Web applications

UniPI

Antonio Cisternino

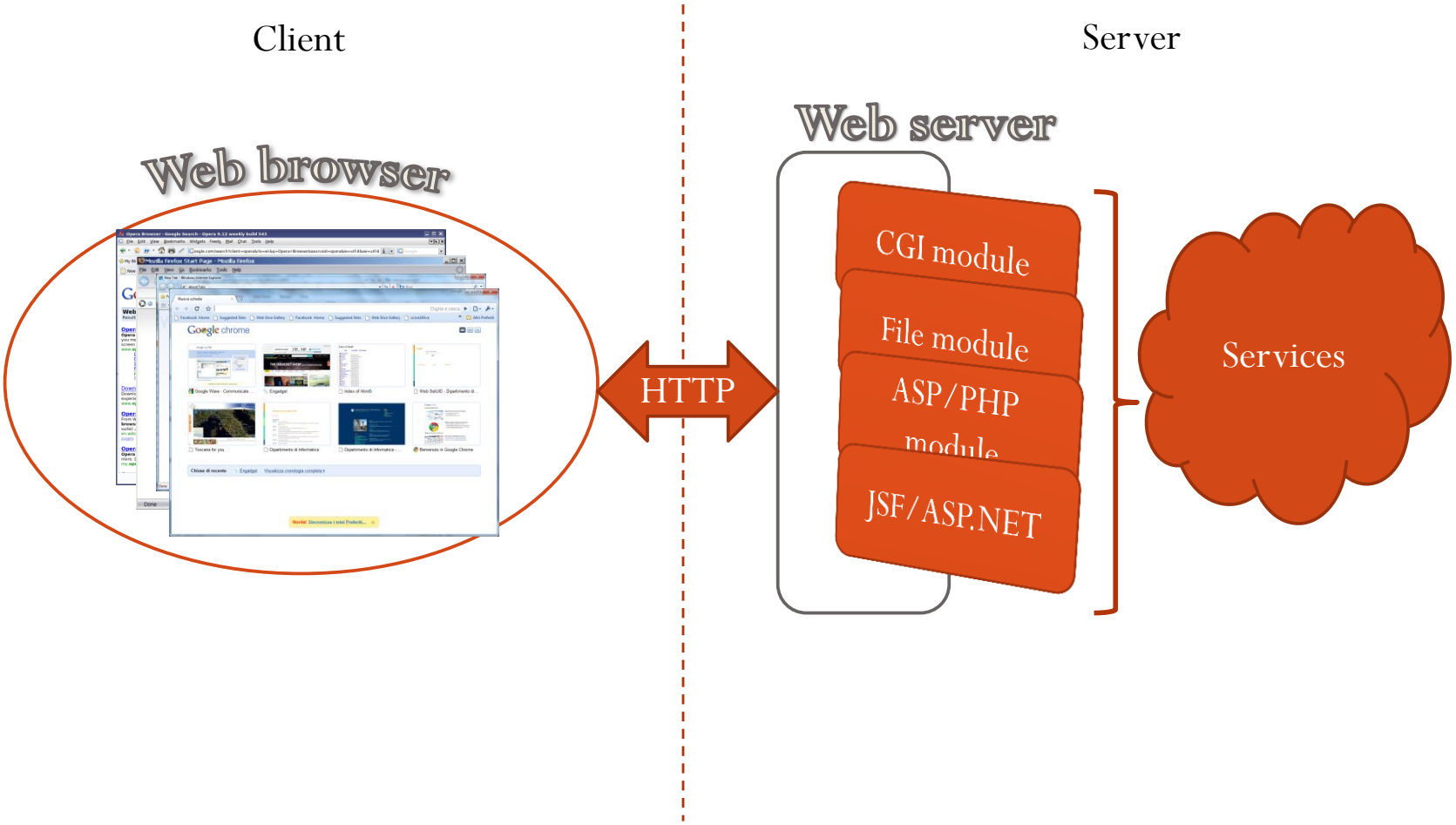
Motivations

- Define a formal model to define the notion of Web Application encompassing all different implementations available nowadys
- Definition of a refinement-based taxonomy of available frameworks for comparison (i.e. ASP vs. PHP, ASP.NET vs. JSF)
- Study Web application general properties, i.e.: state management (sessions get lost, client and server state go out-of-sync) and consistency

Goals

- Give an abstract definition of what Web applications are capable of:
  - Include the whole class of applications
  - Associate similar approaches (i.e. PHP and ASP)
  - Discriminate different models (i.e. PHP/ASP vs. JSF/ASP.NET)
- Prove properties about Web apps:
  - State coherence
  - Event flowing
  - State persistance

Key Elements



Web application essentials

- a **transmission protocol** which is universal (can carry any data), stateless (meaning an individual transaction is composed of just a request and a response, after which no state is preserved), and in which only the client can initiate exchanges;
- a **browsing context** on the client side, a renderer and a DOM, plus network capabilities, whereas the presence of an interpreter is optional;
- a **mapping function** that can uniquely associate a request/response pair to an individual browsing context; such function must be available to the server;
- a **server** program handling the incoming requests.

Ongoing activities

- Models:
  - Web browser
  - Web server
  - ECMA script 262 interpreter (reworking after spec update)
- ASM extensions: ambient

Server side

```
WEBSERVER =
  let request = pick from requestsQueue
  let env = SERVERENVIRONMENT(request)
  let module = SELECTMODULE(request)
  if module ≠ undef then
    module(request, env, VERB(request))
  else SENDERROR(request, 500)
```

CGI-Module

```
CGIMODULE(request, env, verb) =
 let resourceName = RESOURCENAME(request), queryString = QUERYSTRING(request)
 let process =
  EXECPROCESS(MAKEENV(env, queryString), MAKEPATH(ROOT(env), resourceName))
 if process = undef then SENDERROR(request, 404)
 else
  SENDRESPONSE(request, 200) step
  let stdin = INPUTSTREAM(process), stdout = OUTPUTSTREAM(process)
  let reqOut = REQUESTOUTPUTS(request), reqIn = REQUESTINPUTS(request)
  if verb = 'POST' then
   while DATAAVAILABLE(reqIn) do
    data := READDATA(reqIn) step WRITEDATA(stdin, data)
  while DATAAVAILABLE(stdout) do
   data := READDATA(stdout) step WRITEDATA(reqOut, data)
 CLOSE(request)
```

Scripting Module

```
ScriptingModule(request, verb, env) =
 let resourceName = ResourceName(request),  queryString = QueryString(request)
 let script = MakePath(Root(env), resourceName)
 if FileNotExists(script) then sendError(request, 404)
 else
  amb ScriptingModule in
   let id = SessionID(ScriptingModule, request)
   if Sessions(id) = undef then
    Sessions(id) := MakeSession(ScriptingModule, request, env, id)
   let application = ApplicationName(resourceName)
   if  Applications(application) = undef then
    Applications(application) := MakeApplication(ScriptingModule, request, env, application)
   step
   amb request in
    script := transformScript(script)
    Request := MakeRequestHostObject(request, verb)
    Response := MakeResponseHostObject(response)
    Session := MakeSessionHostObject(parent(curamb).Sessions(id))
    Application := MakeApplicationHostObject(parent(curamb).Applications(application)) step
    ScriptInterpreter([ Request; Response; Session; Application ], script) step
    Close(request)
```

Syntax conversion from quotation to full script:

<% S %> -> S

<%= S %> -> Response.Write(S)

Line -> Response.Write('Line')

Scripting module

- Can be further refined into PHP or ASP
- It can also be refined to a JSP module in which operations are performed by the JVM and no syntax translation is required (in this case script is in fact a class file)
- ASP can be further refined in different flavors using different scripting engines (with the ActiveScript technology)
- State management across different invocations is performed through the session notion and the ability to find a unique id to associate to a request
- Ajax requests require no special handling

Object module

- This module attempts to capture the components tree rendering approach of ASP.NET and JSF frameworks
- Before invoking user code the module builds a tree of components, restore their state and perform dispatch of events originated on the client
- In this case Ajax requests must be handled properly
- State management is done not only using sessions, but also encoding information in the HTML sent to the client

State consistency

- Web application state is shared between server and client
- Ensure state consistency is crucial for Web applications (think for instance to credit card transactions)
- As frameworks evolve state gets managed more and more automatically featuring more abstract views of what the application is
- When a Web page aggregates HTML from different Web applications is also important to be able to reason about data isolation
- State reconstruction is crucial when the client can change and its state must be persisted to another (i.e. From desktop to mobile)