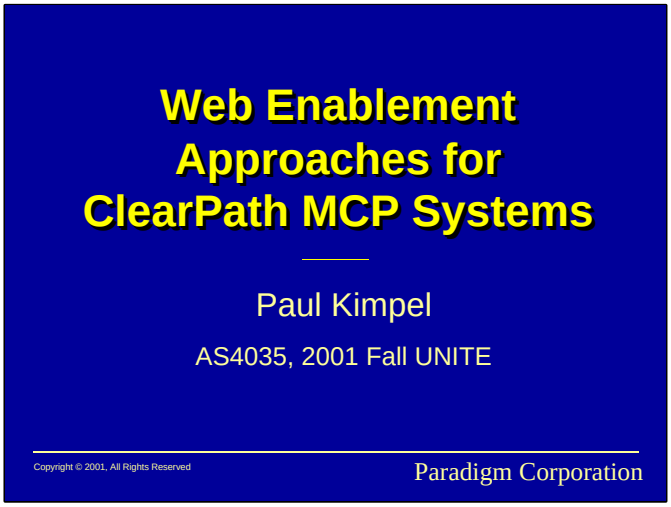



# Web Enablement Approaches for ClearPath MCP Systems

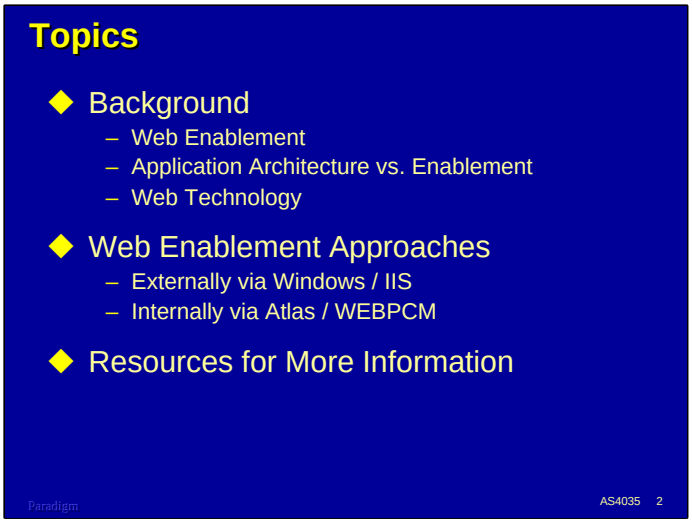
2001 Fall Unite Conference, Phoenix, Arizona.

Paul Kimpel

Paradigm Corporation
San Diego, California

http://www.digm.com

e-mail: paul.kimpel@digm.com




This presentation discusses a number of ways that you can web enable existing ClearPath MCP applications.

By "web enable" we mean implementing an interface that allows users to access the data and procedures of the application from a standard web browser over the Internet, or over an internet-like network.

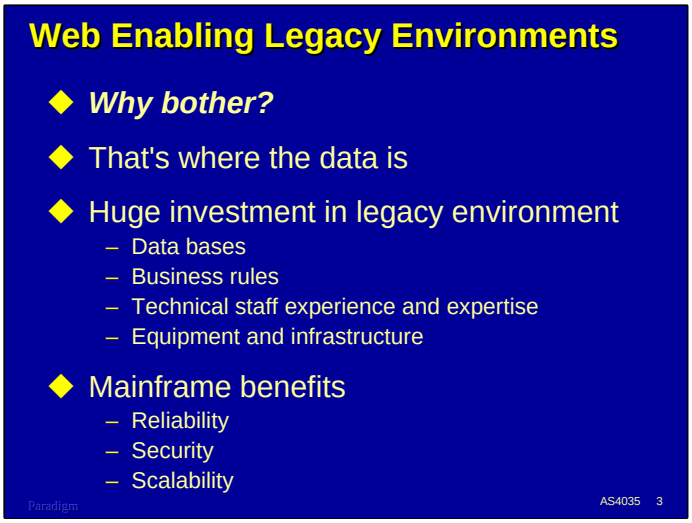
We'll first talk about why you might want to web enable your application and the most common methods currently used for doing so.

In order to understand how web enablement can affect an existing application, we'll also talk about the general architecture of most on-line applications and how the various enablement methods interact with that architecture. We'll also briefly review basic web technology and the components that are typically used in web-based applications.

The major part of the presentation will discuss the methods for web enablement using two separate approaches

- Externally using the Microsoft Windows Internet Information Server (IIS) with Active Server Pages (ASP).
- Internally using the Unisys Web Transaction Server for ClearPath MCP, also known as Atlas, and the WEBPCM module of COMS CCF.

Finally, we'll discuss some resources you can explore for more information on the subject.




Why bother even trying to web enable your existing legacy application environment? Why not just chuck the whole mainframe and all those old programs and do it over with the platforms and systems that everyone else seems to be using (or that some clever marketing by major vendors leads you to believe everyone else is using)?

The notorious stickup artist Willie Sutton, when asked why he robbed banks, reportedly replied, "Because that's where the money is."

The same thing can be said for why we should web enable legacy mainframe environments –

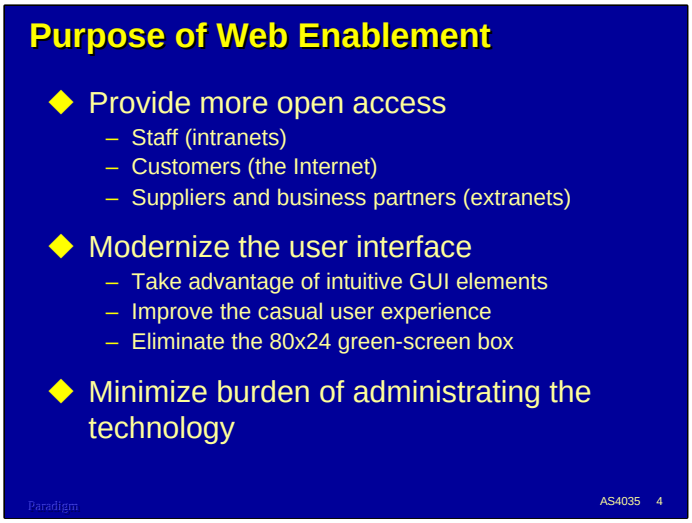
*because that's where the data is.*

Over the years we have built up an enormous investment in our legacy applications. The data bases are designed and tuned and running reliably. The business rules and operating procedures have been ironed out, and are often expressed only in the running code of the existing applications. The technical staff has in many cases lived with the applications for years, and has learned how to enhance and maintain the environment, even using traditional – and dated – tools like COBOL. Finally, there's the investment in the mainframe server itself and it's system and environmental software.

That is a lot to just throw out so you can provide some web access. What's more, experience over the past ten years shows that the throw-it-out-and-start-over approach is *very* expensive and *very, very* risky. The success rate on such projects is pretty poor, in fact.

Another thing we've learned over the past ten years is that the mainframe is not quite the dinosaur it was made out to be. Mainframe systems have advanced with the times, and are continuing to be superior platforms in terms of reliability, security, and scalability.

(Footnote: Late in his life, Willie Sutton admitted that he never uttered that famous quote. It apparently was made up by a reporter. What Willie did say was that he robbed banks because he enjoyed it).

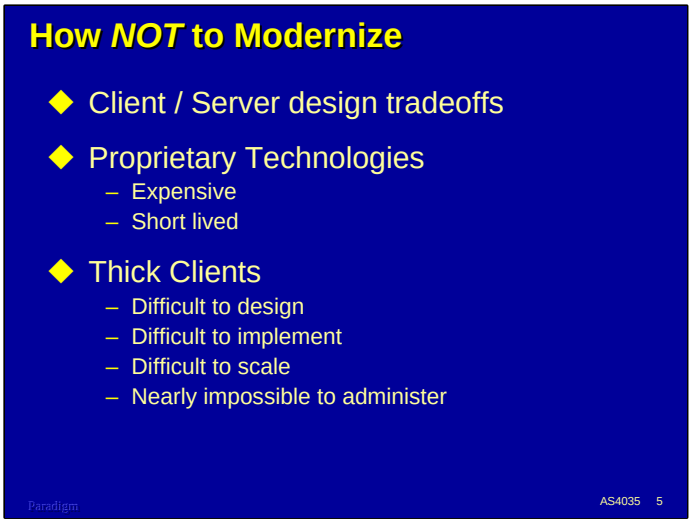



What are we really trying to do when we web enable an existing application?

The primary goal is usually to provide more open access to the application and its data. This openness may be needed for internal staff via a corporate intranet; it may be needed for customers via the general Internet; it may be needed for suppliers and other business partners via the Internet or an extranet. You may potentially have a need to open your applications to any combination of these three.

A second goal is often to modernize the user interface. Modern graphical user interfaces are very appealing, especially for casual users. If properly done, GUIs can approach the efficiency and throughput of traditional "green-screen" interfaces. One of the major improvements possible with web browser interfaces is eliminating the crowded, 80-column by 24-line box that is necessary with green screens.

A third goal is to provide openness and an improved user interface without undertaking a lot more administrative and support overhead. Web enablement offers a number of advantages in this area, since most of the process remains on the same central server as the application, or on a related server nearby.



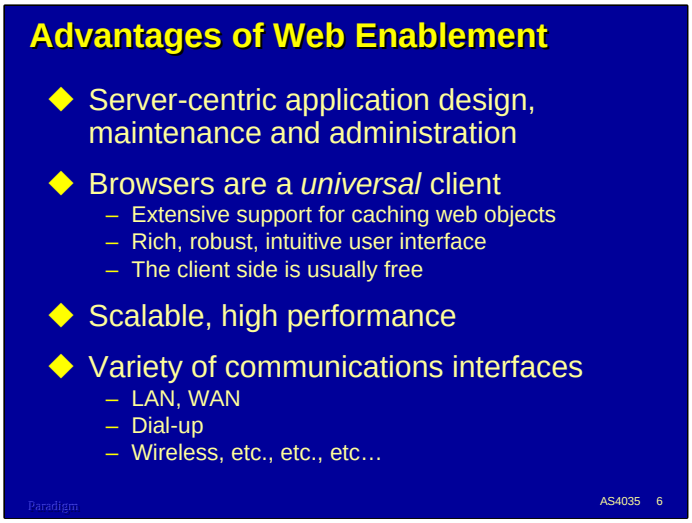

One of the bitter lessons for IT during the late 1980s and 1990s is that the power and flexibility of modern workstations doesn't necessarily translate into more efficient operations, improved productivity, or lower overall system costs.

It seemed like such a great idea – move the user interface and a good piece of the application processing out to these extremely fast workstations and reduce the central server to little more than a data base box.

In reality, designing a distributed application like this turned out to be much more complex and difficult than almost anyone anticipated. A number of proprietary technologies and tools hit the market, but most of them proved to be expensive and short lived.

Thick clients running on those powerful workstations also had their problems. They were quite difficult to design and implement, and distributed designs using them proved to have serious scalability problems. The thing that drove the stake through the heart of the initial client-server initiative, however, turned out to be ongoing administration and support. All of that custom code had to be installed on the workstations scattered throughout the enterprise, and what's worse, every time the application changed, you had do to it all over again. This proved to be a lot more trouble than simply recompiling a program on the host and bouncing the on-line copy.

This approach to application modernization virtually required you to buy into one vendor's proprietary approach, along with their ready supply of platforms, software, and consulting services. It was really good for Microsoft and Intel and Oracle. It was really bad for almost everybody else.




Web enabling an application has most of the benefits that we sought with the original client-server approach, but with a lot fewer drawbacks.

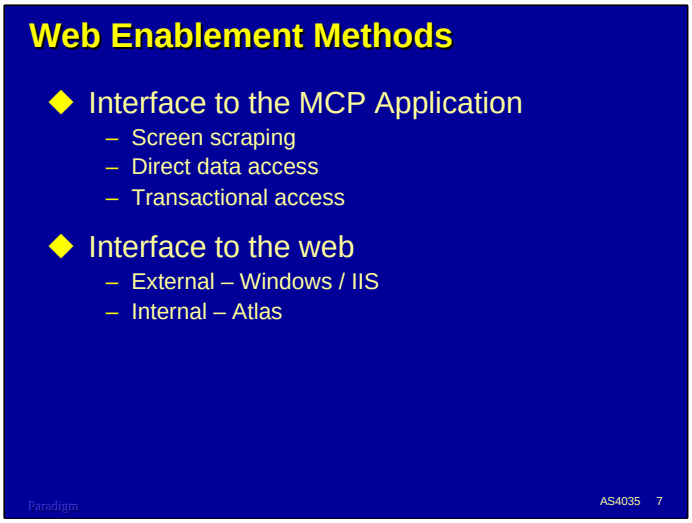
What we tended to forget during the cognitive white-out from the marketing blizzard surrounding the early client-server thrust was that the mainframe has *always* been a client-server architecture. True, the clients were really thin (green screens, preceded by punched cards and key-to-tape), but there was a clear delineation of function and responsibility that did not require a lot of distributed administration and support.

Web enablement carries this traditional separation of function forward while adding a lot of value. Modern web browsers are anything but a "thin" application. What makes them good for thin client implementations is that they are a *universal* client. They know nothing about your specific application, and nothing else on the workstation has to know anything about the application, either. All the application-specific elements of the user interface are stored, maintained, and administered from a central server. A really nice set of benefits accrue from the use of web protocols in this environment:

- The elements of the user interface are loaded to the client from the server automatically when needed.
- The client can cache them so they do not need to be loaded over and over again as they are used.
- The client can efficiently determine when elements of the user interface have been updated, and automatically re-cache them the next time they are referenced.

Graphical web browsers, along with current client-side technologies like Java, JavaScript, and Cascading Style Sheets, provide a rich and intuitive user interface. Properly designed, such interfaces can be as good as, or better, than workstation GUIs. Best of all, the browser usually comes at no additional cost.

In addition, web-based applications are highly scalable and can fairly easily achieve high performance. A variety of communications interfaces can be used, including dial-up modems and the new wireless networking technology.




There are a number of ways to web enable your existing application. They vary based on two main considerations:

- How the web enablement interfaces to your application code.
- How it interfaces to the web itself.

We will talk about three general methods for interfacing web capabilities to an existing application:

- Screen scraping – translating the existing green-screen data to a web representation.
- Direct data access – bypassing the application code and accessing the data base directly.
- Transactional access – bypassing the existing green-screen interface, but retaining the rest of the legacy application.

We will also talk about two common ways of implementing web enablement for ClearPath MCP applications

- External to the MCP environment, using Microsoft IIS running under of Windows.
- Internal to the MCP environment, using the Atlas web server.

## Web Enablement Matrix

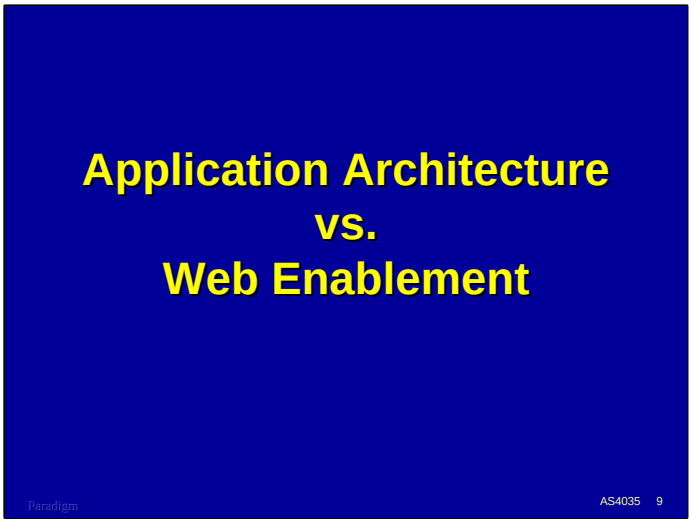
| | External – IIS | Internal – Atlas |
|---|---|---|
| Screen scraping | ASP+VBS+COMTI<br>Proprietary front-ends | WEBPCM with COMS<br>Processing items |
| Direct data | ASP+VBS+ADO+ODBC<br>ASP+VBS+ADO+OLE DB<br>Proprietary tools | (nothing at present) |
| Trans-actional | ASP+VBS+OpenTI+OLTP<br>ASP+VBS+COMTI<br>Proprietary tools (ICE, …) | WEBPCM+Custom app<br>AAPI+Custom app |



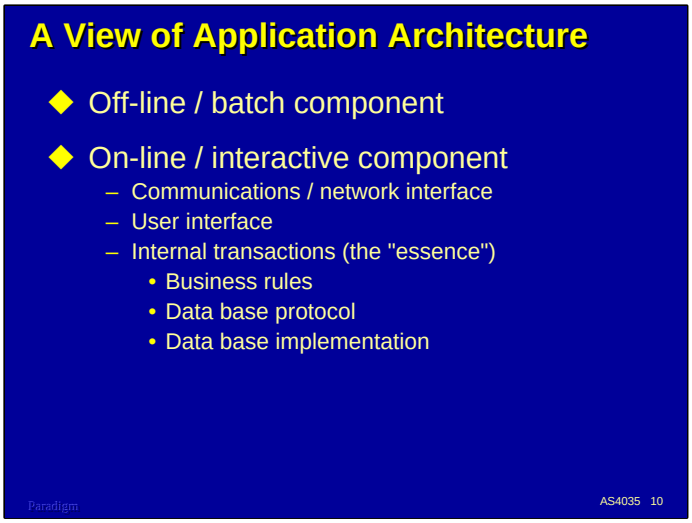
Combining all these options we get a matrix. Each cell of the matrix shows tools and technologies that can be applied for that set of options.

Note that this matrix is not at all comprehensive, and that we will be discussing just some of the most common or well-known tools and technologies.

Also note that, in almost all the cases, several tools or technologies must be combined to achieve a solution. This is especially true for the Windows/IIS approaches. We will discuss each of these items, and show how they fit together as part of a total enablement solution.




Before talking about the various web enablement methods in detail, we need to discuss how most legacy on-line applications are structured. Different enablement methods connect to different internal pieces of an application, and understanding how this affects the design, implementation, and functionality of the resulting interface is important.



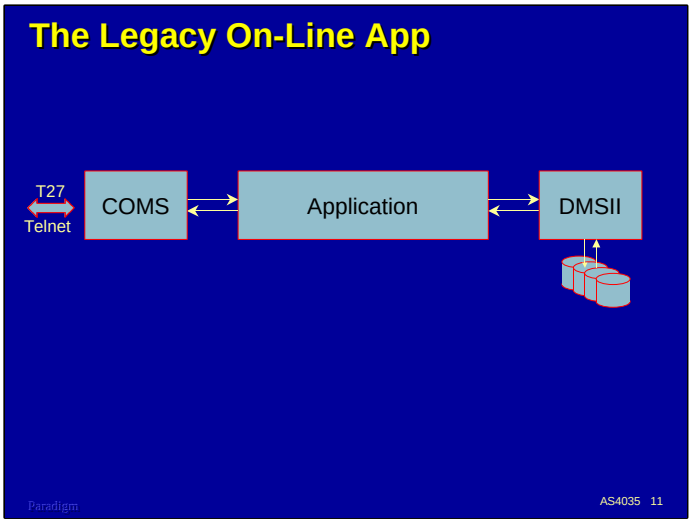

The typical legacy application, say one that has been implemented or reengineered since the 1970s, has two main components:

- The off-line or batch component. This part of the application is usually concerned with reporting, bulk update of the data, housekeeping chores, and other activities that either operate in the background, or which do not require a quick response.
- The on-line or interactive component. This part of the application is usually concerned with responding in real time (or near-real time) to queries for information and to transactions that update the data.

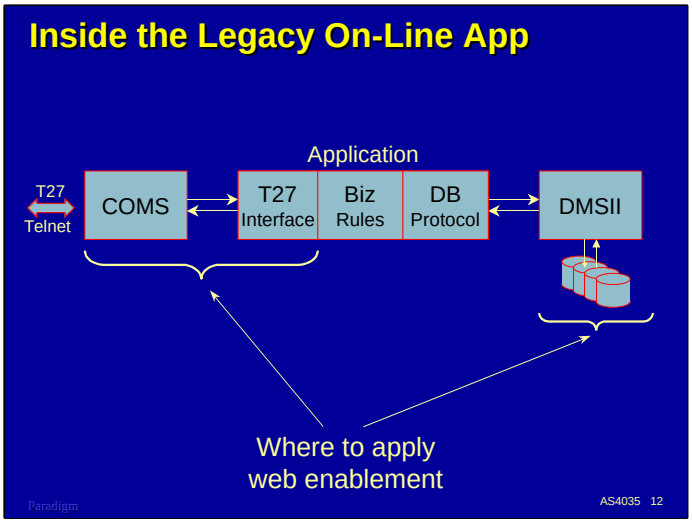
For the purposes of this discussion, we are only going to consider the on-line portion of the application. The on-line portion can be broken down into three main parts:

- The communications or network interface. This is the part of the system that connects the application software to end users. Today on ClearPath MCP systems, this typically consists of COMS, Telnet, and TCP/IP.
- The user interface. This is the part of the system that receives application-specific messages from end users, parses and verifies the messages, and formats output messages in reply. This is the portion of legacy applications that deal with T27 forms, numeric editing, upcasing text, and so forth.
- Internal transactions. This is the part of the system that does all of the heavy lifting for the application. It contains the "essence" – the business rules that distinguish this application from all others and implement the organization's business policy. It also contains the data base, and the application-specific protocol that translates between business rules and the data base structures.

Note that this is an idealized view of the application. In the actual implementation, you probably won't see nice clean divisions between the user interface and the transactional essence, much less between the internal components of the transactions. This is an important view of the application to keep in mind, however, because it has a lot to do with how you choose to web enable the application and what it's going to take to accomplish that.




If we look at a typical on-line program from a typical legacy application for ClearPath MCP systems, the program stands between a DMSII data base on one end, and COMS on the other.

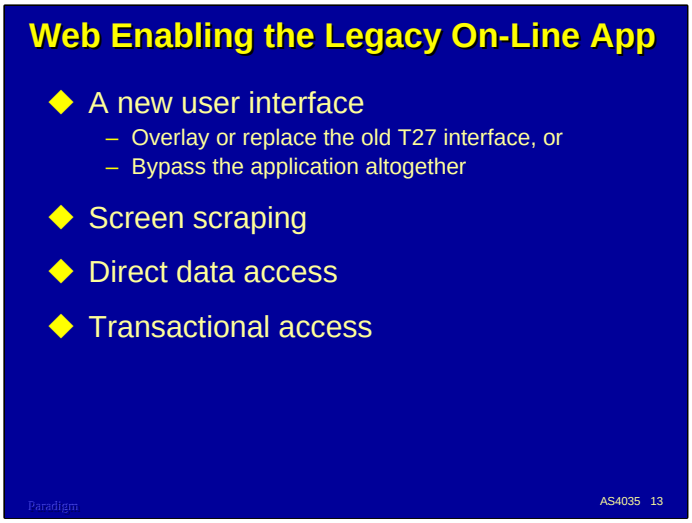



If we look inside that legacy on-line program, we typically see a general scheme like the following:

- The program receives a message from COMS.
- The fields of the message are parsed and verified.
- Based on the contents of the message, the program retrieves (and possibly) updates records in the DMSII data base. In the process of accessing the data base, additional verifications may be performed (presence or absence of records, validity of codes and identifiers, etc.). This is the essence of the transaction – the implementation of the business rules.
- Finally, the program formats a reply message, probably using T27 forms mode, and returns it to the end user.

An application usually has many such transaction processing programs, but their high level interface to the rest of the system is generally the same.

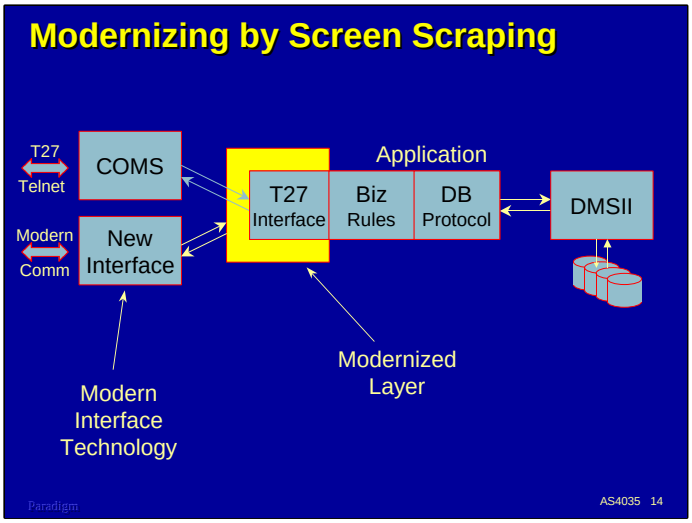
The important point here is that web enablement can be applied either to the existing user interface or directly to the data base, bypassing the application altogether. In most cases, you don't want the business rules or data base to change – they are, after all, what define the application.




Therefore, what we are really trying to do in web enabling an application is putting a new face on it and delivering that new face over a more open communications network. We can do that by overlaying (hiding) the existing T27 interface, building a new interface that either replaces or works in parallel with the T27 one, or bypassing the application altogether and going straight for the data base.

The next slides show the three main ways of doing this:

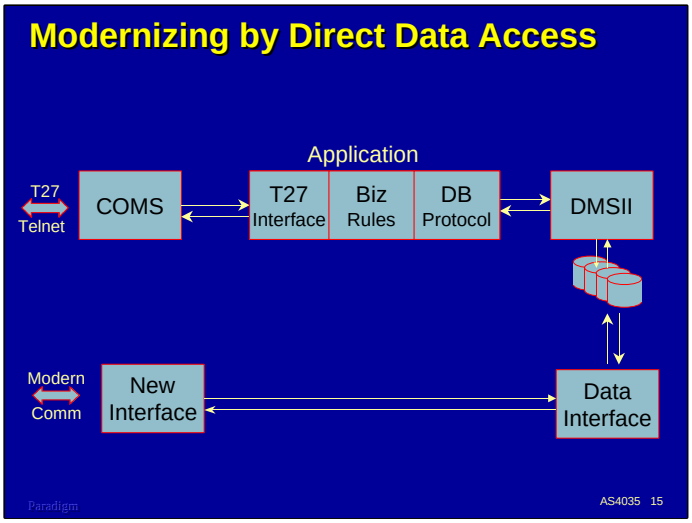
- Screen scraping,
- Direct data access, or
- Transactional access




With screen scraping, we leave the existing application completely untouched. To provide a new interface, we capture the data stream going and coming from the application and transform it for the new interface.

For web enabling an MCP application, this usually means capturing the T27 data stream on either side of COMS, transforming T27 formatting commands to HTML forms, and transforming input from HTML forms to the fixed-field data format which the application is expecting.

The transformations do not even have to occur on the MCP host. There are several proprietary solutions that implement this on an external system (usually a Windows NT/2000 server). The external server typically talks Telnet to the MCP host and HTML to the external network.

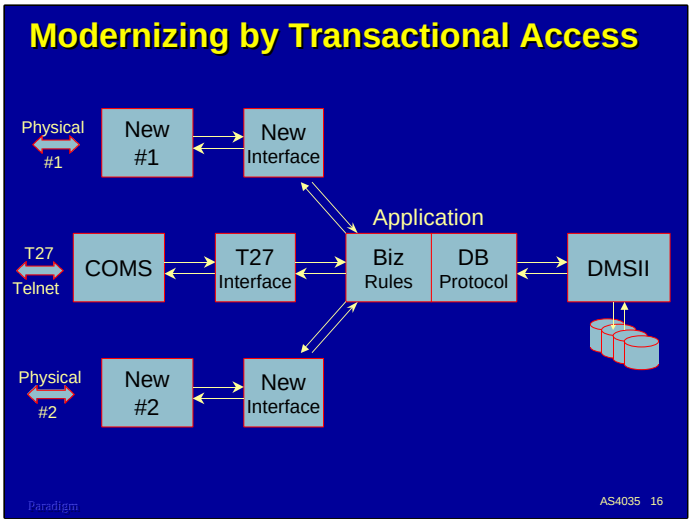



With direct data access, we bypass the application entirely and talk directly to the data base.

This is a very popular approach, especially for access from external web servers, since it avoids the problem of interfacing to the existing application code. The data interface typically uses some form of SQL, which provides a very powerful mechanism to retrieve data, especially when the queries must be composed from user input.

This is an excellent approach for implementing ad hoc query, and offers a great deal of flexibility in formatting the results for the end user. However, it is usually not a very good approach for updating the data base. The reason for this is that updates typically require extensive editing and validation before the data can be stored, and frequently require that multiple records in the data base be accessed and updated in a particular fashion.

In the legacy application, the business rules and data base protocol components perform this detailed verification and coordination. If similar update transactions are to be implemented through an external data interface, all of this code must be replicated in the external interface. Not only is it often a challenge to extract all this application knowledge from the legacy app, but ongoing maintenance and enhancement of the application must now be applied in two places. This nearly doubles the maintenance cost. In addition, keeping modifications in sync between the two implementations is extremely difficult.




With transactional access, we isolate the internal transactions – that essence of the application – and either replace or bypass the legacy user interface to implement a new interface that talks directly to the internal transactions.

Except for the powerful ad hoc query capabilities that typically come with direct data interfaces, this approach is the most flexible and general. You can completely redesign the user interface to better suit the capabilities of web browsers and take full advantage of the browser's larger canvas, GUI elements, and client-side interactivity. Since you are retaining the existing business rules and data base protocol, update transactions are much easier to implement and maintain.

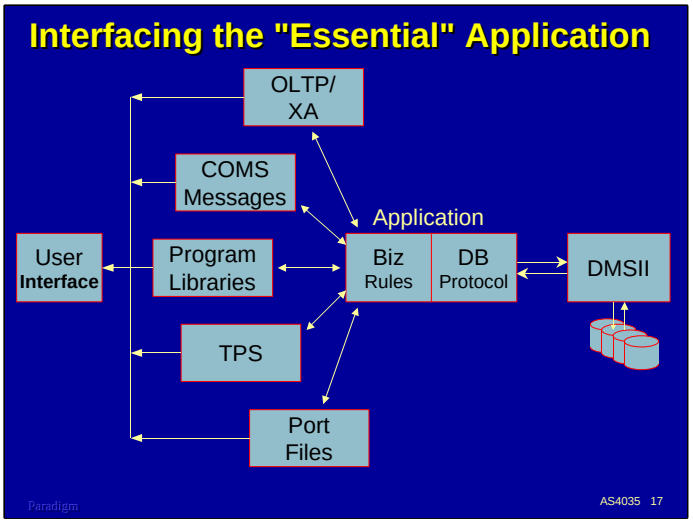
The main disadvantages to this approach are:

- It's usually a lot of work, and
- Most legacy applications are not structured with a nice, clean interface to the internal business rules. This usually means that you need to do some reengineering first to expose the internal transactions in a way that multiple user interfaces can connect to them independently.

Even though this is a lot of work and often difficult, over the long run it may be the best choice, especially if update is involved.

There can be other advantages to restructuring an application this way. Many applications implement essentially the same transaction in multiple forms already (think about all the ways there are to deposit money in a checking account), and isolating and consolidating the internal transaction will probably have long-term maintenance and enhancement benefits.

Also, nothing says that you have to pick just one enablement method and stick with it. You can start with screen scraping, since that usually delivers a basic web interface very quickly, implement direct data access for ad hoc query, and work on transactional access for those updates and other transactions which are too complex to replicate in an external interface.




The idea of ultimately being able to isolate the business rules and data base protocol from the user interface(s) of an application is a powerful one, and so important, I want to discuss for a minute the various ways you can do this.

The program library facility of the MCP is an excellent mechanism for isolating and implementing internal transactions. You put the business rules and data base protocol in the library, and define transaction record layouts and library entry points to communicate with the internal transactions. You can then access these transactions from COMS modules, batch programs, XA/OLTP interfaces, and whatever else will be coming along in the future.

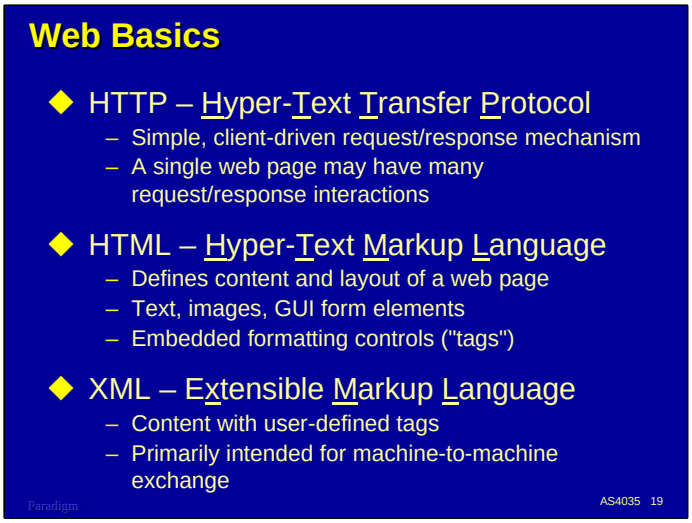
Unisys has a product, the *Transaction Processing System*, or TPS, that builds on program libraries to implement exactly this kind of internal transaction mechanism. It is a part of the DMSII suite, but is a separately-licensed product. You define transaction records in a DASDL-like language called TFL, and write libraries that implement business rules and data base protocols which operate on the transaction records. The TPS then generates libraries to connect and manage the flow of data between client tasks and the transaction libraries.

The TPS is an excellent facility for factoring the internal essence from an application's user interface, and probably the least-used and least-understood product in the ClearPath MCP software catalog.

Aside from library-based methods there are a few other techniques that can be used to isolate the internal transactions from the user interface:

- You can separate the user interface and internal transaction code into separate COMS programs and communicate among them using standard COMS TP-to-TP messaging.
- You can also use the distributed transaction processing (XA/OLTP) facility in the MCP to isolate transaction code and provide multiple user interfaces to it. We will discuss some enablement approaches later that make use of this facility.
- Finally, you can use port files to separate user interface code from internal transaction code in much the same way you can do it with COMS TP-to-TP messages.


As a final piece of background before launching into specific web enablement approaches, we will discuss the basic technologies involved in implementing a web interface.



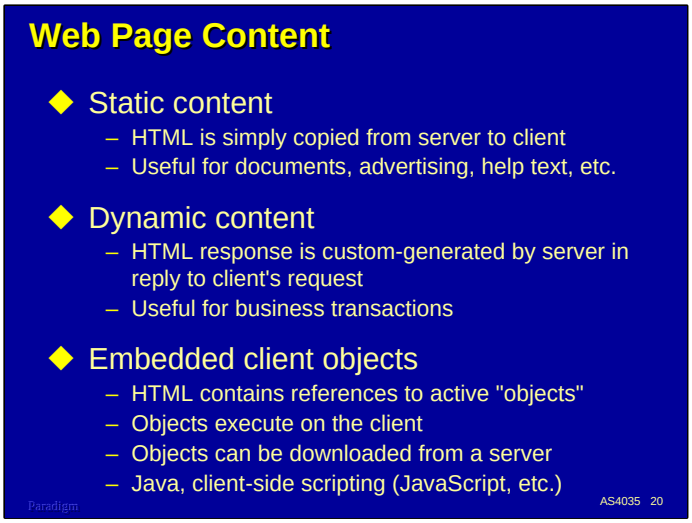

At the risk of telling everyone something they already know, the World Wide Web is based on two fundamental standards, HTTP and HTML.

HTTP, the Hyper-Text Transfer Protocol, is a standard for web browsers communicating with web servers. The basic protocol is relatively straightforward – the browser (or "user agent", in the standard's parlance) opens a TCP/IP connection to a server and sends requests. The server responds on the same connection with results. In the simplest case, the browser requests a file and the server simply sends it. A typical web page is built from many of these request/response interactions – one for the base page, plus additional ones for the images and other elements referenced by the base page. When all of the request/response interactions are complete, the browser closes the TCP/IP connection. This is why the web is "stateless" – there is no long-term connection or session between the browser and the server.

HTML, the Hyper-Text Markup Language, is the notation used to describe the contents and layout of web pages. An HTML message usually contains bodies of text, organized and formatted by embedded control information called "tags". The HTML standard is basically a definition of all the types of tags and their effect on the formatting of the page. The "hyper" in HTML comes from its ability to embed links to other web documents in the text and for the browser to retrieve those documents when the links are activated by the end user.

In addition to text, HTML can specify links to images and other graphical elements, and can describe a basic set of GUI elements arranged in a "form". Activating one of the elements of the form causes the browser to send the contents of all form elements to the server in a specially-formatted text string. HTML forms follow much the same idea as do green-screen forms, which makes HTML attractive for implementing user interfaces to legacy applications. Proper use of the GUI form elements can produce a much more attractive and intuitive user interface than green screens.

Development of the HTML standard has currently reached a plateau. Further development of web interfaces has recently been focused on a generalization of HTML, the Extensible Markup Language, or XML. While HMTL defines both the syntax and semantics of its tags, XML defines only the syntax, leaving the semantics up to another process. XML could be used to develop a richer page layout facility than HTML offers, but a lot of interest has been directed toward using XML as a vehicle for direct machine-to-machine exchange – the over-hyped but infinitely promising "e-commerce".

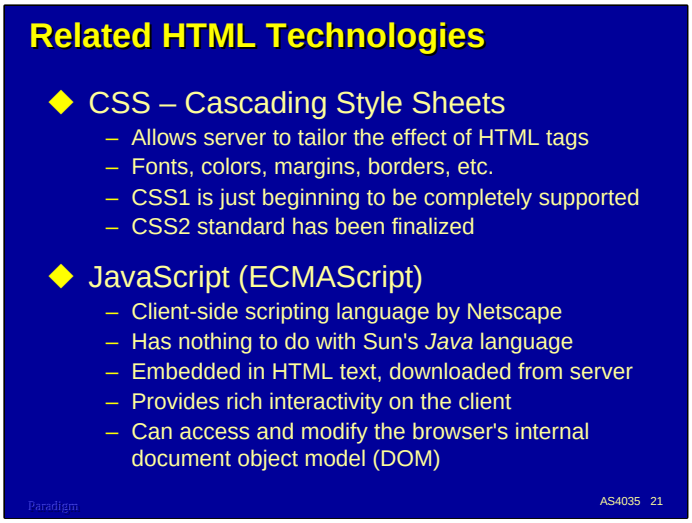



Web pages can be composed in three basic ways. These can be used singly or in combination.

The first is static content. The HTML and other elements for the page can be prepared in advance and stored as files on the web server. When the page is requested by a browser, these files are simply downloaded to the browser without modification. This is a very powerful and efficient mechanism for publishing documents, providing standardized information ("boilerplate"), and distributing advertising and product information – all things that are typically prepared in advance and do not change very often. This is what got the web started, and is still an important part of it.

The second way to compose a web page is with dynamic content. From the browser's point of view, there is no difference between this and static content. The difference is at the server. Instead of simply pumping files down the network, the server creates the HTML and other page elements dynamically, based on the user's request. This is not all that different from the way that legacy applications communicate with end users – the host analyzes information from the user and composes a customized message in reply. Dynamic content is almost always used with for interfaces to business applications, and is the one we will focus on the most.

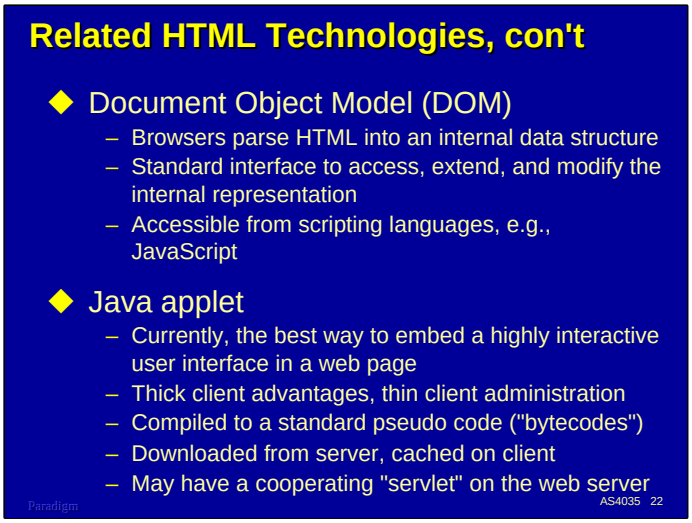
The third way to compose a web page is with embedded client objects. With the first two methods, the browser simply interprets the HTML stream coming from the server. There may be some limited client-side interaction with GUI form elements, but once the page is built on the client, the browser cannot do much else with it. It is possible, however, to embed active objects and executable instructions in the HTML stream. These enable the browser (and the end user) to interact more intimately with the page, and take advantage of the powerful processor in the client workstation. The two best-known active client enablers are Java and JavaScript.




There are a number of technologies that can be used to enhance the capabilities of HTML on the client side.

Cascading Style Sheets are a mechanism that can temporarily modify the semantics of standard HTML tags that deal with the style, or visual appearance, of the web page. Using CSS, you can easily adjust fonts, colors, margins, borders, and other visual characteristics of the page. The CSS1 standard has been out for several years, but browsers are just now beginning to provide complete support for it. The CSS2 standard has been finalized, but implementations are just beginning to be built.

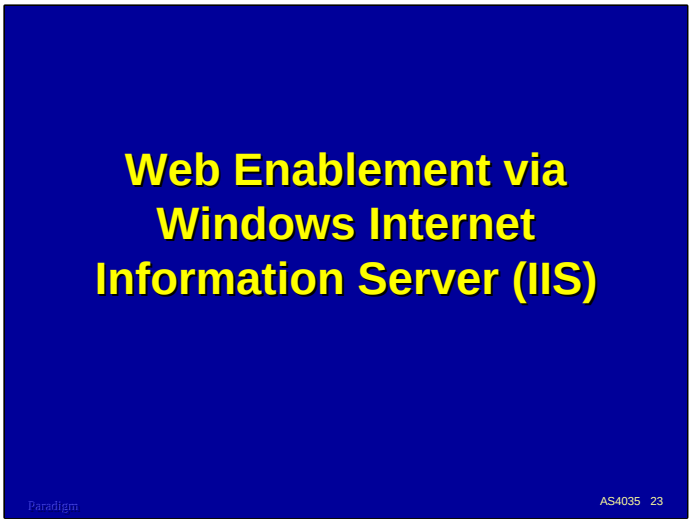
JavaScript, also known as ECMAScript, is an interpretive language that is downloaded in text form and interpreted on the browser. It was originally developed by Netscape for their Navigator browser. The European Computer Manufacturers Association has recently developed a standard for the language, hence ECMAScript. Except for a vague similarity in syntax, JavaScript is not related at all to Java. It is frequently used to enrich the client-side user interface and perform local editing before sending a form to the server. JavaScript can also access the browser's internal Document Object Model to access, and actually modify, the displayed web page.




The Document Object Model is another technology which as been around for a while, but until recently has suffered from a lack of standardization. Browsers parse HTML into an internal representation which they use to manage the display of the resulting web page and interaction with the end user. The DOM provides an interface to this internal representation, allowing active embedded elements in the web page to manipulate and modify the web page itself. JavaScript and other scripting languages can access the DOM. Now that the model has been standardized, we should begin seeing more effective use of this extremely powerful concept.

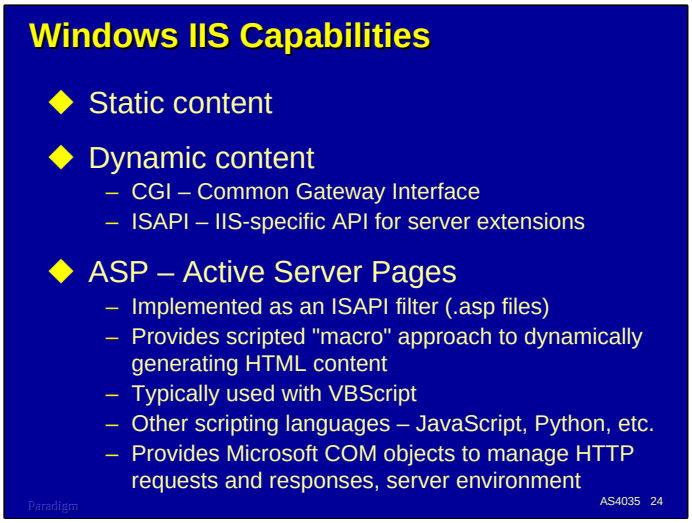
Java applets (and to some extent, Java itself) has suffered the fate of hype – an excellent technology embarrassed by outlandish expectations. Even so, it remains the best way to embed a highly sophisticated and interactive user interface in a web page without the administrative drag of installing custom software on the workstation. In effect, it delivers most of the advantages of a thick client with almost none of the disadvantages. Java applets operate using an interpreted pseudo code ("bytecodes") downloaded on demand from the server. The browser contains an interpreter which executes the bytecodes. In addition to enhancing the user interface, applets may communicate with Java "servlets" resident on the web server.

All of the related HTML technologies discussed above operate on the client. When we mention HTML in the rest of this presentation, we will assume this to include HTTP and these related technologies as well.




The first set of enablement approaches we will discuss in detail are those used with Microsoft Windows and its Internet Information Server, IIS.

In the following set of examples, the web server operates on a separate system outside the MCP environment, perhaps on the Intel side of a ClearPath system. The web server and its various enabling tools communicate with the MCP environment, typically over TCP/IP or the internal ClearPath VLAN.




IIS is a very full-featured web server, offering facilities for serving up both static and dynamic content.

Static content consists simply of files stored in the normal Windows file space. For these sources, IIS does little more than send the files on request.

IIS has two primary methods for generating dynamic content. One is CGI, the Common Gateway Interface, a general API that has been around since the original CERN web server. CGI allows the web server to fire up an external program, pass an HTTP message to it, and receive an HTML response for forwarding to the browser. This type of interface has a lot of overhead.

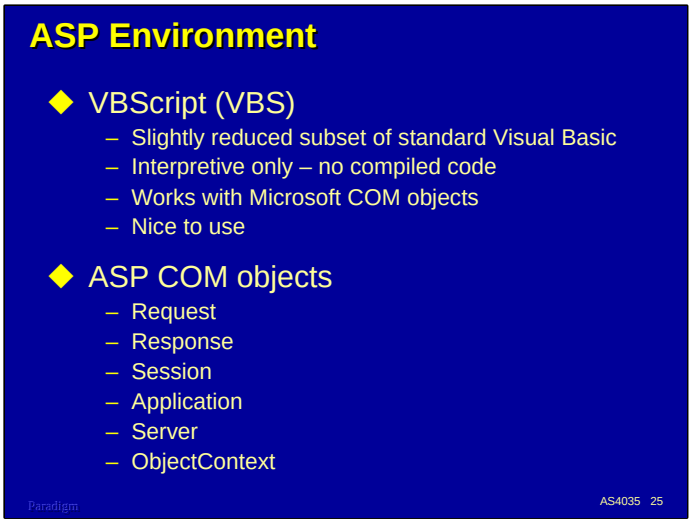
ISAPI is the second method for generating dynamic content, and the one most often used. This is an IIS-specific API that is much more efficient, offers a broader range of features than CGI, and also exploits a number of useful hooks in the IIS implementation. One of these hooks is the ability to write a "filter" process. Filters get to process messages as they come and go in IIS, and can examine or transform the messages in any way they wish. Microsoft supplies a number of filters with IIS. One of the best known is Active Server Pages (ASP).

ASP performs three main functions:

- It provides an environment for processing input messages and formatting output messages through a scripting language. ASP is not a scripting language itself – it merely provides interfaces for a compliant scripting language to use. The default language is VBScript, but other languages such as JavaScript and Python can be used as well.
- It examines incoming HTTP messages and encapsulates them in a set of COM objects. Scripts can interrogate properties of the messages and examine their contents.
- It provides other COM objects that allow scripts to interrogate and control the web server environment. These objects also provide the capability to create and control individual user sessions to compensate for the stateless nature of HTTP.

With ASP, browsers do not request a file of static HTML. Instead they reference a file containing the text of a script. When IIS encounters a URL with a ".asp" extension, it invokes the ASP filter, which causes the script to be interpreted. The script then generates the HTML to be sent to the browser.

This is an extremely powerful and flexible mechanism. The scripts can examine requests from browsers and generate customized output based on the contents of the requests. There are also facilities for the scripts to communicate with data bases and other systems, which as we will see is the basis for using ASP to web enable MCP applications.




Although ASP can use a number of different scripting languages, Visual Basic Scripting Edition, or VBScript, is the most common choice, at least partly because it comes with Windows.

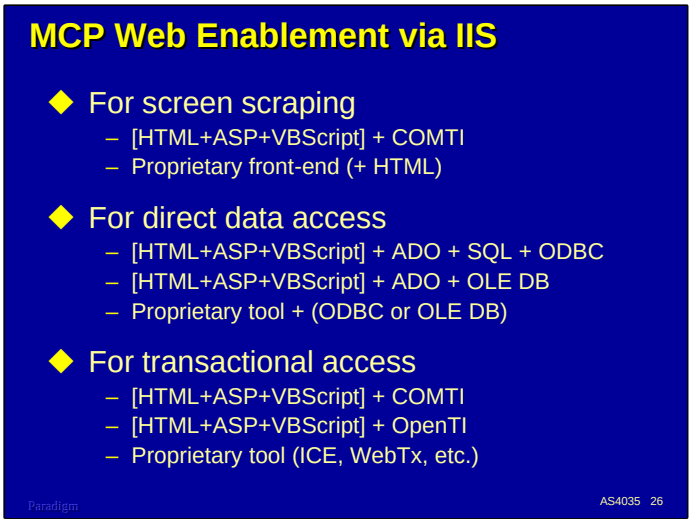
VBScript is a slightly reduced subset of the Visual Basic language. A few VB elements have been removed, but the bulk of the language has been retained. VBScript is compiled to an internal pseudo code, but is only interpreted. You cannot get compiled code or a standard .exe file out of VBScript.

The thing that makes VBScript really useful for ASP (and a number of other things) is its ability to use standard Microsoft COM objects. These are object-oriented packages you can simply plug in to a script to perform functions that VBScript itself does not. COM objects expose three types of interfaces: properties, methods, and events. There are a large number of COM objects available from Microsoft and third parties, including ones to access data bases, access the Windows file system, read and write files, and communicate with other systems.

VBScript is fairly easy to learn. If you already know VB or the VBA variant that comes with the Microsoft Office product, you can start coding in VBScript immediately.

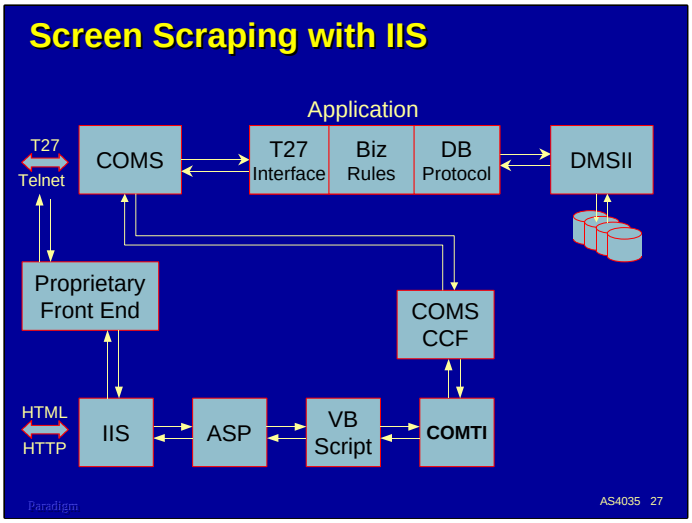
ASP provides several COM objects that allow scripts to process messages and interact with the web server environment:

- Request. This object gives a script access to an incoming HTTP message. Headers and fields of the message are pre-parsed and available through properties of the object, making it easy for scripts to decode both URL query strings and HTML form submissions.
- Response. This object gives a script access to the response message that will be sent back to the browser. Properties and methods allow the script to control buffering, set HTTP headers, and format the text of the message.
- Session. ASP generates a session for each end user based on a temporary HTTP cookie. This object gives a script access to the session properties and allows it to store data on a per-session basis. It can be used to compensate for the stateless nature of the web.
- Application. Similar to the Session object, this one gives a script access to properties of the ASP application and allows it to store global data for the application.
- Server. This object gives the script access to properties of the server system and some general utility methods for formatting HTML and HTTP strings.
- ObjectContext. This object is used with the Microsoft Transaction Server (MTS, now called Component Services) to implement recoverable, two-phase transactions.




In the following slides we will discuss how IIS can be used for the three MCP web enablement approaches: screen scraping, direct data access, and transactional access.

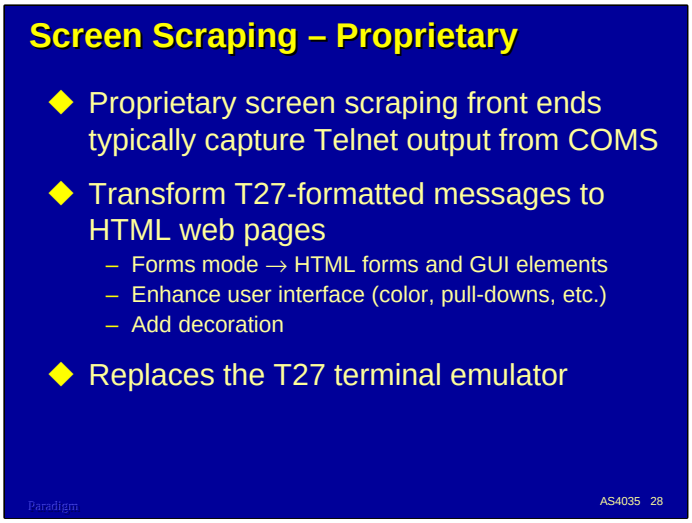
For each of these methods, you can write your own solution or you can use any number of proprietary tools. We won't talk about the proprietary tools that much in this presentation. Our focus will be on custom solutions using a base of HTML, ASP, and VBScript, plus one or more additional components that enable the ASP environment to communication with the MCP application environment.




The simplest approach to web enabling the MCP environment is screen scraping. You have two main choices: a proprietary solution or COMTI.

The proprietary solutions typically intercept the T27 data stream from MCP applications at some point in the communications path and transform it to and from HTML. A common solution is to implement a front end process on a Windows server, accept Telnet connections from the MCP host, transform messages between T27 and HTML formats, and exchange the HTML messages with IIS on the same server. IIS then communicates with the browsers using HTML over HTTP.

COMTI also intercepts T27 data streams, but handles them in an entirely different manner, as we will see. Instead of trapping the data stream on the network, COMTI communicates directly with COMS on the MCP host through the COMS Custom Connect Facility, CCF.

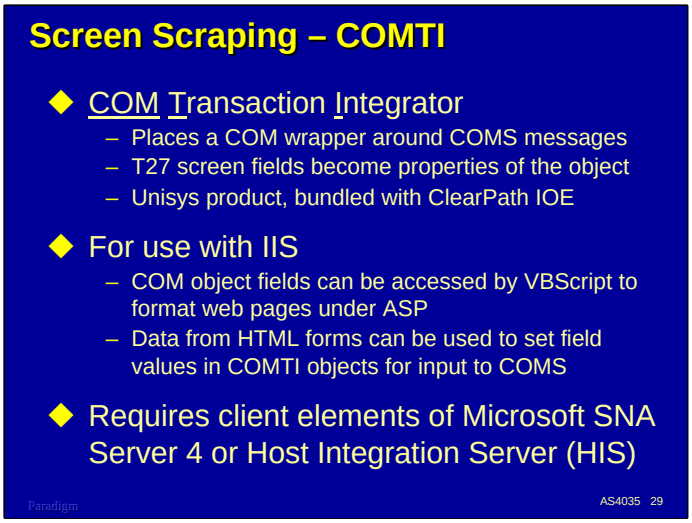



Using a proprietary screen scraping product is often the fastest way to put your existing green-screen interfaces on the Web. T27 forms mode translates nicely to HTML forms and GUI elements.

Most of the proprietary tools allow you to enhance the presentation by replacing text boxes with more intuitive controls such a pull-down lists, check boxes, radio buttons, and the like. You can also reorder fields and "decorate" the page to make it more attractive or easier to use. Some products allow you to make more radical changes to the interface, such as splitting T27 screens into multiple web pages or combining multiple screens into one page.

Screen scraping basically allows you to replace your T27 terminal emulator with a web browser. This has been a favored approach by terminal emulator vendors, and definitely has a place in the web enablement toolkit, especially early in your enablement effort.

Unisys bundles a product with the ClearPath IOE called Web Enabler for ClearPath MCP (formerly NX/WebStation for Java). This is basically an automatic screen scraping tool, but is implemented as a Java applet and communicates to the MCP environment through COMS CCF.

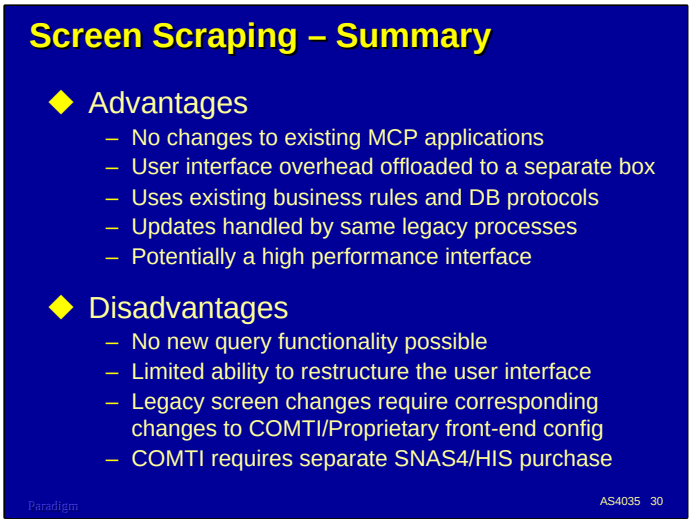



The COM Transaction Integrator, COMTI, wraps a COM object around the T27 message. This wrapper exposes the individual fields of the message as properties of the object, which in turn allows an ASP script to easily examine and set these fields. This is a Unisys product which runs in the Windows environment. It is bundled with the ClearPath IOE.

When used with IIS and ASP, the script is responsible to accessing the fields of the COM wrapper object and transforming the message to and from HTML. On input from the browser, the script typically uses the ASP Request object to extract fields from the HTTP message and format them for the COMTI object. On output from the MCP application, the script extracts fields from the COMTI object and uses them to format the HTML response.

It typically takes more effort to build a web interface using COMTI than with a proprietary tool, but you have a lot more flexibility and control over the result. In addition to being able to provide more sophisticated GUI support, you can split and combine T27 messages to partially restructure the end user interface.

One drawback of COMTI is that is requires some elements of Microsoft SNA Server 4, or the newer implementation of SNA Server, Host Integration Server (HIS). COMTI does not use SNA Server or HIS itself, only the client-side components of this product. Unfortunately, you must purchase the entire product in order to get these client-side components.

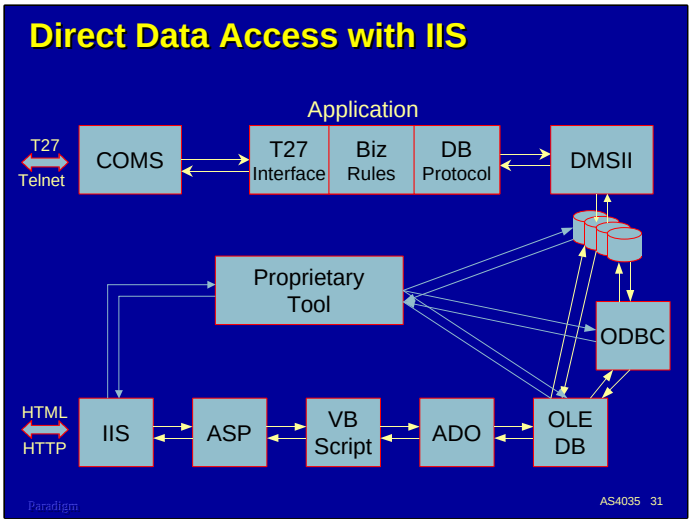



In summary, screen scraping through IIS and ASP has a number of advantages:

- No changes to the legacy MCP applications are necessary. This method simply intercepts existing message formats and transforms them to web formats.
- Essentially all of the extra overhead to perform the transformation and manage the web communications is offloaded to another system.
- The existing user interface, business rules, and data base protocol are unaffected. This means that update transactions are handled by the same code as before, and ongoing maintenance to business rules and the data base only has to be done in one place.
- Because no additional load is being placed on the MCP host, and the HTML transformations are generally straightforward, this approach has the potential to supply a high performance interface between the web and the MCP applications.

There are also some disadvantages to screen scraping:

- Since the existing MCP application is not affected, the web interface cannot provide any new application functionality. It can only rearrange the user interface.
- The ability to restructure the user interface is somewhat limited. Some screens can be split or combined, but ultimately the new interface has to be translated into the old one.
- As changes are made to the legacy screen formats and processing sequences, corresponding changes often need to be made to the web interface.
- You must have SNA Server or HIS to use COMTI.

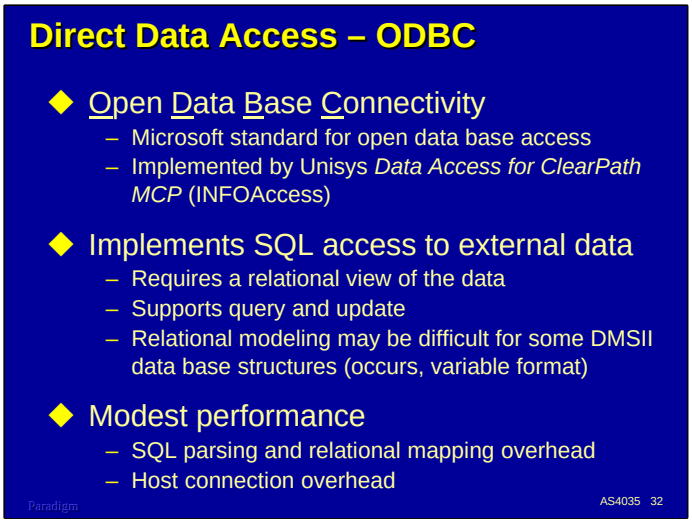



The second enablement method for IIS is direct data access. This is a very popular approach, and another easy way to get started on web enablement, especially if you are interested in query-only access to the legacy application's data.

With direct data access, you bypass the legacy application and connect directly to the underlying data base. This is typically a DMSII data base, but the same tools can also be used, in a somewhat more restricted manner, with ISAM and flat files.

There are two methods for connecting to DMSII data bases: ODBC and the newer OLE DB. In turn, these can be accessed from ASP scripts through a set of Microsoft COM objects collectively called ActiveX Data Objects, or ADO.

There are also proprietary tools that interface directly between IIS and ODBC or OLE DB, and potentially directly to DMSII itself. These tools are typically companions to or offshoots from reporting tools, such as Crystal Reports or the EZSPEC/DARGAL suite.



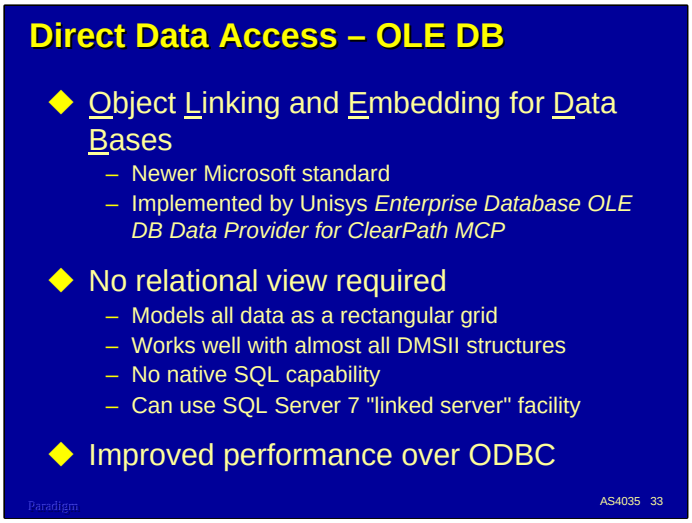

Open Data Base Connectivity, or ODBC, has been around for a while. It is a Microsoft standard for connecting applications to remote data bases. For ClearPath MCP systems, the current product is Data Access (formerly INFOAccess).

ODBC implements access to data bases using System Query Language, or SQL. SQL is a very powerful language for querying and updating data bases, but it is a relational data base tool, and requires a relational view of the data. Some DMSII structures, particularly occurring items, groups, and variable format data sets, do not fit the relational model. These must be mapped into relational terms, which can complicate the interface between the DMSII data base and the SQL application.

When used with ASP scripts, you access ODBC indirectly through ADO components, which we will discuss shortly. This indirect interface results in a lot of layers through which the query must travel, but it is fairly easy to program for in the scripting environment.

The Data Access product operates in two parts. The SQL query engine runs on a Windows server and communicates over TCP/IP to a data base agent that runs on the MCP host. The query engine is often installed on the Intel side of a ClearPath MCP system, to take advantage of the high-speed VLAN connection between the Windows and MCP environments.

ODBC, when used between ASP and DMSII data bases, currently offers only modest performance. There is significant overhead in parsing SQL queries and decomposing them to primitive DMSII operations, but the major factor limiting performance appears to be the overhead of establishing connections between the ASP environment and the Data Access environment. Connection pooling could help this, but is not currently supported by the Data Access product.




OLE DB is a newer Microsoft standard for connecting applications to remote data bases. For ClearPath MCP systems, this functionality is provided by the Enterprise Database OLE DB Data Provider product.

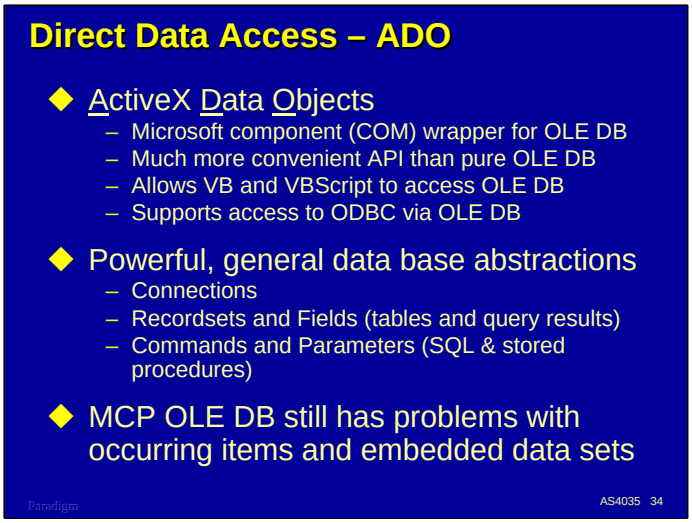
OLE DB is not dependent on the relational model. Instead, it works with data as a rectangular grid, or "rowset". Anything that can be transformed into a two-dimensional grid can be modeled by OLE DB. Each cell of the grid can itself be a rowset, which allows hierarchical data structures to be represented. As a result, OLE DB works well with almost all DMSII structures, including occurring items and variable format data sets.

The Unisys data provider exposes record structures, sets, and subsets. Using recordset objects in ADO, you can program DMSII data base access through OLE DB in much the same way you do in COBOL and Algol in the native MCP environment. It is relatively easy to search for records by key values, and to move sequentially forward and backward through data sets and indexes. These features can be used in lieu of SQL to construct queries and update transactions.

The OLE DB standard supports high-level query processors, such as SQL, but the Unisys data provider does not implement an SQL interface. You can still perform SQL queries using the Unisys provider, however, if you have Microsoft SQL Server 7 or 2000. SQL Server implements a facility called "linked server", which essentially allows SQL Server to act as an OLE DB client.

When you configure a linked server for the Unisys OLE DB provider, you can submit SQL queries directly to SQL Server. SQL Server then decomposes the query into more primitive data base operations and passes these to the Unisys provider, which in turn passes them to DMSII on the MCP host. Another advantage to linked servers is that you can access multiple data bases (potentially of different types and on different servers) within one SQL query.

OLE DB offers a substantial performance improvement over ODBC, especially in the area of connection establishment. In addition, the Unisys provider supports connection pooling.



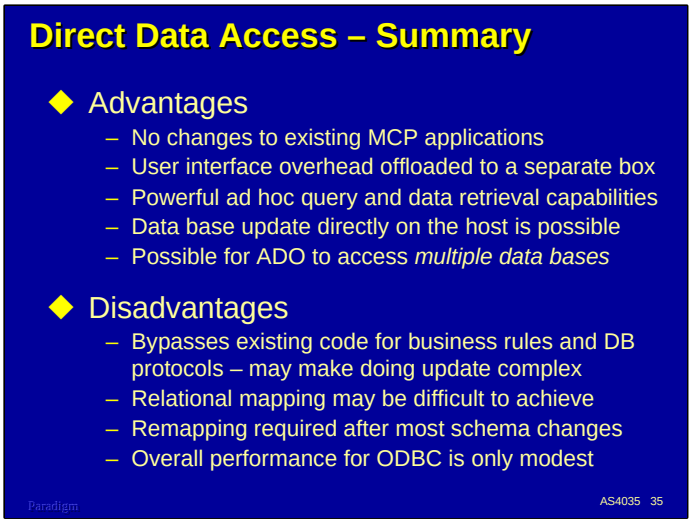

ODBC and OLE DB are powerful tools, but most scripting languages, including VBScript, cannot interface directly to them. Instead, ASP scripts interface to ODBC and OLE DB through ActiveX Data Objects. ADO is a more convenient API than the native OLE DB interface, and provides a number of useful abstractions and capabilities for accessing data bases.

ADO is actually a component-based front end to OLE DB. Microsoft has an OLE DB provider that will interface to ODBC, and this is how ASP scripts can access ODBC data sources.

ASP scripts use ADO to access data bases through a set of COM objects. The principal objects are:

- Connections. These define the remote data base and manage the communications path between the scripting environment and the data base host system.
- Recordsets. These are extremely powerful components that can be used either to expose a table (data set, set, or subset) to the scripting environment, or to hold the results of an SQL query. Recordsets give scripts access to the rows of the OLE DB data grid.
- Fields. These objects expose individual data items in the recordset rows and allow scripts to obtain or set the value of the corresponding data base items.
- Commands. These objects are used to manage SQL queries and (for data bases that support them) stored procedures.
- Parameters. These objects supply parameter values for queries and stored procedures.

Although the Unisys OLE DB provider for DMSII handles non-relational structures like occurring items and embedded data sets, the initial implementation does this in a manner that is incompatible with both ADO and SQL Server. At present, these structures must be mapped into a more relational form for them to be accessed through ADO or SQL Server. For occurring items, the Unisys provider allows you to "unroll" the occurrences as a set of elementary items.

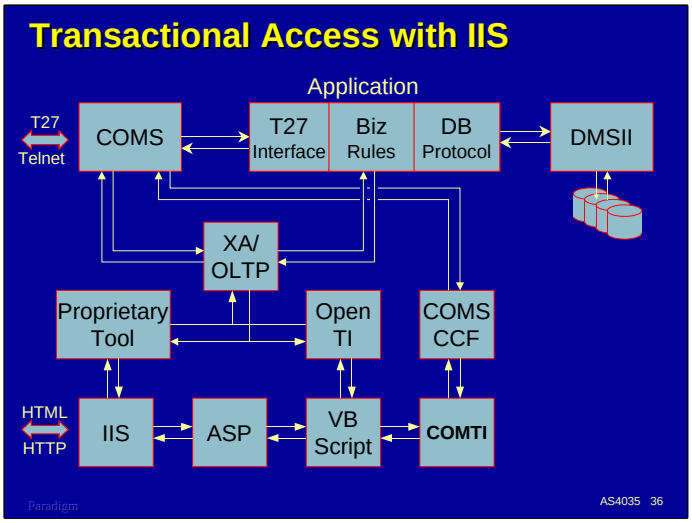



In summary, the direct data access approach through IIS and ASP has a number of notable advantages and disadvantages. Among the advantages are:

- No changes to the existing MCP applications are necessary, since this approach bypasses the application altogether.
- As with screen scraping, the user interface is offloaded to an external system.
- SQL provides very powerful query and data retrieval queries, especially for ad hoc situations.
- Both ODBC and OLE DB support update. These result in normal, audited DMSII transactions. Both interfaces allow you to control the transaction envelope (BEGIN- and END-TRANSACTION) so that multiple data base operations can be treated as an atomic unit.
- Using ADO, you can create multiple data base connections in one script. This allows you to implement queries and transactions that operate on multiple data bases, potentially of different types. It is relatively easy to have a single ASP script access, say, two DMSII data bases, an SQL Server data base, and an Oracle data base, all on separate servers. The linked server facility of SQL Server also supports this type of distributed data base access.

There are also a number of disadvantages to direct data access.

- Since this approach bypasses the existing MCP application, it cannot take advantage of business rules or data base protocols implemented under the MCP. This is generally not a problem for query, but can be a major obstacle for update transactions.
- If you will be using SQL for queries, achieving a sufficiently relational mapping of the DMSII data may be difficult.
- When changes are made to the DMSII schema, corresponding changes usually need to be made to the ODBC and OLE DB configurations. OLE DB does a better job of handling this than ODBC.
- Overall performance for ODBC is only modest. OLE DB performance, even when linked through SQL Server, is usually much better.



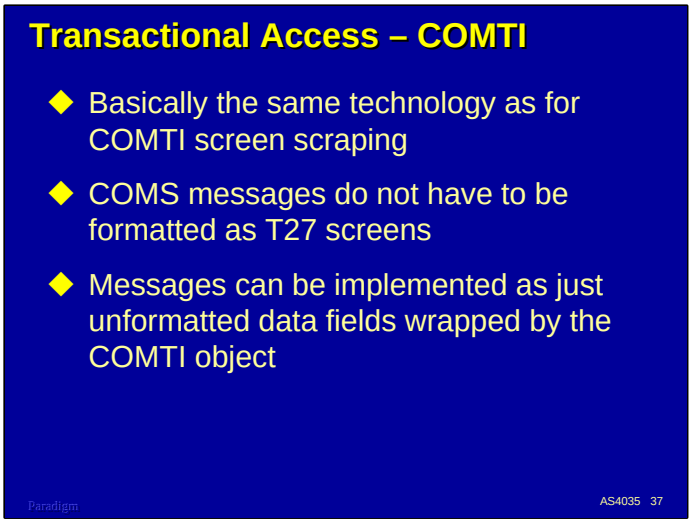

The third and final approach to consider with IIS and ASP is transactional access. In this case, we want to connect the web interface directly to the business rules and data base protocol of the legacy MCP application. There are a number of ways to do this.

The first is COMTI. This is basically the same approach as for screen scraping.

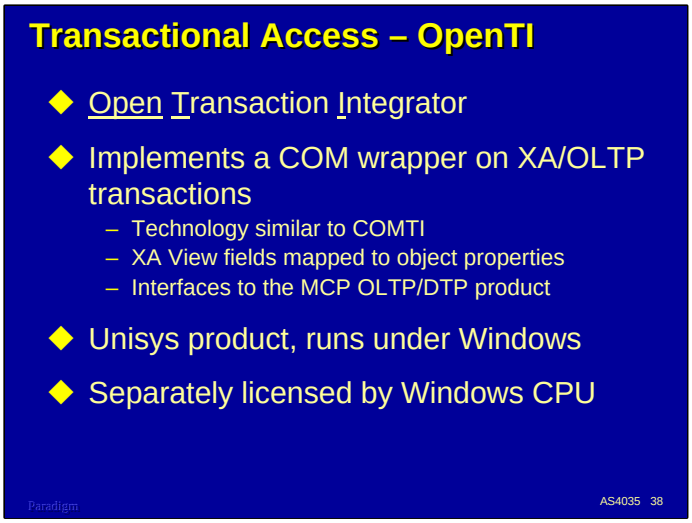
Another method uses a Unisys product called the Open Transaction Integrator, or OpenTI. This is COM object for distributed processing (XA/OLTP) and runs in the Windows environment. It interfaces to the XA/OLTP product that runs in the MCP environment. OpenTI can be used in two ways:

- The OLTP product has an interface to COMS, and can submit transactions from OpenTI for routing to OLTP-compliant COMS programs.
- There is also a direct API for OLTP, which can be used to bypass the user interface in the legacy application and connect directly with the business rules.

There are also proprietary tools (some from Unisys) which use the XA interface and can connect to the OLTP product in the MCP environment. The Web Transaction Enabler (WebTx) and Internet Commerce Enabler (formerly Cool ICE) are examples of this approach.



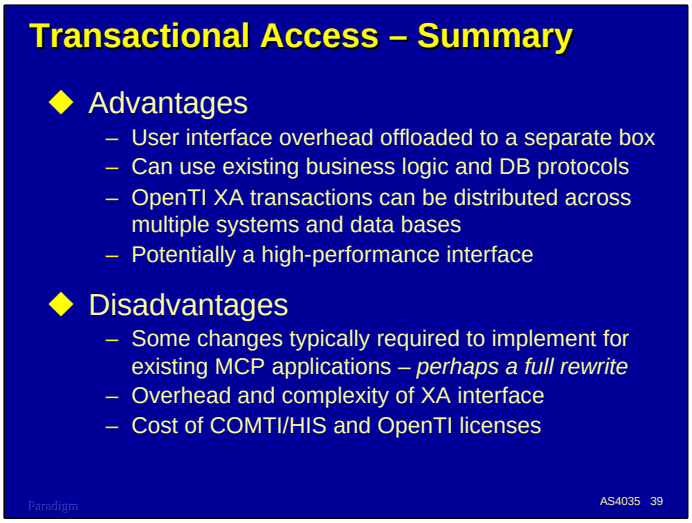

Using COMTI for transactional access is basically the same approach as using it for screen scraping. The difference is only in how the COMS messages are formatted. Instead of formatting T27 screens, the COMS messages can simply consist of unformatted data fields, much as they would be for TP-to-TP messages. The "user interface" in the MCP application would then be just a thin layer that passed the data fields to the business rule interface. The COMTI wrapper would still provide access to the data fields for the ASP scripting environment.




The Open Transaction Integrator is a Unisys product that runs in the Windows MTS/COM+ (now called Component Services) environment. Its purpose is to provide a COM wrapper for XA/OLTP transactions and interface to external XA-compliant systems, such as the OLTP product under the MCP. In this respect, it a similar technology to COMTI.

The COM wrapper exposes fields of the transaction. This allows ASP scripts to extract data items for formatting in outgoing HTTP messages, and insert fields from incoming HTTP messages into the transaction object.

OpenTI is separately licensed and priced by the number of CPUs on the Windows server.

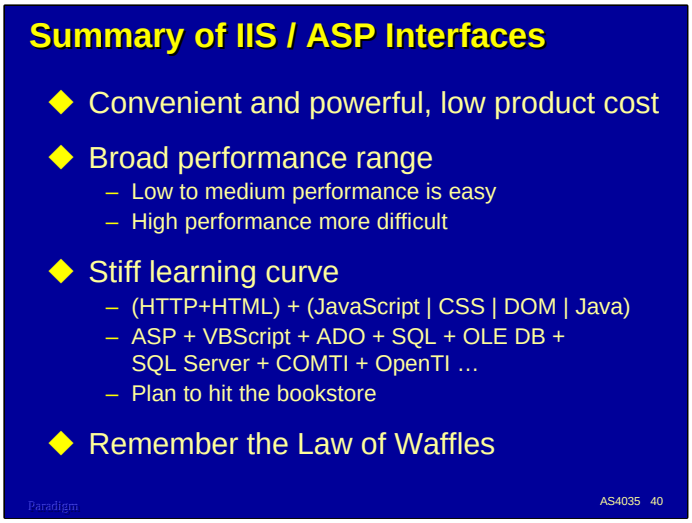



As with each of the other approaches, transactional access for IIS and ASP has its set of advantages and disadvantages. Among the advantages are:

- Like all IIS/ASP approaches, this one offloads the extra overhead of managing the user interface and web communications to a separate server.
- Transactional access can potentially use the same business rules and data base protocols as the existing MCP application. This makes this approach a prime candidate for use with update transactions.
- Because it uses the XA standard, OpenTI can be used to implement a web interface for transactions that are distributed across multiple systems.
- Transactional access has the potential to support a high-performance interface to the legacy application.

Among the disadvantages are:

- Many legacy MCP-based applications do not have a clean interface to their business rules and data base protocol. This usually means that some changes must be made to the application to isolate and expose that interface. In the worst case, this could involve a major reengineering effort or a total rewrite.
- Because the XA interface must be concerned with distributed processing and two-phase commit across multiple systems, it is not as efficient as more direct interfaces. Also, because of its general, distributed nature, XA has a more complex programming model.
- Finally, the HIS/SNA Server components required by COMTI, and the OpenTI software, must be separately licensed. The cost of these may be an issue for some organizations.




In summary, the web enablement approaches through IIS and ASP offer a number of advantages. They support convenient and powerful mechanisms to interface to the MCP environment in a number of ways. Most of the tools are bundled with either Windows NT/2000 server or with the ClearPath MCP IOE.

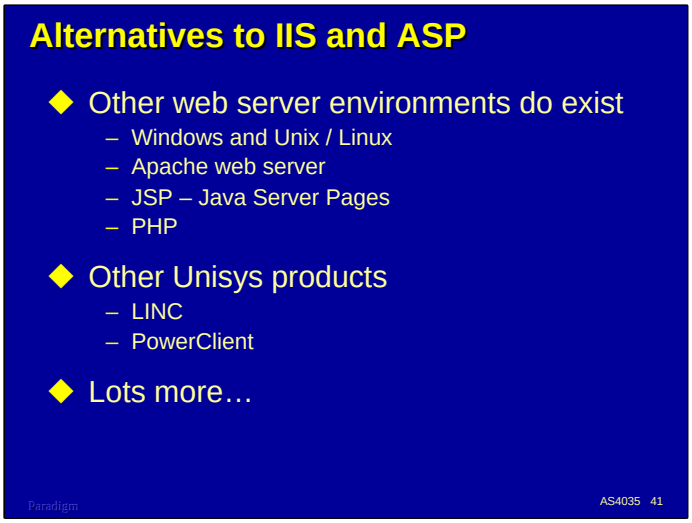
The potential performance of web enabling applications through IIS and ASP covers a broad range. Low to medium performance is relatively easy to achieve. High performance interfaces usually require using more advanced techniques, and just as with mainframe applications, can be difficult and expensive to build.

On the down side, there is a very stiff learning curve for all this new technology. If you are just getting started with web development,

- You need to know a little about HTTP and a lot about HTML.
- You will probably need to delve into some combination of JavaScript, Cascading Style Sheets, the Document Object Model, and perhaps Java.
- You will need to learn the ASP components and VBScript.
- You will almost certainly need to learn at least some of ADO, and probably SQL and the capabilities of OLE DB. If you plan to use SQL with the Unisys OLE DB provider, you will need to learn a little about SQL Server administration as well.
- Finally, if you plan to use COMTI or OpenTI, you will need to learn their object models and configuration/administration tools.

This is a lot to take on, and it will take some time. Plan to spend both money on books and a lot of time reading them.

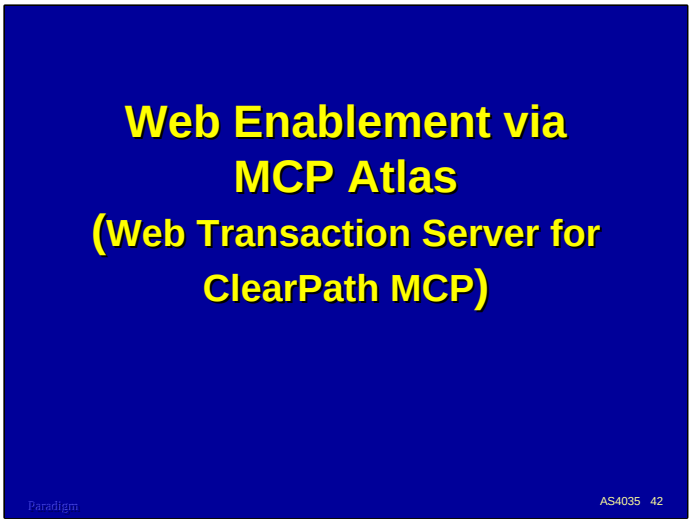
Also plan to invest some time in building and exercising some relatively small, simple projects you can play around with. Remember the **Law of Waffles** – the first one out of the iron usually isn't that good and probably should be thrown away. You'll learn a lot making that first waffle, though.



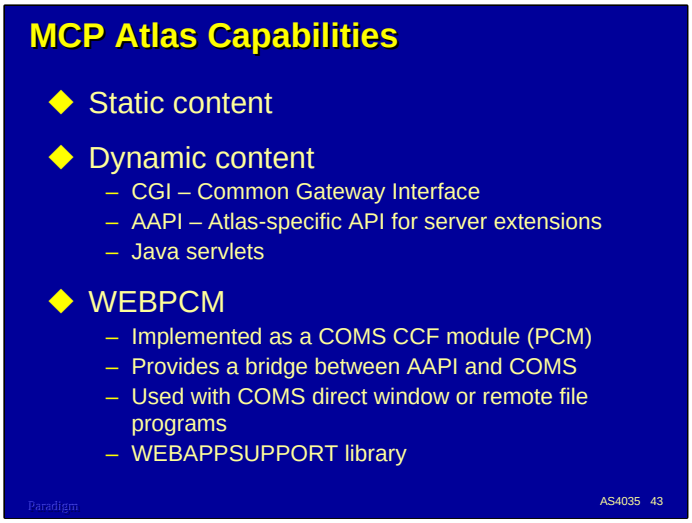

While this discussion so far has concentrated on using Windows, IIS, and ASP, these are not the only technologies that can be applied for web enabling through external servers. Unix and Linux are good platforms for hosting web interfaces, and there are any number of other web servers, like Apache, and dynamic content generators, like Java Server Pages and PHP, which can be considered.

There are also other Unisys products that provide web interfaces to MCP-based applications. Of particular relevance here are LINC (now the Enterprise Application Environment) and PowerClient (now the Graphical Interface Workbench).

There are, of course, lots more alternatives out there.




Having considered web enablement by means of an external Windows system, we now turn to the approaches that can be used for web enabling a legacy application within the MCP environment. The primary tool for these approaches is the Web Transaction Server for ClearPath MCP, also known as Atlas.




Like IIS, Atlas offers facilities for serving up both static and dynamic web page content.

Static content is served in much the same was as it is by IIS. Files are simply copied from the MCP's file system and transmitted to the requesting browsers.

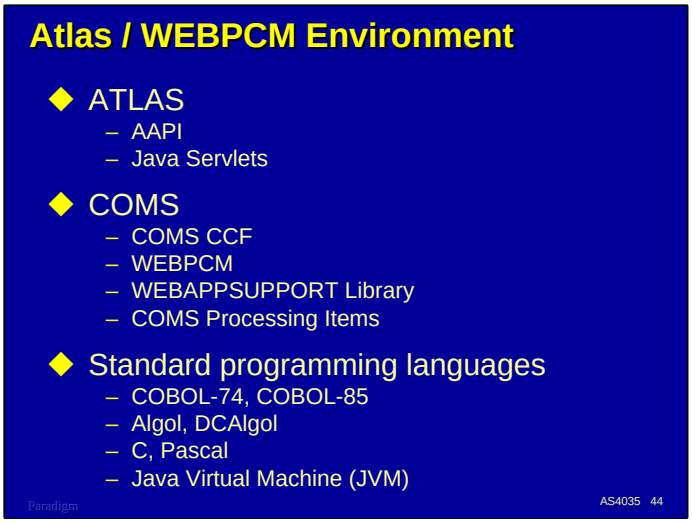
Atlas can provide dynamic content in several ways.

- The CGI API is similar to the like-named interface in IIS. It is intended to provide compatibility with programs written for the older NX/WebServer product, which is now obsolete.
- AAPI is an Atlas-specific API. It is designed to be a very high performance interface. Unfortunately, it is implemented using Connection Libraries, which means that you must program in Algol or NEWP in order to use it.
- Atlas also supports Java servlets as part of the new Virtual Java Machine for ClearPath MCP product. Servlets are the server-side complement to Java applets and provide a convenient means for browser-based applets to communicate with counterparts on a web server. We won't discuss servlets in any detail here, but they are a very promising technology for MCP web enablement.

The nature of AAPI's reliance on Connection Library interfaces initially made Atlas extremely difficult to use with COBOL applications. Relief from this situation arrived in HMP 5.0 with the release of the WEBPCM.

WEBPCM is a module of COMS Custom Connect Facility. It provides a bridge between the Atlas AAPI and COMS. HTTP messages arriving from browsers are routed by Atlas to the WEBPCM, which in turn are routed through CCF's CUCI interface to COMS. COMS then routes the messages in the normal manner to a direct or remote file program for a window. This provides an easy means for MCP applications written in virtually any language to become web enabled.

HTTP messages look nothing like T27 messages, however, and are quite difficult to parse and compose in COBOL. It's not all that easy in Algol, either. Therefore, the WEBPCM supplies a utility library, WEBAPPSUPPORT. Procedures in this library perform much the same role as the properties of ASP Request and Response objects, allowing COMS programs to conveniently parse HTTP messages and generate HTML responses.

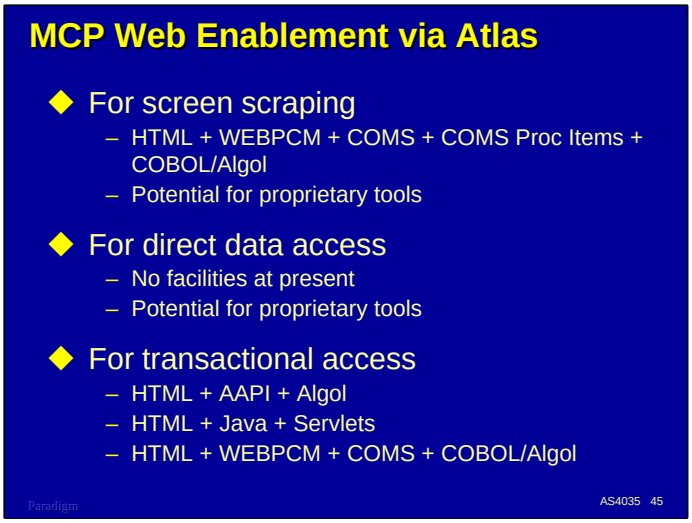



The minimum you need to web-enable an MCP application is the Atlas web server. Since HTTP operates over TCP/IP, you also need to have the TCP/IP network provider configured and running.

You could get by with just Atlas and its AAPI, but that would restrict you to programming in Algol or NEWP (or building a bridge to applications in other languages using Algol or NEWP). You can also build servlets using an external Java development environment (such as Borland's JBuilder) and running the resulting bytecode files under the JVM.

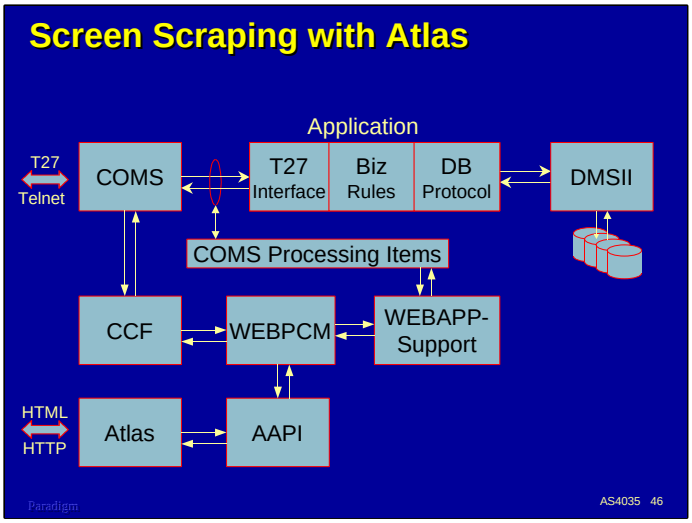
Most MCP-based transaction programs are written for COMS in one of the COBOL dialects, so you should consider using the WEBPCM and implementing a web interface with traditional tools. The WEBAPPSUPPORT library provides a number of very useful facilities for processing HTTP messages. You can also use COMS processing items to transform incoming HTTP messages to traditional COMS/COBOL formats and outgoing messages to HTML.

Since the WEBPCM interfaces to standard COMS APIs, you can program your MCP web interface in any language that supports either COMS direct windows or remote files. Algol, COBOL, C, and Pascal are obvious choices for consideration, but in principle you could even web enable a FORTRAN or RPG program.




The options for web enabling applications within the MCP environment are a little more limited than for IIS, but there is still plenty of capability to exploit.

- Screen scraping approaches can be implemented using COMS processing items. At present there are no packaged solutions for screen scraping within the MCP.
- There are currently no facilities available for direct data access, although in principle this could be done with DMINTERPRETER or the Application Data Access component of the Data Access ODBC product. We won't discuss this option for Atlas any further.
- For transactional access, you can use AAPI with Algol, Java Servlets, or the WEBPCM with COMS and any of the programming languages which can interface to COMS.



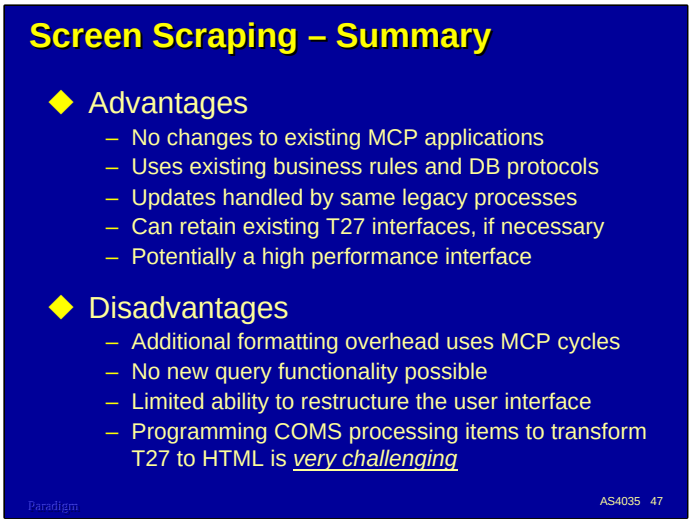

The first enablement approach to consider with Atlas is screen scraping. Recall that screen scraping works by intercepting the traditional message data stream at some point and transforming it to HTTP/HTML.

Since the Atlas web server is running inside the MCP environment, we can't intercept the data stream on the network. However, COMS gives us a way to do this through Processing Items.

Processing items are procedures in a program library. They have access to COMS messages as they enter and leave COMS programs and can transform and modify both the message text and parts of the control information in the header structure. You can have separate input and output processing items, and you can define lists of processing items that operate on a message in series.

Processing items have a parameter sequence that is defined by COMS. They must be written in Algol, although you could write just a procedural shell and use either the Binder or program libraries to connect to code in other languages. They are attached to the COMS agenda for direct window programs.

To implement screen scraping, you normally need to connect to Atlas through the WEBPCM. This converts the HTTP message for Atlas into a standard COMS message. You would then need to write two sets of processing items, one that would transform input messages from the WEBPCM into a traditional format, and one that would transform output messages from the application to the format WEBPCM sends to Atlas. Both sets of processing items would need to call on the WEBAPPSUPPORT library to assist them in parsing input and formatting output for WEBPCM.

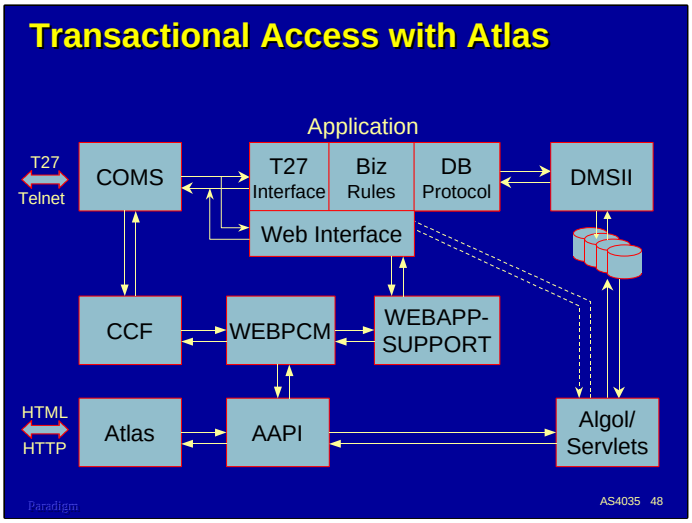



Screen scraping with Atlas has all the advantages it did with IIS on an external server.

- No changes are required to the existing MCP applications. COMS programs are unaware that processing items are intercepting their messages. If the application is already using processing items to pre- or post-process messages (perhaps for T27 forms), the items for the web interface can simply be appended to the existing processing item list.
- Since the application is not altered, all the existing business rules and data base protocols remain in place.
- Update transactions are not a problem, since they operate through the same legacy business rules and data base protocols.
- COMS processing items can be applied to messages based on a COMS entity called the Device Type (DT). This allows you to continue to use the application with legacy terminals and emulators in parallel with the web interface.
- Processing items are relatively efficient, so this approach has the potential to produce a very high performance interface.

Screen scraping with Atlas shares some of the same disadvantages as it does with IIS, plus a significant new one.

- The additional overhead for dealing with HTML formatting and HTTP is all done within the MCP environment, so it will consume additional MCP resources, especially CPU time.
- Since nothing about the application is changing except the user interface, no new query functionality is possible.
- Processing items can reformat messages in any way desired, and have some ability to split or combine messages, but ultimately they must generate the same traditional data stream as before. This limits the amount of restructuring you can do for the user interface.
- The biggest disadvantage is that using processing items for this type of message transformation is not very easy. Building a capability for a specific set of message formats is a significant programming task. Trying to solve the problem for the general case is very challenging. Because of the technical difficulty of this approach, and the need for Algol programming, it may be beyond the skill level for some organizations. This is an area where an enterprising vendor could potentially supply a general-purpose solution.




The alternative to screen scraping with Atlas is transactional access. There are two main options to consider.

In the first case, you could use AAPI and write an entirely new interface to the data base. This could be done using either Algol directly with AAPI, or with Java servlets. On the diagram, it looks like direct data access, but effectively what you are doing is re-implementing the on-line portion of the application from scratch. This could be a big job.

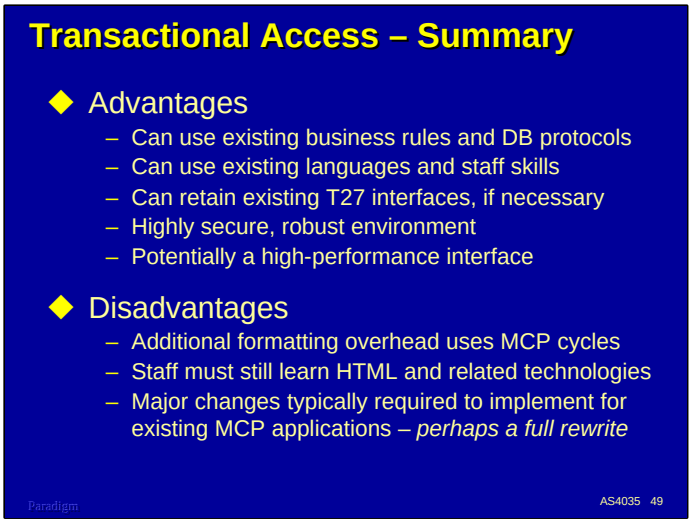
As a variation on this first case, you might be able to write a web front end to your application that could interface either to your existing user interface or the business rules of the application. By the time you do this, however, you will probably have implemented a form of screen scraping, or a solution like the the following one.

In the second case, you probably want to retain as much of the existing application as possible, and probably need to keep the existing interface to legacy terminals. You also probably want to leverage as much of the staff's existing skills a possible. The solution is to simply add a new user interface to your application that operates in parallel with the old one.

The easiest way to do this is with the WEBPCM. You can build this new interface in COBOL (or whatever language your application is written in), and continue to use COMS as the gateway to the outside world.

Exactly how this new interface would fit in with your existing application code depends a lot on how the existing application is structured. If there are fairly clean interfaces between the legacy user interface and the business rule/data base protocol components, adding another interface for the web is pretty easy. If the legacy user interface is intimately intertwined with the rest of the application, plugging in a new interface could be very difficult. In the extreme, it may require a major restructuring of the code or a complete rewrite. We will discuss this issue further in just a minute.

Of course, if you are planning a new application, or major changes to an existing one, you can take the need for a web interface (or even multiple user interfaces) into account in the design phase. Building in this capability from the beginning is usually a lot easier than adding it after the fact.

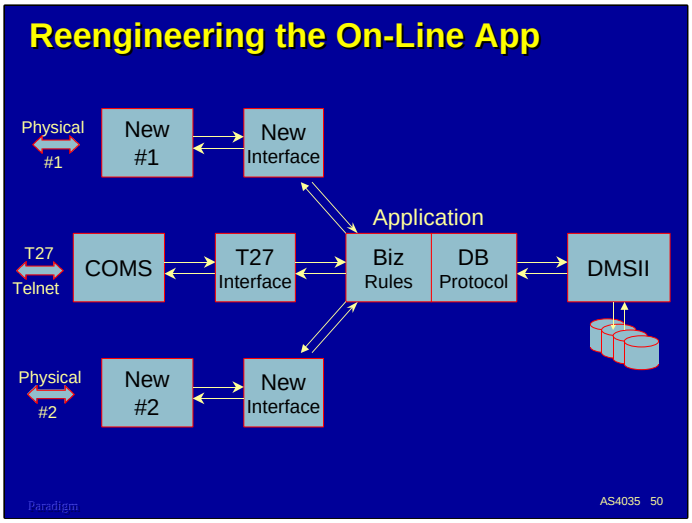



As with all the approaches, using Atlas for transactional access has its advantages and disadvantages. On the plus side,

- There is at least the potential for retaining the existing business rules and data base protocols for the application, especially if you use the WEBPCM. This is a particularly important consideration if you need to process update transactions from the web.
- You can continue to use the same programming language and COMS facilities as before. This leverages existing staff skills, and is a big advantage.
- It is usually easy to implement the web interface so that it can run in parallel with the existing legacy user interface for T27 terminals and emulators.
- The ClearPath as a whole provides a highly secure, robust, and reliable environment. Designing for the web carries with it its own set of security issues, but at least the underlying platform is as secure and reliable as it has always been.
- Atlas and WEBPCM provide fairly efficient gateways to the web, so this approach has the potential to support a high performance interface for the application.

There are also some disadvantages to transactional access through Atlas.

- As with screen scraping, all of the additional overhead necessary to deal with HTTP and HTML formatting occurs in the MCP environment, so it consumes additional MCP resources.
- Even though you can leverage existing staff skills, they still must learn quite a bit about HTTP, HTML, and perhaps some of the related technologies – CSS, JavaScript, etc. This is an investment you have to make regardless of the approach you take.
- Not many MCP based applications are structured so you can just plug in a new user interface. In most cases, at least some restructuring of the application will need to take place. If the internal divisions between user interface and the business rules is completely blurred, this may require major reengineering or a complete rewrite.

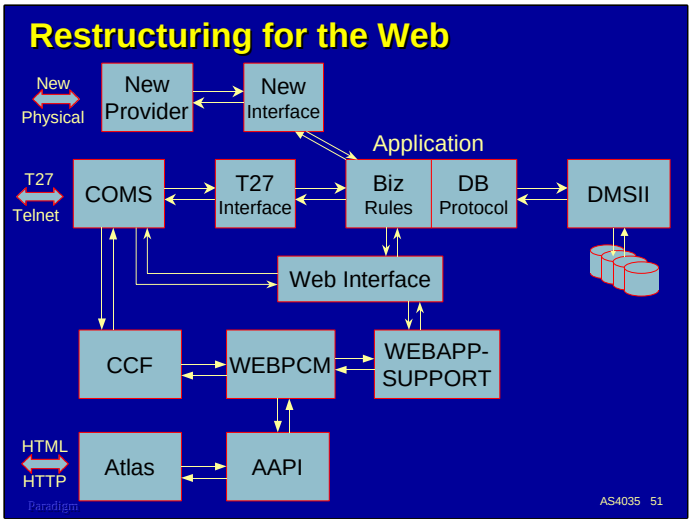



Remember this picture? It illustrates an idealized application architecture where the user interface is isolated from the rest of the application (the internal "essence" of the application) so that multiple user interfaces can be implemented easily.

Building this capability into a new application is not all that hard, and has a number of other benefits. However, reengineering an existing application – especially an older one – so it supports this kind of flexibility may not be particularly easy.

On the other hand, if you are adding a web interface to this application, you are either going to have to isolate the internal transactions so multiple interfaces can use them, or replicate the code for each interface. Isolation carries a one-time cost. Replication carries a maintenance cost that you will pay for the rest of the life of the application. This makes isolating the user interface(s) from the business rules and data base protocol a very important issue to consider.

Another thing to keep in mind is that, as technology advances, the need for additional external interfaces tends to run in the sequence "one, two, many…" If e-commerce is going to become a reality, we are going to have to learn how to build new interfaces onto our existing applications, and design new applications so they can easily adapt to the need for new interfaces as those needs arise. The only other option is to rewrite everything from scratch, and we already know that's not an option.

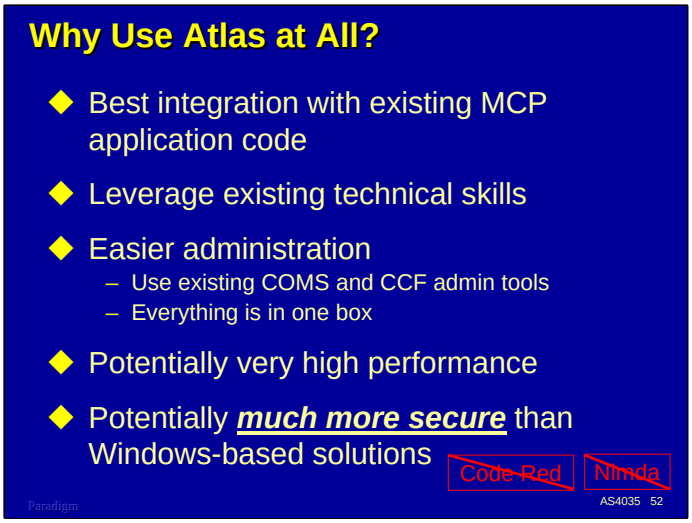



If you are serious about web enabling your applications, you should start thinking about how the applications should be reengineered to support that. Screen scraping, COMTI, and direct data access are really nice approaches, especially if you are just getting started, but they are all essentially ways to avoid the main problem.

The key to opening up legacy applications for new technologies is to isolate the user interface from the core of the application – the business rules and data base protocol. Once you do that, it's relatively easy to add a web interface, whether it's for the WEBPCM or another technology. It's equally easy to add other interfaces as they are needed – for XML, or EDI, or for things that haven't even been thought of yet.

Restructuring an application this way carries other benefits, too. It helps factor out common processes and identifies areas for code reuse. A well-factored application may actually shrink in size as common code is consolidated and implemented in one place. Restructuring can simplify on-going maintenance and enhancement. It can also allow you to better allocate your staff's skill levels to different parts of the application. Writing a recoverable DMSII transaction is usually harder than writing input validation or output formatting, and you can assign your people accordingly.

Restructuring an application can be a big job, and can have a significant up-front cost. The benefits will only accrue over time. Therefore, it's an investment for the future. Unless you are planning to completely toss your legacy code and go in another direction (which is usually even more costly), it's the best investment you can make for the future of your application, whether you are currently planning to enable it for the web or not.




Using Atlas to web enable your application within the MCP environment probably looks like a daunting prospect. IIS/ASP solutions like COMTI can be applied externally without restructuring the application or dealing with complex processing item design. So why even consider implementing web enablement within the MCP environment at all?

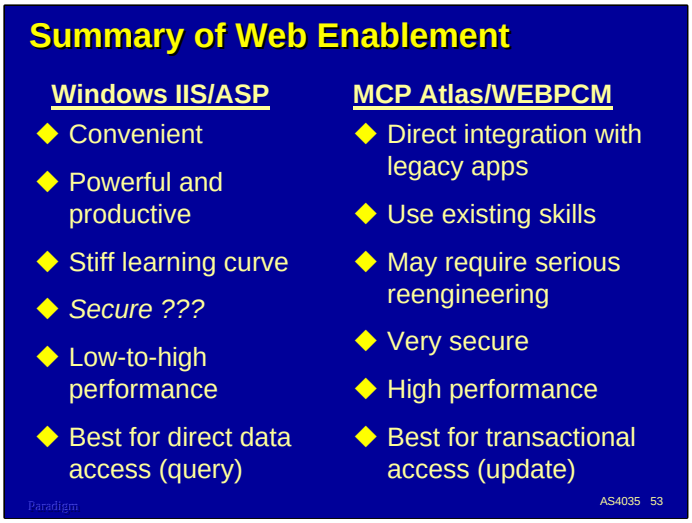
That's a very good question, and one to which there is not a single, clear answer. The external solutions are good ones, and in some cases may be the best ones. Most require learning a lot of new technologies, however, and all of them increase the complexity of ongoing maintenance and enhancement to the application.

By keeping the web interface inside the MCP environment you can achieve the best integration with the existing application code. *This is particularly important for update transactions*, which are heavily dependent on business rules and data base protocols that have been refined over the years, and which may not be accurately defined anywhere else but in the legacy code. As we have discussed, this also allows you to leverage your staff's existing skills. Administration of the new interface is typically easier, since in part you use existing COMS and CCF facilities, and everything is together in one box.

Although implementing the web interface within the MCP environment consumes extra resources, the interfaces for the external solutions place a load on the MCP as well. The tight integration with the application possible within the MCP environment can help compensate for any additional overhead due to HTTP/HTML processing. Implementing the web interface within the MCP has the potential for supporting very high performance applications.

Finally, if security and reliability are major concerns (and they almost always are), the MCP environment still holds a significant advantage over Windows (and to a lesser degree, over Unix and Linux). We have had some nasty experiences this year with viruses and worms, particularly Code Red and Nimda, that are targeted against Windows and IIS. This is a situation that is going to continue to get worse for some time to come.

I won't claim that the MCP can't be hacked, but it's at least an order of magnitude more difficult than for other types of systems. Security can't be based solely on the presumed correctness of the code – multiple levels of protection need to be in place and operate independently. Code Red actually caused a buffer overflow in Atlas, just as it did in IIS. Instead of overwriting some area of memory, though, the bounds checking built into the E-mode architecture trapped the overflow, and Atlas was fault-DSed. Windows and Unix will *never* have this level of protection.



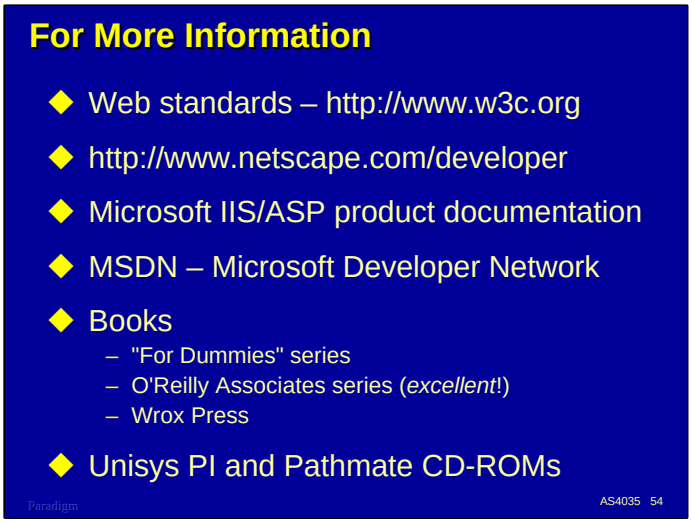

In summary, web enablement externally using IIS and ASP offers a convenient and powerful facility. It is easy to get started with this set of approaches, and in some cases may be the best in the long run. However, ASP-based solutions require learning a lot of new stuff, and takes time. Windows and IIS have some security and reliability issues, and will continue to reveal new ones for the foreseeable future. They offer a broad range of performance levels. ODBC, in particular, is not suitable for high-performance applications.

If there is one area that the IIS/ASP excels in, it is direct data access for query-only applications. If that is what you need, this is probably the best approach.

Web enablement internally using Atlas and the WEBPCM allows you to integrate the new interface directly with your legacy code and maximize the use of existing technical skills. To achieve this, however, you may need to perform some serious reengineering within your application to isolate the internal transaction processing from the old user interface. This can be a big job. The MCP offers a very secure environment, and web enablement within this environment potentially offers the highest overall performance.

If you need to process update transactions from the web, Atlas and the WEBPCM are probably the best long-term approach to consider.

Regardless of the approach you choose, it's very important that you first build some simple prototypes and play with the technology until you gain an understanding of it. You can't win against the Law of Waffles, but you can learn a lot the first time around that will pay dividends when you attempt a "real" project.




Here are some resources for more information about web enablement.

- You can get copies of all the web standards, HTTP, HTML, CSS, DOM, etc., from the World Wide Web Consortium's web site at http://www.w3c.com. If you are interested in following the development of web standards, this is also a good place to go.
- For reference information on JavaScript, go to Netscape's developer site at http://www.netscape.com/developer. Copies of the JavaScript reference can be downloaded in PDF. There are also numerous links to other information resources.
- For more information on IIS, ASP, VBScript, and ADO, you can go to the Microsoft product documentation. This is normally loaded to the Windows server when you install IIS and is accessible from the Start menu.
- You might also want to consider a subscription to MSDN, the Microsoft Developer Network. A basic annual subscription costs about $100, for which you get quarterly issues of a set of CD-ROMs with complete product documentation, white papers, case studies, examples, etc.
- Plan to visit a good bookstore and spend a fair amount of both time and money there.
  - If you are just getting involved with web development, the "for Dummies" series is not a bad place to start.
  - Everything published by O'Reilly Associates that I've encountered has been excellent. There are two major series, the "in a Nutshell" books, which give you a good introduction and grounding in a subject, and the "Definitive Guide" series, which are more comprehensive reference works. *Highly recommended*.
  - Wrox Press is also a good source. Their book on ADO 2.5 is especially worthwhile.
- For web development within the MCP environment and using the COM-based enablers, the Product Information CD-ROM contains most of what you will need. Pathmate (now called the Integration Expert) is also a good resource. On the PI CD-ROM, you should look particularly at:
  - *Web Transaction Server for ClearPath MCP Administration and Programming Guide*
  - *Custom Connect Facility Administration and Programming Guide*
  - *COMTI for ClearPath MCP*
  - *Enterprise Database OLE DB Data Provider for ClearPath MCP Installation and Operations Guide*
  - *Java Servlet API for ClearPath MCP Programming Guide*
  - Documentation for OpenTI is contained on the OpenTI release media.