43

INTERACTIVE NET-SERVICES ON THE WWW

Kenneth J. Rodham and Dan R. Olsen, Jr.
Computer Science Department, Brigham Young University, Provo, UT 84602 USA
E-mail: krodh@tim.cs.byu.edu, olsen@cs.byu.edu

KEY WORDS: Interactive Net-Services, World-Wide-Web, Transportable Interactive Clients, Network-Based Interactive Systems, User Interface Toolkits

ABSTRACT: This paper explores the question of how the World-Wide-Web's ability to support interactive net-services can be enhanced to support a wider range of applications. We give examples of interactive net-services, describe the abstract architectural components needed to support them, and describe our implementation of these components in the NIC (Nucleus for Interactive Computing) application framework.

1. INTRODUCTION

The World-Wide-Web (Berners-Lee, 1994a) and WWW browsers like Mosaic provide a powerful interactive environment for accessing Internet resources. Users can easily access many kinds of documents including PostScript files, images, and videos. User interactions on the WWW are limited primarily to downloading files and presenting them with appropriate viewers. Although highly useful, this style of interaction is also limited because it does not allow users to engage in two-way dialogues with Internet services. Some help is provided by the ability to access telnet-based services through WWW browsers, but telnet only supports character-based interfaces. A substantial improvement is provided by HTML forms (Berners-Lee, 1994b) which support graphical forms-based interfaces. The forms facility is also limited in that some Internet services require more dynamic UIs than forms can provide. Forms also do not allow the UI to perform application-specific computations on the user's local workstation, although many useful UI functions could be performed locally without accessing the service. Finally, the model used for communication between a forms-based UI and a service is restrictive. A query sent to the service takes the form of a string packed with the contents of the form's fields, and the service's reply takes the form of another HTML document to be presented by the WWW browser.

This paper explores the question of how the WWW's ability to support interactive net-services can be enhanced to support a wider range of applications. We begin by giving some specific examples of interactive net-services that require more than a forms-based UI. We then describe the abstract architectural components that must be provided to achieve greater interactivity on the WWW. Finally, we describe our implementation of these components in the NIC (Nucleus for Interactive Computing) application framework.

2. EXAMPLE INTERACTIVE NET-SERVICES

There are many possible examples of interactive net-services. For example, a grocery store might want to provide an on-line service that allows customers to place orders in advance so that they can drive through and pick up their groceries. When a customer accesses this grocery service through Mosaic, an interactive client that provides a UI to the service is downloaded and executed on the customer's computer. The client UI allows the customer to browse through the choices in various departments, view pictures of items, search for items, inspect the nutritional content of items, order items, tally the bill, etc. Once a complete order has been assembled, the order is submitted to the service at the grocery store. By the time the customer arrives at the store, their order is ready to be picked up.

Another example is a grade submission service provided on the WWW by a university for its faculty members. The client UI for accessing this service allows an instructor to download class rolls, enter student scores on assignments, compute various statistics like averages and standard deviations, generate graphs of student scores, compute final grades, and submit final grades to the university.

A third example, which we have implemented using NIC, is an interactive net-service that lets users search NIC's on-line documentation for patterns. This service is called Grep Service because the

K. Nordby et al. (eds.), *Human-Computer Interaction*

server's functions are implemented using the Unix grep command. When Grep Service is accessed through Mosaic, the client UI shown in Figure 1 is downloaded and executed on the user's computer.

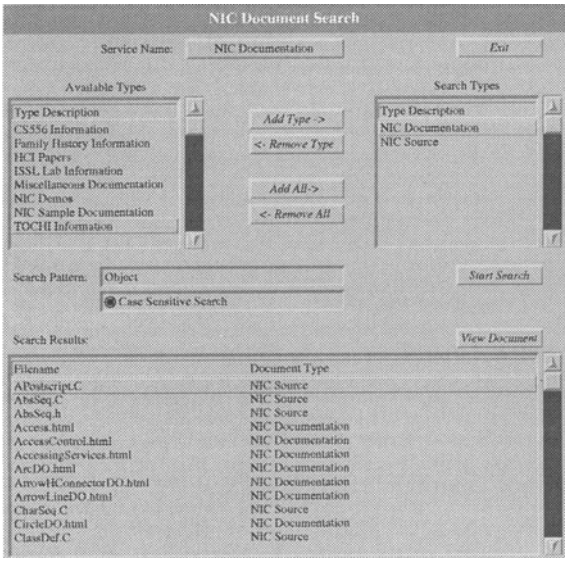

Figure 1

The scrolling list in the upper-left enumerates the various sub-sections of NIC's documentation that can be searched. The scrolling list in the upper-right enumerates the sub-sections that the user has actually selected to be searched. The buttons between the lists are used for transferring sub-sections between the lists. A text field is provided for typing in a search expression, and buttons are provided for starting a search and controlling the case sensitivity of the search. When the user presses the "Start Search" button, the client connects to the Grep Service and sends a request with the search parameters entered by the user. The scrolling list at the bottom of the client UI is a list of files returned by Grep Service that contain the search expression. The user can select files from this list to be viewed using Mosaic. When the user presses the "View Document" button, the client constructs a URL for the selected file and loads the contents of the URL into Mosaic for viewing. The same Mosaic process is used to view each file rather than starting a new instance of Mosaic for each one.

3. ABSTRACT ARCHITECTURE

This section examines the architectural components required to support interactive net-services like those described in the previous section. The sequence of events shown in Figure 2 illustrates these components.

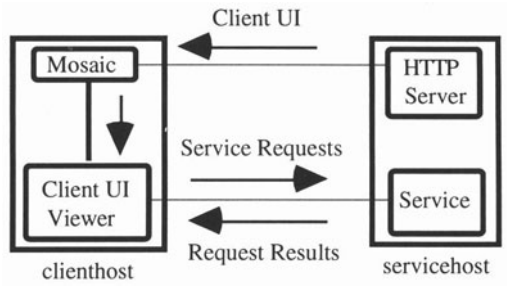

Figure 2

- The user follows a Mosaic hyper-link that points to the client UI.
- Mosaic contacts the HTTP server on servicehost to retrieve the file containing the client UI. This file contains a description of the client's UI as well as its application-specific code and data structures. In order to publish such a client UI on the WWW, we need to be able to represent UIs, application code, and application data in a platform-independent, transportable, storable manner. We refer to client programs that can be downloaded and executed in this manner as *transportable client UIs* to reflect the fact that they must be both transferable over a network and portable across different architectures.
- Having retrieved the client UI over the network, Mosaic spawns a viewer that can instantiate and execute the client. The *client UI viewer* must be able to instantiate the client's UI, manage the interactive dialogue with the user, and execute application code contained in the client. This viewer must be general in the sense that it can support a wide variety of user interfaces just as Mosaic supports a wide variety of HTML documents. The client is much more than a dumb terminal to a remote service; the client can perform substantial application processing on the user's local machine. Interactive clients published on the network will frequently need to include application-specific user interface components. It must be possible to distribute such application-specific interface components along with a client UI.
- The client UI needs to be able to establish connections with the remote net-service, send requests to the service, and receive replies from the service in a simple and efficient manner.
- Building interactive net-services is a difficult and time-consuming programming task. Ideally, UI toolkits will provide tools to simplify their construction.

• Interactive net-services present significant security issues. Client UIs are downloaded from the network and executed locally, and services receive and execute requests from remote clients. In both cases, programs are received from remote sources and executed locally. The activities of programs received from untrusted sources must be restricted.

We have developed a programming framework called NIC (Nucleus for Interactive Computing) that supports interactive net-services by providing the aforementioned architectural components. The remainder of the paper describes NIC's architecture and how it supports interactive net-services.

4. TRANSPORTABLE CLIENT UI'S

A client UI consists of three main components: the user interface, the code that implements application-specific functions, and structures that store the client's data. A client UI must represent each of these components in order to contain sufficient information to execute the UI on the user's workstation. NIC's representation of client UIs is based on a uniform object model that is used to represent all three components of an interactive program. All objects in NIC are subclasses of the abstract class Object. An Object consists of an array part and a list of (Attribute, Value) pairs. Figure 3 shows the textual representation of a NIC object of class Student. The attributes of a Student object contain the student's name, age, and address. The array of a Student object holds a list of the courses in which the student is currently enrolled. Each course is represented by a sub-object whose array elements contain the department and class number.

```
(:Student
   [Name        "Bill Smith"]
   [Age         17]
   [Address     "P.O. Box 1234"]

   (Physics 101) (English 301) (Math 344)
)
```

Figure 3

NIC provides several subclasses of Object that can be used by programs to store their data. For example, the class DataObject implements a generic object whose array elements and attributes can hold any value, including other objects. When no class is specified for an object, it is understood to be of class DataObject. For instance, the sub-objects that represent courses in Figure 3 are instances of DataObject. The Grep Service client UI uses instances of DataObject to store information about what documentation sub-sections are available, which sub-sections the user has selected for searching, and the list returned by Grep Service of documents containing the current search expression.

NIC also provides subclasses of Object that implement a complete set of user interface components. For example, the CheckButton class implements a button that can be on or off. The attributes of a CheckButton object specify the button's label, the style of check to be used, whether the button is currently on or off, etc. A CheckButton object also knows how to process user input events and redraw itself on the screen. The Grep Service client UI is represented as a single composite object that contains sub-objects of class ListEdit for displaying the lists of sub-section titles and file names and PushButton objects for the various buttons.

Application functionality is coded in NIC's interpreted programming language, NIC Script. The Grep Service client UI uses NIC Script actions to transfer sub-sections between lists, to construct requests sent to Grep Service, and to display files in Mosaic on the user's workstation. NIC Script is implemented by several subclasses of Object that provide a complete set of programming constructs. For example, the While class implements a while loop. An object of class While stores the code for the loop's condition in its first array element and the code of the loop's body in its second array element. A segment of NIC Script code is represented as a composite object containing sub-objects for the statements to be executed. Some classes that represent UI components have attributes whose values are segments of NIC Script code. For instance, all buttons have an Action attribute whose value is the code to be executed when the button is pressed.

The objects that comprise the user interface and the application code communicate through shared variables. The variables used in an application's NIC Script code are accessible to its interface components. Any part of an interface object can be constrained to take on the value of a NIC Script variable, thus allowing the UI to access application data and the application to access UI objects.

A complete client UI can be represented as a single composite NIC object. Although possible, it would

be unreasonable to expect programmers to create a client UI by manually creating the object that represents the client. For this reason, a direct manipulation interface editor is provided that allows programmers to create complex programs in a much more intuitive fashion.

Object's method interface provides methods for manipulating the array part and attribute list of any object. All subclasses of Object must support this generic set of object-manipulation methods, but are free to implement them in a manner appropriate for the class. This global support for accessing an object's state allows NIC to provide generic facilities for converting any object to an intermediate form that can be transported over a network or stored in a file. Since all three components of a client UI (i.e., user interface, code, and data) are represented as NIC objects, each can be represented in this distributable, storable form. We currently provide two external forms for NIC objects: the textual human-readable form shown in Figure 3 and a binary form based on XDR (Comer & Stevens, 1993).

5. CLIENT UI VIEWER

NICUI is the client UI viewer provided with NIC. NICUI consists of two main parts, a Dialog Manager and a NIC Script Interpreter. Given an object representing a client UI, the Dialog Manager instantiates the client's interface and proceeds to process interactive events. When the user takes some action that requires application code to be executed, the Dialog Manager invokes the NIC Script Interpreter to execute the code. In the Grep Service client UI, the action on the "View Document" button is a segment of NIC Script that constructs a URL for the selected file and loads the contents of the URL into a Mosaic window. The grocery store client might include NIC Script actions to search the store's inventory or compute sales tax, and the grades service might include NIC Script actions to plot student scores or compute final grades.

This ability to download and execute NIC user interfaces over a network is of great value even if the program does not access remote services. We have found this capability to be especially effective as a medium for publishing educational programs. For example, we have implemented interactive programs that teach the concepts of vector addition and (R,G,B) color matching. These programs can be accessed over the WWW and executed on any machine that has NICUI. Once retrieved, these programs operate locally and involve no further network communication.

Many GUI-based programs require at least one application-specific widget. In order to support the distribution of such programs, NIC provides a mechanism for writing new widgets completely in NIC Script. This allows application-specific widgets to be distributed with programs that require them.

6. CLIENT UI / SERVICE COMMUNICATION

During its execution a client UI will make and break connections with one or more remote services. A model of client UI/service communication must be designed in such a way that network connections are easy for client UI programmers to create and use. Two main issues must be addressed. First, simple mechanisms for establishing network connections need to be provided. Second, the form of communication between clients and services must be as general, easy to use, and efficient as possible. These issues are discussed in this section.

6.1 CONNECTION ESTABLISHMENT

NIC Script provides a Connection class that is used by client UIs to communicate with remote services. Given a WWW URL describing the host and service, Connection's ServiceConnect method establishes a connection with the service. ServiceConnect extracts the host name from the URL and makes a connection with the Service Manager process on that host. The Service Manager listens on a well-known network port for incoming connection requests. Having connected to the Service Manager, ServiceConnect sends the URL to the Service Manager. The Service Manager uses the extension of the URL to determine which service the client wants a connection with. Having determined the appropriate service for the incoming request, the Service Manager spawns a process for the service (if necessary) and sends the new service process the number of an unused network port on which it should listen for incoming client requests. The Service Manager then returns the service's port number to the client. ServiceConnect then breaks its connection with the Service Manager and establishes a connection with the remote service on the port received back from the Service Manager. At this point a client connection has been established with the service. Note that all of this protocol is invisible to those who create client UIs.

The Service Manager supports three classes of service: PerService, PerObject, and PerConnection. Each service is given one of these types. A type of PerService specifies that at most one instance of that service should exist on the host at a time. A type of PerObject specifies that all clients who connect to a service using the same URL are actually connected to the same service instance (i.e., there is one instance for each distinct URL used to connect to it). A type of PerConnection specifies that a new instance of the service is created for each individual connection.

6.2 CLIENT REQUESTS

A client request to a service takes the form of a NIC Script program. This program can be as long or short as the client desires (a one-line request script would be equivalent to a remote procedure call). Upon receiving a script from a client, a service evaluates the script and returns the result to the client. A service simply listens to its client connections and evaluates the scripts received over those connections.

Since all scripts must be executed in the context of a NIC Script variable table, a service maintains a variable table for each client connection. The values in a client's variable table persist across requests, thus allowing values created by a request to be accessed by later requests. A service consists of the NIC Script Interpreter, which it uses to evaluate client request scripts, and a variable table for each client connection. A client sends a request to a service using the Exec method on the Connection class. Exec accepts a script as its only parameter and sends that script to the Connection's service for execution. Exec can be called synchronously or asynchronously. An asynchronous call to Exec accepts two parameters. The first parameter is the script to be executed by the service. The second parameter is a script to be executed in the client UI when the return value is received from the service. This gives the client UI flexibility in how it interacts with a service.

The ability to send complete programs to a service for execution has several benefits. First, it allows client requests to include arbitrarily complex logic, thus allowing a client request to make decisions while executing within the service. Second, it allows multiple operations to be packaged as a single request. Third, it avoids passing values back and forth over the network unnecessarily when the result of one operation is used as input to a subsequent operation. The most important benefit is that this protocol is extremely simple for designers of client UIs. It is, simply, build a NIC Script object, send it to the service, and process the result. It is also important to note that generic client UI viewers (like NICUI) can build scripts containing classes only implemented by the service, thus allowing a client UI viewer to provide access to capabilities far beyond what is implemented in the viewer itself.

7. TOOLS FOR BUILDING INTERACTIVE NET-SERVICES

NIC provides tools for building both the client UI and the service itself. Client UIs are built using NIC's direct manipulation interface editor. Interfaces created using this editor can be published on the WWW just like other types of files.

The thing that distinguishes one NIC service from another is the set of classes it makes available to client scripts. These classes can be implemented in either NIC Script or C++. NIC provides a class pre-processor that automatically generates the C++ code necessary to make C++ service classes available from NIC Script.

NIC also provides a function called NICMain that encapsulates all client connection handling. A service implementor does not need to know anything about handling network connections. When a service is started, it performs any necessary initialization and then simply calls NICMain. NICMain handles the establishment of client connections and the execution of client scripts received over those connections.

Creating a new NIC service requires two steps: 1) implement any special classes to be provided by the service in NIC Script and/or C++ using the class pre-processor and 2) call NICMain from the service's main function to manage the client connections. A programmer can build a service without doing any network programming.

8. ACCESS CONTROLS

NIC's interactive net-service architecture is based on computers exchanging and executing NIC Script code received from remote sources. This provides substantial opportunity for malicious or careless

programmers to do damage to other's systems. For this reason NIC provides some controls over the access given to executing NIC Script programs.

Suppose, for example, that a service wanted to prevent computationally expensive requests. Not allowing clients to include While or For objects in request scripts would prevent any requests that contain loops. Restricting object classes effectively restricts the computational power of the interpreter and thus provides effective protection for the service. These restrictions are specified in the service's configuration file.

Protection is also required in the client UI viewer because commands in the client UI that execute shell commands or write files could do serious damage. Access to these operations can be allowed freely (very unsecure), forbidden (very secure), or only allowed after interactive confirmation by the user. The later option provides some protection with some useful flexibility.

9. RELATED WORK

There are several approaches that have been taken to distributing UIs over networks. X Windows (Jones, 1989) distributes application windows, but still requires all event handling and application processing to take place in the remote service. HTML forms (Berners-Lee, 1994b) provide hypertext access to forms-based interfaces. This allows event handling to take place locally, but still requires the remote service to perform all application processing. NIC's transferrable client UIs are a generalization of HTML forms that allows all dialogue management and much application processing to take place locally on the client's machine, which greatly reduces the burden placed on remote services. Display Postscript/NeWS (Gosling, 1989) provides a programmable window server that could be used to download UIs over a network, although NIC's features are provided at the toolkit level rather than at the lower graphics model and window system level.

We have never seen it used for this purpose, we believe that Tcl/Tk (Ousterhout, 1994) could be used to distribute client UIs over the WWW in a similar fashion to NIC. The Tcl-Dp extensions to Tcl could be used for client/service communication. We did not use Tcl/Tk for several reasons, the most important of which is that Tcl/Tk did not exist when we started this project. We also believe that an object-oriented architecture is the best way to achieve generality and extendibility in a toolkit architecture, and Tcl/Tk is not object-oriented.

In general, any MIME-based (Borenstein & Freed, 1993) system can be used to distribute NIC client UIs over a network. Whether it be through a MIME-based WWW viewer like Mosaic or a MIME-enabled mailer like ELM, the MIME mechanism for launching NICUI to execute NIC client UIs is the same.

10. SUMMARY

This paper describes an approach to increasing the WWW's ability to support interactive net-services. NIC provides transferrable client UIs, a client UI viewer, a general and simple protocol for client UI/service communication, tools that simplify client and service construction, and access controls.

REFERENCES

Berners-Lee, T., et. al. (1994a). The World-Wide Web. *CACM*, 37(8), (p. 76-82).

Berners-Lee, T. (1994b). Hypertext Markup Language Plus. URL:http://info.cern.ch/hypertext/WWW/MarkUp/HTMLPlus/htmlplus_1.html

Borenstein, N. & Freed, N. (1993). MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies. *In Internet RFC 1521.*

Comer, D. & Stevens, D. (1993). Internetworking with TCP/IP, Volume III. Prentice Hall.

Gosling, J., et. al. (1989). The NeWS Book: An Introduction to the Network Extensible Window System. Springer-Verlag.

Jones, Oliver (1989). Introduction to the X Window System. Prentice Hall.

Ousterhout, J. (1994). Tcl and the Tk Toolkit. Addison-Wesley.