Information Technology Standards and Architecture

Application Systems

To progress and stay competitive in the market, we depend on software developed within the Firm and purchases from outside vendors. To get the maximum benefit from our systems, we must make sure that these applications work together as effortlessly and efficiently as possible.
This part of the Technology Blueprint gives you guidelines for developing applications, and for choosing software products from the outside. This section also devotes special attention to developing applications that are not tied to a particular type of network.

Choosing a Model

Which criteria should I consider when choosing a model for a new application?
When you design a new application, your first and most important decisions deal with what type of system best fits your needs. This section discusses three major ways to implement an application system:
Client/Server
Host-based
Object-based
While it is impossible to specify one standard strategy, you should consider the following factors:

Process Flow vs. Data Location

If data and processing are typically consolidated in a single place (as in a General Ledger system, for example), then a Host-based model is a good choice. On the other hand, if data and processing are highly decentralized (as in a worldwide customer support system), then a form of Client/Server architecture should be considered.

Risk

As the complexity of the system increases, so does risk. Business risk increases as the system becomes more visible and available to external agents such as customers and suppliers. Business risk also increases when critical business data (orders, inventory) are exposed to technical risk.
You should minimize these risks in your planning. For example, your first attempt at a Client/Server system should not be an application that will stop the business if it fails.

Cost

Host-based systems generally have high setup costs, along with continuing costs for central support services. Client/Server systems often have a lower initial implementation cost; but because of higher complexity, less mature software, and decentralized implementations, they often generate higher support costs.

Training

Users, developers, and support staff must be trained on both the application and the parts of the technical architecture they will use. While general knowledge of host-based systems has accumulated over the last 10-30 years, we are still in the early stages of Client/Server systems. Therefore, money and time for training should be increased in budgeting and scheduling for Client/Server projects.

Flexibility

Host-based systems do not support rapid changes in system presentation or processing location. Client/Server systems are better able to accommodate these kinds of changes. (The shift to Client/Server systems has been largely fueled by the demand for responsive and flexible systems that are more under the control of the people actually doing the work.)

Client/Server Models

The first type of application system is the Client/Server model. This section describes the two major models of Client/Server architecture:
2-tier model
3-tier model
Since the 3-tier model has the most advantages for the Firm, this section goes into some detail on how to build and use 3-tier applications, and the best ways to convert 2-tier applications to use the 3-tier model.

What is Client/Server Architecture?

The term "Client/Server" has been misused to the point that it can mean almost anything. Here is a list of functions or applications that have been described with the term "Client/Server":

- Any program that happens to run under Microsoft Windows.
- Any program that happen to run on a Local Area Network.
- Multi-user data sharing on a File Server (using Paradox or FoxPro)
- PC access to a Relational Database Management System (SQL Server, Oracle Server, etc.)
- Applications communicating with other across networks using messaging services

Of these, only the last two can really be called Client/Server applications. These are (respectively) examples of 2-tier and 3-tier Client/Server models.

The 2-Tier Model

The 2-tier model is the typical design for Client/Server implementations today. A typical system of this type is a Windows application that uses a database API (Application Programming Interface) to access a local Database Management System. This model works well when developers want to use common development tools (Visual Basic or PowerBuilder) to build front end applications quickly.

But the 2-tier model breaks down in two cases:

1. When the company wants to use new Client/Server applications to connect to legacy systems on mainframe or mid-range computers. Typical GUI-based front-end development tools are not able to create and send on-line messages that can be processed by old applications.
2. When it becomes possible to access many remote database servers from the client applications. The management of multiple remote connections quickly becomes too complex to handle from the workstation.

You must take these limitations into account before you opt for the apparent simplicity of a 2-tier system.

The 3-Tier Model

The 3-tier Model is the best basis for distributable, scalable application systems. A "distributable" application is one that can run in many remote sites, so that the business processes can be implemented in more than one place. A "scalable" application is one that can be deployed to hundreds or thousands of sites without a change in architecture. Example: the key difference between the 2-tier and the 3-tier implementations of a database application is that in the 3-tier model, the client workstation is not directly connected to the database server. Instead, the clients send messages (transactions) to an application server. These messages are routed by a messaging middleware facility to the correct application program on the correct DBMS server.

The 3-tier model has numerous benefits:

- Connections to legacy applications can be made more easily.
- The client can use a single mechanism to send messages to new systems and legacy systems.
- Transaction performance and availability can be monitored more easily.
- The client application does not have to be designed specifically for one server database engine (since there is no direct connection between the two.)
- This model minimizes network traffic, which leads to better performance than comparable 2-tier systems, especially over a Wide Area Network (WAN).

Components of a 3-tier Application System

The three tiers of this type of system are:

- **Presentation Tier** The "front-end" or user interface
- **Application Service Tier** the layer that mediates between the user interface and the data repository or processing engine
- **Data Service Tier** the DBMS or data processing engine

Building the Presentation Tier

The presentation program is the part of the system that is most visible to the users – and most specific to a particular need or use. The presentation program must be built to run on different platforms that user different operating systems. This requirement can complicate the design process for both customer-oriented and internally focused systems. One solution to this problem is to keep the programming in the presentation tier to a minimum. This minimizes the amount of code that you must port from one platform to another. For example: when designing a presentation level application, avoid incorporating logic that defines and executes a business process, and the data that supports this process. This is just more code that will have to be converted or ported to another platform – and re-used in a different application that deals with the same business process.
Ideally, the presentation program should do nothing more than accept and display data on the desktop. The business logic and data should be handled for this and other presentation programs in the Application Server Tier.

Building the Application Server Tier

The middle or Application Server Tier in the 3-tier Client/Server model insulates the presentation level programming from the data server tier (or "back-end"). The Application Server Tier should contain code that is often re-used by the Presentation Tier. This part of the system should contain logic that represents the business processes. And, if this logic is to be accessed in run-time fashion, the Application Server Tier should be written in a language that will easily port to different presentation platforms.

You can also use middleware, in its various forms, to access the business logic of the application tier. Using middleware is another way to decouple the presentation logic from the application logic.

Accessing the Application Server Tier

When the Firm decided to use Middleware as a standard feature of its 3-tier model, we also decided to include an API as the standard interface between the Firm's applications and its Middleware. This simplifies the programming interface, and prevents our programs from being too tightly coupled to a specific vendor's Middleware.

Building the Data Service Tier

Usually, the Data Service Tier is the database of an application system. In the 3-tier model, the Data Service Tier handles data going into and out of the database; database integrity is handled within the database management system itself. Except for data integrity reasons, the Data Service Tier should be insulated from the business logic, as is the Presentation Tier.