CHAPTER 11

APPLICATION ARCHITECTURE AND MODELING
Application Architecture

An **application architecture** specifies the **technologies** to be used to implement one or more (and possibly all) information systems in terms of DATA, PROCESS, and INTERFACE, and **how these components interact** across a network.

Realize that ALL of data, process, and interface is considered part of the application architecture – not merely the physical distribution of computers…

Think of application architecture as a set of **construction blueprints** that a building contractor might require for different building aspects such as framing, electrical, plumbing, ventilation, etc…

In this chapter we will concentrate on:
Objectives

- We will look at the degree to which information systems will be centralized or distributed. Consistent with modern trends, most new systems are distributed across a networks – including the intranet and Internet.

- We will look at the distribution of DATA across the network. (Most modern databases are either distributed or duplicated across networks, either in a client/server or network computing pattern)

- We will look at implementing technologies – which language and tools will be used?

- Will look at the technologies to be used to implement the user interface (inputs and outputs)

- Will look at the technologies to be used to interface with other systems.

- Differentiate between logical and physical data flow diagrams, and explain how physical data flow diagrams are used to model an information system’s architecture.

- We will look at the tools first in this chapter.
Physical Data Flow Diagrams (DFDs)

By changing the Logical DFDs just a bit, LDFDs can communicate technical choices and other design decisions to those who will actually construct and implement the system. These then become Physical Data Flow Diagrams. (PDFD).

Physical data flow diagrams (DFDs) model the technical and human decisions to be implemented as part of an information system.

Note that the process of creating a logical DFD and then a physical DFD still remain with us today.

Structure Charts may also be used
Sample Physical Data Flow Diagram

This diagram intentionally incomplete and oversimplified.

Walk through this...
Physical Processes

• A **physical process** is either
  a processor, such as a computer or person, or
  a technical implementation of specific work to be performed, such as a computer program or manual process.

• Logical processes may be assigned to physical processors such as PCs, servers, mainframes, people, or devices in a network.

• A physical DFD would model that network structure.

• Each **logical process** requires an implementation as one or more **physical processes**. Note that a logical process may be split into multiple physical processes:
  • To define those aspects that are performed by people or computers.
  • To define those aspects to be implemented by different technologies.
  • To show multiple implementations of the same process.
  • To add processes for exceptions and internal control (e.g., security).
Comment:

• Some approaches to teaching these materials no longer teach a separate modeling technique for logical and physical networks because the industry and CASE tools never embraced a single physical modeling notation for networks.

• The authors suggest using physical DFDs to model networks.

• This approach seems to be quite effective.
Physical Process Notation
Physical Process Notation

ID is often a number indicating a hierarchy

This is a typical use of a Process Box.

Also, a **specific office** is sometimes used to indicate ‘responsibility.’ (OPR)

These are normally found at ‘higher levels’ in DFDs.

This is the Gane and Sarson notation. The DeMarco and Yourdon notation does not provide for a separate **symbolic** differentiation between logical and physical processes. Depending on your choice of CASE or automated modeling tool, may required slightly different notations.)
Samples of Physical Processes

Showing physical implementations from the logical process.

<table>
<thead>
<tr>
<th>Logical Process</th>
<th>Sample Physical Process Implementations</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.3 Check Customer Credit Acct Clerk</td>
<td>4.3 Check Customer Credit COBOL/CICS</td>
</tr>
<tr>
<td>4.3 Check Customer Credit Acct Clerk</td>
<td>4.3 Check Customer Credit Visual Basic</td>
</tr>
<tr>
<td>4.3 Check Customer Credit Acct Clerk</td>
<td>4.3 Check Customer Credit Quickbooks</td>
</tr>
</tbody>
</table>

Samples – but mutually exclusive
A Process may be implemented as:

- A purchased **application** software package
  - Also called *commercial off-the-shelf (COTS) software*

- A system or utility program

- An existing application program
  - May require modification

- A program to be written
Physical Processes – last overhead

• Note that the number of processes on a PDFD will almost always be greater than those in a LDFD because of implementation decisions.

• Maturing from ‘what’ to ‘how…’

• The final PDFD must reflect both manual and computer processes required as part of the implementation strategy.
Physical Data Flows
A physical data flow represents *any* of the following:

- The *planned implementation* of an input to, or output from a physical process.
- A *database command or action* such as create, read, update, or delete.
- The *import* of data from, or the *export* of data to another information system across a network.
- The *flow of data between two modules* or subroutines (represented as physical processes) in a program.

- Used to factor a program into modules – perhaps to pass parameters, variables, and arguments between modules of a program.
- This is a reminder that DFDs can be used as programming tools.
There are always alternative physical implementations for any logical data flow.
### Sample Physical Data Flows (continued)

<table>
<thead>
<tr>
<th>Logical Data Flow</th>
<th>Implementation</th>
<th>Sample Physical Data Flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>Update Credit rating</td>
<td>Update a record in a database</td>
<td>SQL Update: Credit Rating</td>
</tr>
<tr>
<td>Delete Employee</td>
<td>Delete a record in a database</td>
<td>SQL Delete: Employee</td>
</tr>
<tr>
<td>Insurance Accident Claim</td>
<td>Import a data file</td>
<td>IMAGE FILE: Insurance Accident Claim</td>
</tr>
<tr>
<td>Schedule of Classes</td>
<td>Export a data file</td>
<td>Comma Delimited File: Schedule of Classes</td>
</tr>
<tr>
<td>Extended Cost</td>
<td>Pass data between modules of a program</td>
<td>Extended Cost</td>
</tr>
<tr>
<td>Course Request</td>
<td>Pass a manual form</td>
<td>Form 23: Course Request</td>
</tr>
</tbody>
</table>

Show manual process or form.
External Agents and Data Stores
Physical External Agents and Data Stores

Physical external agents are carried over from the logical DFD models.

A physical **data store** represents the planned implementation of one of:

- A database
- A table in a database
- A computer file
- A tape or media backup of anything important
- A temporary file or batch
- Any type of noncomputerized file
Physical Data Store Notation

<table>
<thead>
<tr>
<th>ID (opt)</th>
<th>Implementation Method: Data Store Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID (opt)</td>
<td>Data Store Name (Implementation Method)</td>
</tr>
</tbody>
</table>
## Physical Data Store Implementations

<table>
<thead>
<tr>
<th>Logical Data Store</th>
<th>Implementation</th>
<th>Physical Data Store</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Human Resources</strong></td>
<td>A database (multiple tables)</td>
<td>Oracle: Human Resources DB</td>
</tr>
<tr>
<td><strong>Marketing</strong></td>
<td>A database view (subset of a database)</td>
<td>SQL Server: Northeast Marketing DB</td>
</tr>
<tr>
<td><strong>Purchase Orders</strong></td>
<td>A table in a database</td>
<td>MS Access: Purchase Orders</td>
</tr>
<tr>
<td><strong>Accounts Receivable</strong></td>
<td>A legacy file</td>
<td>VSAM File: Accounts Receivable</td>
</tr>
<tr>
<td><strong>Tax Rates</strong></td>
<td>Static data</td>
<td>ARRAY: Tax Table</td>
</tr>
<tr>
<td><strong>Orders</strong></td>
<td>An off-line archive</td>
<td>TAPE Backup: Closed Orders</td>
</tr>
<tr>
<td><strong>Employees</strong></td>
<td>A file of paper records</td>
<td>File Cabinet: Personnel Records</td>
</tr>
<tr>
<td><strong>Faculty/Staff Contact Data</strong></td>
<td>A directory</td>
<td>Handbook: Faculty/Staff Directory</td>
</tr>
<tr>
<td><strong>Course Enrollments By Date</strong></td>
<td>Archived reports (for reuse and recall)</td>
<td>REPORT MGR: Course Enrollment Reports</td>
</tr>
</tbody>
</table>
Where can we use this information later?

- So, Physical DFDs use the same symbols as Logical DFDs

- Can use this physical model to design the **internal** and **external** details for each data store and data flow.
INFORMATION TECHNOLOGY ARCHITECTURE

• Technology is changing all the time.
• Systems Analysts must always learn new technologies:
  – Uses, application domains; restrictions.
  – Read trade journals

• Data, process and interface: key building blocks of all information systems.
• Any software design must address the application’s architecture, that is, the **distribution** of process, data, and interface.

• Nowadays, previous stand-alone mainframe and personal computers are being linked together in various ways using servers and forming very complex networks.
  – Processes are distributed; data is distributed; users are world-wide…
  – Consequently, processes, data, and interfaces are similarly distributed or at least duplicated over these complex networks.
Distributed versus Centralized Systems

Today’s systems are not monolithic, mainframe-based systems. They are built on some combination of networks to form distributed systems.

A **distributed system** is one in which the DATA, PROCESS, and INTERFACE components of an information system are distributed to multiple locations in a computer network.

  -> Processing workload is distributed across the network.

In **centralized systems**, a central, multi-user computer hosts all the DATA, PROCESS, and INTERFACE components of an information system.

  -> Users interact with the system via terminals (or terminal emulators).
Why Distributed Systems?

- Modern business systems: already decentralized and distributed.
- Distributed computing moves information and services closer to the customers and users who need them.
- Distributed computing consolidates the power of personal computers across the enterprise.
- Distributed computing solutions are more user-friendly because they utilize the PC as the end user interface.
- Personal computers and network servers are cheaper than centralized mainframe computers.
  - This may be controversial. Some experts sincerely believe that distributed computing is more costly to maintain. But this belief has not deterred the proliferation of distributed solutions.
Distributed Computing Systems Architectural Layers

- **Presentation layer** — the user interface
  the presentation of inputs and outputs to the user

- Presentation layer logic — such as input editing
  the processing necessary to generate the presentations.

- **Application logic layer** — the business rules, policies, and procedures
  - includes the logic and processing procedures; the computations.

- **Data manipulation layer** —
  - includes the commands and logic to store and retrieve data to and from the database

- **Data layer** — the actual business data itself.

In some books, the presentation and presentation logic layers are combined.
Flavors of Distributed Computing

Discuss:

Various combinations

Let’s look closely at the three primary ‘flavors’ of distributed systems architecture.
File Server Architecture

A **local area network** (LAN) is a set of client computers (PCs) connected to one or more server computers either through cable or wireless connections over relatively short distances.

A **file server system** is a LAN-based solution in which a server hosts only the data layers of an information system. All other layers are implemented on the client computers. Disadvantages include:

- Frequently excessive network traffic to transport data between servers and clients.
- Client must be fairly robust (“fat”) because it does most of the work
  - all applications software loaded on client
- Database integrity can be easily compromised.
  - In use, entire file / table of records must be downloaded to client for data manipulation logic to do its thing. (lockout!) Server only contains data. No data manipulation. Done on client.
  - performance degrades with number of simultaneous users.
- Very popular for small groups; File server databases like Access can be used to construct prototypes for more robust client/server architectures.
File Server Architecture

1. Presentation

2. Request to create, read, update, or delete 1 or more records

3. Entire tables

4. Table locked until client returns table

5. Response to request returns entire tables

6. Entire tables with any updated records

7. Updated tables

8. Unlock tables

Fat Client

User

Presentation, Application, and Data Manipulation Logic all executed here

Client PC

File Server Database (e.g. MS Access)

Only serves to store data. No service other than storage and transport provided here

File Server
– Many students think Access applications that store their data on a file server are truly client/server applications (and many books proliferate the misconception).

– Previous slide, when contrasted with the client/server equivalent slides, is intended to dispel that myth.

– Because the client executes any CREATE, READ, UPDATE, and DELETE commands for a file-server database (such as Access), the entire table must be locked out from other users and transported to the client to execute the command.

– Result is increased data traffic on the network, and slow performance since the table must be returned and unlocked before the next user can do anything with it.
Client/Server Architecture—The Clients

A **client/server system** is a solution in which the presentation, presentation logic, application logic, data manipulation, and data layers are **distributed** between client PCs and one or more servers.

A **thin client** is a personal computer that **does not** have to be very powerful (or expensive) in terms of processor speed and memory because it only presents the user interface to the user.

A **fat client** is a personal computer or workstation that is typically more **powerful** (and expensive) in terms of processor speed, memory, and storage capacity.

Most PCs are considered to be fat clients.
– We are not using “fat client” in a negative sense.

– For the time being, most workers require fat clients for their personal productivity applications (e.g., word processing and spreadsheets).

– But they may use thin client features on those fat clients (e.g., a Web browser or terminal emulator) to run or access remote applications (such as those provided by the information systems unit of the company).
Client/Server Architecture—The Servers – various…

- **A database server** hosts one or more shared databases but also executes all data manipulation commands.
  - examples: Oracle, Microsoft SQL Server, IBM Universal Database

- **A transaction server** hosts services that ultimately ensure that all database updates for a single transaction succeed or fail as a whole.
  - e.g. CICS, Microsoft Transaction Server

- **An application server** hosts the application or business logic and services for an information system.
  - must communicate with front end with clients for presentation and back end with database server. typical technologies: CORBA, or COM+
Client/Server Architecture—The Servers – various...

• A **messaging or groupware server** hosts services for e-mail, calendaring, and other work group functionality.
  - typical technologies: Lotus Notes and Microsoft Exchange Server

• A **web server** hosts Internet or intranet web sites and services, communicating through thin-client interfaces such as web browsers.
  - typically use html for documents, xml for data formats.
These server categories are not mutually exclusive.

For example, many database server and transaction server functions may be integrated on a single physical server.

Also, web and messaging server functions are being integrated with each new release of some products.
Client/Server—Distributed Presentation

A distributed presentation client/server system is a solution in which the presentation and presentation logic layers only are **shifted** from the **server** to reside on the **client**.

Application logic, data manipulation, and data layers remain on the server (frequently a mainframe).
Client/Server—Distributed Presentation

Typically the old CUI (character user interface) are taken from legacy apps and regenerated as GUIs that run on a PC. (only UI is distributed)

Adv1: can be implemented quickly; Most elements of legacy apps remain unchanged.
Adv2: Users get a welcome, familiar interface via a GUI.
Adv3: Extends useful lifetime expectancy of legacy apps.
Dis1: Applications functionality is not significantly improved.

CASE tools called screen scrapers read the CUI and generate a first-cut GUI that can be modified by a GUI editor.
This is sometimes called “the poor man’s client/server.”

In reality, the underlying technology allows organizations with constrained resources to give mainframe systems a client/server facelift that extends their useful lifetime until they can be redesigned to more contemporary architectures.
Client/Server—Distributed Presentation

User → Presentation

Presentation Logic only executed here

Client PC

Output and Instructions for translation to GUI

GUI: Input and/or Commands for Processing

Legacy App (e.g., COBOL)

Application and Data Manipulation Logic executed here

Mainframe

Conventional Files (e.g., VSAM) or Database (e.g., Oracle)

Reads and/or Updates
A distributed data client/server system is a solution in which the data and data manipulation layers are placed on the server(s), and the application logic, presentation logic, and presentation layers are placed on the clients.

This is sometimes called two-tiered client/server computing.

Again, lots of flavors

This architecture characterizes most Visual Basic and Powerbuilder applications in existence today.
Client/Server—Distributed Data

1. Presentation
2. Request to create, read, update, or delete 1 or more records
3. Read requested rows and columns only from the tables
4. Records (only) Locked Until Client Releases Table
5. Response to request returns only the rows and columns needed
6. Application returns only any data to be updated
7. Updated Tables
8. Unlock Records

User
Presentation and Application Logic executed here
Client PC
Database Server
C/S Database stored here
Differences between file server system and distributed data client/server system

– Both use a database server for data storage.
– But: client-server’s *server* also *executes* all data manipulation logic.
  • all application logic, presentation logic and presentation services are handled by the client machine.
– In file server, *everything other than data storage* is done on the client machine.
Advantages / Disadvantages of two-tier client-server architectures

- **Client**: Much less network traffic, since only database requests and records are transmitted over lines.
- Only records that are being used by the client need to be locked on the server. (Simultaneous usage ok – except for record update).
- BUT, **clients** need to be reasonably ‘fat’ because the clients must perform all the application logic.
- **Clients** normally perform their functions with client/server programming language (e.g., VB or Visual C++, whose object code must run on the client. (Sometimes common, stored business procedures may be stored on the server, but this is controversial.)
- The **database server** stores the database and executes the data manipulation commands. Clients merely send their instructions to the server. Servers may return command processing results. (Oracle and Microsoft SQL Server do this).
- **Key disadvantage**: application logic must be duplicated / maintained on all clients. Then version control / upgrades may be an issue.
A **distributed data and application** client/server system is a solution in which: (1) the data and data manipulation layers are placed on their own server(s), (2) the application logic is placed on its own server, and (3) the presentation logic and presentation layers are placed on the clients.

This is sometimes called **three- or n-tiered client/server computing**. It requires **design partitioning**.

**Partitioning** is the art of determining how to best distribute or duplicate application components (DATA, PROCESS, and INTERFACE) across the network.

This is the architecture of your projects.
Client/Server—Distributed Data and Application

- Two-tiered systems cannot handle high number of clients. (All application logic executes on each client!)
- Three-tiered systems have the application logic / transaction server on its own box.
- Three-tiered client/server logic can be written and partitioned across multiple servers using languages such as VB and C++
- In an n-tiered architecture, clients execute very little – usually the user interface and some real stable personal applications logic. Simplifies client configuration.

- **Biggest problem:** complexity in design and development. Biggest problem in here is ‘partitioning.’
Client/Server—Distributed Data and Application

- Presentation Logic only executed here
- Client PC

- Application Logic only executed here
- Application Server

- Data and Service Requests

- Read requested rows and columns only from the tables
- C/S Database stored here

- Records (only) Locked Until Client Releases Table

- Data Manipulation Logic only executed here
- Database Server

- Response to data manipulation request
- Unlock Records

- Request to create, read, update, or delete 1 or more records

- Updated Tables
A **network computing system** is a multi-tiered solution in which the presentation and presentation logic layers are implemented in client-side Web browsers using content downloaded from a Web server. The presentation logic layer then connects to the application logic layer that runs on the application server, which subsequently connects to the database servers on the backside of the system.

The greatest potential of this approach is its applicability to redesign of traditional information systems to run on an intranet. An **intranet** is a secure network, usually corporate, that uses Internet technology to integrate desktop, work group, and enterprise computing into a cohesive framework.
The authors feel that ‘network computing’ is a fundamental shift away from what they just described as client/server. They call this architecture internet-based computing architectures – and that these are shaping the information systems of most businesses.

This is the architecture that is driving e-commerce and e-business.

Expect this architecture to eventually replace most two- and three-tier solutions.
Internet- and Intranet-based Architectures

Web browser
- Only Java applets and Web scripts executed here
- Client PC (thin or fat)

Presentation

User

Web server
- Navigation Commands
- Transaction Data, and Service Requests

Application logic
- only executed here

Application Server

Database Server
- Request to create, read, update, or delete 1 or more records
- Response to data manipulation request

C/S Database
- stored here

Updated Tables
- Unlock Records

Data Manipulation Logic
- only executed here

Recorded rows and columns only from the tables
- Records (only)
- Locked
- Until
- Client
- Releases
- Table

HTML: Page Content and XML: Data Content

Authentication
- Presentation distributed from here
Internet- and Intranet Technologies

• **Java**
  – Mostly for programming server-side application logic called “servlets”
  – Occasionally for programming client-side application logic called “applets”

• **HTML (HyperText Markup Language)**
  – Mostly for programming the presentation layer

• **XML (Extensible Markup Language)**
  – Mostly for programming data content to be transported across the web

• **SQL (Structured Query Language)**
  – Universal standard language for database manipulation

• **Web Browsers**
Data Architectures – Should be quite familiar to you…

A **relational database** stores data in tabular form. Each file is implemented as a table. Each field is a column in the table. Related records between two tables are implemented by intentionally duplicated columns in the two tables.

A **distributed relational database** distributes or duplicates tables to multiple database servers located in geographically important locations.

A **distributed relational database management system** is a software program that controls access to and maintenance of stored data in the relational format.
Types of Data(base) Distribution

**Data partitioning** truly distributes rows and columns of tables to specific database servers with little or no duplication between servers.

- Vertical partitioning assigns different columns to different servers.
- Horizontal partitioning assigns different rows to different servers.

**Data replication duplicates** some or all tables (or parts of tables) on more than one database server. Database technology controls access to, and manages consistency of duplicated data across the servers.
### Data Partitioning versus Data Replication

<table>
<thead>
<tr>
<th>Logical Data Store</th>
<th>Physical Data Stores using Partitioning</th>
<th>Physical Data Stores using Replication</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CUSTOMERS</td>
<td>Oracle 7: REGION 1 CUSTOMERS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Oracle 7: REGION 2 CUSTOMERS</td>
</tr>
<tr>
<td>2</td>
<td>PRODUCTS</td>
<td>Not applicable. Branch offices do not need access to data about customers outside of their own sales region.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Oracle 8i: PRODUCTS (Master)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Oracle 8i: PRODUCTS (Replicated Copy)</td>
</tr>
</tbody>
</table>

**CUSTOMERS**
- Oracle 7: REGION 1 CUSTOMERS
- Oracle 7: REGION 2 CUSTOMERS

**PRODUCTS**
- Not applicable. All branch offices need access to data for all products, regardless of sales region.
- Oracle 8i: PRODUCTS (Master)
- Oracle 8i: PRODUCTS (Replicated Copy)
Interface Architectures: Inputs, Outputs, Middleware

• Architectures and interfaces use to be easy:
  – All systems either batch or on-line or combination
    Batch inputs and outputs

• Today: much more difficult:
  – Batch input / output
  – On-line inputs and outputs
  – Remote batch
  – Keyless data entry (and automatic identification)
  – Pen input
  – Electronic Data Interchange (EDI)
  – Middleware
# Batch Inputs and Outputs

<table>
<thead>
<tr>
<th>Logical Data Flow (input)</th>
<th>Physical Data Flow Implementation (as batch input)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMECARD</td>
<td>KTD Batch: TIMECARDS Comma delimited file: TIMECARDS</td>
</tr>
<tr>
<td>Business collect and process timecards. May enter data: key to disk</td>
<td>KTD Batch: TIMECARDS</td>
</tr>
<tr>
<td>Batch output?</td>
<td>End of Month -1 day</td>
</tr>
<tr>
<td></td>
<td>PAYCHECK Preprinted Form Batch: PAYCHECKS</td>
</tr>
</tbody>
</table>

Business collect and process timecards. May enter data: key to disk

Batch output?
- Generation of invoices, checks, tax forms, etc.
- May use pre-printed forms…

Paycheck
- Generation of invoices, checks, tax forms, etc.
- May use pre-printed forms…
## On-Line Inputs and Outputs

<table>
<thead>
<tr>
<th>Logical Data Flow (input and output)</th>
<th>Physical Data Flow Implementation (as on line input and output; 2 alternatives shown)</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSURANCE CLAIM</td>
<td>Win 2000 Form: INSURANCE CLAIM</td>
</tr>
<tr>
<td>ORDER CONFIRMATION</td>
<td>HTML Form: ORDER CONFIRMATION</td>
</tr>
</tbody>
</table>

- Provide more conversational dialog
- Provide for more immediate feedback
- Identified errors can be fixed quickly
- Permits more human interaction in decision making.
Remote Batch

- Some applications might do both on-line and batch…
- Might edit transactions on-line, then batch them up for later processing as shown in PDFD.
Keyless Data Entry (and Automatic Identification)

<table>
<thead>
<tr>
<th>Logical Data Flow (input)</th>
<th>Physical Data Flow Implementation (optimal mark form batch input)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EXAMINATION ANSWERS</strong></td>
<td><strong>Optimal Mark Form Batch: EXAMINATIONS' ANSWERS</strong></td>
</tr>
</tbody>
</table>

Another modern form of input.....
Electronic Data Interchange (EDI)

Electronic Data Interchange (EDI) is the standardized electronic flow of business transactions or data between businesses. Typically, many businesses must agree to a common data format to make EDI feasible.

<table>
<thead>
<tr>
<th>Logical Data Flow (input)</th>
<th>Physical Data Flow Implementation (automatic ID input)</th>
</tr>
</thead>
<tbody>
<tr>
<td>STUDENT APTITUDE SCORE</td>
<td>EDI: STUDENT APTITUDE SCORES</td>
</tr>
</tbody>
</table>

EDI results in the elimination of paper documents and mail. Often done between businesses.
**Middleware**

Middleware is **utility software** that enables communication between different processes/ors in a system. It may be built into the respective operating systems or added through purchased middleware products.

- Presentation middleware
- Application middleware
- Database middleware

“Process-to-process” communications is facilitated via middleware.

→ Most of you have probably used database middleware products such as ODBC or JDBC that allow a single application to access and use different vendor’s database engines w/o having to write a different version of the program for each database engine.
Middleware - Examples

- **Presentation Middleware** — e.g., http…allows a programmer to communicate with a web browser through a standard API

- **Application Middleware** — e.g., (lots of them…) RPC, message queues, object request brokers, etc. Enables programmer-written processes on different processors to communicate with whatever way is best for them to do so.

- **Database Middleware** — e.g., ODBC (Object data base connectivity, JDBC – JavaBean Database Connectivity) – allows a programmer to pass SQL commands to a database engine for processing through a standard API. ODBC and JDBC automatically translate SQL commands for one database server for user on a different database server, like Oracle to SQL Server.
Process Architectures

- **A Software Development Environment (SDE)** is a programming language and tool kit for constructing information systems software applications.
- Often called Integrated Development Environments – (IDEs)

  - SDEs exist for centralized computing
    - Like COBOL, CICS, TSO, VSAM. DB2 and generally, MVS
  - SDEs exist for distributed presentation
    - E.g. Microfocus Dialog Manager helps to build Windows-based user interfaces that can operate with CICS transaction monitors and mainframe COBOL programs
  - SDEs exist for two-tiered client/server (also called ‘distributed data’)
    - Requires a c/s programming language with built-in SQL connectivity to one or more database engines. E.g. Visual Basic, Delphi, PowerBuilder (client/server editions of each)
    - These APIs contain automatic generation of code templates to address system events, such as mouse clicks, mouse_over, … Merely add the code for the business logic. These APIs present many tools…
– Most of us have used an SDE perhaps without thinking about all this.
– *Visual Basic* is more than a programming language. It is an SDE based upon the Basic programming language.

– Similarly, Micro Focus COBOL is an SDE for *COBOL*,
– Visual Café’ and J Builder are SDEs for *Java*, and
– *Visual C++* is an SDE for *C++*. 
Process Architectures - more

- Sometimes application logic, in the form of **reusable procedures** can be stored (and executed) on the database server.
  - Clients can then ‘call’ for their execution on the server.

- **Positive**: stored procedures better enforce data integrity
  - Reusable and verifiable

- **Negative**: cloud the distinction between the application logic layer and data manipulation layer
  - The application logic is executing on the database server!
Clean Layering

Leads to the concept of **Clean Layering**: Many SDEs support clean layering, the requirement that the presentation, application, and data layers of an application be **physically separated** to allow components of each layer to be replaced or enhanced without affecting the other layers.

- Very important concept.
Process Architectures - continued

– SDEs exist for multi-tiered client/server
  • For enterprise-wide application development
  • Three-tiers and beyond…
  • Normally support > 100 users
    – Very fast response; large databases!!
  • SDEs here are very comprehensive comprising all the requirements needed for 2-tier and much more. (long list)
    – Very sophisticated tools for a variety of computing platforms, reuse, analysis and design, code generation tools, tools to assist partitioning application components between clients and servers, and scalability software.

– SDEs exist for Internet and intranet client/server
  • Standard technologies:
    – HTML – for document / page interfaces
    – XML – for transmitting and formatting data
    – CGI –
    – JAVA,,,, (e.g. Java’s “Visual Café”)
Strategies: Application Architecture Design Strategies

• **The Strategic or Enterprise-Oriented strategy**
  – Defines approved network, data, interface, and processing technologies and development tools
  – Defines a strategy for co-existence and/or integration of legacy systems and technologies
  – Provides for an on-going process to review and improve the above
  – Provides for a process to research and try emerging technologies that fall outside of the above
  – Provides an approval process for variances from the above

• **The Tactical or Application-Oriented Strategy**
  – Defines architecture for each new system on an application-by-application basis as needed.
  – Requires feasibility analysis for each application (covered in Chapter 9)
Method for Application Architecture Design – Steps!

• 1. **Draw a physical DFD** to represent the network architecture. Each physical process symbol will represent a client or server **processor**.
  
  We will represent a ‘class’ of clients with a single processor.

• 2. **For each physical process** on the above network architecture model, **draw** a physical DFD that shows the event processes (from Chapter 8) that are assigned to (or duplicated on) that physical processor.

• 3. **For appropriate processes** on the above system DFDs, **draw more detailed physical DFDs** that factor the events into design units. **This means almost all processes.**

• 4. **Draw physical, primitive DFDs** for appropriate processes from step 3.

(Adapted from the former STRADIS and AD/METHOD methodologies)
Design Units

A Design Unit is a self-contained collection of processes, data stores, and data flows that share similar design characteristics.

A design unit serves as a subset of the total system whose inputs, outputs, files and databases, and programs can be designed, constructed, and tested as a self-contained unit.

A design unit may be a program, subprogram, or a series of programs (left subtree of a structure chart of programs; package, perhaps) that can be treated as a ‘subsystem.’

Facilitates assigning to an individual(s) or team.

Ultimately, design units must be integrated into a whole system.
Prerequisites to Physical DFD Creation

- Need the logical data model (ER Diagrams in Chap 7)
- Need logical process models (DFDs from Chap 8)
- Need repository details for all of these (above)
- Design Constraints (one or more): Might include:
  - architectural standards predetermining choices of Data base systems, network topology, User Interface and/or processing models.
  - Project objectives
  - Feasibility of chosen or desired technology and methods.
The Network Architecture DFD

A network architecture is documented as a physical DFD that allocates processors (clients and servers) and possibly devices (machines and robots) across a network and establishes:

- See textbook – fig 11.11 for Network Architecture DFD
- See servers and their physical locations.
  (also, what they are running; whether anything is replicated, etc.)
- See clients and their physical locations
- See processor specifications (specify RAM, hard disk capacity, graphics display capabilities needed, …)
  (and what they are running; technologies needed to interface with servers)
- See communications protocols necessary (TCP/IP Ethernet…)


Data Distribution and Technology Assignments

- Distribute the data stores to the network processors.
- Already have logical DFDs. Need to know
  - where each will be stored, and
  - how they are implemented.
    - Have options:
      - Store all on database server;
      - Distribute tables on different servers
      - Store subsets of specific tables on different servers, and
      - Replicate specific tables or subsets on different servers.
    - Hosts of problems (and opportunities) here…
SoundStage Data Distribution DFD
Person Machine Boundary

- Factor out parts of DFD that are not computerized.
- Separate these functions from computerized functions.
- May be difficult when ‘part’ of processes may be both computerized and manual
  - May have to create additional processes as part of a totally manual process design unit.
  - If so, rename/redescribe computerized portion on DFD.
Recording a Person/Machine Boundary

Preprinted Form: Backorder

Win32 GUI: InvalidProductID

SQL Read: MemberData

SQL Update: MemberData

Win32 GUI: InvalidMemberID

MemberID + Address

SQL Read: OriginalDatedOrder

SQL Update: Order

VB + SQL: ValidateMember

VB + SQL: CheckOrdProdValid

ValidProductID

AvailableProductAndQty

OrderedProductID

OrderedProductQty

P

P

TotalOrdePrice

OrderStatus

TotalOrderPrice

CalcExtendedCost

PayableOrder

CalcOrdrTotalCost

V8 + SQL: CalcMemberCredit

OrderMethodAndAmount

OrderProdExtendedPrice

OrderProdAvailable

P

P

DB2: dbAccounts_Receive

DB2: dbOrders_Receive

DB2: dbMembership

DB2: dbAgreements

FilledOrder

FilledOrdrdProduct

SQL Update: Order

SQL Read: ProductPrice

SQL Read: ProductID

SQL Read: ProductAvailability

SQL Update: OrderProduct

SQL Update: InventoryCommit

SQL Update: OrderProductStatus

SQL Update: OrderProductID

SQL Read: Membership

SQL Update: MembershipCredits

SQL Read: AgreementStats

SQL Update: ActivityRecord

SQL Update: tblOrders

SQL Server: tblOrders

SQL Server: tblMembership

SQL Server: tblMembers

SQL Server: tblMemberships

SQL Server: tblAgreements

Warehouse

Preprinted Form: PickingTicket