

# Migrating from character-based interfaces to web-based interfaces

Frank Reiff  
Cooperative Systems Engineering Group  
Computing Department  
Lancaster University, LA1 4YR  
United Kingdom  
email: reiff@comp.lancs.ac.uk

## 1. Introduction

The emergence of the world wide web (WWW) is likely to have a lasting impact on organizational IT in the years to come. The popularity of the WWW is due in no small part to **the simple and adaptable user interface** it provides and its capability to **integrate** many disparate services within a **single unifying interface**.

Working the web is astonishingly easy and can be mastered comparatively quickly even by novice users. Moreover, the web offers **unrivaled distribution** by leveraging the internet infrastructure. For companies this means that core services can be made available **irrespective of geographical location** and that access to information can be maximized at a comparatively **low cost**.

The promise of the world-wide-web for companies is best embodied in the **intranet** concept. Based on standard internet technologies, intranets differ from the internet only in that they restrict access to authorized users and are thus not publicly available. In other words an intranet is internal world-wide-web service for an organization.

Intranets make it possible to offer an integrated information distribution, access and entry mechanism across the entire company. The use of standard networking protocols means that the company information can be accessed irrespective of geographical location. The ubiquity of the world-wide-web enables companies to quickly and easily produce **cross-platform** solutions.

**Deployment** and **maintenance** of systems can be greatly eased by the **centralization** inherent in the design of the world-wide-web. The emerging concept of "**network computers**" is related to this and aims to combine the virtues of centralized maintenance with the benefits of distributed execution.

For these reasons and for many others, companies are increasingly interested in making their character-based systems available via web-based user interfaces. This migration provides companies with a cost-efficient way of updating the user interface to their legacy systems. This allows them to gain access to many of the benefits offered by modern graphical user interfaces, both in terms of usability and in terms of training requirements. This type of software reengineering in which the interface is updated in

isolation from the underlying core system, is often referred to as **system revamp**. This revamp might also be undertaken as part of a larger scale reengineering project.

The **mapping** between the existing character-based interface and the web-interface to be developed will usually be a straight forward one. The most common way of data input used on the WWW is **forms-based** and in this respect closely matches that used by most character-based interfaces. A basic replication of the features offered by the character-based system in a web-based system is therefore usually possible at a relatively **low cost**.

Migrating character-based interfaces to the web as well as offering great rewards, also poses some significant challenges.

One of the greatest risks associated with using web-based technologies is their **frenetic pace of change**. Web-based technologies are still in their infancy and few standards have emerged. This lack of stability contrasts strongly with the stability and greater maturity of the technology in the data processing field. When implementing a web-based interface it may be necessary to modify the system at regular intervals to keep on top of the rapidly changing browser and protocol technology.

Another consequence of the immaturity of the technology is that there are a great number of competing technologies on offer and **no clear winners** have yet emerged. The choice of an appropriate technology is thus both particularly important and particularly problematic. Many of the technologies available today may have changed beyond recognition in five years time or they may have been discontinued all together.

Another point to keep in mind is that the web was **not designed as an application user interface technology**. It was in fact developed for information publishing and user interface features were only added to it after it was recognized as a powerful integrating metaphor. Many of the core assumptions that went into the design of the world-wide web still limit its suitability for building user interfaces.

Modern distributed client-server applications with graphical users interfaces still offer superior performance and user experience.

Another important question mark concerns **security**. The standard web architecture is based on non-secure internet protocols. The web as such was envisaged as a public access publishing service and not as a platform for organizational IT. Recent years have seen a flood of embarrassing press reports on security leaks for all major browser and internet technology vendors and these reports have done little for the general confidence in internet security. This flood of security scares was countered by the emergence of a myriad of mutually incompatible security protocols and this contributes to making security a problem for web developers.

Migrating applications from closed network architectures to the ubiquitous access offered by the internet, presents both benefits and drawbacks. **This trade-off between security and openness** is a feature of most modern technology and each company must make its own risk-benefit analysis.

Another problematic area of web-based user interface development is the **performance** aspect. Much work has gone into making client-server networking an efficient platform for enterprise computing. Developers who are accustomed to the benefits offered by this technology will have to learn to live without them again.

As a matter of fact web networking is highly inefficient. First and foremost HTTP (the transport protocol of the web) is **not a connection-oriented** protocol. It does not have a session concept. Each transaction between client and server thus requires a new connection to be established. To make matters worse there is no guarantee that a connection request will be honored. In practice, this means that a single web page may require dozens of connections to be established and that some of these may fail. This puts a great overhead on server, client and network. It also means that response times may be vastly inferior to those offered by modern client-server technology.

Moreover this lack of a session concept means that it can be quite difficult to **track users** through multiple interactions. The standard mechanism for handling active content, the Common Gateway Protocol, is also very wasteful of resources since it requires a new process to be created for each page request.

In this document we will review the **key technologies** for creating web-based interfaces and provide advice on how to implement **key functionality** and on how to address some of the problems that we have mentioned in this section. The document will be rounded off by **advice** on which technologies to choose for your specific project, based on your requirements, infrastructure and expertise.

The next section, however, is dedicated to the important decision of whether or not the web is the right platform for your project.

### *1.1. Are web-based interfaces for you?*

As we have discussed in the previous section, the main advantages of using web-based interfaces are:

- Powerful integrating user interface metaphor
- Platform independence
- Distribution
- Simplified administration
- Good match between character and web-based interfaces
- Cost efficiency

The most compelling reason for using web-based interfaces is that they provide a **powerful integrating metaphor** for all your corporate IT systems. Task-oriented IT, such as an order processing system, for instance, can be integrated seamlessly with other corporate information resources such as a telephone directory, etc. New network services such as workflow or discussion groups can be added equally seamlessly. The single, user-

friendly and easy-to-understand interface afforded by the web is also likely to reduce training overheads and can thus help to increase workplace flexibility.

Moreover, if your company is committed to making its IT services available **across platforms**, web-based interfaces are an attractive option. Web browsers are available for all major platforms and are all that is needed to run web-based applications. Furthermore web browsers are available for other computing appliances such as hand-held computers, that could not possibly be supported economically by other development strategies.

**Distribution** is another strength of the world-wide-web. If your company wishes to support on-the-road information access or is geographically distributed, the internet protocols that the web is based on provide excellent remote access. Using these protocols it is also possible to provide information access to **outside organizations** such as clients and suppliers, or even directly to customers. This also makes web-based interfaces especially attractive for point-of-sale applications such as booking systems. Web access could also be used to provide sales or support personnel with in-the-field information access.

The fact that web servers effectively act as centralized application servers can greatly facilitate tasks such as **system administration and maintenance**. System updates can be performed without creating the need to update client software, a fact which in itself represents a major advantage. The emerging **network computer technology** is certain to expand on these benefits.

Another core benefit of migrating from the existing character-based terminals to web-based interfaces, is that the **good match** that exists between both. Both technologies rely heavily on form-filling interfaces and centralized architectures and this should ease the transition both for end-users and for developers.

Developing basic web-based user interfaces can also be done at comparatively **low cost**. Some interfaces can be ported without requiring major rewrites. There is also no reason why web-based interfaces cannot be used to complement existing or future interfaces, thus providing many of their benefits while still allowing more powerful solutions to be deployed where appropriate.

All of these advantages will need to be weighed against these following serious drawbacks:

- Immaturity of technology and of standards
- Limited performance and scalability
- Limited User Interface
- Limited Evolution Potential

The greatest risk associated with creating web-based interfaces is the fact that the technology that they are based on is still **immature** and that the **standards situation** is still unresolved. Whichever technology is chosen, development will involve a fair amount of "hacking" to get around some built-in limitations. Applications written today will need

to be updated on a continuous basis both to take advantage of the evolution of the technology and to remain compatible with it. This frenetic rate of change makes it important to choose technologies that have proved themselves and are relatively mature while at the same time avoiding technologies that are likely to become obsolete quickly.

Another serious limitation concerns **performance** and **scalability**. The centralized server architecture of the internet protocols owes more to traditional mainframe computing than it does to modern multi-tier distributed computing. The connectionless web protocols make inefficient usage of network bandwidth and their remote procedure invocation mechanisms are wasteful and inelegant. Together these factors severely limit system scalability. This being said, there are many examples of web sites that successfully deal with very large numbers of simultaneous users. The Netscape and Microsoft web sites for instance, deal with several million user requests per day.

While the web offers a very attractive integrated user interface metaphor, it was not designed as an application front-end. In fact the web in its current form offers an **impoverished user experience** when compared with direct-manipulation graphical user interfaces. **Response times** too are insufficient to provide good direct feedback and transactions are not guaranteed to succeed, but may timeout. These limitations can, however, be overcome in the most part by using the latest technologies such as DHTML or Java.

Due to the immaturity of web-based user interfaces they are still ill-suited to the development of code that will be easy to maintain and adapt to the changing requirements of both their business context and the technology. The code implementing web-based interfaces will usually be distributed over a large amount of small components (scripts or applets), which presents special evolution problems. All in all, the **evolution potential** of web-based user interfaces is severely restricted by the immaturity of the technology.

The decision to implement a web-based interface will need to be based on a **trade-off** between the relative advantages that the web offers to your organization and the problems and limitations of the current technology. The development of a distributed client-server architecture with a graphical user interface should be seen as a real **alternative**. At the moment it is still clear that web-based interfaces cannot compete with GUI client-server solutions on either power or flexibility.

### *1.2. Overview of basic web technology*

The world wide web is defined as a series of standards. The **Hypertext Markup Language (HTML)** is the publishing language of the web and defines individual web pages. The **HyperText Transport Protocol (HTTP)** defines the communications between web client and web server. It is implemented on top of **TCP/IP**, the transport protocol of the internet. **Uniform Resource Locators (URL)** are used for uniquely identifying web pages and their components (e.g. image files) across the internet.

The client side of the world wide web is usually referred to as "**browser**", while the server side is referred to as a "**web server**". Taken together the above standards define a uniquely flexible infrastructure for transporting information across the internet. This infrastructure can be extended in various ways by integration with other standard as well as vendor-specific components.

### *1.3.Architectural Scenarios*

#### 1.3.1. Centralized Architecture

In this simple architectural scenario, all the application logic is centralised in the server component. The client (i.e. the web browser) is used only for presentation and simple data input. This task distribution between client and server is very similar to that used in traditional character-based systems.

#### **Advantages**

- Simple distribution
- Complete client-side platform independence
- Rapid development
- Low cost

Most of these advantages are familiar from other centralized architectures. The first advantage is that **distribution** is very simple. All a client machine requires to run the application is one of a number of compatible web browsers. No transfer of application code is necessary. This also means that the maintenance of the system is very simple. A change in the server software automatically propagates across the network, requiring no reconfiguration of the clients.

This architecture is by far the most commonly used for web-based application interfaces since it can be implemented quickly and at low cost. The good match between the centralized architectures used in most legacy systems and the centralized architecture of the web are the main reasons for this.

These advantage do, however, need to weighed against the following drawbacks:

#### **Disadvantages**

- Limited interaction bandwidth
- Limited performance and throughput
- Limited Scalability

The centralized architecture imposes some serious limitations.

First and foremost amongst those is an **impoverishment of the user interface**. Direct user feedback is not possible when using this architecture. Every change in the web page

must be initiated by a user interaction, such as pressing a button or clicking on a hyperlink. This limited interaction bandwidth has serious consequences for the design of the user interface. It is, for instance, not possible to validate input before sending it and this is a problem that we shall reexamine in more detail in a later section. Interaction is also limited by the fact that HTTP has no **session concept** and that each request requires a new connection to be established. The time required to establish such a connection usually represents a significant delay and cannot be predicted or bounded in advance. Connection attempts are also not guaranteed to succeed and the server may refuse a connection if it is too busy.

In summary, this means that web based user interfaces using this implementation strategy are usually slower, offer less interactivity and may result in more operator errors than standard GUI interfaces. From this it is easy to see that this architecture is **not well suited for highly interactive tasks**.

Another severe limitation is that of **scalability**. Web protocols are highly inefficient and require a great number of processes to be created, executed and disposed of. Furthermore, network utilization is poor. While **performance** is a real problem, there are ways in which this limitation can be overcome if necessary. Some sites use multiple, high-performance servers with very fast network access. Proprietary extensions exist that allow some of these performance problems (such as the creation of large number of processes) to be addressed. The technology section of this report will discuss some of these technologies.

### 1.3.2. Distributed Architecture

In this scenario application logic is executed on both the server and the client machines. The server side usually functions both as an information and as an application server. The client-side code is downloaded directly from the server and is executed locally.

As we mentioned in the previous scenario, an impoverished user experience can result from using a pure centralized architecture. Many of the benefits offered by standard GUI interfaces are, however, available to web-based interfaces. The usual approach is to extend the standard web pages with **client-side application logic**. The distribution of application logic between client and server can range across the entire spectrum from simple client-side presentation and input validation to the execution of complete applications on the client machine.

The introduction of client-side logic can also help decrease the processing load on the server and decrease network traffic. This may result in improved system performance and scalability.

A wide range of **technologies** are available to implement client-side application logic. Sun's Java and JavaScript languages can be used to write complete, platform independent applications or small self-contained application modules (applets) that can execute within a web page.

Microsoft's ActiveX technology allows Windows COM components to be downloaded and executed within web pages.

**Browser plug-in modules** can be used to extend the functionality of browsers in many different ways. The industry leading browsers, Netscape Navigator and Microsoft Explorer both use a plug-in architecture to allow the functionality of their browsers to be extended. Plug-ins are not automatically downloaded from the web but require an explicit installation step. Once installed, however, they do not need to be reloaded every time their services are required. Plug-ins are unrestricted client side applications which can access the browser's capabilities, such as output presentation and networking services.

The **advantages** and **disadvantages** of using these client side technologies to supplement the browser functionality depend very strongly on the particular technique that is used. In a later section we will evaluate the potential of each technology individually. The chief advantage of using any of these methods is that they allow the limited capabilities of HTML forms to be extended to compensate for their weaknesses and limitations, while preserving at least some of the advantages offered by web-based interfaces.

## 2. Implementation

In this section we will discuss the steps involved in implementing a web-based interface. We will begin by discussing the **presentation of the user interface** before looking at strategies for **validating user input**. We will then turn our attention to some other important implementation issues, such as system security, the tracing of users through a set of web pages, as well as some general considerations relating to the rapid rate of change and the standards situation in the internet market. We will also briefly address security concerns.

In this section we will limit our discussion to the implementation of **a typical web-based interface**. Some technologies, such as ActiveX and to a lesser extent Java effectively replace the standard components of a web interface. These technologies are therefore only partly within the scope of this document and will be mentioned only where appropriate.

**ActiveX** in particular does little more than host an ActiveX component in a web page and effectively replaces much of the web infrastructure with its own Window's COM based mechanisms. ActiveX may be the right choice for companies with a substantial investment in Microsoft technologies (Windows95/NT, Visual Basic, D/COM, SQL Server, IIS, etc. ), but it could be argued that adopting the technology will provide few of the benefits of web-based application interfaces. Perhaps if ActiveX is the right choice for your project, it would be better to opt for stand-alone Windows application development, rather than a migration to web-based technology.

Sun's **Java** technology offers power comparable to ActiveX, but is more tightly integrated with web technology. Java, JavaScript, D/HMTL and other web technologies are complementary and can be mixed easily. Java applets can implement their own networking and presentation, but they do not need to do this. What is more Java is

platform and browser independent and has wide support across the IT market. In this section we will mention Java several times, but will focus on typical web-based interface implementation.

## 2.1.Presentation

Character-based interfaces usually use a combination of form-filling and menu selection as their main user interface technique. A typical character-based interface looks something like this:

```
LANCASTER UNIVERSITY LIBRARY
ON-LINE CATALOGUE                                felix

Search Options
1 Title
2 Author
3 Author-Title
4 Keyword
5 Subject Index
6 Serials / Journals
7 Classmark (Shelving sequence)
8 ISBN
9 Short Loan

A Personal bibliography
B Borrower facilities

H Help
L Library Information
X Exit the catalogue

Key choice....

TITLE SEARCH (gives a separate sequence for each 'keyword')
Input Title (or Series begin with /): █
Format examples :-      EMPIRE STRIKES BACK
                        TAMING OF THE SHREW
                        /LANCASTER PAMPHLETS

Hint: Enter first word and half of the second word e.g. EMPIRE STR
      This usually gives the best results.
```

It is easy to implement this type of interface using standard HTML 4.0 page definitions. A trivial HTML rendition of the first page can be achieved using only simple components:

# Lancaster University Library On-Line Catalog

## Search Options

[Title](#)  
[Author](#)  
[Author-Title](#)  
[Keyword](#)  
[Subject Index](#)  
[Serials / Journals](#)  
[Classmark \(Shelving Sequence\)](#)  
[ISBN](#)

[Personal Bibliography](#)  
[Borrower facilities](#)

[Help](#)  
[Library Information](#)

The above page uses standard HTML tags. The title uses the Header 1 (<H1>) tag, the title below a Header 2 (<H2>) tag. The underlined words are hyperlinks leading to the appropriate pages.

The second page is equally simple to implement:

## Title Search

*(gives a separate sequence for each keyword)*

**Input Title (or Series begin with /):**

Format examples :- EMPIRE STRIKES BACK  
TAMING OF THE SHREW  
/LANCASTER PAMPHLETS

Hint: Enter the first word and a half of the second word e.g. EMPIRE STR

This usually gives best results.

The **input field** and **submit button** are implemented using standard HTML 4.0 form tags. The form definition is included in the <form> tag definition. When the **submit** button of a form is depressed, the browser sends a URL that is composed of a predefined **action url** and the concatenated contents of all the fields in the form to the web server. In response to the page request the web server will then return a dynamically generated response HTML page definition which will be presented to the user as a normal web page. This mechanism is described in more detail in the technology section dealing with the **Common Gateway Interface (CGI)**.

The migration to web-based user interfaces is, however, also an **opportunity** to revise and extend old user interfaces. The user interface of the *Lancaster University Library On-Line Catalogue* that we have seen above, for instance, was reengineered to provide improved user experience and usability. This was achieved without a substantial reengineering of the catalogue back-end.

This is the new, revamped interface:

## Lancaster University Library

### WebOpac (Online public access catalogue)



This 'live' system is continually being enhanced and developed. Recently introduced features include self-service reservations and self-service renewals.

A fully functional opac is also available using telnet, [felix.lancs.ac.uk](http://felix.lancs.ac.uk). Direct entries from your own Web pages are possible, [click here for details](#).

---

University Open Visit Day (Wed. 29th April 1998) [click here for details](#).

---

Please enter your search data, then click on a search type:

<input type="text"/>	Clear entry	<a href="#">or click here for help</a>	
Title	Author	Author-Title	Subject Index
Serials / Journals	Classmark	Short Loan	ISBN

---

### BORROWER FACILITIES

Enter your Library Number & Password, then click on one of the 'Buttons' below

Library Number:	<input type="text"/>	Password:	<input type="text"/>	Clear entries
Loans	Short Loans	Reservations made	Fines and Accounts	

---

[about this system](#)

---

[Lancaster University Home Page](#) | [Lancaster University Library Home Page](#)

As we can see much of the functionality of the system is now available from the front page. The elimination of the menu based interface is almost certain to result in better user performance. The migration has also brought some of the benefits of graphical users interfaces, such as point and click to the legacy system. Both user experience and usability are improved. What is more, the system is now available from everywhere in the world without requiring any further software installation.

Some legacy systems require **fixed maximum sizes for input fields** (e.g. the author field might be limited to 20 characters). This represents no problem for CGI forms as they allow you to specify the maximum size for a field. Other input controls such as multi-line text input, check boxes, image buttons, radio buttons, file controls, list boxes and a special password field are also available.

Until recently the implementation of **keyboard navigation** through the input fields was left to the particular browser that was used. With HTML 4.0, however, a standardised navigation system has been introduced. The author can now define a **tabbing order** for all controls and/or **access keys** for individual input fields. The mouse can obviously also be used to give the focus to a particular field. Together this provides a good and above all standardised navigation system.

Web page presentation could be further improved using more **recent presentation techniques**, such as **Cascading Style Sheets** (CSS1, which is part of the HTML 4.0 specification) or **Dynamic HTML**.

More complex interfaces can be implemented using a variety of techniques. **JavaScript** allows simple dialog boxes to be displayed and in conjunction with **DHTML** allows the creation of truly interactive graphical user interfaces including direct feedback and direct manipulation techniques.

Sun's **Java** cross-platform technology also allows complete GUI applications to be developed using a web browser as a host. ActiveX allows full featured Windows95/NT applications to be hosted within Microsoft's Internet Explorer browser.

For more information on these technologies please refer to the appropriate technology sections.

## ***2.2.Input Validation***

There are essentially three validation strategies for form input. The first one is to perform the input validation through the server, the second is to perform it locally on the client machine. The third strategy is a combination between local and remote input validation.

### **2.2.1. Server-Side Validation**

This is by far the **most popular** method of input validation for web-based user interfaces. After the form input data is sent to the server, the server validates the input and dynamically creates a page definition which indicates the success or failure of the transaction. The validation can be done easily since the full power of a programming language is available on the server and the server has full local access to all required information.

The main **advantage** of this method is that it can be implemented both quickly and easily. The main **drawbacks** are that it requires a (potentially slow) network connection to be established and that it provides poor user feedback. Web interfaces that rely entirely on server-side validation may not provide adequate interactivity for complex form filling applications, but may be adequate for simpler tasks.

### 2.2.2. Client-Side Validation

The advantage of client-side input validation is mostly **improved interactivity**. Direct feedback will almost certainly improve user performance, particularly on more complex input tasks. It can also improve ease-of-learning and usability.

Unfortunately client-side validation is more difficult to implement and not all required data may be available on the client machine.

As mentioned before HTML 4.0 offers basic client-side input validation, such as field size limitations. For more sophisticated validation (such as text fields that should contain only numbers or should follow a certain format), however, it might be necessary to use a client-side scripting language or a downloadable component to validate user input.

In any case current client-side validation services leave much to be desired and much of the validation will need to be made on the server-side. This problem can be alleviated by an intelligent **choice of input controls**. Multiple-choice fields such as "sex", for instance, are often implemented as single letter input fields (m for male, f for female) in character-based interfaces. The system must then validate the pressed letter to ascertain that it is either M or F. Using two radio buttons or a list box would make this validation superfluous in a web interface. Other input choices such as "date of birth" may be more difficult to implement, but could be validated by JavaScript or another scripting language.

If complex client-side input validation is essential for your project, you may want to consider using more powerful client-side functionality such as **Java** or **ActiveX** controls. The advent of **DHTML** is also likely to improve client-side validation tasks.

### 2.2.3. Distributed Validation

While both previous validation strategies have their respective strengths and weaknesses, for many applications, it will be necessary to combine both approaches. Client-side validation may provide better interactivity but many validation tasks will require access to data that is not available locally on the client-machine. An example of this would be the validation of a log-in user name/ password pair.

For most applications an intelligent combination between simple client-side input control design and server-side validation will provide the best cost-benefit ratio.

## 2.3. Security

The internet has been the playground of computer **hackers** long before it became a means for doing business. The **unrestricted interconnectivity** that is the main strength of the internet is also its Achilles heel in terms of security. Once your system is open to the internet, it becomes available throughout the world and to everyone who can gain access to it, be it rightfully or not. Furthermore security breaches can be difficult to detect and it is often near impossible to trace the perpetrator. The **international** nature of the internet also leads to special problems in bringing computer criminals to justice.

Security then is a major concern for internet-enabled applications. The world-wide-web technology that web-based applications build on was originally conceived as a public-access service. Access control was added to servers only after the intranet concept emerged. Security is still not a standard component of any official web standard and its implementation is largely left to the server software manufacturers.

Little independent information is available on the security aspects of different **server** solutions. The major players in the intranet server market, Netscape with its SuiteSpot servers and Microsoft with its Internet Information Server product, provide some limited security information on their respective web sites.

Another connected security concern relates to the **transmission of information** between browser and server. Various secure transaction standards exist and are well integrated with normal browser operations. The most popular and most widely supported standard for secure transmission is **SSL (Secure Socket Layer)** supported by both Netscape and Microsoft.

Other security concerns revolve around the use of **server-side scripts** through the common gateway interface or other mechanisms that allow the invocation of server software components. In theory, it might be possible for a person to gain access to a server machine by supplying unanticipated parameters to CGI scripts, but in practice this is unlikely. Care must, however, be taken to make sure that outside users cannot install their own scripts in the CGI directories via anonymous FTP (file transport protocol). This is an easy pitfall to avoid, but is quite widely ignored on the internet.

While server-side scripts, information transmission and server security are all important components of the overall system security equation, it is **downloadable components** that have provoked the greatest security concerns in the internet community. **ActiveX** and **Java** have both experienced high profile security scares since their introduction. The risk associated with downloadable application logic is that it might subvert system security.

These security scares have hit ActiveX hardest. ActiveX is basically a wrapping technology for Microsoft Windows COM components and as such they have full access to the client machine, making them particularly dangerous. Microsoft's security mechanism for ActiveX controls, **Authenticode**, is based on a trust model. Controls are signed and certified by a trusted source, such as for instance Microsoft itself. Users can then decide whether they trust that source sufficiently to allow the control to execute. IS managers and developers have proved equally suspicious of this model and few have adopted ActiveX controls.

While this trust model might well be inappropriate for internet use, it is much more acceptable within an intranet system. For a web-based user interface project the technology does not represent a significant risk since the ActiveX components will be developed and distributed only locally and therefore implicitly from a trusted source. The remaining danger is that enabling ActiveX in the client browser may cause insecure

controls to downloaded by users, thus opening the door for abuse of the local machine and through it potentially the whole network.

Sun's **Java** technology too has experienced some security scares, but has emerged substantially unharmed. The reason for this is that Java adopts a different security model. Instead of ActiveX's trust model it uses a **sandbox model**, which in essence limits the interactions between the Java applet and the client machine. This makes Sun's technology less powerful, but very much more secure.

Again though the security implications of using downloadable components written in Java are relatively low in the context of intranets, since again the source of the components (i.e. the company web server) is implicitly trusted.

In summary, the most important security risk associated with web technology is that the servers and the clients are accessible throughout the internet. Security solutions exist for each component of the overall security equation. Most of these solutions are, however, as yet unproven and the security of internet based systems is generally not as good as that for closed systems.

#### **2.4. Tracking User Sessions**

One of the most significant problems when implementing interfaces with web technology is how to deal with the **page-oriented nature** of the web. The web protocols have no session concept, meaning that each page "stands" on its own and that there is no explicit connection between different pages.

For the development of web based interfaces this means that a web page containing an application interface is treated in the same manner as any other web page. The problem that this causes becomes apparent when you think about how **transactions that span multiple pages** can be implemented.

Consider the following example:

*The web-based interface to a client database provides the functionality necessary to add a new customer record to the database. The web interface to this functionality is implemented over a number of web pages, each of which implements a separate step in the account creation process. Depending on the user input at each step different paths through the procedure will be taken.*

Implementing this procedure is not as straight-forward as it appears. The problem is that since there is no session concept, data that was entered in previous steps is no longer available for processing in the current page. What is required, is a way to make data persistent across different pages.

There are a number of ways in which this can be implemented, all of which require the **dynamic creation** of web pages.

The first and simplest way of achieving this data persistence is by including the data of each previous step in the page definition for the next page. In other words when the data

for one step is entered and sent to the server, the server then dynamically creates a new page and embeds all previous information in the definition for the new page.

One simple way to do this is to include the data in **invisible form fields**. These fields are identical to normal form fields except that they are not rendered by the browser and cannot be changed by the user. Using these fields in conjunction with **cgi-forms** is simple. At each step all the parameters are embedded in the new form definition and each time the user presses the submit button, all of these parameters, both old and new, are sent to a cgi-script. One limitation of this mechanism is, however, that cgi-forms need to be used for each step, from the first to the last, otherwise the data will be lost. Another potential problem is that no data can be saved from one transaction or session to the next.

Another similar method is to use the **URL** to make previously entered data persistent. Instead of embedding the data in a form definition you can use the URL for this purpose. Two typical examples of URLs that could be used for this purpose would be:

<http://www.theSite.com/AddClientAccount/reiff/>

<http://www.theSite.com/cgi-bin/AddClientAccount.cgi?user:reiff>

In these examples the user name "reiff" is embedded in the URL. One advantage of this method is that it allows normal hyperlinks to be used for navigating the forms. Again the server will need to dynamically create pages that include previous information.

Another more sophisticated method for making data persistent across multiple pages is the use of **cookies**. Cookies are a mechanism that allows server-side connections (ie. CGI scripts) to store and retrieve information on the client-side of the connection. Cookies are persistent across sessions and are stored on the client-side. When a browser connects to a server for which it has already stored a cookie it sends this cookie along with the page request. This mechanism was originally introduced to allow browsers to keep preferences for different web sites.

A more in-depth discussion of cookies can be found in the appropriate technology section.

## ***2.5. Standardization and Rate of Change***

The internet in general and the world-wide-web in particular, are widely perceived as one of the largest emerging markets in recent history. The corporate battle for market share is therefore predictably fierce. The three major players in the internet market, Netscape, Microsoft and Sun seem determined to defend or even extend their market share at any cost.

**Microsoft's Internet Explorer** and **Netscape's Communicator/ Navigator** products are at the heart of the **browser wars** and the development of these products is driven more by competitive than by technical imperatives. Both products differ in a number of ways and implement standards in different manners. The resulting **incompatibilities** make it very hard for web authors and developers to develop content that works equally well

with both products. These problems are aggravated by the fact that both browsers now have roughly equal market share.

**Netscape's Communicator** is largely built on open standards and offers complete cross-platform compatibility between Windows, Macintosh and Unix environments. Netscape is also working closely with Sun to promote Java technology.

**Microsoft's Internet Explorer** is clearly intended to cement Microsoft's domination of the operating system market and is very closely integrated with the Windows platform. Microsoft's push to integrate its browser product into its operation system has caused considerable controversy and is at the heart of US anti-trust actions against the company.

Microsoft does offer Unix and Macintosh versions of its browser, but they are not as complete or as well integrated as their Windows counterpart. On these platforms new versions are also usually delayed by several months.

While Microsoft has introduced many **proprietary technologies** into its browser product (e.g. ActiveX and Visual Basic Scripting Edition), none of these technologies have yet experienced any significant success. Microsoft has developed its own versions of other proprietary and open standards and integrated them within its own browser. Internet Explorer now features Microsoft's own version of Java, JavaScript (under the name JScript), Secure Socket Protocol, Dynamic HTML, etc.

Another web technology that causes much controversy is **Java**. Netscape and Sun are working together to push Java as a cross-platform solution. Microsoft's own implementation of Java, however, is not 100% compatible with Sun's reference implementation. The Java standard is also currently the subject of a legal battle between Microsoft and Sun.

Moreover, the **slow standard processes** have proved incapable of accommodating the rapid rate of change of the internet market. Rather than waiting for full standards to emerge Netscape and Microsoft both rush to implement draft standards and add their own proprietary extensions (ie. DHTML). The resulting incompatibilities make it very hard to produce cross-browser functionality using advanced technology.

The browser wars as well as the immaturity of the technology have made the web market highly unstable. One consequence of this is that nobody should expect to write web applications that will not require frequent updates over the years as the technology matures. It would also seem advisable to favor standards-based solutions over proprietary solutions and to avoid emerging technologies such as XML or DHTML entirely for now.

The best advice seems to concentrate as much as possible on lowest-common-denominator technology, such as HTML 4.0 and the common gateway interface. Other technologies should only be used where necessary and proprietary extensions to standards should be avoided altogether.

### 3. Technology

#### 3.1. HTML

The standard page description language of the web is **HTML**, the HyperText Markup Language. HTML is a simplified and static version of **SGML**, the Standard Generalized Markup Language defined by ISO 8879. The idea behind markup is to add structural information to documents so that the structure is made explicit and can be processed by computers. SGML is actually a meta-language, that is a language that allows another language to be formally defined.

Unfortunately SGML is difficult to use and extremely complex. This is the reason why SGML is not used to describe web page content and structure and **HTML** is used instead. HTML defines a series of markup tags that define the structure, content and presentation of web pages. Originally, however, HTML was not developed to provide presentation information. It was meant only to make the structure of the page explicit so that it could be searched electronically. So called **logical tags** such as <H1> (Header 1) and <CITE> (citation) were used to externalize document structure, but the on-screen presentation was left entirely to the browser.

Unfortunately, however, with the commercialization of the world-wide-web, the need for more sophisticated on-screen presentation became urgently important and browser vendors added their own presentation specific tags to HTML. Web authors began to use elements such as tables and headers to create more visually sophisticated pages, but in the process began to subvert the separation between structure and presentation. Web pages became once again difficult to search electronically.

HTML is the corner stone of all current web technology. It is defined by the W3C consortium and is now a full standard. The latest version of this standard at press time is HTML 4.0. There are small differences in the implementations of HTML in different browsers, but the technology is generally compatible across browser and platform boundaries.

As mentioned before, however, HTML was not originally conceived to support all of the tasks that it has been put to and has evolved dramatically as a standard. The original HTML was conceived mostly as a simple means of publishing academic papers in a machine interpretable form. Since then it has gained more advanced page layout capabilities, form-based input capabilities, etc.

The many shortcomings of HTML are especially significant as it is such an ubiquitous technology. HTML 4.0 has introduced the concept of **Cascading Style Sheets** as a means of separating presentation and content of web pages. The static nature of HTML has proved to be a limiting factor for creating more complex interactive web-based interfaces, which has led to the creation of **Dynamic HTML (DHTML)**. The static nature of HTML also limits the amount of actual machine interpretable information that it can provide. The **eXtensible Markup Language** or **XML** represents an initiative to replace HTML with a more expressive, albeit complex, SGML dialect.

All of these technologies will be discussed in more detail in their own sections.

### **3.2. Cascading Style Sheets (CSS)**

The first problem with HTML that we have mentioned is the breakdown between the separation of content and of presentation that has resulted from the undisciplined addition of presentation tags into the page definition. HTML 4.0 has addressed this problem by introducing a separate presentation description language, the **Cascading Style Sheets (Level 1)** or **CSS1**.

**CSS** allows different presentation **styles** to be defined for use within a document. These styles are conceptually very similar to those employed in word processing packages. A style can define typographical, location and other presentation attributes of text, tables, etc.. Once defined a style can be applied throughout a document, providing the additional advantage that it makes it easy to redefine styles throughout a document. Style sheets can be included within the HTML page definition directly or they can be placed in a separate file which is linked to the HTML page definition. This affords the possibility of using the same style sheet for different web pages or of applying multiple style sheets to the same page.

The definition mechanism for style sheets is also very powerful. Styles can **inherit** properties from each other and so-called **contextual selectors** can be used to define presentation of elements that have multiple styles applied to them concurrently. Style sheets can also be **cascaded** meaning that more than one style sheet can be applied to a web page concurrently.

In essence style sheets allow the separation between content and presentation to be reestablished making web pages easier to maintain. They also allow for greater control over the presentation of entire documents as well as of individual elements.

### **3.3.DHTML**

**Dynamic HTML** is an extension to the **HTML 4.0** standard and is supported by the W3C standards committee, as well as by Netscape and Microsoft. DHTML is intended to add more **dynamic behavior** that can be executed on the client machine to web pages.

One of the major problems with HTML is that it is static and that as a result web pages cannot be changed dynamically without loading a new page from the network. DHTML adds the facilities necessary to dynamically change web pages.

In particular, DHTML has three core benefits: Pages can include dynamic styles, content and positioning. Four elements need to be combined to create dynamic pages with DHTML: HTML 4.0, Cascading Style Sheets (CSS), the Document Object Model (DOM) and scripts.

The **Document Object Model (DOM)** allows client-side code to dynamically access and manipulate the structure of web pages. Initially DOM will support bindings for **ECMAScript** and **Java**. ECMAScript is the European Computer Manufacturers

Association's Standard Scripting Language for the WWW. It is modeled on JavaScript 1.1, but expands it in a number of ways. At present only Microsoft's **JScript** conforms to the ECMAScript standard, but **Netscape JavaScript** is expected to follow soon.

DHTML offers great **advantages** for web-based user interface development. First and foremost, DHTML is **standards-based**, meaning that it will be platform and browser independent and will hopefully continue to evolve in a controlled manner. For the developer it offers the possibility of building **rich interfaces** with direct manipulation and feedback. It should also result in a significant reduction of network traffic. **Basic input validation** and interactive **information presentation** can also be implemented using this technology.

DHTML as a technology is also **language independent**, which means that it can be used in conjunction with other technologies.

The main **drawback** of using DHTML is that Microsoft and Netscape have implemented it within their browsers in different manners. DHTML in its present form is not 100% portable between Netscape Communicator and Microsoft Explorer. DHTML is currently in the process of becoming a full standard and shortly afterwards should become compatible across browsers. At press time, the main drawback of DHTML is its immaturity both as a standard and as a technology.

+	Platform independent
+	Language independent
+	Will be an open standard
+	Considerable industry endorsement
-	Immature
-	Incomplete and unreliable implementations

### 3.4.XML

As mentioned before, the standard page description language of the web is **HTML**, the HyperText Markup Language. HTML is a simplified and static version of **SGML** the Standard Generalized Markup Language defined by ISO 8879. Unfortunately SGML is difficult to use and extremely complex, which is the reason why it was not adopted for web page definition. One key limitation of HTML, however, is that it is impossible to add new structural tags to documents on a need-to basis. Adding such structural tags would allow authors to make their web pages more easily searchable. If an industry could agree on a set of tags then all of its online information would become much easier to search.

While SGML is too complex for use as a page description language for the web, it is argued that HTML is too limited. The W3C is thus in the process of adding a new page description language to the WWW, the **eXtensible Markup Language** or **XML**. XML

like HTML is based on SGML, but can be easily expanded with new structural tags. The idea is to provide 80% of the functionality and flexibility of SGML at 20% of its cost to the web author.

XML has the **backing** of some major players such as Microsoft and Netscape and it is an initiative of the W3C. It offers obvious and significant advantages. Nevertheless it is far from certain that web authors grown accustomed to HTML will be willing to migrate to XML. A key problem of XML is that for it to be useful, online communities will need to come to an agreement on which tags are to be used. Recent experience has shown how difficult this is to achieve in practice.

The greatest problem for XML is however that it is still **work in progress**. The standard has not yet been defined, implementations in web browsers are not going to be available for some time yet, and there is still no authored content.

+	80% of the power of SGML at 20% of the cost
+	Allows the structure of web documents to be made explicit
+	Will be an open standard
+	Considerable industry endorsement
-	Not yet available at press time

### **3.5.CGI - Common Gateway Interface**

This is the de-facto standard for server-side logic invocation. It is supported by virtually all web servers on all platforms. It is language and platform independent and can be invoked by all current browsers.

CGI transactions typically have the following structure: The browser sends a page request with a URL that points to the location of a cgi program (i.e. <http://www.domain.com/cgi-bin/program.cgi>). Arguments to the cgi program are also passed in the URL (e.g. <http://www.domain.com/cgi-bin/program.cgi?param1>).

When the server receives this URL it invokes the cgi-program and passes the parameters via the CGI interface. The CGI program responds by returning an HTML page definition on the standard output. This page definition is then sent back to the client as a response.

There are several ways in which browsers can create page requests that will invoke server-side programs. The simplest of these is to embed the calling URL in a static **hyperlink**. The most frequently used mechanism, however, is to dynamically create the page request URL via client-side **CGI forms**.

The user fills out the fields of the form and then presses the submit button. In response the client generates a URL containing the field information as well as the location of the CGI program. The invoked program can access the data via the CGI interface and responds by dynamically creating an HTML page definition that is returned to the client.

Another method of invocation that is not supported by all web servers is **SSI** (Server Side Includes). SSI can be embedded directly into the HTML page definition. SSI enabled servers will parse the page definition before sending it on to the client. If the server finds a SSI in the HTML page definition it will execute the specified program and replace the SSI definition with the output produced by this program.

CGI is both **platform and language independent**. All programming languages that are available on the server platform can be used to create dynamic content by accessing the data through the CGI interface. **Scripting languages** are very popular for creating CGI-programs which are also known as CGI-scripts. The most popular scripting languages are PERL, TCL, Python, JavaScript (LiveWire JavaScript) and Unix shell scripts. CGI can also be used by all other popular programming languages including C/C++, Java, Visual Basic, Fortran, etc.

The CGI mechanism allows server side logic to be written quickly and easily and is by far the most **popular** mechanism for real-time creation of web content. Its platform and language independence, ubiquity and its standards-based approach make the best mechanism for implementing server side logic. What is more, a considerable body of free cgi-scripts dealing with many routine tasks are available on the internet.

CGI does, however, also have some serious **disadvantages** and **limitations**. The most serious disadvantage concerns the way in which it executes the server side programs. CGI launches a new process for each web request that it receives. For busy servers this can mean that the same program runs in thousands of separate processes and takes up considerable processor time. Process creation, scheduling and termination are all "heavy-weight" activities that are highly inefficient. In practice this limits the scalability and performance of the mechanism.

While the latest CGI standard (Version 1.1) does not support long-lived processes there are extensions to CGI that can help overcome this problem. Various vendor-specific alternatives to CGI also exist.

+	Simple, quick, flexible
+	Standard
+	Widely supported and popular
+	Quick development
-	Performance, Scalability

### **3.6. Java**

Few programming languages have ever received as much media exposure as Java. Java was developed by Sun Microsystems and is a programming language based on C++. The language itself represents a "bare-bones" version of C++ extended with some new features such as garbage collected memory management as well as internet extensions. The major

characteristic that distinguishes Java from other programming languages is its **object-code level platform independence**.

Java programs are compiled to **byte-codes** that can be executed by a **Java Virtual Machine (VM)**. The Java VM can be implemented on different platforms and applications in Java byte code can be run on any platform for which such a VM is available. This combines some of the benefits of a compiled language such as C/C++ with the advantages offered by interpreted languages such as Tcl/Tk or PERL.

What makes Java particularly interesting for web-based development is the fact that all major web **browsers** (i.e. Netscape Navigator & Microsoft Internet Explorer) incorporate a Java VM. Java programs can be executed within web pages. These small applications that are embedded within an HTML web page are also known as "**applets**".

For developers this offers the opportunity to implement client-side application logic. Java can be used to implement everything from animated icons to full-featured applications.

The **distribution** model for Java applets is simple. The applets reside in byte code form on the server and are downloaded as part of the page definition. Once downloaded applets execute on the virtual machine provided by the web browser. Within the browser they can interact with the user and communicate with the browser. Applets have access to the HTML page definition and when using DHTML (see relevant section) can manipulate it dynamically.

In the context of a **migration to web-based interfaces**, applets allow sophisticated client-side data presentation, user input and input validation to be performed. Java applets can interact directly with the browser and even provide their own networking services.

A new breed of computers called "**Network Computers**", which are supported by a number of vendors including Sun and Oracle is starting to emerge. These network computers are usually envisioned as running Java programs and using only standard internet protocols such as HTTP. Local storage devices are either missing altogether or are used only for data caching. All information, including the Java applications reside on the server machine (or network of machines) and are downloaded on a need-to basis.

The main advantage of these new "**network appliances**" is that they combine the ease of administration afforded by a centralized architecture with the performance advantages of client-side execution. While many major players have pledged their support for this concept, at press time there is still little hardware and/or software support available.

Java allows sophisticated applications to be developed, while remaining platform independent and leveraging the web as a transport and presentation mechanism.

The major **downside** of using Java is that it still is a relatively **immature** technology. At press time, the Java Development Kit (JDK) is at version 1.2, but new versions are released every couple of weeks. Java is also still a **proprietary standard** owned by Sun,

even though attempts to define it as an open standard are progressing. The Java APIs are also still in flux and change frequently. Compared with other proprietary standards, however, Java commands the broadest vendor and developer support and is almost certain to play a major role in organizational IT in years to come.

Other problems revolve around **VM implementations**. There are many different implementations for different platforms, including performance enhancing Just-In-Time Compilers (JITs). Many of these implementations are, however, still unreliable and contain incompatibilities. Even at the best of times, however, Java is relatively slow when compared to native code.

+	Platform independent
+	Powerful client side application logic
+	Well supported
-	Immature
-	Unresolved standards situation
-	Unreliable Virtual Machine Implementations

### 3.7. JavaScript/ JScript/ ECMAScript

The development of **JavaScript** started at Netscape before the introduction of their Java-enabled Navigator browser. Originally called **LiveScript** it was envisaged as a client-side scripting language for Netscape's browser product. When Netscape and Sun formed an alliance intend on exploiting Sun's Java technology in Netscape's browser product, the name was changed from LiveScript to JavaScript and the language was adapted to the new situation.

Today JavaScript is supported by all major **browsers**. JavaScript is an interpreted scripting language that can be embedded in an HTML document. While it lacks the power of Java, it is very much easier to use. JavaScript in fact bridges the gap between the static HTML and the full power of the Java environment. In this sense it follows the age old strategy of providing 80% of the benefits at 20% of the cost.

JavaScript is especially useful for implementing simple tasks such as **basic input validation**.

On its own JavaScript can be a useful, if limited tool. In conjunction with **Dynamic HTML**, however, JavaScript has the potential of providing dynamic web page content at a very low cost. Netscape also offers a JavaScript dialect for server-side scripting called **LiveWire JavaScript**.

The standards and terminology situation of JavaScript can be somewhat confusing. In fact JavaScript is not a standard, but a technology developed by Netscape in conjunction with

Sun. Based on JavaScript 1.1, however, the European Computer Manufacturers Association has created a standard version of the scripting language that has been significantly enhanced. This standard is known as **ECMAScript** and is used in the definition of the W3C's **Dynamic HTML standard**. Microsoft have implemented ECMAScript in their browser and renamed it **Microsoft JScript**. The net result of this is that most, but not all scripts written in JavaScript will run on both Navigator and Internet Explorer.

+	Platform independent
+	80% of the benefits of Java at 20% of the cost
+	ECMAScript is an open standard
+	Well supported
-	Immature
-	Incompatibilities between implementations

### *3.8. Microsoft ActiveX/ DCOM/ COM+/ COM*

**ActiveX** was seen by many as Microsoft's response to Netscape's Java initiative. ActiveX is the brand name for a visual control framework based on Microsoft's Component Technology, COM. COM exists in several different flavors including COM+ and Distributed COM (DCOM).

There is a fair amount of confusion surrounding ActiveX, not least due to the intensive marketing strategy pursued by Microsoft when the technology was first introduced. Microsoft has now clarified the situation and confirmed that ActiveX is merely a packaging technology for COM.

ActiveX controls are descendents of **Visual Basic eXtensions** and provide similar functionality in a language independent manner. Microsoft's browser product, Internet Explorer supports downloadable ActiveX components, which are executed on the client machine.

This means that you can use the **full power of Windows applications** from within a web browser interface. It also limits the use of ActiveX controls to the **wintel platform**. Effectively their use is also limited to **Microsoft's IE browser**, since Netscape does not support the technology directly.

ActiveX components can be dropped into web pages in much the same manner as VBX extensions can be dropped into Visual Basic forms and may be attractive to Visual Basic Developers. While the lack of platform and browser independence is a major drawback for this technology, it might be acceptable to companies that already have a substantial commitment to Microsoft's component technology and to Windows95 and Windows NT as a platform.

ActiveX development while sharing many of the advantages of COM **development** also shares many of its disadvantages. COM is not an easy framework to work with and requires considerable local expertise. ActiveX is also commonly criticized for being "slow" and "too complex". A number of **security** problems have also tainted the technology.

Using ActiveX controls for simple tasks such as **form input and input validation** is clearly overkill, while using fully featured ActiveX packaged applications through IE does not seem to offer any clear advantage over using it as a standalone application. Perhaps then it makes more sense to invest the development effort required to create a web-based interface with ActiveX, into creating a robust stand-alone client-server application with a "proper" graphical user interface.

It is clear that ActiveX technology has **not been widely adopted** by web authors and web application developers. Faced with rumors of the demise of ActiveX as a web development tool, Microsoft insists that ActiveX "is not dead and will not go away"...

+	Provides complete COM functionality on Windows machines with Internet Explorer
-	Not platform independent
-	Not browser independent
-	Proprietary standard
-	Costly development

### 3.9.ASP - Active Server Pages

The de-facto standard for server-side application logic through the WWW is the Common Gateway Interface (CGI) that we have discussed previously. One considerable problem for CGI is that it is highly inefficient and thus does not scale well. In fact for each request made through CGI a new process must be forked. Large numbers of requests mean that large numbers of processes must be created, executed and terminated, resulting in very large overheads.

Some vendors specific alternatives to this model exist. Microsoft's solution to this problem is called Active Server Pages (ASP). ASP is proprietary and exists only within Microsoft's Internet Information Server (IIS) web server product.

To MS IIS customers it offers a "**compile-free**" API that allows the combination of normal web pages with scripts and ActiveX Server components. Compile-free in this context means that scripts that need to be recompiled are automatically compiled when they are requested.

ASP offers native scripting support for Visual Basic Scripting Edition (formerly known as VBScript) and Microsoft JScript. It also supports ActiveX scripting which allows it to be tied in with any compatible scripting engine though this is more complicated. Through

the use of ActiveX Active Server Components complex and powerful applications can be created.

The main advantage of ASP is that it allows server-side code to be run "**in-process**" taking advantage of multithreading and offering efficient and scalable performance. Its main disadvantage is obviously that it is tied to Microsoft's IIS server and the Windows platform. Other server vendors offer similar functionality, but as yet there is no independent standard that allows in-process execution.

+	Integrated with Windows platform
+	In-Process execution
-	Limited To Microsoft Information Server
-	Limited to Windows NT server platform
-	Proprietary protocol

### 3.10. Visual Basic Scripting Edition

At the beginning of Microsoft's marketing push to secure internet market share, it released its own scripting language to compete with Netscape and Sun's **JavaScript**. This scripting language was initially called **VBScript** and was based on Visual Basic and Visual Basic for Applications (VBA).

In the meantime, Microsoft has implemented JScript, its own version of JavaScript based on the European Computer Manufacturer's Association's **ECMAScript** and has deployed it in its Internet Explorer browser. Microsoft has ported VBScript to a number of platforms including Solaris, but this cannot conceal the fact that it is essentially a Windows standard.

VBScript is also used in **Microsoft's Internet Information Server (IIS)** product as a server-side scripting language. Its functionality is not substantially different from that offered by JavaScript/ JScript/ ECMAScript.

In summary, VBScript duplicates the functionality of the standards-based ECMAScript and can be recommended only to developers with substantial investment in Visual Basic technology who wish to combine it with ActiveX technology.

The main disadvantages of using VBScript, rather than JScript, are that VBScript is available only in Internet Explorer, is effectively limited to the Windows platform and is in direct competition with Microsoft's own JScript.

+	Well suited for developers with Visual Basic or VBA experience
+	Works well with ActiveX technology
-	Limited to Internet Explorer browser
-	Effectively limited to the Windows platform

-	Proprietary standard
-	In-house competition with JScript

### 3.11. Cookies

As mentioned in the implementation section dealing with the tracking of users, the stateless nature of the HTTP web protocol can lead to problems for the implementation of web-based interfaces.

Cookies are a standardized technology designed to overcome the problems of statelessness.

Cookies are a mechanism that allows server-side connections (ie. CGI scripts) to store and retrieve information on the client-side of the connection. Cookies are **persistent** across sessions and are usually stored on the client-side. When a browser connects to a server for which it has already stored a cookie it sends this cookie along with the page request. Cookies are essentially persistent variables that can be transmitted between client and server. As such they are a good way to **add state** to the state-less web protocols.

Cookies are usually implemented on the **client-side** using **JavaScript**, but other languages such as **Java** or **VBScript** can also be used. Cookies can also be used in some web servers to maintain state information. **Netscape's SuiteSpot** servers for instance allow server-side cookies to be used in conjunction with **LiveWire**, its server-side scripting language based on JavaScript.

The main **reservations** about the use of cookies spring from concerns about **privacy** and **security** and recent surveys suggest that the majority of internet users now disallow their use in their personal browser settings. The main way in which cookies can be abused is to **"track" users** across the internet and to create databases of their browsing behavior. Many people regard this as an infringement to their right for privacy.

Within the context of internal IT projects, this is a less relevant as the IT department could simply require their users to activate cookie support. Nevertheless due to their recent bad press, the future of cookies is less than clear.

+	Allow client and server side state to be maintained across pages and sessions
-	Privacy and security concerns

## 4. Guidelines

This section aims to provide some general advice on how to make your technology and implementation decisions.

There are some **general guidelines** that should be followed when migrating to web-based interfaces. Those guidelines, however, need to be evaluated against a number of requirements that are specific to your project.

Each **project** has different aims and **objectives**. These objectives will guide your technology and implementation decisions. The most **common objectives** for a migration to web-based interfaces are as follows:

- Leverage the web as a powerful integrating user interface metaphor
- Provide platform independence
- Achieve location independent access
- Simplify the administration of client applications, software and hardware
- Revamp the user interface at low-cost

These objectives need to be weighed against the following major drawbacks:

- Immaturity of technology and of standards creates an unstable development platform
- Performance and scalability of solutions is limited by technology
- User interface design is subject to technology limitations
- Evolution potential of solutions is limited and unclear

In the following sections we will first recommend some **general guidelines** that should be used when building applications for the web. We will then review a number of **common project requirements** and briefly discuss their impact on technology and implementation decisions.

#### **4.1. General Guidelines**

Web technology is still in its infancy and is changing rapidly. This **lack of maturity** has implications for all web-based implementation projects.

A good general guideline for web-based development then is to avoid the most **recent** and thus most **immature technologies**, since they do not provide a stable development platform for your project. For the same reasons, you should also avoid **proprietary technologies** wherever possible and opt instead for **standards-based technologies**.

In practice, this means that you should **default to the simplest and most common technology choices** such as server-side application logic through the **Common Gateway Interface (CGI)** and **HTML 4.0** as a page description language. This combination is by far the most common and the most mature. It is also sufficient for most projects.

Some projects, however, will find this combination to be **too limiting**. In this case preference should be given to the more mature and standardized technologies.

#### **Summary:**

- Avoid immature technology
- Avoid proprietary technology

- Avoid unnecessary complexity

## **4.2.Project Requirements**

Every project is different and technology and implementation choices depend on the **requirements** of your project. Many projects will, however, share a number of **common requirements** and it is these, most common requirements, that we will discuss in this section.

We will try to make some general **recommendations** based on these requirements. For more information and recommendations, please consult the **technology summary** at the end of this document as well as the relevant **detailed implementation and technology discussions** in the previous sections.

### 4.2.1. Use as an Integrating User Interface Metaphor

This is one of the most common rationales for migrating to web-based interfaces. Fortunately, it does not impose any significant limitations on either your implementation strategy or your technology choice.

In general, the rewards of this integrating metaphor in terms of training time and ease of use will depend mostly on the consistent use of user interface elements across your applications. Also in this spirit you should avoid using technologies such as **ActiveX** or **Java** which "break" the standard user interface metaphor.

#### **Summary**

- Make consistent use of user interface elements
- Do not mix the "normal" web user experience with the ActiveX or Java user experiences

### 4.2.2. Creating Cross-Platform Solutions

Another common reason for choosing web-based technologies is the **cross-platform support** that they offer. If this is an important requirement for your project you will want to avoid technologies that lock you into a specific platform and give preference to cross-platform technologies.

Most platform dependent technologies are limited to the Windows95/NT platform and if you require wider support you should avoid these technologies. The major technologies to avoid in this case are **ActiveX** and **Visual Basic Scripting Edition**. If client-side application logic is required **JavaScript/ JScript / ECMAScript** and **Java** provide good cross-platform compatibility.

#### **Summary**

- Avoid Windows95/NT specific technologies
- Use cross-platform technologies such as ECMAScript or Java for client-side logic

### 4.2.3. Creating Cross-Browser Solutions

Even if cross-platform compatibility is not a major requirement of your project, cross-browser compatibility may well be. The two major browsers, **Netscape Communicator/Navigator** and **Microsoft Internet Explorer** share much functionality and it is thus not difficult to create applications that work on both.

The browsers differ mostly in their implementations of **recent technology** such as **DHTML** and **CSS**. Internet Explorer also contains a number of Microsoft only technologies such as **Visual Basic Scripting Edition** and **ActiveX**. Avoiding these technologies will provide you with basic cross-browser compatibility.

#### Summary

- Use mature cross-browser technologies
- Avoid browser specific technologies such as VBScript and ActiveX
- Avoid immature technologies

### 4.2.4. Response-Time Requirements

For many **data entry tasks** system response time is an important requirement. As mentioned before, web servers have no guaranteed response times and connections may or may not be successful. If response times are important for your application, then the web may well not be suitable for its implementation.

Response time depends on many factors. On local area networks, response time depends mostly on **server performance**, **network performance** and **server load**. A deliberate over-specification of the server hardware and the LAN network are likely to result in very good response times and will ensure that unsuccessful connections occur only rarely.

For **remote access** via the internet adequate external **connection bandwidth** is crucial, but comes at a price. Even then, however, performance will depend on the overall traffic routed via the internet and access times may be poor.

Other factors can influence the response time of web servers. The performance problems of the **Common Gateway Interface (CGI)** are discussed in the relevant technology section and will not be reiterated here.

If response time is crucial for your application, it might be necessary to implement your own **connection-oriented networking** in **Java** or **ActiveX** to get better performance. Doing so will, however, dilute the benefits that you can expect to gain by using web technology.

#### Summary

- Consider opting for a client/server solution with GUI interface
- Deliberately over-specify server hardware and internal as well as external network bandwidth

- Consider implementing your own connection-oriented networking protocol on top of the web protocols

#### 4.2.5. Interactivity Requirements for Complex Tasks

Many migration projects do not require a high **degree of interactivity**. If that is the case for your project, it is probably best to avoid client-side logic until the technology has further matured.

If your application does, however, require **direct feedback** and **client-side data validation** it may be necessary to implement client-side logic to fulfill these requirements.

Microsoft's **ActiveX** and **Visual Basic Scripting Edition** provide excellent interactivity, but at the cost of higher development overheads and the loss of both platform and browser independence.

Sun's **Java** technology offers similar advantages, but with the added benefit of good cross-platform and cross-browser compatibility.

**Dynamic HTML** in combination with **JavaScript/ JScript / ECMAScript** promises low-cost, standards-based and cross-platform development of highly interactive content. The technology is, however, still immature and can therefore, at press time, not be recommended for development. This situation may, however, improve rapidly and make this an excellent choice for future web-based user interface development.

#### Summary

- Carefully assess the interactivity requirements of your application
- If necessary consider adding client-side logic to your application

#### 4.2.6. Evolution potential

The **rapid pace of change** of web technology limits the evolution potential of any web-based user interface solution. The best strategy to avoid **technology obsolescence** and costly redevelopment is to use only standards-based technologies.

As mentioned before even though the standards processes have proved unable to keep up with the rapid rate of change in the web market, they nonetheless insure **continuity** in the technologies that they define. At this moment in time there are few full-standards in the internet market and many standards are still at the draft stages. Our recommendation would be to use only **full standards** wherever possible.

Full international standards exist for the following technologies: **HTML 4.0 / CSS1**, **Common Gateway Interface (CGI)** and **ECMAScript**.

While it is generally a good idea to avoid proprietary standards, one possible exception would be Sun's **Java** technology. Java has secured wide industry support and has gathered considerable momentum in the market place. Java is likely to have a considerable impact on the commercial IT market in the near future.

## Summary

- Use only full standards wherever possible
- Do not discard Sun's Java technology
- Consider opting for more mature technology such as client / server computing

### 4.2.7. Cost-efficiency

A migration from character-based interfaces to web-based interfaces can often be achieved at comparatively **low cost**. By far the most cost efficient way of achieving this migration is by using the server-side logic invocation afforded by the **Common Gateway Interface (CGI)**. Server-side scripting languages such as **PERL, UNIX shell scripts, TCL/TK, Visual Basic Scripting Edition** and **LiveWire JavaScript** allow for simple and rapid development of such server-side logic.

The development of more highly interactive interfaces involving **client-side logic** comes at a cost. Simple client-side scripting can be done at relatively low cost, but is not very powerful. This will most likely change as DHTML matures.

Development in **Java** or **ActiveX** allows for more powerful interfaces to be implemented, but the development costs may not be substantially different from those associated with full-featured client-server/ GUI development.

ActiveX web development, in particular, is little different from implementing a Windows95/NT client-server application.

## Summary

- Keep it simple
- Use server side scripts in conjunction with CGI
- Avoid complex client-side scripting or application programming

### 4.2.8. Security Requirements

Security is another important requirement for commercial IT systems and is discussed in more detail in the relevant **implementation section**.

In summary, the web does not offer the most secure development platform available. Real problems exist, but can be overcome by using adequate technology.

## Summary

- Carefully assess your security requirements
- Use a server which supports the Secure Socket Layer (SSL)
- Exercise caution when assigning access privileges for executable content
- Avoid ActiveX and other downloadable components

#### 4.2.9. Organizational Requirements

Technology and implementation decisions must take into account the existing IT infrastructure, knowledge and skill base as well as the IT policies of an organization.

Each organization is different and the **specific context** in which a solution will be developed and deployed necessitates a case-by-case assessment of implementation strategy and technology choice. Many organizations have a firm commitment to certain platforms and implementation tools which may influence technology choices. In this section we will specifically discuss two **common scenarios**, but similar implications exist for many other technologies.

In the past few years, the dominance of the Windows operating system has led many companies to make a strong commitment to **Microsoft technology** and the **Windows platform**. A common strategy for such enterprises is to develop **Visual Basic** front-ends to their core business applications. For such organizations the technologies offered by Microsoft and its **Internet Explorer** browser product provide a smooth and low-cost progression to web-based applications. **ActiveX** and **Visual Basic Scripting Edition** as well as **Active Server Pages** and the **Microsoft Internet Information Server** are easy to integrate into such an environment and may be the right choice for such enterprises.

Another long-term commitment which may dramatically affect the choice of an implementation strategy is a commitment to IBM's **Lotus Notes** groupware system. Recent versions of Lotus Notes have included the **Domino** web server, that provides a world-wide web gateway for most of the functionality offered by the full system. Companies that have already invested in this technology may find that it offers the best way for them to migrate to web-based interfaces.

#### Summary

- Depending on your local circumstances different migration paths may exist
- If your company is already heavily committed to Microsoft technology, Internet Explorer only technologies may provide a good alternative
- If your company is committed to Lotus Notes, the simplest migration path is through Domino

## 5.

## Appendices

### 5.1. Summary of recommendations

<b>General</b>
Avoid immature technology
Avoid proprietary technology
Avoid unnecessary complexity
<b>Using the web as an integrating user interface metaphor</b>
Make consistent use of user interface elements through your applications
Do not mix the "normal" web user experience with the ActiveX or Java user experiences
<b>Developing cross-platform solutions</b>
Avoid Windows95/NT specific technologies
Use cross-platform technologies such as ECMAScript or Java for client-side logic
<b>Developing cross-browser solutions</b>
Use mature cross-browser technologies
Avoid browser specific technologies such as VBScript and ActiveX
Avoid immature technologies
<b>Satisfying Response-time requirements</b>
Consider opting for a client/server solution with GUI interface
Deliberately over-specify server hardware and internal as well as external network bandwidth
Consider implementing your own connection-oriented networking protocol on top of the web protocols
<b>Satisfying interactivity requirements for complex tasks</b>
Carefully assess the interactivity requirements of your application
If necessary consider adding client-side logic to your application
<b>Improving the evolution potential of the solution</b>
Use only full standards wherever possible
Do not discard Sun's Java technology

Consider opting for more mature technology such as client / server computing
<b>Producing cost-efficient solutions</b>
Keep it simple
Use server side scripts in conjunction with CGI
Avoid complex client-side scripting or application programming
<b>Satisfying security requirements</b>
Carefully assess your security requirements
Use a server which supports the Secure Socket Layer (SSL)
Exercise caution when assigning access privileges for executable content
Avoid ActiveX and other downloadable components
<b>Organizational Requirements</b>
Keep in mind that depending on your local circumstances, different migration paths may exist
If your company is already heavily committed to Microsoft technology, Internet Explorer-only technologies may provide a good alternative
If your company is committed to Lotus Notes, the simplest migration path is through Domino

## **5.2.References**

The migration of character-based interfaces to web-based interfaces has, as yet, not received much attention from either the commercial or the academic world and as a result few documents exist that specifically address these issues.

The best repository for information on the development of web-based interfaces is the world-wide-web itself. In this section we will provide some pointers to relevant information on the internet.

### **5.2.1. Software and Systems Reengineering**

The **RenaissanceWeb** is a good starting point for exploring the software reengineering field. In it you will find pointers to web-sites dealing with all aspects of software systems reengineering including the migration to web-based interfaces.

The RenaissanceWeb is located at:

<http://www.comp.lancs.ac.uk/projects/RenaissanceWeb/>

### 5.2.2. Web-based Application Development

A large body of online material is available on this subject. Rather than making a futile attempt at listing all of these sites in this reference section, we will instead provide you with pointers to extensive online lists of such materials, which are certain to be updated regularly.

<b>Web Programming in General</b>
<a href="http://www.yahoo.co.uk/Computers_and_Internet/Internet/World_Wide_Web/Programming/">http://www.yahoo.co.uk/Computers_and_Internet/Internet/World_Wide_Web/Programming/</a>

The following table provides pointers to specific technologies and implementation issues that we have discussed in this document:

<b>Active Server Pages</b>
<a href="http://www.eu.microsoft.com/iis/guide/aspoverview.asp?A=2&amp;B=9">http://www.eu.microsoft.com/iis/guide/aspoverview.asp?A=2&amp;B=9</a>
<b>Cascading Style Sheets</b>
<a href="http://www.yahoo.co.uk/Computers_and_Internet/Information_and_Documentation/Data_Formats/HTML/Cascading_Style_Sheets/">http://www.yahoo.co.uk/Computers_and_Internet/Information_and_Documentation/Data_Formats/HTML/Cascading_Style_Sheets/</a>
<b>Common Gateway Interface</b>
<a href="http://www.yahoo.com/Computers_and_Internet/Software/Internet/World_Wide_Web/Browsers/Mosaic/NCSA_Mosaic_for_X/Common_Client_Interface_CCI/">http://www.yahoo.com/Computers_and_Internet/Software/Internet/World_Wide_Web/Browsers/Mosaic/NCSA_Mosaic_for_X/Common_Client_Interface_CCI/</a>
<a href="http://www.yahoo.com/Computers_and_Internet/Internet/World_Wide_Web/Programming/Forms/">http://www.yahoo.com/Computers_and_Internet/Internet/World_Wide_Web/Programming/Forms/</a>
<b>Cookies</b>
<a href="http://www.yahoo.com/Computers_and_Internet/Internet/World_Wide_Web/HTTP/Protocol_Specification/Persistent_Cookies/">http://www.yahoo.com/Computers_and_Internet/Internet/World_Wide_Web/HTTP/Protocol_Specification/Persistent_Cookies/</a>
<b>Dynamic HTML</b>
<a href="http://www.yahoo.co.uk/Computers_and_Internet/Information_and_Documentation/Data_Formats/HTML/Dynamic_HTML/">http://www.yahoo.co.uk/Computers_and_Internet/Information_and_Documentation/Data_Formats/HTML/Dynamic_HTML/</a>
<b>HTML</b>
<a href="http://www.yahoo.co.uk/Computers_and_Internet/Information_and_Documentation/Data_Formats/HTML/">http://www.yahoo.co.uk/Computers_and_Internet/Information_and_Documentation/Data_Formats/HTML/</a>
<b>HTTP</b>
<a href="http://www.yahoo.com/Computers_and_Internet/Internet/World_Wide_Web/HTTP/">http://www.yahoo.com/Computers_and_Internet/Internet/World_Wide_Web/HTTP/</a>

<b>Java</b>
<a href="http://java.sun.com/">http://java.sun.com/</a>
<a href="http://www.yahoo.com/Computers_and_Internet/Programming_Languages/Java/">http://www.yahoo.com/Computers_and_Internet/Programming_Languages/Java/</a>
<b>JavaScript</b>
<a href="http://www.yahoo.com/Computers_and_Internet/Programming_Languages/JavaScript/">http://www.yahoo.com/Computers_and_Internet/Programming_Languages/JavaScript/</a>
<b>Microsoft ActiveX</b>
<a href="http://www.yahoo.com/Computers_and_Internet/Software/Operating_Systems/Microsoft_Windows/Windows_95/Technical/ActiveX/">http://www.yahoo.com/Computers_and_Internet/Software/Operating_Systems/Microsoft_Windows/Windows_95/Technical/ActiveX/</a>
<b>Security</b>
<a href="http://www.yahoo.com/Computers_and_Internet/Internet/World_Wide_Web/Security/">http://www.yahoo.com/Computers_and_Internet/Internet/World_Wide_Web/Security/</a>
<b>Visual Basic Scripting Edition</b>
<a href="http://www.eu.microsoft.com/scripting/default.htm?/scripting/vbscript/default.htm">http://www.eu.microsoft.com/scripting/default.htm?/scripting/vbscript/default.htm</a>
<b>XML</b>
<a href="http://www.yahoo.co.uk/Computers_and_Internet/Information_and_Documentation/Data_Formats/XML/">http://www.yahoo.co.uk/Computers_and_Internet/Information_and_Documentation/Data_Formats/XML/</a>