Safer Web servers with Ada and AWS

J-P. Rosen
Adalog

rosen@adalog.fr

AWS

- Ada Web Server

  Authors: Pascal Obry, Dmitriy Anisimkov.

- History and availability

  Project started on January 2000

  Free Software (GMGPL)

  100% Ada (except SSL based on OpenSSL and LDAP based on OpenLDAP/MS LDAP)

  Windows - GNU/Linux - FreeBSD...

  Download:

  - http://libre.act-europe/aws/ (english)
  - http://www.obry.org/contrib.html (french)
  - bleeding edge (Git): https://forge.open-do.org/anonscm/git/aws/aws.git

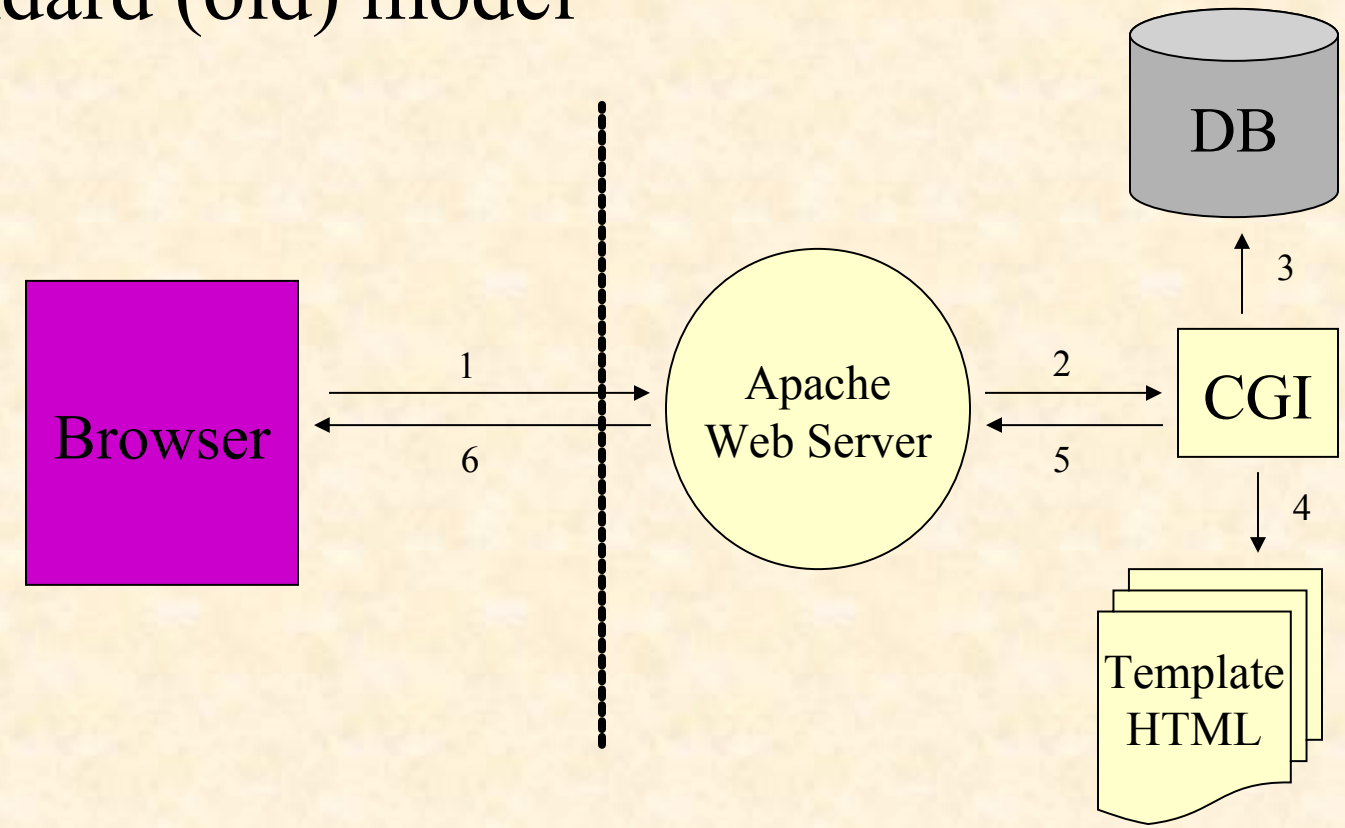
What is AWS?

82 (user) packages !

- A set of packages for managing protocols
  - http/https, SOAP, LDAP, Jabber, SMTP, POP…
  - Server side
  - Client side
- Facilities for managing pages (dispatchers)
- Facilities for building pages (templates parser, web blocks)
- Facilities for making distributed applications
- Other facilities (Resources, WSDL…)

Web Development

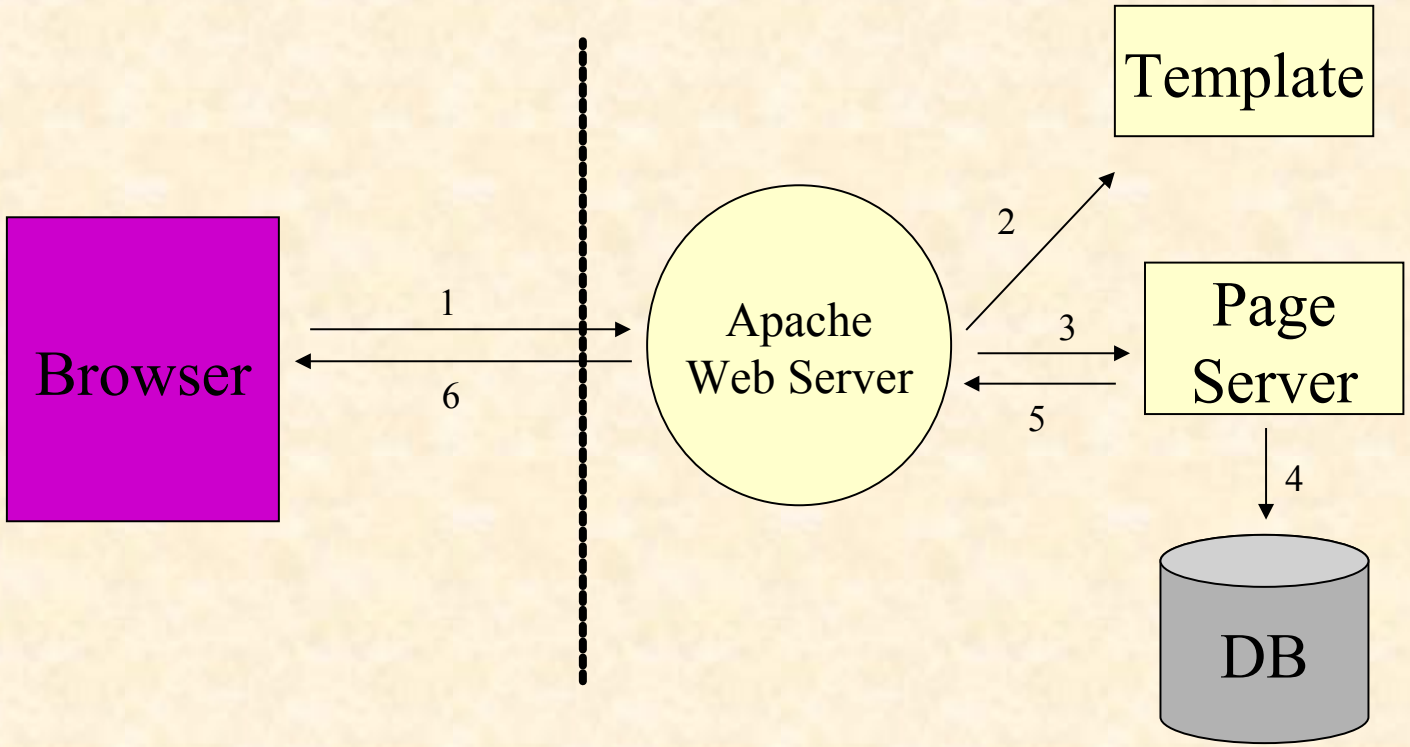
- Standard (old) model



The program is separated from the server

Web Development

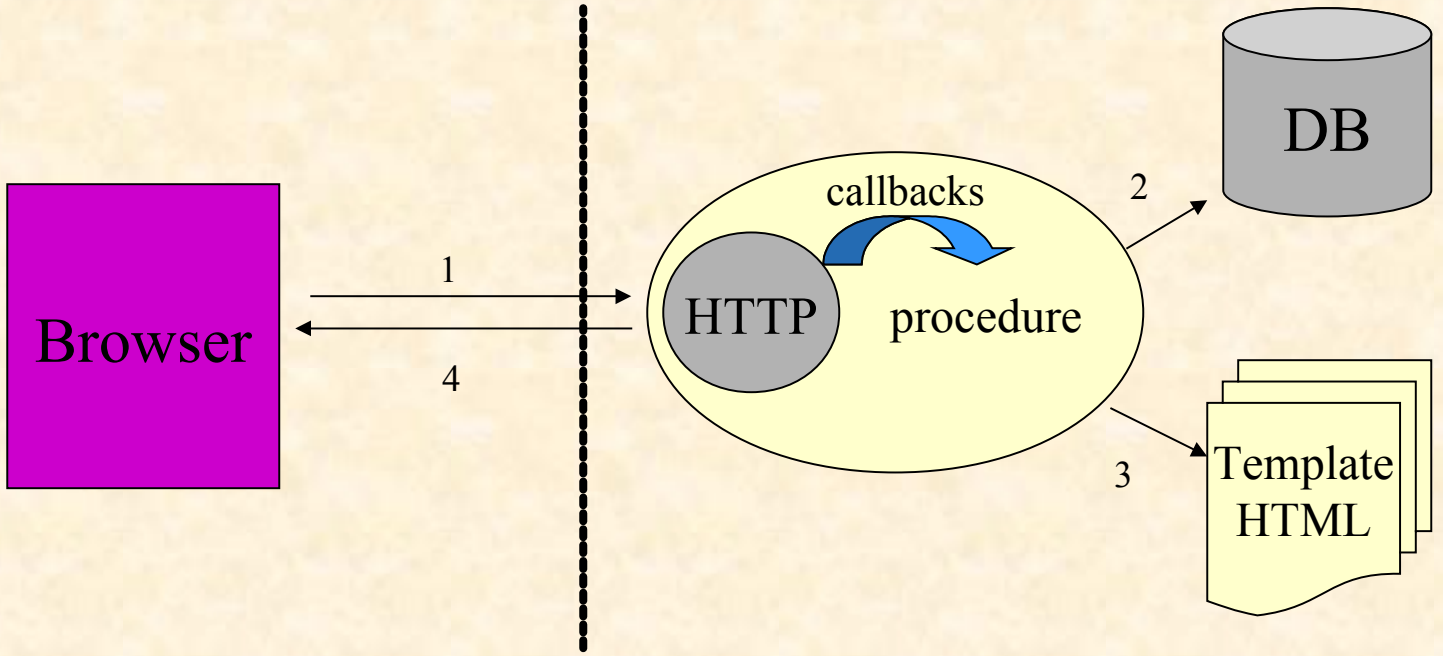
- Scripting model (Server side inserts)



The program is inside the server

Web Development

- AWS based model



The server is inside the program

What Can AWS Be Used For?

- HTTP services
  - Lightweight page server
  - Virtual site
- HTML as a Graphical User Interface
- Regular application with Web access
  - Remotely monitoring a process, an experiment…
- Client-server applications
  - HTTP communication
  - SOAP

Basic Behaviour

- AWS :

  opens the HTTP(S) message

  Gets answer using the user's callback procedure

  Encapsulates answer and sends it back to browser

```
procedure Start(Web_Server : in out HTTP;
                Callback   : in     Response.Callback;
                Config     : in     AWS.Config.Object);

type Callback is access
   function (Request : Status.Data) return Response.Data;
```

with safety!

The callback is the "script", but the language is full Ada.

Using AWS (1)

- User:

  Declare server to handle the HTTP protocol.

  Start the server (several overloaded `Start` procedures)

```
procedure Demo is
   WS : Server.HTTP;
begin
   Server.Start (WS, "demo server", Service'Access, 3,
                 "/Admin-Page", 1024,
                 Security => True, Session => True);
```

Simultaneous connections

Callback procedure

Status page

Port

HTTPS

Session handling

Using AWS (2)

- Do not exit from the main program

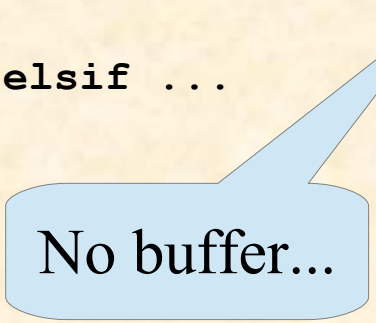
```
   ...
   Server.Wait (Server.Q_Key_Pressed);
   --  Wait for the Q key to be pressed

   Server.Wait (Server.Forever);
   -- Wait forever, the server must be killed

   Server.Wait (Server.No_Server);
   --  Exit when there is no server running (all of them
   --  have been stopped)
end Demo;
```

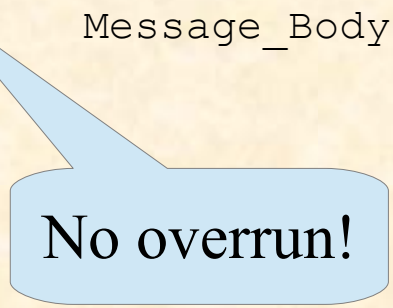
Using AWS (3)

- Develop the callback procedure which is called by the server.

  Used to provide answer for the requested URI.

```
function Service (Request : in Status.Data) return Response.Data
is
   URI : constant String := Status.URI (Request);
begin
   if URI =  "/givemethat" then
      return Response.Build (Content_Type => "text/html";
                             Message_Body => "<p>Hello there !");
   elsif ...
```





Using AWS (4)

- The form's parameters

```
function Service (Request : in Status.Data) return Response.Data is
   P_List : constant Parameters.List := Status.Parameters (Request);
   --  List of parameters
    N  : constant Natural := Natural'Value
                                (Parameters.Get (P_List, "count");

    --  Numbers is a list with multiple selections enabled
    V1 : constant String := Parameters.Get (P_List, "numbers", 1)
    V2 : constant String := Parameters.Get (P_List, "numbers", 2)
begin
    …
```

Using AWS (5)

- A response is built with one of the AWS.Response constructors.

From a string :

```
function Build
   (Content_Type  : in String;
    Message_Body  : in String;
    Status_Code   : in Messages.Status_Code  := Messages.S200;
    Cache_Control : in Messages.Cache_Option := Messages.No_Cache)
return Data;
```

From a file:

```
function File
  (Content_Type : in String;
   Filename     : in String;
   Status_Code  : in Messages.Status_Code := Messages.S200)
return Data;
```

Object Oriented AWS

- A tagged type can be used instead of a call-back function

```
package AWS.Dispatchers is
   type Handler is abstract new Ada.Finalization.Controlled
      with private;
   procedure Initialize (Dispatcher : in out Handler);
   procedure Adjust     (Dispatcher : in out Handler);
   procedure Finalize   (Dispatcher : in out Handler);

   function Dispatch (Dispatcher : in Handler;
                      Request    : in Status.Data)
   return Response.Data is abstract;
…

procedure Start (Web_Server : in out HTTP;
                 Dispatcher : in     Dispatchers.Handler'Class);
…
```

Example : Hello_World

```
with AWS.Response;
with AWS.Server;
with AWS.Status;

procedure Hello_World is
   WS  : AWS.Server.HTTP;

   function Service (Request : in AWS.Status.Data)
     return AWS.Response.Data is
   begin
      return AWS.Response.Build ("text/html", "<p>Hello world !");
   end Service;

begin
   AWS.Server.Start (WS, "Hello World",
                     Callback => Service'Unrestricted_Access);
   AWS.Server.Wait (AWS.Server.Q_Key_Pressed);
end Hello_World;
```

Example : A Static Page Server

```
function Service (Request : in AWS.Status.Data)
  return AWS.Response.Data
is
   URI      : constant String := AWS.Status.URI (Request);
   Filename : constant String := URI (2 .. URI'Last);
begin
   if OS_Lib.Is_Regular_File (Filename) then
      return AWS.Response.File
        (Content_Type => AWS.MIME.Content_Type (Filename),
         Filename     => Filename);
   else
      return AWS.Response.Acknowledge
        (Messages.S404, "<p>Page '" & URI & "' Not found.");
   end if;
end Service;
```

Secure Server (HTTPS)

- Just set Security to True in the call to "Start"
  - Uses a default certificate
  - To use another certificate:

```
AWS.Server.Set_Security (Certificate_Filename => "/xyz/aws.cert");
```

- Protocols
  - Supported : SSLv2, SSLv3
  - Unsupported : TLSv1
- Why use HTTP?
  - HTTPS is slightly slower
  - HTTPS is very hard to configure… with Apache!

The Templates Parser

- 100% code and design separation.
- An independent component…
  - but extremely useful with AWS!
- The template: a text file (or string) parameterized with
  - Commands
  - Variables (tags)
- The parser replaces tags with their values and executes commands.

Ada for the code, some HTML tags to layout the data.
No scripting in the HTML.

Tag Substitution

Template file simple.tmplt):

```
@@-- A simple template
@@-- NAME : User's name
<HTML>
<P>Hello @_NAME_@</P>
</HTML>
```

Resulting HTML:

```
<HTML>
<P>Hello Bill</P>
</HTML>
```

```
procedure Simple is
  Translations : Translate_Table
    := (1 => Assoc ("NAME", "Bill"));
begin
  Put_Line (Parse ("simple.tmplt",
                   Translations));
end Simple;
```

Templates Commands

- Comments

```
@@-- Any text
```

- Conditions

```
@@IF@@ <expression>
  …
@@ELSIF@@ <expression>
  …
@@ELSE@@
  …
@@END_IF@@
```

- Table
- Include

Some advanced services (1)

- Transient pages

  Pages built on-the-fly, automatically deallocated

- Split pages

  Logical pages automatically split over several real pages, with automatic index generation

- Sessions

  Store/retrieve per-user data

- Streams

  Build pages in memory

Some advanced services (2)

- File upload
- Server Push
- Status page
- Authentication
  - Control access based on user name / password
  - Supports Basic and Digest authentication
- Logging
  - History of what's happenning
  - Same file format as Apache

Some advanced services (3)

- Mailing
  - As client (SMTP)
  - As server (POP)
  - A simple Webmail server is provided as an AWS callback
- Miscellaneous Services
  - Directory browser, URL, Translator (Base64, Zlib), Exceptions, web blocks, web elements, AJAX

Provided Dispatchers (1)

- URI dispatcher

  Dispatches to other functions according to the URI

- Page dispatcher

  Considers the URI as a file name and returns the corresponding file. Parses 404.thtml if not found.

- Method dispatcher

  Dispatches to other functions according to the HTTP method.

  Use: ???

- Virtual host dispatcher

  Dispatches to other functions according to the host name

Provided Dispatchers (2)

- Time dispatcher

  Associates various functions to different periods of time, and dispatches according to the time of the request.

- Transient pages dispatcher

  Linked to another dispatcher

  If the other dispatcher replies "404", tries to interpret the URI as a transient page.

- SOAP dispatcher

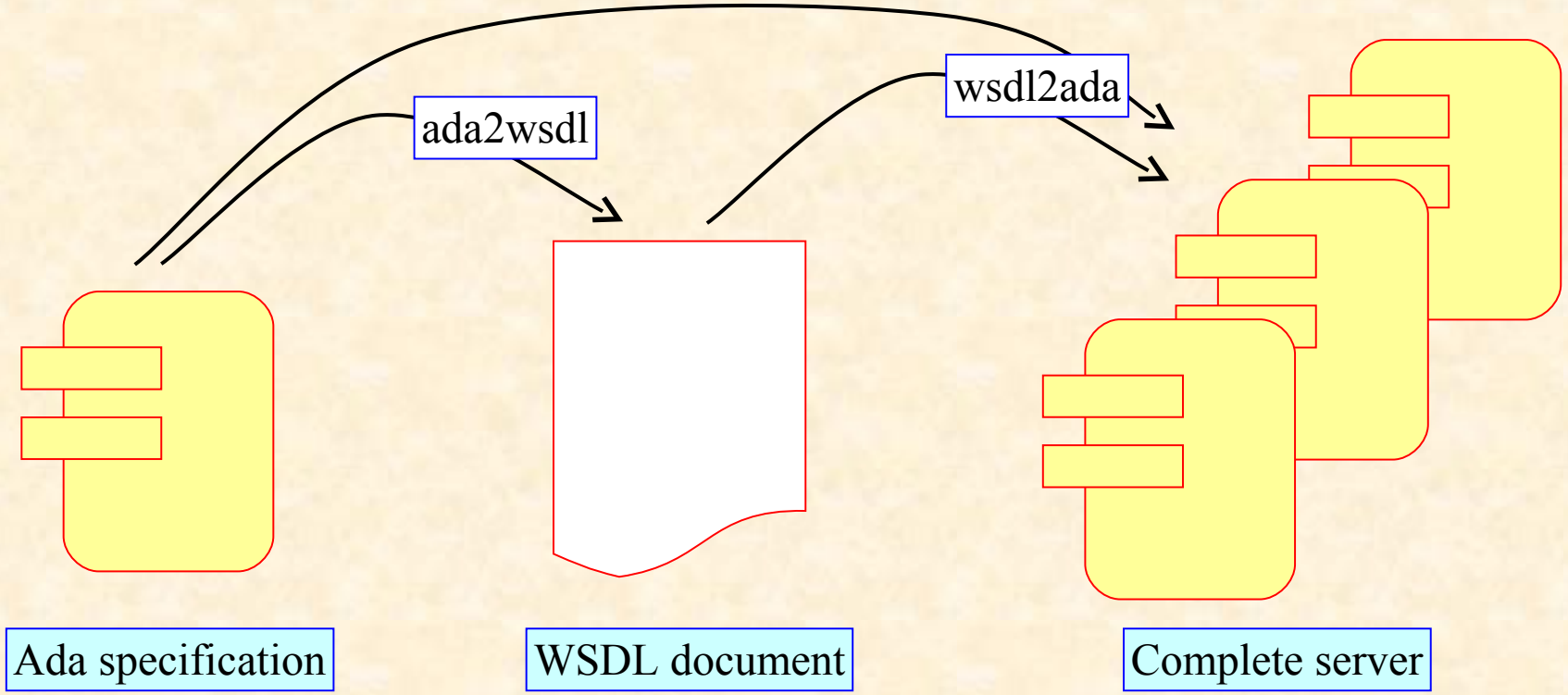
  Provides two call-backs, one for HTTP requests, one for SOAP requests.

AWS for Distributed Computing

- Exchanging simple data:
  - Simple communication
  - HTTP client
- Distributed server:
  - Hotplugs
- Remote services:
  - SOAP
  - LDAP
  - JABBER
- And you can still use Annex E in addition…

Writing a SOAP/WSDL server

- aws2wsdl and wsdl2aws work together!



AWS vs. Other Technologies (1)

- The application is a single executable, not a set of scripts
  - Must recompile when functionalities are added/changed
  - NOT when presentation changes (thanks to templates)
- Separate processing from display
  - Unlike servlets
- Easy to deal with concurrent access
  - Thanks to protected types!
- What's difficult with Apache made easy
  - HTTPS, logs, …

AWS vs. Other Technologies (2)

- Efficiency

  No need to start a process for each request

- Ease of distribution

  Simplified deployment (no Web server to install and configure, a single executable to install).

- Mixed applications

  When the Web interface is only part of the application

  Possibility of having a control panel

AWS Usage

- Commercial support provided by AdaCore
- Users

  EDF/R&D (WORM (shared bookmark), Internet share)

  Vision2Pixels (Photo-club, http://v2p.fr.eu.org)

  Adalog (Gesem)

  SETI@Home module (T. Dennison – 1 to 3 millions users)

  ACT (Gnat tracker)

  Ada-Russia (http://www.ada-ru.org)

  Frontend to access Oracle via a Web interface.

  Philips (DOCWEBSERVER and OESM)

  Currency change (D. Anisimkov,40 to 50 requests/s.)

- Statistics

  > 300 users (and growing), a mailing-list with 120 people.

Conclusion

- A mature technology
- AWS is more than a Web server

  Full HTTP API
  - Communication (client/server).
  - Sessions
  - PUSH

  Other protocols:
  - SOAP
  - SMTP / POP / LDAP / Jabber

  More than a simple server
  - Several servers, hotplugs
  - Virtual hosts
  - distributed computing


Questions