CONSTELLATION

A WEB-BASED DESIGN FRAMEWORK FOR DEVELOPING NETWORK APPLICATIONS

Nino Vidovic, Dalibor F. Vrsalovic

Abstract

Constellation is a Web-based design framework for developing distributed applications which allows a single user or group of users to concurrently access and manipulate different aspects of a distributed application from a simple Mosaic-like front-end tool. Users can edit and build programs and documents, manage source code, debug and instrument running distributed programs, read manual pages and other documents, browse through source code and much, much more from a simple yet powerful front-end tool. Constellation is designed to work in a heterogeneous networked environment. It works with different host types and different OS environments, and supports different communication protocols. It is designed to work with hybrid client/server applications that consist of new as well as legacy code. Constellation is easily extensible to support new protocols, hosts, servers, services and back-end tools. ***Keywords:*** *Distributed applications, client/server applications, design framework, distributed debugging, group debugging, development environment*

Introduction

In response to societal demand, computer networks have been proliferating rapidly in recent years. Such networks include local area networks (LANS) and wide area networks (WANS) comprising a number of computers that may communicate with one another. Apart from sending messages, this communication between networked computers allows programs to be run on more than one computer in a network. For example, an airline reservation service may present a user interface on a client computer while data input to the interface is transmitted to a server computer where a reservation database is accessed. This type of program execution, known as distributed programming, may be much more complicated than the above example but is nonetheless extremely common and efficient.

Distributed programs, however, are often written, developed and tested on different computers with different tools which are tightly bound to the particular machine upon which they operate and are integrated around language, computing platform, or type of application.

This problem is illustrated in Figure 1.

In the above example, the graphic user-interface portion may be developed on the client computer with a set of program tools for the Visual C++ programming language for PC Windows and a different set of tools for a graphic user-interface (GUI) builder. Similarly, the reservation database may be developed on the server with a set of program tools from the C++ Workshop programming language running under Unix. In Figure 1, those development environments are denoted as DE1, DE2 and DE3. Furthermore, a program initially developed with a tool set can frequently be redeveloped under a later version of the same tool set, and possibly from a different manufacturer.

Thus, distributed programs present substantial difficulties to programmers since they must learn to operate the tool set that was used to develop each distributed program segment. These tool sets are usually quite detailed and require days, weeks, and even months to master. Furthermore,

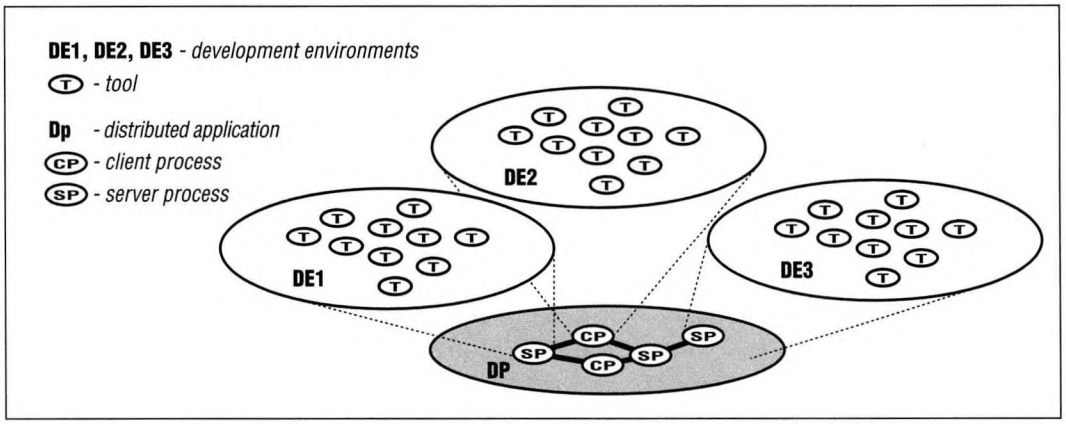

Figure 1: Designing network applications using traditional programming environments

to edit, view, or debug the distributed program currently requires performing these functions under one tool set for a particular program segment, exiting the tool set, locating the tool set for a different program segment, which may be on a different computer, and entering that tool set. This type of operation is extremely time consuming, especially where a program has a large number of segments, each developed with a different tool set.

By employing a particular protocol, the World Wide Web has met the challenge of allowing users, through a single front-end tool, to browse documents that reside on a large number of different platforms. The World Wide Web, however, which has been in existence for a number of years, does not provide for any other types of functions apart from browsing and the previously described problems presented by distributed programs remain.

So, there is a need for a system that provides the capability to develop distributed programs that operate on different computers, operating systems, and communication protocols, while requiring only one set of tools. Furthermore, there is a need for a system that allows such integration for programs that have already been partially developed or completely developed under a variety of tool environments and that require modification. More broadly, there is a need for a system that allows a single front-end tool to perform operations on a plurality of files that reside on different platforms without requiring a user to separately access each separate platform-specific piece on that platform.

The Constellation framework answers these and many other needs.

Constellation

The Constellation framework provides methods and mechanisms for a front-end navigating tool (the "Navigator") that may access and manipulate files distributed across different platforms. Figure 2 depicts Constellation framework as a cone with the Navigator at the top and a distributed program in its base. The front-end navigating tool communicates with a multitude of server processes, resident on networked servers, to perform all types of file manipulations such as debugging and editing. The server processes communicate with gateway processes (resident on the same machine as the calling server process) that perform the desired function on any of a multitude of program segments that may be distributed across a network of computers.

Debugging is one example of a function that may involve distributed files. To service a debugging

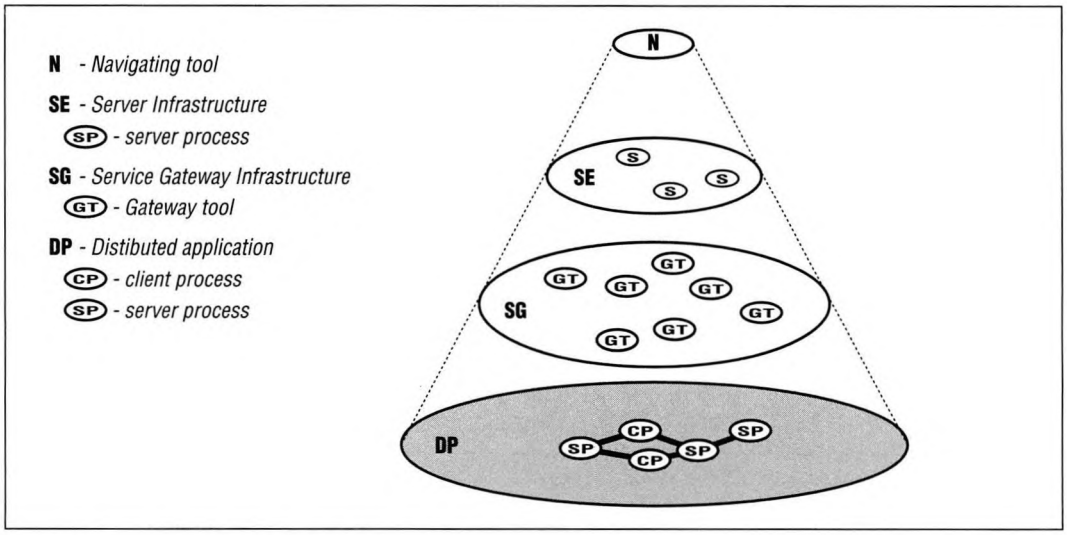

Figure 2: Cone model of a distributed programming environment

request, the navigator issues a request according to the URL protocol. Thus, a request is of the form: <server_type://<machine/<request, where server type is a protocol such as, for example, HTTP or process, machine is the actual server address, and request is the program or file that is requested. The appropriate machine and server process is then contacted by the navigator and provided with the name of the file, *file A*. The server process in turn selects the appropriate gateway process to perform the desired function. The gateway process attaches to the desired program, which is subsequently debugged. If the program calls a program on a different machine, the Constellation framework provides a variety of mechanisms, transparent to the user, for allowing debugging to continue on the called program. In a preferred embodiment, the gateway process provides the server process with the address of the called program, file B. The server process notifies the navigator which then automatically assembles a request to the server process on the different host. The request is sent and the server process selects the appropriate gateway process, which then attaches to the target program, *file B*. If *file B* returns to *file A*, then control is again passed to the navigator which calls *file A* as before except that the gateway process has maintained the appropriate address of the instruction after the call instruction to *file B*. Debugging then continues on *file A*.

Browsing, editing, and any other function requests are similarly serviced through the front-end navigator. A URL link contacts an appropriate server process resident on a target machine and the server process in turn selects the proper gateway process, which performs the desired function on the target file. The target file may include hypertext links to other files, and functions can easily be performed on these files by clicking on the file names and then indicating a desired function. In this manner, users can efficiently access and manipulate distributed files through a single front-end tool.

System Architecture

Figure 3 is an overview of the architecture of the Constellation framework. The architecture comprises a front-end navigating tool (Navigator) that communicates with a multitude of server processes (PROCD, HTTPD and PROGD) to perform different functions such as debugging, source-

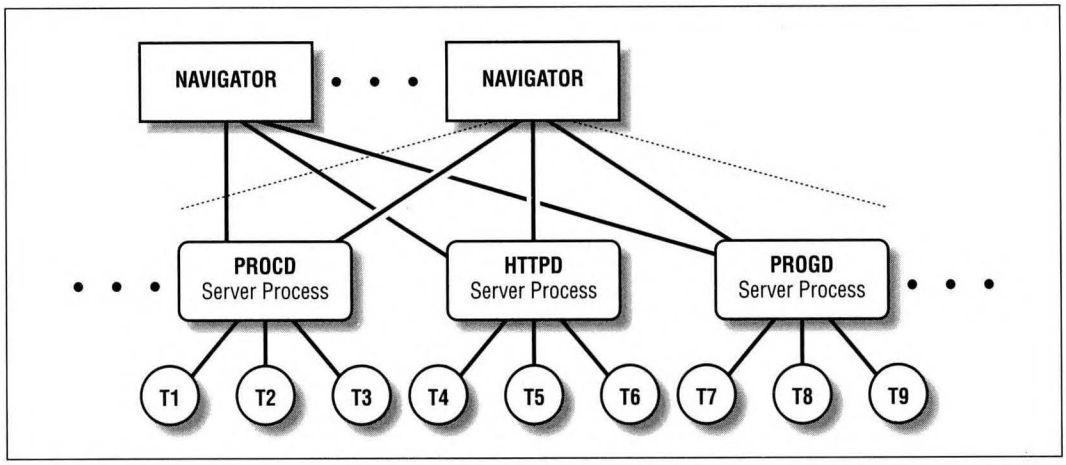

Figure 3: Architecture of the Constellation design framework

code management, and source-code browsing and editing. The server processes PROCD, HTTPD, and PROGD communicate with gateway processes (tools) T1, T2, T3, T4, T5, T6, T7, T8, and T9 that perform a desired function on any of a multitude of program segments that may be distributed across a network of computers. As will be described more fully below, the navigating tool comprises a browser, editor, and interactive shell and allows users to perform a variety of functions on distributed program segments. For example, the server process HTTPD may comprise a document server that communicates with file, *man2html* and mail gateway processes T4, T5 and T6 to perform browsing functions. Similarly, the server process PROCD may comprise a process server that communicates with dbx, PC-debug, and gdb gateway processes T1, T2 and T3 to perform debugging functions. The architecture illustrated in Figure 3 may be easily extended to include many other server processes and gateway processes as indicated by the dashed lines. Furthermore, Constellation framework allows two navigators to simultaneously access the same server process, gateway tool, and file, Also, two Navigators can communicate with each other in a networked environment.

As will be described more fully below, each of the server processes PROCD, HTTPD and PROGD illustrated in Figure 3 may reside on a plurality of physical machines. The architecture of the Constellation framework provides for the integration of a variety of tools, including debugging, document-and source-code browsing and editing, source-code management, and program development and building. The implementation of these tools according to the architecture of the Constellation framework will be described in the following sections of this paper. The architecture of the Constellation framework as illustrated in Figure 3 may be applied to many other types of tools, including user-defined tools.

Navigating front-end tool

The Navigator is a browser and at the same time an editor and an interactive shell. Browser capabilities are similar to those of Web browsers such as Mosaic or HotJava. As an editor, the Navigator has basic text-editing functionality of point and click editors, such as Textedit with command bindings for Emacs and Vi. Editor's capabilities are augmented to support hyper-text (i.e., HTML) program annotations. Interactive shell capability allows users direct interactions with browser's interpreter engines as well as with interactive

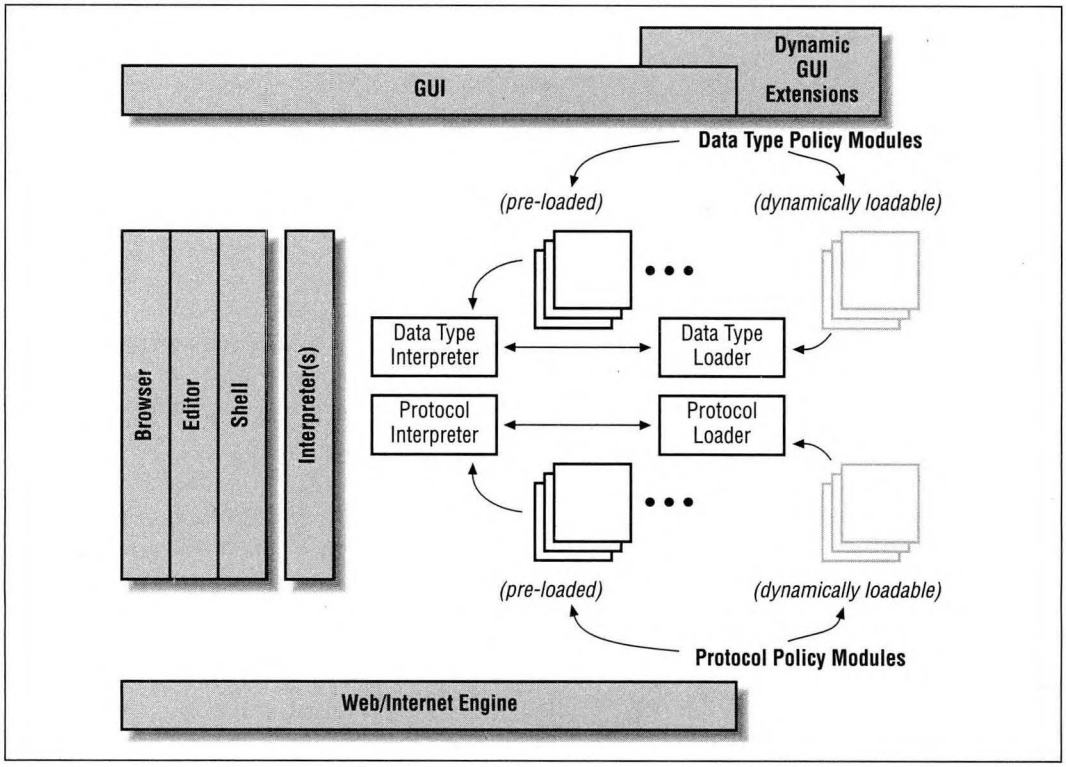

Figure 4: An architecture of the Navigator front-end tool

programs with which the Navigator has established links (e.g., gateway debug engines such as dbx).

The architecture of the Navigator is shown in Figure 4.

The Navigator consists of the following basic components: Web/Internet engine, Browser, Editor, Interactive shell, Interpreters (e.g., Tcl or Java), Data-Types Processing Engine, Protocol-Processing Engine and GUI. The Web/Internet engine provides access to the Internet and Web. Browser provides basic Web client functionality. Editor allows pages in the Navigator to be edited. Interactive shell capability allows Navigator to hold interactive sessions. The Interpreters are mechanisms which allow dynamic extension of Web protocols and data types, as well as Navigators GUI. Data-Type Processing Engine allows dynamic retrieval of policy modules (e.g., Tcl or Java programs) to process unknown data types. Protocol-Processing Engine allows the Navigator to process unknown protocol requests by dynamically loading protocol policy modules (e.g., Tcl or Java protocol driver programs).

The GUI module creates a control panel for the Navigator and maps the contents of users' searches into a graphic domain. The novel feature of the Navigator is its capability to dynamically reconfigure its control panel based on the type of the request and the domain in which browsing is performed. For example, when the user makes a request to debug a program, a debug server that services browsing requests in process domain will be contacted, and the appropriate protocol policy module in Navigator will be downloaded (if not already present) and activated. In a debugging example, this processing

would result in a debugging menu being created and then attached to the Navigator's command panel. The debugging menu would stay attached for the duration of the debugging session. Upon completion of the debugging session, the debugging menu would be automatically removed from the Navigator's control panel.

The Navigator can operate in a stateless or statefull mode. Stateless mode is used in events such as directory browsing or information retrieval. Statefull mode is used during distributed debugging sessions. The statefull sessions are characterized by long-live connections into a browsing domain. The Navigator keeps track of all long-live connections for all active statefull sessions. The long-live connection represents a bi-directional communication channel into a browsing domain. The user can post a request to the server which is servicing given domain. Also, the server can generate events and send them back to the Navigator.

No limits exist on a number of the concurrent long-live connections. The Navigator allows all long-live connections to be active at the same time. Policy modules are responsible for coordinating events received from domain servers.

Constellation's service layer

The Constellation service layer consists of a collection of domain servers and gateway tools. Gateway tools are either interface, to service providers (e.g., database access) or service providers. The following list is a sample of domain servers and related gateway tools:

- Document Servers and Service Gateways
- http, file, ftp...
- *man2html, sccs2html, names2html*
- Process Servers and Service Gateways
- Debugging (Dynamic Services) server
- dbx, PC-debug, gdb, RTP, etc.
- Program Servers and Service Gateways
- Program (Static services) server
- cscope, c++class
- Teleconferencing Server and Service Gateways
- showme conference manager
- Audio, video, white board

Debugging

Figure 5 is a block diagram showing how the architecture of the Constellation framework communicates to service a debugging request.

First, the Navigator issues a request according to the Universal Resource Locator (URL) protocol. Thus, a request is of the form: <server_type:// <machine/<request, where server type is a protocol such as, for example, HTTP or process, machine is the actual server address, and request is the program or file that is requested. The appropriate server process PROCD resident on a host 1 is then contacted by the Navigator and provided with the name of the file of a running client process, file A. The server process PROCD in turn selects the appropriate gateway tool PC-debug to perform the desired function.

The gateway tool PC-debug attaches to the target program (application's client process), which is subsequently debugged. If the target program calls a program (application's server process) on a different machine, the Constellation framework provides a variety of mechanisms, transparent to the user, for allowing debugging to continue on the called program. In Figure 5, the gateway tool PC-debug provides the server process PROCD with the address of the called program, *file B*. The server process PROCD notifies the Navigator, which then automatically assembles a URL request to a server process PROCD on the different host. The request is sent and the server process PROCD selects the appropriate gateway process dbx which then attaches to the target program (application's server process), *file B*. If *file B* returns to *file A*, then control is again

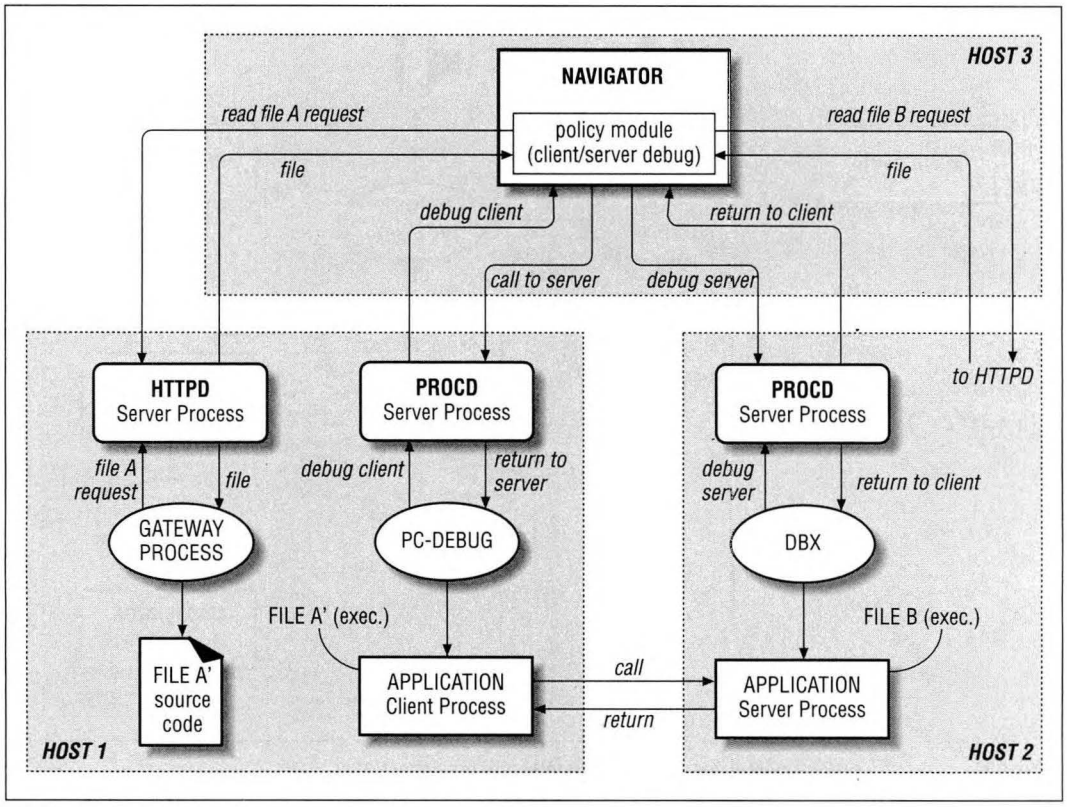

Figure 5: Communication sequence when servicing a debugging request

passed to the navigator which calls *file A* as before except that the gateway process PC-debug has maintained the appropriate address of the instruction after the call instruction to *file B*. Debugging then continues on *file A*. In this manner, programs that are debugged by two different tools, for example dbx and PC-debug, may be debugged through a single front-end Navigator.

Constellation Framework supports groupware functions such as group debugging sessions and teleconferencing. Figure 6 is a block diagram showing how the architecture of the Constellation framework communicates to service a group debugging request.

First, the Navigator on host 1 issues a request according to the Universal Resource Locator (URL) protocol. Thus, a request is of the form: process://<machine/pid=7lt;pid_number, where machine is the actual server address and pid number is the process-identification number of a running program. The appropriate server process PROCD resident on a host1 is then contacted by the Navigator and provided with the pid of a running server process. The server process PROCD in turn selects the appropriate gateway tool dbx to perform the desired function. The gateway tool dbx attaches to the target program which is subsequently debugged.

Request to clone The Navigator is then sent to host 1. After cloning navigator on host 2 is connected to the server process PROCD on host 1, a joint debugging session of the application server program is established. Furthermore, during the cloning process, navigator on host 1 issued a talk

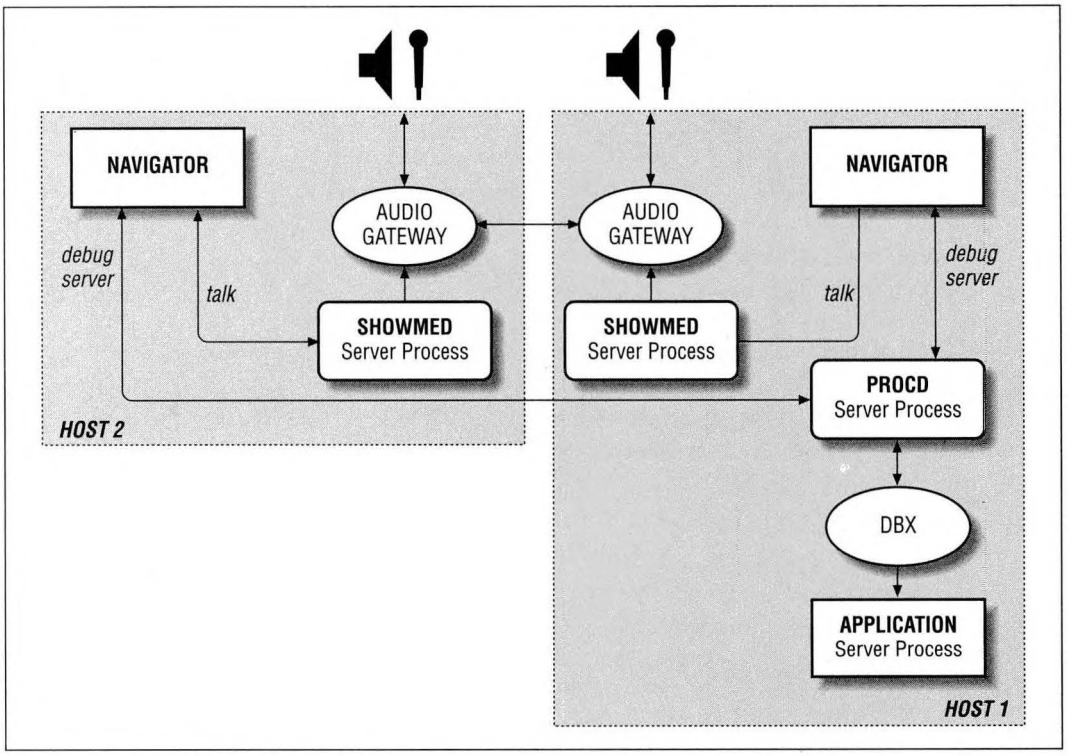

Figure 6: Concurrent debugging session with teleconferencing

request to showmed conference manager. After the manager accepts talk request, audio gateway tools are launched on both hosts enabling navigators to concurrently have voice communications and group debugging session.

File browsing and editing

Figure 7 illustrates one possible block diagram for browsing and editing files, which may comprise source code or any other type of document. A user can browse and edit documents in an infinite number of ways and Figure 7 illustrates one possible browsing and editing session to illustrate the operation of a preferred embodiment of the Constellation framework. To read a file, *file A*, the Navigator issues a request according to the URL protocol, as previously described. The HTTPD server process on the appropriate machine is provided with the request, and the appropriate gateway tool is contacted to retrieve the file. The file, *file A*, is then provided to the navigator for viewing and editing. Unlike World Wide Web browsing tools, the Constellation framework provides the ability to edit files and replace an old file with an edited file. Thus, the navigator allows the user to edit the file and the navigator then formulates the appropriate URL, indicating the file, *file B*, when the user desires to save the edited file. The appropriate server process and gateway process are contacted and the edited file is stored.

In addition to browsing, editing, and saving files, the Constellation framework dynamically generates information concerning the content of retrieved files. For example, a user may desire to analyze a program according to its data structures, function calls and other characteristics. To perform such an operation on a file, the naviga-

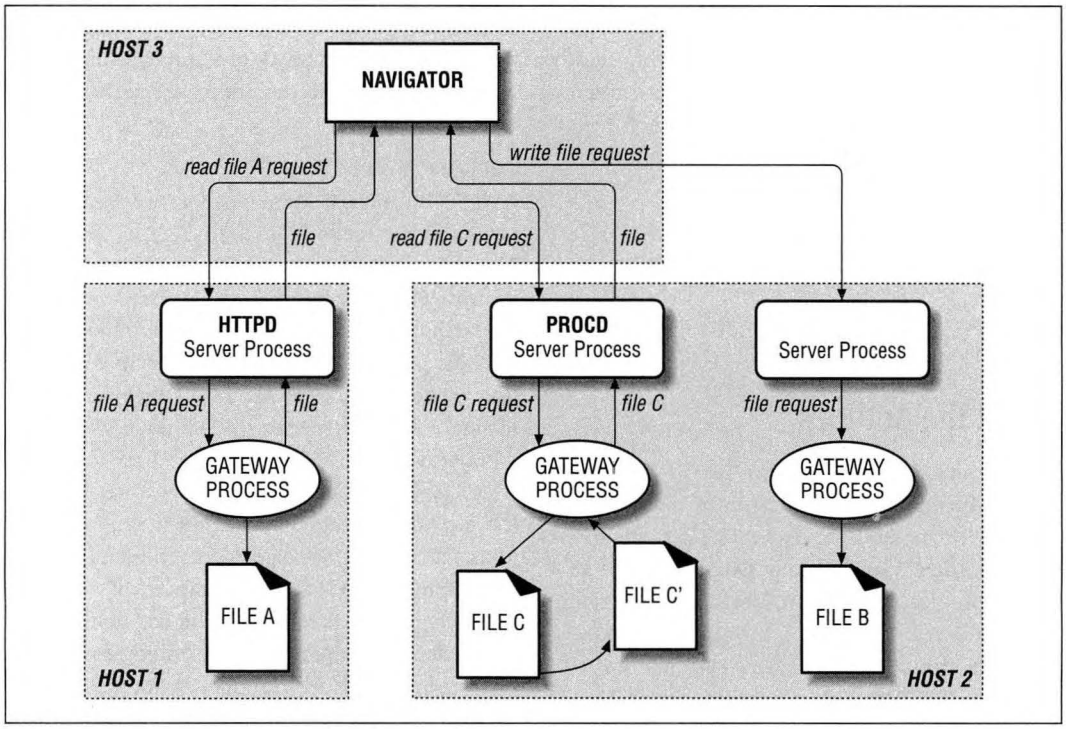

Figure 7: Editing and source-code browsing functions of the Constellation framework

tor issues a request according to the URL protocol, as previously described. An appropriate server process PROGD on the target machine is provided with the request and an appropriate gateway tool (e.g., cscope) analyzes the file and provides the results of the analysis to the navigator.

Summary

The Constellation framework provides methods and mechanisms for a front-end navigating tool that may access and manipulate files distributed across different physical machines and platforms. The front-end navigating tool communicates with a multitude of server processes, resident on networked servers, to perform all types of file manipulations such as debugging, source-code management, and editing. The server processes communicate with gateway processes (resident on the same machine as the calling-server process) that perform the desired function on any of a multitude of program segments that may distributed across a network of computers.

The Constellation framework supports many functions, such as debugging, source-code management, source-code browsing, multi-platform builds, editing, and document browsing. The Constellation framework prototype has been developed mostly in Tcl7.4/Tk4.0 with approximately one full-time and one part-time engineer in a 6-month period. It works on Solaris and Windows NT platforms. ■

Acknowledgments

Thanks to John Ousterhout for early release of Tcl7.4/Tk4.0. Thanks to the DOE debugger team, Jon Masamitsu and Andrew Davidson, for bearing with me while installing DOE and integrating

DOE debugging support. Thanks to DevPro's Ivan Soleimanipour for volunteering to extend dbx's API with yet another call and Achut Reddy for sharing his experience and code for program browsing. Special thanks go to Steven Li for porting Constellation to the Windows environment.

References

1. "The World Wide Web, a global information initiative," *http://www.w3.org*
2. Ousterhout, J.K., *Tcl and the Tk Toolkit*, Addison-Wesley, 1994.

About the Authors*

Nino Vidovic
AT&T—Business Communication Services
Dr. Vidovic recently joined AT&T as Advanced Network Clients Technology Director in the BCS Operations group. His responsibilities include architecture development and implementation of next-generation network client systems. Prior to joining AT&T, Dr. Vidovic served as the Senior Staff Engineer in Advance Techology Group, Sun Microsystems, Inc. While at Sun Microsystems, he was responsible for identification of new technologies for development of network-enabled applications. He holds a doctorate in computer science from the University of Zagreb, Croatia, and an M.S. in computer engineering from Carnegie Mellon.

Dalibor F. Vrsalovic
AT&T—Business Communication Services
Dr. Vrsalovic recently joined AT&T as Advanced Technology Vice President in the BCS Operations group. His responsibilities include providing leadership to the architectural development effort for host platforms to be used in the next generation of network-based services as well as establishing AT&T's long-term leadership in global services and distributed computing. Prior to joining AT&T, Dr. Vrsalovic served as the Chief Scientist for Sun Microsystems, Inc. While at Sun Microsystems, he was responsible for the architecture development and implementation of software products. In addition to R&D, he has held leadership roles in bringing new technologies to general business use and product quality assurance. He holds a doctorate in computer science and an M.S. in computer engineering from the University of Zagreb, Croatia.

* This work was done while authors were working at SunSoft Inc.