# Web-enabling Ada Applications with AWS

J-P. Rosen
Adalog
An AXLOG Group company

rosen@adalog.fr

# AWS

Many thanks for the slides!

- 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 (CVS): :pserver:anoncvs@libre.act-europe.fr:/anoncvs

# What is AWS?

82 (user) packages !

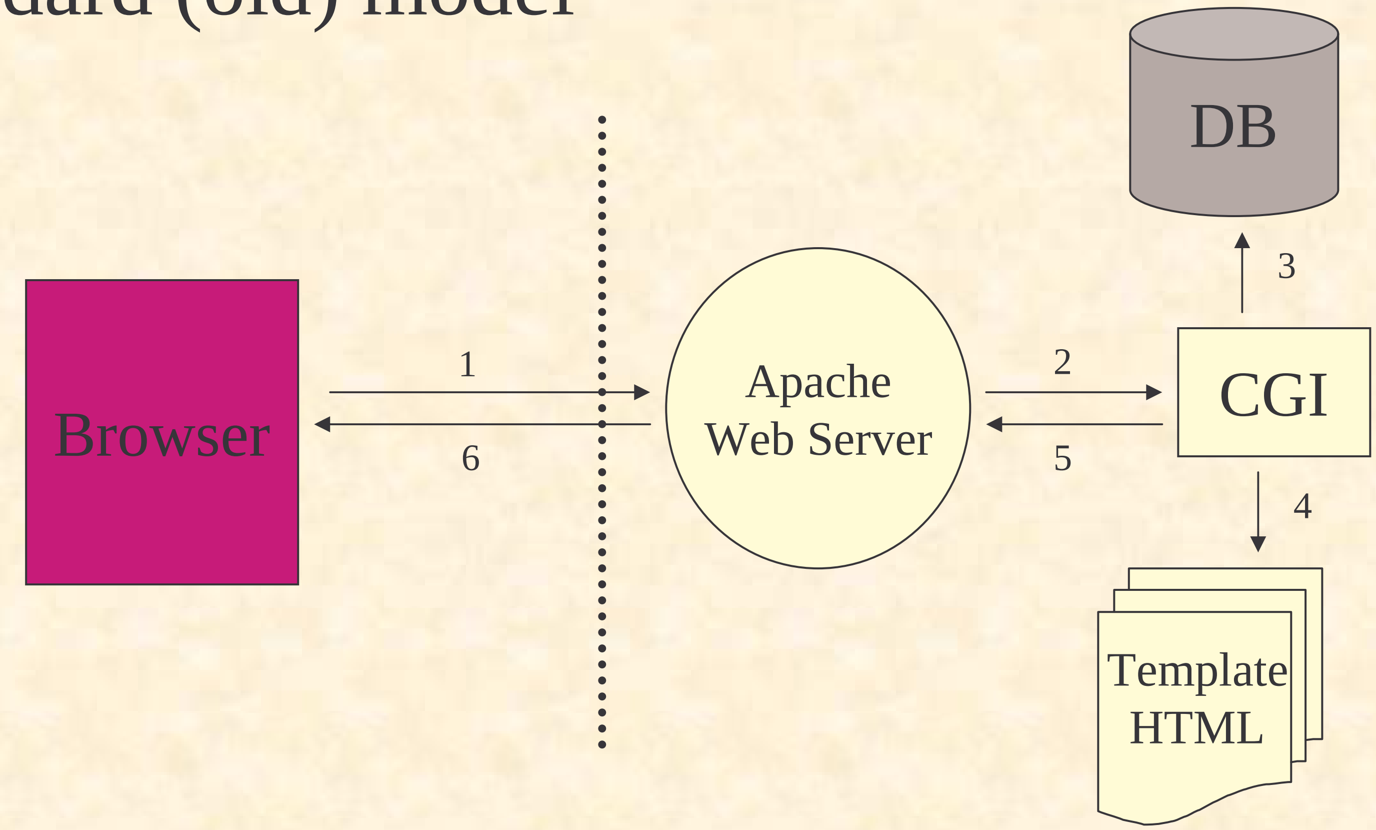
- A set of packages for managing protocols
  - ☞http/https, SOAP, LDAP, Jabber, SMTP, POP…
  - ☞Server side
  - ☞Client side
- Facilities for managing pages (dispatchers)
- Facilities for building pages (templates parser)
- Facilities for making distributed applications
- Other facilities (Resources,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

# What Can AWS Be Used For?

- HTTP services
  - Lightweight page server
    - A full web server is another story…
  - 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

# Web Development

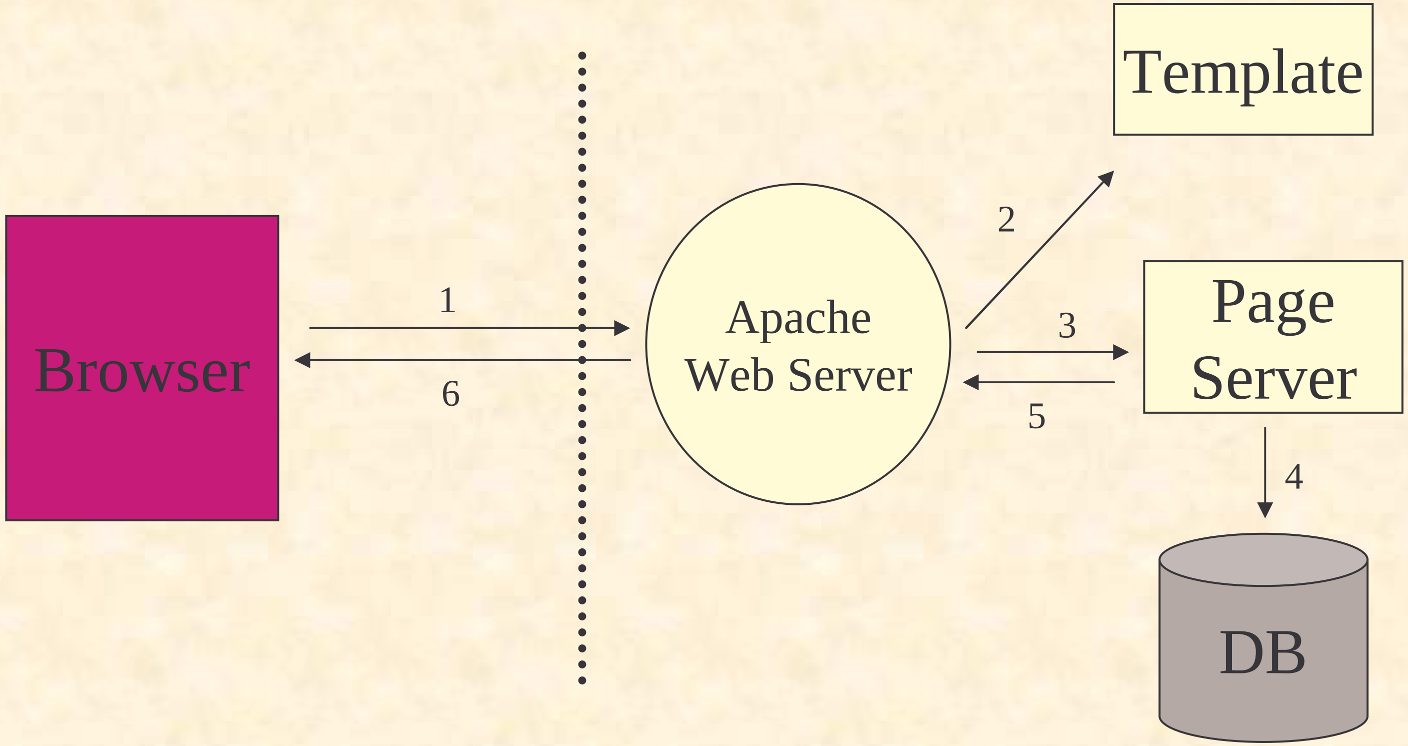
- Standard (old) model



**The program is separated from the server**

# Web Development

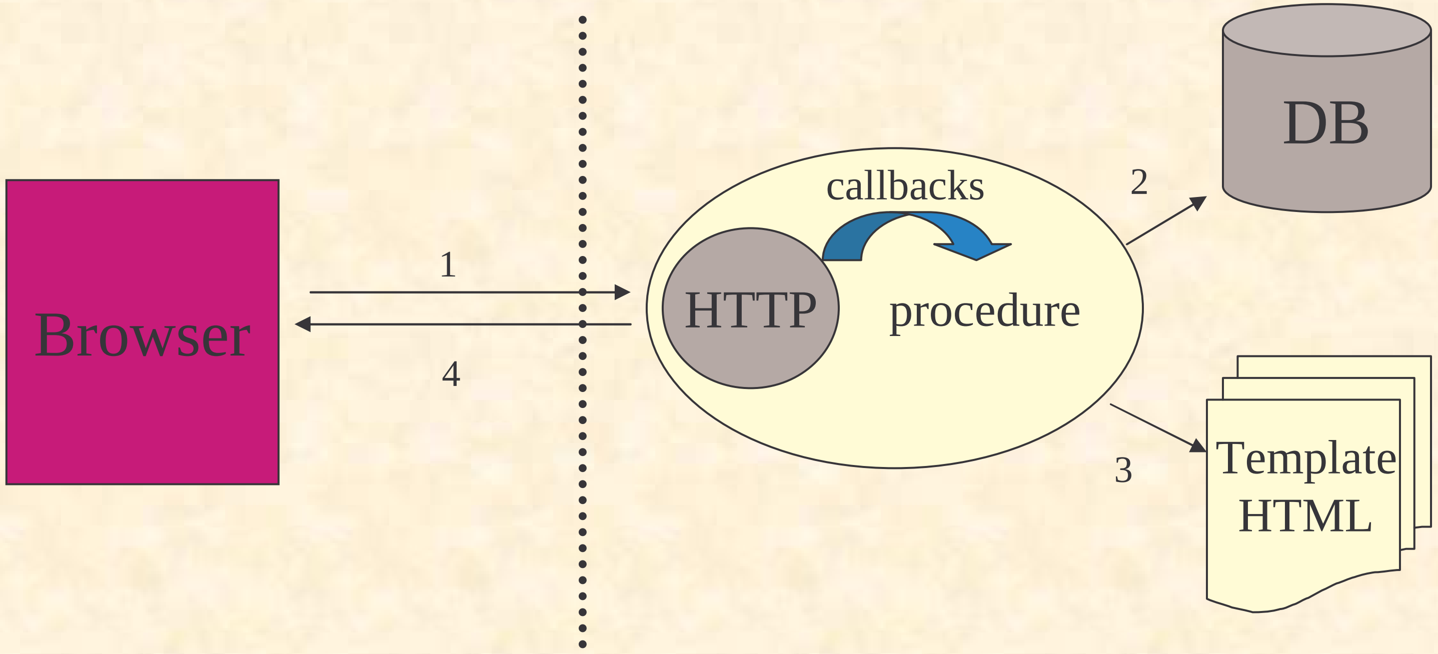
- Scripting model (Server side inserts)



**The program is inside the server**

# Web Development

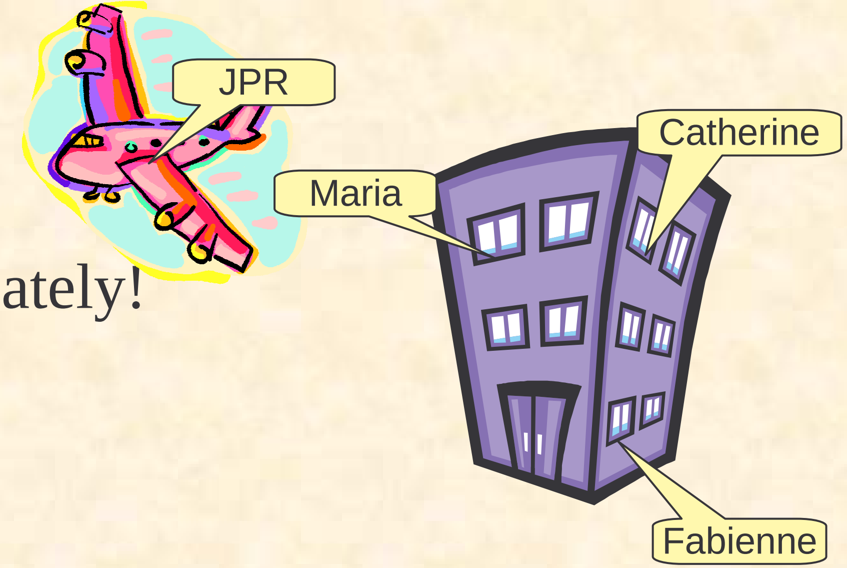
- AWS based model



**The server is inside the program**

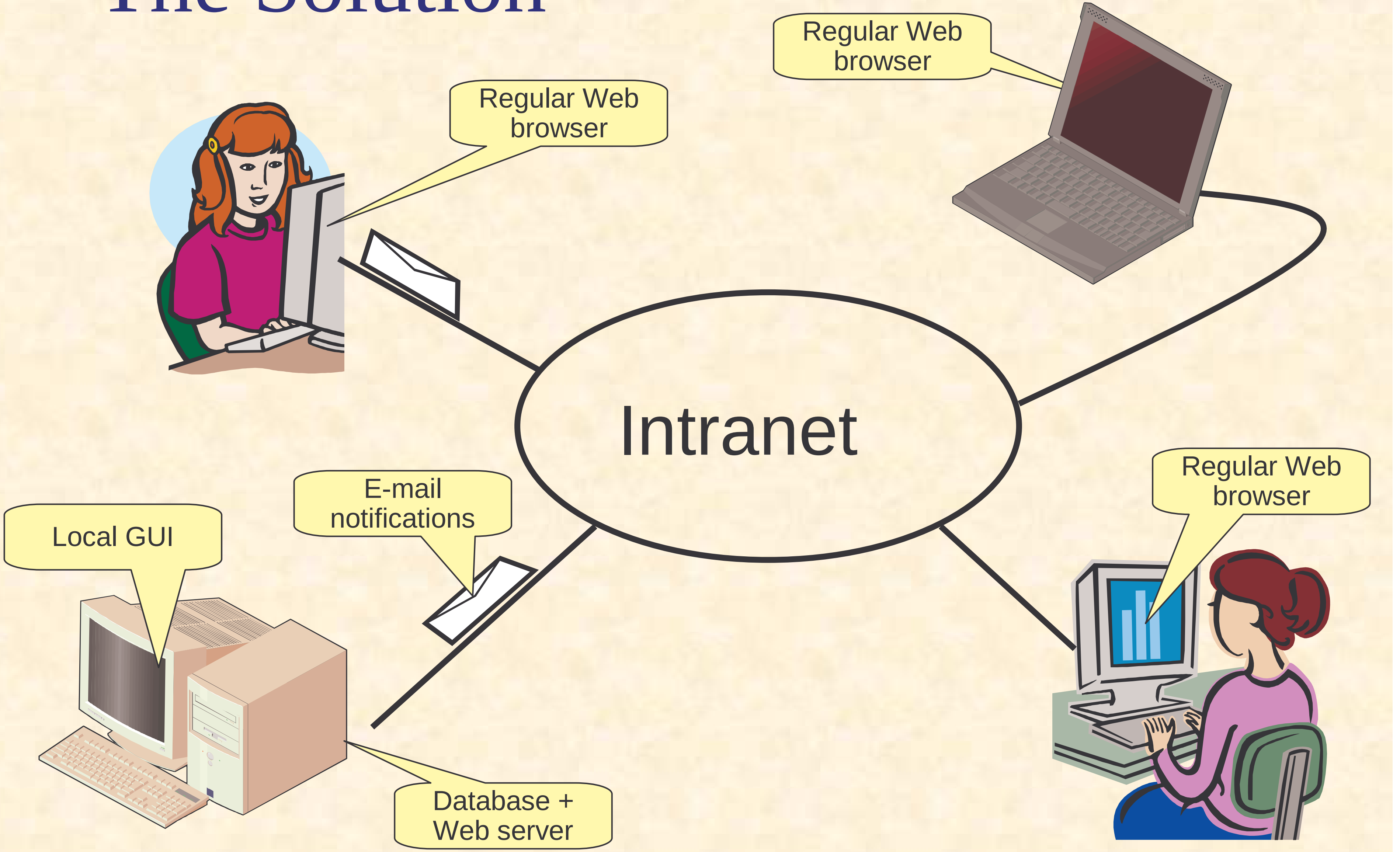
# Example: Adalog's Gesem

- Managing the registration to training sessions
  - Several persons in charge
  - In various locations,
    not available at the same times
  - Must answer the phone immediately!



- Pinging people
  - Prepare hand-outs
  - Reserve restaurant
  - …
- Managing mailing
  - Classical database extraction

# The Solution



# Basic Behaviour

- AWS :
  - ☞opens the HTTP(S) message
  - ☞Gets answer using the user’s callback procedure
  - ☞Encapsulates answer and sends it back to browser

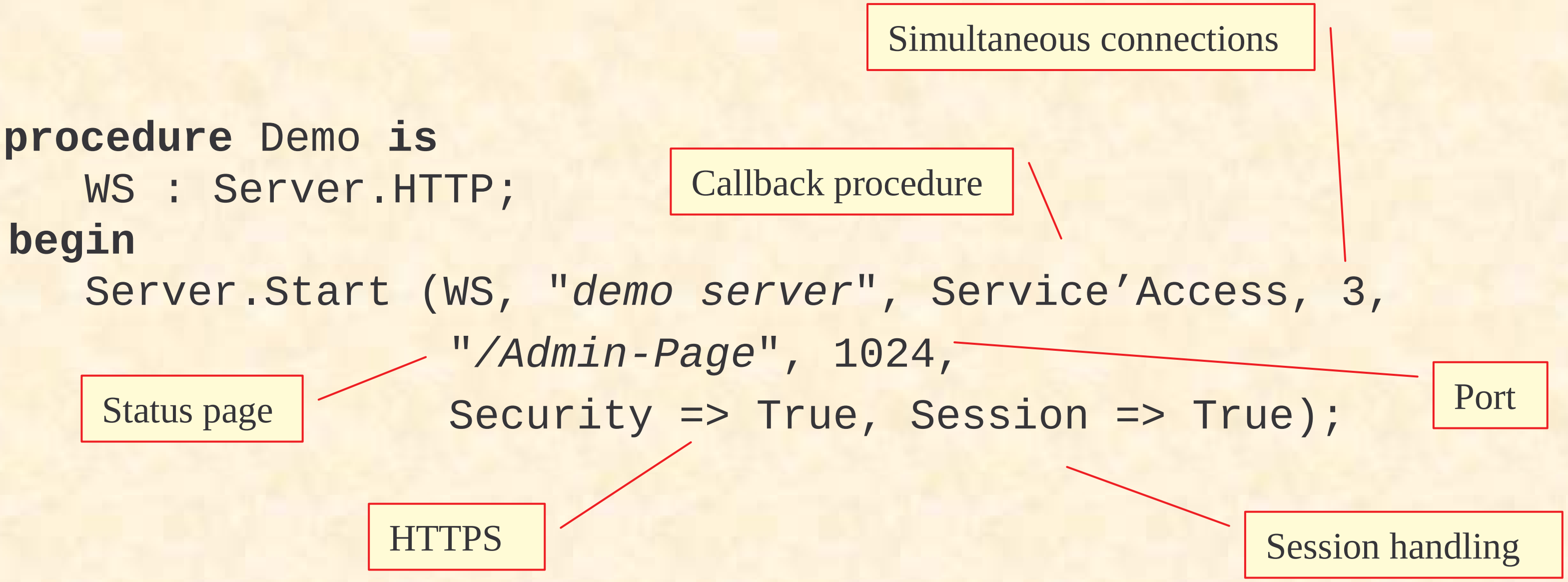
```
procedure Start(Web_Server : in out HTTP;
                Callback   : in     Response.Callback;
                Config     : in     AWS.Config.Object);

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

**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

Port

Status page

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 ...
```

**The callback procedure must be thread-safe.**

# 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 (1)

- 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);
…
```

# Object Oriented AWS (2)

- Benefit: the dispatcher can be extended
  - For example, a function to register a call-back (or another dispatcher) for pages matching a given pattern
  - An ordered set of rules with the corresponding action.
  - Helps manage the complexity of large projects.
- Provided: AWS.Dispatchers.Callback
  - A simple wrapper around the regular callback procedure
  - Adds:

```
function Create (Callback : in Response.Callback)
   return Handler;
```

- More dispatchers later…

# 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;
```

Because the call-back is a local function

# 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.**

# Tags

- A tag is a named variable
  - ☞ appears in template as `@_NAME_@`
- A translation table is an array of associations
  - ☞ Name => Value
- Associations have constructors for:
  - ☞ Scalar
    - String, Unbounded_String, Integer, Boolean (True, False)
  - ☞ Vector
    - One-dimensional array
  - ☞ Matrix
    - Two-dimensional array (actually, a vector of vector-tags)

# Setting Tags

```
procedure Tags is
  use type Vector_Tag;
  use type Matrix_Tag;

  B : constant Boolean    := True;
  V : constant Vector_Tag := +"10" & "30" & "5";
  M : constant Matrix_Tag := +V & V;
  S : constant String     := "a value";

  Translations : constant Translate_Table
     := (1 => Assoc ("TEST", B),
         2 => Assoc ("VECT", V),
         3 => Assoc ("MAT",  M),
         4 => Assoc ("VAL",  S));
```

# 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;
```

# Tag Modifiers

@_{FILTER:}Tag['ATTRIBUTE]_@

- Filters:
  - @_UPPER:VAR_@
  - @_ADD(3):VAR_@
  - @_EXIST:VAR_@
  - @_MATCH("Adalog.*"):VAR_@
  - @_FORMAT_DATE("%H-%M-%S"):NOW_@
  - @_YES_NO:VAR_@
  - @_WEB_ESCAPE:WEB_NBSP:CAPITALIZE:TRIM:VAR_@

And many more

- Attributes:
  - @_VECT'LENGTH_@
  - @_MAT'LINE_@
  - @_MAT'MIN_COLUMN_@
  - @_MAT'MAX_COLUMN_@

# 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

# 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.

# Configuration

- Many things can be configured…
  - Admin_URI: the status page name
  - Log_File_Directory: where to store log files
  - Max_Connection: number of simultaneous connections
  - Server_Port: the port to connect to
  - And many more…
- Configuration is initialized from:
  - Parameters of the Start procedure
  - aws.ini: for all applications started from the same directory
  - <progname>.ini: for application <progname>
  - A configuration object can be initialized from a file

# Deploying an AWS Server

- Resources
  - ☞It is possible to include any file (HTML, Images, icons, templates…) used by the Web server into the server executable.
  - ☞Resources are compiled with awsres.
    - Creates a hierarchy of packages, one for each resource
    - Resources can be compressed
    - Just "with" the root package
- No Web server is easier to distribute, install and launch !

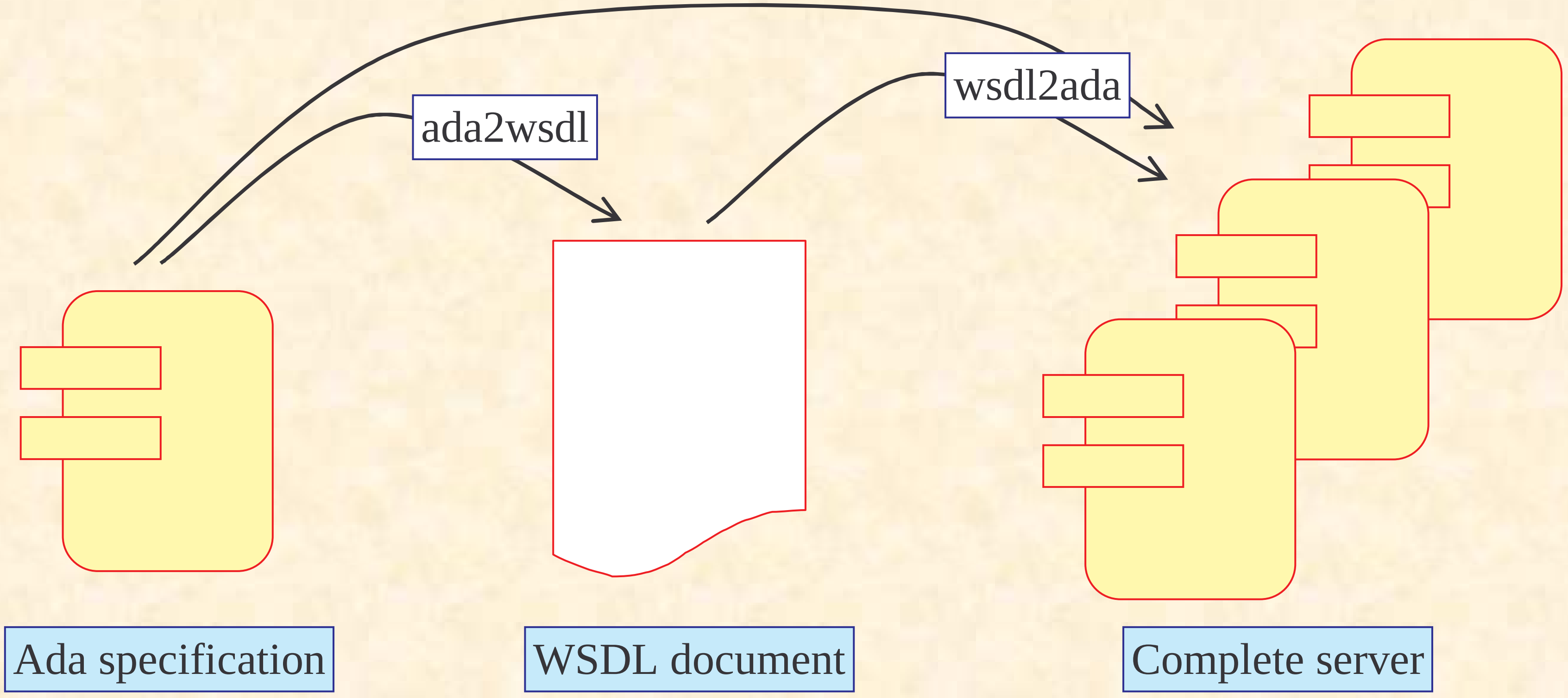
**A single, self contained Web server executable**

# 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

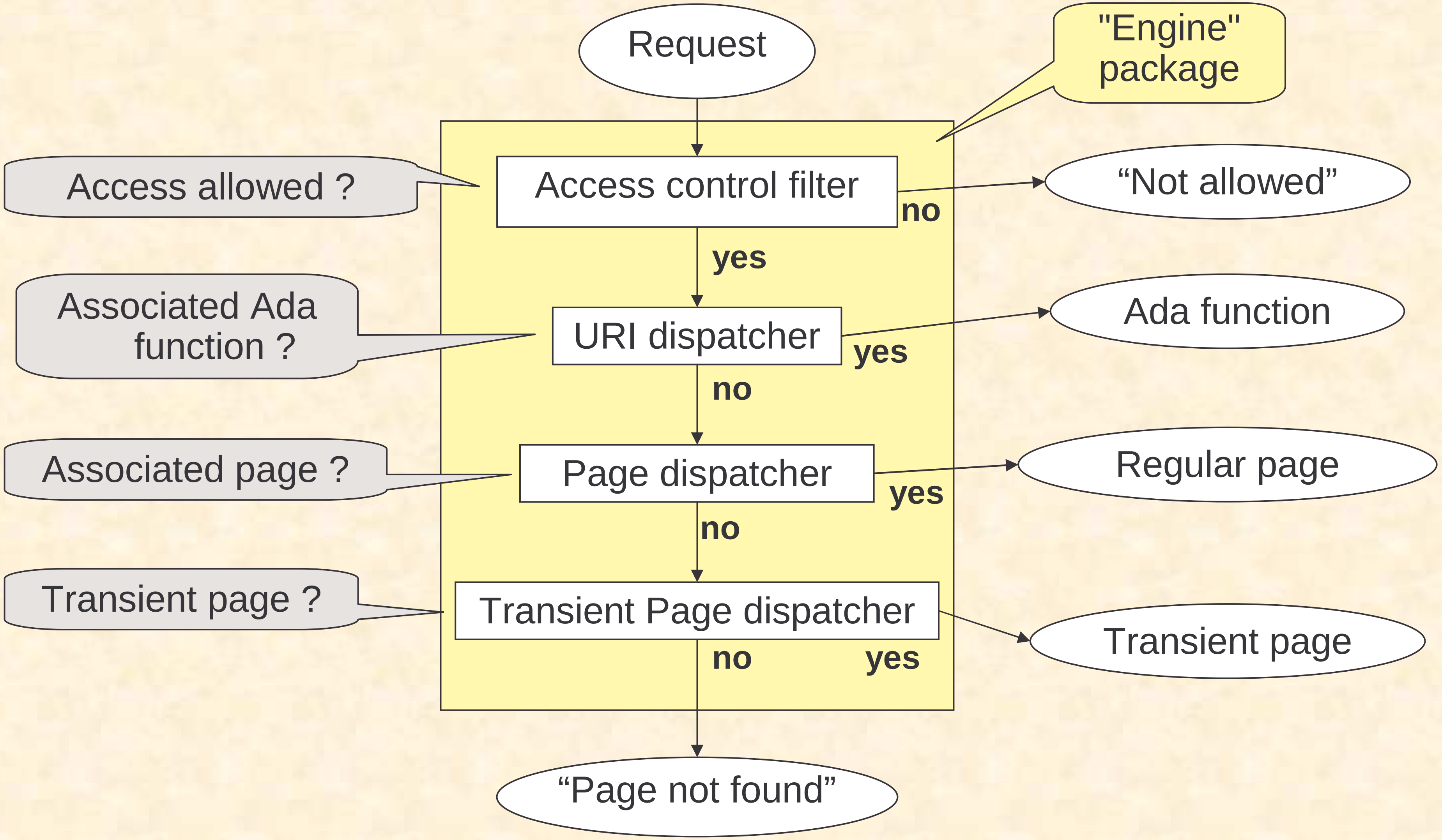
- aws2wsdl and wsdl2aws work together!



# Gesem's Implementation

- Unusual constraints:
  - Use free software
  - User interface usable by casual users
  - Availability on Windows and Linux
  - Independent of any particular DBMS
  - Easily modifiable
  - Deal with concurrent accesses
  - Efficiency *is not* a concern
  - Reliability *is* a concern

# Gesem Filters and Dispatchers



# The Page Design Pattern

```
with AWS.Response;
package Pages.Some_Page is
  function URI (<parameters>)return String;
end Pages.Some_Page;


package body Pages.Some_Page is
  My_Name : constant String := "some_page";

  function Build (<Parameters>)
    return Response.Data is ...

  function Buttons (Request : in AWS.Status.Data)
    return Response.Data is ...

  function Page (Request : in AWS.Status.Data)
    return Response.Data is ...

  function URI (<parameters>)return String is ...
begin
  Engine.Register(My_Name, (Root    => Page'Access,
                            Buttons => Buttons'Access));
end Pages.Some_Page;
```

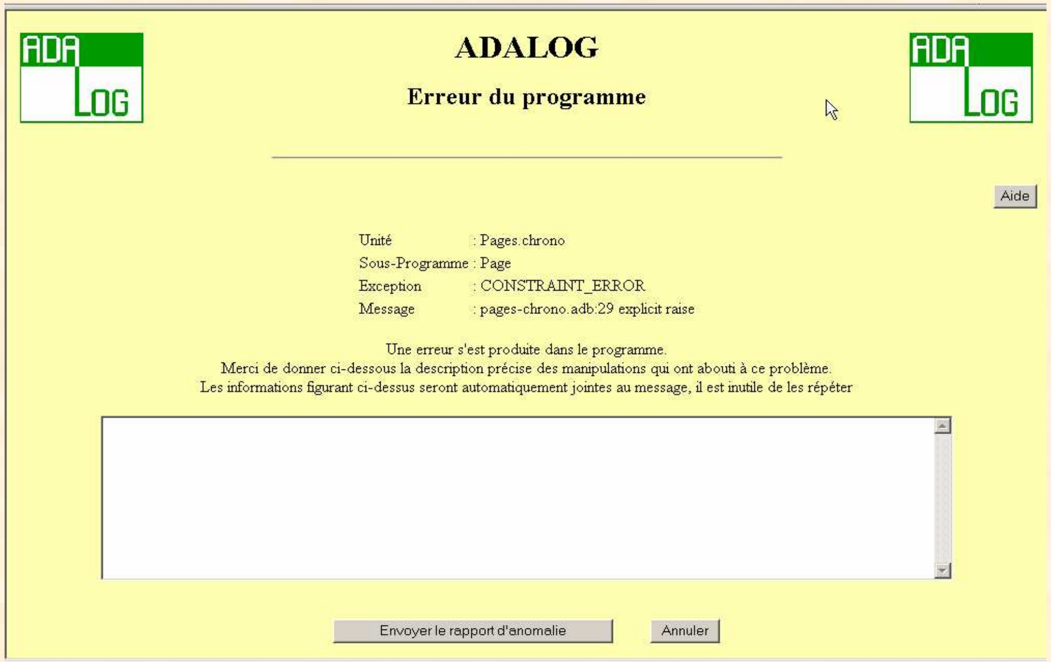
some_page.html

some_page.btns

# Reliability

- Every page has an exception handler:

```
exception
   when Occur : others =>
      return URL (Pages.Error.Build
                      (Unit       => "pages." & My_Name,
                       Subprogram => "Name of subprogram",
                       Occur      => Occur));
```



**ADALOG**

**Erreur du programme**

```
Unité           : Pages.chrono
Sous-Programme  : Page
Exception       : CONSTRAINT_ERROR
Message         : pages-chrono.adb:29 explicit raise
```

Une erreur s'est produite dans le programme.
Merci de donner ci-dessous la description précise des manipulations qui ont abouti à ce problème.
Les informations figurant ci-dessus seront automatiquement jointes au message, il est inutile de les répéter

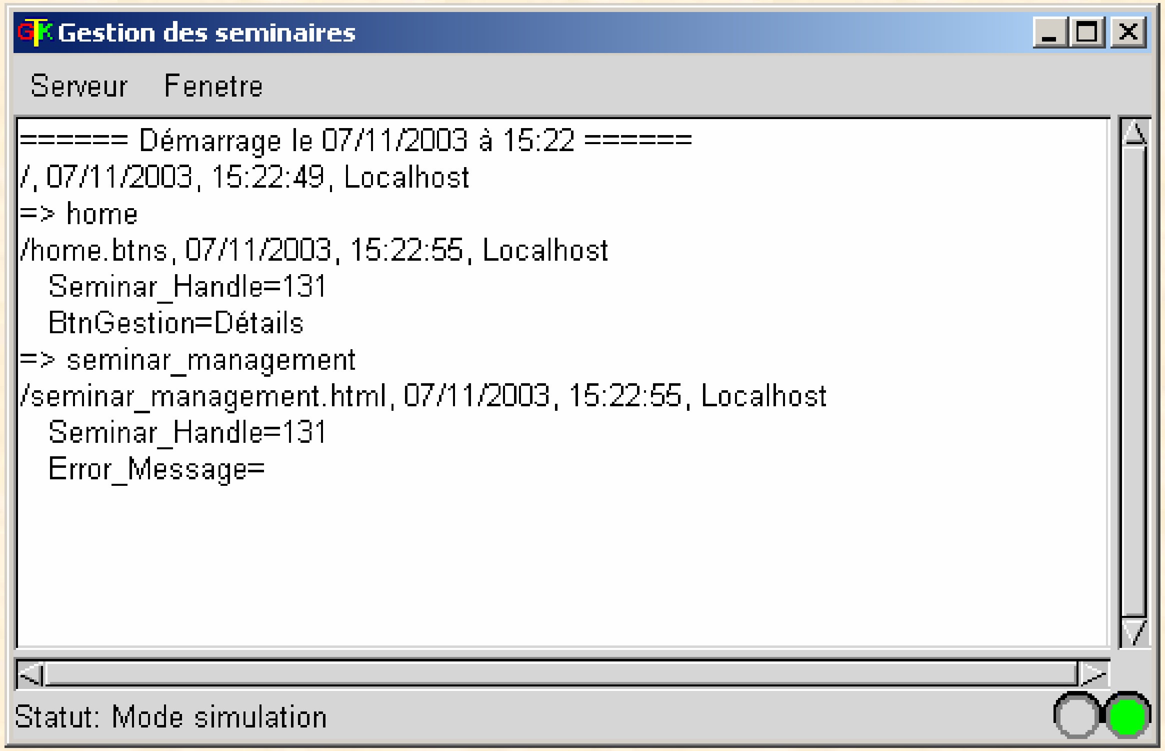
# Concurrency

- Concurrent access is extremely unlikely, but possible
  - ☞ Recognize users from their IP address
  - ☞ Use a global lock:
    - Only one user can modify at any one time
    - "Modify" button on each page to grab the lock
- But beware of "back" button
  - ☞ Display a page
  - ☞ Modify it (get lock)
  - ☞ Validate (release lock)
  - ☞ Back page: the page is modifiable, but the user doesn't own the lock !
  - ☞ Checked by the access control filter => page expired

# Local Interface

- Manages the application
  - ☞ Stop, lock database…
  - ☞ Shows uncommitted transactions
- Monitors requests
  - ☞ Clear window
  - ☞ Save content to file

- Plain GTK
- Generated automatically with GLADE



```
====== Démarrage le 07/11/2003 à 15:22 ======
/, 07/11/2003, 15:22:49, Localhost
=> home
/home.btns, 07/11/2003, 15:22:55, Localhost
   Seminar_Handle=131
   BtnGestion=Détails
=> seminar_management
/seminar_management.html, 07/11/2003, 15:22:55, Localhost
   Seminar_Handle=131
   Error_Message=
```

Statut: Mode simulation

# Objects Design Pattern

```
with Globals, Data_Manager, AWS.Templates;
use Globals;
package Objects.Abstraction is
   type Data is
         record
            …
         end record;
      -- Operations on Abstraction.Data

   function Image (Item : Data)return Array_Of_Unbounded;
   function Value (Item : Array_Of_Unbounded)return Data;
   package Manager is new Data_Manager
     (Data      => Data,
      Data_Name => "my_data",
      Columns   => "col1, col2, col3");
   subtype Handle is Manager.Handle;
   type List is array (Positive range <>)of Handle;

   function Associations (Item : Handle) return Translate_Table;
   function Associations (Item : List)   return Translate_Table;
   function Extract (Param : AWS.Parameters.List) return Data;
end Objects.Abstraction;
```

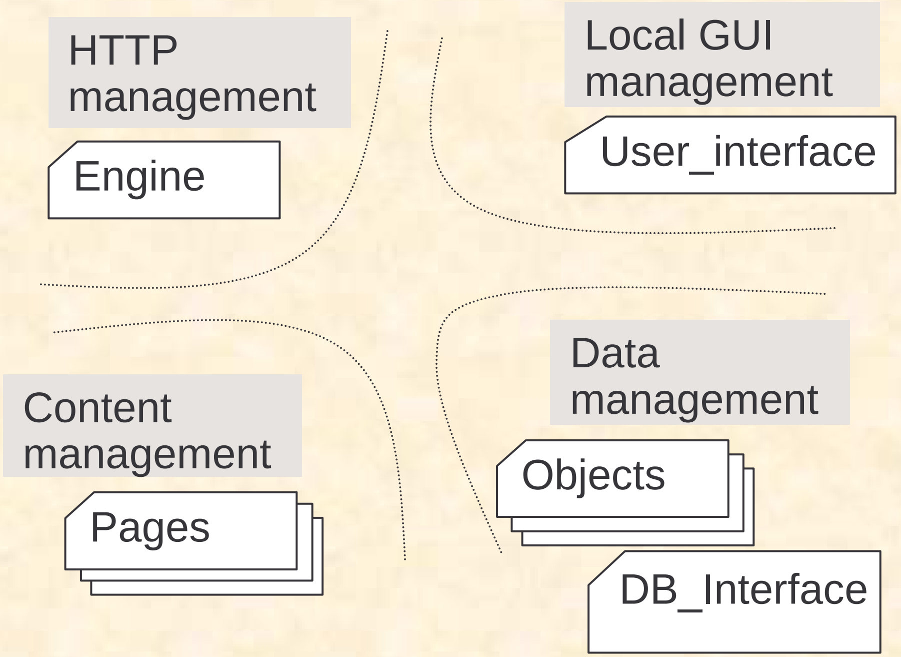
Ada view

Database view

Templates (HTML) view

# Lessons learned (1)

- Separate concerns



HTTP management

Engine

Local GUI management

User_interface

Content management

Pages

Data management

Objects

DB_Interface

- Reliability
  - ☞Exceptions are great!
- AWS is powerful enough
  - ☞No Javascript, no Java
  - ☞The template parser is great!

# Lessons learned (2)

- A web interface is difficult to manage
  - ☞User can close the browser at any time (even with uncommitted transactions), but the application is not aware!
  - ☞User can call "previous page" at any time: no global state
- Portability
  - ☞ > 10_000 SLOC in 81 compilation units
  - ☞Network interface + GUI + Database interface
  - ☞**No** difference between Linux and Windows version
  - ☞Ada is great!

# 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

- Users
  - ☞EDF/R&D (WORM (shared bookmark), Internet share)
  - ☞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, a mailing-list with 87 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

A complete Web development framework

# Questions