IBM eNetwork Host On-Demand: The beginning of a new era for accessing host information in a Web environment

by Y. S. Tan
D. B. Lindquist
T. O. Rowe
J. R. Hind

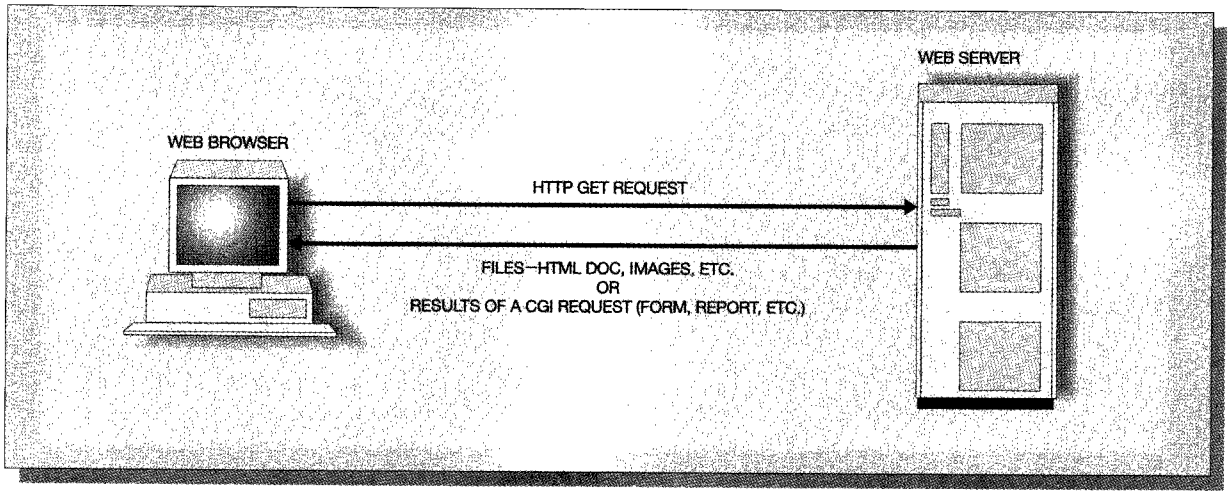
This paper describes software technology that makes it possible to run host applications seamlessly in a World Wide Web environment. Web technology has opened up the potential for unprecedented access of data and applications by all types of users. Yet, the incompatibility between host technology (e.g., 3270) and Web technology (e.g., HTTP) has prevented Web users from directly accessing hundreds of thousands of host applications. The cost for re-implementing these applications would be enormous, and the effect to ongoing business would be disruptive. Host On-Demand extends Web access to reach existing host applications directly by using Java™ technology and provides a number of powerful features including: emulation functions on demand, persistent connections, customized session windows, multiple sessions, platform flexibility, and host security. This paper summarizes these features, describes Java implementation techniques, and outlines the need for some new network computing services.

Web browsing is a simple client model that provides great flexibility and robust access from many different sources. Because information is published in the format of a Web page, how to include the mass of valuable information sitting on various host systems has been a major concern in the industry. Until recently, there were two general approaches. One packages a 3270 emulator with the browser. The other uses mapping software on the Web server to convert the 3270 data stream to HyperText Markup Language (HTML) format. Sun's Java** technology makes a third approach possible: using a Java 3270 emulation applet from a browser. This paper describes IBM's Host On-Demand Version 1, a Java applet emulator that lets users access host information from a browser. It also explains how Host On-Demand represents a new generation of networking tools and capabilities provided by IBM's eNetwork* software business.

The Web browser model

Web technology[1] is based on HTML[2] and the HyperText Transfer Protocol (HTTP).[3] HTML provides a common representation for information, and HTTP defines the common protocol for transferring information between Web clients and Web servers. The Web browser serves as the end-user interface; it is responsible for sending user requests to the appropriate Web server (normally via a Web proxy gate-

Figure 1 Web-browsing HTTP protocol



way) and for formatting and displaying HTML data streams returned to the client device.[4]

Normally (see Figure 1), a Web browser communicates directly with a Web server (or proxy server) over a Transmission Control Protocol (TCP) connection using HTTP. The user specifies a uniform resource locator (URL) to address the object requested. This object may be a stored HTML text document or an HTML data stream that is generated by a program. In the latter case, the Web server invokes the program via the Common Gateway Interface (CGI) that is defined as part of the Web technology. The HTML object returned to the Web browser client may contain hyperlinks to other HTML objects and directly embedded graphic (e.g., GIF or JPG) objects. It is the responsibility of the browser to issue additional requests for the embedded objects on behalf of the end user until the entire document is complete.

If there is a need to access host information, the user must open a tn3270 emulator window, separate from the browser. Alternatively, the user could open a Web page provided by 3270/HTML mapping software. Let us examine these alternatives.

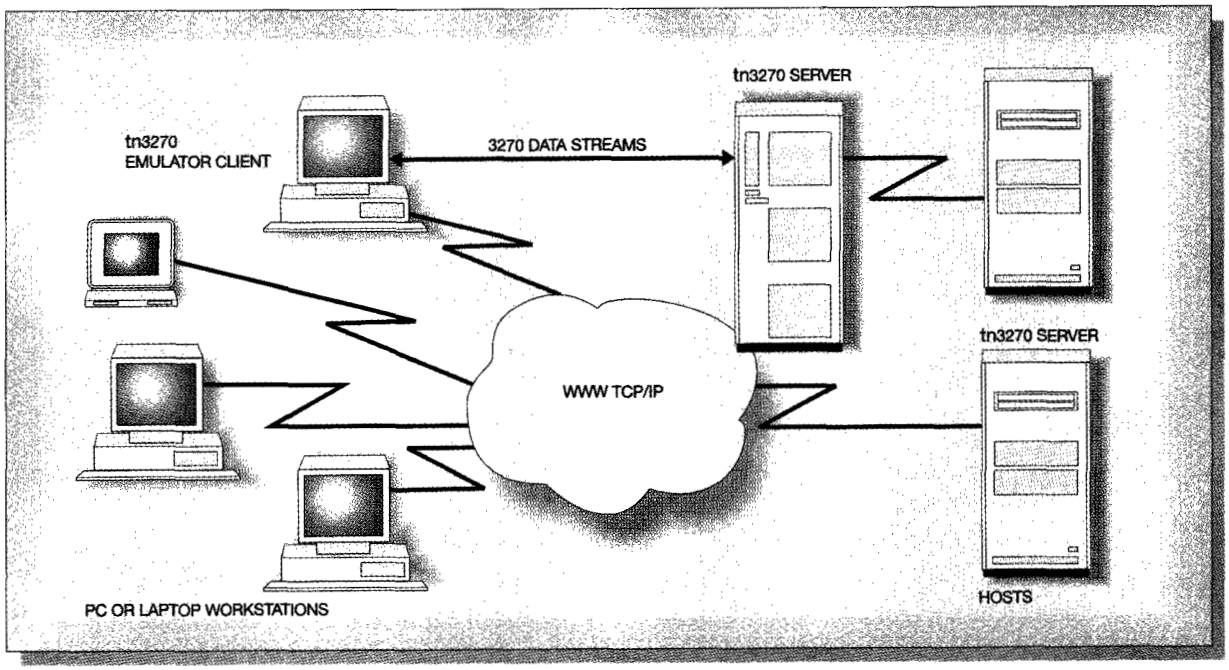
The stand-alone tn3270 emulator model

The stand-alone tn3270 emulator (see Figure 2) is PC or workstation software installed locally on a client. The emulator starts as a Windows** application and communicates with the tn3270 server over a TCP connection using the Telnet protocol.[5] The user enters a command or presses a keyboard function key from a machine with an emulated terminal screen. The emulator constructs a 3270 inbound (to host) data stream to send to the tn3270 server over the TCP connection. The tn3270 server unwraps the data stream from the TCP transport format into a Systems Network Architecture (SNA) format and forwards the SNA transmission unit to the host. The host responds with an outbound data stream, and the reverse takes place. Some hosts have a tn3270 server installed as part of the Transmission Control Protocol/Internet Protocol (TCP/IP) connectivity software. In that case, the TCP transport will be used end to end, and SNA conversion is done only inside the host.

The stand-alone tn3270 emulator model has the following characteristics:

- *High performance:* 3270 is an efficient data stream designed to meet the subsecond response time objective (and support many hundreds of simultaneous users).
- *No browser integration:* The tn3270 emulator runs under a separate process in a separate window from the browser. There is no interaction between the browser and the emulator (beyond cut and paste functions).
- *Persistent session connections:* Unlike the Web model, the emulator is session-based. The emu-

Figure 2 Stand-alone tn3270 emulator access to hosts



lator establishes a session from the client to the host that persists until the user logs off.

- *Productivity:* 3270 applications leverage a wide range of graphical user interfaces (GUIs), keyboards, and macros to enhance productivity (but do not match windows GUI models).
- *Traditional client/server model:* It requires individual client product installation, configuration, software distribution, upgrade maintenance, and unique platform dependencies.
- *Host security:* The traditional user log-on password and Resource Access Control Facility (RACF*) protections are used (separate from any client or network passwords or security).

The 3270/HTML mapper model

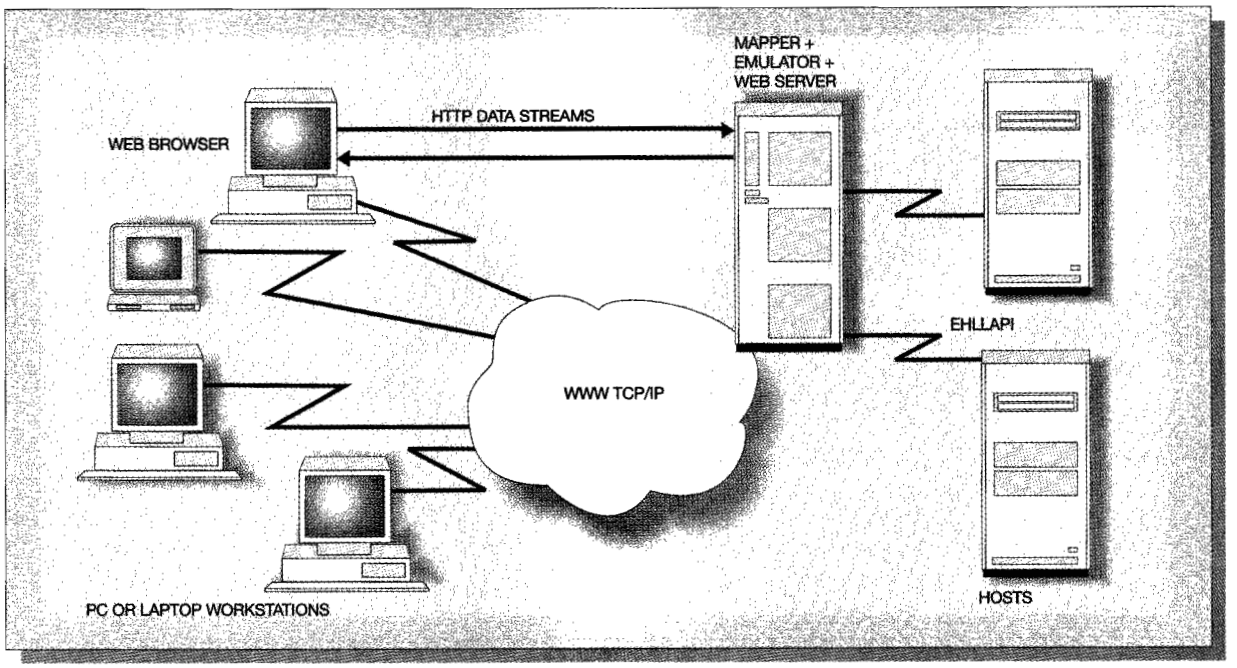
The 3270/HTML mapper (see Figure 3) is intermediate software installed on a Web server that generates HTML pages on the fly from 3270 data streams exchanged between the Web server and a host,[6] typically implemented using HTML, HTTP, and CGI scripts.

The Web browser model as illustrated by Figure 1 is preserved. The Web browser user specifies a URL to request host access. This URL points to an HTML data stream that is generated by the mapper program. The Web server invokes the program via the CGI interface. The program sits on an emulator session established for the user from the server to the host. It interacts with the 3270 host screen using the Emulator High-Level Language Application Programming Interface (EHLLAPI). The host screen output is used to generate an HTML form to return to the Web browser client.

The 3270/HTML mapper model has the following characteristics:

- *Performance:* The HTTP protocol overhead and 3270/HTML mapping function insert overhead and increase response time.
- *Full browser integration:* The host information is returned in HTML format and acts no differently than other Web pages.
- *Nonpersistent session connections:* The Web page can be dropped through normal Web navigation. Users can lose the state of their host sessions easily. They must also start multiple browsers to have both Web and 3270 access.

Figure 3 3270/HTML mapper



- *Loss of 3270 functionality:* The browser does not handle keyboard function keys and does not receive real-time screen updates from hosts. It also cannot handle all the 3270 data attributes host applications use.
- *Web-centric:* The browser is the universal user interface for network access to information.
- *Browser security:* The secured Web browser and server provide user authentication and encryption to prevent outside intrusion of sessions opened on the server (though separate log-on IDs are still needed).

The 3270/HTML mapper function provides a subset of useful emulation function through a browser, at the cost of performance and loss of 3270 functionality.

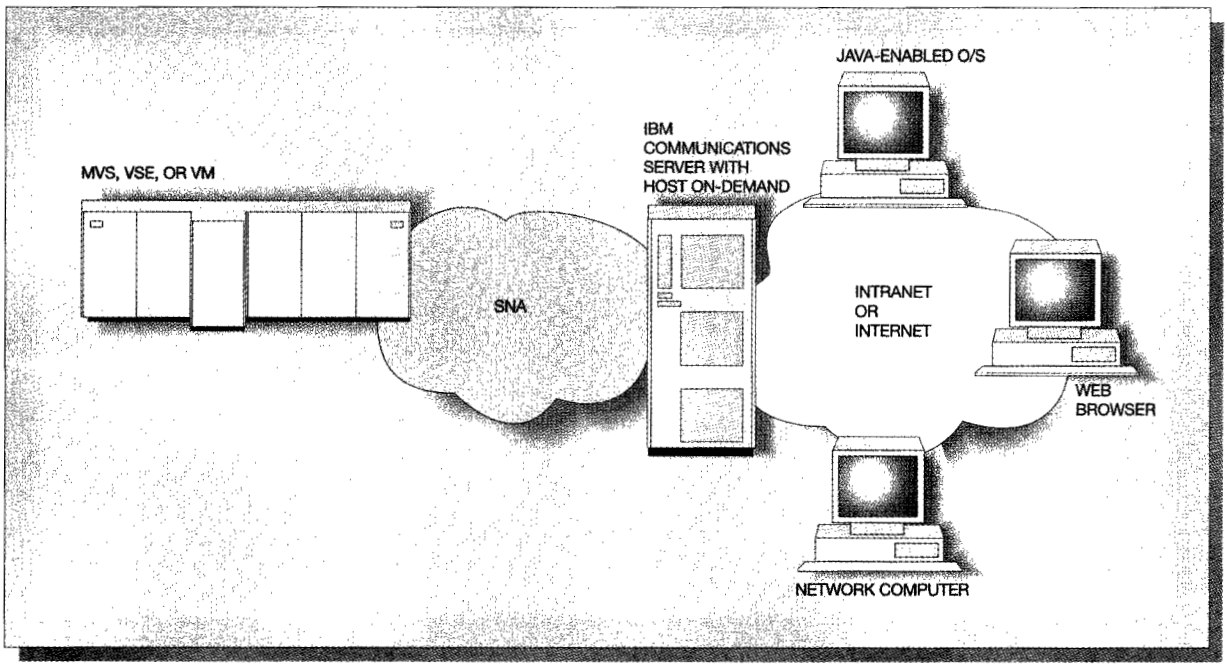
The 3270 Java applet model

One of the most exciting Web technologies is Java and the Java applet model. In this model, an application written in Java can be downloaded to a Web user's machine and started, while still maintaining a secure link back to the data server. This enables information servers to dynamically deliver their information to Web users, independent of the unique application protocols. Recognizing this capability, we can blend terminal emulation with Web browsing without sacrificing the performance and functionality of the 3270 applications.

The Java applet model gives software developers a variety of options to incorporate 3270 data stream processing into the Web HTTP-based processing. By embedding Java applet tags[7] in an HTML document, Java code segments can be pulled dynamically from the Web server to heterogeneous clients to provide interoperation with server applications. Pieces of traditional desktop emulation software can be implemented as Java applets and downloaded to the Web user's computer when the user needs to access host information. These applets can be as simple as remote GUIs. Or an applet can handle complete 3270 data stream processing, like IBM's Host On-Demand (see Figure 4).

IBM's Host On-Demand[8] is an Internet-to-SNA interconnectivity solution that provides 3270 application access through the World Wide Web (WWW). IBM communications servers maintain Host On-De-

Figure 4 3270 Java applet

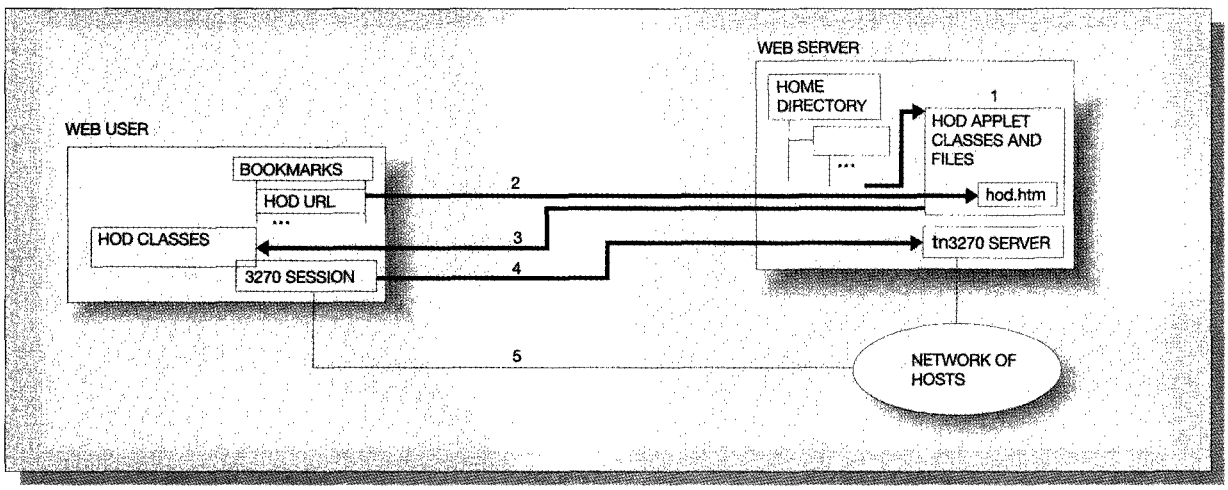


mand, including new releases or upgrades. Web users need not worry about typical installation, configuration, distribution, and maintenance problems.

Host On-Demand has the following characteristics:

- *High performance:* It uses the native Telnet protocol, thus eliminating the overhead typically associated with HTML and HTTP; resulting response time is close to the native emulator.
- *Emulation function on demand:* Users need no additional software on their computers. Host On-Demand dynamically downloads its Java-based 3270 emulator to a user's computer.
- *Persistent session connections:* It provides a true, bidirectional session between the Web user and 3270 host applications, eliminating the interruptions from navigating Web pages. Users receive real-time host screen updates and can use their keyboard, PF keys, and mouse.
- *Customized session windows:* It provides the choice of opening 3270 windows within an existing Web-browser window or as a new window. A user can customize the appearance of his or her 3270 window. A default window setting, menu bar, iconic toolbar, keypad, and automatic scalable fonts are provided.
- *Multiple sessions:* It provides multiple concurrent host access sessions.
- *Platform flexibility:* Host On-Demand supports any Java-enabled client platform such as Windows 3.x**, Windows 95**, Windows NT**, Operating System/2* (OS/2*), Advanced Interactive Executive (AIX*), UNIX**, and Mac**. It can be downloaded from any standard Web server such as Windows NT, AIX, OS/2, MVS, Netware**, and UNIX.
- *Host security:* Persistent connections deny unauthorized connection to an authorized session, such as the exposure found in the HTML mapper or remote GUI implementations. Traditional host computer mechanisms such as RACF and Advanced Communications Function/2 (ACF2) are available.
- *Investment protection:* It leverages existing client-side computing power to deliver host application access and leverages customer investment in distributed computing power. Contrast it with the HTML mapper (and some Java clients) approach that uses server cycles to provide host application access.

Figure 5 Establish persistent session



Host On-Demand implementation of the Java applet model

Host On-Demand is implemented using Java technology and its applet download capability. Traditional 3270 emulation client function is completely re-implemented in Java as a Java applet. A new processing structure breaks the traditional model into several discrete, self-contained objects. These objects are implemented as Java classes, and the entire class library is installed on a server path pointed to by the Web server home directory structure. The Web administrator sets up a URL pointing to a default Host On-Demand applet initial HTML page. When Web users open this URL, the server initiates the download of Host On-Demand Java-class files that run in the Web user's computer memory to automatically establish a session from Web user to host. The session has the same end-to-end persistent characteristics as those established from a resident emulation package to conventional tn3270 servers. Host On-Demand requires no modifications of the network or server environment.

Figure 5 illustrates the flows involved in establishing persistent sessions. The numbers in parentheses here correspond to the numbers shown in the figure. The Host On-Demand Java applet is stored on the server at a resource location (1). A user selects this URL (2) to cause the Host On-Demand applet download and execution (3) on the user's machine. This opens a socket (4) and establishes a persistent end-to-end session to a host (5).

Figure 6 depicts the main components of Host On-Demand. At the center is the core process of host access that establishes the session. The GUI processing and run-time environment processing are logically separated from the host access session processing. The GUI processing (on the right) consists of several classes used to construct a visual display of the session on the user's workstation. The run-time environment processing (on the left) provides a default run-time control that initiates sessions, maintains session limit, tracks user GUI options, and resolves file references. The entire solution is based on Java technology, and the same piece of code runs on any Java-enabled platform. These components are further discussed in the following sections.

Core 3270 emulation

Figure 7 shows the structure of the main classes of the "host access" component (center column of Figures 6 and 7). At the bottom is the transport class, which is responsible for session negotiation, termination, and flowing of 3270 data streams over sockets. Outbound 3270 data streams (from host to user) are parsed in the data stream class into discrete pieces and passed on to the presentation space class that processes these pieces and maintains a virtual 3270 screen in memory, storing all data fields and attributes. The presentation space class also takes user input, updates the virtual screen, and initiates inbound 3270 data stream flows (from user to host). Inbound data streams are assembled in the data

Figure 6 Host On-Demand main components

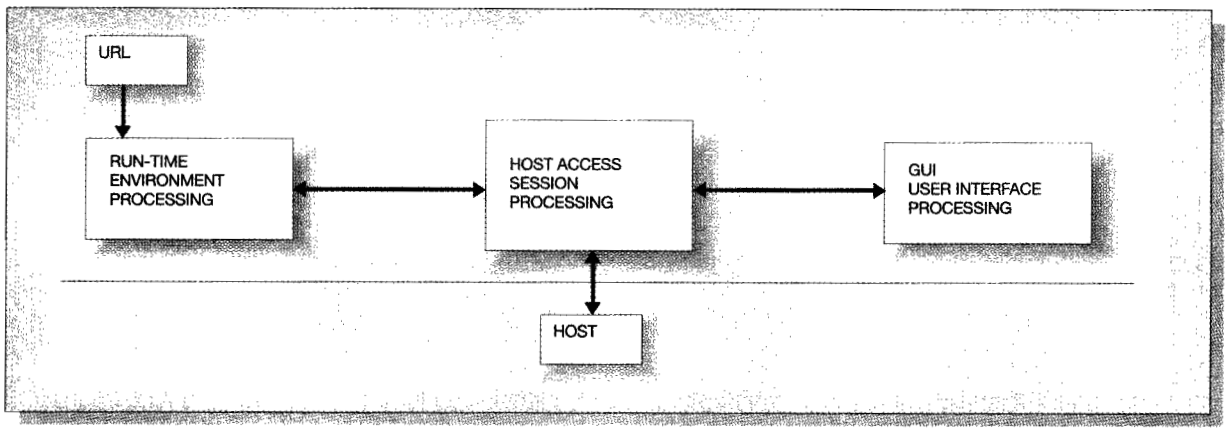
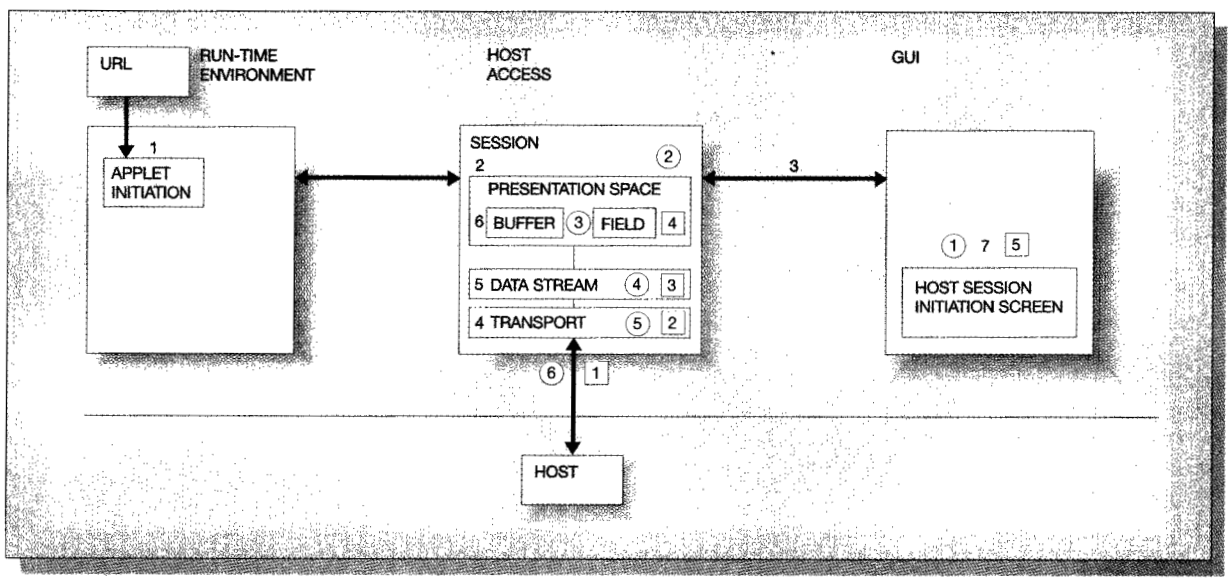


Figure 7 Host access detailed flows



stream class with information from the presentation space class, and passed on to the transport class. These classes are glued together by the session class to represent the complete state of a session. Multiple host connections are created by multiple instantiations of the session class. The result is a self-contained session object structure that forms the basis of a persistent session from the user to host.

Figure 7 also illustrates several detailed flows. First described is the detailed flow of session initiation, where the plain numbers in the figure correspond to the numbers in parentheses here. A user opens the URL to initiate applet download and start execution (1). The applet instantiates the session class, which in turn instantiates the host access classes (2), and instantiates and attaches GUI classes (3). The transport class instantiation opens a socket connection to the tn3270 server port, sets up a continuous socket read loop, and performs session negotiation (4). When the negotiation is complete, the host sends a "welcome to host" logo data stream that is passed

on to the data stream class (5). The data stream class parses the data stream and invokes presentation space methods to store the information in the presentation space class structures (buffer and field) (6). When the data stream processing is complete, the presentation space class invokes the GUI processing to paint the "welcome to host" logo screen (7). The session initiation is then complete. Host On-Demand is now ready to handle user input and host application output.

Before getting into more discussions of the input and output processing, it is necessary to mention briefly the 3270 data stream architecture. The 3270 data stream[9] is a well-defined architecture transmitting data between an application program and a display with a keyboard, typically used by mission-critical business applications. The data stream is further divided into outbound and inbound formats. An outbound data stream is sent from the application program to the display device and consists of commands, orders, control characters, attributes, and data. An inbound data stream is sent from the display device to the application program and consists of an attention identifier (AID) followed by data. The nondata portion of the data stream defines the action to be performed on the data.

An example of an outbound data stream for writing a message to the bottom of a 3270 screen may look like this: an Erase/Write command to clear the screen and write new data, followed by a write control character to reset the keyboard after the write operation is complete, followed by an order to set the starting position to the last line of the screen, followed by another order to create a field at the starting position with field attributes (input and displayable), and finally the message character string to be written. Note that every piece of required information is packed in the data stream for the device to process in one shot to display on the screen. In contrast, an HTTP stream may traverse back and forth between client and server several times when displaying a single page. This is why 3270 host technology is so efficient and has been in good demand for driving critical business applications.

In Figure 7 the detailed flow of inbound 3270 data traffic is shown, with the circled numbers in the figure corresponding to the numbers in parentheses here. The user enters data and a function key from the screen (1), which are routed to the host access session class and handed over to the presentation space class (2). The presentation space class updates the buffer and field classes and invokes the data stream class for inbound send (3). The data stream class assembles the inbound 3270 data stream (4), then invokes the transport class to send it as a socket (opened for this session) output stream (5) that flows to the host (6). The tn3270 server at the other end receives it as a socket input stream and extracts the 3270 data stream to pass it on to the host application.

Figure 7 also illustrates the detailed flow of outbound 3270 data traffic, the same processing as painting the "welcome to host" logo screen, except that the data streams could be more complicated. The numbers in squares in the figure represent the outbound flow, in which the host application sends a data stream over the socket opened for this session to the user workstation (1), which is received by the transport class (in its continuous read loop) (2). The transport class extracts the 3270 data stream and invokes the data stream class for an outbound receive (3). The data stream class parses the 3270 data stream by looping through the 3270 commands, orders, control characters, attributes, and data sequences and stores the results in the presentation space classes (4). Since multiple 3270 data streams could be embedded in a single socket stream, the presentation space class invokes the GUI processing at the end of each 3270 data stream to repaint the host application output screen (5).

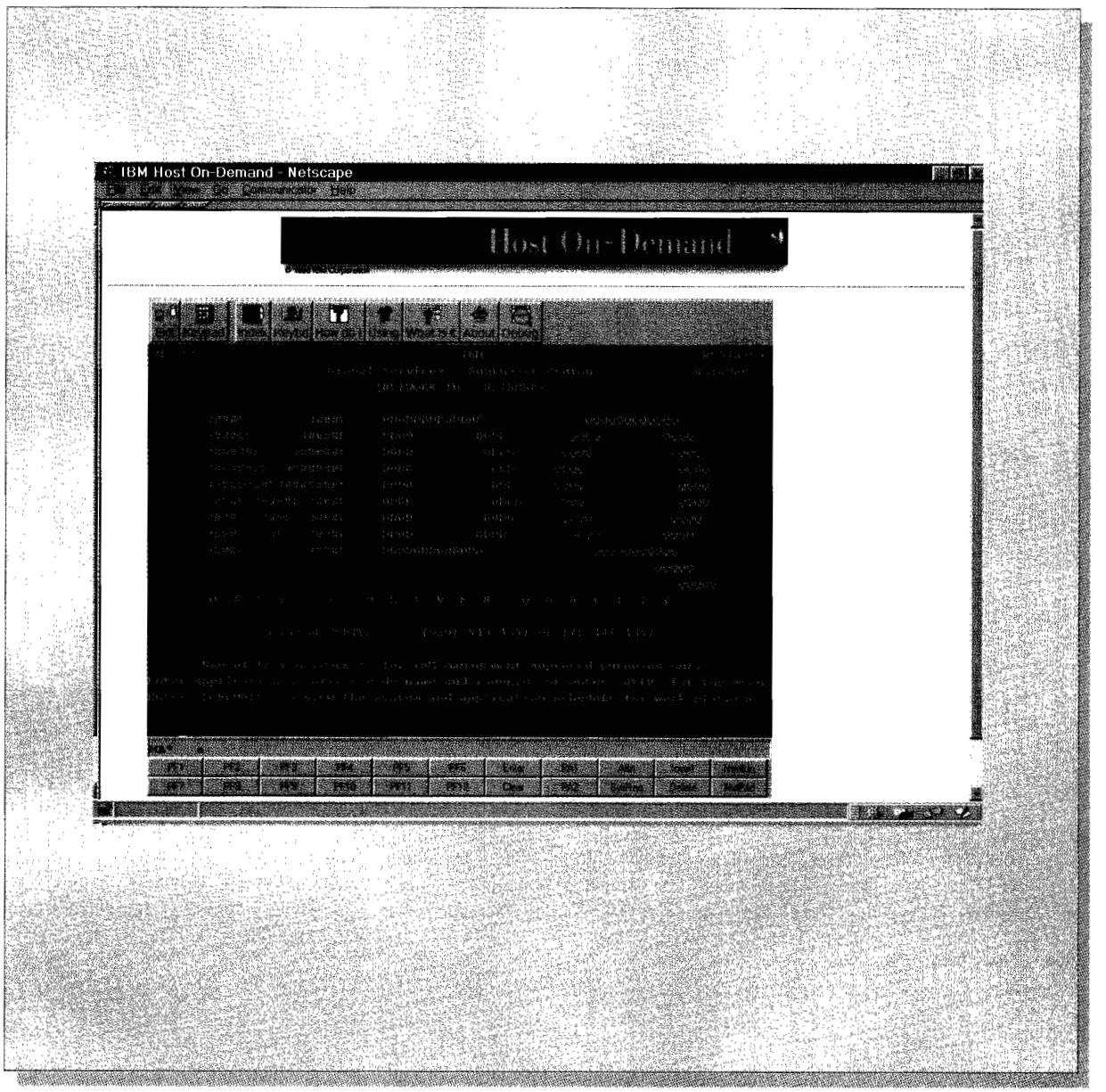
Concurrent sessions are supported via multithreading. Each transport instance runs on its own thread. Therefore, the user will not be tied up by a single host data traffic path when working with multiple hosts simultaneously.

By separating the core host access processing from other aspects, Host On-Demand has paved the way for new Web applications to use host information in a way not previously used. The IBM eNetwork Host On-Demand Host Access Class Library is provided in Host On-Demand Version 2,[10] which started shipping in September 1997. Customers can write their own Java applets and applications to interact with host applications without displaying 3270 screens.

3270 GUI

Host On-Demand GUI consists of an applet initiation (HTML) page and a 3270 screen display window. The applet initiation page contains simple options that a user can select to start a host session. The 3270

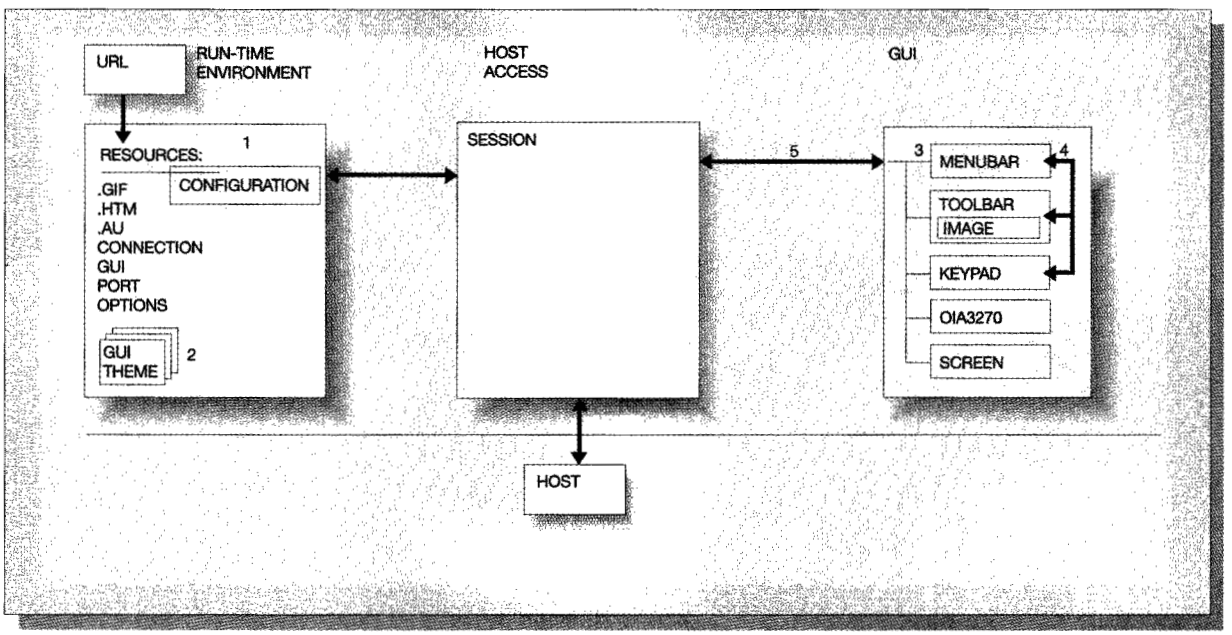
Figure 8 Host On-Demand with Netscape Navigator 3.0



screen display can be opened in its own window or in the window of the browser. If it is in its own window, the applet initiation page sets up a hyperlink pointing to the 3270 screen display window. Figure 8 is a Host On-Demand screen shot when it is opened as an embedded window inside a Netscape** browser.

The 3270 screen display window is similar to the IBM Personal Communication (PCOMM) product. It consists of a menubar, toolbar, keypad, OIA, and screen. OIA, the operator intervention area, is a specific term used in terminal emulation software for displaying session status, such as when the system is in a wait

Figure 9 Theme-based GUI processing



state or inhibited for input. Users can control the display of the toolbar and keypad, size the screen window, duplicate a session window, and jump between session windows. All GUI elements are implemented in the Java AWT (Abstract Windowing Toolkit) classes. The session class provides the container for holding the GUI subcomponents. It calculates the dimension required for each subcomponent that is showing and aligns it properly in its resize() method.

Host On-Demand contemplates the fact that users may prefer to use the interfaces they are familiar with. For example, browser users may not be familiar with the green-screen look and feel of the emulator interface and instead prefer a browser-like interface. The GUI processing allows easy OEM (original equipment manufacturer) modifications by employing "theme-based" GUI class structures. The term "theme-based" means that GUI elements such as menubar items, toolbar icons, and coloring schemes are set according to the style and terminology of a specific application, such as PCOMM. The Host On-Demand default is an emulator "theme-based" GUI modeled after PCOMM.

A real example of the application of this concept is the Netscape theme-based GUI shipped with the version of Host On-Demand in Netscape Communicator Professional Edition**.[11] It uses a different set of menu item names and toolbar icons and shows fewer toolbar items. The GUI is based on Netscape's user interface conventions. The specific theme is defined in a Java class that maps GUI elements to internal GUI processing function calls. (It could be a file, but Host On-Demand chooses to use Java class.) The map also includes settings to control the enabling and disabling of GUI elements. The function calls remain constant. The theme can be mapped to a subset of the function calls. Each theme selects its own terms and images.

Figure 9 illustrates the flow of the GUI processing. The theme is set at applet or application initiation time, based on how Host On-Demand is packaged, and in the applet configuration processing the configuration class (1) instantiates the theme class prior to session initiation (2). The theme class is used when the session class is instantiated to draw the GUI presentations (3). The session class instantiates the GUI classes (menubar, toolbar, keypad, OIA3270, and screen) and obtains the GUI subitems to add (to the menubar, toolbar, and keypad) from the configuration class. The configuration class (1) takes into account the enablement bit settings and skips the items

that do not fit the current environment. When a user clicks on the GUI items (4), they are converted into an internal function code and processed in the host access component (5).

The GUI process also detects environmental differences and dynamically enables and disables GUI elements at session startup time. This is a useful feature to support the "write once, run anywhere" Java capability. As an example, the Host On-Demand help information is HTML-based, which requires a browser. If Host On-Demand is not running in a browser context, such as running under the applet viewer or as an application, help information requests will cause errors. Host On-Demand automatically detects the environment and disables the help-information-related menu items and buttons in that situation. Another application of this concept is the automatic adjustment of the Host On-Demand toolbar icons for help-related information. If the applet is opened in its own window, the help information is accessible from the menu items, and therefore, the related icons are disabled from the toolbar. When the applet is opened in the browser window, no menu items are available, and these toolbar icons are enabled.

It is not unusual for Windows applications to provide some degree of end-user GUI customization. Most leading emulator products allow users to tailor their individual iconic toolbars. Host On-Demand uses a different approach for customization by providing theme-based and "canned" GUI packages that automatically adjust to the run-time environment. The customization is at the OEM level, through a simple Java class definition.

Applet environment

The third main component of Host On-Demand is the applet run-time environment support. Its function is to keep other components from having to directly interface with the surrounding Web environment, keep track of session configurations, maintain hot session links, and preserve the persistent session state.

Figure 10 illustrates the class instantiation flow when the applet is loaded. With the arrow pointers indicating direction, the applet instantiates the session configuration class, then the session class, and so on, to start the entire Host On-Demand processing. The run-time environment classes are listed outside the session rectangles. These classes control an applet-wide configuration, including the active session count, access to system services such as tracing, helps, and National Language Support (NLS), and are responsible for masking complexities associated with the different operational environments. For example, graphical images are retrieved differently when running in Java applet and application contexts. This is because different getImage() methods are provided based on whether it is an applet or application. The Java applet class has a getImage() method for retrieving images, whereas an application must use the getImage() method from the default Java Toolkit class.

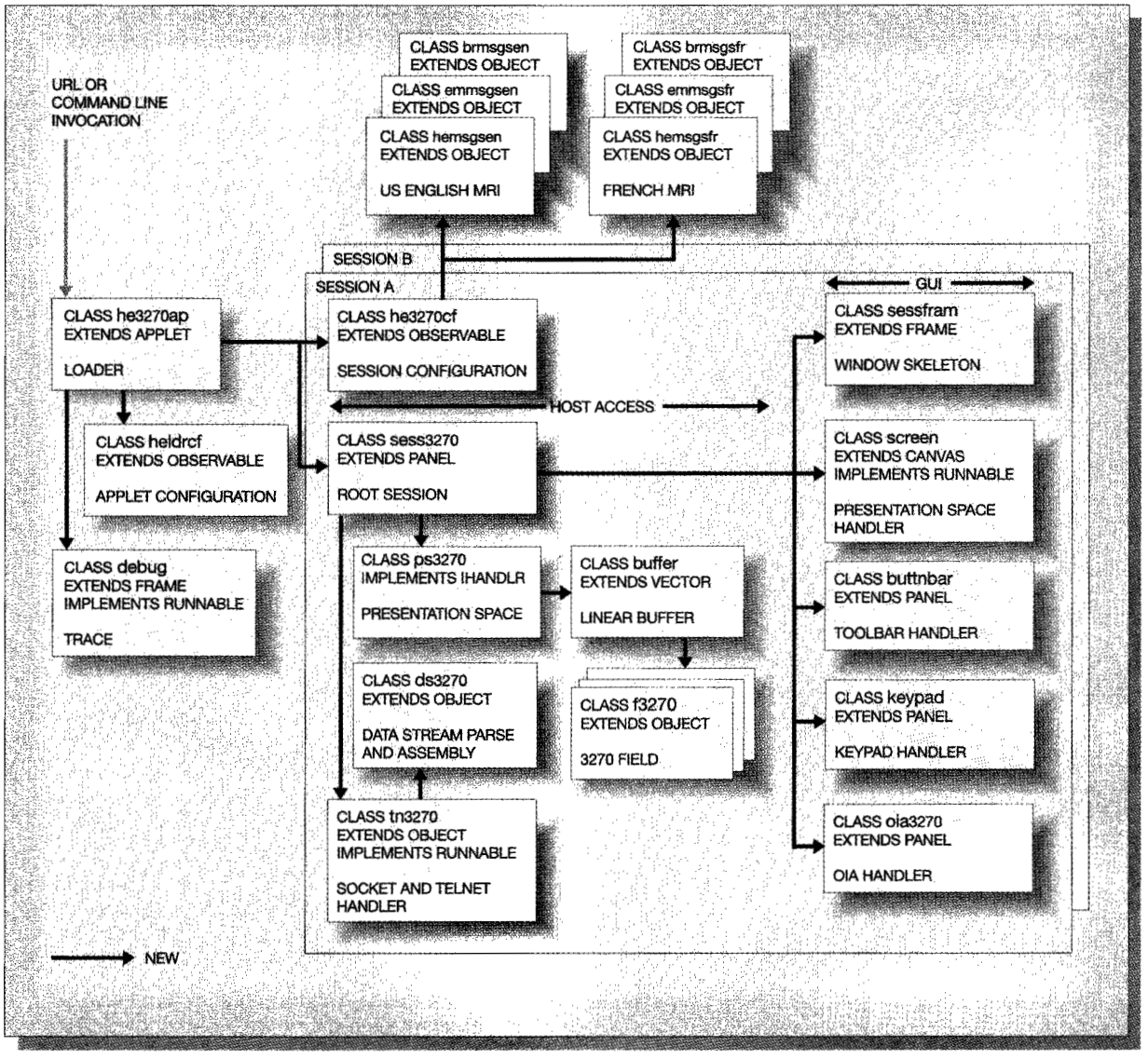
The configuration object class (he3270cf) is used to keep track of user selections made for a session. Each session instance has one configuration object. If the session needs to obtain an external file, such as when a user clicks on a help menu item, the session class will invoke the configuration class methods to obtain the help HTML file. The configuration class also controls the drawing of GUI elements. The code segment in Figure 11 illustrates how GUI processing uses configuration class methods to create the graphical toolbar.

The function of preserving persistent sessions is implemented using the session vector class (heldrcf). Unfortunately, for persistent 3270 sessions, Web browsing is a stateless environment. If a user is browsing the Web while a host session is active, the applet page may be refreshed or reloaded during Web navigation. The state of the active session would be lost. Host On-Demand saves active session objects in a global structure that survives applet instantiations. When an applet is first initiated, the session vector is created and stored in a static (global) object variable. Active sessions are tracked in the session vector. If the applet HTML page is reloaded in the current browser session, the session vector is reassociated with the new instance of the applet. Therefore, active sessions are not disrupted and continue to operate under the new instance of applet.

Host On-Demand dynamically maintains hot links on the applet HTML page to link the page directly to the active session windows. Users can leave their host sessions at any time to navigate the Web. They can return to the sessions with a simple point and click from their applet page.

We have discussed the main components of Host On-Demand. Host On-Demand also provides NLS and

Figure 10 Host On-Demand classes



problem determination capabilities. Both areas posed an interesting implementation challenge in the Java environment. Host On-Demand enables NLS through a message text scheme, and provides a built-in debugging facility that dynamically interacts with the main components. These approaches may be of interest to general Java developers. Let us examine them closely.

Message text

Host On-Demand Release 1 was designed for level 1.02 Java Virtual Machines (JVMs), which does not include the comprehensive NLS functionality associated with the locale class and text package in 1.1-level JVMs. In spite of the JVM 1.02 restrictions, the product was structured to enable NLS and facilitate

Figure 11 Code segment for creation of graphical toolbar

```
.....
String[] toolbar = config.getToolbar( );
                  // Get toolbar object for this locale
  for  (i=0; toolbar [i+1] !=null; ++i) {
                  // for each button in the toolbar
    if (config.isToolbarEnabled(toolbar[i])) {
                  // if button is available in current context
      Image image = config.getToolbarImageID(toolbar[i]);
                  // get image file name
      String helpText =
         config.getHEMsg(config.getToolbarHelpID(toolbar[i]));
                  // get popup help message
      buttonBar.addButton(image, toolbar[i], helpText);
                  // add button to toolbar
    }
    // else pass this button
  }
```

Figure 12 HTML applet tag

```
<applet archive="he3270ap.zip" code="he3270ap.class"
          width=900 height=600 align=center>
  <param  name=NLS_CODE            value=EN>
  <param  name=CABBASE             value=he3270ap.cab>
  <param  name=AUTO_CONNECT        value=NO>
  <param  name=DEBUG               value=YES>
  <param  name=SEPARATE_WINDOW     value=YES>
  <param  name=TN3270E_SERVER_PORT value=23>
  <param  name=GRAPHICS            value=YES>
  <param  name=AUDIO               value=NO>
</applet>
```

translations when the JVM restrictions are removed. The approach is to separate message and GUI text from the mainline processing, allocate an individual national language directory per country, carry country-specific code page tables, and use the configuration class methods to separate others from retrieving message and GUI text strings.

Release 1 implemented locale-like support for its message text and help information. When the Host On-Demand applet class is initialized, it examines the value of an optionally provided *NLS_CODE* parameter to determine its locale. For example, a value of *EN* instructs Host On-Demand to instantiate a U.S. English message table, code page 850 ASCII/EBCDIC translation tables, U.S. English menus, toolbar, and keypads using static data objects contained in discrete Java class files. Further, the NLS_CODE of EN instructs Host On-Demand to use U.S. English HTML-based help information contained in the EN directory. Using this scheme, Host On-Demand can dynamically tailor the national language displayed to the end user by having the user point to the appropriate HTML page. For example, the HTML applet tag in Figure 12 invokes Host On-Demand with U.S. English machine readable instructions (MRIs).

When the he3270ap applet initializes, it instantiates a configuration class object using the passed NLS_CODE. Subsequently, the configuration object instantiates MRI-dependent instance data. The top row of Figure 10 describes this flow. The Java code in Figure 13 illustrates the language-specific class instantiations.

Figure 13 Language-specific class instantiations

```
... if (nls_code.equals("EN")) {
      String[] base_msgs          = new hemsgsen() .msgs;
        // U.S. English messages..
      short[]  ebc2asc            = new hemsgsen() .ebc2asc;
        // ..EBCDIC->ASCII translation table
      short[] asc2ebc             = new hemsgsen() .asc2ebc;
        // ..ASCII->EBCDIC translation table
      String[] menu_bar_context   = new
         emmsgsen().menu_bar_context;      // ..Menus
      String[] tool_bar_context   = new
         emmsgsen().tool_bar_context;      // ..Toolbar
      String[] keypad_context;    = new
         emmsgsen().keypad_context;        // ..Keypad
    }
    else if (nls_code.equals("FR")) {
      String[] base_msgs          = new hemsgsfr().msgs;
        // French messages..
      short[]  ebc2asc            = new hemsgsfr().ebc2asc;
        // ..EBCDIC->ASCII translation table
      short[]  asc2ebc            = new hemsgsfr().asc2ebc;
        // ..ASCII->EBCDIC translation table
      String[] menu_bar_context   = new
         emmsgsfr().menu_bar_context;      // ..Menus
      String[] tool_bar_context   = new
         emmsgsfr().tool_bar_context;      // ..Toolbar
      String[] keypad_context     = new
         emmsgsfr().keypad_context;        // ..Keypad
    }
    else...
```

Methods throughout Host On-Demand do not "hard-code" message strings to be displayed to the end user. Rather, methods are invoked in the configuration object to retrieve the appropriate string. The following Java code creates a label using message #28 which is retrieved from the configuration object using the getMsg method; the message number serves as an index into the currently instantiated message table:

```
Label label3=new Label(config.getMsg(28));

    // Get text for "Separate window" field
```

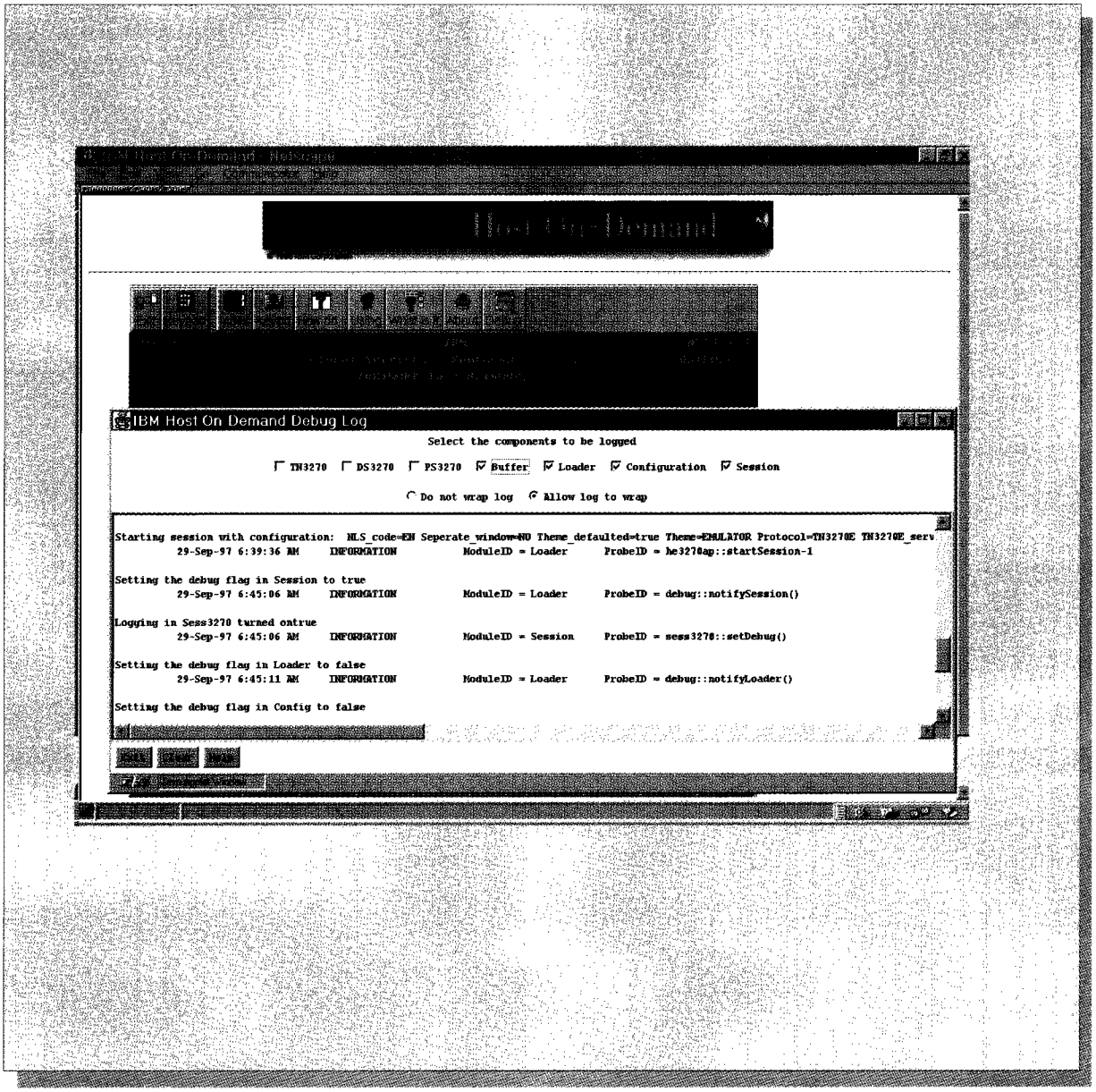
By isolating the MRI from the other Java source code, the discrete Java class files containing the static MRI data objects and the HTML-based help information can be submitted for translation without affecting the product development cycle.

Debugging facility

The debugging facility is intended for internal developers, vendors, and problem determination specialists who are involved in handling error situations. The conventional approach would be to generate the traces, messages, and error logs and provide a separate utility to format the information. That approach does not fit well with the Java applet model because typically an applet does not have access to local files. Having a separate utility also means running as a separate applet that could cause unwanted complexity of inter-applet communications.

Host On-Demand uses a compact debugging solution that dynamically displays formatted problem determination information. The debugging facility is controlled by a Host On-Demand applet parameter. If the applet parameter specifies no debugging, there is no indication of its existence from the toolbar and menubar. Therefore, normal users would not bother with the debugging function. If debug is specified in the applet parameter, a debug button and menu item will appear on the 3270 screen window for turning the function on and off. The debug information is organized by host access classes, which provide a layered level view. Figure 14 shows the debug window. The component selections are implemented as Java Checkboxes and the trace and message display area is a Java TextArea. The window runs in a separate background thread from Host On-Demand processing.

Figure 14 Host On-Demand debug facility



Several display options were considered for the debugging facility before it was decided to have Host On-Demand provide its own debug window. The obvious one was to use the System.out.println standard output stream from the java.lang.System class to print messages to a console. This works fine for internal software development but causes a problem in the user environment because not all browsers provide output consoles. Netscape Navigator** has a Java console, but Microsoft Internet Explorer** does not—it has a log file instead. Finally, Host On-Demand's own debug window is selected to provide a level of consistency across all environments.

In Figure 10, the left-most column indicates the instantiation of the debug class. The debug class runs

Figure 15 Flow of scheme for intercomponent communications

```
/**
 * debug Class
 */
public class debug extends Frame implements Runnable {
  .........
  public debug() {
    .........
    tn=new Checkbox("TN3270",null,false); // create the TN3270 Checkbox
    .........
    msgarea=new TextArea(20, 80);
    .........
  }

  public void println(String textlog) {
    msgarea.appendText(textlog+"\n");
    .........
  }

  public boolean action(Event evt, Object arg) {
    if(evt.target.equals(tn)) {          //the TN3270 Checkbox was hit
    notifyTN3270(tn.getState());
    ..........
  }

  private void notifyTN3270(boolean state) {
     tn3270.setDebug(state);
     ..........
  }
  ....................
} // debug class

/**
 * tn3270 Class
*/
Public class tn3270 extends Object implements Runnable {
  .........
  private static boolean tn_debug=false;
  ..........

    static void setDebug(boolean dbg) {
    tn_debug=dbg;
  ..........
  }

  public void run() {
    ..........
    if(tn_debug) he3270ap.debug.println("----TN3270: connected to host .........");
  ..........
  }
  ....................
} // tn3270 class
```

on its own thread and listens to the Checkboxes options to notify individual components when to start and stop generating traces and messages. Each component has a debug switch that controls the built-in trace and message logging of the component. The debug information is written to the TextArea of the debug window using methods provided by the debug class. The notification mechanism sets and resets the switches. This simple scheme is useful for general intercomponent communications, before JavaBeans**.[12] The code segment in Figure 15 illustrates this flow.

The future

The Java model is a server-centric application model. Java applets are dynamically downloaded "on demand" to requesting clients, enabling users to inter-

Figure 15 Flow of scheme for intercomponent communications (*continued*)

```
/**
 * he3270ap Class
*/
Public class he3270ap extends Applet {
  ..........
  static public debug debug;
  static private Thread debug_thread;
  ..........
  public void init() {
    ..........
    start_debug();
    ..........
  }

  void start_debug() {
    debug=new debug();
    ..........
    debug_thread=new Thread(debug);
    debug_thread.start();
  }
  ..................
} // he3270ap class
```

act with virtually any information source from anywhere. As described, Host On-Demand provides on-demand access to host applications from home, the office, and even on the road. Host On-Demand Version 2 added new capabilities and broadened the application base. In addition to 3270 applications, Host On-Demand can now access 5250, VT (virtual terminal), and CICS* (Customer Information Control System) gateway applications. It also supports Host File Transfer, Telnet redirection, Secure Sockets Layer (SSL) security persistent user configuration, Host Access Class Library, and much more. This new application model is quite powerful from a universal access perspective, but the dynamic nature of network application access presents many new performance and management challenges and a seemingly endless opportunity for new functions.

Java applets are placing an increasing demand on the Internet and intranets. Applets are typically transferred from the source server to the requesting client with each invocation. The applet transfer time varies depending upon the size of the applet and the available network bandwidth and server utilization, creating an opportunity for applet caching technologies to reduce transfer costs and maintain acceptable response times. Preferably, the caches will automatically detect which applets should be cached, probably based on frequency of use, and apply updates when changes to the applets are detected. In addition to applet transfer times, the mobile or remote office user may find the response times of some applications will be affected by the available network bandwidth. For Host On-Demand, we have employed technology from other IBM products, the IBM eNetwork WebExpress[13] and eNetwork Emulator Express,[14] to satisfy our applet caching and response time needs when limited network bandwidth was available—less than 28.8 kilobits per second.

The Java applet model eliminates many user tasks traditionally associated with application installation and software distribution. The next step in reducing administrative costs will be to automatically configure the appropriate network gateways, servers, personal preferences, etc., as required by the specific application or user location. Directory services will play an important role in addressing configuration issues. As Java applets enhance their use of directories, users will no longer be burdened with supplying network configuration information or personal preferences across multiple applets (or even invocations of the same applet). Administrators will be able to specify configuration information for their users and manage user information access capabilities. Another related set of management services will address security, both authorization and encryption, and single sign-on capabilities. The ease with which users may access applications in this new on-demand model will accelerate the value of these new management services to businesses and their user populations.

Java has gained wide industry support and acceptance as the key technology for network computing. Its platform-independent nature allows write-once, run-everywhere programming, and the applet downloading mechanisms simplify software distribution. JavaBeans is the latest addition to Java, providing reusable components that can be manipulated visually in software development tools. With JavaBeans, new Java applications can reuse components to build complex applications. Many of the components from Host On-Demand would make useful "beans," for example, a session screen bean to start a host session and present a display area for host output and input. Other host beans may include file transfer, bookmark, or keypad. An intriguing thought is to deploy some of the services needed for performance and management as beans.

Conclusions

We have shown that Host On-Demand provides a natural and practical solution for running host applications from the Web. A key ingredient that makes this possible is Java. By implementing core emulation functions in Java classes, the functions can be stored on the Web and sent as applets to any computer that has a Java Virtual Machine. Java provides the downloading mechanism through the HTTP protocol, transparent to the applets. The applet logic itself is entirely built on existing host data communication technology, and hundreds of thousands of host applications are readily available to Web users immediately. Applets can also run as local applications. Combined with a Java-based software distribution mechanism, local applications will have the benefit of Java applets but with reduced downloading frequency and added flexibility in desktop integration. Future Host On-Demand technologies will switch the emulation services dynamically between applets and applications and blend host information with the Web and other sources into customized forms to meet individual user's needs. The future is exciting! The rapid evolution and proliferation of network computing will change the way we work and live.

*Trademark or registered trademark of International Business Machines Corporation.

**Trademark or registered trademark of Sun Microsystems, Inc., Microsoft Corporation, Open Systems Ltd., Apple Computer Corporation, Novell Inc., or Netscape Communications Corporation.

Cited references

1. T. Berners-Lee et al., *The World-Wide Web* **37**, No. 8, 76–82 (August 1994).
2. T. Berners-Lee and D. Connolly, *Hypertext Markup Language Specification—2.0*, Internet Draft, Internet Engineering Task Force (IETF), HTML Working Group (June 1995); available at http://www.ics.uci.edu/pub/ietf/html/html2spec.ps.gz.
3. T. Berners-Lee, R. Fielding, and H. Frystyk, *Hypertext Transfer Protocol—HTTP/1.0 Specification*, Internet Engineering Task Force (IETF), Internet Draft (August 13, 1995); available at http://www.ics.uci.edu/pub/ietf/http/draft-fielding-http-spec-01.ps.Z.
4. B. C. Housel and D. B. Lindquist, *WebExpress: A System for Optimizing Web Browsing in a Wireless Environment*, MOBICOM '96, Rye, NY (November 10–12, 1996).
5. B. Kelly, *TN3270 Enhancements*, RFC 1647, Auburn University, Decatur, AL (July 1994).
6. "Web Publishing of Host-based Information," *Microsoft SNA Server Market Bulletin*, Microsoft Corporation, Redmond, WA (October 1996).
7. G. Cornell and C. S. Horstmann, *Core Java*, SunSoft Press, Mountain View, CA, ISBN0-13-565755-5 (1996).
8. *IBM Host On-Demand*, IBM White Paper, IBM Corporation, Armonk, NY (November 1996); see http://www.networking.ibm.com/hex/white_paper_en.html.
9. *3270 Information Display System Data Stream Programmer's Reference*, GA23-0059-07, IBM Corporation; available through IBM branch offices.
10. IBM Host On-Demand Version 2.0, product announcement, IBM Corporation, Armonk, NY (September 9, 1997); see http://www.networking.ibm.com/ene/news9hex.html.
11. Netscape Communicator Professional Edition, Netscape Communications Corporation (June 1997); see http://home.netscape.com/comprod/products/communicator/index.html.
12. *Component-Based Software with JavaBeans and ActiveX*, White Paper, Sun Microsystems, Inc.; see http://www.sun.com/javastation/whitepapers/javabeans/javabean_ch1.html.
13. IBM eNetwork Web Express, IBM Corporation, Armonk, NY; see http://www.networking.ibm.com/art/artexp.htm.
14. IBM eNetwork Emulator Express, IBM Corporation, Armonk, NY; see http://www.networking.ibm.com/art/artemul.htm.

Accepted for publication August 29, 1997.

Yih-Shin Tan *IBM Software Solutions Division, 3039 Cornwallis Road, P.O. Box 12195, Research Triangle Park, North Carolina 27709 (electronic mail: ystan@us.ibm.com).* Mr. Tan joined IBM in 1978, specializing in networking software development, and later in MVS architecture, design, and development. In 1989 he took an assignment with the Enterprise Systems Group to review S/390 software-related issues and later joined the Networking Systems Division in 1994 as a member of the technology staff, where he focused on exploring new workstation and Web-based communication software that leverages legacy systems. His work led to the concept of Host On-Demand with the result that he was assigned the leadership role for the architecture, design, and implementation of Host On-Demand. Mr. Tan is currently a senior software engineer in the Networking Software Laboratory at the IBM Research Triangle Park facility. He received a B.S. degree in mathematics from Fu-Jen Catholic University, and dual M.S. degrees in mathematics and computer science from The Ohio State University in 1977.

David B. Lindquist *IBM Software Solutions Division, 3039 Cornwallis Road, P.O. Box 12195, Research Triangle Park, North Carolina 27709 (electronic mail: lindqui@us.ibm.com).* Mr. Lindquist joined the IBM Data Systems Division in 1982, specializing in large-system performance, and later in large-system architecture and design. In 1990 he joined the Networking Systems Division as a member of the technology staff where he focused on distributed multimedia and mobile products. His research has led to numerous patents in the area of distributed processing, and recognition as an IBM Master Inventor. Mr. Lindquist is currently a Senior Technical Staff Member in the Software Solutions Division. He received a B.S. in computer engineering from Boston University in 1982.

Thomas O. Rowe *IBM Internet Division, P.O. Box 12195, Research Triangle Park, North Carolina 27709 (electronic mail: trowe@us.ibm.com).* Mr. Rowe received a B.S. in computer science from Lehigh University and joined IBM in 1984. He has worked in the field of real-time microprocessor-based programming his entire career. He has designed and implemented software for statistical multiplexers, signal processors, SNA communications subsystems, and most recently, the Java-based Host On-Demand product. He is currently working in the Network Computing Framework (NCF) architecture department with a focus on server-side Java and Internet push technology.

John R. Hind *IBM Software Solutions Division, 3039 Cornwallis Road, P.O. Box 12195, Research Triangle Park, North Carolina 27709 (electronic mail: hind@vnet.ibm.com).* Mr. Hind received a B.S. in electrical engineering with an equivalent major in computer science and an M.S. in computer science and applications, both from Virginia Polytechnic Institute and State University in 1971 and 1973, respectively. Subsequently, he worked as a research instructor at Virginia Polytechnic Institute and State University Department of Electrical Engineering; as a systems analyst, Division of Consolidated Laboratories, Commonwealth of Virginia; and as associate, Planning Research Corporation, Information Sciences Company, concentrating on various aspects of distributed processing. In 1977 he joined IBM in Kingston, New York, to work on the DPPX communications subsystem development team. Subsequently he has worked on various aspects of networking, including document distribution, e-mail, library, directory, security, message queuing, transaction routing, multiprotocol transport, and most recently, Web integration and optimization.

Reprint Order No. G321-5668.