

An Open Environment for Common Gateway Interface Programming

Konstantin Läufer
Loyola University of Chicago
laufer@math.luc.edu

To appear in Proc. Tenth Annual Midwest Computer Conference (MCC), Chicago, March 1996

Goals

- interactive World Wide Web applications
- visual design of user interface
- support for stateful server-side programming
- using existing technology (HTTP, CGI)



Problems

- HTTP is a *stateless* protocol
- no notion of history
- CGI programming is tedious, low-level, and fragile
- CGI programming and document authoring are mixed up

Approach

- applications modeled as finite state machines
- object-oriented framework for FSM-based applications
- application runs on the server as a CGI program
- state simulated and maintained via *hidden input fields* in HTML forms
- extensions of HTML to relate application and associated documents
- tool and framework support for this HTML extension

User's View of an Interactive Application

Welcome

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

Card number

PIN

Selection

What would you like to do?

Withdrawal

From which account would you like to withdraw?

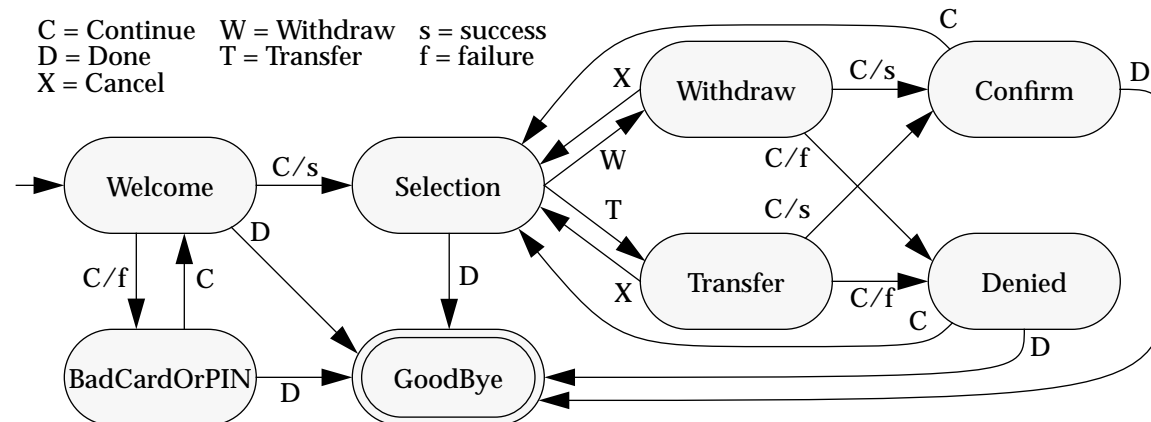
How much would you like to withdraw? Please select or enter an amount.

Amount:

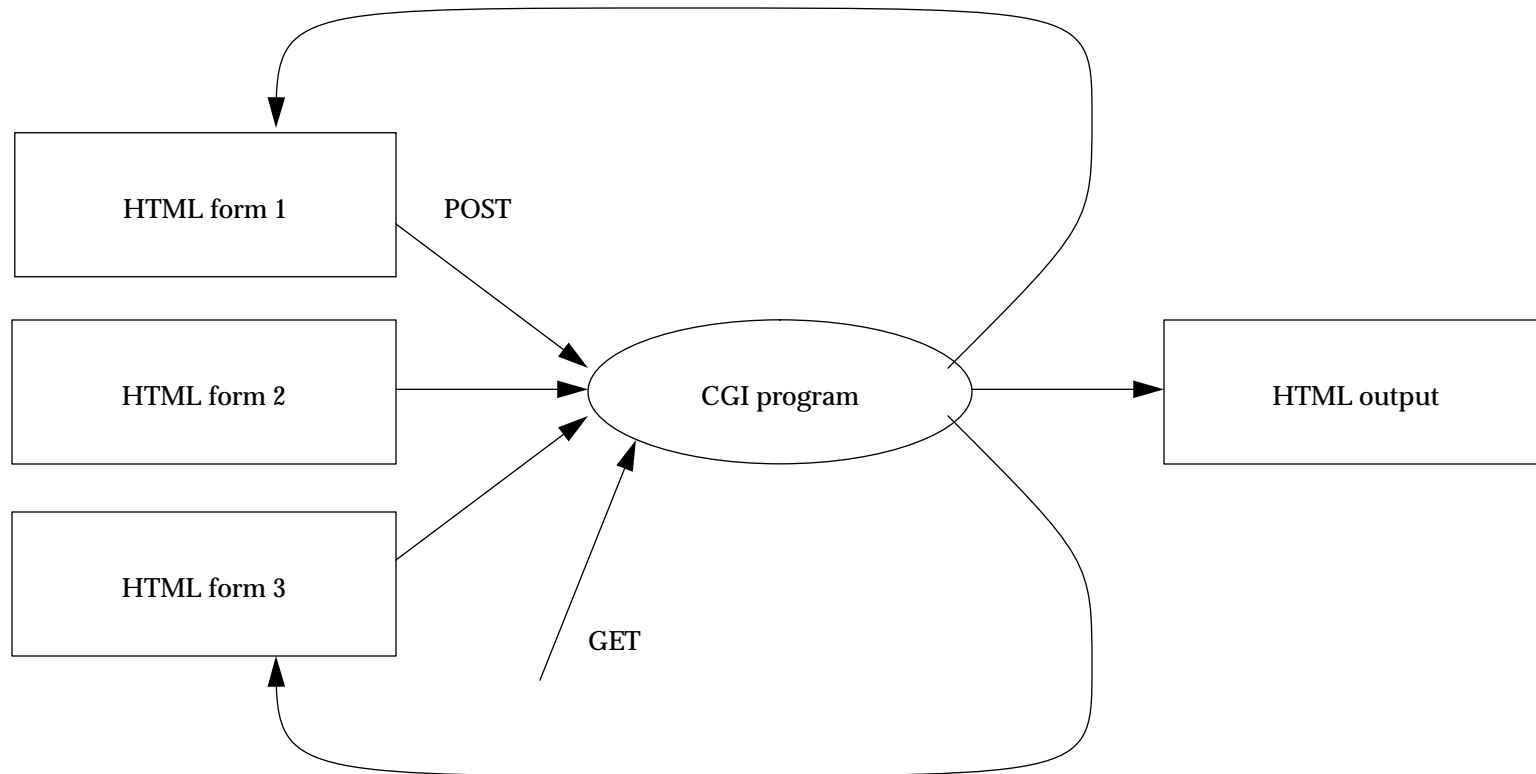
Bad Card or PIN

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

Application Modeled as a Finite State Machine

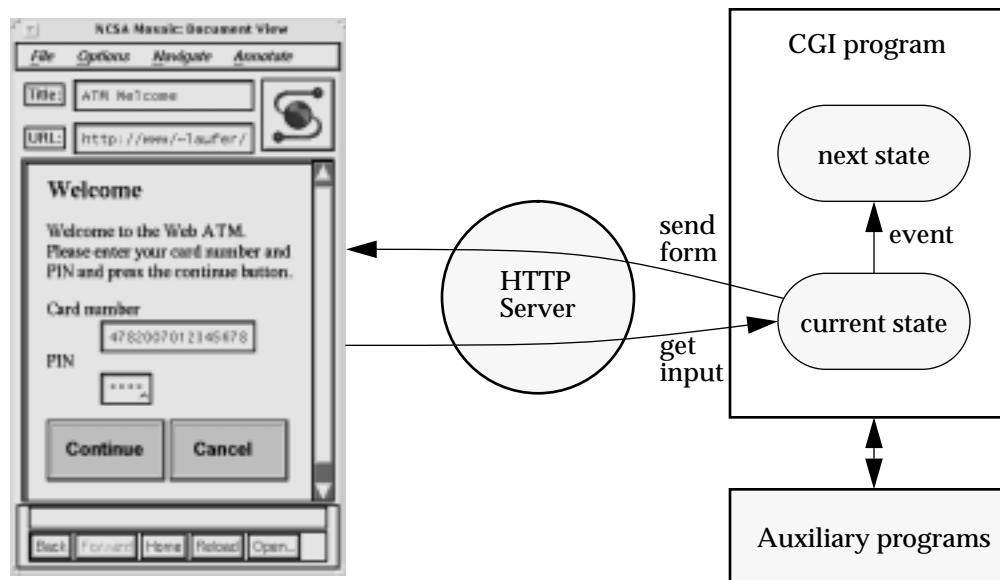


Architecture of a CGI Program



- same program, different forms (POST) or directly (GET)
- forms have different input fields; want to extract safely
- output generated depends on the input
 - entirely different forms
 - variable (structured) output in single form

Simulating State



- encode only the *name* of the current state in a *hidden field*

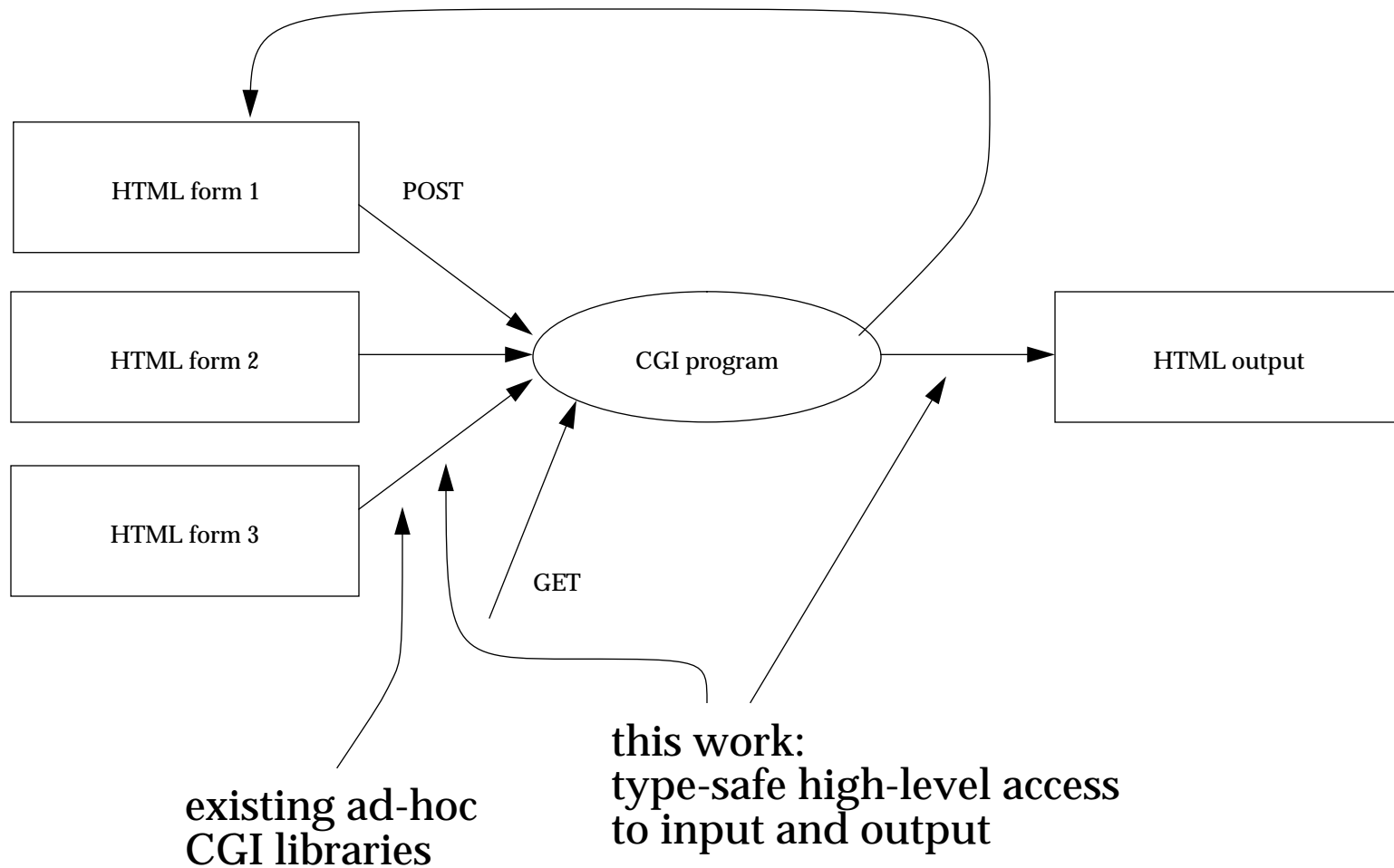
OOF for FSM-based applications

- `class Application`
- `class State`
- `class Component`
- `class Event`

Specific Goals

- want type-safe, high-level access to input and output
- want to design the output *visually* (using our favorite HTML authoring tool)
- want automatic insertion of variable output information in the right places
- want abstract views of CGI input and HTML output
- want to ignore any *fixed* contents of the HTML output document

Add-Ons to the Architecture of a CGI Program



Application Programmer's View

- CGI input of type `string` -> `string list`
(programmer has to convert this to useful types)
- HTML output in the form of an abstract type
 - create instance from existing document (parse and look for special output tags)
 - create instance from scratch
 - query for a list of fields modifiable by the programmer, for existence of a given field, and for types of fields
 - modify value of a given field

Document Author's View

- new HTML tags for output values
(parsed and made accessible to the CGI program)

```
<OUTPUT NAME="FieldName" TYPE="FieldType">
```

- needs to know allowed field types
- needs to know how field types are rendered
- other attributes possible

How to Connect the Two Views?

Two possible modes of operation:

- parse each HTML output document into a singleton object
 - each output field then becomes a member of this object
 - provides static typing and fast execution
- create one object from each HTML document at runtime
 - look up fields by their names given as strings
 - lose static typing
 - provides give higher flexibility
 - documents are available to the server anyhow

System Components

- any HTML *authoring tool*
 - extended to create special output tags where needed
- *html2cpp*
 - parses an HTML document to a C++ class
 - output fields are accessible in a type-safe way as class members
- *CGI++* class library
 - parse CGI input into a usable data structure
 - parse an HTML document for output tags on the fly
 - builder for assembling HTML documents on the fly
(using *Builder* and *Composite* patterns)
 - insertion of building blocks into output documents
(considering types specified for the corresponding output fields)

Sample Document and Generated Class

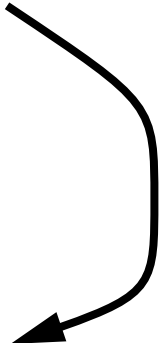
```
...text1...  
<OUTPUT NAME="SearchString" TYPE="String">  
...text2...  
<OUTPUT NAME="ResultList" TYPE="List<String>">  
...text3...
```

```
class SearchResult {  
  
    String _SearchString;  
    List<String> _ResultList;  
  
    void setSearchString(String arg) { _SearchString = arg; }  
    void setResultList(List<String> arg) { _ResultList = arg; }  
  
    void Print {  
        cout << "...text1...";  
        _SearchString.Print();  
        cout << "...text2...";  
        _ResultList.Print();  
        cout << "...text3...";  
    }  
};
```


Sample Application Program with Output

```
int main()
{
    SearchResult doc;
    List<String> results;
    // ...
    doc.SetSearchString("Toyota");
    results.Add("String 1");
    results.Add("String 2");
    // ...
    doc.SetResultList(results);
    doc.Print();
}
```

```
...
<UL>
<LI> String 1
<LI> String 2
...
</UL>
```



Dynamic Use

```
// ...
HTMLdoc d1("filename");
if (d1.HasField("SearchString") && d1.HasType("String"))
    d1.SetValue("SearchString", "Toyota");
// ...
```

Conclusion

- content (produced by CGI program) separate from presentation (HTML document)
- CGI programming as easy VB or VC application programming
- language-independent approach
- prototype with target language C++ coming up...

Related Work

- CGI libraries for Perl, Tcl, Python
- C libcgi from EIT
- HtmlWriter C++ class library from CyberCon
- W3Kit for Objective-C from Wisconsin
- MAWL!!!