Programare web

Arhitectura

Exemplu suport

Standardul CGI

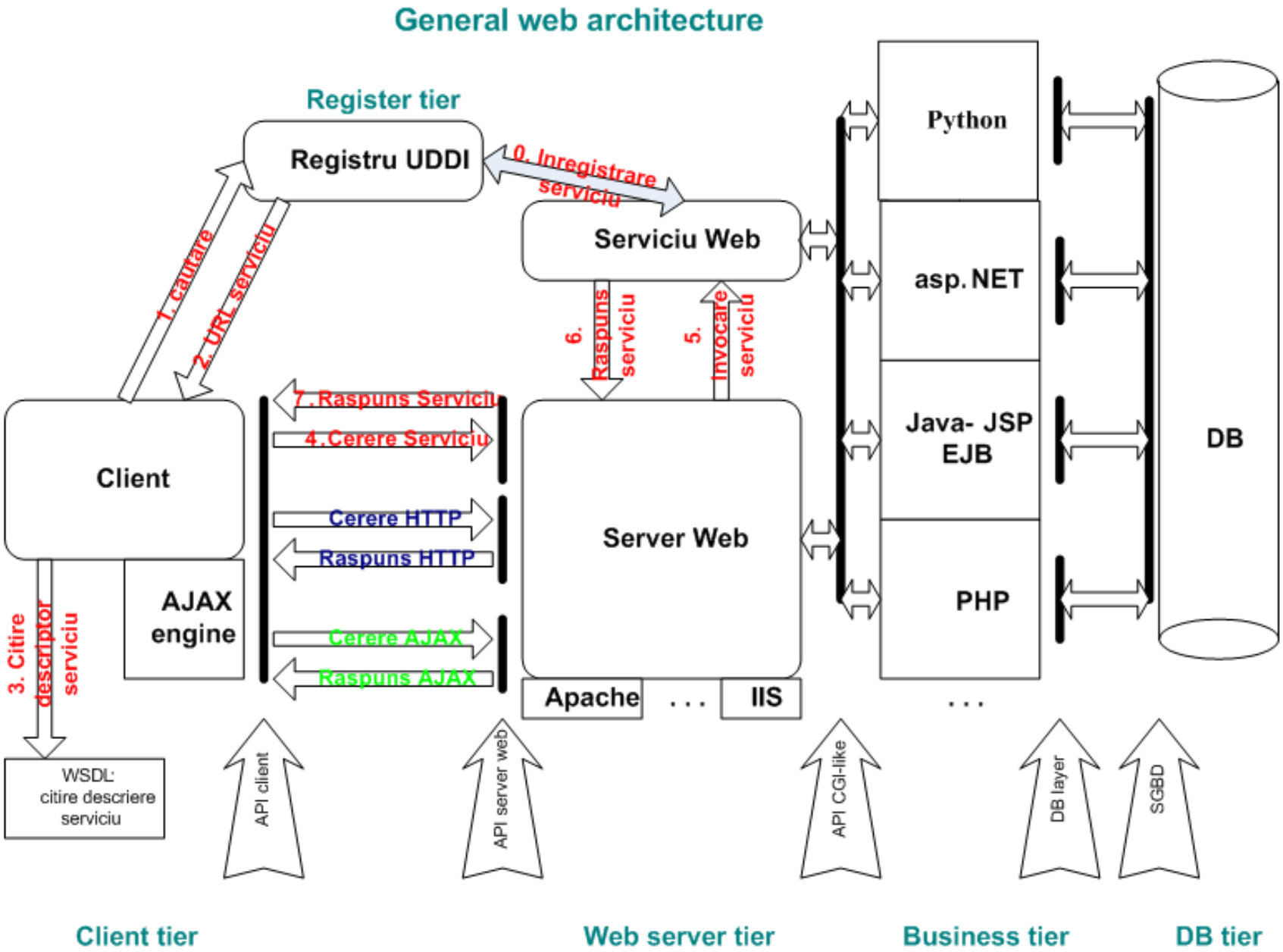
Exemple CGI + servlet + Node.js

Configurare Apache pentru CGI

Exemplu de implementare server web ce lucreaza cu CGI: MiniHttpd si clienti ai lui

Scripting Server Pages: ASP,NET, JSP, PHP, PSP

Clienti standalone, AJAX


General web architecture
Register tier
Registru UDDI
0. Inregistrare serviciu
1. cautare
2. URL serviciu
Serviciu Web
6. Raspuns serviciu
5. Invocare serviciu
Python
asp. NET
Java- JSP EJB
PHP
DB
Client
AJAX engine
7. Raspuns Serviciu
4. Cerere Serviciu
Cerere HTTP
Raspuns HTTP
Cerere AJAX
Raspuns AJAX
Server Web
Apache
. . .
IIS
. . .
3. Citire descriptor serviciu
WSDL: citire descriere serviciu
API client
API server web
API CGI-like
DB layer
SGBD
Client tier
Web server tier
Business tier
DB tier

CGI este un program executabil (*elaborat in orice limbaj*) care are urmatoarele caracteristici:

- Sa aiba setat **dreptul de run** asa incat sa poata fi lansat in executie de catre programul server web.
- Sa poata citi **variabilele de mediu** folosind apeluri sistem de tip **getenv**.
- Daca metoda HTTP **nu este GET**, atunci trebuie sa poata citi fisierul standard **stdin**.
- Trebuie sa poata scrie in fisierul standard **stdout**
- Fisierul executabil CGI trebuie sa fie plasat intr-un anumit director stabilit de catre administratorul serverului web.
- Uneori (administratorul poate impune ca) numele fisierului executabil sa respecte o anumita sintaxa (de exemplu terminarea cu **.cgi** sau alte restrictii).

Daca cererea HTTP este GET, atunci CGI obtine stringul de cerere ca valoare a variabilei de mediu **QUERY_STRING**
Daca cererea HTTP este POST, PUT, DELETE etc. atunci stringul de cerere se obtine de la fisierul de intrare **stdin**

Cercetările şi experimentele efectuate, ca şi concluziile trase în urma exploatării timp îndelungat a unor aplicaţii web, au impus CGI ca şi standard de facto pentru nivelul business de langa serverele web. Pe baza CGI s-au elaborat şi se elaborează în mod direct aplicaţii web. De asemenea, pe baza CGI se proiectează şi implementează *instrumente* şi *frameworkuri* specifice. Acestea uşurează substanţial şi simplifică elaborarea aplicaţiilor web.

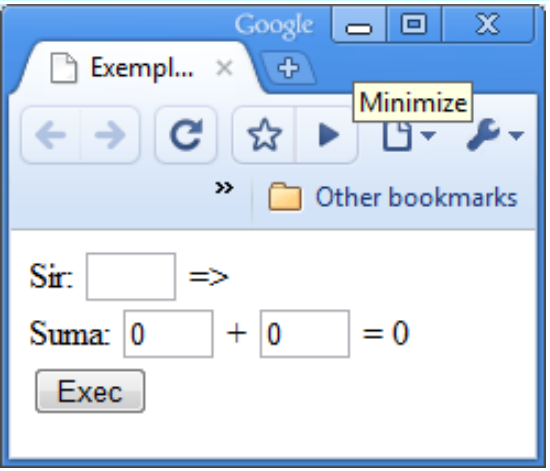
Variabile de mediu CGI

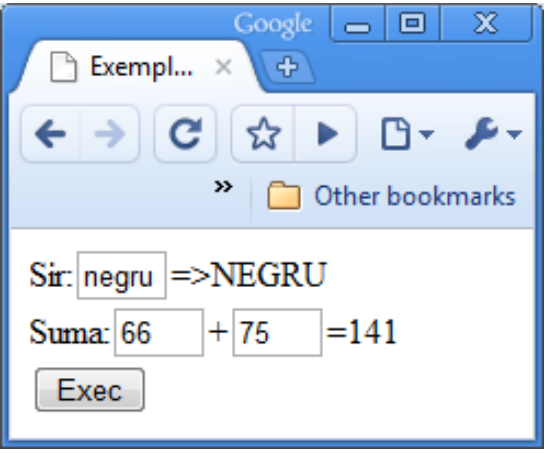
Serverul web preia numele CGI din URL, preia fisierul executabil din directorul destinat si lanseaza in executie un proces nou.

In urma incarcarii, sunt setate o serie de variabile de mediu, printre care amintim:

SERVER_NAME	Numele sau adresa IP a server-ului Web
SERVER_SOFTWARE	Software-ul server-ului Web (IIS, Apache etc.)
SERVER_PROTOCOL	Protocolul de comunicare
SERVER_PORT	Portul de aşteptare
REQUEST_METHOD	Metoda de cerere: GET, POST, PUT, DELETE etc.
SCRIPT_NAME	Calea şi numele CGI (de ex. /cgi-bin/TestCgi.cgi)
DOCUMENT_ROOT	Rădăcina server-ului Web
QUERY_STRING	Şirul de cerere din URL , metoda GET
REMOTE-HOST	Numele maşinii client
REMOTE_ADDR	Adresa IP a maşinii client
REMOTE_USER	Numele utilizatorului client
CONTENT_TYPE	Tipul MIME al cererii (de exemplu text/html)
CONTENT_LENGTH	Lungimea (în octeţi) a corpului cererii
HTTP_USER_AGENT	Numele browser-ului client
HTTP_REFFERER	URL-ul documentului accesat de client înainte de CGI

Exec: exemplul suport al cursului






```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
  "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/2899/xhtml" >
  <head><title>Exemplu suport</title></head>
  <body>
    <form action="- - - " method="{GET|POST}">
      <div>
        Sir:
       <input type="text" name="sir" size="1" value="">
        =&gt;
        <label id="upcase"></label>
      </div>
      <div>
        Suma:
      <input type="text" name="nr1" size="1" value="0">
        +
      <input type="text" name="nr2" size="1" value="0">
        =
        <label id="suma">0</label>
      </div>
      <input type=submit value= "Exec">
    </form>
  </body>
</html>
```

In **1CgiSiEchivalente/** sunt prezentate implementari CGI ale exemplului suport in C, Python, NodeJs, precum si Servlet, echivalentul Java al CGI.

Setarile de configurare CGI la Apache se fac in fisierul **httpd.conf**
Directorul implicit pentru CGI: **/cgi-bin/**
Poate fi setat sub forma: **ScriptAlias /cgi-bin "c:/xampp/cgi-bin/"**
Se pot stabili si alte directoare ce pot gazdui CGI inafara spatiului httpd daca se seteaza adecvat taguri Directory. De exemplu mai jos se pot defini pagini web in toate subdirectoarele pe trei nivele sub /home/scs daca acolo se afla un fisier public_html. In cadrul acestuia subdirectorul cgi-bin poate gazdui CGI-uri proprii :

```
<Directory /home/scs/*/*/*/public_html>
   Options Indexes FollowSymLinks ExecCGI
   AllowOverride None
</Directory>
```

Se pot defini tipuri speciale de fisiere executabile (in absenta pot fi de orice tip) sub forma:

```
AddHandler cgi-script .cgi .pl
```

Pentru conectare Tomcat-Apache, vezi
http://www.cs.ubbcluj.ro/~florin/TPJAD/ExempleSurseDocumentatii/4Servlets/BFM_TPJAD_Servlets.pptx
Slide-urile 24-25 Conexiune server web – container servlet

Surogat de server web prin socket (vezi **2SurogatServereWebSocket/**)

```
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((“localhost”, 80))
s.listen(1)
conn, addr = s.accept()
while 1:
    data = conn.recv(1024)
    if not data: break
    print data
conn.close()
```

Acest program (din stanga), odata lansat, este un surogat de server web. Dupa lansarea in browser a exemplului suport, programul surogat de server web afiseaza un text ca mai jos:

```
POST /cgi-bin/Exec.exe HTTP/1.1
Host: localhost
Connection: keep-alive
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US)
AppleWebKit/532.5 (K
HTML, like Gecko) Chrome/4.0.249.89 Safari/532.5
Content-Length: 23
Cache-Control: max-age=0
Origin: file://
Content-Type: application/x-www-form-urlencoded
Accept:
application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,i
mage/png,*/*;q=0.5
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-US,en;q=0.8
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3

sir=negru&nr1=66&nr2=75
```

Surogat de client browser prin socket (vezi **3SurogatClientiSocket/**)

```
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((“localhost”, 80))
req = """
POST /cgi-bin/Exec.exe HTTP/1.1
Host: localhost
Content-Length: 23

sir=negru&nr1=66&nr2=75
"""
s.send(req)
while 1:
  data = s.recv(1024)
  if not data: break
  print data
s.close()
```

Acest program (din stanga) este un surogat de browser. Mai intai se lanseaza un server web pe masina locala (de exemplu Apache) care contine si un CGI de tratare a cererii. Dupa ce se lanseaza acest surogat de browser, se va afisa ca rezultat textul de mai jos.

```
HTTP/1.1 200 OK
Date: Wed, 17 Mar 2010 07:00:55 GMT
Server: Apache/2.2.14 (Win32) DAV/2 mod_ssl/2.2.14 OpenSSL/0.9.8l mod_autoindex_
color PHP/5.3.1 mod_apreq2-20090110/2.7.1 mod_perl/2.0.4 Perl/v5.10.1
Transfer-Encoding: chunked
Content-Type: text/html

218
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML
1.1//EN""http://www.w3.org/TR/xhtml11/D
TD/xhtml11.dtd"><html xmlns="http://www.w3.org/2899/xhtml" ><head><title>Exemplu
 CGI</title></head><body><form action="/cgi-bin/Exec.exe" method="GET"><div>Sir:
<input type="text" name="sir" size="1" value="negru">=&gt;<label id="upcase">NEG
RU</label></div><div>Suma:<input type="text" name="nr1" size="1" value="66">+<in
put type="text" name="nr2" size="1" value="75">=<label id="suma">141</label></di
v><input type=submit value= "Exec"></form></body></html>

0
```

Implementarea serverului MiniHttpd (vezi **4MiniHttpd/**)

Pentru a ilustra functionarea unui server web cu un CGI am implementat un exemplu ultrasimplificat de server web: **MiniHttpd.** Implementarea este realizata printr-o arhiva jar autolansabila **MiniHttpd.jar**.
Fisierul de parametri **MiniHttpd.properties** defineste:

- **port** la care asteapta cereri (in absenta va fi portul 80),
- **cgiDir** directorul absolut unde sa caute programele CGI si
- **htmlDir** directorul absolut de unde sa ia fisierele HTML solicitate.

MiniHttpd.java este programul principal. Citeste fisierul de parametri, creaza socketul de asteptare, iar pentru fiecare cerere creaza un thread caruia ii transmite socketul de schimb.

ThreadClient.java serveste cererea unui client. Deschide canalele de schimb pe socket, afiseaza o serie de mesaje informative, dupa caz cere de la **Cgi.java**, respectiv de la **Html.java** raspunsul la cerere. Scrie acest raspuns la client.

Cgi.java lanseaza prin **Runtime.getRuntime().exec** executia CGI-ului, prelucrand intrarea standard, variabilele de mediu si iesirea standard ale acestuia.

Html.java citeste fisierul HTML solicitat si il trimite ca raspuns pe iesirea standard.

Uri.java este clasa de serviciu care intoarce componentele unui URI, clasa pe cara am mai prezentat-o.

Clienti MiniHttpd (vezi **5ClientiMiniHttpd/**)

Am construit cativa clienti foarte simpli pentru serverul MiniHttpd scrisi in HTML sau in Python.

Sunt trei clienti HTML, apelabili din browser:
http://localhost/h.html http://localhost/g.html http://localhost/p.html

- **h.html** este un document simplu HTML.
- **g.html** si **p.html** sunt doua formulare HTML care invoca CGI-ul **Exec.{exe|py…}** ce implementeaza exemplul suport, prin metoda GET respectiv prin metoda POST.

Atentie! Browserul, care “asteapta un server web veritabil”, va solicita de la acesta si un fisier imagine **favicon.ico** pe care noi nu il avem, asa ca, pe langa executie, MiniHttp va arunca `java.io.FileNotFoundException: - - - favicon.ico - - -`

Urmatorii trei sunt clienti standalone apelabili:
`python h.py` `python g.py` `python p.py`

- **h.py** cere de la **MiniHttpd** documentul **h.html**.
- **g.py** si **p.py** cere de la **MiniHttpd** sa execute CGI-ul **Exec** ce implementeaza exemplul suport, prin metoda GET respectiv prin metoda POST.

Ca rezultat, acesti clienti afiseaza pe iesirea standard documentele HTML de raspuns.

O categorie aparte de instrumente de dezvoltare, bazate de asemenea pe CGI, sunt aşa-numitele *pagini de scripting server* – pe care le vom numi generic ***SSP Scripting Server Pages*** si in ele sunt incluse tipuri de pagini: ***ASP, ASP.NET, JSP, PHP, PSP s.a.*** Punerea acestora sub acelaşi termen generic este o chestiune naturală, după cum se va vedea din implementarile SSP ale exemplului suport.

O pagină SSP **este scrisă în două grupuri de limbaje:**
Un prim grup descrie forme şi comportări *la nivel de client*: **(X)HTML, CSS, JavaScript**.
Un limbaj de scripting descrie comportamente şi *acţiuni la server*:

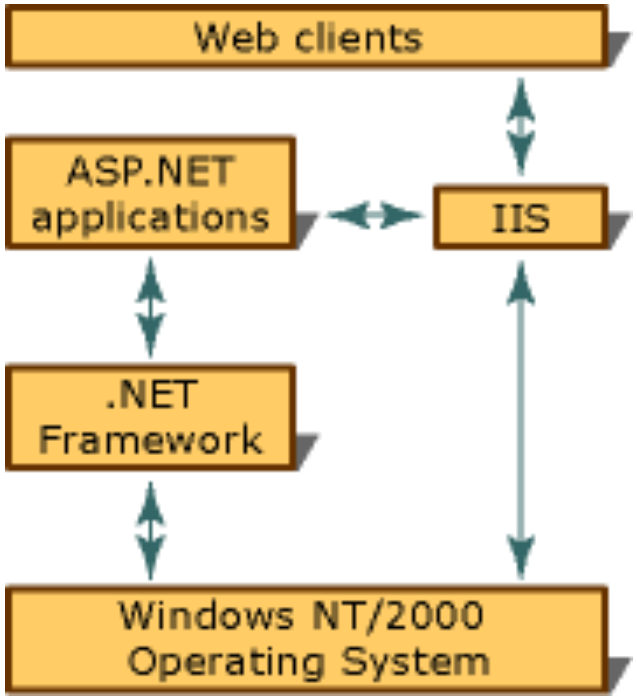
- **ASP** - limbajul MS Visual Basic sau Vbscript.
- **ASP**.**NET** - unul dintre limbajele C#, Visual Basic .NET, Vbscript .NET etc.
- **JSP** – limbajul Java (vezi cursul TPJAD)
- **Thymeleaf** derivat din Java folosit de Spring pentru wiev
- **PHP** – limbajul PHP
- **PSP** – limbajul Python

Specificarea scripturilor pentru partea de server:

ASP, ASP.NET, JSP, PSP: **<% - - - %>**.
PHP **<?php - - - ?>**, optional: **<% - - -%>**.
ASP.NET compatible XML: **<asp:nume - - - />**.
JSP compatible XML: **<jsp:nume - - - />**.

ASP.NET

In esenta, IIS transforma paginile ASP.NET in cod echivalent al unui CGI scris in limbajul CLR (Common Language Runtime). Apoi acest CGI este lansat in executie:



PHP

1. Browserul face o cerere HTTP la server pentru pagina (cu cod) PHP.
2. În funcție de configurația serverului, acesta isi va da seama (bazat pe directorul și / sau extensia de fișier) că cererea nu este un conținut static, ci o solicitare de a evalua codul PHP dintr-un document HTML.
3. Se extrage din pagina si se leaga codul PHP din serverul web la evaluatorul de cod PHP. Acest evaluator se comporta ca si un CGI care are ca intrare textul de evaluat.
4. Evaluatorul de cod PHP executa in mod interpretativ codul si intoarce la stdout rezultatul evaluarii, exact cum o va face un CGI oarecare, rezultat preluat de serverul web.
5. Serverul web intoarce raspunsul catre client (browser).

Urmeaza implementarile exemplului suport in SSP

```
<%@ Page Language="C#" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<script runat="server">
  void runExec(Object sender, EventArgs e) {
    string s = sir.Text;
    int n1 = int.Parse(nr1.Text);
    int n2 = int.Parse(nr2.Text);
    sir.Text = s;
    nr1.Text = n1;
    nr2.Text = n2;
    upcase.Text = s.ToUpper();
    suma.Text = (n1 + n2);
  }
</script>
```

```
<html xmlns="http://www.w3.org/2899/xhtml" >
  <head runat="server"><title>Exemplu ASP.NET</title></head>
  <body>
    <form id="form1" runat="server">
      <div>
        Sir:   <asp:TextBox id="sir" runat="server" size="1" value=""/>  =&gt;
        <label id="upcase"></label>
      </div>
      <div>
        Suma: <asp:TextBox id="nr1" runat="server" size="1" value="0"/>  +
        <asp:TextBox id="nr2" runat="server" size="1" value="0">     =
        <label id="suma">0</label>
      </div>
      <asp:Button text="Exec" runat="server" OnClick="runExec"/>
    </form>
 </body>
</html>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
  "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/2899/xhtml" >
  <head><title>Exemplu JSP</title></head>
  <body>  <%   String sir = request.getParameter("sir");
      int nr1 = Integer.parseInt(request.getParameter("nr1"));
      int nr2 = Integer.parseInt(request.getParameter("nr2"));    %>
    <form action="/Exec.jsp" method="get" >
      <div>
        Sir:  <input type="text" name="sir" size="1" value="<%=sir%>">  =&gt;
        <label id="upcase"><%=sir.toUpperCase()%></label>
      </div>
      <div>
        Suma:  <input type="text" name="nr1" size="1" value="<%=nr1%>">   +
        <input type="text" name="nr2" size="1" value="<%=nr2%>">    =
        <label id="suma"><%=(nr1 + nr2)%></label>
      </div>
      <input type=submit value= "Exec">
    </form>
  </body>
</html>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
 "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/2899/xhtml" >
 <head><title>Exemplu PHP</title></head>  <body>    <?php
    $sir = $_GET['sir'];       $nr1 = $_GET['nr1'];       $nr2 = $_GET['nr2'];     ?>
   <form action="/Exec.php" method="get" >
    <div>
Sir:<input type="text" name="sir" size="1" value="<?php echo $sir; ?>">  =&gt;
      <label id="upcase"><?php echo strtoupper($sir); ?></label>
    </div>
    <div>
Suma: <input type="text" name="nr1" size="1" value="<?php echo $nr1; ?>">+
     <input type="text" name="nr2" size="1" value="<?php echo $nr2; ?>">      =
      <label id="suma"><?php echo ($nr1 + $nr2); ?></label>
    </div>
    <input type=submit value= "Exec">
   </form>
 </body>
</html>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
  "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/2899/xhtml" >
  <head><title>Exemplu PSP</title></head>  <body>    <%
sir = form["sir"]
nr1 = int(form["nr1"])
nr2 = int(form["nr2"])    %>
    <form action="/psp/Exec.psp" method="get" >
      <div>
        Sir: <input type="text" name="sir" size="1" value="<%=sir%>">   =&gt;
        <label id="upcase"><%=sir.upper()%></label>
      </div>
      <div>
      Suma:<input type="text" name="nr1" size="1" value="<%=str(nr1)%>">   +
        <input type="text" name="nr2" size="1" value="<%=str(nr2)%>">     =
        <label id="suma"><%=str(nr1 + nr2)%></label>
      </div>
      <input type=submit value= "Exec">
    </form>
  </body>
</html>
```

Conexiuni URL ale clientilor standalone (vezi **6ClientiUrl/**).

Scopul demersului nostru este acela de a studia posibilităţi de a accesa resurse Web prin intermediul programelor de aplicaţii fara a utiliza browsere. Pentru aceasta, programele clienti trebuie sa realizeze conexiuni la resursele web. Vom ilustra, pentru clientii standalone scrisi in C#, Java, PHP, Python si NodeJs, care sunt modalitatile de realizare ale acestor conexiuni.

In **6ClientiUrl/** sunt prezentate astfel de conexiuni pentru exemplul suport in cele 5 limbaje. Caracteristic, aceste conexiuni folosesc:

- **C#** foloseste pentru conexiune obiecte de tip HttpRequest si HttpResponse.
- **Java** foloseste obiecte de tip URL, HttpUrlConnection (uneori doar UrlConnection).
- **PHP** foloseste biblioteca CURL
- **Python** foloseste modulul http.client si obiecte de tip HTTPConnection
- **NodeJs** foloseste modulul http

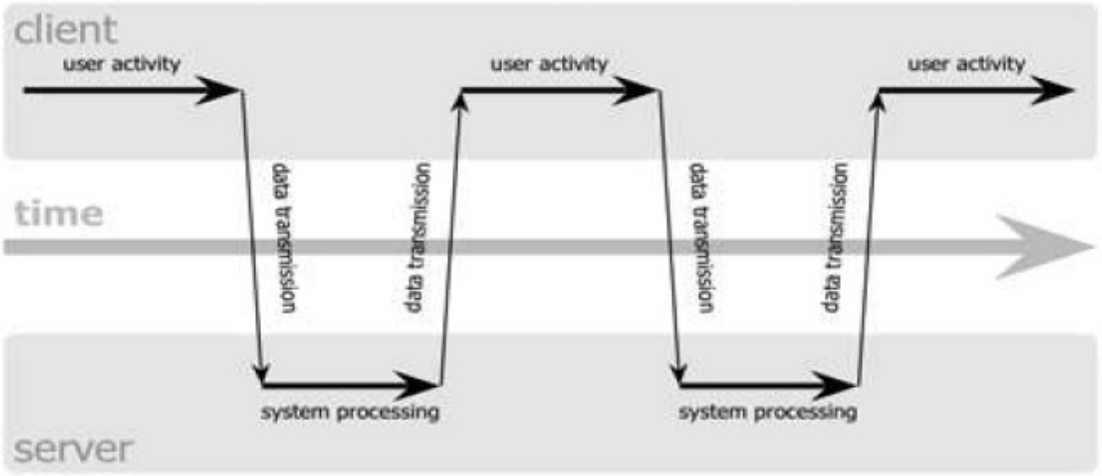
Ajax – combinatie de tehnologii open source:

- Limbaje de prezentare bazate pe XHTML şi pe CSS;
- Interacţiuni şi afişări dinamice bazate pe DOM (Document Object Model);
- Schimb de date şi manipulari folosind XML si XSTL;
- Transport asincron de date folosind XMLHttpRequest;
- Integrare la client folosind JavaScript.

Tehnologii bazate pe Ajax:

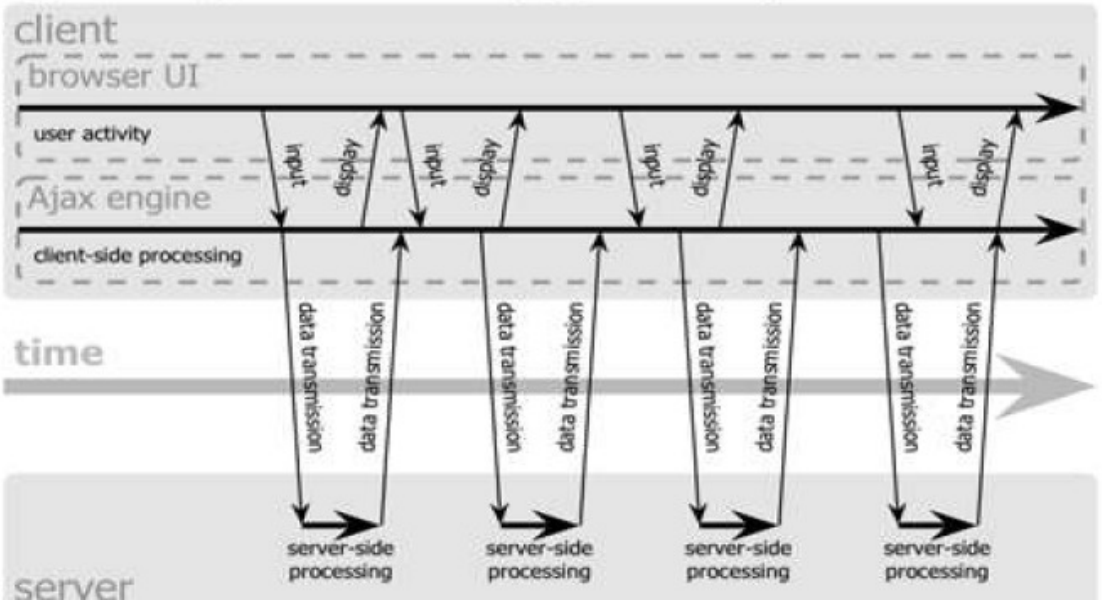
- Prototype JavaScript framework (**http://prototype.conio.net**)
- Ajax Pages (**ajax-pages.sourceforge.net**)
- DWR Direct Web Remoting (**http://getahead.ltd.uk/dwr**)
- ZK de la Potix Corporation (**zk1.sourceforge.net**)
- Dojo toolkit (**www.javapassion.com**)
- **Google**: Orkut, Gmail, Google Groups, Google Suggest, Google Maps etc.

classic web application model (synchronous)



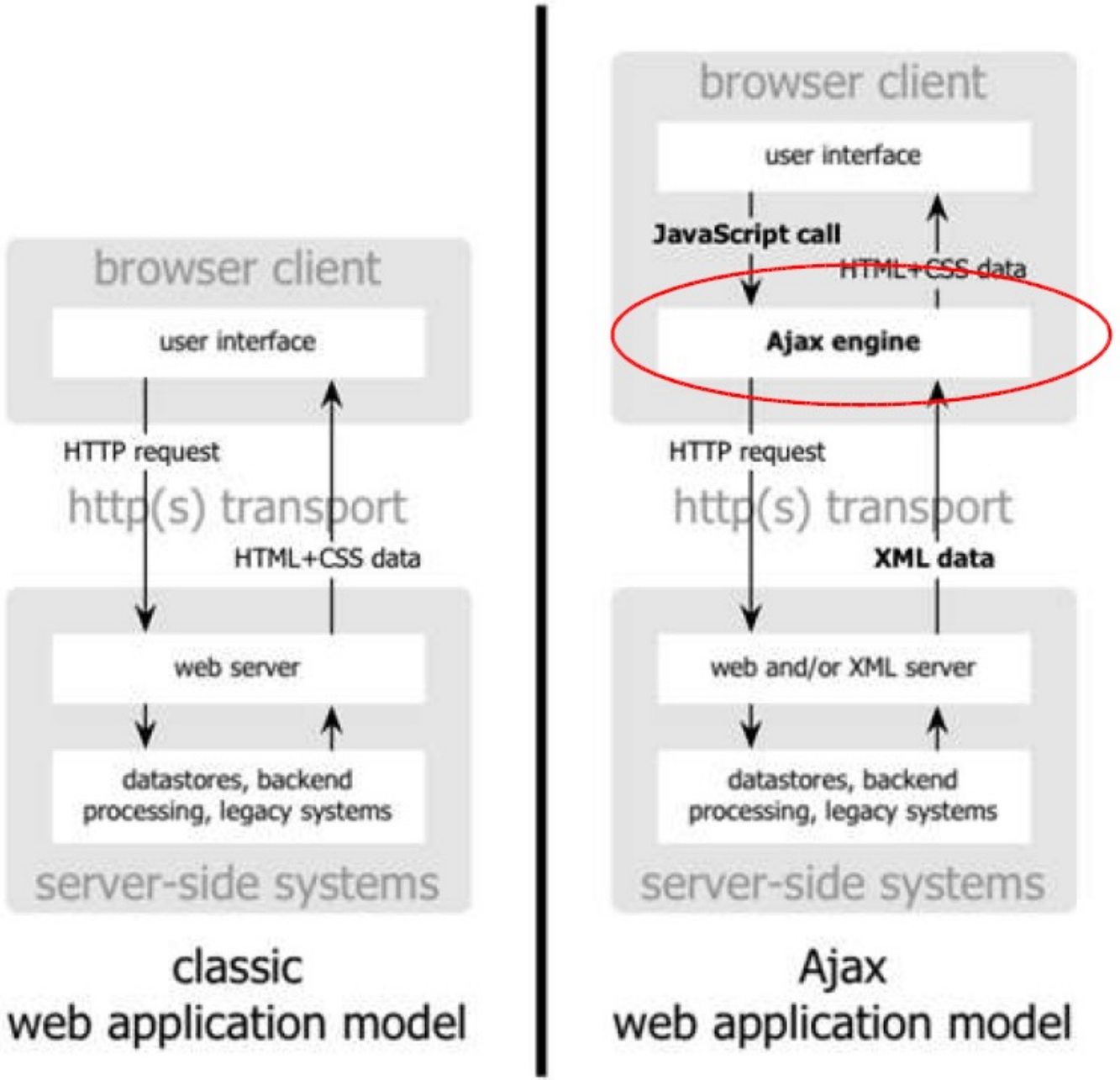
Interrupted user operation while the data is being fetched

Ajax web application model (asynchronous)



Uninterrupted user operation while data is being fetched

Acces clasic vs. acces Ajax



XMLHttpRequest – definire, metode

```
var cerere = new XMLHttpRequest(); // Mozilla, Opera, Netscape, Firefox, Safari etc.

var cerere = new ActiveXObject('MSXML2.XMLHTTP');  // IE # versiuni
var cerere = new ActiveXObject('Microsoft.XMLHTTP'); // IE # versiuni
```

Metoda	Descriere
open(method, URL) **open(method, URL, async)** **open(method, URL, async, userName)** **open(method, URL, async, userName, password)**	Specifică metoda, URL, şi alte atribute opţionale ale cererii: method poate avea una dintre valorile "GET", "POST", "PUT“, “DELETE” URL poate fi unul relativ sau unul absolut async specifică dacă cererea se face asincron sau nu. De regulă valoarea este true. In cazul false există riscuri de impas!
send(content)	Trimite stringul de cerere (nume=value&nume=value . . .)
getResponseHeader(headerName)	Intoarce valoarea headerului specificat.
setRequestHeader(label, value)	Adaugă o pereche label/value la headerul HTTP care va fi trimis.
getAllResponseHeaders()	Intoarce setul complet de headere.
abort()	Anulează cererea curentă.

Proprietate	Descriere
readyState	Intoarce starea obiectului: 0 = neiniţializat (nu s-a apelat open) 1 = open - cerere în curs de încărcare (înainte de send) 2 = sent - cerere încărcată, dar răspuns încă nesosit 3 = receiving – răspuns în process de sosire 4 = loaded – cerere rezolvată
onreadystatechange	Oferă mecanismul de tratare a evenimentului readystate==4. Valoarea ei este, de regulă, funcţia (JavaScript) care procesează răspunsul.
status	Codul răspunsului (Http) de la server: 200 for "OK", 404 for "Not Found" etc .
statusText	Răspunsul de mai sus dat ca şi string: “OK”, “Not Found” etc.
responseText	Răspunsul, prezentat ca şi un string.
responseXML	Răspunsul, prezentat ca şi un document XML. Acest răspuns poate fi parsat (la browser – motorul) folosind W3C DOM şi apoi să actualizeze pagina.

Sursele acestui exemplu se gasesc la 7**ExecAjax/**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
 "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/2899/xhtml" >
 <head>   <title>Exemplu AJAX PHP</title>
  <script type="text/javascript" src="ExecAjax.js"></script>
 </head>
 <body>   <form>     <div>
    Sir:
    <input type="text" id="sir" size="1" onBlur="doRequestSir(this.value)" value=""/>
    =&gt;
    <input type="text" id="upcase" size="1" disabled="disabled"/>
  </div>     <div>
    Suma:
    <input type="text" id="nr1" size="1" onBlur="doRequestSuma('nr1', this.value)" value="0"/>
    +
    <input type="text" id="nr2" size="1" onBlur="doRequestSuma('nr2', this.value)" value="0"/>
    =
    <input type="text" id="suma" size="1" disabled="disabled"/>
  </div>   </form>   </body>   </html>
```

Exemplu AJAX cu trei campuri: **ExecAjax.js** (1/2)

```
var url, request, id;   url = "http://localhost/ExecAjax.php";

function doRequestSir(value) {
  // value - stringul de transmis la server
  request = getXmlHttpRequest(); // necesar la fiecare apel!
  request.onreadystatechange = waitResponseSir;  // Functia ce proceseaza raspunsul
  request.open("GET", url+"?sir="+value, true); // deschide conexiunea cu serverul
  request.send(""); // trimite cererea la server
}
function doRequestSuma(name, value) {
  // name  - numele campului transmis la server;  value - stringul de transmis la server
  request = getXmlHttpRequest(); // necesar la fiecare apel!
  request.onreadystatechange = waitResponseSuma;  // Functia ce proceseaza raspunsul
  request.open("GET", url+"?"+name+"="+value, true); // deschide conexiunea cu serverul
  request.send(""); // trimite cererea la server
}

function getXmlHttpRequest() {   try { request = new XMLHttpRequest();
  } catch (e) { try { request = new ActiveXObject('MSXML2.XMLHTTP');
    } catch (e) { try { request = new ActiveXObject('Microsoft.XMLHTTP');
      } catch (e) { alert("Browserul nu poate lucra cu AJAX: "+e); request = false; } } }
  return request;}
```

Exemplu AJAX cu trei campuri: **ExecAjax.js** (2/2)

```
function waitResponseSir() { // Depune textul de la server in campul indexat cu id
  if (request.readyState == 4) { // cerere rezolvata
    if (request.status == 200) { // raspuns Ok
      document.getElementById("upcase").value = request.responseText; // depune raspunsul
    } else {
      alert("Eroare request.status: "+request.status);
    }
  }
}

function waitResponseSuma() { // Depune textul de la server in campul indexat cu id
  if (request.readyState == 4) { // cerere rezolvata
    if (request.status == 200) { // raspuns Ok
      document.getElementById("suma").value = request.responseText; // depune raspunsul
    } else {
      alert("Eroare request.status: "+request.status);
    }
  }
}
```

Exemplu AJAX cu trei campuri: serverul **ExecAjax.php**

```php
<?php
  if (isset($_GET['sir'])) {
    $sir = $_GET['sir'];
    echo strtoupper($sir);
  } else if (isset($_GET['nr1'])) {
    $nr1 = $_GET['nr1'];
    setcookie('nr1', $nr1);
    $nr2 = 0;
    if (isset($_COOKIE['nr2'])) $nr2 = $_COOKIE['nr2'];
    echo $nr1 + $nr2;
  } else if (isset($_GET['nr2'])) {
    $nr2 = $_GET['nr2'];
    setcookie('nr2', $nr2);
    $nr1 = 0;
    if (isset($_COOKIE['nr1'])) $nr1 = $_COOKIE['nr1'];
    echo $nr1 + $nr2;
  }
?>
```

Web programming: prezentare generala + Ajax?

TITLU