Interactive Web Applications Based on Finite State Machines

Konstantin Läufer

Loyola University Chicago
laufer@math.luc.edu

Abstract

Interactive applications on the World-Wide Web are supported by the CGI interface, which allows transferring information from the browser to programs invoked by the server. Information is obtained through fill-out forms embedded in documents written in the HTML language and rendered by the browser as collections of user interface objects. While the World-Wide Web is based on the stateless HTTP protocol, state can be simulated by encoding it in the information transmitted between browser and server. Many interactive applications have a finite number of states and perform state transitions based on the user's response. Finite state machines are a suitable modeling technique for user interfaces of such applications. We are developing W++, an object-oriented application framework that assists in writing interactive, state-based web applications. The framework consists of classes for applications, states, and user interface components.

Keywords

World-Wide Web; HTML forms; CGI programs; interactive applications; finite state machines; class library.

1 Introduction

The World-Wide Web [11] is a distributed hypermedia information network. The user navigates through this information in mainly static, but context-sensitive ways with a browsing tool [7]. Browsers are client programs that run on the user's local machine, request information from server programs on remote machines, and display the information to the user.

Support for interactive applications on the World-Wide Web is provided in the form of the Common Gateway Interface (CGI) [2]. This interface allows transferring information from the client back to the server, where the information can be further processed by programs invoked by the server on the remote machine. Fill-out forms [6] are a convenient way of requesting information from the user for processing on the server. These forms can be embedded in web documents written in the Hypertext Markup Language (HTML) [5] and are rendered by the browser as collections of user-interface interaction objects.

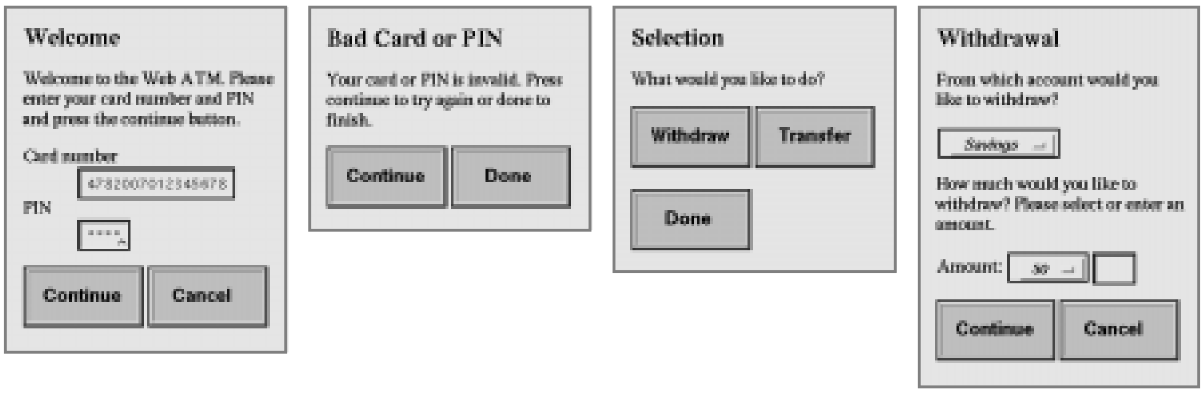
Interactive applications typically have a finite number of states and perform transitions from one state to another based on the user's response. Thus finite state machines are a suitable model for user interfaces of such applications. By contrast, the World-Wide Web is based on the stateless Hypertext Transfer Protocol (HTTP) [9]: the web server's response depends only on the command it receives. Nevertheless, there are techniques to simulate state in this stateless environment by encoding it in the information transmitted to and from the server [10, 13]. In this way, state exists only in the user's mind, while the application runs only to perform a transition from one state to the next. An alternative approach to interactive web sessions uses a translation server [15] to handle the translation between stateless protocols and stateful ones such as library systems.

We are developing W++, an object-oriented application framework that assists in writing interactive, state-based web applications. The framework consists of a class hierarchy for user interface components of web documents and abstract classes for states and applications. We illustrate the feasibility of our approach using an automated teller machine as a running example.

Even though interactivity on the web is soon to be supported directly by developments such as HotJava [4] and Virtual Reality Modeling Language (VRML) [8], we argue that a light-weight technique suitable for current browsers and network speeds is quite useful.

2 Modeling Applications as Finite State Machines

A large number of interactive applications are dialog-based [14]. An application first enters a designated


Welcome

Welcome to the Web ATM. Please enter your card number and PIN and press the continue button.

Card number 4782007012345678

PIN ****

Bad Card or PIN

Your card or PIN is invalid. Press continue to try again or done to finish.

Selection

What would you like to do?

Withdrawal

From which account would you like to withdraw?

Savings

How much would you like to withdraw? Please select or enter an amount.

Amount: 50

Figure 1: The user's view of the ATM

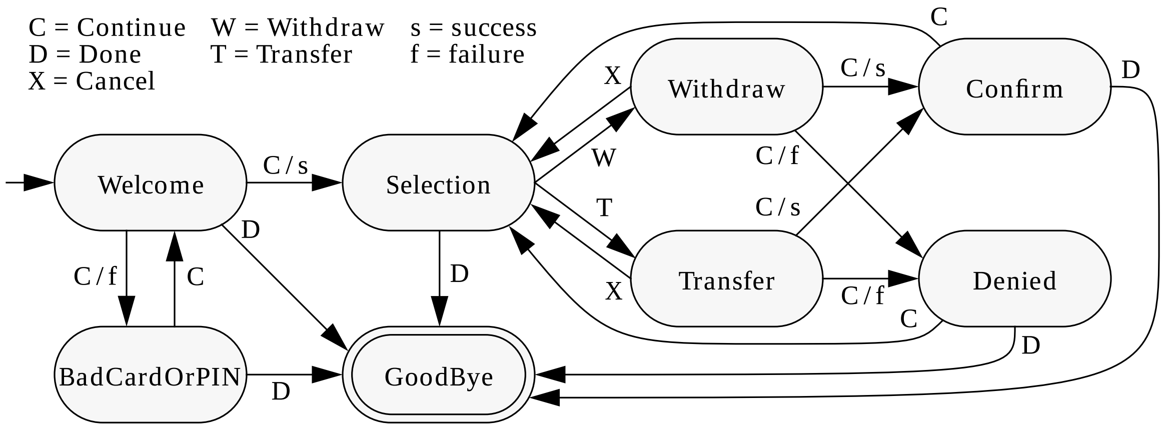


Figure 2: The ATM as a finite state machine

initial state. Then, in its current state, the application waits for an event such as a response by the user. The application then selects a transition to another state based on the event received. The dialog ends as soon as a designated final state is reached. We do not consider other events, such as time-outs, since they require a multi-threaded application model and cannot be handled easily in a stateless protocol.

We illustrate the finite state model by an example of an automated teller machine (ATM). When approaching the machine, the user first sees a welcome screen associated with the initial state. The user then inserts his or her ATM card and keys in the corresponding personal identification number (PIN). The user now triggers a transition by pressing either the "continue" or the "cancel" button. If the user presses the "cancel" button, the machine simply returns the card. If the uses presses the "continue" button, the machine shows either a rejection screen or a selection screen based on whether card and PIN are valid. From the selection screen, the user can request a transaction such as withdrawal. The user's view of this dialog is shown in Figure 1.

Formally, this dialog can be modeled in terms of the underlying finite state machine. There is one state for each user screen, including the initial welcome state, the rejection state, the selection state, a state for each type of transaction, confirmation and denial states, and a final good-bye state. Events are functions of user responses and input values. For example, the event that takes the machine from the welcome state to the selection state occurs when the user presses the "continue" button and the card and PIN are valid. Finite state machines can be represented as directed graphs, where states are nodes and transitions are edges labeled with the events that trigger them. The finite state machine for the ATM is shown in Figure 2.

3 Putting Interactive Applications on the Web

Following ideas from the W3Kit [10] and Gleeson and Westaway [13], our aim is to use the World-Wide Web as a platform for interactive applications by creating an illusion of state. The web provides two ingredients to support interactive applications; we discuss them in the rest of this section.

Fill-out Forms

The usability of interactive applications depends on having suitable interactor objects that present information to and request information from the user. Various browsers provide this functionality via fill-out forms [6], typically by giving access to the interactor objects of the toolkit used to build the browser. For example, the Mosaic browser [7] gives access to various Motif widgets [12], including text entry fields, toggle buttons, radio buttons, menus, scrollable lists, and sensitive images. While fill-out forms are embedded in web documents residing on the server, the actual rendering of the form is entirely up to the browser. Besides interactor objects, fill-out forms provide special *hidden* fields that are invisible to the user, but allow information to be encoded in forms.

The Common Gateway Interface

The Common Gateway Interface (CGI) [2] allows transferring information from the client back to the server, where the information can be further processed by programs invoked by the server on the remote machine. In particular, this gateway can be used to process information obtained from the user through a fill-out form. This works as follows: The form specifies a CGI program to which the information is to be submitted. Once the user submits the form, the CGI program starts up and receives the information as name-value pairs via either environment variables or the standard input. The program then parses the information, processes it as desired, and terminates.

Simulating State

The problem we are facing is that the HTTP protocol is stateless. The response by the server generally depends only on the user's request, not on the history of requests. However, interactive dialogs depend on a notion of state. The W3Kit [10], an object-oriented framework for interactive web applications written in Objective-C, uses a technique for creating an illusion of state by encoding the current state in hidden fields of fill-out forms transmitted between the browser and the CGI program. When a form is submitted, the indicated CGI program starts up and reconstructs the current state. The program then evaluates the information and sends back to the browser a form that again encodes the entire resulting state in a hidden field. When the program terminates, the browser shows the form corresponding to the new state. Although the program is invoked only to perform transitions from one state to the next, the user has the illusion of a continuous interactive application.

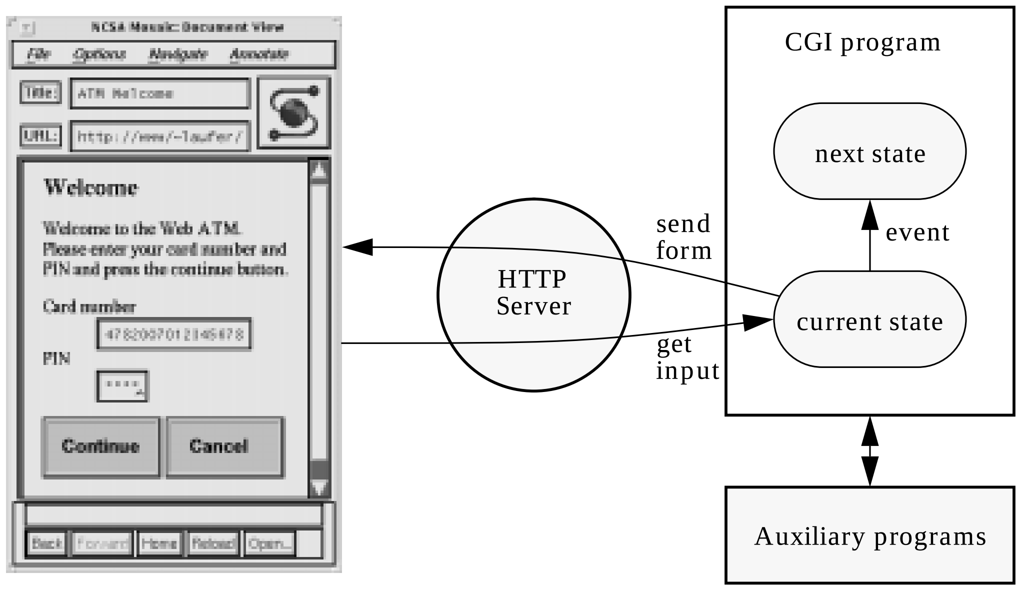
Simulating Finite State Machines

In the W3Kit approach, the CGI program encodes an entire form in a hidden field, so that it can later decode the hidden field and reconstruct all interactor objects in the form. While this approach allows for high flexibility, it requires the server to send and receive large chunks of information and the CGI program to do much work encoding and reconstructing forms.

We argue that a considerably simpler approach than the one used in the W3Kit suffices for a large class of applications that can be described by finite state machines. The idea is that we need to encode only the *name* of the current state in a hidden field instead of the entire form associated with this state. This reduces the amount of information transmitted and eliminates the need to encode and decode entire forms. The CGI program reconstructs the current state based on the value of the hidden field. It then maps the input submitted by the user to the interactor objects associated with the current state and figures out which event in the finite state machine has occurred. The program then performs the transition to the resulting state and sends the corresponding form to the browser. This infrastructure is illustrated in Figure 3.

4 W++: An Object-Oriented Web Application Framework in C++

We are currently developing W++, an object-oriented framework in C++ for programming interactive web applications based on finite state machines. To our knowledge, this is the only such framework in C++ at present. Existing frameworks include the W3Kit [10] in Objective-C and the CyberNation classes [3] in C++. While the CyberNation classes provide parsing CGI input and writing HTML forms, they do not support object-oriented application development. Most other CGI software [1] is written in Perl, C, and Tcl.


NCSA Mosaic: Document View

Title: ATM Welcome

URL: http://www/~laufer/

Welcome

Welcome to the Web ATM.
Please enter your card number and PIN and press the continue button.

Card number: 4782007012345678

PIN: ****

Figure 3: Simulating state on the web

We now introduce the main classes of the W++ framework.

- class `Application`

  This class describes an interactive application. Specific applications are created by subclassing from this class. States and transitions are added by using the protected methods in the constructor of the subclass. In the main program, we define an instance of the subclass and invoke the `run` method. The `run` method starts by parsing the CGI input coming from the environment and the standard input. If the current invocation of the application is the initial one, the `run` method brings the application to its initial state. Otherwise, it performs the transition to the next state and sends the resulting form.

- class `State`

  This abstract class describes a single state corresponding to a user screen. To build specific states, we subclass and override the `create` and `event` methods. The `create` method sets up a user screen by inserting appropriate interactor objects of class `Component` into the HTML document for that screen. The `event` method evaluates the user input and returns the resulting event. Both methods are invoked by the auxiliary methods `initialState` and `nextState` in class `Application`. In this way, the states need not know about each other.

- class `Component`

  This abstract class is the root of the HTML component class hierarchy. Components know how to represent themselves in HTML. There are various subclasses for specific components, including passive components such as ordinary HTML text and reset buttons as well as active interactor objects such as text fields, buttons, and menus. Active components keep track of the corresponding values entered by the user. Some interactor objects, such as special submit buttons and sensitive images, result in the submission of the form. These objects know whether or not they were invoked. Sensitive images also keep track of the coordinates of a mouse click that occurred on them.

- class `Event`

  This auxiliary class encapsulates events, which are uniquely identified by their name.

5 An Automated Teller Machine in W++

We argue that our approach is feasible by building the ATM application within the W++ framework. First, we create a subclass of class `State` for each state in Figure 2. In each subclass, we define instance variables that point to the interactor objects. Now we override the `create` method to set up the screens seen by the

```
class Application {
public:
    void run();
    void parseInput();
    void printForm();
protected:
    Application();
    void addInitial(State*);
    void addState(State*);
    void addTrans(State*, char*, State*);
    Event* event(const char*);
private:
    bool initialRequest();
    void initialState();
    void nextState();
    ...
    State* current;
};

void Application::run() {
    parseInput();
    if (initialRequest())
        initialState();
    else
        nextState();
    sendForm();
}
```

```
class State {
public:
    State();
    virtual void create();
    virtual Event* event();
    void parseInput();
    void printForm();
protected:
    void addTitle(const char*);
    void addComponent(Component*);
    void addComponent(const char*);
private:
    List<Component> components;
};

class Component {
public:
    void printForm();
};

class Active : public Component {
public:
    void getValue();
    const char* value();
};
```

Figure 4: The main classes of the W++ framework

user. Strings are inserted literally as HTML text, while the other components are inserted as interactor objects. We override the `event` method to decide what event corresponds to the user input. For example, there are two possible events resulting from pressing the "continue" button on the welcome screen, depending on the validity of the card and PIN. Next, we create a subclass for this application. Pointers to the individual states are declared as instance variables. We provide a constructor to create the states and transitions in Figure 2. Finally, we create an instance of the `ATM` class and invoke the `run` method.

References

[1] A CGI programmer's reference. http://www.halcyon.com/hedlund/cgi-faq/.

[2] The Common Gateway Interface. http://hoohoo.ncsa.uiuc.edu/cgi/.

[3] Cybernation guest book demo. http://penny.ibmpcug.co.uk/cybercon/guestbk.htm.

[4] HotJava home page. http://java.sun.com/.

[5] Hypertext Markup Language (HTML): Working and background materials. http://www.w3.org/hypertext/WWW/MarkUp/.

[6] Mosaic for X version 2.0 fill-out form support. http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/Docs/fill-out-forms/overview.html.

[7] NCSA Mosaic home page. http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/.

[8] NCSA Relativity Group VRML page. http://jean-luc.ncsa.uiuc.edu/Viz/VRML/.

[9] Overview of HTTP. http://www.w3.org/hypertext/WWW/Protocols/.

[10] W3Kit. http://www.geom.umn.edu/software/download/w3kit.html.

[11] T. Berners-Lee, R. Cailliau, A. Luotonen, H. F. Nielsen, and A. Secret. The World-Wide Web. *Communications of the ACM*, 37(8):76–82, 1994.

[12] P. Ferguson and D. Brennan. *Motif Reference Manual*, volume 6B of *The Definitive Guides to the X Window System*. O'Reilly, 1993.

[13] M. Gleeson and T. Westaway. Beyond hypertext: Using the WWW for interactive applications. Technical report, University of Melbourne, 1995. http://www.its.unimelb.edu.au:8001/papers/.

[14] J. Larson. *Interactive Software: Tools for Building Interactive User Interfaces*. Yourdon Press, 1992.

[15] L. Perrochon. Translation servers: Gateways between stateless and stateful information systems. In *Proc. Network Service Conference*, London, November 1994.