Utilizarea tehnicilor servlet

Prof.dr.ing. Marian DOBRE, ing. Cristian MARINESCU
Catedra de Calculatoare, Universitatea POLITEHNICA Bucuresti

Servlet-ii reprezinta o tehnica noua, introdusa relativ recent în mediul Java, care deschide noi perspective în realizarea serverelor de web. Aceleasi facilitati si avantaje care au facut din Java un limbaj ideal pentru scrierea de aplicatii orientate client, ofera o platforma ideala de dezvoltare si pentru aplicatiile orientate server. Acestea vor beneficia atât de portabilitatea Java, cât si de securitatea si facilitatile de dezvoltare ulterioara rapida.
În lucrare este prezentata experienta de implementare a unui mediu de testare a cunostintelor asistata de calculator (teste grila), utilizând tehnicile servlet. Platforma Intranet care suporta acest mediu a fost realizata în cadrul proiectului PIP 214 (componenta "Colegiu"), finantat de CNFIS.
Cuvinte cheie: *aplicatie multi-tier, CGI, client-server, HTTP, Java, obiecte de tip Cookie, servlet.*

1. Introducere

De curând, firmele Sun si JavaSoft au introdus o noua componenta în Java, API-ul *Servlet*. Lumea Java se mai îmbogateste cu înca o componenta, dar nu una oarecare. Mediul Java nu mai este astfel orientat numai spre programarea clientilor prin applet-uri scrise sa ruleze în browser-ele de WWW, sau spre scrierea de aplicatii Internet. API-ul *Servlet* aduce simplitatea si flexibilitatea limbajului Java si în serverele de Web. Java devine astfel o platforma ideala pentru scrierea de software care ruleaza împreuna cu serverul de aplicatii orientate WWW.
Exista în momentul de fata o singura limitare, anume viteza. Codul scris în Java ruleaza de aproximativ 3-5 ori mai încet decât codul cu aceeasi functionalitate scris în limbajul C. Dar acest neajuns va disparea în timp, Java impunându-se ca o solutie portabila si flexibila pentru orice problema de programare.

1.1. Prezentare generala

Servlet-ii sunt componente scrise în Java, independente de platforma sau protocoale, care extind cu noi functionalitati serverele ce ofera suport Java. Se poate aprecia ca servlet-ii joaca la server acelasi rol pe care îl au applet-urile în cadrul clientului. Structura lor se bazeaza pe modelul *cerere – raspuns*. Acest model este utilizat într-o multitudine de limbaje distribuite si mecanisme oferite de acestea. Exemple notabile sunt: Remote Procedure Call (RPC), rendezvous, client-server.
Datorita faptului ca servlet-ii ruleaza în cadrul unor servere, ei nu beneficiaza de interfata grafica cu utilizatorul. Acest dezavantaj se poate remedia, acolo unde este cazul, prin introducerea unui thread de afisare a unor rezultate, sau chiar prin implementarea unei interfete pentru administratorul serverului.
În figura 1 este prezentata structura generala a unei aplicatii bazate pe Java Web Server. Clientii se conecteaza la server prin intermediul oricarui tip de retea, ei putând rula pe platforme diferite. Aplicatia se realizeaza prin adaugarea unor servlet-i la Java Server.
Tehnologia *Servlet* nu a fost dezvoltata doar în ideea de a extinde un anumit server de WWW. Proiectarea si realizarea ei a fost facuta în scopul de a extinde orice platforma ce ofera suport limbajului Java. Servlet-ii sunt suficient de flexibili pentru a sprijini servicii standardizate, cum ar fi servirea de pagini Web statice sau dinamice prin protocoalele HTTP (HyperText Transfer Protocol), HTTPS (o extindere a protocolului HTTP care include si criptarea pa-

chetelor), sau servicii de *proxy*. Deoarece au fost conceputi ca o extensie dinamica, facilitatile lor pot fi utilizate în realizarea unor motoare de cautare sau a altor servicii de retea.

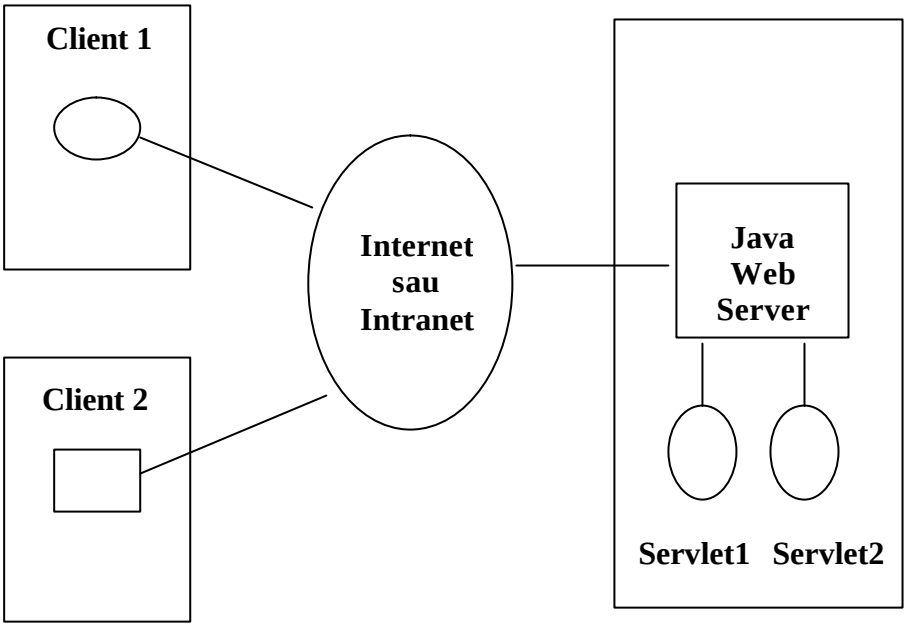

Fig. 1. Structura generala a unei aplicatii client-server care utilizeaza servlet-i

Desi toate aplicatiile cu servlet-i sunt realizate în Java, nu se impune nici o restrictie la scrierea aplicatiilor în partea de client, clientii putând fi implementati în orice limbaj de programare. La rândul lor, servlet-ii pot deveni clienti în aplicatiile mai mari de retea ce implica servicii mai complexe. De exemplu, un servlet poate utiliza facilitatile API-ului JDBC, pentru a realiza conectarea la un server de baze de date prin intermediul unui driver specific.

Exista multe variante si posibilitati de utilizare a facilitatilor puse la dispozitie de API-ul *Servlet*. Un servlet simplu poate prelucra informatia ce a fost trimisa prin intermediul unui formular HTML, trebuind sa faca înregistrari într-o baza de date si având posibilitatea de a raspunde cererii de prelucrare a informatiei continuta de formular. Se pot imagina nenumarate aplicatii de acest tip, plecând de la serviciile cele mai simple si terminând cu comertul electronic.

Ciclul de viata al servlet-ilor este o proprietate specifica a acestora. Servlet-ii se încarca în mod dinamic, serverele oferind facilitati de administrare a încarcarii si initializarii servlet-ilor. Exista si posibilitatea, de loc de neglijat, de a specifica încarcarea anumitor servlet-i la lansarea în executie a serverului. Servlet-ii sunt încarcati utilizând facilitatile normale ale oricarei clase scrise în Java, încarcarea putând fi facuta si de la o locatie aflata la distanta. Odata încarcati, ei devin parte integranta a serverului. Procesul de încarcare este transparent pentru utilizator, acesta trebuind sa specifice doar locatia servlet-ului (local sau la distanta), numele clasei ce contine servlet-ul si numele sub care servlet-ul va fi cunoscut de catre server (alias). Se ofera astfel o si mai mare flexibilitate în utilizarea acestei tehnologii orientate spre programarea de retea. Detaliile difera totusi de la server la server.

Dupa ce au fost încarcati, în ciclul de viata

al servlet-ilor intervin trei etape principale: initializare, utilizare si distrugere.
- Servlet-ii sunt activati de catre server printr-un apel a metodei *init* (pusa la dispozitie de clasa *HttpServlet* din pachetul *Servlet*). Programatorul poate supradefini aceasta metoda, pentru a efectua operatiile de initializare si pe cele de cost ridicat ce nu trebuiesc executate la fiecare apel al servlet-ului. Astfel se pot obtine performante superioare.
- Dupa initializare, servlet-ii servesc o multime de cereri (apelând metode cum ar fi *doGet*, *doPost* etc.). Fiecare client genereaza o cerere de serviciu (cererile diferitilor clienti pot fi concurente). O posibilitate de utilizare a unor date în comun de catre cereri este folosirea unor variabile declarate statice.
- Cererile sunt servite pâna când servlet-ul este descarcat în mod explicit de catre serverul de Web, prin apelul metodei *destroy*. În acest moment, spatiul ocupat de catre clasa *Servlet* devine disponibil pentru a fi recuperat de catre "garbage – collector".

Servlet-ii au fost proiectati pentru a putea oferi acces concurent la informatii. Aplicatia poate utiliza mecanisme de sincronizare, devenind astfel un motor pentru servirea unor cereri concurente multiple.

1.2. Comparatie între servlet-i si CGI-uri

CGI-urile (Common Gateway Interface) definesc o modalitate de a schimba informatii cu o aplicatie externa. Folosind CGI-uri, serverele de Web pot accesa programe, pot comunica cu acestea si pot realiza functii utile, cum ar fi accesul la baze de date. Pe lânga avantaje, CGI-urile introduc însa si o serie de probleme. De fiecare data când serverul de Web primeste o cerere ce necesita executia unui CGI, serverul trebuie sa activeze CGI-ul, aceasta implicând initializarea, executia lui si în final întoarcerea rezultatelor HTML. S-a recurs la solutia de a permite programatorului sa integreze în server diferite biblioteci (acestea nu necesita operatii de activare). A aparut însa o alta problema: o biblioteca prost scrisa poate foarte usor bloca serverul.

Servlet-ii reprezinta o încercare de rezolvare a problemelor introduse de catre CGI-uri. Prin faptul ca servlet-ii sunt clase Java, acestia nu duc la blocare în cazul aparitiilor unor erori. Clasele sunt încarcate de server si invocate de acesta. Odata încarcate, aceste clase devin parte a serverului de Web. În loc sa fie activati la fiecare cerere, servlet-ii se încarca o singura data, ducând la o îmbunatatire considerabila a performantelor. Pentru a mari stabilitatea si siguranta, multe din serverele de Web executa servlet-ii în asa zise cutii negre, izolate de restul serverului. Spre deosebire de plugin-urile scrise în C++, clasele Java nu vor duce la blocarea serverului.

Un alt avantaj îl reprezinta posibilitatea de a memora starea dintre doua apeluri. În protocolul HTTP aceasta caracteristica se poate dovedi foarte utila: servlet-ii pot supraveghea si înregistra actiunile utilizatorului în scopuri statistice sau în scopuri de securitate, putând genera pagini HTML individualizate în functie de activitatea utilizatorului. Un avantaj major este acela ca servlet-ii mostenesc caracteristicile limbajului Java, cum ar fi portabilitatea între platforme, recuperarea spatiului disponibil ("garbage-collection"), thread-urile si sincronizarea. În ceea ce priveste portabilitatea, aceasta se realizeaza nu numai fata de orice platforma, ci si fata de orice server si orice protocoale ale acestuia. Presupunând ca se implementeaza un algoritm într-un servlet, prin intermediul servlet-ului acesta poate fi introdus în diverse produse, indiferent daca acestea sunt servere de Web (HTTP), servere de fisiere (NFS), servere de posta electronica (SMTP) etc.

2. Solutia Servlet

2.1. Moduri de utilizare

Exista mai multe posibilitati de folosire a servlet-ilor, desi nu toate serverele suporta în acest moment integral modurile de functionare a acestora. Principalele directii

care se contureaza relativ la tehnicile de utilizare a servlet-ilor sunt:

- Modul de baza, orientat pe o functionare de genul cerere - raspuns. Este un mod ce permite specificarea de noi protocoale de comunicatie între servlet-i si applet-uri, si nu numai.
- Modul de lucru în "banda de asamblare". Un server poate înlantui o serie de servlet-i pentru a organiza o structura de lucru de gen filtru.
- Servlet-i specializati în lucrul cu protocolul HTTP, un înlocuitor ideal pentru CGI-uri. Tot în acest domeniu, servlet-ii pot deveni extensii orientate server pentru generare dinamica de pagini HTML.

Este la latitudinea programatorului sa aleaga modul de utilizare cel mai convenabil.

2.2. Generarea dinamica de pagini HTML

Posibilitatea de a genera direct raspunsul în format HTML mareste utilitatea servlet-ilor. Cu ajutorul clasei *java.io.PrintWriter* acest lucru devine foarte simplu. Nu este nevoie de un script pentru a genera în mod dinamic pagini HTML. În serverele de Web ce furnizeaza suport Java, servlet-ii pot fi apelati pentru a ajuta la o preprocesare a paginilor.

Exista doua modalitati de a lansa cereri dintr-un browser spre un servlet. Prima modalitate consta în includerea unui tag servlet în codul HTML, ca în exemplul din figura 2.

```
<SERVLET NAME="NumeServlet">
<PARAM NAME="NumeParam" VALUE=
"ValoareParam">
Daca ati reusit sa vizualizati acest text în-
seamna ca browserul dvs. nu cunoaste Tag-
ul Servlet.
</SERVLET>
```

Fig. 2. Exemplu de includere a unui tag servlet în cod HTML

Acest tag HTML indica faptul ca un servlet preconfigurat trebuie încarcat si initializat la server, apoi apelat cu parametrii corespunzatori. Initializarea si încarcarea se fac doar în cazul în care acestea nu au fost deja facute.

A doua posibilitate ar fi aceea de a-i asocia unui servlet un alias, un nume sub care sa fie cunoscut de catre server. Acest nume poate avea eventual extensia *.html*, sau poate contine o cale de cataloage. Un *tag link* catre aliasul servlet-ului va genera o cerere, care va fi convertita la server, iar dupa gasirea servlet-ului corespunzator se va genera un apel al acestuia cu parametrii actualizati. Toata aceasta operatie este transparenta pentru utilizator. Acesta poate apela servlet-i având senzatia ca lucreaza cu pagini HTML statice.

Utilizând aceste doua facilitati, servlet-ii ce recunosc formatul HTML pot genera în mod dinamic pagini de Web. În mod obisnuit, servlet-ii accepta ca parmetri de intrare:

- Sirul de intrare (input stream), ce poate fi generat de un applet,
- URL-ul cererii,
- Intrari de la alti servlet-i sau alte servicii,
- Parametrii introdusi în formularul HTML.

Acesti parametri vor fi utilizati pentru a genera raspunsul în format HTML. În cele mai multe cazuri, servlet-ul va interoga continutul unei baze de date, sau va face înregistrari într-o baza de date.

Servlet-ii ce se utilizeaza împreuna cu protocolul HTTP pot implementa si oferi suport oricareia din metodele *GET*, *POST*, *HEAD* etc. Ei sunt capabili sa redirectioneze cererile spre alte locatii, sau sa genereze erori specifice pentru protocolul HTTP. Ei pot avea acces la parametrii introdusi printr-un formular HTML, inclusiv la metoda (care trebuie executata în asociatie cu acest formular) si la URL-ul ce identifica destinatia cererii. În figura 3 sunt prezentate câteva exemple de metode specifice programarii cu servlet-i.

```
// Obtinerea metodei indicate în formularul HTML.
String metoda = cerere.getMethod();
// Obtinerea parametrului “Prenume” specificat în formularul HTML.
String prenume = cerere.getParameter(“Prenume”);
// Obtinerea parametrului “Nume” specificat în formularul HTML.
String prenume = cerere.getParameter(“Nume”);
```

Fig. 3. Exemple de metode specifice programarii cu servlet-i

Clasa *javax.servlet.http.HttpServlet* sta la baza ierarhiei ce implementeaza servlet-ii HTTP. Ea este responsabila de rezolvarea problemelor ce tin de caching, subprotocoale sau tratarea erorilor. Pentru servlet-ii HTTP datele sunt întotdeauna furnizate în format MIME (initial cu semnificatia Multipurpose Internet Mail Extension, ulterior prescurtare pentru Multipurpose Internet Multimedia Extension). Servlet-ul va specifica tipul datelor de raspuns (ce format MIME se foloseste la raspuns). Aceasta optiune permite servlet-ilor sa poata raspunde în orice fel de format cum ar fi: HTML, imagini JPEG, MPEG sau GIF, sau alte formate recunoscute de aplicatii specializate. Servlet-ii implementeaza interfata *Servlet*, extinzând o implementare generica sau una specifica HTTP/HTTPS. În figura 4 este prezentat un exemplu simplu de Servlet specific HTTP:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class SimpleServlet extends HttpServlet {
        public void doGet(HttpServletRequest cerere,
                HttpServletResponse raspuns)
                throws ServletException, IOException {
    // Se obtine streamul de iesire al Servletului.
    ServletOutputStream out = raspuns.getOutputStream();
    // Se seteaza continutul documentului cu care se raspunde.
    raspuns.setContentType("text/html");
    // Se scrie la iesire un document în format HTML.
    out.println("<HEAD><TITLE>Servlet foarte
        simplu.</TITLE></HEAD><BODY>");
    out.println("<h1>Exemplu de Servlet foarte simplu</h1>");
    out.println("<P>Informatie tiparita de un servlet foarte simplu.");
    out.println("</BODY>");
    // Inchidere stream de iesire.
    out.close();
    }
  public String getServletInfo() {
    return "Exemplu de servlet simplu";
    }
}
```

Fig. 4. Exemplu de servlet specific HTTP

În exemplul considerat, metoda *doGet* este dotata cu parametrii cerere si raspuns. Primul parametru contine datele trimise de client, facilitând accesul la diversi parametri de

intrare, iar al doilea permite servlet-ilor raportarea de erori si trimiterea unui raspuns catre client. Sirul de intrare poate fi obtinut aplicând metoda *getInputStream* variabilei cerere:

```
ServletInputStream in =
        cerere.getInputStream();
```

Din sirul de intrare servlet-ul îsi va afla parametrii (de exemplu trimisi dintr-un applet). Standardul nu impune nici o restrictie asupra formatului de intrare sau iesire, un applet putând schimba date cu un servlet printr-un protocol oarecare, sau prin schimb de obiecte serializate. Raspunsul servlet-ului poate fi în format HTML, sau în alte formate, de exemplu formate de imagini (JPEG, GIF etc.).

Fiind clase, în momentul initializarii, servlet-ii au acces la date specifice fiecarei instante. Diferitele instante ale aceleiasi clase servlet pot fi initializate cu date diferite si utilizate ca atare. Datele transmise servletului în momentul initializarii permit acestuia realizarea unor operatii specifice cu ajutorul obiectului de tip *ServletConfig*, disponibil ca parametru al metodei *init*.

Servlet-ii HTTP sunt superiori programelor CGI deoarece aduc îmbunatatiri atât din punct de vedere al performantelor, cât si al flexibilitatii si portabilitatii.

2.3. Servlet-ii în aplicatii multiserver

Servlet-ii pot fi componente foarte importante în multe aplicatii orientate Internet. Un exemplu remarcabil este cel al aplicatiilor multi-server. Plecând de la modelul client – server, în multe aplicatii este necesara introducerea unuia sau mai multor niveluri suplimentare. Nu toate cererile pot fi servite de un server de Web, de exemplu; tratarea unei cereri necesita interventia unui program specializat. Serverul de Web este cel capabil sa primeasca cererea si sa lanseze, prin intermediul unui servlet, o cerere spre alte servere specializate. Un alt servlet va intercepta raspunsul de la serverul specializat, pentru a traduce acest raspuns într-un format oarecare pentru clientul ce a lansat cererea.

În figura 5 este prezentat modelul unei aplicatii multiserver, cunoscut sub numele *three - tier model*. Primul nivel este reprezentat de browser-ele ce trimit cereri spre un server, în exemplul considerat acesta fiind Java Web Server. Acesta reprezinta al doilea nivel al modelului. Java Web Server contine încapsulati un numar de servlet-i specializati în traducerea cererilor si trimiterea lor spre diferite alte servere (care reprezinta nivelul al treilea al modelului). Serverele specializate reactioneaza la aceste cereri printr-un raspuns sau un mesaj de eroare, raspunsurile primite fiind convertite de servlet-i în format HTML si trimise browser-elor pentru afisare

Orice organizatie care doreste sa puna la dispozitia tuturor utilizatorilor informatii dintr-o baza de date, fara a da acces utilizatorilor externi la aceasta baza de date, poate recurge la o astfel de solutie. Instalând serverul de Web pe un calculator cu rol de firewall, se ofera accesul la baza de date printr-un serviciu asemanator cu cel de *proxy*, fara însa a exista pericolul distrugerii prin acces neautorizat din afara. Accesul va fi limitat la actiunile servlet-ului asupra serverului de baze de date din interior.

3. Platforma Intranet pentru verificarea automata a cunostintelor

În vederea realizarii verificarii automate a cunostintelor prin teste grila pentru studentii de la Colegiu, a fost proiectata si realizata o platforma Intranet care pune la dispozitia profesorului o unealta flexibila de testare a cunostintelor asistata de calculator, usor de utilizat, instalat si administrat. Aplicatia se bazeaza pe o arhitectura client-server si utilizeaza toate facilitatile puse la dispozitie de limbajul Java. S-a optat pentru tehnologia Java din dorinta unei portabilitati usoare pe alte platforme. A fost realizata atât o interfata prietenoasa cu utilizatorul, cât si o securitate deosebita a aplicatiei si a informatiilor stocate în baza de date. S-a pus la dispozitia utilizatorilor un mediu de verificare a cunostintelor, mecanismele necesare administrarii rezultatelor la învatatura ale

studentilor, dar si instrumente pentru administrarea comoda a bazei de date.
Deoarece se doreste extinderea utilizarii pachetului de programe pentru desfasurarea examenului de licenta (care implica testarea unui numar foarte mare de subiecti) au fost implementate doua solutii si s-au facut masuratori de performante în vederea alegerii variantei optime si a elaborarii unor specificatii complete pentru produsul final. Prima varianta s-a bazat pe o arhitectura clasica client-server, iar a doua (care face obiectul prezentei lucrari) a fost realizata pe baza API-ului *Servlet* cu ajutorul produsului Java Web Server. Aceasta a doua solutie permite utilizarea oricarui browser de Web pe post de client.
.

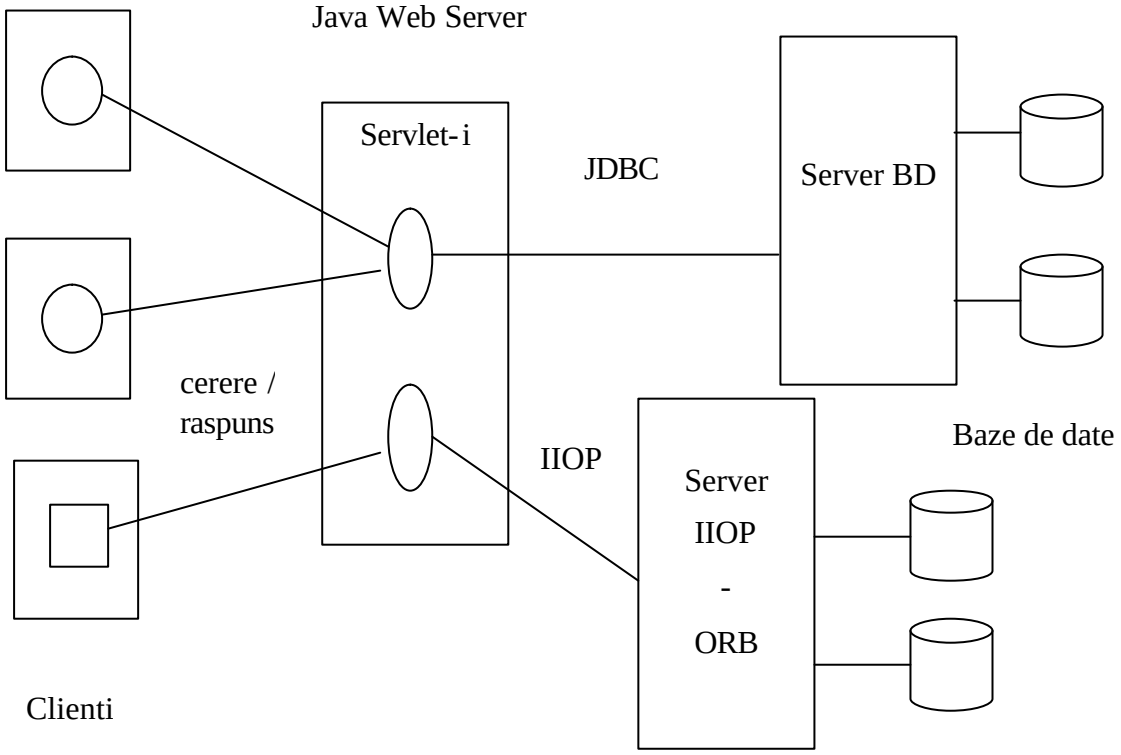

Fig. 5. Modelul three-tier

Solutia da posibilitatea de verificare a cunostintelor într-un mediu sigur, atât în mod grafic cât si în mod text. Datorita faptului ca proiectul a fost realizat pentru platforma existenta în Facultatea de Automatica si Calculatoare (servere accesate de la terminale alfanumerice), interfata a fost proiectata pentru medii text. Paginile HTML ale aplicatiei au fost generate special pentru vizionare cu un browser în mod text, ele putând însa la fel de bine sa fie afisate si într-un browser cu capacitati grafice.
Schema bloc a aplicatiei este prezentata în figura 6. S-a apelat la facilitatile de generare dinamica de pagini HTML ale servlet-ilor. A fost creat un lant de trei servlet-i care îndeplinesc urmatoarele functii:

- crearea unei pagini HTML pentru introducerea informatiilor necesare pentru autentificarea utilizatorului (**LoginText**),
- crearea unei pagini HTML cu întrebarile testului grila (**TestText**),
- crearea unei pagini HTML cu rezultatele testului (**CorecteazaTest**).

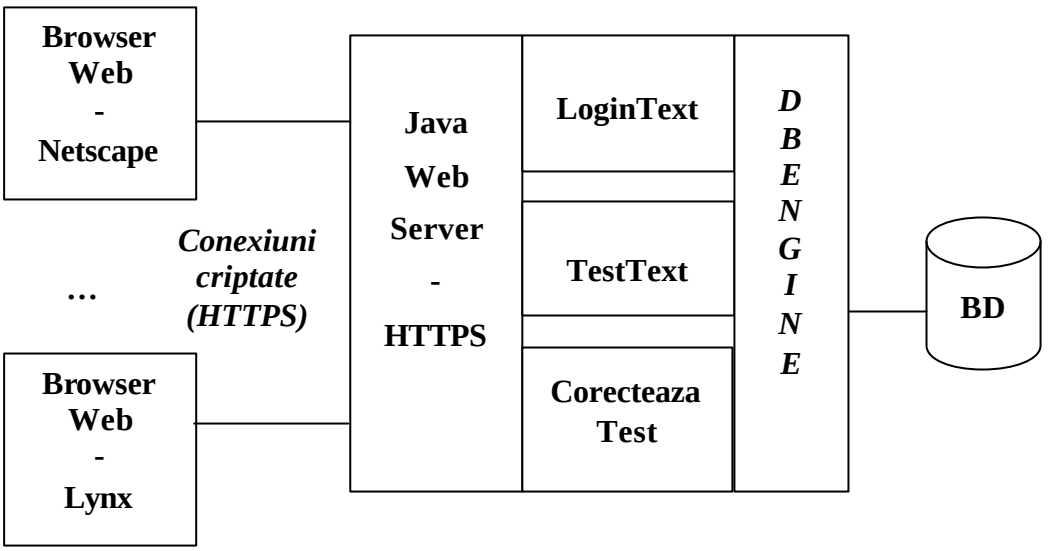

Fig. 6. Sistem de verificare automata a cunostintelor prin teste grila

Începerea testului grila se face prin apelarea unei pagini HTML statice, ce poate contine diferite informatii referitoare la natura testelor grila. Apoi se ofera posibilitatea de a alege între un test grila în care se folosesc capacitatile grafice ale browser-ului si unul alfanumeric, în functie de browser-ul utilizat si de terminalul disponibil.

Legatura inserata în pagina HTML apeleaza fisierul LoginText.html (fisier simulat de server prin crearea unui alias **/Test/LoginText.html** catre **LoginText**, clasa construita pentru logare). Clasa ce se ascunde în spatele acestuia citeste baza de date cu teste, pentru a furniza disciplinele existente în sistemul de testare si genereaza un formular HTML ce permite introducerea numelui, prenumelui, materiei, parolei, a grupei si a directiei de specializare a utilizatorului. Aceasta pagina este prezentata în figura 7.

La apasarea butonului **Submit** se trimite cererea de generare a paginii HTML cu întrebarile din testul grila ce urmeaza a fi parcurs de catre utilizator. Pentru început, clasa *TestText* verifica informatiile de autentificare. Daca datele introduse de utilizator sunt valide, urmeaza setarea unor obiecte de tip *Cookie* la browser-ul clientului, astfel încât sa se poata recunoaste persoana care va prezenta raspunsurile la test. Se acceseaza baza de date cu întrebari corespunzatoare disciplinei si se alcatuieste formularul cu întrebari. În cazul în care informatia de autentificare este incorecta, se genereaza o pagina de eroare.

La încheierea testului, prin apasarea butonului **Submit** din formularul cu întrebari, raspunsurile sunt trimise la server catre servlet-ul **CorecteazaTest**. Urmeaza o etapa de prezentare a obiectelor Cookie setate anterior si în cazul în care valorile acestor parametri corespund, se trece la corectarea automata a raspunsurilor. Se parcurge din nou baza de date cu întrebari si se compara raspunsurile subiectului testat cu raspunsurile corecte. În final se genereaza o pagina cu rezultatul obtinut la testul grila. Pe parcursul întregului test erorile care apar sunt captate si se genereaza pagini HTML de eroare.

4. Concluzii

Prin introducerea mecanismului servlet, limbajul Java ofera posibilitati promitatoare de extindere a serverelor Internet, cu functionalitati diverse. Servlet-ii reprezinta pentru server ceea ce applet-urile au însemnat pentru clienti.

Servlet-ii ofera prin capabilitatile lor un mediu foarte prietenos, beneficiind si de

facilitatile sistemului performant Java de dezvoltare de aplicatii.

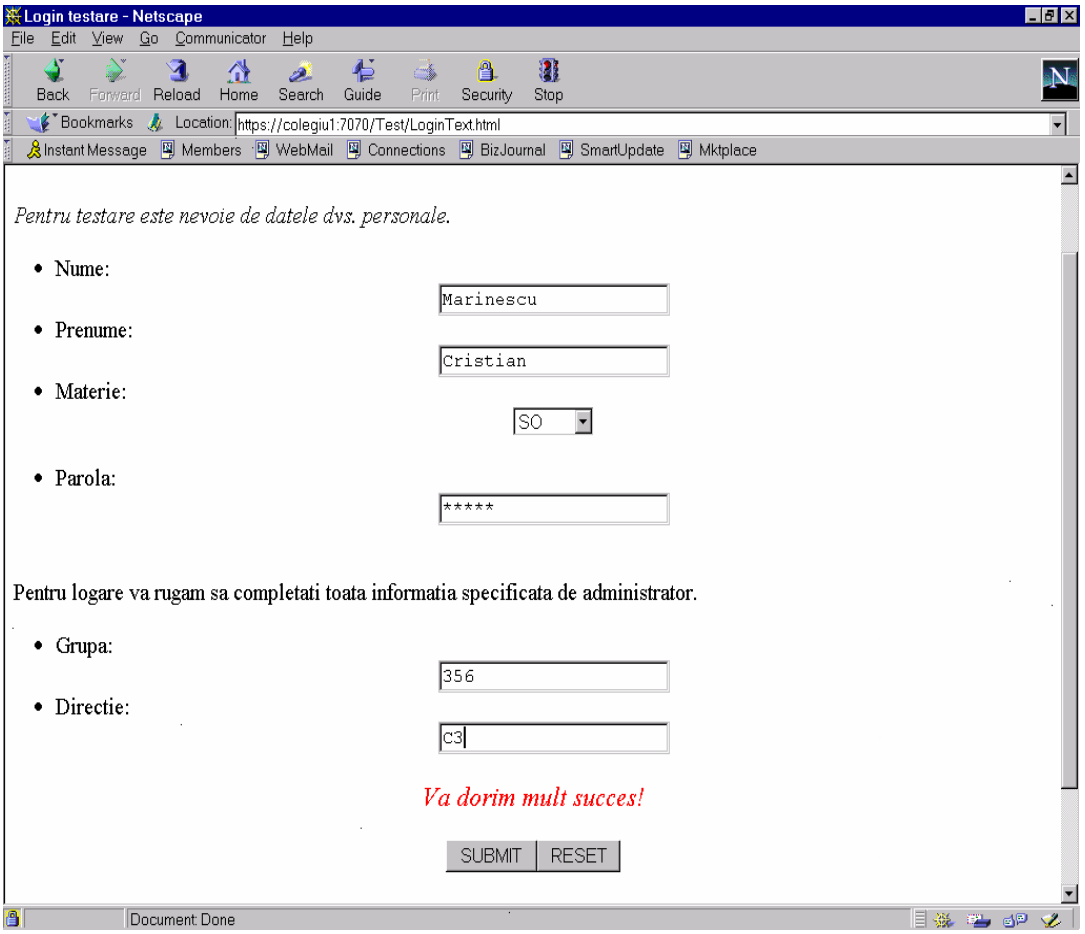

Fig. 7. Pagina de deschidere a sesiunii de testare

Datorita implementarii masinii virtuale Java si pe arhitecturi multiprocesor, servlet-ii care ruleaza în aplicatii dezvoltate pe astfel de arhitecturi asigura un timp de raspuns deosebit de bun.

Utilizarea servlet-ilor aduce avantaje indiscutabile în raport cu tehnicile CGI, mai putin ineficienta acceptata a aplicatiilor Java. Unul dintre cele mai mari avantaje ale servlet-ilor în raport cu CGI-urile este acela ca nu este necesara crearea de noi procese pentru fiecare cerere lansata de un client. Pe cele mai multe platforme, servlet-ii evolueaza în paralel cu serverul în cadrul aceluiasi proces. Ei ruleaza ca thread-uri, comutarea între diferite thread-uri facându-se deosebit de rapid. La CGI-uri este necesara o comutare între procese, ce are ca efect un overhead considerabil mai mare comparativ cu cel din cazul thread-urilor. Deoarece servlet-ii pot deservi în paralel mai multe cereri, se cauta încarcarea lor la initializare, astfel încât sa se elimine operatiile redundante de la fiecare cerere. Exista posibiliatea de a partaja anumite resurse (date) între diferite instante ale aceluiasi servlet; eficienta este sporita, deoarece se beneficiaza si de caching-ul asigurat de sistem.

Tehnicile *Servlet* prezentate în aceasta lucrare au fost utilizate cu succes pentru proiectarea si implementarea unui pachet de programe pentru verificarea cunostintelor prin teste grila, verificare facuta automat cu ajutorul calculatorului. Produsul obtinut este pe deplin functional si poate fi utilizat de catre orice profesor pentru a-si usura munca de verificare a cunostintelor studentilor sai.

Aplicatia urmeaza sa fie dezvoltata pentru a putea fi utilizata în cadrul examenului de licenta.
Java Web Server s-a dovedit a fi mecanismul ideal pentru implementarea si ulterior dezvoltarea aplicatiei în directiile propuse.

Bibliografie

[1] Blaga A., *"Dynamic HTML"*, PC - Report nr.5/1998
[2] Clip P., *"Programare - Servleturi: alternativa la CGI"*, Byte nr.5/1998
[3] *** *"Documentatie: JDK1.2, JDBC, AWT, Servlet"*
[4] Eckel B., *"Thinking in Java"*, Prentice Hall, (1998)
[5] Knuth D., *"The Art of Programming"* - The definitive encyclopedia of algorithms: Volume 1: *"Fundamental Algorithms"*, 3rd Edition, Addison-Wesley (1997); Volume 2: *"Seminumerical Algorithms"*, 3rd Edition Addison-Wesley; Volume 3 *"Sorting and Searching"* 2nd Edition, Addison-Wesley.
[6] Laffra C., *"Advanced Java, Idioms, Pitfalls, Styles, and Programming Tips"*, (Java 1.0) Chapter Sections 11-20, Prentice Hall, (1997)
[7] Nguyen J., Fraenkel M., Redpath R., Nguyen B. Q., and Singhal S. K., *"Building High-Performance Applications and Servers in Java: An Experiential Study"*, IBM Software Solutions, IBM T.J. Watson Research Center, http://www.ibm.com/java/education/javahipr.html
[8] *** *"Optimizing Java for Speed "*, http://www.cs.cmu.edu/~jch/java/speed.html
[9] Orchard D., *"Performante mai bune cu exceptiile Java"*, Byte nr.3/1998
[10] *** *"Sun's Java document page on the JDK Java interpreter"*, http://java.sun.com/products/JDK/tools/win32/java.html
[11] Tannenbaum A. S., *"Retele de calculatoare"*, C.P.Agora, (1997)