

## **6 USER INTERFACE STANDARDS**

### **6.1 Introduction**

This Section defines standards to govern the development of the Project EASI/ED user interface. These standards will enable Project EASI/ED as a whole to present a consistent user interface to its user community.

These standards are intended to address the “look and feel” of the user interface, not how it should function, which is application-specific. However, particularly with respect to Web-based applications, this document provides standards with respect to techniques used in constructing the interface. Given the scenario of multiple contractors being assigned responsibility for specific systems and/or functional areas of Project EASI/ED as a whole, these standards will assure uniformity in the work products that are received by SFA.

The audience to which this document is directed includes those personnel within SFA who will be managing the effort to transform the existing Title IV systems, and the developers that will actually be performing the analysis, design, and construction activities. Consequently, at certain points in the document the content is of necessity quite detailed and technical in character.

This Section is organized into the following subsections:

- Subsection 6.1, Introduction
- Subsection 6.2, User Interface Standards Overview
- Subsection 6.3, Project EASI/ED User Interface Requirements and Design Goals
- Subsection 6.4, Project EASI/ED User Interface Standards

### **6.2 User Interface Standards Overview**

This subsection provides an overview of user interface standards for Web-based, client/server, interactive voice response (IVR), and interactive facsimile applications within Project EASI/ED. It is organized into the following subsections:

- Subsection 6.2.1, Overview of Interface Components and Standards
- Subsection 6.2.2, Regulatory and Government User Interface Standards and Guidelines

#### **6.2.1 Overview of User Interface Components and Standards**

The following subsections provide an overview of user interface components and standards—industry best practices, technologies, issues, and representative example products. The purpose of this discussion is to provide readers with a technical context for better understanding the Project EASI/ED system-wide user interface standards. It is organized into the following subsections:

- Subsection 6.2.1.1, User Interface Standards for Web-based Applications
- Subsection 6.2.1.2, User Interface Standards for Client/Server Applications
- Subsection 6.2.1.3, User Interface Standards for Interactive Voice Response Applications
- Subsection 6.2.1.2, User Interface Standards for Interactive Facsimile Applications

### 6.2.1.1 User Interface Standards for Web-based Applications

This subsection provides an overview of standards that apply to the development of Web-based applications, in order to establish a technical context for the Project EASI/ED system-wide user interface standards for Web-based applications that are presented in Section 6.4.1 below. This discussion assumes a basic familiarity with the concepts and terminology associated with Web-based applications.

It is critically important to recognize that this subsection addresses standards for user interface (UI) design for Web-based application systems, as opposed to standards for layouts of simple “static” Websites. The difference is that with the former, a user is attempting to purposefully accomplish something, whereas with the latter the user is essentially a passive observer (much as if the user were watching television). Thus, the primary goal of a user interface for a Web-based application is to help the user accomplish a task, not capture the user’s attention with eye-catching design features.

The following topics are addressed in this subsection:

- **Application Architecture.** The architecture of the World Wide Web will be explained and related to other variants of client/server distributed systems architecture, and the implications for Web-based applications will be discussed.
- **HTML.** The different standards in use for HTML will be defined, with specific emphasis on the directions being taken by the World Wide Web Consortium (W3C) as it evolves the language.
- **Cascading Style Sheets (CSS).** The evolving standards for controlling the stylistic elements of HTML page layout will be discussed.
- **Dynamic HTML (DHTML).** The W3C’s Document Object Model (DOM) and the capabilities it provides for developers to create dynamic, interactive Web content will be explained.
- **XML.** The significance of this “metalanguage” will be discussed, particularly in relation to the future of HTML and the manipulation of data in Web-based applications.
- **Javascript.** The origins and capabilities of this powerful, industry-standard scripting language will be explained, particularly as they apply to the development of Web-based applications.
- **Frames and Framesets.** The strengths and weaknesses of this technique of employing multiple “panels” within a browser window will be examined, especially with respect to application design.
- **Multiple Application Windows.** This technique of using multiple browser instances to replicate the conventional multi-window client/server application interface will be discussed. Although not a standard *per se*, the use of multiple application windows is a common technique in Website design.

This subsection provides a comprehensive overview of the standards and issues relevant to the development of user interfaces for Web-based applications, ranging from how the user interface relates to the rest of the application architecture down to specific standards with respect to details of the “look and feel” for the user interface.

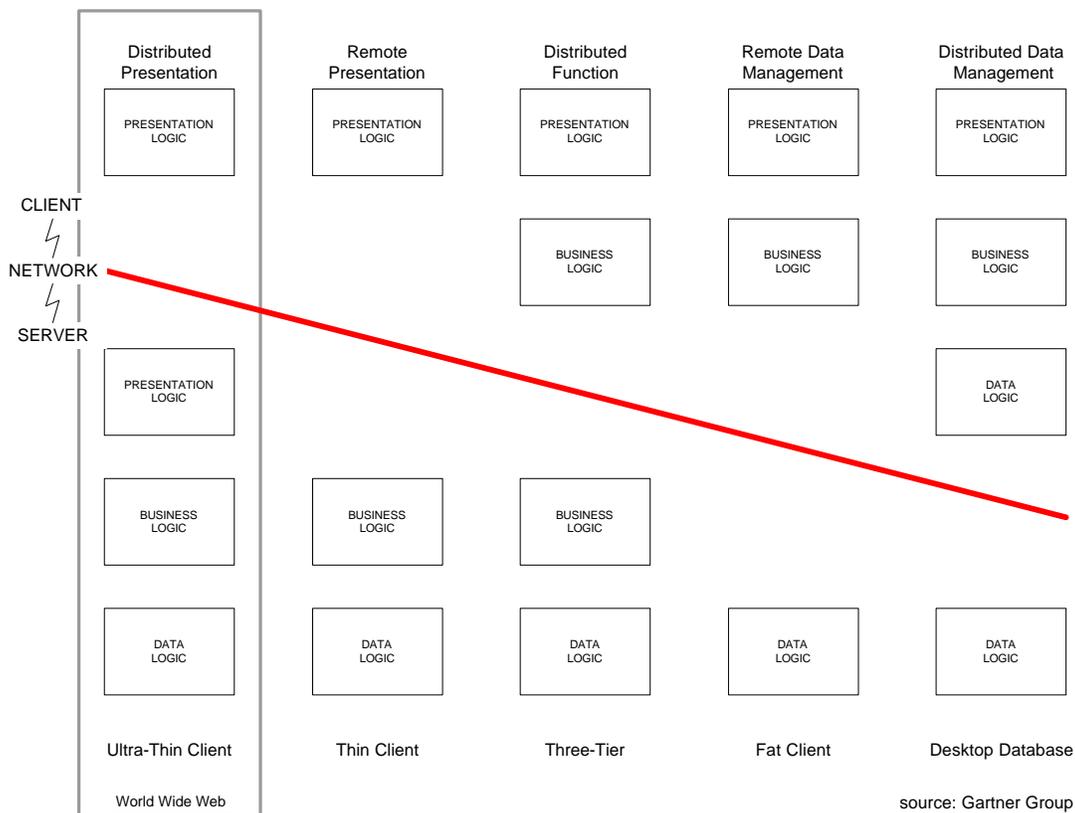
## Application Architecture

In order to discuss Web-based applications, one must first discuss the concept of “application partitioning.” Application partitioning refers to the fact that a client/server software application can be partitioned into three logical “tiers”:

- Presentation logic, which controls the display of information on the desktop “client”
- Business logic, which controls how information is processed through the implementation of business rules
- Data logic, which controls how information is stored in the database

The Gartner Group defined a model that shows the five different ways that client/server applications can be partitioned in terms of these logical tiers.

As shown in the diagram below, the World Wide Web is simply a variant of client/server architecture, the “Distributed Presentation” variant in the far left column, characterized by having only presentation logic resident on the client, and having even that partitioned between client and server. Ironically, when the Gartner Group first published this model in the early 1990s, the



**Figure 6-1: The Gartner Group Model of Application Partitioning**

World Wide Web did not yet exist, and the Distributed Presentation variant referred to the use of “screen scrapers” that translated character-based screens into a graphical user interface.

However, the Web functions in a virtually identical way, since the presentation logic (in this case, an HTML file) is downloaded across the network and rendered by the Web browser, which is a generic engine for displaying downloaded presentation logic.

The Web browser offers the potential to serve as the “Universal Client” for distributed applications. The software industry has seized upon the Web as a way to construct distributed systems, based on the use of “application server” technology in architecting applications. Application server technology, such as the Sun/NetDynamics Java application server or the Oracle Web Application Server, utilizes an “n-tier” *physical* architecture comprising desktop clients, Web server(s), application server(s), and database server(s). The business logic and the majority of the presentation logic reside on the application server(s) as opposed to the desktop client of the end-user. The business logic executes entirely on the application server, and the presentation logic is downloaded from the application server to be displayed within the Web browser. This is a tremendously scalable and flexible method of implementing distributed systems.

In particular, this approach offers the benefits of tremendous simplification—and commensurate cost reduction—in terms of installation, configuration and maintenance. No application client software other than a Web browser needs to be installed on each desktop, and browsers are now available as “freeware” from both Microsoft and Netscape. When modifications are made to the application, they are made to the presentation logic and business logic on the application server, and require no changes to the configuration on the desktop.

There are issues, however, with respect to constraints on how an application can be developed using this “distributed presentation” architecture. In a conventional client/server application constructed with tools such as PowerBuilder or Visual Basic, the operating environment for the *frontware* (the client application that contains the presentation logic and typically some degree of the business logic) is almost always known in advance and is usually a carefully controlled factor. What distinguishes Web-based applications from conventional client/server applications more than any other criterion is the fact that developers have no control over the environment in which the client-side application will operate. There is a great deal of variability among the Web browsers from different vendors, between versions from the same vendor, and even between implementations of the “same” version from the same vendor on different operating systems.

This variability is rooted in the evolving standards that extend the capability of the Web beyond merely the display of static text, and the degree to which the major vendors (i.e., Netscape and Microsoft—various estimates put their combined market share between 75% and 99%) adhere to these standards in their respective products. The standards, and the issues associated with them and the way they are implemented, are discussed in detail below.

## **HTML Standards**

The World Wide Web Consortium (W3C, <http://www.w3.org>) is an international industry consortium, led by Tim Berners-Lee, creator of the World Wide Web. Services provided by the W3C include a repository of information about the World Wide Web for developers and users; reference code implementations to embody and promote standards; and various prototype and sample applications to demonstrate use of new technology. The W3C is vendor neutral, working with the global community to produce specifications and reference software that is made freely available throughout the world.

The current specification (labeled a “recommendation”) from the W3C for HTML is for version 4.0. HTML 4.0 builds on earlier versions (specifically HTML 3.2 and HTML 2.0) and includes new or enhanced features in the following areas:

- CSS (explained in detail below)
- Internationalization features
- Accessibility features
- Tables and forms
- Scripting and multimedia

In addition, there are three variants of HTML 4.0, and the variant that is in use is specified by the Document Type Definition (DTD) in the header of the HTML document. The three variants are:

- Transitional — Retains the use of deprecated (i.e., soon to be obsolete) syntax in order to provide downward compatibility with older browsers. These deprecated tags are primarily in the area of layout, such as the BODY tag with its bgcolor, text and link attributes.
- Strict — Relies completely CSS for layout, with no reliance on deprecated tags.
- Frameset — Employed when HTML frames are used in the design of the HTML documents (see below for further discussion on the use of frames).

The version of HTML most consistently completely supported by the widest range of Web browsers is HTML 3.2, which specifies the following features:

- Head
- Body
- Block-level elements
- Lists
- Tables
- Text-level elements
- Frames within framesets (developed by Netscape and later incorporated into the HTML 3.2 specification)

The W3C is currently working on reformulating HTML as an application of eXtensible Markup Language (XML). This will permit the modularization of HTML into building blocks that can be used as needed by content developers. Content developers will be able to focus on utilizing HTML components that are known to be supported by the targeted device (e.g., conventional Web browsers, handheld computers, portable phones) as opposed to attempting to develop HTML syntax that provides the necessary functionality even though certain features may or may not be supported on specific platforms. XML is discussed further below.

Essentially, the W3C is attempting to stratify the construction of documents so that HTML is used for structural markup features (headings, paragraphs, lists, hypertext links, etc.), CSS and dynamic HTML are used for layout, and the DOM defines how elements can be manipulated at runtime via scripting. This will increase the feature set available to content developers and eliminate some of the practices regularly used by content developers as a workaround solution for layout (such as the use of tables for positioning purposes), and thereby increase the likelihood that

the document will appear as intended across platforms. CSS, DHTML and the DOM are discussed in detail below.

## **Cascading Style Sheets**

CSS is essentially the HTML analog for the page layout parameters used for document formatting in leading word processors. The “cascading” term refers to the fact that the style definitions in the style sheet override stylistic elements in other parts of the HTML document (e.g., the <BODY> tag attributes).

Despite the fact that the first draft of the CSS specification was completed in 1996 with the cooperation of Microsoft and Netscape, neither vendor has an adequate implementation of the first CSS specification (CSS1), and the W3C has already released CSS2. Because of the inconsistent support for CSS among vendors and between versions from any given vendor, the use of CSS is not widespread at this time. However, the W3C has just released a CSS1 Test Suite to enable Web developers to validate the CSS1 support in their browsers and for product vendors to improve CSS1 conformity.

The inability of the major browser vendors to completely implement the CSS technology without proprietary limitations has left Web designers with a difficult choice. They can utilize CSS and attempt to accommodate browsers that cannot parse the CSS syntax. Or, they can resort to layout and formatting techniques that use features (primarily the functionality associated with HTML tables) in ways that are slow, subject to variable results across browser versions and across vendors, and not at all in accord with the intended purpose.

It remains to be seen whether the release of the CSS1 standard conformity test will ensure that the forthcoming version 5.x browsers from Netscape and Microsoft accurately and completely implement the CSS1 standard. Work is also proceeding on the eXtensible Style Language (XSL), which will provide the same flexibility and expanded capabilities with respect to stylistic elements that XML promises with respect to document markup

## **Dynamic HTML**

DHTML offers the potential to provide Web developers with the capability to turn Websites into a dynamic, interactive medium. However, again, Microsoft and Netscape released significantly different implementations in their version 4.x browsers.

DHTML rests upon the DOM. As defined by the W3C, the DOM is “a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents.” The informal “Level 0” version of the DOM is equated to the functionality exposed in Netscape Navigator 3.0 and MS Internet Explorer (MSIE) 3.0. The “Level 1” specification builds on this existing technology by specifying functionality for document navigation and manipulation of the content and structure of HTML and XML documents. The Level 1 specification is now a W3C “recommendation”. Work is proceeding on the Level 2 specification.

Netscape based its Navigator 4.x implementation of DHTML on a set of proprietary extensions to HTML and Javascript on the one hand, and a proprietary “Layer” element on the other. Netscape also developed a proprietary method (Javascript Accessible Style Sheets, or JASS) to associate style sheets with HTML pages. Both the Layer element and JASS were decisively rejected by the W3C, and Netscape subsequently agreed to support the W3C standards. Microsoft based its

implementation of DHTML largely on the Level 1 specification, but failed to implement the specification completely, and included proprietary MSIE-only extensions as well.

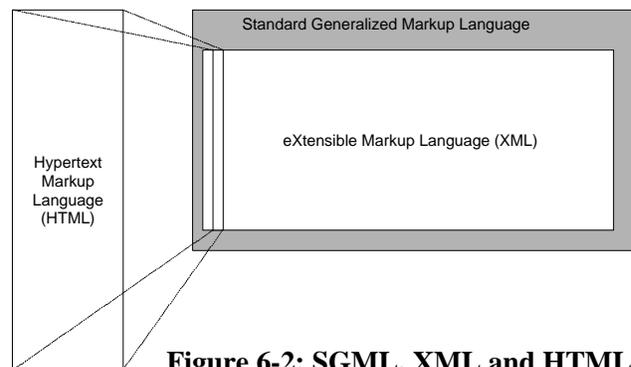
For all its tremendous potential for creating dynamic, interactive interfaces, DHTML is “not ready for prime time”, primarily because Netscape and Microsoft have thus far steadfastly refused to consistently and completely implement the standard. As with CSS, it remains to be seen whether the forthcoming version 5.x browsers from Netscape and Microsoft will accurately and completely implement the DOM Level 1 standard and associated DHTML functionality.

## eXtensible Markup Language

XML is not in itself a single markup language like HTML. It is instead a set of rules for designing a markup language, or adding extensions to existing markup languages like HTML. XML was developed using the Standard Generalized Markup Language (SGML), which is codified as ISO 8879, the standard from the International Organization for Standardization (ISO) for defining descriptions of the structure and content of different types of electronic documents. HTML itself is a “document type” defined in SGML, and (as noted above) is being reformulated into XML to make it more adaptable across platforms. According to the XML specification, XML was designed “to make it easy and straightforward to use SGML on the Web: easy to define document types, easy to author and manage SGML-defined documents, and easy to transmit and share them across the Web.”

As shown in Figure 6-2, SGML can be used to describe literally thousands of different document types, and XML is a subset of SGML that is optimized for defining new document types and supporting delivery and interoperability over the Web. HTML is just a single document type among the countless number that can be defined using XML.

The virtues of XML are that it frees Web developers from dependence on a single, inflexible document type (HTML), and that it streamlines the powerful but hard-to-program syntactical constructs of SGML.



**Figure 6-2: SGML, XML and HTML**

The specific benefits of XML include:

- Web developers can design their own document types, as opposed to being restricted to using HTML. DTDs can be tailored to a specific purpose, so that the cumbersome tricks required in HTML to achieve special effects should no longer be required: Web developers can invent their own markup elements.
- The hypertext linking features of XML are vastly superior to those of HTML. Specifically, unlike HTML, XML allows Web developers to specify the *type* of link using one of the following five types: Simple, Extended, Locator, Group, or Document. Each of these types has multiple attributes that greatly refine the control that developers have. In fact, the Extended type allows one-to-many links, and the Group type allows one-to-many links where the child links are treated as a set. Thus, the concept of a link has evolved towards supporting a database-style structure.

- XML can provide improved presentation and performance features on the Web.
- Information will be more accessible and reusable, because the flexible markup of XML can be interpreted by any XML-compliant software, as opposed to being subject to the kind of proprietary extensions and limitations that have plagued CSS and DHTML.
- XML files are true SGML files, and so can be utilized in other environments beside the Web.

This is obviously a powerful combination. Driving an XML document by using Javascript to manipulate the DOM, one could incorporate powerful new functionality on the client side (such as altering the display or even sorting data) that is independent from the server. Furthermore, it would be theoretically possible to update parts of a displayed page via contact with the server without having to refresh the entire page. This would greatly increase the throughput of the Web by reducing the substantial overhead associated with having to refresh an entire page any time one of its elements changes.

XML is presently still in the experimental stages. The open-source beta of Netscape 5 (code named “Gecko”) and Beta 2 of MS Internet Explorer 5 both nominally support XML. However, it again appears that Microsoft is retaining its own proprietary extensions to the DOM, which has a potentially dramatic impact on the portability of XML documents.

### **Javascript**

Javascript is described as a “scripting” language instead of a programming language, which can mislead people into thinking it is as simplistic as something like MS-DOS batch commands. In fact, Javascript is a full-featured, object-oriented programming language of great power. While the name “Javascript” is owned by Netscape (Microsoft calls its implementation “Jscript”), the language itself has been standardized by the European Computer Manufacturers Association (ECMA) as ECMA-262 and is on the fast track for standardization by ISO as ISO-10262. The standard version is labeled ECMAScript to avoid any bias towards a particular vendor.

There are two types of Javascript: client-side and server-side. Client-side Javascript is embedded in the HTML document, either at construction or at runtime using an INCLUDE file. In conjunction with the DOM, it gives developers the ability to dynamically manipulate virtually all aspects of an HTML document at runtime. Server-side Javascript is an alternative to Common Gateway Interface (CGI) scripts that supports such things as file and database access on the server side, and dynamic generation of HTML pages in response to previous events. However, it is a much more proprietary implementation than client-side Javascript and is not nearly as widely used.

Unfortunately, while some progress has been made in tools like Microsoft Frontpage and Allaire Homesite in adding support for Javascript/Jscript, there is still as of this writing no tool that provides an integrated development environment (IDE) including an editor and debugger for Javascript programming. Thus, any sophisticated coding in Javascript must be debugged largely by hand, and tested across a variety of browsers for compatibility.

### **Frames and Framesets**

Although the ability to implement frames and framesets is not a “standard” *per se*, it has been incorporated as a capability supported under HTML 3.2 and now HTML 4.0. What makes this

ability worthy of special attention is that the use of frames is problematic from a number of perspectives.

Frames and framesets were an innovation from Netscape. Essentially, framesets are an implementation of the Microsoft Multiple Document Interface (MDI) standard. Within the “parent window” of the browser itself, an initial HTML document is loaded that defines and instantiates the frameset. The frameset can consist of any number of frames, which are equivalent to “child windows” in MDI terminology. Typically at least one of these frames contains an HTML document with a menu, and selections from that menu cause other HTML documents to load in one of the other frames, while the contents of the first frame remain constant. This essentially allowed developers to develop Websites almost as if they were conventional client/server interfaces with sophisticated menu bars and data windows. It simplified layout, because the frames enforced layout designs that otherwise could only be enforced through the use of tables for formatting content. It also simplified sophisticated Javascript coding as well, by encapsulating document events within each separately framed HTML document.

While on the face of it this divergence from the document model that is at the heart of the hypertext transfer protocol (HTTP) and the Web may seem to be a positive evolutionary leap, it has proven in many cases to be more trouble than it is worth. This is best exemplified by the fact that even Netscape no longer uses frames on its own public Website.

There are many problems posed by the use of frames. One of the key issues is loading speed. When a site uses frames, it must load (at least) three pages instead of one when it is first contacted: the first page defines the frameset, and the second and third pages populate the frames that have been defined for the frameset. From then on, there is a one-to-one equivalence with non-frame sites in terms of page loading speed.

Another issue is how a site with frames is handled by text-only browsers such as Lynx, which are typically used by the visually impaired to drive text readers and which do not support frames. The W3C Web Accessibility Initiative (WAI) guidelines specify that, at a minimum, a non-frames version should be supplied either as a link from the frames version or, preferably, as an automatic destination with the <NOFRAMES> tags. As with any HTML, if the browser does not recognize the element, it ignores it. Non frame-capable browsers would ignore the frameset definition—including the <NOFRAMES> tags—but will display whatever is enclosed in the <NOFRAMES> tags (which can be any HTML at all) because that is what is recognized. On the other hand, frame capable browsers will preferentially display what is set up by the frame elements, unless the browsers provide a mechanism where the display of frames can be turned off, in which case they may display this alternative content. In any event, it means that two separate versions of the Website must be maintained.

Finally, an issue that regularly causes confusion is the performance of the “BACK” button on the browser when used within a site with frames. Early browsers (version 2.x and 3.x) from Netscape and Microsoft would call up the previous Uniform Resource Locator (URL) or frameset—not the previous frame within a frameset—when the “BACK” button was pressed. This situation has somewhat improved with the 4.x browsers from both vendors.

## Multiple Application Windows

Another feature supported by HTML 3.2 that is not a standard *per se* but that is worthy of attention in its own right is the ability to utilize multiple application windows. Specifically, the

developer can designate that a link to a target document loads that document into a new browser window, rather than replacing the calling document within the existing browser window. This feature creates a new instance in memory of the browser itself to host the target document. There are a number of shortcomings associated with this feature.

The primary drawback to the use of this feature is that by default the new instance of the browser appears exactly over the existing instance of the browser. Thus, the user may assume that the target document has simply replaced the calling document within the existing browser, and be baffled—and rapidly irritated—when the “BACK” button is suddenly disabled.

A drawback that is nearly as serious is the impact on system resources of opening a second instance of the browser. Unless the user’s system is equipped with a large amount of memory, a significant degradation of performance is to be expected when more than one instance of a browser is loaded. This performance degradation can be minimized by restricting the features instantiated in the new instance (e.g., specifying that the new instance of the browser not contain toolbars or that it be small in size). Nevertheless, it is possible to repeatedly open up new browser instances until the system runs out of memory and crashes. This situation can be rendered less likely by using a defined, constant name for the new browser instance—thus repeated calls to the target document will simply reload it in the named browser instance instead of creating another new instance. However, if the application offers numerous opportunities to open new browser instances it risks freezing or crashing the user’s system.

## **Conclusions**

The standards that have been examined above—HTML 4.0, CSS, DHTML, the DOM, and XML, along with the capability to utilize frames and multiple application windows—offer tremendous potential for enabling Web developers to create robust, dynamic user interfaces with virtually all the capabilities associated with conventional client/server GUI interfaces and more. However, primarily because of vendor intractability, the implementation of such standards has been incomplete and/or tainted by proprietary extensions. Furthermore, most typical users do not automatically upgrade their browsers—or any software—as soon as a new version is released, and upgrading a browser in particular usually entails downloading a 20-30MB file on a 28.8 – 56Kbs connection. These factors have rendered the use of such cutting-edge capabilities hazardous at best.

### **6.2.1.2 User Interface Standards for Client/Server Applications**

The following subsections provide an overview of standards that apply to the development of user interfaces for client/server applications, in order to establish a technical context for the Project EASI/ED system-wide user interface standards for client/server applications that are presented in Section 6.4.1 below. This discussion assumes a basic familiarity with the concepts and terminology associated with client/server applications.

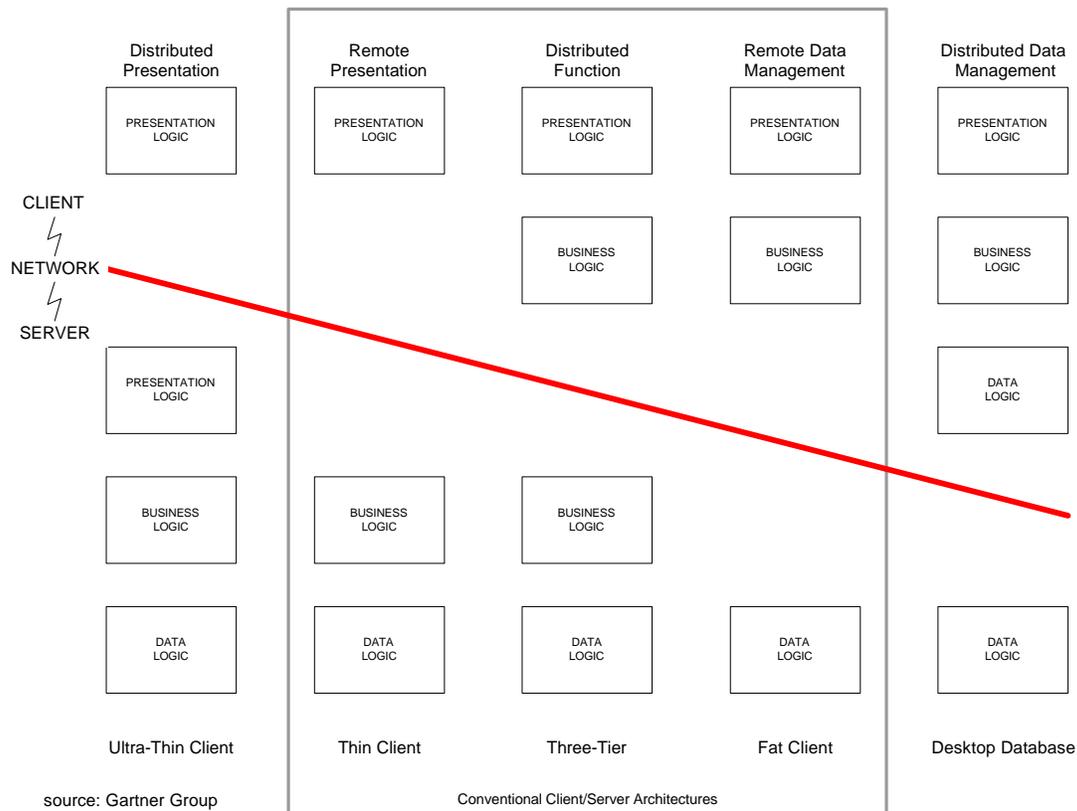
The following topics are addressed in this subsection:

- **Application Architecture:** The evolution of client/server architecture will be discussed, with specific emphasis on its implications for user interface development.
- **Graphical user interfaces (GUIs):** The components of GUIs will be explained.
- **Industry standards:** The major industry standards for GUI design will be presented.

## Application Architecture

Client/server architecture was pioneered with the release of Sybase SQL Server in 1987. Until then, applications using relational database management systems (DBMSs) such as Oracle and Ingres were entirely host-based (e.g., COBOL/DB2 on mainframes and C/Oracle on UNIX). Sybase introduced a number of innovative features in the SQL Server product, but foremost among these was an open Application Program Interface (API) that let any type of software running on a desktop access the database remotely through a network as long as the software used the API, which enabled developers to leverage the power of software running on personal computers. Thus, the ability to partition an application between client and server was introduced, and it revolutionized information technology.

As shown in the Figure 6-3 below, the Gartner Group Model of Application Partitioning, there are three variants of this architecture that are considered “conventional” client/server. As discussed above, the Distributed Presentation variant originally referred to the use of “screen scrapers” that translated character-based screens into a graphical user interface. However, the Web functions in a virtually identical way, since the presentation logic (in this case, an HTML file) is downloaded across the network and rendered by the Web browser. Conversely, the Distributed Data Management variant refers to LAN-based implementations of “desktop DBMSs” such as dBASE, FoxPro and Clipper, where only the shared data files reside on a



**Figure 6-3: The Gartner Group Model of Application Partitioning**

regular network file server.

Initially, the industry approach to client/server architecture used the Remote Data Management configuration, which was labeled "two tier" (also known as "fat client"), where the data logic resided on a server and the business and presentation logic resided on each client. The business and presentation logic was written in Fourth Generation database programming languages (4GLs) such as Sybase Corporation's Powerbuilder, Gupta's SQLWindows and Microsoft's Visual Basic, or Third Generation languages (3GLs) like C or C++.

There were two main disadvantages to this kind of architecture. First, it did not scale well beyond about two hundred users, because the DBMS on the server had to manage connections as well as data handling. Second, each installation of the application logic had to be individually maintained on each desktop client. The software engineer had to plan for version upgrades and ensure that each client was running the most current release of the business logic. This led to serious distribution and version control problems.

Scalability was also a critical problem, and the solution that arose was a three-tier physical architecture that introduced the concept of "middleware". The middleware could provide two types of services. First, transaction monitors like BEA's Tuxedo or IBM's CICS/6000 could offload the management of connections from the server. Second, some or all of the business logic of the application could be moved from the client to the middleware in the form of an "application server". The clients would therefore execute a minimum of the overall system's components. The user interface and some relatively stable business logic were the only things that were executed on the clients. This greatly simplified client configuration and management.

The term "n-tier" is becoming common to describe physical architectures. While three tiers are the maximum for *logical* partitioning, "n-tier" refers to the fact that a client can theoretically connect to more than one application server, each of which can in turn connect to many servers. The message flow can get quite complex. The ultimate example of this is the Common Object Request Broker Architecture (CORBA), where a client can request an object from an object request broker (ORB), which in turn locates and instantiates the object, which in turn may access one or more data stores. The n-tier physical architecture also makes it easier to support heterogeneous client devices because the client software only includes the presentation layer, whereas the business logic and data logic reside on the server side of the network. The main disadvantage of n-tier client/server is the complexity of designing and developing the application. Frequently the most difficult aspect of this type of client/server application design is the process of correctly partitioning the application.

## **Components of Graphical User Interfaces**

GUIs are presentation interfaces between users and application software that are designed to show information to users in graphical form. A comprehensive GUI environment includes four interrelated components:

- The graphics library provides a high-level graphics-programming interface.
- The user interface toolkit, built on top of the graphics library, provides application programs with mechanisms for creating and managing the dialog elements of the WIMPS (windows, icons, menus, pointers and scroll bars) interface.
- The user interface style guide specifies how applications should position the dialog elements to present a consistent, easy-to-use environment to the user. The standards

defined for user interface services are needed to support a consistent look and feel in products from different vendors.

- Consistent applications ensure that the user will know how to navigate within an application, and what to expect in response when various controls are used.

Currently available GUIs present the developer with a broad range of capabilities for obtaining input and displaying the outputs. These range from the appearance of screen entities such as windows and buttons down to the actions that users must perform to navigate the display and utilize the application's functions. While this breadth of capability is powerful, it permits applications that are built using the same GUI to appear the same but behave in different ways.

### **Industry Standards for GUI Design**

Presently the two main contending standards for GUIs are MS Windows, promoted by Microsoft, and X Windows, promoted by The Open Group. MS Windows is a proprietary technology of Microsoft, while X Windows is an open standard for UNIX operating systems. With the dominance of MS Windows in the Desktop market, MS Windows systems are also being used as a front end to UNIX systems that would traditionally use X Windows. Currently, the MS Windows 95 interface model is dominant in the industry. An emerging standard in this area is the use of active Desktops and integrated Browsers as provided with MS Windows 98 and MS Windows NT5.

The following tables present the relevant standards for the Windows and UNIX environments. Table 6-1 presents standards applicable within the Windows environment, which is Microsoft-proprietary.

Table 6-2 presents standards applicable within the UNIX environment, which is much more heterogeneous. It is not a coincidence that standards are issued and supported by organizations such as The Open Group (itself a result of the merger between the Open Software Foundation and the X/Open Company, Ltd.), which are consortia of different UNIX hardware and software vendors.

Area	Standard	Comments
Toolkit and Related Tools.	Microsoft Win32 API	Win32 is the Windows API (set of procedure calls and associated event mechanisms). It provides facilities for windows creation, style, manipulation and user interaction. It also provides toolkit component creation (push-buttons, radio-buttons, check-boxes, scrollbars, entry fields, drop-down list, list boxes, menus, etc.), associated user interaction (pushing buttons, etc.) and other facilities.
Application to User Protocol.	“Windows Interface Guidelines for Software Design”	This publication is produced and marketed by Microsoft Press.
Application to Application Protocols	Object Linking and Embedding (OLE 2.0).	OLE is Microsoft’s application level protocol designed to enable applications to contain objects within them that are created and manipulated by another application.
	ActiveX	ActiveX is Microsoft’s brand name for the technologies that enable interoperability between applications using the Component Object Model (COM) and the Distributed Component Object Model (DCOM).

**Table 6-1: Selected User Interface Standards for MS Windows**

Area	Standard	Comments
Toolkit	<p>Motif Toolkit API - The Open Group C320, Apr 95</p> <p>X Toolkit Intrinsic - The Open Group C509, May 95</p>	<p>The Motif Toolkit API is built on the base facilities provided by the Xlib interface and protocol. This API provides facilities for using the toolkit components (e.g. push-buttons, scrollbars, menus etc.) and responding to user-interactions (pushing buttons, etc.). It also includes a windows manager (an application enabling the user to resize and move windows). Motif is built on the facilities provided by the X Toolkit Intrinsic "toolkit for toolkits". Applications are likely to call Xt directly.</p>
Application to Application Protocols	<p>Window Management (X11R5): File Formats and Application Conventions The Open Group C510, May 95</p>	<p>Inter-Client Communications Convention Manual (ICCCM) is a guide for applications to run in X Windows environments. It defines conventions for inter-client communications (e.g. selection mechanism, cut buffers, window management, etc.). Conditional on this facility being required.</p>
Application to User Protocol	<p>Motif 2.1 Documentation - Full Set The Open Group T252, Oct 97</p> <p>XCDE Definitions and Infrastructure The Open Group C324, Apr 95</p> <p>XCDE Services and Applications The Open Group C323, Apr 95</p> <p>Calendar and Scheduling API (XCS) The Open Group C321, Apr 95</p>	<p>The Motif 2.1 Document Set includes the combined CDE 2.1 and Motif 2.1 users and Style Guides, the Motif 2.1 Programmer's Guide, Programmer's Reference and Widget Writer's Guide. The set provides guidelines for the creation of a consistent user interface for applications that run Motif. It specifies guidelines on how to develop a conformant application.</p> <p>Common Desktop Environment defines a consistent set of APIs for a common desktop environment that can be implemented on operating environments which support X Windows and OSF/Motif. CDE provides end users with a consistent graphical user interface, capable of supporting advanced multimedia applications. It provides software developers with a consistent set of APIs, including X11 and Motif. Conditional on requirement existing for a common desktop user environment in UNIX environments.</p>
Application to User Protocol (Cont.)	<p>Window Management (X11R5): Xlib - C Language Binding The Open Group C508, May 95,</p> <p>Window Management (X11R5): X Window System Protocol The Open Group C507, May 95.</p>	<p>The Open Group's CDE 2.1 Documentation - Full Set, T253, contains the combined CDE 2.1 and Motif 2.1 User and Style Guides, as well as the CDE 2.1 Programmer's Reference and Application Developer's Guide.</p>

**Table 6-2: Selected User Interface Standards for X Windows**

### 6.2.1.3 User Interface Standards for Interactive Voice Response Systems

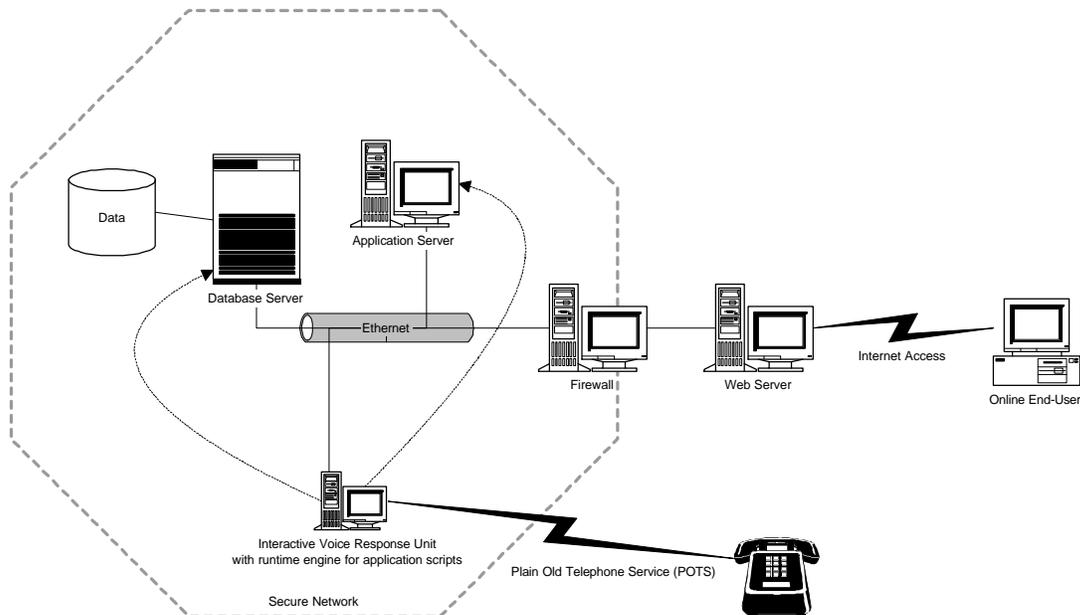
The following subsections provide an overview of standards that apply to the development of user interfaces for IVR applications, in order to establish a technical context for the Project EASI/ED system-wide user interface standards for IVR applications that are presented in Section 6.4.1 below. This discussion assumes a basic familiarity with the concepts and terminology associated with IVR applications.

The following topics will be addressed:

- **Application architecture.** The architecture that supports an IVR application will be explained.
- **Industry standards.** The industry standards relevant to IVR interface design will be presented.
- **Representative products.** Examples of applicable technology will be offered.
- **Evolving standards.** New directions for IVR technology will be discussed.

#### Application Architecture

An IVR system is one where the user utilizes a telephone to access the system. Menus are recorded during development of the IVR application, and presented by the interactive voice response unit (IVRU). The IVRU is typically a dedicated platform that handles incoming calls and transmits updates and queries through the network either to an application server or directly to the database server. Figure 6-4, Example of IVR Application Architecture, illustrates one possible configuration.



**Figure 6-4: Example of IVR Application Architecture**

Many IVRUs come with proprietary scripting languages that support the construction of menus and functions (e.g., MasterVox from MasterMind Technologies). The scripts run on top of a runtime engine, hosted on the same platform as the telephony circuit boards, that interprets the scripts and enables direct database access. If, however, the IVR application must access an application server in order to leverage the business logic there, then the application server must provide an API and the IVR application must typically be custom-coded, usually in a 3GL such as C/C++.

The results of queries are either presented verbally to the user (the IVRU assembles the correct phraseology out of phonemes and/or word lists) or faxed back to a number the user has provided. IVR applications range from simple front-ends that replace a human switchboard operator to complex interfaces with dozens of hierarchically arranged menu choices that allow users to view account information, fulfill requests, produce reports and interface with host systems.

### Industry Standards

There are several industry standards that are applicable to the development of IVR applications. These are presented in Table 6-3.

Standard Title	Organization / Standard Name	Description
Telephony Application Programming Interface (TAPI).	Microsoft TAPI: 1993	TAPI links the telephone to the computer at the operating system level.
User-System Interfaces and Symbols committee.	ANSI X3V1: 1993	Voice recognition standard.
Digital audio signal interchange: Uncompressed digital audio systems.	ISO/IEC JTC, 1/SC18/WG9	The audio Engineering Society's working group 9 is working on a VMUIF.

**Table 6-3: Interactive Voice Response System Standards**

### Representative Products

Table 6-4 presents products that support interactive voice response systems.

Product Name	Vendor	Product Type	Applicable Standard
Internet Phone Release 4 with video	VocalTec Communications	Telephony	TAPI
MasterMind	MasterMind Technologies	Telephony	TAPI
InfoPress Voice Response	Castelle	Telephony	TAPI
Voicetel Generations	Voicetek Corporations	Telephony	TAPI

**Table 6-4: Representative IVR Products**

## Evolving Standards

Within the past year, Motorola released a new Web-based technology called VoxML to add an IVR interface to Web applications. This software application is based on the XML standard, and basically replicates the graphical Web browser and HTML presentation language, so that the “voice browser” can access the same Web server as the graphical browser does, with all the associated functionality that the graphical browser can access. The primary selling point of this technology is that now the same business logic and access pathways can be shared between a conventional Web application and an IVR application, even if the presentation logic may differ somewhat.

This product is relatively new to the market (i.e., it is not a mature technology) and has not yet had the exposure necessary for it to become a standard. There are not enough users of this product to warrant labeling it a standard, but Motorola is actively working with other leading telecommunications and speech technology developers to create a broadly supported markup language for voice applications. It is likely that within two years, there will be a widely supported, robust IVR-specific markup language that will truly become a standard.

### 6.2.1.4 Interactive Facsimile

The following subsections provide an overview of standards that apply to the development of user interfaces for interactive facsimile (fax) applications, in order to establish a technical context for the Project EASI/ED system-wide user interface standards for interactive fax applications that are presented in Section 6.4.1 below. This discussion assumes a basic familiarity with the concepts and terminology associated with interactive fax applications.

The following topics are addressed in this subsection:

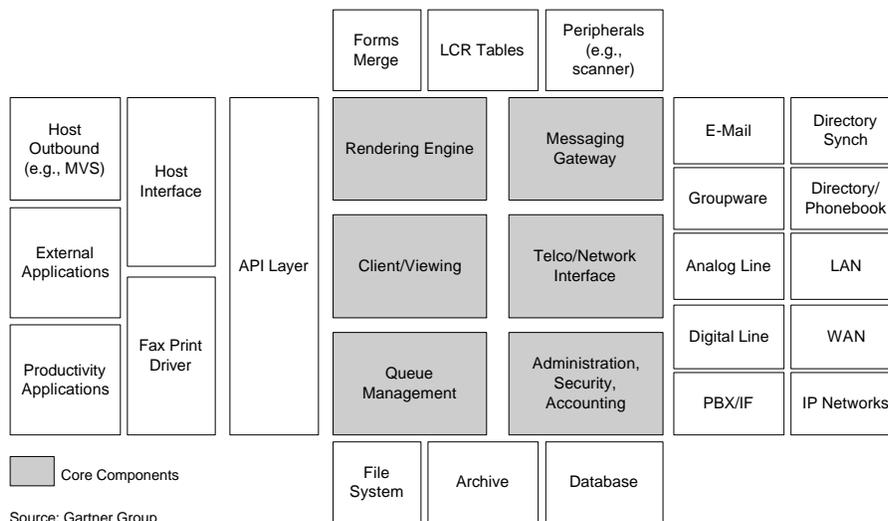
- **Application architecture.** The architecture that supports an interactive fax application will be explained.
- **Industry standards.** The industry standards relevant to interactive fax interface design will be presented.
- **Representative products.** Examples of applicable technology will be offered.
- **Evolving standards.** New directions for interactive fax technology will be discussed.

### Application Architecture

Interactive facsimile refers to the ability to have users send and receive faxes in order to update and query an application. The technology typically employs a cluster of networked fax machines that have an interface to a fax server. The main component of this architecture is the fax server, which distributes information to the fax machines to send a fax, and receives a fax via a designated fax machine. The fax server can be used to output data requested via an IVR system, incoming faxes or by on-line users.

Figure 6-5 illustrates this configuration. As shown in the diagram, the fax server includes the following core components:

- **Rendering Engine.** The document must be rasterized for faxing (e.g., converting a word document to International Telecommunication Union (ITU) group 3 format). The rendering engine takes application output and transforms it to fax-specific output, possibly performing file format conversions such as PostScript and HP Printer Control Language (PCL) conversions.
- **Messaging Gateway.** Fax servers often use messaging gateways to link fax capabilities to E-mail users. This may require that the E-mail gateway comply with standards such as simple mail transfer protocol (SMTP) to support integration with the fax system.
- **Client/Viewing.** Fax clients allow user access to the server and support viewing, annotation and basic manipulation (e.g., rotate, optical character recognition). Clients can be delivered in many permutations, including HTML-based, Java-based and tightly integrated with GroupWare (e.g., Lotus Notes). Simple integration with applications often occurs through Windows print drivers.
- **Network and Telephone Interfaces.** Depending upon the architecture, the fax server may run as a set of services in an NT, Unix, NetWare or other server environment.
- **Queue Management.** Sophisticated fax servers use queue management to balance loads and reduce bottlenecks. Some fax servers are better at managing the resource pool than others manage and can detect new servers and their resources.
- **Administration/Security/Accounting. Security.** These and similar, related management capabilities are critical. Often overlooked, the administrative ease of establishing users and maintaining logs, forms, routing tables and other support databases is a significant variable from vendor to vendor.



**Figure 6-5: The Gartner Group Fax Server Component Architecture**

## Industry Standards

Table 6-5 presents standards that apply to interactive facsimile applications.

Standard Title	Organization / Standard Name	Description
Standardization of Group 3 facsimile terminals for document transmission.	ITU T.4 1996	This specification provides transmission protocols for facsimile machines that use Group 3 compression.
Facsimile coding schemes and coding control functions for Group 4 facsimile apparatus.	ITU T.6: 1988	This specification provides transmission protocols for facsimile machines that use Group 4 compression.
Procedures for Document Facsimile Transmission.	ANSI/TIA/EIA 466-A:1966	Provides procedures used to ensure proper transmission/reception of data.

**Table 6-5: Facsimile Standards**

## Representative Products

Table 6-6 presents products that support interactive facsimile technology.

Product Name	Vendor	Product Type	Applicable Standard
Model 7033	Xerox	Facsimile	Group 3 Facsimile Equipment
Model 3000 Series	Xerox	Facsimile	Group 3 Facsimile Equipment
TF601	Toshiba	Facsimile	Group 3 Facsimile Equipment
TF610	Toshiba	Facsimile	Group 3 Facsimile Equipment
UF-332	Panasonic	Facsimile	Group 3 Facsimile Equipment
UF-342	Panasonic	Facsimile	Group 3 Facsimile Equipment
FAXCOM	Biscom	Fax Server	Windows NT, Unix compatible
Fax Sr.	Omtool	Fax Server	Windows NT, Unix compatible
FAXLink/FAXServer	Global Fax Network	Fax Server	Windows NT, Unix compatible

**Table 6-6: Interactive Facsimile Representative Products**

## Evolving Standards

The latest technology with regard to facsimiles is the Group 400 (Group 4) fax machine, which has a protocol for transmitting a fax over ISDN networks. The benefit to having one of these machines is that the Group 400 protocol supports images of 400 dots per inch (dpi) resolution, as compared to the Group 300 resolution of 203 by 98 or 196 dpi. With this type of resolution, comparable to that of a laser printer, faxed items are clear, delivered quicker and more refined in image. However, one can still use the Group 300 protocol when sending a fax to a Group 400 machine or receiving a fax from a Group 400 machine.

There are several new and innovative ways to request faxes that will increase user satisfaction and enhance the system's performance:

- **Broadcast Fax:** Using faxing software and a fax server, staff can send a fax to thousands of users in a few short steps.
- **Direct Fax from Desktop:** Staff can send a fax from their desktop to any user.
- **E-mail Fax:** Staff can have fax services tied into their E-mail system, thus making it possible to send and receive faxes utilizing E-mail.
- **Fax on Demand:** This option provides the user the ability to request a document via an IVR interface with the fax server.

### 6.2.2 Regulatory and Government User Interface Standards and Guidelines

There are regulatory and government compliance requirements that impact the design of user interfaces for application systems developed by the Federal government. In particular, the Americans with Disabilities Act (ADA) of 1990 defines guidelines and provisions for accommodating the disabled with respect to access to public areas and information. The ADA addresses the issue that handicapped or impaired persons have the right to equal access to all places and information that are opened to the public.

The ADA defines who is and who is not considered legally disabled. It explains that in order to be considered disabled, a person's impairment must be a long-term or a lifelong problem and must significantly limit one or more of life's primary activities. A vital mandate that the ADA embodies is that "places of public accommodations" be proactive to ensure sufficient communications with disabled parties. Furthermore, the term communication applies to the Internet as well as telecommunications.

The ADA has specific implications for Project EASI/ED. First, ED must provide special assistance for disabled persons to utilize the services the Department offers. Next, ED must itself abide by and require compliance with the Act by all employees when developing new systems for the general public to access. To readily comply with these guidelines, the EASI/ED user interface needs to support the special tools utilized by the disabled in order to operate different access mechanisms—Web, client/server, IVR, and interactive fax—without undue interference or complication.

In addition, user interface development should comply in all respects with ED's own regulations in this area. Specifically, ED's Internet Working Group has issued its World Wide Web Policy & Procedures (March 1998) that includes technical standards and a document submission guide. The standards presented in subsection 6.4.1.1 of this document comply with these guidelines.

## 6.3 Project EASI/ED User Interface Requirements and Design Goals

This subsection documents user interface requirements and design goals for Project EASI/ED. It is organized into the following subsections:

- Subsection 6.3.1, User Interface Requirements
- Subsection 6.3.2, Design Goals

### 6.3.1 User Interface Requirements

The application-specific user interface requirements for Project EASI/ED are driven by several key factors:

- The business requirements that the system(s) must support
- The types of users and the particular access requirements that each user category exhibits
- The access technology that is available to permit users to utilize the various application components of Project EASI/ED.

However, for the purposes of this document, user interface requirements have been analyzed at a level abstracted from application-specific implementations in order to define a set of requirements that are generally applicable. Rather than focus on the specific business requirements that must be supported, this analysis has instead focused on the characteristics associated with the most fundamental operations performed with a user interface, such as creating, reading, updating and deleting data.

Furthermore, the various types of users have been grouped into the following categories:

- **Customers.** This includes students, parents, and prospective students.
- **Partners.** This includes schools, lenders, guarantors, and servicers.
- **ED Personnel.** This includes all staff within the Department that will access EASI/ED applications.
- **General Public.** This includes casual browsers as well as people with professional interest (ranging from lawmakers to the press).

Within this high-level framework it is possible to derive a set of basic user interface requirements that apply across all access methods. These can be stated as four basic principles that drive how individual, system-specific detailed functional requirements will be determined. Although these principles are discussed primarily in terms of user interface construction for Web-based applications, they apply generally to all interface types (e.g., conventional client/server, interactive voice response, etc.). These principles are:

1. *The interface must accommodate (and should automatically configure itself to support) the different user categories defined above.*

Users should only be presented with the functionality that is appropriate to the user category to which they have been assigned. This presumes that user identification and authentication

will be the first function any specific application will perform. The interface should go further than this, however. Many commercial Websites such as **amazon.com** and **CDNow.com** personalize the user's experience, and this should be the approach adopted for the EASI/ED user interface, whatever access method is being used.

Personalization in this context means that, once a user has been authenticated, the user should be presented with all relevant information in a single, complete picture that is available from a comprehensive, integrated set of options that have been dynamically tailored to that user. The system should know who the person is once he or she has logged in, and provide the person with a seamlessly integrated, customized portal into the EASI/ED environment.

- 2. The interface must accommodate different levels of ability among users (and preferably provide the option to select between "modes").*

Within each user category, there will be individuals who are more experienced and who therefore may desire a more streamlined interface that provides less assistance in performing a given operation. An example of this is software that provides both wizards and a conventional, event-driven set of menus. A wizard is a series of dialog boxes that guides a novice user through a step-by-step process to perform a specific function, but it can be restrictive and even annoying to an experienced user. Conversely, the standard interface can be baffling to a novice.

One of the best implementations of this "dual mode" approach is Nico Mak's Winzip software, which in the last several versions has allowed users to specify during installation whether Winzip should present the wizard interface or the so-called "classic" interface by default. This selection can later be easily changed as users gain experience with the software.

This approach minimizes the likelihood that novice users will be intimidated or stymied by the interface (and therefore place demands upon the organization for technical support). At the same time, it reduces the frustration an experienced user would suffer while being forced to navigate through the system using the software equivalent of training wheels.

- 3. The interface must minimize impediments to access.*

The user interface must be designed to avoid limitations or prerequisites that (for whatever reason) act as impediments to access. Such limitations or prerequisites would in particular include features that cannot be utilized without first performing a potentially expensive and/or onerous task (such as upgrading hardware or software).

With conventional client/server software, the desktop client software (also known as frontware) typically has minimum hardware and operating system requirements that must be accommodated if the software is to perform correctly. This is feasible since most systems based on conventional client/server architecture are implemented in an environment where some control can be exercised over hardware and software configuration—such as a single branch, department, division, or even an entire organization.

However, with Web-based applications intended to be accessed over the Internet by the general public, such control over the hardware and software configuration belonging to each user is clearly impossible. Particularly with respect to Project EASI/ED, where the overarching goal is to facilitate increased customer focus and accessibility of ED services,

even attempting to impose a homogeneous operating environment on such a large and diverse user population would be fundamentally at odds with the objectives of the EASI program.

One can draw powerful lessons from the commercial world—high volume electronic commerce sites such as **amazon.com** or **CDnow.com** typically utilize only the most basic functionality available (e.g., tables and forms) for the client component, and all other functionality (including data validation) is located on the server side. In addition to the virtues of simplicity, fast loading time, reliability, and broad platform support, this approach also has the added value that it facilitates compliance with the requirements imposed by the ADA (see subsection 6.2.2 for further discussion of this topic). The W3C's Web Accessibility Initiative guidelines are very strict as to what features are acceptable with respect to accessibility, and from the WAI perspective, a clean, simple interface is the most accessible.

4. *The interface must be consistent in “look and feel” across applications, to the point the user does not recognize them as different applications, but rather as different aspects of a single application.*

A fundamental tenet of the most successful graphical user interface environments is that standard configurations for the user interface components reduce the time required for users to learn and become productive with different applications. Having the same main menu options located in the same place on the screen, the same behaviors associated with left-clicking and right-clicking the mouse, the same top-to-bottom, left-to-right organization of content on the screen, all contribute to ease of use. A user will know where things are the first time the application is loaded.

The value of this cannot be overstated. A fundamental drawback to the dissociated stovepipe systems that typically exist in most organizations today is that each system's user interface is highly idiosyncratic—even those that operate within a common interface environment such as Microsoft Windows. New rules and restrictions must be learned for each application system, and each system provides its own limited view into the total volume of data flowing through the organization. It is this limitation that has led both users and vendors to hail the Web browser as the “universal client” that will provide a single, all-encompassing window into the enterprise. However, for the Web interface to serve this purpose, a common set of design elements must be employed, or else the chaos of the legacy systems will merely be replicated in a new medium.

The effort to make the user interface recede into the background of the user's attention, so that the user can focus solely on the specific actions that he or she is performing to accomplish defined tasks, is referred to as “transparency engineering”. In a distributed environment such as is envisioned for EASI/ED, transparency engineering ensures that users are not aware they may be moving from one system to another and that they do not have to learn new ways to perform tasks in a different context. Instead, that complexity is concealed and the user simply clicks upon a menu option or hypertext link to be seamlessly connected to the data and functionality needed to perform a specific task—all within a single, familiar context.

These requirements (accommodating different user categories, accommodating levels of ability, minimizing barriers to use, and employing a consistent “look and feel”) drive the design goals for EASI/ED user interfaces presented below.

### 6.3.2 Design Goals

There is a growing body of work with respect to usability issues in user interface design, particularly with respect to the Web. A number of Websites have sprung up dedicated to the topics of good design and appropriate design techniques. Some of the best include:

- <http://www.w3.org/WAI> presents the W3C's Web Accessibility Initiative guidelines.
- <http://www.useit.com> is the Website of Jakob Nielsen Ph.D., a leading industry usability engineer.
- <http://Webreview.com/wr/pub/Usability> is Web Review's online Usability guide.
- <http://www.asktog.com> is the Website of Bruce Tognazzini, founder of the Apple Human Interface Group and former Human Interface Evangelist at Apple.

In addition, formal studies have been done to determine what constitutes "usability" and how it can be enhanced. A good example of this is the work done by Michael Levi and Frederick Conrad of the Bureau of Labor Statistics (BLS):

- "A Shaker Approach to Web Design" (<http://stats.bls.gov/orersrch/st/st970120.htm>)
- "A Heuristic Evaluation of a World Wide Web Prototype" (<http://stats.bls.gov/orersrch/st/st960160.htm>)
- "Usability Testing of World Wide Websites" (<http://stats.bls.gov/orersrch/st/st960150.htm>)
- "Evaluating Website Structure: A Set of Techniques" (<http://stats.bls.gov/orersrch/st/st970070.htm>)

There is a good deal of consensus as to the elements that distinguish a "usable" interface. The following design goals reflect this consensus and establish criteria by which the user interfaces developed for EASI/ED applications can be judged to have met the requirements defined in the preceding section.

#### 1. Consistency

Consistency in design and layout allows users to leverage their existing knowledge to new tasks, acquire new skills more quickly and focus on the tasks at hand, rather than on how to make the interface operate. The user interface should have a clear and consistent conceptual structure. A good user interface should organize and help structure the user's activities. This is achieved by presenting objects and actions appropriate to the tasks being performed and by specifying the flow of activities required for carrying out the task. Consistent use of standard operations and contents helps the user predict the results of an action before it is performed. It helps the user to form a consistent mental model of how the user interface operates. A consistent user interface reduces user errors and improves performance.

Similar components and concepts should be indicated using identical terminology, graphics and commands. The user interface should comply with uniform conventions for layout, formatting, typefaces, labeling, navigating etc. Standard functions should be reused across tasks and be depicted in the same way in each task.

## 2. *Aesthetic and Minimalist Design*

The user interface should be simple, in order to make it easy to learn and use. Simple, however, does not mean ugly or boring—simplicity is central to the concept of *elegance*. This type of simplicity can be achieved by reducing the amount of information presented to the minimum required to communicate adequately. An effective user interface has clarity of purpose—the user interface implicitly reveals to the user what can be done with it, thus eliminating uncertainty.

The degree to which a user interface is easy to learn depends at least partly on the complexity of the content the user interface must present. If the content is complex, then assistance should be provided to the user in the form of prompts, on-line help and instructions. Complexity can be reduced by grouping related tasks together and by reducing the number of different tasks presented at one time. Conceptual complexity can be decreased by giving a clearer picture to the user as to how the application is organized and how different components can be accessed. Providing only the important controls in buttons or control panels and embedding the rest in menus can reduce visual complexity. Using words common to the user’s vocabulary can reduce verbal complexity and misunderstandings.

## 3. *Clear and Immediate Feedback*

People like to perceive the results of their actions. With respect to computer systems, this is particularly important. We all have had episodes where we kept hitting a key or clicking the mouse until we received some feedback—and then the feedback we got was usually in the form of unintended consequences, such as undesired menu options being selected or unwanted text being entered into a document.

It is always a good practice to provide feedback for a user’s action. If noticeable, appropriate and immediate effect is provided in response to user’s action, this allows users to quickly grasp the logic of the user interface. This makes it more likely they will consider it easy to use, which goes a long way towards making their experience with the interface more enjoyable.

Depending on the user interface, visual and audio cues should be presented with every user interaction to establish that the user interface is responding to the user’s input and to communicate details that distinguish the nature of the action. Feedback is effective if it is timely and presented as close to the point of the user’s interaction as possible.

The three types of system feedback are:

- **Status Information.** Providing status information is a technique for informing the users as to what is going on the system. Displaying the appropriate titles on a screen or menu, or identifying the number of screens following the current one all provide required feedback to the user. Navigational status is particularly useful in the Web environment, since in a complex hierarchy of content one can easily lose track of the route one took and it is not always possible in a transaction-oriented site to simply reverse course using the “Back” button. It is especially important to provide status information during processing operations that take more than two seconds, to prevent the user from believing the system is simply inactive.
- **Prompting Cues.** Displaying prompting cues is another type of feedback. When prompting the user for information, it is a good practice to be specific in your request. A

typical example of such a prompt would be: “Enter the nine digits of your Social Security number.”

- **Error and Warning Messages.** Error and warning messages are a critical method of providing system feedback. Messages should be simple, specific and free of error codes and technical terminology. Error messages should appear in approximately the same format and placement each time so that they are easily recognized as error messages. Icons such as a stop sign and color-coding for severity (e.g., red and yellow) also increase the impact.

#### 4. *Prevention of Errors*

A key component of good interface design is to reduce the possibility of user error through predefining the options from which users may select. For example, a command line interface is the most difficult for new users to learn, because they must master both a large vocabulary of command keywords and a frequently cryptic command syntax.

Conversely, the use of pulldown menus, radio buttons, and check boxes reduces the likelihood of errors because users can only select from among correct options. Such unambiguous controls will increase both the speed and accuracy with which a given task can be performed, and thus users will perceive the interface to be easier to learn and use.

Confirmation of significant operations is also an important technique to prevent errors. By requiring users to confirm that they want to perform an update or deletion, this minimizes the likelihood that such an operation will occur inadvertently. Providing an Undo feature and graceful ways to exit an operation if the user changes his or her mind is also extremely important.

However, a caveat is appropriate. As stated in the preceding subsection, one of the user interface requirements for Project EASI/ED is that the interface must accommodate different levels of ability among users (and preferably provide the option to select between “modes”). Thus, in certain situations, it is advisable to offer the option to streamline a user’s interaction with a system so that an experienced user does not feel trapped in a Web of restraints obviously designed to protect a tentative novice from self-injury.

#### 5. *Subjective User Satisfaction*

Subjective user satisfaction is the response to good design. Good user interface design is consistent, unambiguous and self-explanatory, and (for Web applications and client/server systems) pleasing to the eye. This last element should not be underestimated: what is pleasing to the eye is not just a function of color choice or the appeal of certain graphics. The human mind responds to bilateral symmetry and balance, and at least in the U.S. and Europe, we are trained from an early age to read left to right, top to bottom. Thus, what is pleasing to the eye in Web page design includes a certain visual order that is readily comprehensible, and is easily recognized as analogous to the printed page.

Perhaps the most important component of subjective user satisfaction, however, is when the user feels in control of the interface and is able to produce the required results without difficulty or undue effort. As stated earlier, to the degree that the user interface effectively disappears so that the user can concentrate almost entirely on completing the task at hand (instead of on making the interface perform the functions the user needs it to), then the user interface design is successful.

## **6.4 Project EASI/ED User Interface Standards**

This subsection presents the Project EASI/ED user interface standards and design guidelines that address the user interface requirements and design goals defined in the previous subsection. This subsection is organized into the following subsections:

- Subsection 6.4.1, Project EASI/ED User Interface Component Standards
- Subsection 6.4.2, Project EASI/ED Web/Internet Application User Interface Model and Navigation Strategy

### **6.4.1 Project EASI/ED User Interface Component Standards**

This subsection defines the standards to be employed for the development of Web-based, client/server, interactive voice response, and interactive facsimile components of the Project EASI/ED user interface. This subsection is organized into the following subsections:

- Subsection 6.4.1.1, User Interface Standards for Web-based Applications
- Subsection 6.4.1.2, User Interface Standards for Client/Server Applications
- Subsection 6.4.1.3, User Interface Standards for Interactive Voice Response Applications
- Subsection 6.4.1.4, User Interface Standards for Interactive Facsimile Applications

#### **6.4.1.1 User Interface Standards for Web-based Applications**

This subsection specifies the user interface standards that are recommended with respect to Web-based application development under Project EASI/ED. The following topics will be addressed:

- Screens
- Windows
- Reports
- Development languages and tools
- Naming conventions
- System security
- System/user documentation and online help

The standards proposed in this subsection reflect the analysis of industry standards presented in subsection 6.2.1.1. These proposed standards also embody the design goals identified in Section 6.3.2, and comply with the technical standards included by ED's Internet Working Group in its World Wide Web Policy & Procedures (March 1998). The proposed standards have also been implemented in a fully interactive demonstration prototype hosted at <http://www.easi.ed.gov/prototype>, which is considered Appendix D of this document. A Project EASI/ED Web/Internet Applications User Interface Format and Style Guide that documents that the standards embodied in the prototype is included as Appendix H of this document.

As stated earlier, it is critically important to recognize that this document specifies standards for user interface design for Web-based application systems, as opposed to standards for layouts of simple "static" Websites. The difference is that with the former, a user is attempting to purposefully accomplish something, whereas with the latter the user is essentially a passive

observer (much as if the user were watching television). Thus, the primary goal of a user interface for a Web-based application is to help the user accomplish a task, not capture the user's attention with eye-catching design features. Furthermore, a greatly increased emphasis is placed upon scalability, reliability, maintainability, and accessibility.

Within that perspective, the foremost standard proposed in this document with respect to the development of Web-based user interfaces is to strictly adhere to the Distributed Presentation configuration described in subsection 6.2.1.1. To reiterate a point made earlier, it is important to understand the distinction between a conventional client/server application, where the operating environment of the client is known and even controlled, and the Web, where the operating environment is almost always not known in advance and cannot be controlled. Equally important to recognize is that by law (e.g., the ADA), government Websites are required to make reasonable accommodations for disabled users, which in this case is typically interpreted as referring to the visually impaired.

There are two responses to this situation: write browser-specific code and restrict user access to only the supported browser; or, write browser-independent code and support virtually anything. The latter approach typically implies placing all data validation and other types of business logic on the server, so that the presentation logic handled by the browser is only for display of information and capture of input.

Thus, one sees major commercial sites that process thousands of transactions per day (such as **amazon.com** and **CDNow.com**) adopting the latter approach, and that approach is also recommended also for Project EASI/ED. The intent is for the user interface for Web-based applications to be the “thinnest” of “thin clients” possible, which should minimize the possibility of browser incompatibilities interfering with the operation of the design. To the extent that some of the features identified above (e.g., CSS and DHTML) are employed, they must be incorporated in such a way that the features degrade gracefully when the HTML page(s) are viewed with browsers that do not support those features. They should *not* be employed in critical functional areas such as data validation or form operation, unless the Department of ED specifically authorizes an exemption that restricts users of a specific application to certain browser technology that supports the feature(s) in question—and there should be an overriding business reason driving that decision.

This approach places the least possible burden upon the user in order for the user to have access to Project EASI/ED Web-based applications. Such “frictionless” interaction with customers is critical to achieving efficient, customer-friendly systems that make the process of getting financial aid to students simpler, easier, faster—and cheaper.

With respect to the specific technical standards discussed in subsection 6.2.1.1, the following recommendations are made:

- **HTML.** Since it is uncertain whether even the version 5.x browsers from Microsoft and Netscape will support HTML 4.0 completely and accurately, and earlier versions clearly do not, it is recommended that HTML 3.2 be considered the operational standard for the foreseeable future.
- **CSS.** Because of the inconsistent support for CSS among vendors and between versions from any given vendor, the use of CSS for Web-based ED applications to be accessed by the general public is discouraged at this time. If CSS is

implemented, it should only be implemented alongside (and therefore, superfluously to) format and layout syntax allowed under HTML 3.2.

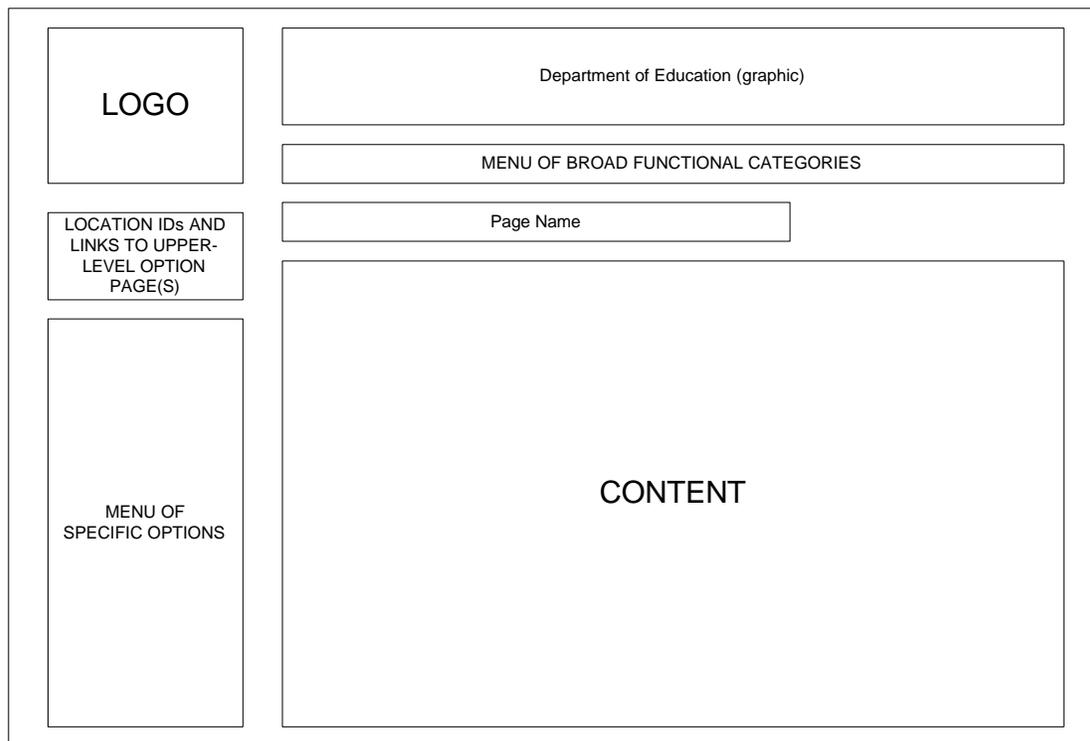
- **DHTML.** The use of DHTML should be implemented only to the degree that it can be implemented consistently across platforms and constructed so as to degrade gracefully on browsers that do not support it.
- **XML.** To the degree that support for XML is available in current browsers, it is nonetheless incompletely implemented and inconsistent between products from different vendors and versions from the same vendor. It therefore should not be employed until it is reliably supported.
- **Javascript.** Because only a subset of Javascript/JScript functionality is supported by both Netscape and Microsoft, developers need to be cautious and thoroughly test their scripts in both environments. Furthermore, some browsers (Microsoft IE 3.x and the text-only Lynx browsers in particular) do not support Javascript, which limits the uses to which Javascript can be applied. It should generally be limited to aesthetic effects such as highlighting of menu options.
- **Frames and framesets.** Given the navigational difficulties posed by the use of frames and framesets, not to mention the configuration management issues raised by having to maintain both a frames version and a non-frames version, the use of frames is strongly discouraged.
- **Multiple application windows.** Because it is possible to repeatedly open up new browser instances until the system runs out of memory and crashes, opening up new browser instances as popup windows should be done very cautiously if at all. The new browser instance should use a constant name, so that repeated calls to target documents will simply reload them in the named new browser instance, instead of opening multiple new instances.

The following paragraphs present more general standards for the development of user interfaces for Web-based applications.

## Screens

Within the context of Web-based applications, “screens” are defined here as the HTML pages that are displayed. Figure 6-6 presents the layout that is recommended for Web-based application development under Project EASI/ED. This design will ensure that Project EASI/ED Web-based applications will have a consistent appearance and structure that will minimize the time required for a user to learn how to navigate in an unfamiliar application.

This screen design implements the EASI/ED design goals through two key features. First, the design leverages the deeply ingrained tendency of most people in the Western world to read left-to-right and top-to-bottom. Second, the design is based upon the categorization of content into broad functional areas, each comprising topics that are then decomposed into sub-topics of increasing specificity. Specific explanatory content associated with each topic or subtopic should be displayed in the content area, so that no link is purely a “category” link (i.e., simply a link to another link). This hierarchical structure allows users to get a broad overview of the content and “drill down” to the desired level of detail. Because it balances breadth and depth, it represents the most efficient structure of content in order to minimize the effort required to navigate to desired information.



**Figure 6-6: EASI/ED Standard Screen Layout**

This is particularly the case because the interface design applies to application systems, not publication of static content. Recent research (Spool, 1998) has shown that sites that specialize in publishing content (especially those typically labeled “portal” sites) benefit in terms of usability from “flattening out” the content hierarchy, so that more links are “content” links as opposed to “category” links, especially when these “content” links are grouped by function into ordered lists. This is because the time required to find a specific piece of information increases as the number of links required to find the specific piece of information increases.

However, the same research has shown that while some people like sites that are easy to navigate, others like sites because they like the content, irrespective of whether the site is optimized for navigability. The conclusion one must draw, then, is that to please all of the people all of the time, one must balance ease of access with making the content as visually appealing as possible, and that is what this design is intended to accomplish.

With a user interface design that applies primarily to application systems, the assumption is that the structure of choices will be relatively narrow in breadth and relatively shallow in depth compared to a conventional, “informative” site. In striking a balance between breadth and depth, designers should conduct usability testing to confirm whether a given design optimizes that balance from the perspective of end users.

For applications that require the user to input data, the various objects of the data entry form should be restricted to the area designated for content, although it may expand to the left since the menu of specific options is generally not present on a page with a data entry form. It is important

to recognize that while it is possible to build a long form where the user must scroll down in order to see it all, this is not recommended for general users for two reasons. First, the user may not realize there is anything below the bottom margin of the browser window, and may think an operation has been completed when in fact it has not. Second, longer forms increase both load time and the time required to process the transaction (since more data is being posted). Thus, the “apparent speed” of the application may well be slower with a longer form, even though smaller form sections that require the loading of multiple HTML pages may take longer in terms of the total time required to complete an operation.

In designing forms to accommodate “power users”, it may be desirable to put the entire form on a single page where the user must scroll through it. If this is the case, the fact that the page is longer than a single screen should be indicated at the top of the page, and the buttons to submit the form should be at the bottom, so that it cannot be inadvertently submitted before it has been completed.

The implementation of data validation routines merits special concern. People accustomed to using—and building—conventional PC-based applications, particularly conventional client/server applications, have a deeply rooted expectation that validation of input will be immediate. This validation typically occurs at the data element level, and includes such techniques as input masks, range checks, and specific methods associated with the data entry field that perform more sophisticated checks.

However, as discussed above, Web-based applications operate under a different set of constraints than conventional applications. In particular, while designers of conventional applications design for an operating environment that is known in advance (and frequently subject to strict control), designers of Web-based applications do not know in advance what the operating environment will be, and are confronted with myriad operating environments—Web browsers—that vary significantly in their support for basic standards.

Data validation should therefore be performed on the application server, *not* on the client-side form. This is because methods of implementing client-side data validation (e.g., Javascript) are not uniformly supported by all browser types, and even where they are supported, they can still be circumvented either accidentally or deliberately. Placing the validation on the server reduces the burden on the user, by not requiring them to download and install a specific version of the browser to utilize the application. It reduces the burden on the developers, by not requiring them to repeatedly verify that the validation code works on browsers from different vendors, and on browsers from the same vendor running on different platforms. Only in cases where there is an overriding business requirement (such as FAFSA On The Web) should data validation be performed using a client-side implementation.

A critical concern in all applications is ensuring that data integrity is protected, and this concern is heightened with the attenuated, “stateless” connections between client and server that characterize Web-based application systems, where the browser does not maintain a single persistent connection with the server through the network. Placing all data validation on the application server ensures that transactions retain the so-called ACID properties (atomicity, consistency, isolation and durability) required to ensure data integrity. The application server is able to maintain “state” during transactions, thus establishing a session that appears continuous to the database server, in order to ensure data integrity.

## Windows

Within the context of Web-based applications, “windows” are defined here in terms of the attributes of the browser instance used to render HTML pages. These attributes are a function of two aspects of the user’s display: screen resolution and color depth.

It is necessary to lay out HTML pages that degrade gracefully as either screen resolution or color depth are reduced down to a specified minimum level. This minimum level of screen resolution for Web applications developed for Project EASI/ED is a resolution of 640 pixels wide by 480 pixels high, which is the VGA standard resolution. Even though most graphics cards and monitors now support greater resolution, many users leave their machines configured at VGA resolution because of small (less than 17”) screen size. It should be noted that the actual width of the browser’s display is approximately 600 pixels or less because of the browser’s own window borders. Width is the more important consideration, since at 640 x 480 resolution a browser with all tool bars visible presents a relatively small amount of vertical space for the display of content and scrolling may be unavoidable.

Color depth is defined in terms of the number of bits per pixel used to store color information. The industry standard color depth scale is as follows:

1 bit = 2 colors  
4 bits = 16 colors  
8 bits = 256 colors  
15 bits = 32,000 colors  
16 bits = 64,000 colors  
24 bits = 16,000,000,000 colors

The minimum standard color depth for Web applications developed for Project EASI/ED is 8 bits per pixel, which provides a 256-color palette. This is greater than the VGA default of 4-bit color, but realistically most monitors and graphics cards now support a minimum of 8-bit color and are configured that way at the factory. In reality, however, an evenly distributed palette that has 256 or fewer colors, and which is based on whole number values for the RGB triplets (the indices indicating the proportion of red, green and blue in a given hue), should contain 216 discrete colors. Thus, there are 40 palette slots not being used, which are at least partially reserved by the operating system (particularly the 20 system colors reserved by Windows). The use of the 216-color browser-safe palette will ensure that colors display accurately across platforms, and also that image quality will not be degraded by color dithering (dithering occurs when the operating system attempts to replicate a color outside its palette by interweaving two other colors).

The question of color depth applies mainly to graphics. Text should generally be in black, the background should be white, and the color of links should not be altered within the HTML code from the browser defaults. This facilitates legibility and gives the interface a crisp, clean look.

## Reports

Generating reports over the Web presents unique problems, because browsers are typically equipped to print only the HTML page that is currently displayed. There are three approaches to coping with this situation:

- **Write out the report as a file.** This file should be in a specified format (e.g., PDF, which has become a *de facto* industry standard) that can then be downloaded or emailed to the user.
- **Output the report as HTML to the browser.** It should be noted that sending a report (or a results set from a query) directly to the browser as HTML runs the risk of exceeding the browser's capacity to render the information. Thus, a result set from a query should be broken down into groups of between 5 to 20 records (depending in the amount of information per record) and sent to the browser as individual pages with one group per page. Similarly, if a report is sent to the browser as HTML, it should be sent as a series of separate pages instead of one voluminous document that must be rendered in its entirety.
- **Use a report viewer.** Because of the limitations of HTML, many report writer vendors provide a downloadable viewer (frequently a Java applet) for displaying reports online.

If more than one option is provided, the ability to select the output format should be provided to the end user prior to the report being generated.

When selecting a Web-based reporting tool, careful consideration must be given to the types of reports that must be generated. Usually a specialized report server is required. These are available from a number of companies and provide a number of options for viewing and printing reports. Some products are optimized for ad hoc reporting by end users, while others are designed primarily for production reporting where the user has little or no control over the output once parametric report criteria have been submitted to the report server.

### Development Languages and Tools

As stated above, the recommended development language of the presentation logic tier of Web-based applications under Project EASI/ED is HTML 3.2, with the option to use limited amounts of Javascript for essentially aesthetic purposes. At the present time, HTML 4.0 is not yet completely and accurately implemented in browsers, and given the necessity of maintaining downward compatibility, a "strict" HTML 4.0 Web page would be unacceptable. Refer to Section 6.2.1.1 for a discussion of the standards with respect to development "languages" for Web-based interfaces.

Development languages could also reasonably be construed as including Java and ActiveX components (which can be constructed in a variety of languages), but the downloading of applets or ActiveX controls represents a shift to the Remote Presentation architecture that contravenes the recommendation that Project EASI/ED adopt a strict Distributed Presentation architecture for Web-based applications. Therefore, these languages are not included in the recommended standards.

There are a plethora of development tools for constructing Web pages. There are fewer (but still a great and growing number) that are optimized for the development of Web-based applications. These typically are associated with specific application server technology (e.g., Microsoft FrontPage with the Microsoft BackOffice products, Allaire's Cold Fusion Studio with its Cold Fusion middleware), so the selection of particular application server technology will be an important driver in the type of development tool that is employed in the construction of the Web-based user interface.

For coding straight HTML, the tool to be selected can be largely left up to the developer's preference. However, it should be recognized that tools that emphasize WYSIWYG (what-you-see-is-what-you-get) design typically generate HTML code that is less efficient than HTML that is constructed with the more "coding oriented" tools such as Allaire's Homesite. The WYSIWYG tools may also use syntax that is version-specific or vendor-specific with respect to browser functionality.

This is particularly the case with word processors such as MS Word that convert a document to HTML, where such conversions typically resort to cumbersome and frequently misapplied syntax in an attempt (usually unsuccessful) to replicate the visual appearance of the original document. These tools should not be used.

### **Naming Conventions**

The primary aspect of importance with respect to naming conventions in the user interface design of Web-based applications is comprehensiveness. For example, it is important that all objects such as graphics and links have clear verbal descriptions provided through the ALT attribute of the tag defining the object, in order to facilitate the use of text-only browsers. Similarly, all elements referenced through Javascript should be identified uniquely and descriptively with the NAME attribute, in order to render the code readable and easily maintainable.

Furthermore, with respect to the naming of files such as HTML pages themselves, several rules should be followed. Because UNIX systems are case-sensitive and Windows 95/NT systems are not, all filenames should be lowercase. UNIX systems also do not permit spaces in long filenames, whereas Windows 95/NT systems do, so no spaces should be used in long filenames. The example below presents correct formatting:

<b>Large Company Logo.GIF</b>	becomes	<b>large_company_logo.gif</b>
	or	<b>large-company-logo.gif</b>
	or	<b>largecompanylogo.gif</b>

Because underscores between words in a long filename can be obscured by the underlining that is the standard indicator that a specific block text is a hyperlink, it is recommended that underscores not be used as separators between words in an HTML document filename that will be part of a URL. The adoption of a naming convention that, as described above, is a subset common to the major file systems will ensure portability of documents across platforms.

Another issue with respect to naming conventions is the use of relative path names when referencing files. When coding for the directory structure, only relative path names should be used. This preserves the portability of the HTML code across platforms. If path names are hardcoded and the site is rehosted, it is virtually certain that the HTML code will have to be adjusted, perhaps significantly.

Thus, an HTML page in the HTML directory should use syntax such as the following when referencing components located in a different directory:

```
IMG SRC="../../images/test.gif"
```

This ensures that the entire directory structure can be picked up and moved without any adjustments having to be made to file location specifiers in the HTML pages. Developers should

be aware of configuration parameters within the development tool that may automatically alter how such path specifiers are formulated as objects such as graphics are placed within the HTML page.

## **System Security**

System security is particularly an issue with application systems that process transactions (especially financial transactions), as opposed to systems that merely publish information. Security technology in this area is evolving at an understandably rapid rate, but it remains problematic.

The issue of system security with respect to Web-based application systems can be broken into three major areas: privacy; authentication; and integrity.

- **Privacy** in this context refers to ensuring that only authorized users have access to information about an individual.
- **Authentication** in this context refers to both the client and server proving to each other that they are who they say they are. In particular, with respect to a single transaction, authentication refers to proof of authorship.
- **Integrity** in this context refers to defending against tampering with data on the network, which includes encryption of data and ensuring that a given transaction is posted only once (a major concern in the “stateless” environment of the Web).

Although authentication raises issues with the user interface (specifically in terms of requirements relating to passwords or other authentication mechanisms), both “privacy” and “integrity” as defined above are almost completely functions of the application architecture on the host (server) side of the network. These issues are discussed in detail in Section 3. The word privacy is also often used in a broader sense to relate to policies on what information is collected electronically about individuals and how it is used. ED should develop a standard privacy policy statement applicable to all ED Websites, and a link to this policy statement should be placed on the initial (“splash”) page of every ED Website.

## **System/User Documentation and Online Help**

System documentation should include every document describing the implementation of the system from initiation to operation, with the specific documents depending upon the mandates of the software development methodology that has been employed. Section 2, Framework for Blueprint Delivery, discusses the lifecycle methodology that ED has adopted.

User Documentation presents a product description for the user, including the operation of the system after implementation.

User Documentation may include the following:

1. Functional description, which may include:
  1. A summary of the system.
  2. An outline of system requirements.

3. A short description of services provided by the system.
2. Introductory manual, which may include:
  1. An informal preface to system.
  2. A description of normal usage of the system.
  3. Suggestion on "How to get started".
  4. Explaining the use of common facilities/functions.
  5. Examples of how to use the system.
  6. Advice on how to recover from mistakes.
3. System reference manual, which may:
  1. Detail system commands and their usage.
  2. Furnish a listing of error messages and how to recover from detected errors.
4. System installation document, written for system administrators. This may include:
  1. How to install the system on a specific environment.
  2. A description of the media the system is supplied on, files of the system, minimal hardware configuration required, permanent files established after installation, and how to start the system.
5. System Administrator's manual. This may:
  1. Describe the messages generated when the system interacts with other systems and how the system administrator should react.
  2. Contain procedures to maintain the hardware.
  3. Explain how to clear faults on the system console.
  4. Give a general synopsis of the system.
6. Quick reference card, which may cover relevant short cuts.
7. On-line help systems.

Because Web-based applications are typically intended to support very large user populations, providing users with hardcopy documentation is typically not feasible. It is possible, however, to post documentation in electronic format so that users can download it at their convenience and print it out themselves. It is strongly recommended that complete user documentation for each specific Web-based application be posted, preferably in a choice of formats such as Microsoft Word or PDF, so that users can obtain comprehensive instructions for using the application.

Online help is a more complex issue, for two reasons. First, the developer has no control over the environment in which the application will execute (i.e., the Web browser), and that environment is highly variable, unlike the operating system environment in which a conventional client/server application executes. Second, the "stateless" nature of the Web constrains the methods that can be used to display online help as distinct from the application itself. A number of creative solutions are presented below, along with the respective shortcomings of each approach.

- **Balloon Help.** In a conventional client/server application, for example, it is possible to implement "tool tips", or "balloon help" as it is often termed, which displays a small descriptive label for a given object when the mouse is positioned over it for a certain length of time. Such a feature can be emulated in version 4.x browsers by using the ALT tag or through the use of DHTML. The former technique is a misuse of the ALT tag, which is intended to provide an "alternate text description" for images. Furthermore, the

technique fails to display anything in Netscape Navigator 3.x. The latter technique has the unfortunate shortcoming that it fails to hide the text of the “tool tip” at all in browsers that do not support Cascading Style Sheets.

- **Specialized Help Pages.** A simple technique to provide access to online help is to put in a link to a specialized help page. The drawback to this technique, however, is that it forces users to leave what they are doing and go to a new HTML page, and then return. Particularly in a Web-based application where transactions are being processed, it is not desirable to disrupt the transaction by jumping to and from a separate help page.
- **Popup Help Windows.** It is possible to implement popup windows that do not cause the entire HTML page to refresh through the use of additional browser instances (see Section 6.2.1.1). Through a relatively simple Javascript function, one can define a new browser instance stripped of toolbars and menus and launch it with an HTML page that is specified as a parameter when the function is called. The call to this function could then be linked to a button that is part of a menu or is located adjacent to a graphical control (such as a simple data entry field) that merits additional explanation. As discussed in Section 6.2.1.1, the name of the new browser instance (the second parameter) should be a constant, so that one does not exhaust available memory by instantiating numerous distinct instances of the browser. This method also relies on Javascript, which some browsers do not support.

The recommended approach to implementing online help in Web-based applications is to construct the online forms as if they were “wizards”, the step-by-step sequence of dialog boxes that Microsoft developed for making complex tasks easier for novice users. Wizards have become familiar to most people who work with computers, either because people have seen a wizard in an application like Microsoft Word, or because they have used an installation routine like those of InstallShield to install a piece of software. Using this approach, one can break the data entry form into pieces that fit on a single HTML page without requiring that most users scroll down to view the entire form, and include in each section of the form the appropriate text instructions to answer in advance any questions the user might have.

By anticipating the questions the user might have and providing the answers in advance, this approach minimizes the frustration the user encounters in using the system. When reading the instructions on each section of the form, users are utilizing the online help *without realizing they are doing so*. This comes the closest to creating a self-explanatory interface that recedes into the background of the users perception so that users focus primarily on accomplishing the task they wish to perform, not on how to use the tool to accomplish the task. The only caveat is that it may be desirable to create a less “friendly” version of the form that is optimized for data entry speed, and offer the option to users to access these forms by using the system in a “power user” mode.

#### 6.4.1.2 User Interface Standards for Client/Server Applications

The following subsection specifies the standards that are recommended with respect to client/server application development under Project EASI/ED. The following topics are addressed:

- Screens
- Windows
- Reports

- Development languages and tools
- Naming conventions
- System security
- System/user documentation and online help

The standards proposed in this subsection reflect the analysis of industry standards presented in Section 6.2.1.2. These proposed standards also embody the design goals identified in Section 6.3.2.

## **Screens**

With the context of client/server applications, "screens" are defined here as the layout of information that is presented on a display monitor. Depending on the task being performed by the application, the screen design requirements will vary.

In general, screen designs should adhere as closely as possible to those for a Web-based application user interface, described in the preceding subsection. There are three key caveats to this direction, however, that have to do with features that can be used in the client/server environment that either do not exist or are not recommended in the Web environment.

The first feature that is common in the client/server environment but not readily supported in the Web environment is a complex menuing system. Unlike the Web-environment, development tools in the client/server environment support the creation of menu bars, pulldown menus, and popup menus that are linked into a cohesive mechanism. The screen layout presented for Web-based applications should be adapted to accommodate the menuing system (i.e., the main menu bar across the top of the screen should have pulldown menus from each main menu pad, instead of menu options on the left side of the screen).

Another feature that is easily accomplished in the client/server environment but that is not recommended in the Web environment is immediate, client-side data validation. The capabilities here range from input format masks to range checks to complex methods encapsulated within each interface object that ensure only valid data is written from the client to the database. These capabilities should be used as extensively as possible, subject to the constraint that hard-coding data for lookup tables or building in business rules that are likely to change should be avoided.

The third feature is online help, which is discussed in detail below.

## **Windows**

In virtually all widely-used GUIs, a window provides the visual means by which the user can interact with an application program. The results of the command or data input by keyboard, mouse, or other devices are displayed in a window. It provides context for an application; all interface objects are organized and presented in windows. Windows provide a means of viewing and editing information as well as viewing the content and properties of objects.

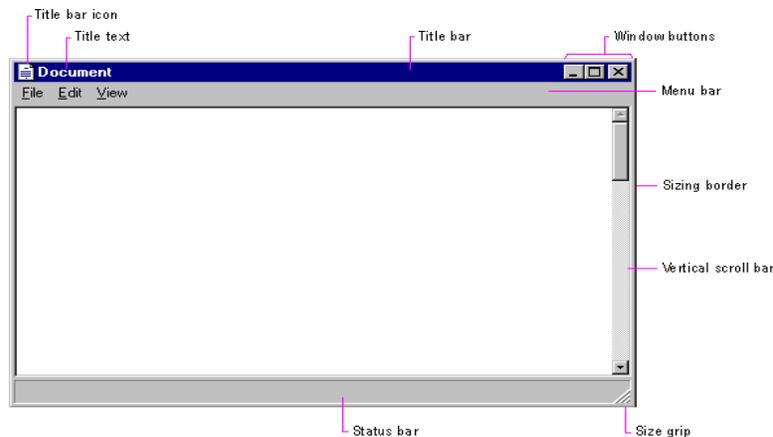
A window is typically rectangular and covers part or all of the display screen. In addition, multiple windows can be displayed at one time. Consistency in window design is particularly important because it enables users to easily transfer their learning skills and focus on their tasks rather than learn new conventions. Commercial GUI designs provide a number of basic functions

thus allowing the user to control window operations. Some of the features that most of the GUIs provide are:

- By opening a window, a task or application is started.
- By closing a window, a task or application is stopped.
- Scrolling allows the user to view the information within a window, even that which is outside the normal boundaries of the window.
- Windows can be stacked on top of each other.
- Windows can be simultaneously presented by either using the tiling or overlapping approach.

There are a number of different types of windows, distinguished by the functions they serve and specific behavioral characteristics:

A *primary window* provides the location of the main user interaction with the application. As shown in Figure 6-7, a typical primary window consists of a frame (or border) which defines its extent and a title bar which identifies what is being viewed in the window. The window can also include other components like menu bars, toolbars, and status bars. Primary windows are independent of each other.



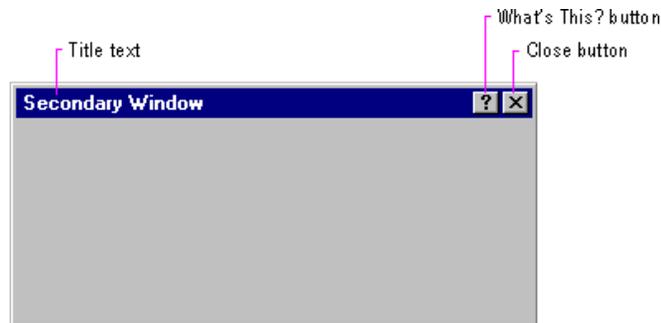
**Figure 6-7: Example of a Primary Window**

Microsoft has distinguished between Single Document Interface (SDI) applications, where there is a one-to-one relationship between the primary window and the content object, and Multiple Document Interface (MDI) applications, where the primary window represents an instance of the application, and visually contains a set of related document or child windows. Each child window is essentially a primary window, but is constrained to appear only within the parent window instead of on the desktop. The parent window also provides a visual and operational framework for its child windows. For example, child windows typically share the menu bar of the parent window and can also share other parts of the parent's interface, such as a toolbar or status bar. Microsoft Notepad, for example, is an SDI application, whereas Microsoft Word, Excel, PowerPoint and Access are MDI applications.

With respect to either SDI or MDI applications, however, there should be one and only one primary window for each instance of the application. With some applications it is possible to open multiple instances of the application simultaneously—in such a situation, each instance will have its own primary window.

Most primary windows require a set of secondary windows to support and supplement a user's activities in the primary windows. A typical secondary window, as shown below in Figure 6-8, includes a title bar and frame; a user can move the window by dragging its title bar. However, a secondary window does not include Maximize and Minimize buttons because these sizing operations typically do not apply to a secondary window. A Close button can be included to dismiss the window.

Secondary windows, if used, must be related to a primary window. Moreover, secondary windows can call other secondary windows and are either modal or modeless. Modal windows require a user to complete or abandon an operation before the user can perform a task in another window. Modal secondary windows restrict the user's choice, and thus should be used sparingly. System modal secondary windows should only be used for serious problems, for example, bad media errors.



**Figure 6-8: Example of a Secondary Window**

A modeless secondary window, however, allows the user to interact with either the secondary window or the primary window just as the user can switch between primary windows. It allows users to perform other operations without dismissing the secondary window. It is also well suited to situations where the user wants to repeat an action, for example, finding the occurrence of a word or formatting the properties of text. Modeless secondary windows should be used for operations that do not need to be completed before the user continues to another operation and dialogs that do not require immediate attention.

The standard color schemes for client/server user interfaces in the Project EASI/ED environment are:

- White background should be used for data entry fields and lists with editable data.
- The color for text should be black or a color with a very low luminance.
- A small group of accent colors should be defined for icons and graphics. The colors should be distinct but muted, with a similar amount of gray.
- The color for the window background should be a gray, which facilitates the sculpted “3D” look characteristic of graphical controls, especially in the Microsoft Windows environment.

## Reports

To be effective, a reporting capability must combine and transform raw data into meaningful information. The two decisions that have to be made while designing a report are, on what medium will the output be recorded, and in what format. A report could be in several different formats. The commonly used formats are tabular, zoned, graphic (bar charts, column charts, pie charts, line charts and scatter charts) and narrative. The output media that are typically used are paper, microfilm, microfiche and on-line display.

Recommended design guidelines for reports are listed below:

- The report layout should be simple to read and interpret. Simplicity leads to ease of use and improved productivity.
- Only data that is necessary should be presented on the report.
- Graphical output should be used wherever possible. Data should be organized so that the most important items appear at the beginning of the report and the least important appear at the end.
- Related data items should be grouped together.
- Every report should have a title, run date, report identifier and page number.
- Arrangement of the data should be visually appealing to the user. Data should be arranged from left to right, with the primary sort sequence in the left-most column. Enough space should be left between columns. Printing repetitive data should be avoided. Key data fields should be strategically positioned and highlighted.
- Section headings should be included to segment large amounts of information.
- There should be column headings for information represented in columns. Legends should be included where section headings and column headings have been abbreviated.
- Technical jargon and error messages should not be present on the reports.
- Logical groups of data elements should be separated from other groups of data elements whenever feasible via a blank row.
- Consistent terminology should be used at all times. Abbreviations and codes not known to the user should be avoided.
- The report should be generated while the information is pertinent to transactions or decisions.

Reports can be constructed quite easily using report writers, which are tools for creating customized reports. Report writers allow users to lay out a report format graphically, and can enable a user to develop sophisticated reports much more quickly than using a query language. There are numerous report writers available, and the selection of a specific product should address the following factors:

- Platforms supported by the product (e.g., Windows, Solaris, MVS)
- Ability to execute the reports on the client or on the server
- Flexibility of output options
- Ability of the product to specify table relationships (e.g., Join conditions)

- Requirement for a runtime component to execute reports
- Complexity of scripts or language syntax used to perform calculations and specify logical conditions

## Development Languages and Tools

Standards for development languages and tools are discussed in detail in Section 2, Framework for Blueprint Delivery.

## Naming Conventions

Naming conventions are an essential part of most application development efforts, especially in the case of multi-developer projects. While defining naming conventions, a balance needs to be reached between developing a structure that is meaningful, that imposes some controls on the developer, and that is easily understandable and yet adaptable enough to be expanded to accommodate new requirements as these are defined. It is important to be as consistent as possible to ensure that the meaning of an individual item name is not misinterpreted. However, it should be noted that tools that generate code do not generally require or even allow developers to specify naming conventions for variables, objects, or even modules.

A commonly used naming convention for variables is the *Hungarian Notation*. Its name is somewhat humorously derived from the ethnic origins of its inventor, Microsoft programmer Charles Simonyi. The consistent use of Hungarian notation improves the readability and maintainability of source code by encoding type, scope, and purpose into variable names.

Identifier names in the Hungarian convention consists of:

**<prefix> <base type><qualifier>**

- **Prefix** indicates the use of the variable. Examples are “a” (array), “c” (count), “d” (difference), “g” (global) and “i” (index).
- **Base Type** is the data type of the variable being named. Examples are “wn” (window), “scr” (screen) and “sz” (null-terminated string).
- **Qualifier** is the descriptive part of the name. Example are, page, stitch and current color.

Developers also have the option of choosing the naming convention standards proposed by the supplier of the development tool. For example, the PowerBuilder class type code is “w” for windows, “d” for data window and “m” for menu. The consistent use of a *documented* naming convention is more important than what that specific convention is.

## System Security

System security is discussed in Section 3, Security Requirements.

## **System/User Documentation and Online Help**

System documentation should include every document describing the implementation of the system from initiation to operation, with the specific documents depending upon the mandates of the software development methodology that has been employed. Section 2, Framework for Blueprint Delivery, discusses the lifecycle methodology that ED has adopted.

Online help in the client/server environment can easily be made far more robust than is reasonably achievable in the Web environment. In addition to a hypertext help file (typically an online version of the entire user manual) accessible from a menu option on the main menu, context-sensitive help can be linked to most objects, and the tool tips (balloon help) described earlier can be applied for most objects. This provides a multi-layered approach to user guidance. The Microsoft "Window Interface: Application Design Guide" manual provides a comprehensive description of the standards for online help that apply in the Windows environment, and it is recommended that these standards be followed to the degree that the development tools employed will support them.

### **6.4.1.3 User Interface Standards for Interactive Voice Response Applications**

The following subsections specify the menu and navigation guidelines that are recommended with respect to the user interface development for an IVR under Project EASI/ED. These guidelines reflect the recommendations made in Section 6.2.1.3, User Interface Standards for IVR. These standards also embody the design goals identified in Section 6.3.2, User Interface Design Goals.

#### **IVR Menu Operation Requirements**

The design of the logical branching within the menu structure of an IVR application will not appear significantly different than that of a client/server or Web-based application. The following are specific requirements with respect to how the menu structure should operate:

- Users should access the IVR system interactive menu by dialing a toll free telephone number.
- The IVR menu should provide three to five (3-5) options per branch menu.
- Each option should have a specific logic flow that will provide information to or obtain information from the user. There should be specific conventions that designate particular touch-tone buttons as critical to the logic flow's proper functionality.
- A consistent voice should communicate all options throughout the menu until a user chooses to exit the menu and terminate the call.
- Once a user has selected an option, thus proceeding through the logic flow, their information should be gathered and stored in a database.
- All menus, as well as all choices and options, should be brief in description.
- The menu should have a Customer Service Representative option at all menu branches in case the user decides to select that specific option.

## IVR Navigation Requirements

The Gartner Group has developed a comprehensive set of guidelines for the design of IVR applications, which are presented in Table 6-7. These represent industry best practices and are adopted as recommendations for IVR applications developed for Project EASI/ED.

Requirements	Example/Comments
Allow users to transfer from the IVR system to a CSR easily.	If the system is accessible 24 hours a day, then a CSR should be available for any questions.
Allow user access to appropriate options per menu branch only.	If the system offers three options and the user hits an inappropriate button, the system should ask for a valid option.
Keep user updated on the "hold time" estimate.	Users may be impatient, so inform them as to how long or short the wait may be to speak with a CSR.
Keep IVR scripts professional.	Humor is relative - what one person finds humors may be offensive to another.
Provide consistent options throughout the menu (common navigation techniques).	If the pound (#) key represents leaving the system in one part of the script, then it should execute that action no matter what menu branch the user is on.
Provide a voice that is indigenous to the user base.	Regional accents may be difficult to understand for some users.
Provide users with a toll free telephone number to access the IVR system.	If users must call long distance, then they might not utilize the system.
Keep the number of options between 3-5 per menu branch.	Users listen more attentively to menu branches that are concise.
Limit touch-tone input.	Too many characters can provide incorrect information and confuse the user.
Caller should be able to return to the previous menu.	There should be an option on every branch menu to return to the previous menu.
Allow the user to exit the system at any time.	The user has the right to terminate the call without having to backtrack out of the system.
Provide messages that detail the system's features and availability.	While a user is waiting for a CSR, have a message that tells the systems operational hours.
Use common phrases and terms that are understood by the general public.	Avoid regional clichés, acronyms, and phony terms.
Allow non-touch-tone phone or impaired callers to access the system.	Direct non-touch-tone phone or impaired users to a CSR immediately.
Keep menu descriptions clear and concise.	When offering the types of financial assistance, provide the type of loans available, not a description of the loan types.
Provide call transferring from the IVR to a CSR.	Queue callers properly when they transfer from the IVR to a CSR.
Request commonly known security verification information.	For example, do not request a user's grandmother's maiden name, this type of information is too difficult to obtain.
Provide a contingency plan if the IVR system shuts down.	Develop a plan in case of the IVR system crashes.
Explain the time frame on user requested information.	If a financial aid application requires 14 days to process, let the user know the precise amount of time any process may require.

**Table 6-7: IVR Navigation Requirements**

#### **6.4.1.4 User Interface Standards for Interactive Facsimile Applications**

The following subsection addresses standards for Interactive Facsimile applications under Project EASI/ED. These guidelines reflect the recommendations made in Section 6.2.1.4. These guidelines also embody the design goals identified in Section 6.3.2, User Interface Design Goals.

Interactive facsimile is unique among the types of user interface described in this document, in that it is essentially a batch input or batch output method as opposed to on-line interactive. Standards therefore apply to the formatting of the batch, and are as follows:

- Input forms should be designed to support OCR functionality.
- Wherever possible, each number or character of input should be indicated by marking a box on a grid that can be easily scanned without error. Standardized tests such as the Scholastic Aptitude Test (SAT) typically use this formatting.
- Data that is necessarily hand-written (as opposed to indicated by filling in a box) should be contained within a cell that delineates the proper location on the form for each character of the hand-written text.
- Output forms and reports should be designed to remain legible when printed at the minimum level of resolution of a fax machine (i.e., 200 by 96 dots per inch).
- Only Courier and Times Roman fonts should be used in input and output forms.
- Fonts should be at least 10 points in size (12 pitch for Courier).

## 6.4.2 Project EASI/ED Web/Internet Application User Interface Model and Navigation Strategy

This subsection will define a user interface model and navigation strategy for Web/Internet applications. The recommendations contained here have been implemented in an online, dynamic prototype. The specific characteristics of the prototype have been documented in the Project EASI/ED Web/Internet Applications User Interface Format and Style Guidelines, which are included as Appendix I.

### 6.4.2.1 User Interface Model

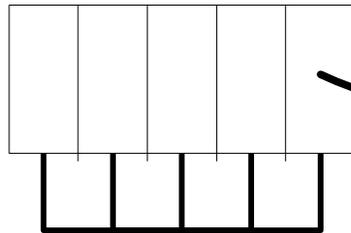
As discussed in subsection 6.2.1.1, the standards that support user interface capabilities in the Web environment are rapidly evolving. Unfortunately, support for those standards is inconsistent and incomplete in Web browser products from major vendors such as Netscape and Microsoft. This severely constrains the functionality that can be safely provided in a user interface if a design objective is to maximize the target audience and minimize the “barriers to entry” experienced by that audience.

In light of this, the user interface model devised for Project EASI/ED Web/Internet applications draws upon both the heritage of the Web itself as primarily a medium for providing access to *documents*, and of GUI design for client/server systems. At the same time it implicitly recognizes the constraints imposed by the current “aggressively heterogeneous” environment of the Web, where vendors are deliberately deviating from standards in an attempt to distinguish their products, captivate users, and gain competitive advantage. Figure 6-9 presents an illustration of the following concepts:

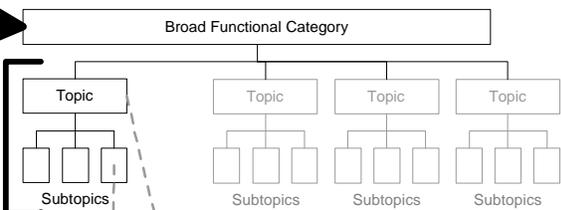
- Content is arranged left-to-right and top-to-bottom, beginning with the organization’s logo being placed in the upper left hand corner and the organization’s name across the very top of the page to the right of the logo.
- Content is divided into a set of broad functional categories, which are associated with menu options across the top of the page below the organization’s name. This arrangement of main menu options is consistent with standards for menu-driven interfaces in computer software, such as the Microsoft Windows Interface Guidelines for Software Design.
- Content within each broad functional category is divided into topics, subtopics, and (where occasionally necessary) sub-subtopics. Once one of the “main menu” options across the top of the page has been selected, the next page that is presented will have the topics associated with that main menu option displayed vertically on the left hand side of the page as a set of menu options. A brief description of the topics will occupy the “content” section of the page.
- Once one of the “topic menu” options on the left hand side of the page has been selected, the next page that is presented will have the topic name as a static (non-link) title on the left-hand side of the page in an enlarged font size (+1), and below that will be the associated subtopics displayed vertically on the left hand side of the page as a set of menu options.
- Should it be necessary to decompose subtopics into sub-subtopics, the same arrangement as for subtopics will be employed, with the following differences: the topic name

becomes an active link and the font will be larger (+2 or +3), and the subtopic name will be the static title below it.

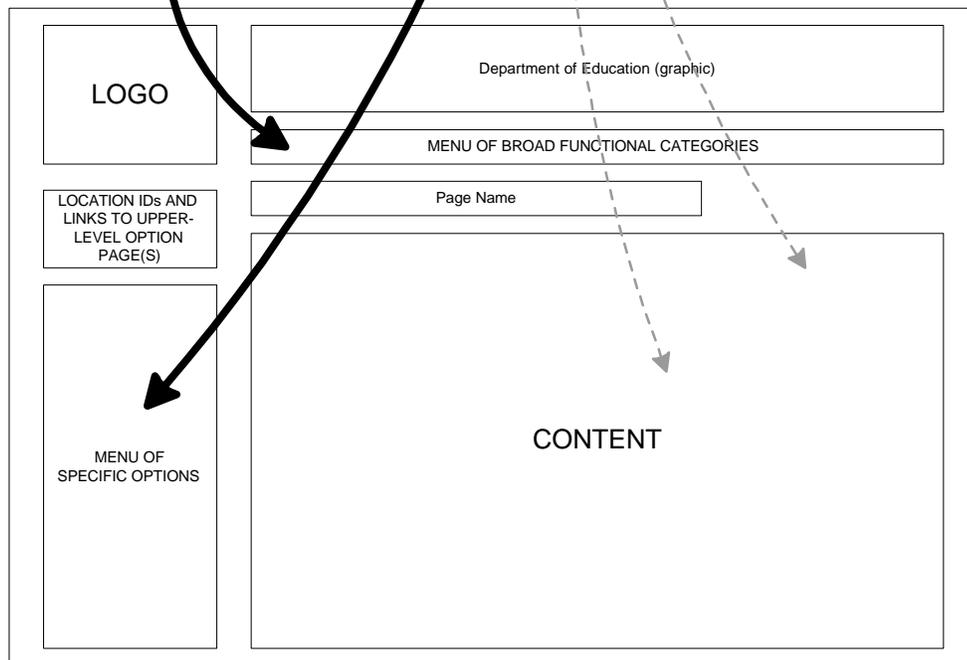
**Content is divided into broad functional categories**



**Content is then decomposed into topics and subtopics**



Topics and subtopics each have associated explanatory content that is displayed in the content area



**Figure 6-9: Project EASI/ED User Interface Model**

In certain instances—specifically when an individual page is a form, and particularly when it is part of a series of pages that implement a “wizard”-style multi-part form—it may be desirable to exclude menu options from a page in order to make the series of pages as “modal” as possible and strictly minimize a user’s ability to deviate from the prescribed pathway through the process.

Even then, it is not possible to disable the “Back” button on the browser unless the form is launched in a separate browser instance that does not show the browser menu or toolbar buttons by definition.

#### **6.4.2.2 Navigation Strategy**

The navigation strategy is derived directly from the user interface model defined in the preceding section. By organizing content into broad functional categories and then successively decomposing each category into topics, subtopics, and (where absolutely necessary) sub-subtopics, an elegant and easily navigable hierarchy of information can be constructed. The navigation strategy addresses navigation to the site from another site; navigation to other sites from within the site; navigation between pages within the site; and navigation within a single page.

#### **Initial Navigation from Outside Points**

Navigation begins with the URL, which is simply an alias for the rather cryptic Internet Protocol (IP) numeric address. The URL needs to be clear, concise and distinctive so that the user can remember it and easily type it, such as **www.ed.gov**. If possible, the use of subdomain names in the URL (prefixes to the primary domain name, such as **www.easi.ed.gov**) should be restricted to one level of subdomain.

#### **Navigation to Other Websites**

External hyperlinks need to be in a designated area or on a specified link bar as opposed to being intermingled with hyperlinks that lead to content within the site. Generally developers should refrain from overriding the browser’s own defaults for formatting of hyperlinks. In certain situations it may be advisable to provide a “text only” link. However, it is preferable to design the site to be usable by text-only browsers such as Lynx, to avoid the extra work of having to maintain two sites. Note that the ED World Wide Web Policy and Procedures require that a notice be given to users when they are exiting a “.gov” domain and linking to a “.com” domain.

#### **Navigation within the Same Website**

The user interface model defined in the previous section has the following specific features to support navigation between pages within the Website:

- “Home” button. The logo in the upper left hand corner of the page should be a link to the “main” page of the site (usually, but not always, “index.html”). This link should be enabled on every page but the main page itself.
- Menu bar. This provides links to all the broad functional categories defined for the content of the Website. It is always at the top of the page. In pages longer than a single screen, a “return to top” link should be provided, or text versions of the links anchored by the main menu should be provided at the bottom of the page.
- Topic menu. This section, arranged vertically along the left hand side of the page, presents the topics and subtopics associated with a main menu option that has been selected.
- Content area. This defined space where content is presented is bounded by the main menu on the top and the topic menu on the left.

## Secondary Menu

In addition to the main menu and topic menus, a secondary menu that provides access to supporting functionality can be provided. This is particularly appropriate for the first (i.e., the “home” or “splash”) page of the site. Links accessed from such a secondary menu could include:

- Site Map
- Search
- Index of topics
- Directory of relevant sites
- Help
- Frequently Asked Questions (FAQ)
- Contacts
- Disclaimers/Legal Notices

Where possible, the site map should be dynamic (i.e., each reference to an item should also be a link to the actual page displaying the item). This is also true of the Index, Contacts and Directory pages, so that each reference to an item or external site should actually be a link. In the case of the Contacts page, this would be a “mailto” link that allows the user to send the contact person an email message if the user’s browser is set up to support email.

## Color Coding

An additional feature that extends the user interface model and reinforces the navigation strategy is the use of color coding. Options on the main menu can be individually color coded, and the topics and subtopics under a specific main menu option would then be presented in the same color as the main menu option they are associated with. This immediately lets users know “where they are” within the site.

## Navigation within a Web Page

Every attempt should be made to break content into “chunks” that fit on a single screen so the user can view all content without scrolling down. If content must of necessity extend beyond a single screen, an opening paragraph should describe the content, and provide internal hyperlinks to sections within the page. As stated above, a “return to top” link should be provided at the end of every section accessed by an internal link and at the bottom of the page. Text versions of the links anchored by the main menu may also be provided at the bottom of the page.