Introduction to AdaCGI

David A. Wheeler
dwheeler@dwheeler.com
September 25, 2000

Outline

- Intro: myself, AdaCGI, CGI, Alternatives
- Using Ada for Web Apps (+ and -)
- High-Level: License, Basics
- Using AdaCGI: Minimal Example, Debugging and Installing
- Special: Get/Post, Cookies, Security, Recent Additions, Limitations
- AdaCGI spec & Long Example (“Search”)

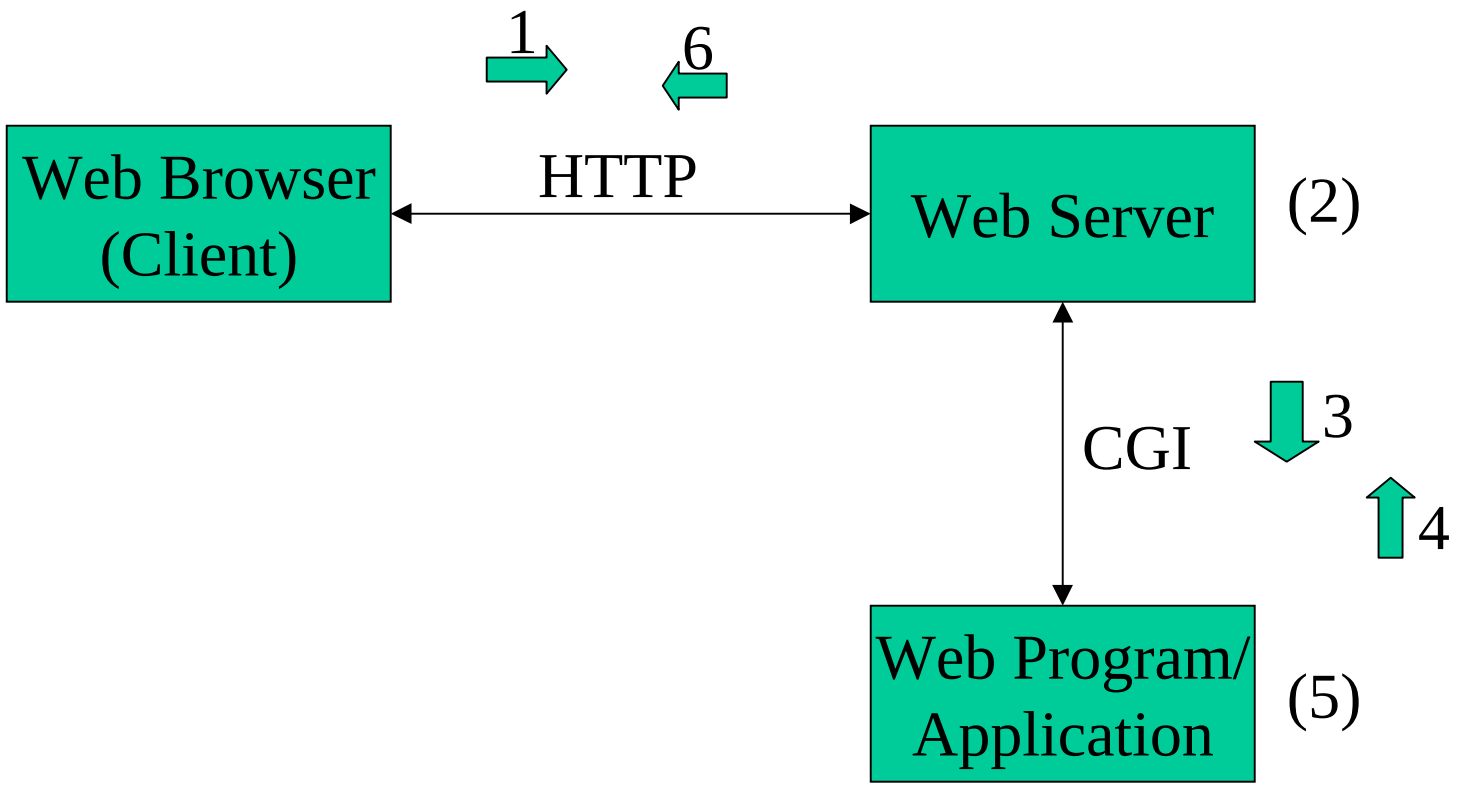
Who is David A. Wheeler?

- RSM at Institute for Defense Analyses
- Interests: Security, Linux, Ada, Internet, …
  - Secure Programming for Linux HOWTO
  - GNOME User's Guide
  - Lovelace (Ada tutorial)
  - IEEE Tutorial on Inspections
  - Linux url(7)
  - Maintain Ada mode for vim & AdaCGI
- http://www.dwheeler.com

What is AdaCGI?

- Ada Library implementing the "Common Gateway Interface" (CGI)
  - CGI is the most common interface for web-enabled programs ("web apps")
  - CGI is language & platform neutral
- Permits development of cross-platform web-enabled Ada programs
- http://www.dwheeler.com/adacgi

CGI: General Concept (1 of 2)



CGI: General Concept (2 of 2)

- When CGI is used, the following occurs:
  1. Web browser sends request to web server
  2. Web server determines that it must start web application; determines which one & starts it
  3. Web application starts, loads data sent by web server (primarily as keys and their values; keys can duplicate)
  4. Web application responds (via stdout), with a header (saying what it's replying) followed by data
  5. Web application exits (quits) once done
  6. Web server passes this data on to user's browser

CGI Alternatives

- FastCGI
  - Keeps web app alive (instead of restarting per request)
  - (+) Better performance (eliminates startup)
  - (-) More work developing app (must reset *all* state)
  - (-) Less robust (app must survive many requests)
- Server-specific (proprietary) APIs
  - (+) Even better performance (eliminates startup & IPC)
  - (-) Lock-in to a particular web server
  - (-) Even less robust (error may take down server)

AdaCGI Alternatives

- Doug Smith's WebAda CGI
  - Derived from an old version of AdaCGI
  - (+) generic iterators, can remove keys, re-parser
  - (-) no cookies, complex use, buggy decoders, little doc, unmaintained
  - We've agreed that I'll remerge his into mine (in time)
  - www.adasmith.com/webada/source
- Binding to C (ugh)
- Un-CGI
  - Converts CGI data to environment vars "WWW_name"
  - No cookies, multivalue keys ambiguous, slow, ugh

Why use Ada for Web Apps?

- Excellent Run-Time Performance
  - better than interpreters (Perl), can be > typical JVM
  - CGI low performance, so relevant iff compute-bound
- Excellent compile-time checking
- Highly readable (especially vs. Perl)
- Increased security over C/C++ (bounds checking)
- Prefer Ada
- Have existing Ada applications

Weaknesses of Ada for Web Apps

- Wordiness (not best for short scripts)
- Less convenient string handling
- Regular expressions not built-in
  - Can use GNAT's library, but fewer capabilities and can't optimize like Perl
- Fewer web-app-specific and related support libraries
- Often, must separately install Ada library

AdaCGI License

- AdaCGI is free, free software, open source
- Open Source License: LGPL + 2 clauses:
  - "GNAT clause": don't need to distribute separate object files
  - Web users must be able to get and redistribute your version of the AdaCGI library
- Can use to develop proprietary programs, but the AdaCGI library must stay open

AdaCGI Basics

- "with CGI": initialization autoloads data
- Two ways to access CGI data:
  - an associative array (given key & optional key count, returns value)
  - indexed sequence (given index, =>key or value)
- Call Put_CGI_Header to start returning data
  - by default, to return HTML
- Then send results (HTML?) to standard out
- Use String or Unbounded_String directly

“Minimal” Web App Example

```
with CGI, Text_IO; use CGI, Text_IO;
procedure Minimal is
begin
  Put_CGI_Header;   -- We will reply with a generated HTML document.
  -- Output <HTML><HEAD>….</HEAD><BODY>:
  Put_HTML_Head("Minimal Form Demonstration”);
  if CGI.Input_Received then  -- Check if input was received.
    Put_Variables;  -- Input received; show all variable values.
  else
    -- No input received; reply with a simple HTML form.
    Put_Line("<FORM METHOD=POST>What's your Name?<INPUT
      NAME=""name""><INPUT TYPE=""submit""></FORM>");
  end if;
  Put_HTML_Tail;  -- End HTML doc, sending </BODY></HTML>
end Minimal;
```

Debugging/Testing a Web Program

- For debugging & test scripts, can start directly:
  - setenv REQUEST_METHOD GET
  - setenv QUERY_STRING *key1=value1&key2=*...
  - compile, link, run (“./minimal”)
- Output should look like:

Content-type: text/html

<HTML><HEAD><TITLE>Minimal Form Demonstration</TITLE>
</HEAD><BODY>
<FORM METHOD=POST>What's your Name?<INPUT NAME="username">
<INPUT TYPE="submit"></FORM>
</BODY></HTML>

Installing a Web Program

- To really use it, set up with web server, e.g.:
  - su
  - cp minimal /home/httpd/cgi-bin
- Run
  - http://localhost/cgi-bin/minimal
  - http://localhost/cgi-bin/minimal?
name=David%20Wheeler&
email=dwheeler@dwheeler.com
- Odd problems? Try Put_Variables

Get vs. Post

- CGI supports two sub-protocols
  - *Get*: data can be included in URLs
  - *Post*: data can be voluminous
- AdaCGI supports both, merging them
  - Can use *either* subprotocol at any time
  - API hides difference; access data the same way
  - If you need to know, AdaCGI will say which, but using this difference is not recommended

Cookies

- Cookies are small pieces of data
  - Sent by the server
  - Stored by the client and sent back when the client re-communicates with the server
- Often used in e-commerce (the cookie is an ID indicating the transaction we're in)
- Potential privacy risk: Permits servers to track users (loss of anonymity)

Security Issues

- CGI data comes from untrusted users
  - identify legal values, and prohibit anything not meeting the legal criteria (min, max, patterns, etc.)
  - don’t assume that these values are trustworthy (“price”)
  - in particular, never trust a filename or directory name
  - you may need to escape all metacharacters
  - NIL character
- See the many documents available on CGI programming security

Additions in Version 1.5

- AdaCGI Version 1.5 added the following:
  - Cookie_Count
  - HTML_Encode: & becomes &
  - URL_Encode/Decode: % becomes %25
  - Generic Iterators
  - Key_Value_Exists: Has Key been given Value?

Important Limitations

- No separate IndexedKey ADT/OO type for parameters & cookies
  - parse, modify, remove, save, reload
- Doesn’t support file uploads
  - Rarely used in practice
- Only supports GET/POST commands
  - Others useful for web maintenance (WebDAV)

Less Important Limitations

- Auto-initializes data on startup
- Doesn’t support FastCGI
  - Usually implemented separately anyway
- Doesn’t auto-gen form with initial values
  - Can be done with a higher-level package
- String/Unbounded_String awkward
- Some subprogram names too similar
- Could be broken into multiple packages (?!)

AdaCGI Spec: Get-by-key

- function Value(Key : in String;
  Index : in Positive := 1;
  Required : in Boolean := False)
  return Unbounded_String;
- function Key_Exists(Key : in String;
  Index : in Positive := 1)
  return Boolean;
- function Key_Count(Key : in String)
  return Natural;

AdaCGI Spec: Get-by-Position

- function Argument_Count return Natural;
- function Key(Position : in Positive) return String;
- function Value(Position : in Positive)
  return String;

AdaCGI Spec: Starting Output

- procedure Put_CGI_Header(Header : in String := "Content-type: text/html");
  - Puts CGI Header to Current_Output, followed by two carriage returns.
  - This header determines the program's reply type.
  - Default is to return a generated HTML document.
  - Warning: Make calls to Set_Cookie before calling this procedure!

AdaCGI Spec: Generating HTML Basics

- procedure Put_HTML_Head(Title : in String;
  Mail_To : in String := "");
  - Puts an HTML header with title “Title”:
    <HTML><HEAD><TITLE> _Title_ </TITLE>
    <LINK REV="made" HREF="mailto: _Mail_To_ ">
    </HEAD><BODY>
- procedure Put_HTML_Heading(Title : in String;
  Level : in Positive);
  - Put an HTML heading, e.g. <H1>Title</H1>.
- procedure Put_HTML_Tail;
  - Put HTML tail, I.e.: </BODY></HTML>

AdaCGI Spec: Watch Out, Excessively Similar Names!

- Put_CGI_Header
  - Content-type: text/html
- Put_HTML_Head
  - <HTML><HEAD>…</HEAD><BODY>
- Put_HTML_Heading
  - <H1>…</H1>

AdaCGI Spec: Generating HTML Miscellania

- procedure Put_Error_Message(Message : in String);
  - This Puts an HTML_Head, an HTML_Heading, the message, and an HTML_Tail.
  - Call "Put_CGI_Header" before calling this.
- procedure Put_Variables;
  - Put to Current_Output all of the CGI variables as an HTML-formatted String.

AdaCGI Spec: Miscellaneous

- function Input_Received return Boolean
- function My_URL return String;
- function Get_Environment(Variable : in String) return String;
- Line_Count, Line_Count_of_Value, Line, Value_of_Line: handle multi-line values

AdaCGI Spec: Cookies (new feature of AdaCGI 1.4)

- Set_Cookie(Key : String;Value : String;
  Expires : String := "";
  Path: String := …; Domain: String := …;
  Secure: Boolean := False);
  - Sets a cookie value; call this BEFORE calling Put_CGI_Header.
- function Cookie_Value(Key : in String;
  Index : in Positive := 1; Required : in Boolean := False) return Unbounded_String;
- function Cookie_Value(Position : in Positive) return String;

“Search” Example: Top

```
with CGI, …; use CGI, …;
procedure Search is  ...
begin
  Put_CGI_Header;
  if Key_Exists("query") and Key_Exists("file") then
    Process_Query;
  else
     Generate_Blank_Form;
 end if;
 Put_HTML_Tail;
end Search;
```

“Search” Example: Generate_Blank_Form (1 of 4)

```
   Put_HTML_Head("Text Search Form");
   Put_HTML_Heading("Text Search Form", 1);
   Put_Line("<P>You may search for a text phrase from any of the given
    files.<P><FORM METHOD=POST>");
   Put_Line("What do you want to search for:<P>");
declare
      Query_String : constant String := CGI.Value ("query");
      File_Value   : constant String := CGI.Value ("file");
  begin
     Put_Line("<INPUT NAME=""query"" SIZE=40");
     if Query_String /= "" then - - if query set, use as default
        Put(" VALUE="); Put(String'(Value("query")));
     end if;
     Put_Line("><P>");
```

“Search” Example: Generate_Blank_Form (2 of 4)

```
   - - if file set, then save it in form & display its value.
   - - otherwise, let the user select the file to search.
   if Key_Exists("file") and File_Value /= "" then
      Put("<INPUT TYPE=""hidden"" NAME=""file"" VALUE=""");
      Put(String'(Value("file") & """>"));
      Put("<P>You will be searching file <I>");
      Put(String'(Value("file")));
      Put_Line("</I><P>");
   else
      Put_Line("Where do you want to search?<P>");
      Put_Select_List;
   end if;
end; -- declare block
```

“Search” Example: Generate_Blank_Form (3 of 4)

```
- - if “casesensitive” set, save in form invisibly, else ask user
if Key_Exists("casesensitive") then
  Put_Line(String'("<INPUT TYPE=""hidden""
 NAME=""casesensitive"" VALUE=""" & Value("casesensitive") &
 """>"));
else
  Put_Line("Do you want this search to be case-sensitive?");
  Put_Line(”<DL><DD><INPUT TYPE=""radio""
    NAME=""casesensitive" " VALUE=""yes""> <I>Yes.</I>");
  Put_Line("<DD><INPUT TYPE=""radio"" NAME=""casesensitive""
    VALUE=""no"" CHECKED> <I>No.</I>");
  Put_Line("</DL>");
end if;
```

“Search” Example: Generate_Blank_Form (4 of 4)

```
- - Generate submit and reset buttons for form
Put_Line("<P> <INPUT TYPE=""submit"" VALUE=""Submit
  Query""">");
Put_Line("<INPUT TYPE=""reset""> ");
Put_Line("</FORM>");
```

“Search” Example: Process_Query (1 of 2)

```
procedure Process_Query is
   User_File_To_Search : constant String := CGI.Value("file");
   File_To_Search : constant String :=  - - Don’t trust user to set this!
                              Real_File_Name(U(User_File_To_Search));
   Pattern    : constant String := Value("query");
   Case_Sensitive : Boolean := False;
   Case_Sensitivity : constant String := Value ("casesensitive");
 begin
   Put_HTML_Head("Query Result");
   Put_HTML_Heading("Query Result", 1);
   Put_Line(String'("<P>The search for <I>" & Value("query") &
    "</I>"));
   Put_Line(String'(" in file <I>" & Value("file") & "</I>"));
```

“Search” Example: Process_Query (2 of 2)

```
  if Case_Sensitivity = "yes" then
    Case_Sensitive := True;
    Put_Line(" in a case-sensitive manner");
  end if;

  Put_Line("produced the following result:<P>");
  Put_Line("<PRE>");
  Flush;
  Put_Matches(File_To_Search, Pattern, Case_Sensitive);
  Put_Line("</PRE>");
end Process_Query;
```

In Conclusion...

- You can make web apps in Ada!
  - Get it at http://www.dwheeler.com/adacgi
  - EMailing: adacgi-list-request@adapower.com message body “subscribe”
  - Linux packaging: http://www.gnuada.org
- Use the current version of AdaCGI
  - Currently version 1.5
- Patches welcome
- Go forth & have fun!