Transcoding: Extending e-business to new environments

by K. H. Britton
R. Case
A. Citron
R. Floyd
Y. Li
C. Seekamp
B. Topol
K. Tracey

The promise of e-business is coming true: both businesses and individuals are using the Web to buy products and services. Both want to extend the reach of e-business to new environments. Customers want to check accounts, access information, and make purchases with their cellular phones, pagers, and personal digital assistants (PDAs). Banks, airlines, and retailers are competing to provide the most ubiquitous, convenient service for their customers. Web applications designed to take advantage of the rich rendering capabilities of advanced desktop browsers on large displays do not generally render effectively on the small screens available on phones and PDAs. Some devices have little or no graphics capability, or they require different markup languages, such as Wireless Markup Language (WML), for text presentation. Transcoding is technology for adapting content to match constraints and preferences associated with specific environments. This paper compares and contrasts different approaches to content adaptation, including authoring different versions to accommodate different environments, using application server technology such as JavaServer pages™ (JSP™) to create multiple versions of dynamic applications, and dynamically transcoding information generated by a single application. For dynamic transcoding, the paper describes several different transcoding methodologies employed by the IBM WebSphere™ Transcoding Publisher product, including HyperText Markup Language (HTML) simplification, Extensible Markup Language stylesheet selection and application, HTML conversion to WML, WML deck fragmentation, and image transcoding. The paper discusses how to decide whether transcoding should be performed at the content source or in a network intermediary. It also describes a means of identifying the device and network characteristics associated with a request and using that information to decide how to transcode the response. Finally, the paper discusses the need for new networking benchmarks to characterize the server load and performance characteristics for dynamic transcoding.

The phenomenon of the Internet has caused businesses to rethink their business models, customer relationships, and internal processes. Technology advances have created new opportunities to reach employees and customers, wherever they are, with information that is tailored to their needs and preferences. This information is often already stored and used in the business, but the delivery system must be re-engineered to exploit the new technology and tailor the content so it is more usable. A common problem with which this re-engineering must deal is that data or information is stored in some form on some system, when it is needed somewhere else in a different form.

The problem of adapting or customizing existing content to new applications and deliveries is not new. The user interaction model advanced by the Internet browser, along with portable and interoperable features of new technologies such as the Java** language[1] and Extensible Markup Language (XML)[2] have created a new opportunity to address this problem with some common techniques. In contrast, the rapid appearance of wireless and other new networks with widely varying characteristics and the preponderance of new devices with a wide variety of capabilities creates new constraints on the solution. Devices that are designed to be easily carried and used in the field trade off some capabilities to gain this portability. To be easily carried, they must be light

and small. This requirement limits the types of user interfaces they can support. Large screens and full keyboards are impossible; a small screen and telephone keypad are more realistic, although some devices may have only a voice interface. To run on battery power for useful periods of time, power consumption must be carefully managed, forcing the use of designs with little storage and processing capability. To be connected from anywhere requires wireless or intermittent wired connections, which limit the types of interactions and bandwidth available for accessing content.[3]

All of these constraints create difficult challenges in designing a useful system for delivering content to a wide array of devices. However, if such an information delivery system can be created quickly and cost-effectively and if it integrates with existing information systems, the value to customers can be immense. Transcoding, or adapting content from one form to another, is a key part of meeting these requirements for rapid, inexpensive deployment of new ways to access existing content.

The media signal processing industry first used the term "transcoding" to refer to the task of converting a signal, say a television program, from one format to another (for example, converting the NTSC (or National Television System Committee) standard, used in America and Japan, to the PAL (or Phase Alternating Line) standard, used in much of the rest of the world), while preserving the content of the program. Although the term has lately been used to mean many different things, we use the term transcoding to refer to the tasks of summarizing or filtering (which modify content without changing its representation) and translating, or converting, content from one representation to another.

Every Web application is associated with a variety of costs, including those associated with the design of business logic, creating an attractive layout, deciding which parts of the application to make secure, publishing the various resources associated with the application to a Web server or Web application server, testing the visual appeal of the resulting pages, testing to see whether the outcome of the application is correct, and testing to see whether the application works correctly with various browsers and various different browser versions. Over time, there are also costs associated with maintaining an application, including operational costs for server processing, storage, and network bandwidth, and development costs for adding new features, updating the presentation, retesting, and keeping track of the resources.

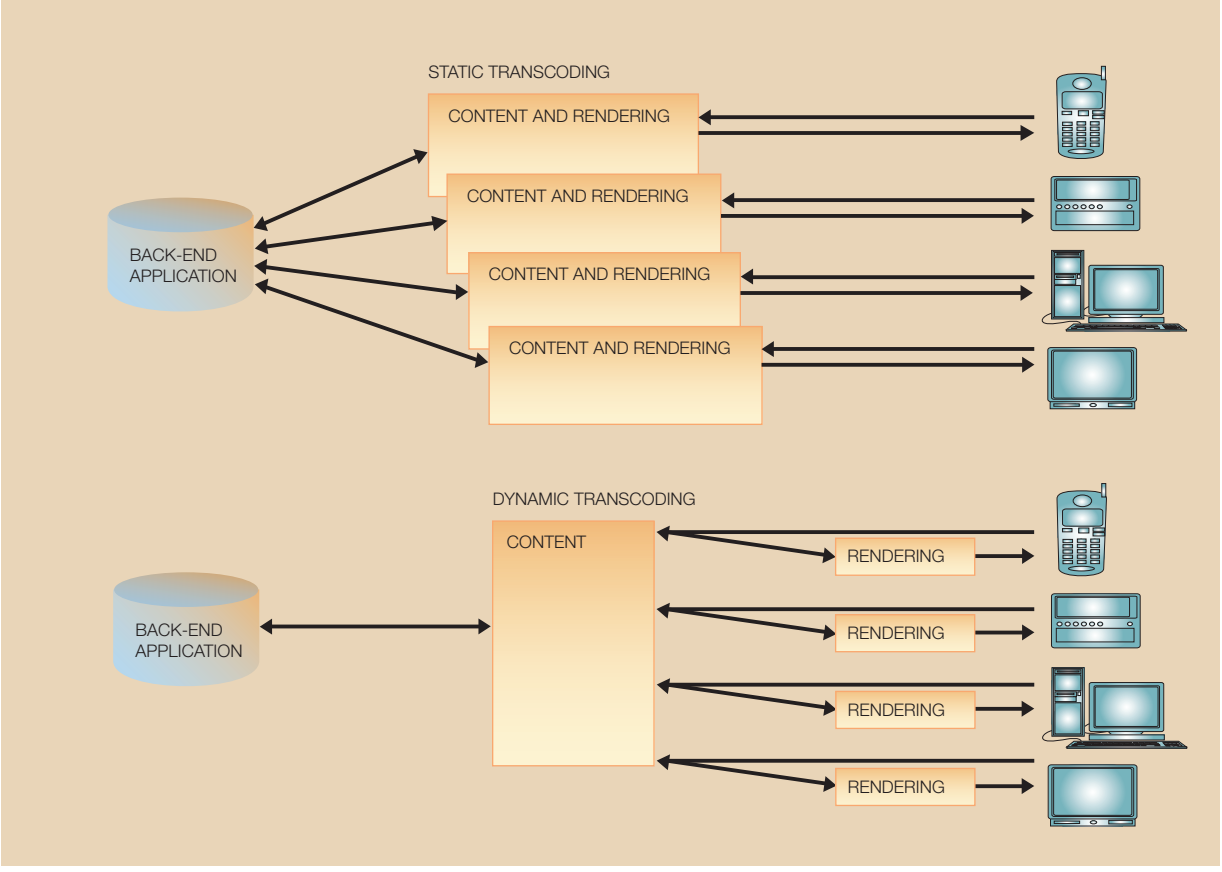
The proliferation of wireless devices with very different form factors multiplies the costs associated with Web applications. Different devices require pages to be presented in different markup languages, such as HyperText Markup Language (HTML), compact HTML[4] (including a version called imode that is popular in Japan), Unwired Planet's Handheld Devices Markup Language (HDML),[5] Wireless Markup Language (WML),[6] and VoiceXML. Even with a particular markup language such as WML, different presentations may be required on different specific devices. For example, for one WML-based phone, choices are most appropriately presented as buttons. For another phone, they might best be presented as hyperlinks. Other constraints come into play. For example, different phones can accept different size decks, where a deck is a set of information that can be sent to the phone at one time. Sending a phone a deck that is too large will cause the application to fail to display properly. Some forms of content may be much more expensive for a server or network to process than others.[7] The existence of a growing number of client devices with different presentation requirements presents new problems for maintaining an e-business site.

In addition, there are business opportunities for network intermediaries, such as Internet service providers (ISPs), to make existing Web applications available to a greater market by making them accessible from different devices. ISPs do not necessarily own the content that they serve to their clients. Instead, they provide value-added services, such as performance improvements via local caching. Making information available in different forms for different client devices is an additional possible value-added service for them.

Content adaptation

In the following discussion, *application* refers to any form of program that generates information in response to requests from Web users, *content* refers to the information returned in answer to a request, and *rendering* refers to the way the content is presented to the user, for example, how it is laid out on a Web page and how the user navigates through the information. Some markup languages, including most XML dialects, contain content along with tags that identify the meaning of different fields in the content. Other markup languages, such as HTML and

Figure 1 Static and dynamic transcoding



WML, carry both content and rendering instructions, since the tags are primarily concerned with the way the content is presented. Transcoding can modify either the content or the rendering associated with an application. For example, a subset of the content can be selected for a small-screen device, and it can be rendered differently to achieve a more pleasing presentation.

Static Web content includes pages that are presented to all users in the same way, generally retrieved from the file system of a Web server. Dynamic Web content is generated at run time based on information provided by the user. Dynamic Web content includes pages that are generated by programs, such as servlets and JavaServer Pages** (JSP**) that run on Web application servers. In both cases, documents—generally in some markup language—are produced to be sent back to the requesting user.

There are two main approaches to handling the need to present the content delivered by an application differently in different circumstances, for different devices, and for different levels of network service. The first is to design each application for each set of presentation requirements, transcoding the application statically at design time. The second is to modify the application output automatically on the fly, essentially transcoding the application dynamically at run time. These two approaches can be combined in various ways to match the requirements of particular applications. Both static and dynamic forms of transcoding can be performed with both static and dynamic Web content. Figure 1 shows a simple comparison between static and dynamic transcoding.

Static transcoding. With static transcoding of static content, different versions of the same content are

produced by a designer, generally using various studio tools, and stored in the file system of the Web server. The main problem to be solved is selecting the right version of the content for a particular user with a particular device. One way to perform this selection is to provide the user of a specialized device with a special home page, often preconfigured in the device. This home page contains a set of links to content in the appropriate form for the device. All the links in this content point to other content appropriate for the device. If the users of phones are using them for specific applications rather than for general Web surfing, this method provides appropriate content with no user actions.

Both static and dynamic transcoding have a place in creating effective e-business sites.

With static transcoding of dynamic content, either different versions of the application are provided or the application contains logic that produces different versions of the content. In any case, the choice of version is based on information from the incoming request that describes the device. For example, there can be a version of a particular JSP that produces HTML for a full-function browser, another that produces WML, another that produces VoiceXML, and so on. Once again, the JSP for the different presentations are designed by humans who are cognizant of the presentation on the specific target devices. The Web application server may provide assistance by selecting the correct JSP for the incoming request. For example, IBM's WebSphere* Application Server[8] 3.5 includes an extensible file called client_types.xml that it uses to map patterns found in the HyperText Transfer Protocol (HTTP) headers to specific output markup language requirements.

The primary advantages of static transcoding are quality of presentation and performance.[9] Each page is constructed by a human designer, generally working with tools that show how the application will look or sound and thus allowing the presentation to be hand-tailored for each environment. Also, since the transcoding is done at design time, the information is generated directly in the form required by the user's device. This reduces the run-time processing required to generate the information.

The primary disadvantage of static transcoding is the number of different pages and applications that have to be created, tested, organized, and maintained. As the number of devices increases, the burden of hand-generating different versions of each application or page for different presentations will become onerous.

Dynamic transcoding can also work with both static and dynamic content. In fact, by the time dynamic transcoding occurs, the content is just content, and the means of generating it is not important.

Dynamic transcoding. Dynamic transcoding promises to allow the problem of creating content to be separated from the problem of creating different presentations. Dynamic transcoding consists of a set of techniques for tailoring the information generated for a user to match the specific presentation requirements of a given user on a given device on a given level of network service. Various dynamic transcoding mechanisms are possible. Several examples are described later in this paper. They include a mechanism to select the proper stylesheet to convert an incoming XML document into a presentation appropriate for a particular output device, a mechanism to translate from HTML into various other markup languages, including HDML, compact HTML, and WML, and *clipper* mechanisms that extract the subset of the content that is most appropriate for display on a small-screen device. To the extent that these mechanisms can be used across a wide variety of content for a wide variety of devices, they reduce the overall development and maintenance costs of Web applications.

Both static and dynamic transcoding have a place in creating effective e-business sites. Some pages will be accessed frequently and will be seen as the signature pages for a business. For these, hand design will often be important. However, there are a number of considerations that will make dynamic transcoding more cost-effective. These include:

1. Is information to be presented that is not under the user's direct control? This situation can occur for various reasons, such as (1) an organization has separate groups working on content authorship and its presentation to different devices; (2) an ISP or a portal provider has legal agree-

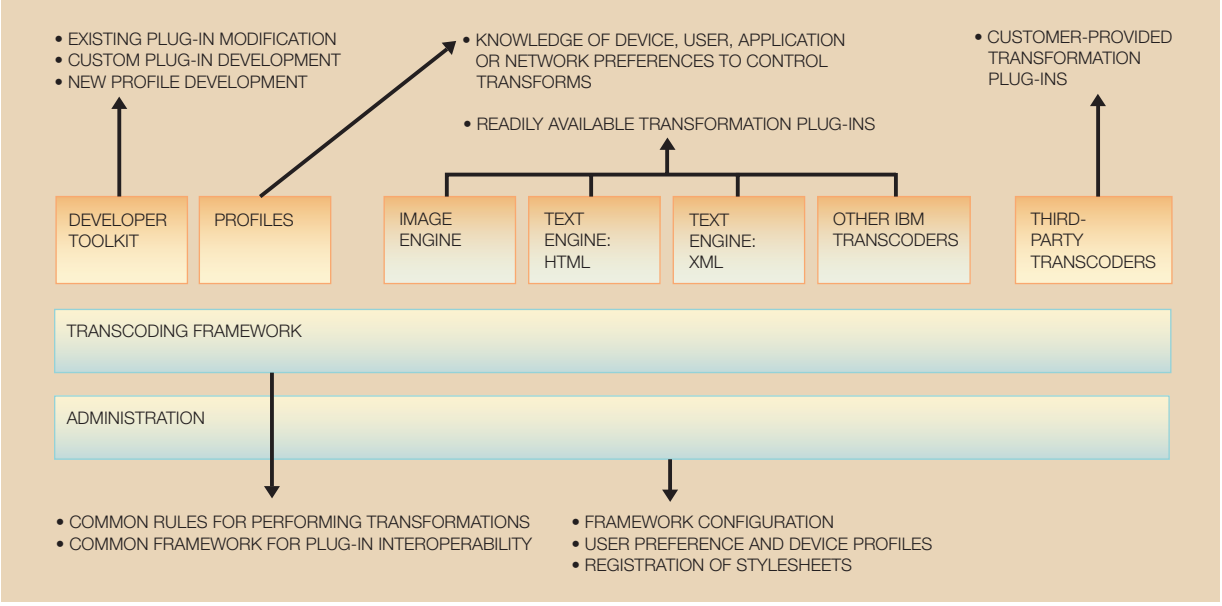
ments with the content owners that allow them to tailor it for presentation to different devices.
2. What skills does the user have that can be used to develop the complete system? Static transcoding may need to be done by persons skilled in Web page design or dynamic content generation. Dynamic transcoders may need to be built by persons skilled in Extensible Stylesheet Language (XSL) stylesheet development or clipper development. The most successful projects use a design that allows for a clean delineation of roles and responsibilities. That is, the content experts can generate the content without having to understand the details of page design, server administration, or device support. The site designers can create a user experience without understanding all the details of the content or the many different client device types. The presentation customization experts can focus on the required devices and their capabilities without the need to understand either the content or site design in detail.
3. Is there only concern about preparing content for a single device, such as a WML-enabled phone, or are there other target content types or device requirements? For example, some Wireless Application Protocol (WAP) predecessors send HDML. In Japan, another HTML variant called imode is popular. Some customers want to present information in a speech markup language, such as VoiceXML. The more different target formats required, the more work there is to author content or content generation programs directly for each one.
4. With WAP phones, are multiple types of phones being served with different deck sizes? Transcoding can include "deck fragmentation" that takes care of breaking a page into pieces that can be accepted by a specific phone, and creating navigation links between them. This facility makes the job of generating the original content—even if already in WML—simpler for a variety of phones with different deck sizes.
5. Can the need be foreseen to support new devices, such as automotive browsers, within days of their appearance on the network? If there are a large number of applications that involve static transcoding, it may be necessary to upgrade every application to support the new device. Any time an application is modified, the existing support may need to be retested.
6. How much content maintenance is there? Are pages or content applications modified frequently? Rapidly changing content may preclude static transcoding.
7. Is any of the content produced in XML, where there is a need to apply stylesheets to generate the right output content for the client device? Where and how is the linkage between devices and stylesheets to be maintained? Is it necessary to be able to send XML documents to browsers that are not able to apply stylesheets?

Use of XML and XSL stylesheets for content adaptation. XML is the universal format for structured documents and data on the Web and is a standard recommendation maintained by the World Wide Web Consortium (W3C**). Because XML is an open, standard data representation language with ever-increasing support from both business and educational institutions, it is becoming an almost universal language used for a growing number of purposes.

Unlike HTML, which mixes data and display information, XML is intended to represent the content and structure of data, without describing how to display the information. Although XML allows the structure of a set of data to be defined, it does not usually provide information for changing the structure of the data to conform to a different set of requirements. For instance, if business A uses one structure to represent a set of data and business B uses a different structure to represent the same type of data, XML itself does not provide the information to transform the data from one structure to the other. However, the World Wide Web Consortium maintains a standard recommendation called XSL Transformations (XSLT). XSLT is a language for transforming XML documents into other documents. It can be used to transform XML into markup languages suitable for display (like HTML), or into XML documents having a different structure. Documents written for use by XSLT engines are called *stylesheets*.

One of the major design issues related to the economics of maintaining Web applications is whether to use XML in place of rendering-based markup languages, such as HTML or WML. For many businesses, it makes sense to extract content into a single-use format that describes the semantics of the information. This format is more widely useful than the same content in an HTML format or WML format that cannot be easily manipulated for other data access requirements, such as business-to-business interactions. Reduced information technology infrastructure costs can come from writing the data extraction programs once into a single form, in this case XML, that can be used for many different data processing requirements. Once the decision to move to an XML

Figure 2 The structure of the WebSphere Transcoding Publisher



deployment model has been made, transcoding via XSL stylesheets can transform the data into a wide variety of business interchange and rendering formats, including HTML and WML.

However, there are costs involved in writing and maintaining XSLT stylesheets. For these costs to be lower than maintaining different versions of the application, stylesheets have to be written intelligently, with parameters that allow a single stylesheet to work for multiple presentations and with reusable chunks that can be used across a large number of applications. Otherwise, an organization will end up needing to write and maintain a large number of stylesheets. Moreover, as XML and XSL mature, tools that reduce the effort to create stylesheets will become readily available, thus further reducing the amount of time required to be dedicated toward writing stylesheets. Finally, well-known techniques for representing user interfaces in a generic fashion such as those employed by Dharma [10–13] can be leveraged to make XSL a more general-purpose language. These approaches cause the original XML content to be first transcoded to an XML document that represents a user interaction. From there, general-purpose stylesheets can transcode the content to the rendering markup language appropriate for a specific requester.

Dynamic transcoding as implemented in WebSphere Transcoding Publisher

In March 2000, IBM released WebSphere Transcoding Publisher (WTP), [14] a product that performs several kinds of dynamic transcoding and can be deployed in different ways. WTP is designed as a "backbone" and "plug-ins." This structure is shown in Figure 2. The backbone includes the function to dynamically route individual requests to the proper transcoders, based on the document source URL (uniform resource locator), device characteristics, network capabilities, and user preferences.

Specific types of dynamic transcoding implemented in WTP are described in this paper, including:

- HTML simplification: modifying HTML documents to tailor them for reduced function devices, for example, by removing unsupported features
- HTML to WML transcoding: transforming HTML documents into Wireless Markup Language documents suitable to be sent to WML-based phones
- XML stylesheet selection and application: selecting the right stylesheet to convert a particular XML document for presentation on a particular device
- Fragmentation: breaking a document into fragments that can be handled by a particular device

and creating the navigation links between the fragments
- Image transcoding: modifying images by converting them to different formats or reducing scale or quality, or both, to tailor the image for presentation on a particular device

The paper also explains the different deployment options, including criteria for selecting a particular option. WTP may be deployed:

- As a stand-alone proxy that is interposed between the client device and the Web server
- As a proxy that pulls transcoded results through a separate caching proxy product, such as WebSphere Traffic Express or the National Science Foundation's Squid
- As a MIME (Multipurpose Internet Mail Extensions)-filter servlet running in WebSphere Application Server
- As a set of JavaBeans** transcoders that can be called by servlets, JSP, or other programs.

Finally, the paper discusses some future transcoding possibilities that could be implemented as WebSphere Transcoding Publisher extensions or as separate products.

HTML simplification. Although powerful full-function workstation browsers such as Netscape Communicator** 4.5 (and later) and Microsoft Internet Explorer** 4.5 (and later) are widely used, many less powerful devices have browsers that support only a subset of the HTML functions supported by the more powerful browsers. In addition, because some features, although supported, might so strain the resources of the device that they are troublesome, a user might prefer to have a less expensive construct substituted.

For example, there are numerous handheld computers running the Windows CE** operating system, which uses a reduced function version of Microsoft's Internet Explorer browser. Early versions of the browser do not support Java applets. Devices with even more constrained resources, especially memory and processing power, may have browsers with even less support for HTML functions, such as frames, JavaScript**, and perhaps even tables. As the different types of somewhat limited devices that provide some type of HTML browser to allow accessibility to the Internet grow in number, the permutations of HTML capabilities that are and are not supported also grow.

Despite the large number of limited-function devices, much of the HTML content being written today is produced to take advantage of the advanced capabilities of powerful workstation browsers. Rather than each content provider producing multiple versions based on the capabilities of each possible client device, it is desirable to allow the content to be created once. Transcoding can be used to transform the content into a form suitable for the client device. In WebSphere Transcoding Publisher, this process of transforming the HTML content based on the capabilities of the client HTML browser is called *HTML simplification*. This transcoding could be done in two ways: (1) create transcoding function unique to each different device type, or (2) characterize the capabilities of devices in some way and use these characteristics as parameters to direct the transcoding function.

For the reasons discussed earlier, and considering the rapidly growing number of different types of devices supporting HTML browsers with differing levels of support, the WebSphere Transcoding Publisher took the second approach. By using a set of properties that describe the capability of a given browser to render various HTML elements, a *profile* describing the browser can be generated. This profile provides sufficient information to direct the transcoding process. In WTP these descriptive properties are referred to as *preferences*, because in some cases these parameters are truly what is *preferred* in the context of the environment in which such devices might be used, such as using a wireless connection to access the Internet. In a wireless environment, the textLinksPreferredToImages preference might be set to *true*, indicating that the HTML simplification process should generate a link to a separate page containing an image rather than automatically downloading it and rendering it in-line in the page, thus giving the user the choice of whether to spend the time and byte transmission cost of downloading the image to the device.

As another example, some limited-function browsers are unable to render HTML TABLE elements. For this case, WTP uses a convertTablesToUnorderedLists preference that can have a Boolean *true* or *false* value. For a limited browser that does not support tables, this value is set to *true* to indicate that the table should be converted into an unordered (nested) list. This preserves the relationship between nested tables by representing them as nested unordered lists.

Table 1 HTML simplification with preferences indicating that images should be converted to images to links and that tables should be converted to unordered lists

HTML Fragment	Simplified HTML
<HTML> <body> <img src="images/house.gif" alt="Picture of a House"> <table> <tr><td>991105001PB</td> <td>Ranch</td> <td>$207,000</td> <td>4</td><td>2</td></tr><tr> <td>991105002PB</td> <td>Split Level</td> <td>$237,000</td> <td>5</td> <td>2.5</td> </tr> </table> </body> </HTML>	<HTML> <body> <A HREF="images/house.gif/PVC_PHANTOM_GENERATOR"> Picture of a House 991105001PB - Ranch $207,000 4 2 991105002PB - Split Level $237,000 5 2.5 </body> </HTML>

Currently, approximately 20 preferences in WTP can be used to guide the HTML simplification process, including preferences that indicate which types of image formats are not supported, whether frames are supported, and whether images should be removed entirely.

HTML simplification is only applicable if the content is marked in HTML. The HTML simplification described here is available in WebSphere Transcoding Publisher. It is customizable using the preferences, but flexibility is currently limited to those preferences. Table 1 shows an example of HTML simplification. Figures 3 and 4 show the before and after HTML files from Table 1 rendered in a browser.

XML stylesheet selection and application. An increasingly used approach with XML is to generate content in XML and use XSLT stylesheets to transform the information into a form suitable for display, or to transform the structure to a different one suitable for another party. Thus, XSLT stylesheets can be written to support display of XML content as well as to support the use of XML in electronic data interchange (EDI).

WebSphere Transcoding Publisher provides a framework to aid in using XSLT stylesheets to transform XML documents. The WebSphere Transcoding Publisher includes a version of the *Xalan* XSLT processor, an open-source XSLT processor supported by the Apache XML Project.[15] The major benefits that WTP provides in dealing with XSLT stylesheets and their application to XML documents is a structure for organizing stylesheets and support for defining the criteria used for selecting the proper stylesheet to be used in the transformation of XML documents. For example, an organization might generate content in XML and create different stylesheets to render the content for display on different device types. They might create one stylesheet for generating an HTML version, another for a WML version, and so on. A WTP administrator registers a given stylesheet with WTP, providing the location of the stylesheet, the content type generated (for example, text/HTML), and selection criteria that must exist in the context of an incoming XML document request in order for the stylesheet to be used in transforming that XML document. For example, key-value pairs representing properties associated with a given device, such as the descriptive name of the device, can be used as criteria, guaranteeing that the stylesheet will only be used on requests originating from such devices. Additionally, conditions matching all or a portion of a given document URL can be specified, guaranteeing that the stylesheet will only be used on one or a small number of documents originating from a given server location.

As more and more enterprises use XML and stylesheets, many interesting issues arise. For example, the document type definitions (DTDs) defining

Figure 3 Original HTML page displayed in a browser

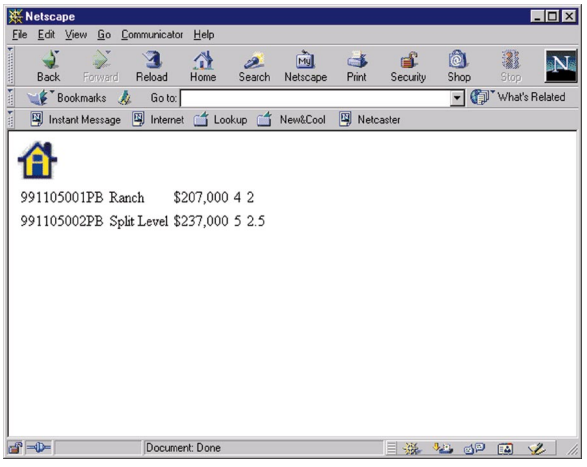
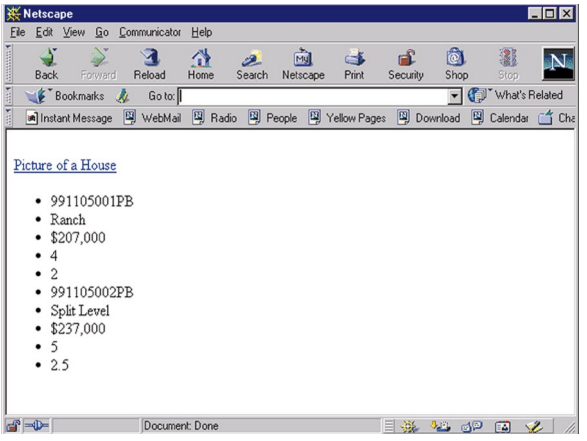


Figure 4 Transcoded HTML page showing the simplified layout



a given type of structure of an XML document and the stylesheets used to manipulate such documents might be considered important proprietary resources and need to be kept secure. Also, as the number and types of devices allowing Internet access increase, an approach based on different stylesheets for each device type could cause the number of stylesheets an organization needs to create and maintain to increase precipitously. As WTP and stylesheet tools evolve, there will be increasing support for stylesheet parameterization and stylesheet internalization, and increasingly complex criteria for selecting the right set of stylesheets for a given purpose. Stylesheets are standard, general, and extensible, but using stylesheet transcoding requires that an XSL stylesheet be written to perform the desired transcoding.

Transformation of HTML into WML. The Wireless Markup Language (WML) is an XML-based markup language intended for use in specifying content and user interface information for narrowband devices, including pagers and cellular phones. It is part of the Wireless Application Protocol (WAP) standard from the WAP Forum.[16] WAP-enabled phones are becoming more and more prevalent, especially in Europe, with the Nokia Group and others manufacturing phones that support the WAP wireless standard. Many enterprises that now provide information access to users using desktop browsers wish to support their mobile users who would like to access their information by using WAP devices, but the enterprises do not wish to expend the resources to provide content in both HTML for workstation browsers and WAP devices. WTP provides transcoding support for transforming HTML into WML.

WTP transforms HTML into WML in a two-step process, with an optional third step to further refine the result. The first step takes the original HTML document and converts it into a Document Object Model (DOM) representation using an HTML parser developed by a team at the IBM Tokyo Research Laboratory. The DOM is essentially a tree structure, conforming to a World Wide Web Consortium requirements standard.[17] Using standard Java application programming interfaces (APIs) for traversing and manipulating the DOM, the second step modifies the HTML DOM, creating a DOM containing WML elements, conforming to the WML standard. This transformation is done on an element-by-element basis. For example, an HTML document starts with an *HTML element* (node), whereas a WML document starts with a *WML element*. Thus, the transformation logic responsible for processing the HTML node replaces this node with a WML node in the DOM.

The logic for dealing with a particular HTML element falls into one of three categories:

1. The transformation requires unique logic, such as the HTML example described above.
2. The source element, and all the children under it, are simply deleted. An example is the APPLET element, which represents a Java applet. Since it is not possible to transform the Java applet into some

other form for execution on the WAP device, the APPLET element and its contents are simply deleted.
3. The source element is replaced by its children. That is, everything under the top element replaces the top element itself. For example, the SUP element, which indicates text contained within it should be rendered in a superscript font, is replaced by the text it contains, since superscript fonts are not supported in WML. In text form, this would be seen as replacing "^{special text}" with simply "special text."

The first case above requires special logic for each transformed element, whereas the second and third cases can each be handled by a single processing module, with the HTML elements to be handled simply by providing them to the module in the form of some list. Thus, the module responsible for deleting elements and their children is given a list of element names that should be transformed in this way.

Once a document has been converted to the target markup language, it can be further tailored by a step called *text clipping*. There are a couple of reasons why this might be beneficial:

1. The application of generic transformation logic to a specific document, although resulting in a valid WML document, may still not result in the most visually pleasing results.
2. The transcoded document may contain a large amount of unnecessary content that tends to obfuscate the key information the user was requesting. For example, although it may be reasonable to provide links to every product and department in a company in a table at the top of a page rendered on a powerful desktop browser, these links may just frustrate the user as he or she has to scroll past them to find the information actually requested.

There are three ways in which the text clipping process might be accomplished:

1. Because WML is an XML-based language, XML stylesheets can be used.
2. The transcoding logic can work with the text form of the document, using traditional string manipulation functions to locate the points at which to remove text.
3. The transcoding logic can operate on the DOM directly.

All of these approaches require that specialized logic be written, in either Java or XSL, to perform the desired clipping.

The first approach, applying stylesheets to XML, was explained in the previous subsection and is not explained further here. Experience has shown that the second approach can have serious limitations unless the content never changes. Assume that the content is a weather forecast and that the main forecast description is preceded by the string "Local Forecast." Suppose, "Local Forecast" was enclosed in a B (bold typeface) element:

<B>Local Forecast</B>

Note that if we search simply for the string "Local Forecast" and remove the text prior to this point, the result is

Local Forecast</B>

which is invalid, because there is an ending tag (</B>) without the beginning tag (<B>). The solution to this problem is to search for the *exact* text, including tags. However, even a small change to the content, such as the B tags being replaced with I (italic) tags, would make the transformation fail.

The third text clipping approach, working directly with the DOM, has distinct advantages. Individual nodes in the DOM tree represent the beginning and ending tags of the element, so that when a node is removed, there is no problem with removing the beginning tag without the ending tag. Also, it is possible to search a portion of the DOM for text regardless of the element nodes of which they are children. For example, it would be possible to search for the text node "Local Forecast" regardless of the fact that the text was a child of a B or an I node. To aid in the creation of text clippers that operate on the DOM, WTP provides functions such as findNodeMatchingPattern, which searches under a particular node in the DOM for a text node whose text matches a given regular expression pattern, and findSiblingMatchingPattern, which searches under siblings of a given node and their children for a text node matching a given pattern. With use of such functions to allow searching and modification of the DOM based on string patterns, it is possible to write text clipping transformations that can withstand minor changes to the source content without requiring updates to the text clipping logic.

Figures 5 and 6 show tree representations of a DOM before and after clipping. The DOM in Figure 6 is created from the DOM in Figure 5 by a Java clipper performing the following steps:

Figure 5 DOM created by an HTML to WML transcoding step

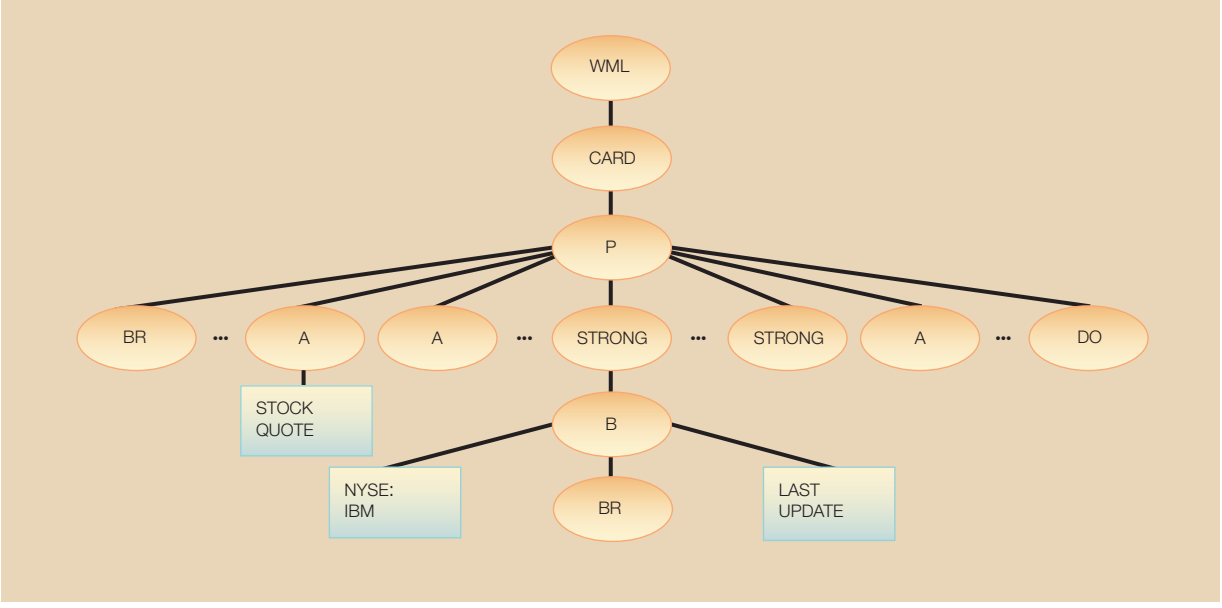
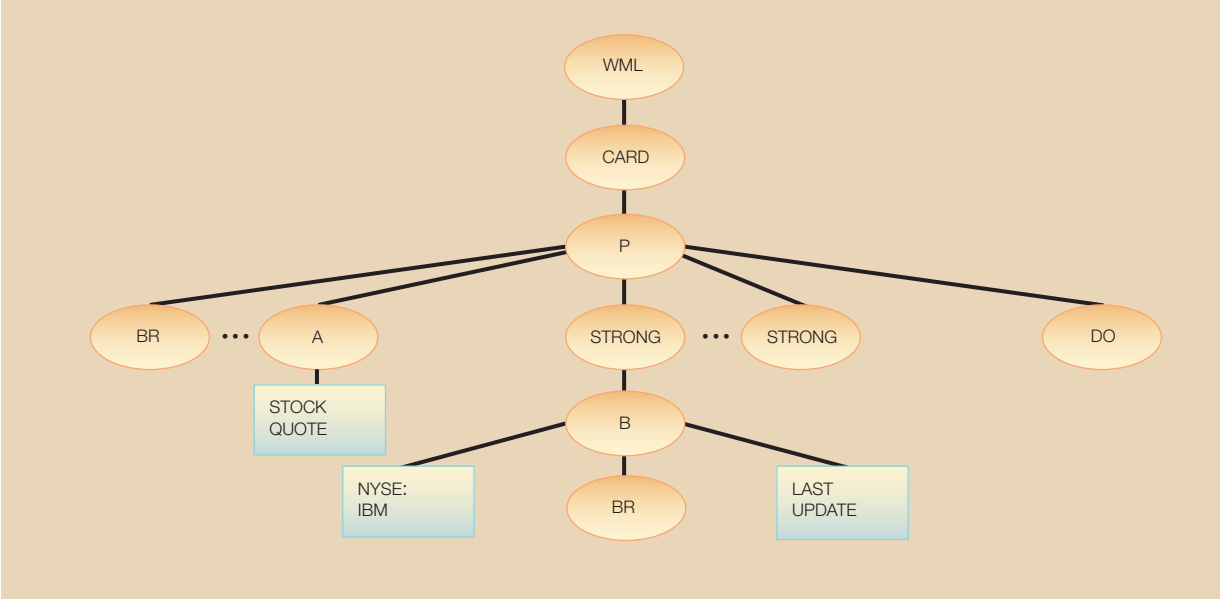


Figure 6 "Clipped" DOM, created from the previous DOM by a Java clipper



1. Find the paragraph (P) node in the document. There should only be one of these generated by the previous HTML to WML transcoding step.
2. Search for a child node of the P node that has a text node somewhere beneath it containing the string "Stock Quote."

3. Remove all siblings before the node found in step 2.
4. Look at all siblings following the node found in step 2 until we reach the first anchor (A) node.
5. Starting with the anchor node found in step 4, delete all nodes until we find the node that contains a text node beneath it containing "Last Update."
6. Starting with the node found in step 5, keep all sibling nodes until the next anchor node is reached. (Note that the last node before the first anchor node found is a STRONG node.)
7. Starting with the anchor node found in step 6, delete all following sibling nodes until a DO node is found.

The HTML to WML transcoding approach described above can be used for transforming HTML to other markup languages, such as HDML, compact HTML, or imode.

WML deck fragmentation. WML content, as defined by the WAP Forum, is organized for rendering in a *card*, similar to a page in HTML. A collection of cards is known as a *deck*. A single WML file contains a deck and its component cards, allowing several cards to be delivered to the phone in one request, even though only one card is displayed at a time. Many devices such as WML phones have a limited memory capacity. Thus they present a special challenge for transcoding, since the transcoded content delivered to the device must not exceed the capacity of the device. In general, HTML pages that are converted to WML as described above will not fit in the limited storage available on these devices. Even content that is generated as WML originally may exceed the device limit. All such content must be subdivided so that each subdivided piece will fit in the device. In WTP, the process of breaking up such pages into acceptably sized pieces is called *WML deck fragmentation*. This procedure is performed by the *WML fragmentor*. Although the initial implementation works with WML, the same techniques are easily applied to other markup languages such as imode.

If the WML fragmentor needs to fragment a WML deck generated by the HTML-to-WML transcoding process, it can retrieve and operate on the DOM created for the document during that process. Otherwise the fragmentor parses the WML text itself in order to create a DOM. In either case, the fragmentor performs all of its operations on the deck as represented by a DOM, not on the plain WML text.

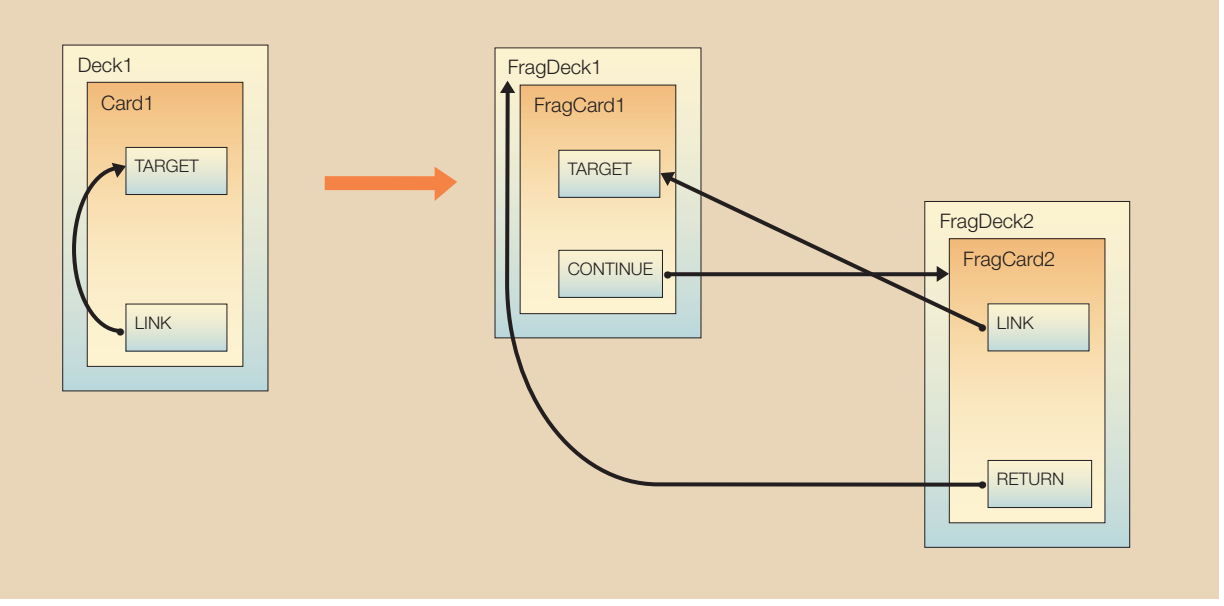
When presented with a WML deck that may need to be fragmented, the fragmentor must first determine both the maximum size deck that may be delivered to the device, and the size of the input deck as it will be presented to the device. The first piece of data is a configuration parameter associated with the user-agent field present in the HTTP request that produced the deck. The second is determined by "walking" the DOM that represents the deck and counting the size of each node as it will be seen by the device. When doing this counting operation, the fragmentor takes into account the binary encoding for WML tags that will be performed by the WAP gateway.

If the deck exceeds the maximum allowed size, then the fragmentation process begins. First, the individual cards are removed from the deck so that each may be considered independently. If a single card by itself exceeds the maximum card size, it must be fragmented into smaller subcards. A card is fragmented by walking the DOM that represents the card, keeping track of places in the tree where the card may be fragmented. The fragmentor will fragment on either <p> or
 tags. As the fragmentor visits each node in the tree, it determines the size of the card that would be produced if the main card was fragmented at this node. The fragmentor continues walking the DOM until this new card size exceeds the maximum. Once this limit is reached, the fragmentor ceases walking the DOM and fragments the card at the last noted fragmentation point. This process may be repeated for the card remaining after fragmentation, if its size still exceeds the maximum allowed. Each card that is oversized is fragmented in this fashion.

Note that as a card is fragmented, links are added so that the user may navigate from one card to another. Each fragment contains a <prev> element to allow navigation to the previous card, and a <go> element that will move the user to the next card in sequence. Figure 7 shows an example of linking fragments together in this way.

Once all of the individual cards are small enough, they are arranged into decks so that each deck is not larger than the maximum deck size. Unique deck names are generated by the fragmentor based on the original request URL. The final step in the fragmentation process is called *link rebinding*. During this step, the text of the HREF (the parameter specifying the URL of a linked resource) attributes of all links found in the decks is adjusted so that the target name

Figure 7 Example of WML deck fragmentation; arrows represent links, either within or between fragments



is correct, based on the name of the deck in which each card was placed.

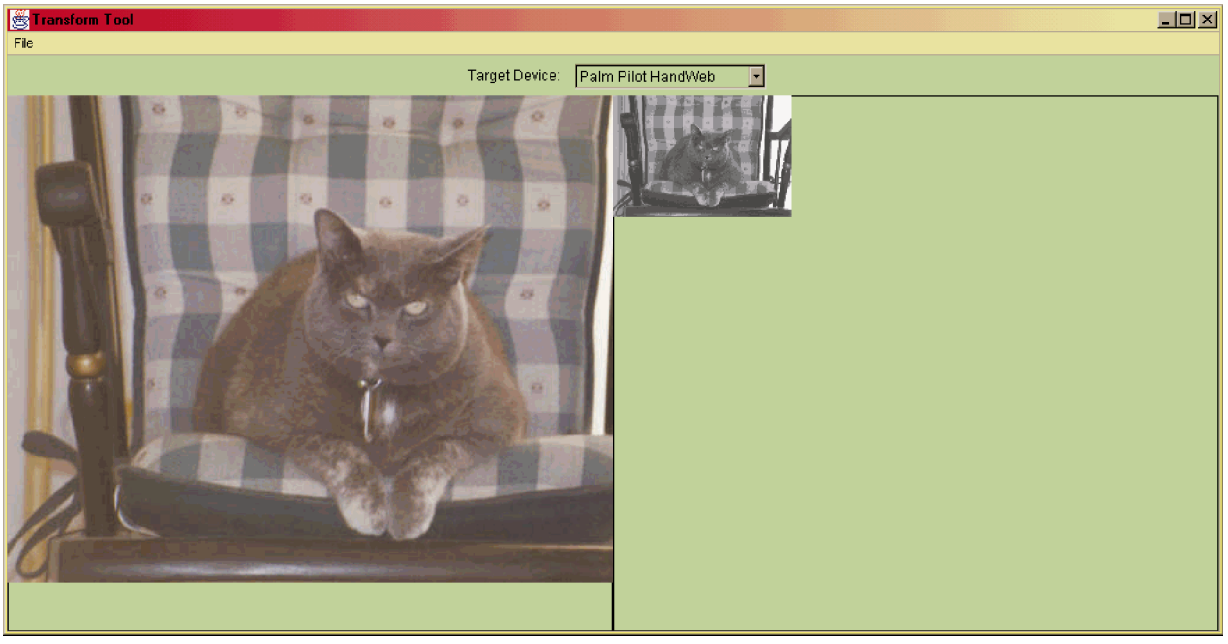
At this point the fragmentor sends the first generated deck to the device and stores the remaining generated fragments in the resource repository. The resource repository is simply a general-purpose storage facility maintained by WTP that allows for retrieval of one of the decks when a request is received for it. Such a request will be generated when a user follows any of the links in the first deck that point to a card in one of the other generated decks. Decks in the resource repository must eventually time out and be removed. Any one of a variety of cache algorithms may be used to determine when to remove decks from the repository. If a request is received for a deck that has been removed from the repository, then a deck is generated that contains an error message indicating that the fragment has expired and the original deck must be requested.

The WTP WML fragmentor described here works on any WML deck without any customization.

Image transcoding. The final type of dynamic transcoding we discuss is image transcoding. Images that view well and render (reasonably) quickly on a desktop browser connected to a local area network (LAN) may not appear in a usable form on the low-quality and small-sized screens of handheld devices. In addition, they may take unacceptably long to download and render on such devices. Finally, some devices do not support all image formats; therefore, to display an image at all it may be necessary to convert from the original image format to one supported by the device before transmitting the image to the device.

Image transcoding is performed in WTP by the *image engine*. Figure 8 is an example of an image processed by the image engine. The image on the right in the figure has been scaled and converted to gray scale to reduce the image size for display on a wireless device. Given an input image to transcode, the image engine must determine three things: the output format that should be used for the image, the scale factor to be applied to the image, and the output quality of the image. "Output quality" is a somewhat vague term. It is a measure of how "good" the image will appear on the screen. High-quality images will look sharp and clear, but will generally be considerably larger (in bytes) than lower-quality images. Also, note that low-quality screens may be incapable of displaying a high-quality image. In this case it is preferable to use a lower-quality image as output, since that will speed both transmission and rendering time.

Figure 8 Example of image transcoding—a screen capture from the "Transform Tool" in WTP



Deciding the output format for an image is straightforward. If the input image is in a format accepted by the device, the format is not changed. Otherwise, the output format is set to be the first of the formats supported by the device (as determined by the user-agent, or UA, field in the HTTP request).

Scale factor may also be a simple configuration parameter associated with the requesting device. Generally, it will be set so that images appear relatively as big on the screen of the requesting device as they would on a desktop screen. Windows CE devices, however, do not have a fixed screen size. Rather, the screen size is indicated by the UA-pixels header included in the HTTP request. When this header is present, the image engine can use this information to dynamically calculate a scale factor based on the difference in size between the screen described by a UA-pixels header and some (configurable) standard screen size.

Finally, output quality is determined by combining information about the quality of the screen of the requesting device and the configured trade-off that should be made between quality and size for this request. The screen quality is a configured parameter for the device, and may be set to "high," "medium," or "low." The trade-off configuration parameter will generally be associated with a network type: Over a slow wireless link it would be better to favor small size over high-quality images, whereas over a LAN connection high quality would be preferred, since there is little to be gained in reducing the image size. Over a dial-up connection it would be appropriate to compromise between the two. The image engine first chooses a tentative output quality based on the screen capability, and then adjusts this size downward if the trade-off parameter is set to either compromise or favor small size.

The WTP image engine described here works without customization using defaults for preferences based on the client device type.

Source identification and preference aggregation

One of the major goals of WTP is to tailor the content sent to the user based on the user's environment—especially the device and connectivity limitations. The following two major steps are needed to accomplish this:

1. Identify the characteristics, constraints, and preferences for each environment *source*. Such sources include device types, network types, and users. This step is a configuration step. In WTP the characteristics, constraints, and preferences are all given the single name *preferences* and are defined as key-value pairs consisting of a name and a value. For a given incoming request for some document or image from a user, identify the environment sources. For example, when a user requests www.ibm.com via a browser running on some client device, WTP tries to determine source information such as device type and network type[18] to determine what information is necessary to provide the most suitable transformation of the requested document. This step is called *source identification*.
2. Given multiple sources of preference information, provide the value of a given preference to use when requested by some transcoding logic. This issue is trivial when only one of the sources provides a value for a given preference. However, it is possible, for example, that both the device source and the network source provide a value for a given preference. In such a case, it is desirable to have some single mechanism responsible for taking multiple values for a given preference and providing a single value. With use of such an approach, numerous transcoding modules can be written using preference values, but without regard to how these values were determined. In WTP, the mechanism that provides this function is called the *preference aggregator*, because it "aggregates" multiple values into a single value.

Source identification. Identifying the device source type for an incoming HTTP request is based on the *user-agent* field in the HTTP header. For example, the user-agent field for an English language version of the Netscape Communicator 4.7 browser running on a Windows NT** system is "Mozilla/4.7 [en] (WinNT; U)." The user-agent field for the Netscape Communicator 4.6 version was identical, except that it specified "4.6" rather than "4.7." Rather than requiring source identification on exact matches on the entire user-agent string, WTP supports source identification based on a portion of the user-agent string. For example, specifying the pattern "*Mozilla/4.*" would mean that any browser with a user-agent string containing the string "Mozilla/4." would match the pattern. Additionally, WTP supports device source identification via pattern expressions combined via logical operators. Thus, the expression:

```
(user-agent=*Mozilla/4.*)|(user-agent=
                                *Mozilla/5.*)
```

means that any user-agent string containing "Mozilla/4." or "Mozilla/5." would match the pattern expression.

For network source identification, WTP currently does not have the benefit of an HTTP header field like the user-agent field used for device source identification. In the future, if WTP is used with some type of (perhaps wireless) gateway, it may be possible to receive some network connection identification information. Currently, WTP provides a much more basic mechanism. When operating as a transcoding proxy, WTP supports the use of separate Transmission Control Protocol ports to identify the type of network connection being used. That is, WTP can be configured to understand that port 8089 is a wireless network connection. Thus, if the browser of a client device is configured to connect to the WTP proxy via port 8089, a request from that device will then be assumed to be coming in via a wireless connection. Therefore, the *wireless* network profile will be used. The standard WTP configuration comes with *default*, *wireless*, and *14.4* profiles configured on different ports, but these can be changed or additional ones added.

Preference aggregation. For a given request, if more than one preference source has a value for a given preference requested by a transcoding module, the preference aggregator must provide a single value. In most cases, for a given preference name, the value specified in one source profile should take precedence over the value in another source profile. For example, the value of compressSource preference for the Windows CE device profile is *false*, meaning that no attempt should be made to reduce the size of HTML elements (by reducing noncritical attributes) sent to such devices. However, the wireless network profile specifies a value of *true* for the compressSource preference. For a request from a Windows CE device using a wireless connection, how should the value of compressSource be resolved? The preference aggregator allows a precedence ordering to be specified to help resolve such conflicts. The default precedence ordering that is configured is:

```
user, network, device, user-default,
  network-default, device-default
```

This ordering states that for a given preference name, if there is a specific (not default) user profile and it

contains the preference value, it should be used. If not, if there is a specific (not default) network profile and it contains the value, it should be used, and so on. Thus, in our example, since WTP does not currently have a way of identifying specific users, there is a default user profile in addition to the wireless network profile and Windows CE device profile. Thus, given the above ordering, since there is a specific network profile (*wireless*) with a value of compressSource, its value (*true*) takes precedence. Because the same generic precedence ordering may not be appropriate for all preference values, WTP allows a different precedence ordering to be specified for preference names for which such an ordering is appropriate.

In some cases, rather than choosing a single value based on a precedence ordering, it might instead be desirable to combine the multiple values in some way. For example, it might be desirable to say that for a given preference whose value is Boolean, if either the network or device value is true, the value should be true. WTP also supports this capability on a preference-by-preference basis.

In order to provide such flexibility, the preference aggregator employs a *rules* mechanism. Generic rules, such as the one providing the default precedence ordering, are supplied, but preference-specific rules can be supplied as needed. As is typical with any rules engine, once a given preference has been resolved, the rules are not needed to resolve the value again. In WTP, a preference value remains for the life of a request because each request is distinct and can have different source information. The rules themselves, although actually Java classes, can be written in text form and then parsed and compiled. The rules language uses a compiled scripting language, NetRexx,[19] as an underlying scripting language so that powerful expressions can be supported without having to use a totally new scripting language. NetRexx, which is compiled into Java, can be used on any platform on which Java can be used.

At initialization time, and when preference profiles are updated because of configuration changes, rules based on the key-value pairs contained in the preference profiles are dynamically generated. Although the rules are compiled, there is still a run-time cost associated with interpreting the rules. In order to minimize the performance impact of using these rules, an analysis of the rules is done immediately after the dynamic generation of rules from preference profiles is done, and those rules that are derived based solely on source identification are cached so that their values can be "looked up" based on source identification information from the request without having to actually interpret the rules.

The rules-based mechanism provides almost unlimited flexibility in extending the preference aggregation mechanism, while allowing transcoding module writers to use preferences without worrying about how the values are resolved. In addition, the preference aggregation mechanism can be updated by an administrator without installing a new version of the product and without having to be a Java programmer.

Deployment models

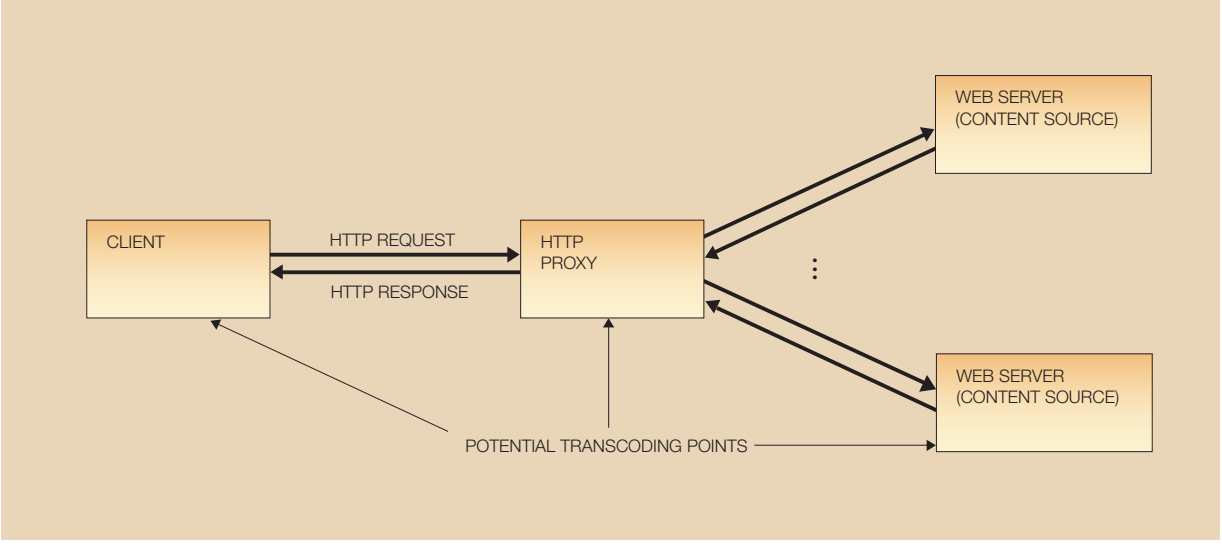
Transcoding must be interposed at some point between the generation of the initial content and the final rendering on the client device. Three locations are available in the current Web architecture:

1. The client device
2. An intermediate point in the network, such as a proxy or firewall
3. The source of the content

Figure 9 shows a simplified picture of a network configuration and the potential transcoding placement locations. Each has its strengths and weaknesses. Therefore, combinations of the approaches may be useful in some situations.

The Web experience is, today, still largely oriented toward a client model that assumes a high-bandwidth network connection and workstation capabilities. Content that is designed based on these assumptions is not usually appropriate for display on lower-capability devices or for delivery on low-bandwidth or high-cost connections. This is the crux of the problem when performing client transcoding. When we first started looking at client transcoding options for WebSphere Transcoding Publisher, we tried visiting a variety of sites on the Web. It could and did take 20 minutes or more to view some pages using a dial-up or wireless connection. Even over faster links, the very limited rendering speeds and capabilities of handheld devices will usually make pure client-side transcoding an unworkable solution. Image transcoding can be particularly problematic for clients because of its computationally intensive nature,

Figure 9 Placement options for transcoding function in a network



large image sizes, and the potentially very high compression rates.[20]

Independent of the transcoding placement issue, there still needs to be a Web-capable presence on the client device. Additional client-side presence can be useful in providing transcoding-specific user options (such as image-specific transcoding options) or a controlled client environment.[21] Such a client can also provide additional information about client device capabilities, although emerging standards such as composite capability/preference profiles (CC/PP)[22] are expected to supplant this role.

WTP assumes only that there is Web rendering software of some sort on the client device and that the HTTP headers generated by the device either specify the capabilities of the device or allow it to be inferred. Currently available Web-capable devices meet this requirement through the use of user-agent and other descriptive headers. This allows WTP to effectively transcode for a wide variety of devices without additional client presence.

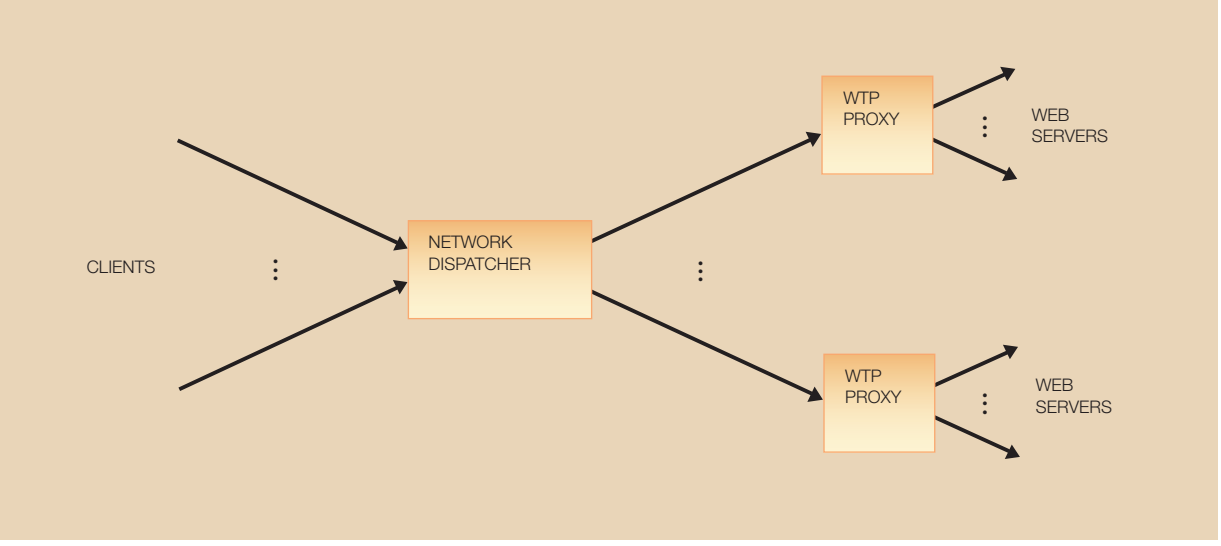
A second deployment option is at an intermediate point in the network, such as a firewall or proxy. In this model, all client requests are directed to the intermediate representative that forwards them on to the specified destination server. Most Web clients support use of proxies for at least HTTP. WTP can be configured to run as an HTTP proxy, and in this mode can transparently transcode content for these clients.

In the proxy model, all HTTP responses flowing to the client can be transcoded, regardless of the server producing the content. This approach, combined with the preference-based transcoding of WTP, allows new client devices to be easily added to a network. For example, a business that wishes to give employees with WAP phones access to internal Web content can do so by using a WAP gateway (for protocol conversion) and WTP proxy (for content transcoding), without requiring content to be specifically authored for the phones, and without impacting servers generating content.

There are some drawbacks to the proxy approach for transcoding. The use of encryption such as secure sockets layer (SSL) to protect data precludes transcoding via a proxy. This drawback could be addressed by allowing the proxy to act as both an endpoint and a source for SSL connections. However, this additional point of exposure may not be acceptable to some users. It also imposes a significant performance penalty at the proxy.

Another potential concern of the proxy-based approach is legal, rather than technical. Some providers of content also wish to control the presentation

Figure 10 Using Network Dispatcher



of this content, and so will not want it to be transcoded outside their control. Although this restriction will not be a concern in the corporate intranet example presented above, it can limit deployment in the external Internet environment.

A final area of concern with this approach is in extending the proxy with new transcoders. In the Web server environment, *de facto* standards such as the servlet API provide a widely supported method for extending function. There is no similar API yet in place for proxies. WTP is based on the Web Intermediaries (WBI) framework,[23] and supports extension via the WBI API. In addition, WTP allows transcoders written to the servlet API to be deployed in a WTP proxy.

Dynamic transcoding can be a relatively resource-intensive activity. Fortunately, techniques available for spreading requests across Web servers or proxies may also be used with a transcoding proxy. IBM's Network Dispatcher, for example, can serve as a front end for multiple transcoding proxies. This configuration is shown in Figure 10. WTP instances do not share transcoded data, and so session affinity mechanisms in Network Dispatcher are used to ensure that requests depending on a previous state (for example, deck fragmentation) are routed back to the correct instance of WTP.

A final option is to provide transcoding at the source of the data. This option, content source transcoding, is attractive for a number of reasons. It requires no client configuration since there is now no intervening proxy. For clients that do not support HTTP proxies, such as some WAP phones, it may be the only workable option. Since the transcoding is coresident with the content, the content provider has tighter control over what is transcoded and how it is presented. Content source transcoding allows transcoding to be done before content is encrypted, and so it can support secure connections. If transcoding is being done to control who sees what data (for example, only allowing users from certain locations full access to data), performing transcoding at the source enforces these views.

The major drawback to content source transcoding is its limited scope. Deploying transcoding in the content source only affects targeted content from that server. It is not appropriate when a client is viewing content from a range of servers. Performing transcoding on the content source server will also impose an additional load on the resources of the server, and so may require additional infrastructure enhancements.

The options for deploying content source transcoding depend on the configuration capabilities of the

server hosting the content. For this reason, WTP provides two models for content source transcoding:

1. Servlet filtering
2. Transcoding JavaBeans

In the first option the output from an existing servlet or JSP is run through a filter servlet after it is generated. This option allows content generated by servlets to be transcoded without change to the existing servlet. Not all Web servers support servlet filtering, since it is not yet part of the servlet API specification (although this is expected to change). Servlet filtering does, however, enjoy enough widespread support to make it a useful deployment option. WTP requires both the output from the existing servlet and information on the original request to successfully transcode content in this model.

If the Web server does not support servlet filtering, or if more control is desired over the transcoding process, the content provider may wish to or need to explicitly invoke transcoding on generated content. WTP provides JavaBean transcoders that can be used to perform image or text transcoding on data streams.

All Web servers in common use today include extension APIs that allow the requests and responses flowing through the server to be intercepted and modified at various points. A transcoding extension written to these APIs provides another option for content source transcoding. However, since these APIs are Web server specific, they are less portable and more difficult to deploy on a range of platforms. WTP does not currently provide support for deployment at this level.

Application scenarios

Transcoding is used to help make different types of content available in different formats, allowing new applications of the content that may be different from the application for which the content was originally developed. We provide some example application scenarios to describe how transcoding can be used in practice.

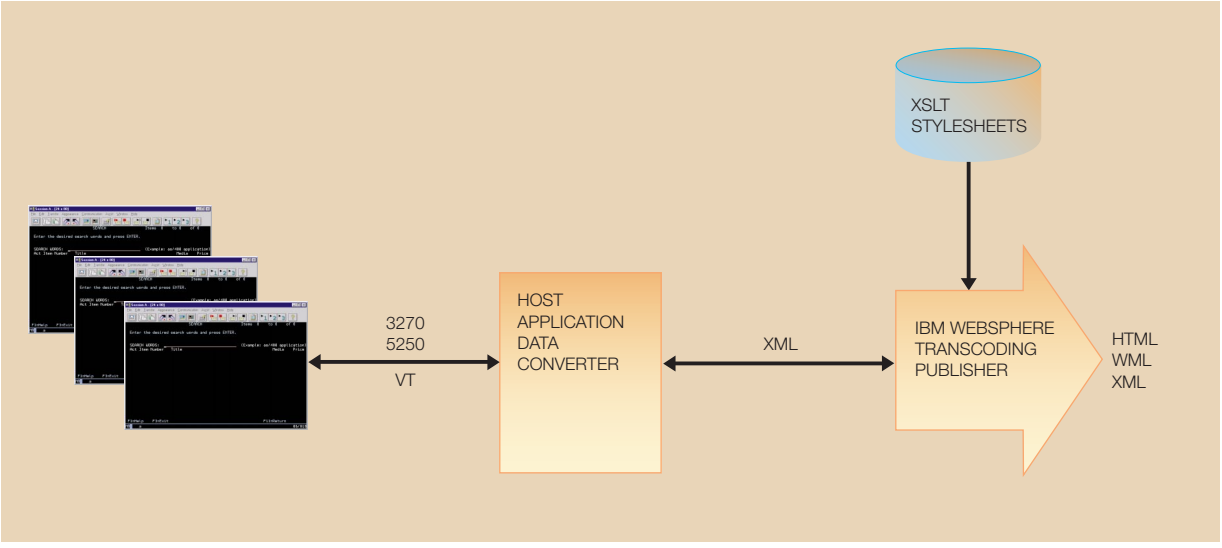
Web browsing. One application of transcoding is to make general Web content available to devices with limited or specialized user interfaces. In this application, the transcoding function is deployed as a proxy so that no changes are required to any of the Web servers that are hosting the content. Many of the transcoding functions described above—such as HTML simplification, image transcoding, conversion of HTML to other markup languages like WML, and deck fragmentation—are relevant in constructing this application. Since typical Web pages have many different layouts and semantic information about the type and value of the content is scarce, it may be difficult to create a set of transformations that work well enough for all content of interest. For a Web content transcoding application to work well in practice, it is usually necessary to limit the viewed pages to a selected set and to provide specialized "clippers" that customize the transcoding or clipping for specified content. These provisions allow a great deal of control over how specific pages are transcoded for specific devices.

Rendering vertical XML. We expect more and more content to be published in XML dialects that are specifically designed for a given application domain. These dialects are called "vertical XML" since they are specialized for specific applications. These vertical XML dialects have a great deal of semantic information about the content, but little or no information about how it is to be rendered or displayed. Such rendering information can be defined in XSL stylesheets, and these stylesheets can be applied to the vertical XML dynamically to customize the rendering for specific devices or users at the time the content is delivered. The XSL stylesheet selection and application function discussed earlier are the key operations for this application.

Host integration. An important application of transcoding technology is to leverage the existing investment in legacy applications of an enterprise while extending its reach to pervasive devices and business partners via the Internet. Many companies execute their business processes on host systems (S/390* and AS/400*) running applications that use 3270, 5250, or VT protocols. Original legacy applications were built in an environment where users employed dumb terminals (and later, terminal emulators) to interact with host programs. The potential risks and costs prohibit companies from porting these existing applications to any new technology for "competitive advantage." Using the transcoding technology will enhance host integration solutions for use by a variety of devices and business partners.

Following are several ways to extend the reach of existing host applications to the Internet environment without modifying existing host applications.

Figure 11 Architecture of the IBM host integration solution



- Develop a downloadable host application emulator using the traditional graphical user interface (GUI) emulator. These emulators are usually Java applets[24] or ActiveX** controls[25] that run inside a desktop browser. This approach does not fit into pervasive devices that do not support Java and ActiveX control.
- Develop a customized user interface by programming against host applications. This approach is sometimes called "screen-scraping." Programmers can write code to interact with host applications and generate a customized user interface for the end user. The customized GUI could be in HTML or WML that can be supported by different devices. The disadvantage of this approach is its cost, since programs have to be written for presentations on different devices.
- Apply transcoding technology as a third approach. First, we convert host application data into a common XML format that is display-independent. Then XSL stylesheets are developed to transform the host application data to different presentation formats that fit into different devices. Figure 11 shows the architecture for this approach. Figure 12 shows an example of a screen from a host application. Figure 13 shows the same screen, transcoded for use on a WAP-enabled phone.

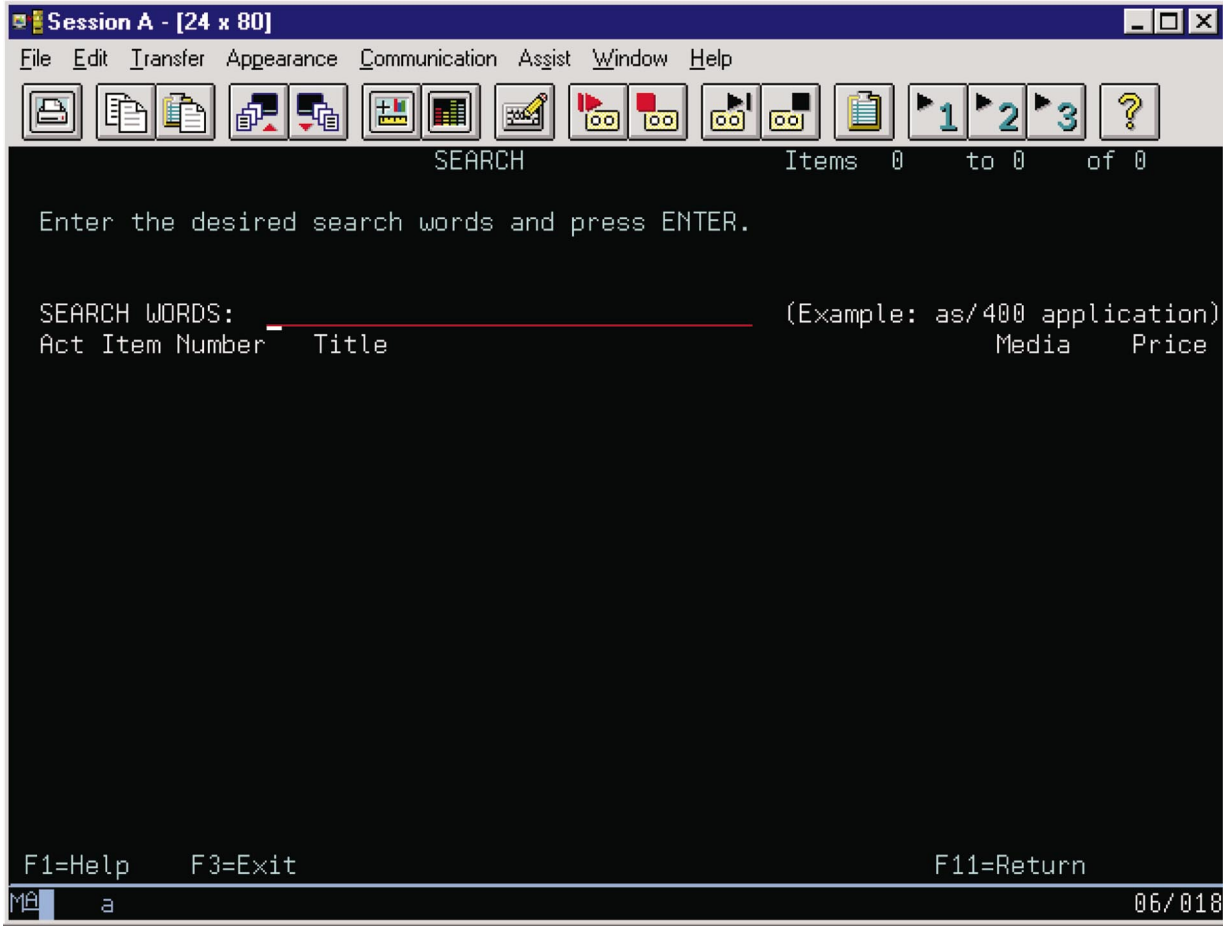
At the design time, the transcoding service is used for stylesheet association, stylesheet deployment, and stylesheet storage. At run time, the transcoding service retrieves the stylesheet and applies the transformation.

There are many advantages to incorporating transcoding technology in a host integration solution. The reengineering of host applications is needed only once. After the XML data have been generated, data transformations can be performed to generate the proper format for different client devices and business partners. Great flexibility comes from the fact that the work of extracting host content is done only once and the host application can be delivered to any type of client by applying a proper style transformation. When new devices are used to access host applications, only new stylesheets need to be developed. The result is quick delivery with a minimum development cost of legacy applications to clients using different devices or to business partners.

Performance considerations

Transcoding a document from one format to another is a computationally intensive operation. To convert HTML to WML or any other format involves parsing the HTML document. Converting an image file from one with the JPG extension (the JPEG, or Joint Photographic Experts Group, format) to one with the GIF (graphics interchange format) extension or lower resolution JPG involves reading and processing the

Figure 12 Example of an original screen of a host application



entire image and performing floating-point operations on the data.

The payoff is that Web-based files can be viewed by devices that normally would not be able to view those files. An additional benefit for image files is that the resulting file is much smaller than the original file. Sometimes the resulting image is less than one-tenth the size of the original file. This is particularly important if the user is connected over a slow wireless connection.

To handle a large number of end users, Transcoding Publisher employs a variety of methods to distribute the load and reduce the number of transforming documents that it has already transformed.

Transcoding Publisher will use an external cache if it is configured to do so. The cache interface is an industry standard, so any caching product that supports the industry standard will work. When using an external cache, the transcoder attempts to fetch the already-transformed page from the cache. If the page, already transformed for the requesting device, is found in the cache, it is returned to the requester. This fetch operation avoids the potentially costly step of converting the document. If the data are not in the cache, the act of requesting the data through the cache mechanism causes the transformed file to be stored in the cache for the next time data are requested.

Even with an external cache, there will be cache misses. A processor can become saturated transform-

Figure 13 Example of application in Figure 12 transcoded for presentation on a smart phone

ing documents. Several instances of WebSphere Transcoding Publisher can be run in conjunction with IBM Network Dispatcher,[26] which distributes the load among them. The client devices do not have to know that there is more than one transcoder in the network since they are all configured to use the same proxy Internet Protocol address. With Network Dispatcher, more transcoding computers can be added when a site adds more customers. This permits scaling in a modular fashion.

Benchmarks. There are a number of issues with using existing Web server or Web proxy benchmarks such as WebStone[27] or SPECweb**[28] to evaluate transcoding performance. Those benchmarks were developed with the wired Web and full-function browsers in mind and may not be relevant to the users of small devices with slow Internet access. The audience targeted by Transcoding Publisher is often accessing the Web wirelessly using devices that do not handle the same formats or input devices. The standard Web server and proxy measurement tools assume a certain workload or set of workloads. It is not clear that any of these workloads will be generated by users of pervasive devices. For example, the user of an Internet-capable cellular phone probably does not have the same "think time" characteristics as the user of a desktop Web browser. A desktop browser has a comparatively large screen. Therefore, there is more information to read before another request for a page is required. In contrast, it is difficult to input information on a cellular phone, which may lower the request rate.

The user of a desktop browser with high-speed Internet access often does not think twice about fetching a new page. However, the user of a wireless device will quickly realize that it takes a long time to download data from the Web. This realization is likely to affect usage patterns. For example, someone using a desktop browser will not hesitate to use a search engine to find sites of interest. The user of a cellular phone is more likely to perform one or two simple queries. Cache hit rates may also be affected by different usage patterns. For cellular phones, a larger percentage of requests is likely to be parameterized queries such as "What is the current value of xxx stock?" Since the answer to the query may change every time it is asked, caching is not usually beneficial. Therefore, the mix of simple page fetches to parameterized queries has a big impact on transcoding throughput. Transcoding is sensitive to the type of device that is requesting the information because different transformations are performed for different devices. The standard benchmark tests do not take the requesting device type into account.

The current Web measurement tools need to be changed to include sets of pages that are typically fetched by pervasive devices, an appropriate mix of queries and simple fetches, and an appropriate mix of different devices. Based on these changes, a new set of benchmarks needs to be developed to make it possible to compare competing products, which today present performance information in very different ways. In addition to standard benchmarks, capacity planning algorithms are needed in order to help network designers plan the number of servers required to support certain workloads.

Some performance measurements. Despite the lack of relevant, standardized benchmarks, it is still necessary to have some idea of the capacity of the Transcoding Publisher. To do this, the WebStone source was modified to send the HTTP headers that

Table 2 Performance measurements made on an RS/6000 44P Model 270, 375 MHz processor with 2GB RAM running AIX; Java tuning parameters: ms = 256, mx = 512

Test Case	Number of Processors	Number of Concurrent Users	Response Time	Throughput (megabits/sec)	Pages/Sec
WAP clients only (HTML to WML; 1K to 3K size text files; no images)	1	1000	4.32	.33	22.97
	2	1400	4.26	.46	32.58
	4	2000	3.93	.72	50.43
PDA clients only (HTML simplification; 1K to 3K size text files; images up to 16K)	1	1300	4.38	.61	29.44
	2	1600	4.05	.81	39.13
	4	2500	4.15	1.24	59.77

the transcoder uses to identify different devices. Workloads were also defined that might be typical for users of devices that have small screens and slow Internet connections.

In Table 2, the results for two scenarios are shown: a WAP scenario and a Palm Pilot** scenario. For the WAP scenario, all the clients are WAP phones fetching HTML pages. The HTML pages range from 1000 to 3000 bytes. No images are fetched. For the Palm Pilot scenario, all the clients are Palm Pilot devices. The same HTML pages are fetched as in the WAP scenario. In addition, some images (JPG and GIF) are fetched. They range from 500 bytes to 16000 bytes. Images are fetched one-tenth as frequently as HTML.

The table shows the number of users that can be supported while maintaining a four-second response time. All measurements were made on single-processor, and two-way and four-way multiprocessor IBM RS/6000* systems running the Advanced Interactive Executive (AIX*) 4.3.3 operating system. Version 1.1.2 of Transcoding Publisher was tested.

Because WebStone clients have no think time, the tests tend to drive processor utilization very high, and the apparent scalability is less than we expected. In other experiments we added up to 15 seconds think time between requests. For those runs, the scaling from one to two to four processors was almost linear. The results shown here are for the zero think time test. The number of concurrent users is an estimate of how many real users (who do think about or read a page before requesting another) can be using the system concurrently.

Related work

There are several commercial products and research prototypes whose capabilities are similar to that provided by IBM WebSphere Transcoding Publisher.[29] In this section we describe these systems.

The ProxiNet** transcoding engine[30] focuses on delivering existing Web (i.e., HTML) information to mobile handheld devices. It uses an ultra-thin client and relies on a middle tier to perform any necessary transcoding.[31] The ProxiNet engine can filter out content not compatible with the capabilities of a handheld device, reformat HTML tables and frames as lists, and transcode images for these devices by performing color depth reduction and scaling.

The Spyglass Prism** transcoding engine[32] also supports the transcoding of existing Web content for Web-enabled non-PC devices. The Spyglass Prism engine is server-based and translates richly formatted Web content such as tables, JPEGs, and frames into formats that match the relatively limited display capabilities of non-PC devices. The engine is also capable of transcoding images for these devices by performing color depth reduction and scaling. In addition, Spyglass Prism is capable of transcoding HTML to WML.

The Online Anywhere transcoding engine is capable of transforming content to a variety of information appliances, including Web-connected TVs, PDAs, wireless devices (pagers, data phones), and voice-only products.

A *portal* is a gateway that serves as a starting site from which a user begins Web-browsing activity. There

are general portals such as the Netscape and Yahoo!** Web sites, as well as specialized portals such as Garden.com for gardeners and Fool.com for investors. Typically, a portal has many links to many other Web sites. Portals can be a significant source of revenue because of income provided by selling advertising space (e.g., for banner advertisements) on the portal site or payments received for directing significant user traffic to other sites from the portal.

A portal can also provide a means to provide access to transcoded content. That is, it can serve as a starting point for a set of sites, all of which have been transcoded for a certain class of devices. For example, one can design a WML portal as a starting point to sites that have been transcoded for WML devices. Rather than having to transcode all possible HTML (Web) sites, one can then limit the number of pages that need to be transcoded into WML to those pages that are accessible from the portal. WTP can be used to implement such portals, for example, by using dynamic transcoding in conjunction with *text clippers* for those sites that can be accessed from the portal. There are other products in the marketplace, such as the Oracle Portal-to-Go**, that provide support for generating such portals, especially for wireless devices.

Han et al.[33] present an analytical framework for determining whether to transcode and how much to transcode an image for the two cases of store-and-forward transcoding as well as streamed transcoding, and also provide practical adaptation policies based upon transcoding delay, transcoded image size (in bytes), and the estimation of network bandwidth.

The InfoPyramid transcoding engine[34,35] is capable of transcoding content to a variety of client devices. InfoPyramid is able to perform transformations on images, video, text (i.e., HTML), and audio to reduce the amount of content sent to client devices. In addition, InfoPyramid can also perform modality-based transformations. InfoPyramid can convert video to images, video to audio, video to text, text to audio, and audio to text.

Conclusions and future work

As new types of devices become more prevalent, the need to adapt existing content for display on a variety of devices is becoming greater. The techniques presented in this paper are valuable in constructing solutions that reuse existing content for the new applications that are made possible by the new device and network types. We expect these technologies to continue to evolve to address more applications and environments and to make it easier to customize the solutions. The extensive use of standards such as HTML, XML, and the Java language has been a key factor in enabling us to quickly build and deliver a product—WebSphere Transcoding Publisher—that incorporates these technologies and delivers on the promised value.

There are several aspects of WebSphere Transcoding Publisher that are excellent avenues for future work. Transcoding Publisher currently does not provide automatic conversion of HTML content to a format suitable for voice-recognition-based browsers. Providing this type of function (e.g., HTML to VoiceXML conversion) would enable Transcoding Publisher to reach a greater range of devices.

The ability of Transcoding Publisher to transform content in a variety of ways could be leveraged to adapt content to enable it to be more suitable for users who have accessibility issues. For example, Transcoding Publisher could be used to increase the font size of content to enable it to be read by users with limited vision capabilities, and content could be modified such that it was presented in a simpler form, thus making it suitable for users with attention deficit disorder or dyslexia.

The clipping and transformation capabilities of Transcoding Publisher are well suited for the creation of portals for wireless devices. Adapting Transcoding Publisher to serve in this capacity is a valuable avenue for related work.

Transcoding Publisher currently relies upon user agent identifiers transmitted by browsers to determine the capabilities of the client. As more formal methodologies for sending information regarding client capabilities such as CC/PP mature, the policies of Transcoding Publisher for determining how to transcode for a device should be updated to exploit this detailed description of capabilities.

Transcoding Publisher could be extended to dynamically adapt its transcoding policies as the environment in which it is being used changes over time. For example, image reduction policies could change when network bandwidth is suddenly reduced.

Finally, Transcoding Publisher needs to be extended to support emerging formats such as the MPEG-7 (Moving Picture Experts Group) multimedia format

to allow it to continue to provide first-class support for cutting-edge wireless devices.

In addition to functional enhancements, it would be useful for comparing the performance of different products if standardized performance benchmarks for mobile devices accessing the Internet were developed.

Acknowledgments

The authors acknowledge the large international team that designed, developed, tested, documented, and marketed WebSphere Transcoding Publisher, including the core IBM team in Research Triangle Park, North Carolina; an IBM team in Austin, Texas, led by Matt Rutkowski that developed the caching support, especially Andrew Hately who endured four snowed-in days in North Carolina to make sure it worked; the IBM Tokyo Research team that developed the HTML parser vital to the HTML to WML transcoding, especially Goh Kondoh and Shinichi Hirose; the IBM Hawthorne Research team that contributed the image transcoding technology, especially John R. Smith, Chung-Sheng Li, and Richard LaMaire; the Lotus XSL team that developed the Xalan XSL transformation engine, especially Scott Boag who also worked with our customers to improve their stylesheets; and the IBM Almaden Research team led by Paul Maglio and Rob Barrett that contributed the WBI framework at the core of WebSphere Transcoding Publisher and is still working with us to enhance it. We thank every individual on this extended team for an exceptional degree of long-distance cooperation.

Cited references and notes

1. Java, Sun Microsystems, Inc., http://java.sun.com.
2. XML, World Wide Web Consortium, http://www.w3.org/XML.
3. K. F. Eustice, T. J. Lehman, A. Morales, M. C. Munson, S. Edlund, and M. Guillen, "A Universal Information Appliance," *IBM Systems Journal* **38**, No. 4, 575–601 (1999).
4. Compact HTML for Small Information Appliances, World Wide Web Consortium, http://www.w3.org/TR/1998/NOTE-compactHTML-19980209/.
5. Proposal for a Handheld Device Markup Language, World Wide Web Consortium, http://www.w3.org/TR/NOTE-Submission-HDML.HTML.
6. Wireless Markup Language is part of the Wireless Application Protocol standard from the WAP Forum, http://www.wapforum.org.
7. S. Chandra, C. S. Ellis, and A. Vahdat, "Differentiated Multimedia Web Services Using Quality Aware Transcoding," *INFOCOM 2000—Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies* (March 2000), http://www.cs.duke.edu/~surendar/infocom00.pdf.
8. IBM WebSphere Application Server, IBM Corporation, http://www.ibm.com/software/webservers/appserv/.
9. T. F. Abdelzaher and N. Bhatti, "Web Content Adaptation to Improve Server Overload Behavior," *8th International World Wide Web Conference* (1999). http://www8.org/w8-papers/4c-server/web/web.pdf.
10. F. Kitayama, S. Hirose, and K. Kuse, "Dharma: A Framework for Development of Web Applications for Pervasive Terminals—Systems Overview and Application Objects," *IPSJ 57th Annual Convention*, in Japanese (1998).
11. S. Hirose, F. Kitayama, and K. Kuse, "Dharma: A Framework for Development of Web Applications for Pervasive Terminals—View Object Generation and HTML Generation Mechanism," *IPSJ 57th Annual Convention*, in Japanese (1998).
12. F. Kitayama, S. Hirose, and K. Kuse, "Design and Implementation of Web-Application Development System for Business Objects and Pervasive Terminals," *IPSJ '98 Object Oriented Symposium*, in Japanese (1998).
13. F. Kitayama, S. Hirose, K. Kuse, and G. Kondoh, "Design of a Framework for Dynamic Content Adaptation to Web-Enabled Terminals and Enterprise Applications," *IPSJ/IEEE APSEC '99* (December 1999).
14. IBM WebSphere Transcoding Publisher, IBM Corporation, http://www.ibm.com/software/webservers/transcoding/.
15. The Apache Software Foundation, http://xml.apache.org.
16. WAP Forum, http://www.wapforum.org.
17. Document Object Model, http://www.w3c.org/DOM.
18. Currently WTP does not have a mechanism to identify individual users. In the future, WTP will be used in environments where user identification is possible. In this case, preferences from the user source will also be used.
19. The NetRexx Language, IBM Corporation, http://www2.hursley.ibm.com/netrexx/.
20. R. Han, "Factoring a Mobile Client's Effective Processing Speed into the Image Transcoding Decision," *WOWMOM '99* (August 1999).
21. A. Fox, S. Gribble, Y. Chawathe, and E. Brewer, "Adapting to Network and Client Variation Using Infrastructural Proxies: Lessons and Perspectives," *IEEE Personal Communications* **5**, No. 4, 10–19 (August 1998).
22. CC/PP, World Wide Web Consortium, specifications at http://www.w3.org/TR/NOTE-CCPPexchange; http://www.w3.org/Mobile/IG.
23. R. Barrett and P. Maglio, "Intermediaries: An Approach to Manipulating Information Streams," *IBM Systems Journal* **38**, No. 4, 629–641 (1999).
24. *IBM SecureWay Host On-Demand 4.0: Enterprise Communications in the Era of Network Computing*, SG24-2149-01, IBM Corporation.
25. Attachmate Corporation, http://www.attachmate.com.
26. IBM Network Dispatcher; IBM Corporation, http://www.ibm.com/software/network/dispatcher/.
27. WebStone benchmark, http://www.mindcraft.com/webstone/.
28. SPECweb99, http://www.specbench.org/osg/web99/.

29. R. C. Henderson, B. Topol, C.-S. Li, R. Mohan, and J. R. Smith, "Taxonomy of Network Transcoding, in Multimedia Computing and Networking 2000," K. Nahrstedt and W.-C. Feng, Editors, *Proceedings of SPIE* **3969**, pp. 65–72 (2000).
30. ProxiNet, http://www.proxinet.com/technology/, now part of Puma Technology, Inc., http://www.pumatech.com.
31. A. Fox, S. D. Gribble, E. A. Brewer, and E. Amir, "Adapting to Network and Client Variability via On-Demand Dynamic Distillation," *ASPLOS-VII*, Cambridge, MA (October 1996).
32. Spyglass, Inc., http://www.spyglass.com/, now part of OpenTV, Inc., http://opentv.com/.
33. R. Han, P. Bhagwat, R. LaMaire, T. Mummert, V. Perret, and J. Rubas, "Dynamic Adaptation in an Image Transcoding Proxy for Mobile WWW Browsing," *IEEE Personal Communications Magazine* **5**, No. 6, 8–17 (December 1998).
34. J. R. Smith, R. Mohan, and C.-S. Li, "Transcoding Internet Content for Heterogeneous Client Devices," *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*, special session on Next Generation Internet (June 1998).
35. J. R. Smith, R. Mohan, and C.-S. Li, "Content-Based Transcoding of Images in the Internet," *Proceedings of the IEEE International Conference, Image Processing (ICIP-98)* (October 1998).

Accepted for publication September 22, 2000.

Kathryn Heninger Britton *IBM Application Integration Middleware Division, P.O. Box 12195, Research Triangle Park, North Carolina 27515 (electronic mail: brittonk@us.ibm.com).* Ms. Britton is a Senior Technical Staff Member in the Application Integration Middleware Division. She is the technical leader of the worldwide research and development team that produced the IBM WebSphere Transcoding Publisher product, first shipped in March 2000. Since joining IBM in 1981, she has contributed to IBM's network architecture, with a focus on two-phase commit protocols and multiprotocol networking. She has also worked on the development of several software products for networking and mobile computing. She has served as IBM's representative to the X/Open XNET working group. Prior to joining IBM, Ms. Britton worked for the Naval Research Laboratory on software engineering research and technology transfer. She holds a bachelor's degree in English from Stanford University and two master's degrees, one in computer science from the University of North Carolina at Chapel Hill.

Ralph Case *IBM Software Group, 4205 South Miami Boulevard, Research Triangle Park, North Carolina 27709 (electronic mail: caser@us.ibm.com).* Mr. Case is a senior programmer developing transcoding strategy and technology. He also works on APPN® architecture and chairs the APPN Implementers Workshop (AIW). Previously, he helped develop S/390 hardware, specializing in channel subsystems, ESCON®, and total systems test. He has experience building automated tools and cooperative processing systems using communications protocols, knowledge-based systems, and object-oriented programming. He holds a B.S. in electrical engineering from Rensselaer Polytechnic Institute.

Andrew Citron *IBM Software Group, 4205 South Miami Boulevard, Research Triangle Park, North Carolina 27709 (electronic mail: citron@us.ibm.com).* Mr. Citron is currently the team lead for the Transcoding Publisher and Host Publisher performance team. He has also been a software developer on Web Express and Mwave®. Prior to that he was lead architect for IBM's APPC architecture.

Rick Floyd *IBM Software Group, 4205 South Miami Boulevard, Research Triangle Park, North Carolina 27709 (electronic mail: raf@us.ibm.com).* Dr. Floyd received the B.S. degree from Iowa State University in 1977 and M.S. and Ph.D. degrees from the University of Rochester in 1982 and 1989. He was a member of the technical staff at the Clinton P. Anderson Meson Physics Facility from 1978 to 1981, and at BBN Laboratories Incorporated from 1987 to 1989. He has been with IBM in Research Triangle Park since 1989. His research interests include distributed systems and mobile computing.

Yongcheng Li *IBM Application Integration Middleware Division, 4205 South Miami Boulevard, Research Triangle Park, North Carolina 27709 (electronic mail: ycli@us.ibm.com).* Dr. Li is a software engineer in the Advanced Design and Technology Group of the Application Integration Middleware Division. He currently works on XML-based application integration and data transformation. After joining IBM in 1997, he worked on Web-host integration. Dr. Li received his Ph.D. degree in computer science from Tsinghua University in 1993. He is a coinventor on 16 filed patents.

Christopher Seekamp *IBM Software Group, 4205 South Miami Boulevard, Research Triangle Park, North Carolina 27709 (electronic mail: seekamp@us.ibm.com).* Mr. Seekamp is a programming consultant and has been the primary developer and development team leader for the text-related portions of the WebSphere Transcoding Publisher since early in its development. He has held numerous key positions working on the development of the various communications software projects. For most of the last several years he has worked on issues dealing with providing content to various types of mobile devices, both inside and outside of IBM. Over 10 years ago, Mr. Seekamp was an early adopter of object-oriented design and development techniques and has been employing object-oriented design and development approaches in his work ever since.

Brad Topol *IBM Software Group, 4205 South Miami Boulevard, Research Triangle Park, North Carolina 27709 (electronic mail: btopol@us.ibm.com).* Dr. Topol is a member of the AIM/Business Connection Advanced Technology Group at IBM in Research Triangle Park. He received the B.S. and M.S. degrees in computer science from Emory University in 1993 and the Ph.D. degree in computer science from the Georgia Institute of Technology in 1998. Currently, he is actively involved in advanced technology projects in the areas of content adaptation, distributed systems, networking, and graphical user interfaces.

Karen Tracey *IBM Software Group, 4205 South Miami Boulevard, Research Triangle Park, North Carolina 27709 (electronic mail: kmt@us.ibm.com).* Dr. Tracey received B.S., M.S., and Ph.D. degrees in electrical engineering from the University of Notre Dame in 1987, 1989, and 1991, respectively. She joined IBM at Research Triangle Park in 1991. She has worked in development for various products, most recently for Communications Server/390 and WebSphere Transcoding Publisher.