group. 1993)

Interoperability between N vendor databases ends up being an N*N problem
(McKeen Al-Ghosein. 1995)
Dynamic Linking

Module Menu;
  Import DisplayMenu, UserSelection;
  Procedure MenuStatus(...);
  Procedure ProcessMenu(...);
  begin
    DisplayMenu.MainMenu(...);
    MenuStatus(...);
    UserSelection.Accept(...);
    DisplayMenu.MinorMenu(...)
  end
end Menu.

Module DisplayMenu;
  Procedure MainMenu(...);
  Procedure MinorMenu(...);
end DisplayMenu.

Module UserSelection;
  Procedure Accept(...);
end UserSelection.

Object module Menu;
  Entry table
    entry0: MenuStatus@addr-1
    entry1: ProcessMenu@addr-2
  Object code
    /* code for MenuStatus */
    addr-1: ...
    /* code for ProcessMenu */
    addr-2:
    addr-3: indirect call via link0
    relative branch to addr-1
    addr-4: indirect call via link1
    addr-4: indirect call via link2
  Link table
    link0: DisplayMenu.MainMenu
    link1: UserSelection.Accept
    link2: UserSelection.Accept

+ one copy, no recompilation of every component
- slower, type mismatch

Middleware: An Illustration

- A system integration from many functional areas, for their information processing needs
- Each platform can have a local DB
  E.g., Laboratory stores CAT scan results and other lab graphics;
- Many real benefits:
  reduction in cost
  substantial reduction in time delay between lab examination and doctor’s evaluation
  [Fox Chase Cancer Center]
- Access to external services (e.g., Health Insurance) and access from external sites (e.g., doctors on trip, at home)
The NAS Computing Environment

The NAS API (Application Programming Interface) consists of:
- **PL Spec**: defines, for each PL, the syntax and semantics of the common requestable functions
- **Environment Spec**: (API service profile) defines which NAS services are available on a particular platform
- **API Service Spec**: defines the syntax and semantics of the functions provided by a specific NAS service

The NAS system interface (SI) spec. defines how an NAS service accesses the services of OS/Network, enhancing platform-independence.

Freedom in choosing network architecture, operating system architecture, and hardware transportation mechanisms.

A (superset) implementation of industry middleware standards of DCE (Distributed Computing Environment) by OSF (The Open Software Foundation).

- by DEC towards open systems
  - application portability
  - application interoperability
  - user interface portability

- The NAS system interface (SI) spec. defines how an NAS service accesses the services of OS/Network, enhancing platform-independence.
The NAS API (Application Programming Interface) consists of:

- **PL Spec:** defines, for each PL, the syntax and semantics of the common requestable fns
- **Environment Spec:** (API service profile) defines which NAS services are available on a particular platform
- **API Service Spec:** defines the syntax and semantics of the functions provided by a specific NAS service

The NAS system interface (SI) spec. defines how an NAS service accesses the services of OS/Network  

**Platforms/Network Transport Mechanisms**

**NAS Services**

<table>
<thead>
<tr>
<th>Presentation Services</th>
<th>Communication Services</th>
<th>Control Services</th>
<th>Information Services</th>
<th>Computation Services</th>
<th>Management Services</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windowing</td>
<td>Mail</td>
<td>Appl. Ctrl Arch</td>
<td>Data Access/</td>
<td>Date/Time</td>
<td></td>
</tr>
<tr>
<td>Terminal</td>
<td>Access Repository</td>
<td>TP</td>
<td>File Sharing</td>
<td>Numeric</td>
<td></td>
</tr>
<tr>
<td>Printing</td>
<td>File Sharing</td>
<td></td>
<td>Directory</td>
<td>Text</td>
<td></td>
</tr>
<tr>
<td>Hyperinfo</td>
<td>EDI</td>
<td></td>
<td>Repository</td>
<td>Utility</td>
<td></td>
</tr>
<tr>
<td>Forms</td>
<td>Msg Queuing</td>
<td></td>
<td>Compound Doc</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Graphics</td>
<td></td>
<td></td>
<td>Arch</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Consider desktop publishing multiple tools, multiple services, multiple documents, multiple objects
OLE/COM

Embedding: for editing objects within a container object without creating a separate appl window
Linking: for the display of common data in multiple documents with updates
Drag and Drop: for dragging & dropping subsets of docs between similarly enabled OLE2 applns
Uniform Data Transfer: a clipboard facility for adding OLE2 data objects to the clipboard
Compound Files: to persistently store multiple objects from multiple applns within a container object
COM: defines the basic interface mechanisms for invoking OLE2 objects
Moniker: a naming facility, supporting linking using file pathnames
Automation: similar to Dynamic Invocation Interface, for controlling appln thru a dispatch fn

Lawrence Chung
OLE/COM

Structured Storage (Data Architecture Specification)

- enable storage & partitioning of complex data from multiple embedded objects into a common file
- created and managed by OLE2 container objects

CORBA

- Common Object Request Broker Architecture

- A specification for a standard OO architecture for applications
  - not a low-level design/implementation
  - platform (OS, HW)-independence, PL-independence
- defined by the Object Management Group (OMG) since Nov 1990
  - currently >500 members
- CORBA clients and servers do not need direct knowledge of each other
  - the broker knows the locations and capabilities of the servers on the network
- A client request can be fulfilled by several (competing) servers
  - the broker should know who can provide the service fastest and cheapest
- An Object Model requires abstraction, encapsulation, inheritance & polymorphism

"The ability to create simplifying abstractions is a key innate talent of the software architect. Few individuals practicing in the software industry have this ability - perhaps as few as one in five software designers." [Coplien, '94]
An OMG IDE file describes the data types, operations, and objects
that the client can use to make a request and
that a server must provide for an implementation of a given object
OMG IDL and C code are very similar in appearance
The internal representation of an object reference might differ,
but all object references have the same external representation for a given PL

- complex data structure, forward referencing, readonly object attributes
- single inheritance, multiple inheritance within & across modules
OMG Interface Specification

Client Stub
- one PL-specific routine for each operation (e.g., "hire") in source code format;
  compiled & linked into the client application

Header File
- define data types (structures and constants)
  to be included into the client/server application source code

Server Skeleton
- for mapping client operations to methods in the implementation;
  generated in source code format; compiled and linked into the server application

CORBA Interfaces

Client Stub
- one PL-specific routine for each operation (e.g., "hire") in source code format;
  compiled & linked into the client application

Header File
- define data types (structures and constants)
  to be included into the client/server application source code

Dynamic Invocation Interface
- Unlike Static Invocation Interface (Static Client Stub),
  DII lets you postpone selection of object type and operation until run time + asynchronous mode.

Server Skeleton
- for mapping client operations to methods in the implementation;
  generated in source code format; compiled and linked into the server application

Dynamic Skeleton Interfaces
- communicate with remote ORBs

Object Adapters
- activating and invoking a server (object implementation)
Basic Object Adapters

Object Implementation

**BOA routines**
- ORB receives call
  - 1. ORB receives call
- Impl_ready
  - 2. check repository: not (active (server) or active (object))
    - -> pass info
  - 3. pass the obj. ref to the target obj
    - If previously active retrieve the prev state from repository
- Methods
  - send
  - response deactivate_impl deactivate_obj
- deactivate_obj
deactivate_impl

**Server routines**
- Activate Server
- Activate Object
- Pass Invocation Skeletons

**ORB**

\[ \text{Shared server policy: a single server running a number of objects, one per interface} \]
\[ + \text{reduced process-initiation overhead} \]
\[ - \text{hard to enforce memory and security isolation of one object from another} \]
\[ \text{Unshared server policy: only 1 object of a given impl at a time can be active on a server} \]
\[ \text{appropriate when exclusive resource (e.g., a printer)} \]
\[ \text{Server-per-method policy: a separate server for each method invocation} \]
\[ \text{good for load balancing or isolation of servers from each other (security/administration)} \]

So,

**All systems have subsystems and all systems are parts of larger systems.**

The value added by a system must come from the relationships between the parts, not from the parts per se.

- The art of systems architecting

**Architectural design can, literally, make it or break it.**