Capitolul 2
INTEGRAREA unor tehnologii informatice în tehnologia bazelor de date

Notă. Se cunosc aspectele fundamentale referitoare la SBD distribuite şi SBD orientate obiect (vezi modulul SBDE – Sisteme de Baze de Date Evoluate).

2.1. Concepte din tehnologia orientată obiect

CLASELE (TIPURILE) DE OBIECTE sunt un tip abstract de date prin care se defineşte structura obiectelor *(proprietăţile*) şi comportamentul (*metodele*) acestora.

OBIECTELE reprezintă o colecţie de proprietăţi care se referă la aceeaşi entitate.

Obiectul are:

- un nume prin care este referit ;
- un identificator unic atribuit de sistem;
- o implementare care este privată;
- o interfață care este publică.

METODA reprezintă operațiile permise asupra obiectului, deci comportamentul (funcționalitatea) acestuia.

MESAJUL reprezintă cereri adresate obiectelor pentru a returna o valoare sau o stare.

CARACTERISTICI (principii) fundamentale (de bază) ale obiectelor:

- ***încapsurarea*** : descrierea obiectelor se face astfel încât nu se poate avea acces din afara obiectului la datele sale;
- ***polimorfismul*** : diferite obiecte pot răspunde diferit la aceleaşi mesaje;
- **moştenirea** : capacitatea unui obiect de a-şi deriva datele şi funcționalitatea din alt obiect.

INSTANȚA unei clase reprezintă realizarea unei clase, dată de valorile variabilelor aferente

2.2. Programarea şi limbajele de programare

- ✓ ***Programarea*** (programming) este activitatea de scriere a programelor într-unul sau mai multe limbaje de programare respectând anumite tehnici de programare. Elaborarea programelor este activitatea care urmează după analiza şi proiectarea unei aplicații informatice cu baze de date. Programarea presupune parcurgerea următoarelor activități: întocmirea unor specificații de programare (pe baza rezultatelor de la analiză şi proiectare), întocmirea schemelor logice / pseudocodului, alegerea limbajelor de programare, scrierea codului/programului sursă, testarea şi punerea în funcțiune a programelor.

Exemplu. Scrierea codului sursă în SQL; generarea codului sursă, cu un produs specializat – Query Builder, pentru realizarea regăsirii dintr-o bază de date.
Notă. Programarea este tehnică (mijloacele folosite) şi artă (experiență, inspirație, fantezie).

- ✓ ***Tehnici de programare*** (programming techniques). Activitatea de programare nu se desfăşoară la întâmplare, ci ea presupune tehnică şi artă. *Arta* de a programa înseamnă abilitatea programatorului (experiență, fantezie, soluții) de a scrie, într-un limbaj de programare, cod sursă eficient. *Tehnica* de programare înseamnă respectarea de către programator a unor reguli, restricții, etape la scrierea diferitelor programe. *Exemple* de tehnici de programare: structurarea (secvențială, alternativă, repetitivă), modularizarea, lucrul cu subprograme, meniuri (orizontale, verticale), ferestre, videoformate, rapoarte, recursivitatea, programarea vizuală etc. Fiecare limbaj de programare are instrucțiuni care permit utilizarea anumitor tehnici de programare, în funcție de tipul de limbaj şi de facilitățile oferite de acesta.

*Exemplu.*Programarea relațională (descriptivă) presupune doar structuri secvențiale (în SQL), programarea procedurală (imperativă/algoritmică) presupune toate cele trei structuri fundamentale – secvențială, alternativă, repetitivă (PL/SQL).

- ✓ ***Limbaje de programare*** (programming language). O mulțime de cuvinte rezervate, cu o anumită semnificație, împreună cu gramatica aferentă (reguli de sintaxă şi semantică). Orice limbaj folosit pentru descrierea datelor şi a fenomenelor din lumea reală, care este înțeles de un calculator. Descrierea fenomenelor poate fi făcută algoritmic şi/sau declarativ, rezultând limbaje de programare corespunzătoare. Programatorul pleacă de la lumea reală, identifică fenomenele şi datele corespunzătoare, realizează o descriere a lor prin specificații de programare, pe care le foloseşte pentru întocmirea schemelor logice sau a pseudocodurilor aferente. În continuare, îşi alege un limbaj de programare în care scrie *programele sursă* aferente. Acestea sunt apoi "rulate" pe calculator parcurgând *fazele*: compilare (pleacă de la prgramul sursă şi produce programul obiect), link-editare (pleacă de la programul obiect şi produce programul executabil), execuție (pleacă de la programul executabil şi produce rezultate).

***Categorii* de limbaje de programare**.
În funcție de domeniul de utilizare, limbajele de programare pot fi:
1. *de asamblare* (specific fiecărui tip de calculator – ASM pentru PC);
2. *procedurale* (imperative / algoritmice):
- *universale* (se folosesc pentru orice domeniu din lumea reală – Fortran, Cobol, Basic, Pascal, C etc.)

- *specifice* / speciale (se folosesc doar în anumite domenii – exemplu, baze de date):
    - *cu limbaj gazdă* (PL/SQL)
    - *cu limbaj propriu* (limbajul din VFP)

3. *neprocedurale* (descriptive / declarative):

- limbaje relaționale (SQL, QUEL, QBE);
- limbaje orientate obiect (Smalltalk, Delphi, C++, Java);
- limbaje pentru inteligența artificială:
    - funcționale (LISP) – pentru robotică (Japonia)
    - logice (PROLOG) – pentru sisteme expert (USA)
- limbaje de simulare (Simula, Ada)

4. *de scenarii* (script – comenzi şi marcatori)

- limbaje de comandă (MS-DOS);
- limbaje de script (HTML, JavaScript, BasicScript, PHP).

Exemplu. Orice SGBD are cel puțin un limbaj de programare (exemplu SQL) cu instrucțiuni pentru *descrierea* datelor (LDD – Create, Alter etc) şi instrucțiuni pentru manipularea datelor (LMD – Insert, Update, Delete, Select etc.). Aceste limbaje permit implementarea unui model logic de date pentru BD.

Notă.

Macro >> Comenzi >> Instructiuni >> Cod maşină (assembler) >> microinstrucțiuni

✓ *Inițiere în limbajul PL/SQL*

PL/SQL este un *limbaj procedural*, propriu sistemului Oracle, care lucrează stil compilator şi care dă posibilitatea să se dezvolte structuri procedurale de program, suportând în acelaşi timp o parte dintre comenzile SQL. El a fost dezvoltat pe o structură de program Pascal.

Instrucțiunea PL/SQL are drept terminator caracterul punct şi virgulă **(;)**.

Expresiile în PL/SQL sunt formate din operatori şi operanzi. Tipul unei expresii este dat de tipul operatorilor utilizați.

Operatorii sunt de următoarele tipuri: *aritmetici* (+,-, *, /, **), *logici* (AND, OR, NOT), *de comparație* (=, !=, <, >, <=, >=), *alți operatori speciali* (LIKE; IN; BETWEEN etc.).

Operanzii pot fi variabile, constante, atribute (%TYPE, %FOUND etc.).

Unitatea de bază pentru structurarea unui program în limbajul PL/SQL este *blocul*.

Structura unui bloc este: DECLARE
Instrucțiuni de declarare (neexecutabile)
BEGIN
Instrucțiuni executabile (proprii sau din SQL)
EXCEPTION
Proceduri de tratare a erorilor (excepțiilor)
END;

Notă. Dintre secțiunile de mai sus, **doar cea executabilă (între BEGIN şi END) este obligatorie, într-un bloc PL/SQL.**

Comentariul se indică ca un text scris între **/* şi */** (analog ca în SQL) sau după două caractere liniuță de unire **(- -).**

Instrucțiunile dintr-un bloc pot fi precedate de <<*etichetă*>>, care se poate folosi apoi pentru referire.

Limbajul PL/SQL acceptă şi *subprograme* care pot fi de următoarele tipuri: *proceduri* (PROCEDURE), *funcții* (FUNCTION), *pachete* (PACKAGE), *declanşatori* (TRIGGER).

Comenzi suportate din SQL

INSERT, UPDATE, DELETE, SELECT, COMMIT, ROLLBACK, SAVEPOINT, LOCK, TABLE, SET TRANSACTION.

Instrucțiuni proprii

:= este instrucțiunea de atribuire
BEGIN desemnează începutul părții executabile a uni bloc
CLOSE închide un cursor explicit
DECLARE marchează începutul unui bloc şi a părții nexecutabile a acestuia.
Pentru a declara variabile şi constante: NUMBER, CHAR, VARCHAR, DATE, BOOLEAN, atributele (%nume)
Pentru a declara un cursor: CURSOR
Pentru a declara o excepție (tratarea erorilor): EXCEPTION

END încheie o serie de instrucțiuni (bloc – END, structură repetitivă – END LOOP, structură alternativă – END IF)
EXCEPTION marchează începutul părții de tratare a erorilor dintr-un bloc.
EXIT forțează ieşirea necondiționată dintr-o structură nesecvențială de program.
FETCH accesează următoarea linie din mulțimea selectată de un cursor explicit.
GOTO salt necondiționat la o etichetă dintr-un bloc.
IF…END IF structura alternativă simplă de program.
LOOP… END LOOP structura repetitivă de program tip WHILE sau FOR
NULL este instrucțiunea care nu are nici un efect
OPEN deschide un cursor explicit
RAISE opreşte execuția unui bloc şi transferă controlul unei secțiuni de excepție.
SELECT…INTO instrucțiune de regăsire, care întoarce o linie dintr-o tabelă şi o plasează într-o variabilă de memorie indicată prin clauza INTO.

Funcții

SQLCODE() numărul codului unei erori detectate
SQLERRM() mesajul unui cod de eroare specificat
Notă. Majoritatea funcțiilor din SQL*Plus sunt suportate şi de PL/SQL.

Exemplu

Să se scrie un bloc PL/SQL care conține o structură repetitivă. Blocul va adăuga în tabela TEMP(numcol) numere de la 1 la 10 cu excepția celor din mulțimea (6,8), iar execuția blocului se va termina la numărul 10.
REZOLVARE:

```
BEGIN
        FOR var IN 1..10   LOOP
              IF var NOT IN (6,8) THEN
                 INSERT INTO temp(numcol) VALUES (var);
              END IF;
         END  LOOP;
END;
```

Platforma JAVA

Platforma Java a fost realizată şi promovată de firma Sun Microsystems USA.

În construcția platformei s-a plecat de la realizarea limbajului de programare Java în anul 1995, ca o soluție de adaptare a limbajelor de programare universale la noul context informatic, dominat în principal de mediul Internet.

Ulterior au fost create noi produse, bazate pe tehnologia Java, toate împreună formând platforma Java actuală:

- limbajul de programare Java
- limbajul de scenarii JavaScript
- Servlets
- Java Server Pages (JSP)
- Java DataBase Connectivity (JDBC)
- Java Beans
- Enterprise Java Beans (EJB)
- Business Components for Java (BC4J)
- Java Enterprise Edition Edition (JEE)
- SQL Java (SQLJ)

Limbajul de programare Java

Evoluția limbajului de programare Java a fost următoarea:
1994 – James Gosling, pornind de la limbajul C++, crează un nou limbaj de programare numit *Oak*.
1995 – Firma Sun Microsystems sprijină dezvoltarea aplicațiilor cu limbajul Oak, pentru rularea lor cu un browser Web Runner şi îi schimbă numele limbajului în *Java*.
din 1996 – Mari firme de software (Netscape, Oracle etc.) cumpără licențe şi promovează limbajul Java, alături de celelalte produse din platforma Java.
În Oracle în 2011 este implementat *JSE 8* – Java Standard Edition care presupune: lizibilitate mai bună (caractere speciale ca separatori), acceptă constanta binară, automatizarea administrării resurselor, JVM reutilizează infrastructura de calcul (o mulțime de limbaje pot lucra pe JVM), interfața tip API pentru mecanismul de concurența şi prelucrare paralelă, modularizare masivă în platforma Java, suport pentru microprocesoare multi-nucleu (core), evoluție şi nu revoluție.

Caracteristicile limbajului Java sunt:

- *universal*: este un limbaj de programare orientat obiect, orientat pe comunicație, cu facilități procedurale, aplicabil în toate domeniile de activitate;
- *simplu*: s-a păstrat în limbaj, față de limbajele universale precedente, doar părțile strict necesare;
- *orientat obiect:* are facilități de programare orientată obiect (clase de obiecte, obiecte, proprietăți, metode, mesaje, caracteristici ale obiectelor), dar păstrează şi facilități procedurale (structura alternativă, structura repetitivă etc.);

- *familiar:* este uşor de învăţat de programatorii care ştiu deja C++. Au fost eliminate conceptele din C++ care făceau programarea greoaie (exemplu pointerii). Au fost adăugate noi concepte preluate din alte limbaje (exemplu din Smalltalk);
- *robust:* s-a mărit gradul de siguranţă al codului de program prin două niveluri de verificare – la compilare şi la execuţie;
- *comunicaţie:* programele Java pot rula nativ pe reţele de calculatoare, în sisteme distribuite;
- *dinamic:* programele pot fi actualizate la momentul execuţiei;
- *portabilitate:* programul Java poate rula pe diferite calculatoare, sub diferite sisteme de operare şi sub diferite alte medii software;
- *interconectare:* numeroase browsere Web (Netscape Navigator, Internet Explorere, Opera etc.), dar şi alte sisteme software au inclusă maşina virtuală Java (JVM – Java Virtual Machine)
- *principii Java:* citirea este mai importantă decât scrierea, limbajul nu ascunde ceea ce se întâmplă etc.

Elemente de limbaj Java

Câteva *diferenţe faţă de C++* în Java: programul este o lista de clase, nu există funcţii independente, nu există variabile externe, nu există pointeri, nu există apeluri externe, nu există caracteristica de moştenire multiplă, nu există fişiere *header* (*.h), nu există destructori, există un colector de resturi (garbagge collector) care disponibilizează automat spaţiul de memorie care nu mai este necesar, există programe care nu au funcţia main(), este posibilă tratarea excepţiilor, pot fi aplicaţii de sine stătătoare (standalone) sau *applet*-uri *(browser* cu maşina virtuala Java).

Constantele din limbajul Java există constante: *numerice* (întregi, reale)*, booleene, caracter* şi *şir de caractere.* Constantele numerice sunt *întregi* sau *reale*.

Variabila în limbajul Java este o locaţie de memorie care poate păstra o valoare de un anumit tip. Există variabile care îşi pot modifica valoarea şi variabile care nu şi-o pot modifica (numite în Java *variabile finale*). Orice variabilă trebuie să fie declarată pentru a putea fi folosită. Declaraţia unei variabile are sintaxa : <*tip*> <*nume_variabila*>;
Fiecare variabilă trebuie să aibă o *clasă de memorare*. Această clasă ne permite să aflăm care este intervalul de existenţă şi vizibilitatea variabilei în contextul execuţiei unui program.

Operatorii *aritmetici* : +, -, *, /, % (modulo, doar pentru tipurile întregi). Operatorii + şi - pot fi şi operatori semn. Operatorul + este folosit şi pentru concatenarea a doua şiruri de caractere.
Operatori de *incrementare* ++ şi *decrementare* -- sunt unari.
Operatori *logici* : && (şi), || (sau) şi operatorul ! (negaţie) au operanzi booleeni.
Operatori *pe biţi* : & (şi), | (sau), ^ (sau exclusiv) şi ~ (negaţie). Alţi operatori pe biţi sunt >> (deplasare dreapta), << (deplasare stânga), >>> (deplasare dreapta fară extensie de semn). Au operanzi întregi.
Operatorii *de comparaţie* : >, >=, <, <=, == (egalitate), != (diferit).
Operatori de *atribuire combinaţi* : +=, -=, *=, /=, &=, |=, ^=, %=, <<=, >>=, >>>=.
Operatorul de *decizie* (?:) este ternar şi are forma : *expresie1 ? expresie2: expresie3*.
Operatorul *instanceof*, care este binar, întoarce true dacă obiectul din stânga lui este o instanţă a clasei sau interfeţei specificate în dreapta şi false în caz contrar.

Tipurile de date în limbajul Java sunt: tipuri *primitive* (aritmetice, întregi, reale, caracter, logic) şi tipul *referinţă* (obiectele – instanţele claselor, tablourile).
Limbajul Java poseda pentru fiecare tip primitiv şi câte o clasă ale cărei instanţe reprezintă obiecte similare cu valorile din tipul primitiv respectiv. Aceste clase sunt : Byte, Short, Integer, Long, Float, Double, Character, Boolean.
În Java, tipul referinţă este singura modalitate de accesare a obiectelor

Comentariile pot fi pe mai multe linii, închise între /* şi */. Comentarii pe o singură linie care încep cu //.

Limbajul Java are *cuvinte rezervate.*

O *excepţie* este un semnal prin care se indica faptul că a apărut o situaţie excepţională: *excepţii propriu-zise* (depăşire de indice, adresă necunoscută etc.) şi *erori* (depăşire de stivă, depăşire de adresă etc.).

Clase şi obiecte. Un program Java este format din una sau mai multe clase. O *clasa* este şablonul care descrie entităţi de un anumit tip, precizând structura şi funcţionalitatea acestora. Declararea unei clase este similară cu declararea unui nou tip de date. *Obiectele* se mai numesc şi *instanţe* ale unei clase deoarece sunt variabile referinţa declarate de tipul unei clase.

Metodele descriu comportamentul obiectelor şi dacă nu returnează nici o valoare (cazul procedurilor din alte limbaje) se foloseşte cuvântul cheie void. Tipul metodei precum şi tipul, numărul şi ordinea parametrilor poartă numele de "semnătura metodei" (sau prototipul ei).
Punctul de intrare in orice aplicaţie Java este metoda main() care trebuie să se găsească în una (şi numai una) dintre clasele aplicaţiei. Compilatorul Java caută această metodă şi execută instrucţiunile din cadrul acesteia. Prototipul (amprenta) funcţiei main() este tot timpul acelaşi.

Constructorii din limbajul Java permite o altă modalitate de iniţializare a variabilelor membre ale unui obiect încă din momentul rezervării spaţiului in memorie. Aceştia sunt metode speciale ale unei clase care sunt apelate implicit în momentul instanţierii unui obiect al clasei. Constructorii au sintaxa obişnuita a unei metode cu deosebirea ca nu au tip iar numele lor este acelaşi cu cel al clasei.

Modificatorii de acces din Java specifică nivelul de vizibilitate al variabilelor şi metodelor în raport cu alte clase. Java are patru nivele de acces : public, private, protected şi package.

Suprascrierea metodelor înseamnă folosirea aceluiaşi nume pentru mai mult decât o metoda, ceea ce diferă fiind lista parametrilor (numărul, ordinea şi/sau tipul) şi implementarea metodelor. Acest mecanism este foarte des întâlnit în procesul de moştenire.

Caracteristicile obiectelor în Java sunt date de două mecanisme de folosire a facilitaţilor unei clase în interiorul alteia *: clasa componenta* şi *clasa care extinde (moşteneşte)* altă clasă. O clasa componentă apare atunci când în interiorul unei clase se foloseşte o variabilă de tipul altei clase. În Java fiecare clasă declarata este derivată din alta clasă

Câteva instrucţiuni Java
Instrucţiunea vidă - nu execută nimic.
Instrucţiunea de atribuire este o setare de valoare într-o locaţie de memorie (*Loc=val*).
IF – instrucţiune alternativă simplă.

SWITCH – instucțiune alternativă multiplă.
WHILE, DO, FOR - instrucțiuni de ciclare.
BREAK – ieşire forțată dintr-o buclă.
RETURN - părăsirea corpului unei metode. În cazul în care `return` este urmată de o expresie, valoarea expresiei este folosită ca valoare returnată de metodă.
THROW - semnalizează o excepție de execuție.
TRY - inițiază un context de tratare a excepțiilor.
SYNCHRONIZED - introduce o secvență critică de instrucțiuni. *O secvență critică de instrucțiuni* trebuie executată în aşa fel încât nici o altă parte a programului să nu poată afecta obiectul cu care lucrează secvența dată.

Exemple

1. *Programare structurată*: să se contorizeze numărul de elemente dintr-un vector care au valoarea 22 şi să se afişeze rezultatul obținut.

```
int contor=0;
int vect[]=tab[10];
for (int i=0; i<vect.length; i++)
{
if (vect[i]==22)
      contor++;
}
System.out.println("\n Contor="+contor);
```

2. *Programare orientată obiect*: să se construiască o rutină privind afişarea orei.

```
interface Ceas
{
      public String  Timp(int ora);
}
class Ceas1  implements Ceas
{
      public String Timp(int ora)
      {
      return new String("Ora este=");
      }
}
```

Limbajul de scenarii JavaScript

JavaScript este un limbaj de scenarii (script) dezvoltat de Netscape. Cu ajutorul limbajului JavaScript se pot creea *pagini Web interactive*.

JavaScript *seamănă* cu limbajul Java în ceea ce priveşte sintaxa expresiilor şi posedă aproape toate structurile de control de program. Spre *deosebire* de Java care este un limbaj compilat, JavaScript este executat de interpretorul navigatorului sub care rulează.
Limbajul Java este folosit, de regulă, de programatori pentru crearea de noi obiecte şi *applet*-uri, iar JavaScript este utilizat de dezvoltatorii de pagini Web, pentru scrierea de scenarii care descriu comportamentul acestor obiecte în funcție de evenimentele care apar.

Pentru a rula secvențe (scripturi) scrise în JavaScript este nevoie de un *browser* care suportă JavaScript (Netscape Navigator de la 2.0, Microsoft Internet Explorer de la 3.0).

Elemente de limbaj JavaScript

Constantele sunt de următoarele tipuri: numerice (întregi – 42, reale - 3.14159.), logice / boolean (*true, fals*), şiruri ("ok", 'ok'), neprecizat (*null*), neinițializat (*undefined*).

Caracterele speciale au următoarea semnificație: \b - Backspace, \f - Form feed, \n - New line, \r - Carriage return, \t – Tab, \' - Apostrof, \" - Ghilimele, \\ - Backslash.

Variabilele sunt identificate printr-un nume (încep cu o litera, apoi litere şi/sau cifre)
Declararea variabilelor nu este obligatorie şi nu necesită precizarea tipului. Exemple:
var i, n, pi; var stud = "Popa Anda"; var x =22;
Tipul variabilelor se precizeaza dinamic, la momentul execuției, prin instrucțiuni de atribuire. Este posibil ca pe durata execuției, unei variabile să i se modifice tipul de mai multe ori (exemplu *var x; . . . x=22; . . . x='ok';*)..

Expresiile pot fi de urmatoarele tupuri: aritmetice, logice, caracter (în funcție de tipul operatorilor).

Operatorii sunt de următoarele tipuri: aritmetici (+ – * / ++ -- %), de comparatie (> < <= >= == != ===(egalitate stricta)), logici (& ^ |), speciali (*new* – creare instanță, *void* – evaluează expresie fără a întoarce rezultat, *delete* – şterge instanță).

Instrucțiunile pot fi terminate sau nu prin caracterul punct şi virgulă (;) . Acest caracter este obligatoriu doar atunci când sunt mai multe instrucțiuni pe acelaşi rând.

`atribuirea ;`	`x=22;`
`if ( conditie ) instructiune_1 else instructiune_2`	`if ( a == undefined) document.write("A = ?")`
`for ( expr1; expr2; expr3 ) instructiune`	`for (i = 0; i < a.length; i++) { if (a[i] = theValue); break; }`

while (conditie) { statements }	**n = 0 x = 0 while (n < 22) { n ++ x += n ;}**
do { instructiuni } while (conditie)	**do { i+=1; document.write(i); } while (i<5);**
with (obiect){ instructiuni }	**var a, x, y var r=10 with (Math) { a = PI * r * r x = r * cos(PI) y = r * sin(PI/2) }**
switch (expresie){ case eticheta1 : instructiuni; break; case eticheta2: instuctiuni; break; ... default :instructiuni; }	**switch (expr1) { case "unu" : document.write("unu. "); break; case "doi" : document.write("doi. "); break; default : document.write("altceva. "); }**
function nume(parametri) { instructiuni }	**function f1() { fer1=open(„f.htm”); }**

Comentariul se desemnează prin două caractere *slash* (//) urmate de textul dorit.

Codul JavaScript poate fi introdus direct într-o secvență HTML, ca în exemplul care urmează.

```
<html>
<body>
<br>
Mesaj1 HTML.
<br>
 <script language="JavaScript">
   document.write("Mesaj2 în JavaScript")
 </script>
<br>
Mesaj3 HTML.
</body>
</html>
```

Rezultatul acestei secvențe este afişarea pe ecran a celor trei mesaje astfel: două în HTML şi unul în JavaScript.
Tot ceea ce este între marcatoarele <script> şi </script> este interpretat drept cod JavaScript, adică se foloseşte comanda (metoda) *document.write()*.

Evenimentele sunt importante în programarea JavaScript şi, de cele mai multe ori, sunt declanşate de acțiuni ale utilizatorului (dacă se apasă un buton atunci un eveniment *Click* are loc, dacă mouse-ul este deasupra unei legături atunci un eveniment *MouseOver* are loc).
Pentru ca un program JavaScript să reacționeze la anumite evenimente se utilizează "event-handlers" – *gestionarul de evenimente*. De exemplu, un buton poate crea o fereastră atunci când este apăsat. Aceasta înseamnă că fereastra apare ca o reacție la evenimentul *Click,* iar în acest caz gestionarul de evenimente care trebuie utilizat este numit *onClick*.

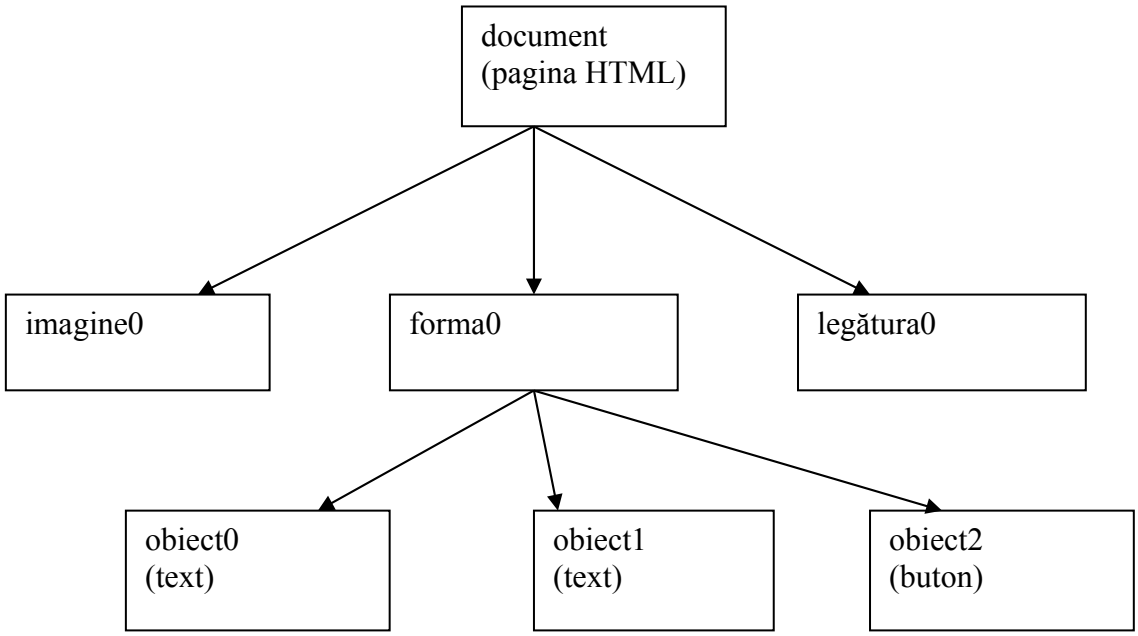
```
<form>
<input type="button" value="Apasa" onClick="alert('Yo')">
</form>
```

JavaScript organizează toate elementele unei pagini de Web într-o **ierarhie**. Toate elementele sunt văzute ca *obiecte,* care au anumite proprietăți şi metode. Câteva *tipuri de obiecte* dintr-o pagină JavaScript sunt: ferestre, documente, locații.
Exemplu să realizăm o pagină HTML cu următoarele obiecte: o legătură, o imagine, o formă (cu două câmpuri text şi un buton).

```
<html>

<head>
<title>Pagina test    // Legătură
</head>

<body>
<center>
<img src="home.gif" name="pic1" width=200 height=100>   //  Imagine
</center>
<p>
<form name="Forma1">  //  Formă cu trei obiecte
Nume:
<input type="text" name="name" value=""><br>
E-mail:
<input type="text" name="email" value=""><br><br>
<input type="button" value="Apasă" name="Buton1" onClick="alert('Yo')">
</form>

</body>
</html>
```

Comentariu.
Din punctul de vedere JavaScript fereastra browserului este un obiect tip fereastră (*window)*. În interiorul ferestrei se poate încărca un document HTML (sau un fişier de alt tip). Această pagină este un obiect tip *document*. Aceasta înseamnă că obiectul document este chiar pagina HTML încărcată la un anumit moment dat. Toate obiectele HTML sunt proprietăți ale obiectului document (un obiect HTML este, de exemplu, o legătura sau o formă).

Urmatoarea imagine ilustreaza ierarhia creată de pagină construită mai sus, notarea nodurilor facându-se de la zero:



Dacă se doreşte adresarea imaginii din pagina HTML trebuie studiată ierarhia şi se începe de sus. Primul obiect este denumit *document*. Imaginea este reprezentată prin *imagine0*. Aceasta înseamnă că putem accesa acest obiect prin intermediul JavaScript cu ajutorul *document.images[0]*.
Dacă de exemplu dorim să accesăm primul obiect din formă atunci referirea se va face prin *document.forms[0].elements[0]*

Pe lângă obiectele tip fereastră şi document mai există şi *obiectul locație (location*). Acesta reprezintă adresa documetului HTML încărcat. Deci, dacă se încarcă pagina *http://www.xyz.com/page.html* atunci *location.href* este egală cu această adresă. *Exemplu:* un buton încarcă o nouă pagină de pe Yahoo în fereastra de browser.

```
<form>
<input type=”button”  value="Yahoo"
  onClick="location.href='http://www.yahoo.com'; ">
</form>
```

Cadrele (frame) reprezintă modul de împărțire a ferestrei unui browser în mai multe zone. Fiecare cadru prezintă în interior un document propriu (de cele mai multe ori document HTML). Pentru crearea de cadre se folosesc doi marcatori: <frameset> şi <frame>.
Exemplu. O pagină HTML care crează două cadre poate fi obținută prin secvența următoare.

```
<html>
<frameset rows="50%,50%">
  <frame src="page1.htm" name="frame1">
  <frame src="page2.htm" name="frame2">
</frameset>
```

```
</html>
```

Fiecare cadru are un nume unic specificat cu ajutorul proprietății *name* în marcatorul <frame>. Aceasta va ajuta pentru accesarea cadrelor cu ajutorul JavaScript.

Construirea ferestrelor. Deschiderea unei noi ferestre de browser este una dintre facilitățile JavaScript: fie se poate încărca un document (de exemplu un document HTML) în noua fereastră, fie se poate *crea* un nou document.

1. *Încărcarea unui document* în fereastră.

Următorul script deschide o nouă fereastră browser (prin metoda *open()*) şi încarcă o pagină (document) existentă dintr-un fişier.

```
<html>

<head>
<script language="JavaScript">

function fer() {
  fer1= open("f.htm");
}

</script>
</head>

<body>

<form>
<input type="button" value="Deschide fereastra " onClick="fer()">
</form>

</body>
</html>
```

.htm este încărcat în noua fereastră prin intermediul metodei *open()*.

Proprietățile unei ferestre ajută la poziționarea ei pe ecran şi la stabilirea formei şi conținutului ei (height, location etc.).

Exemplu:

```
fer1=open("f.htm","fereastra","width=400,heght=300,status=no,toolbar=no,menubar=no");
```

Închiderea unei ferestre se realizează prin metoda *close()*.

---

2. *Crearea de noi documente* într-o fereastră existentă

Exemplu: se crează un document simplu HTML care va fi afişat într-o nouă fereastră existentă

```
<html>

<head>
<script language="JavaScript">
```

```
<!-- hide

function fer1() {
  fer= open("", "afisez fereastra",
    "width=500,height=400,status=yes,toolbar=yes,menubar=yes");

fer.document.open();

  // creez documentul
  fer.document.write("<html><head><title>Titlu1 ");
  fer.document.write("</title></head><body>");
  fer.document.write("<center><font size=+3>");
  fer.document.write(" HTML-documentul a fost creat cu JavaScript ");
  fer.document.write("</font></center>");
  fer.document.write("</body></html>");

fer.document.close();
}

// -->
</script>
</head>

<body>

<form>
<input type=button value="On-the-fly" onClick="fer1 ()">
</form>

</body>
</html>
```

Comentariu. Mai întâi se deschide o nouă fereastră browser. Primul argument este un sir gol (*″ ″*), - aceasta înseamnă că browserul nu trebuie să aducă un document, pentru că - JavaScript va crea noul document.
Pentru început se defineşte variabila *fer*. Cu ajutorul acestei variabile se poate accesa noua fereastra.
Dupa ce s-a deschis fereatra se poate deschide şi documentul, pentru a fi pregătit apoi să primească intrările (conţinutul documentului) prin comenzile *fer.document.write()*.
În final se închide documentul.

Obiectele predefinite sunt acceptate de JavaScript, de exemplu: Date (data calendaristică) şi Array (masive).

Obiectul dată calendaristică (Date) permite lucrul cu data şi timpul.
Exemplu pentru afişarea datei şi orei curente trebuie creat mai întâi un nou obiect de tip Date: *data1= new Date()*
Obiectul tip *Date* oferă câteva metode care pot fi folosite: *getHours(), setHours(), getMinutes(), setMinutes(), getMonth(), setMonth()* etc.

Obiectul masive (Array) este acceptat începând cu JavaScript 1.1 (Netscape Navigator 3.0). Se poate crea un nou masiv prin: *masiv1= new Array()*. Se pot da valori acestui masiv prin: *masiv1[0]= 2; masiv[1]= "Anda"; masiv[2]= "Maria";*
Masivele din JavaScript sunt flexibile: dimensiunea lor poate creşte dar nu are abilitatea de a se diminua.

Straturile (layers) sau nivele sunt facilități apărute începând cu Netscape Navigator 4.0. Ele permit poziționarea absolută a obiectelor pe ecran, adică obiectele pot fi mişcate sau mutate în pagină şi, de asemenea, obiectele pot fi suprapuse în pagină.

Pentru *crearea unui strat* trebuie folosit fie marcatorul <layer> fie <ilayer>, care au o serie de proprietăți (left, top, width, z-index etc.).
Marcatorul <layer> este utilizat pentru straturile care pot fi poziționate în mod clar. Dacă nu specificați o poziție (cu ajutorul proprietăților *left* si *top*) stratul va fi poziționat în colțul stânga sus al ferestrei.
Marcatorul <ilayer> creeaza un strat al cărui poziție depinde de modul de afişare al documentului.
Exemplu: Se doreşte creearea a două straturi. În primul strat se creează o imagine (z-index=0), iar în cel de-al doilea un text (z-index=1). Se doreşte afşarea textului deasupra imaginii.

```
<html>

<layer name=pic z-index=0 left=200 top=100>
<img src="img.gif" width=160 height=120>
</layer>

<layer name=txt z-index=1 left=200 top=100>
<font size=+4> <i> Exemplu</i> </font>
</layer>

</html>
```

Identificarea straturilor în JavaScript se face printr-un nume care i se atribuie (<layer ... name=strat1 >). Ulterior, stratul se poate accesa prin *document.layers["strat1"]*. Straturile se pot accesa şi prin intermediul unui index număr întreg. Pentru a accesa stratul cel mai de jos se scrie *document.layers[0]*.

Există câteva *proprietăți* ale unui strat care pot fi schimbate prin intermediul JavaScript. Exemplul următor prezintă un buton care permite ascuderea sau vizualizarea unui strat. În acest sens, se crează o funcție (av) care va fi apelată la apăsarea unui buton.

```
<html>
<head>
<script language="JavaScript">

function av() {
  if (document.layers["strat1"].visibility == "show")
    document.layers["strat1"].visibility= "hide"
  else document.layers["strat1"].visibility= "show";
}
```

```
</script>
</head>
<body>

<ilayer name=strat1 visibility=show>
<font size=+1><i>Text in strat </i></font>
</ilayer>

<form>
<input type="button" value="Vizualizare/Ascundere" onClick="ave()">
</form>

</body>
</html>
```

Exemplu

Să se scrie codul sursă JavaScript pentru deschiderea într-o pagină Web a unui document tip *html.*

```
<head>
<script language="JavaScript">
function fer()
{
fer1=open("ExHTML1.htm");
}
fer();
</script>
</head>
```

Servlets

Servlets sunt clase (module script) Java, necesare pentru construirea unor pagini Web dinamice, încărcate şi menţinute rezident pe server. Se extinde astfel, funcţionalitatea unui server cu un *mic server Web dedicat.*

Tehnologia *Servlets* este o *alternativă oferită de Java* pentru rezolvarea problemelor legate de timp apărute odată cu programarea *CGI* (Common Gateway Interface). Aceasta facilitează dezvoltarea unor module care permit serverelor Web să se conecteze şi să prelucreze informaţia în mod dinamic, adică să ruleze aplicaţii Web şi nu doar să transfere documente statice. Soluţia Java, menţine executabilul persistent pe server, între cererile clienţilor, spre deosebire de CGI unde fiecare cerere client lansează un nou proces pe server (ceea ce duce rapid la epuizarea resurselor serverului Web, adică la creşterea timpului).

Servlet este o componentă software în partea serverului, scrisă în JAVA şi care extinde dinamic funcţionalitatea unui server (de obicei un server HTTP). Spre deosebire de applet-uri, servlet-urile nu afişează o interfaţă grafică către utilizator, ci se returnează doar nişte rezultate către client (de obicei sub forma unui program HTML).

Servlets sunt clase JAVA care se conformează unei interfeţe specifice ce poate fi apelată de către server. Funcţionalitatea furnizată de un servlet nu este restricţionată serverelor WEB, ci se poate extinde la orice server care suportă JAVA şi Servlet API (cum ar fi: FTP, Telnet, Mail etc).

Servlets furnizează un cadru pentru crearea de aplicaţii care implementează paradigma cerere/răspuns. De exemplu, când un browser trimite o cerere către server, serverul o trimite mai departe unui servlet. Modulele Servlet procesează cererea (eventual accesând o bază de date) şi construieşte un răspuns convenabil (de obicei în HTML) care este returnat clientului.

Avantajele *servlet* faţă de alte tehnologii similare (exemplu CGI):

- **performanţa**. Modulele Servlet rulează de obicei în acelaşi spaţiu de proces ca şi serverul şi sunt încarcate doar o dată. Astfel, ele sunt capabile să răspundă mai rapid şi eficient la cererile clienţilor. În contrast, CGI trebuie să creeze câte un nou proces pentru deservirea unei cereri.
- **compilarea codului JAVA** oferă execuţii rapide faţă de programele scrise în limbaje script. Multe din erori sunt îndepărtate încă de la compilare, deci modulele servlet sunt mai stabile.
- **rezistenţe la blocarea sistemului**. Maşina virtuală JAVA nu permite modulelor servlet accesul direct la locaţiile de memorie, deci se elimină posibilitatea blocării sistemului ca rezultat al accesului la memoria invalidă (pointerii în C puteau conduce la acest lucru). JVM va propaga o excepţie sau va rezolva eroarea fără blocarea sistemului.
- **portabilitatea platformei**. Caracteristica JAVA "scris o dată, rulează oriunde" permite servlet-urilor să fie distribuite uşor fără a rescrie codul pentru fiecare platformă. Servlet-urile operează identic fără modificări când rulează pe UNIX, Windows sau alt sistem de operare.
- **durabilitate**. Servlet-urile rămân în memorie până când există o instrucţiune specifică de ştergere a lor. Astfel, acestea necesită doar o instanţiere pentru a satisface

mai multe cereri. De exemplu, se obişnuieşte ca la încărcarea unui servlet să se creeze şi mai multe conexiuni la baze de date.

- **încărcare dinamică**. Ca şi modulele applet, cele servlet pot fi dinamic încărcate local sau prin rețea. Încărcarea dinamică asigură faptul că modulele servlet nefolosite nu consumă resurse.
- **extensibilitatea**. Noile instrumente de dezvoltare, noile biblioteci de clase JAVA şi driverele de baze de date sunt disponibile şi astfel pot fi utilizate de servlet-uri.
- **concurența**. Spre deosebire de C/C++, mecanismele de concurență (thread) sunt moştenite în JAVA şi sintaxa sincronizării lor este consistentă pe toate platformele.
- **orientarea obiect**. Servlet-urile furnizează o arhitectură simplă pentru dezvoltarea aplicațiilor de rețea. Asta deoarece Servlet API încapsulează toate informațiile esențiale şi funcționalitatea în obiecte bine construite. De exemplu, Servlet API furnizează clase care lucrează cu obiecte abstracte, cum ar fi: cereri, răspunsuri, sesiuni .
- **independența de protocol**. De obicei, modulele servlet sunt folosite pentru extinderea funcționalității serverelor HTTP. Modulele Servlet sunt complet independente de protocol, acestea suportând comenzi FTP, SMTP, POP3, sesiuni Telnet, sau orice alt protocol (fie standard sau creat de programatori).
- **securitatea**. Deoarece servlet-urile sunt scrise în JAVA, nu sunt posibile accesele nevalide la memorie sau violări de tip. În plus, există *trei proprietăți* care fac modulele servlet mai sigure decât CGI. 1) Servlet folosesc un manager de securitate pentru crearea unor reguli de securitate. Un manager de securitate poate restricționa rețeaua sau accesul la un fişier pentru un servlet de neîncredere. 2) Un manager de securitate poate acorda drepturi depline pentru un servlet de încredere. 3) Un servlet are acces la toate informațiile conținute în fiecare cerere a clientului. Această informație conține date de autentificare HTTP. Când se folosesc în conjuncție cu protocoalele de securitate cum ar fi SSL, modulele servlet pot verifica identitatea fiecarui client.

Funcționalitatea *servlet* permit interacțiune în ambele sensuri între client şi server :

- **Construiesc** dinamic şi returnează un fişier HTML pe baza cererii clientului;
- **Procesează** intrarea utilizatorilor trimisă de un formular HTML şi returnează un răspuns apropiat;
- **Facilitează** comunicațiile dintre grupuri de utilizatori prin publicarea de informații trimise de mai multi clienți;
- **Furnizează** autentificarea utilizatorului şi alte mecanisme de securitate;
- **Interacționează** cu o serie de resurse ale serverului cum ar fi bazele de date şi fişierele cu informații utile pentru client;
- **Procesează** intrările de la mai mulți clienți pentru aplicații cum ar fi jocuri în rețea;
- **Permite** serverului să comunice cu un *applet* client printr-un protocol specific şi păstrează conexiunea în timpul conversației;
- **Ataşează** automat elemente de proiectare pentru pagini Web, cum ar fi antete sau note de subsol, pentru toate paginile returnate de server;
- **Întoarce** cereri de la un server la altul în scop de echilibrare a încărcării;
- **Partiționează** serviciul logic între servere sau între servlet-uri pentru a procesa eficient o problemă.

Structura de bază a unui servlet.

Cel mai uşor mod de definire al modulelor servlet este prin *extinderea* uneia din cele două clase de bază: *GenericServlet* şi *HttpServlet*. De fapt, nu este obligatoriu moştenirea acestor clase, ci este suficientă implementarea interfeței Servlet.

Toate modulele servlet *suprascriu* cel puțin o metodă necesară funcționalității lor. Metoda care este automat apelată ca răspuns la cererea fiecărui client se numeşte *service()*. Această metodă poate fi suprascrisă pentru a furniza o funcționalitate implicită. Cu toate acestea, servlet-urile care extind HttpServlet pot să nu suprascrie metoda *service()*. În acest caz, implementarea implicită a acestei metode va apela automat altă metodă pentru a răspunde cererii clientului.

Mai există încă două metode implementate de majoritatea modulelor servlet: *init()* şi *destroy()*. Metoda init() se apelează o singură dată, când este încărcat modulul servlet (similar cu un constructor). Metoda destroy() este apelată când modulul servlet este descărcat, eliberând resursele acestuia.

Un *schelet* pentru servlet poate fi (fără a preciza parametrii şi excepțiile):

```
public class ScheletServlet extends HttpServlet
{
  public void init()
  {
    // aici este codul de inţializare
  }
  public void service()
  {
    // partea  de lucru
  }
  public void destroy()
  {
    // eliberarea resurselor
  }
} // sfârşitul clasei ScheletServlet
```

Dacă modulul servlet extinde HttpServlet, dezvoltatorul servletului poate să nu suprascrie metoda *service(),* ci să implementeze altă metodă care va fi automat apelată de metoda mostenită. Metoda automat apelată de *service()* depinde de tipul cererii HTTP (de exemplu, *doGet()* este apelată pentru cererile GET si *doPost()* este apelată pentru cererile POST).

Ciclul de viață a unui servlet. Procesul apelării (de către server) a unui servlet se poate împărți în opt paşi:
1. Serverul *încarcă* modulul servlet când acesta este cerut de client sau la pornirea serverului dacă aşa impune configurația. Servlet poate fi încarcat local sau dintr-o locație de la distanță folosind o facilitate de încărcare a claselor JAVA.

2. Serverul *crează* o instanță a clasei servlet pentru deservirea tuturor cererilor. Folosind fire de execuție multiple, cererile concurente pot fi deservite de o singură instanță a modulului servlet. (Servlet s = (Servlet) c.newInstance();).

3. Serverul *apelează* metoda *init()* a modulului servlet, care garantează că termină execuția înaintea procesării primei cereri pentru servlet. Dacă serverul crează mai multe instanțe pentru servlet, metoda *init()* se apelează de fiecare dată pentru fiecare instanță.

4. Serverul *construieşte* un obiect, atunci când se primeşte o cerere pentru servlet, ServletRequest sau HttpServletRequest din datele incluse în cererea clientului. De asemenea, acesta construieşte un obiect ServletResponse sau HttpServletResponse care furnizează metode pentru returnarea răspunsului. Tipul parametrului depinde dacă modulul servlet extinde GenericServlet sau HttpServlet.

5. Serverul *apelează* metoda *service()* (care pentru servleturi HTTP poate apela o metodă specifică cum ar fi doGet() sau doPost()), trimițând ca parametri obiectele construite la pasul anterior. Când sosesc cereri concurente, metodele *service()* pot rula simultan în fire de execuție separate (cu excepția faptului când servlet-ul implementează interfața SingleThreadModel).

6. Metoda *service() procesează* cererea clientului prin evaluarea obiectului ServletRequest sau HttpServletRequest. Apoi acesta răspunde folosind obiectul ServletResponse sau HttpServletResponse.

7. Dacă serverul primeşte încă o cerere pentru acest servlet, *procesul începe din nou* la pasul cu numărul cinci.

8. De fiecare dată când containerul modulului servlet determină că un servlet *trebuie descărcat* (din motive de corecție sau dacă acesta a "căzut"), serverul apelează metoda *destroy()* după terminarea firelor de execuție ale metodei *service()* sau după terminarea limitei de timp definită de server.

Caracteristicile principale ale tehnologiei Servlets sunt:

- când micul server Web dedicat (servlet) este încărcat, el este *inițiat* prin apelul unei metode;
- la inițiere se pot deschide *conexiuni* la baze de date care devin astfel rezidente între apelurile clienților;
- clasele şi metodele necesare pentru a defini şi utiliza un server Web dedicat sunt *încapsulate* în pachete Java (exemplu Javax.servlet);
- tehnologia foloseşte *protocolul HTTP* (Hyper Text Transfer Protocol);
- tehnologia se utilizează pentru a dezvolta *soluții bazate pe Web:*
  - accesul securizat la pagini Web;
  - asigurarea interacțiunii cu bazele de date;
  - generarea dinamică paginilor HTML (Hyper Text Markup Language);
  - manipularea informațiilor care identifică unic un client pe parcursul uneia sau mai multor sesiuni.
- orice modul CGI poate fi *rescris* cu tehnologia *servlet;*
- se poate *crea* câte un mic server dedicat (servlet) pentru fiecare funcție din paginile Web (conectare, înregistrare, actualizare etc.) sau un singur server dedicat care să gestioneze toate tranzacțiile din paginile Web respective, în mod dinamic;
- tehnologia este o *soluție optimă* pentru aplicațiile care utilizează intensiv bazele de date (serverul se ocupă de accesul la date iar clienții formulează cererile de regăsire). Partea de

cod se scrie o singură dată şi se stochează rezident, o singură dată, pe server. La actualizarea codului se va face o singură înlocuire, pe server, şi nu la fiecare utilizator în parte;

- *comunicarea* client-server se face astfel: clientul formulează şi trimite către server o cerere Web; serverul o direcționează către serverul dedicat (servlet) pentru a fi procesată (ceea ce implică de multe ori şi accesul la o bază de date); răspunsul (sub formă de pagini HTML, imagini etc.) este returnat serverului şi apoi clientului care a formulat cererea;
- este permisă *memorarea urmei* (trace) clienților pentru a avea informații completedespre utilizatorul respectiv;
- un server dedicat (servlet) poate fi *apelat* dintr-o pagină HTML (de exemplu printr-un navigator – *browser*) sau dintr-un mic client Web dedicat (applet). **Applet** este un modul (clase) Java construit şi stocat pe un client. La execuția lui se poate face apel la un server dedicat Web de tip *servlet*;
- domenii de *utilizare* ale tehnologiei:
  - pentru generarea unor pagini Web dinamice;
  - în aplicații multinivel (multi-tier) cu JDBC (Java DataBase Connectivity): serverul dedicat poate accesa o varietate de baze de date prin intermediul JDBC şi poate realiza, parțial sau total, interfața cu utilizatorul prin pagini Web dinamice.

Notă. Prin extinderea micilor servere Web dedicate (servlets) s-a ajuns la tehnologia JSP.

Java Server Pages (JSP)

JSP este o tehnologie, din platforma Java, utilizată pentru construirea unor aplicații care să conțină pagini Web cu un conținut dinamic (HTML, DHTML, XML).

Tehnologia a fost creată la firma SUN, în 1999, pentru a oferi proiectanților de aplicații o interfață utilă pentru *construirea aspectelor estetice* ale paginilor Web (amplasare, aspect etc.) şi mai puțin utilă în ceea ce priveşte sursa şi manipularea datelor dinamice.

JSP a rezultat ca tehnologie, prin *integrarea experienței* unor producători de software de: servere Web, servere de aplicații, sisteme tranzacționale, interfețe de dezvoltare.

Procesul dezvoltării de pagini de Web dinamice este accelerat de către JSP din următoarele *considerente*:

- *Separarea* generării conținutului de prezentare
- *Proiectanții de pagini* folosesc marcatori (tags) obişnuiți din HTML sau XML pentru formatarea rezultatului şi marcatori JSP pentru generarea conținutului dinamic al paginii. Logica ce stă în spatele generării conținutului este cuprinsă în marcatori şi componente JavaBean, legătura dintre acestea făcându-se în secvențe (scriplets) şi totul fiind executat pe server. Astfel, proiectanții de pagini pot edita şi lucra cu pagini JSP fără a afecta generarea conținutului dinamic.
- *Pe partea de server, un motor* (nucleu) JSP interpretează secvențele (scriplets) şi marcatorii JSP, generează conținutul cerut (accesând componente JavaBean, baze de date folosind JDBC sau prin includerea de fişiere) şi trimite rezultatele înapoi sub forma unei pagini HTML (sau XML) către browser.
- Permite *reutilizarea componentelor* precum JavaBeans, Enterprise JavaBeans sau a *marcatorilor*, atât independent, cât şi în cadrul unor interfețe interactive de dezvoltare a componentelor şi paginilor de Web. Dezvoltatorii de pagini Web nu sunt întotdeauna programatori familiarizați cu limbaje de scenarii. JSP încapsulează funcționalitățile necesare pentru crearea de conținut dinamic în marcatori de tip XML specifice JSP. Marcatorii JSP standard pot accesa şi instanția componente JavaBean, pot seta sau obține atribute ale acestor componente, pot f descărca (download) applet-uri şi pot executa funcții ce ar fi dificil de implementat.
- Tehnologia JSP este *extensibilă* prin dezvoltarea unor biblioteci de marcatori definite de utilizator, pentru funcțiile folosite cel mai frecvent.
- Tehnologia JSP este complet *independentă de platformă* atât în ceea ce priveşte paginile de Web dinamice, cât şi serverele de Web şi componentele acestora. Aceasta este explicabil deoarece limbajul de script pentru paginile JSP se bazează pe Java şi în special pe modul de manipulare a obiectelor în acest limbaj.

Structura unei pagini JSP. O pagină JSP (*.jsp) este o pagină HTML sau XML ce cuprinde elemente adiționale (marcatori, declarații, secvențe (scriplet)) pe care motorul JSP le procesează returnând o pagină standard HTML/XML. Ea corespunde unui *document* ce descrie procesarea unei cereri pentru a crea un răspuns.

O pagină JSP cuprinde în structura sa următoarele **componente**:

- *cod HTML/XML standard* - cod ce rămâne neinterpretat de motorul JSP;
- *directive JSP* - directive ce furnizează informații globale independente conceptual de o anumită cerere adresată paginii JSP;
- *marcatori (tags) JSP* - spre deosebire de directive, marcatorii depind de fiecare cerere în parte adresată paginii JSP;
- *elemente de scripting* - acestea putând fi: declarații, secvențe (scriplet) şi expresii.

O pagină JSP poate *crea şi/sau accesa*, la procesarea unei cereri, anumite **obiecte Java**. Obiectele astfel create pot deveni vizibile prin variabile în limbajul de scripting. Acestea vor conține, în timpul etapei de procesare a cererilor, referințe către obiectul respectiv. Obiectele create au un atribut numit *scope* ce defineşte domeniul de vizibilitate al acestora: când există o referință la acest obiect şi când aceasta va fi înlăturată.
Valorile pe care le poate avea atributul *scope* sunt:

- *page* - accesibil doar în cadrul paginii în care a fost creat obiectul;
- *request* - accesibil din paginile ce procesează aceeaşi cerere în care a fost creat obiectul;
- *session* - accesibil din paginile ce se sunt în aceeaşi sesiune în care a fost creat obiectul;
- *application* - accesibil din paginile ce procesează aceeaşi aplicație în care a fost creat obiectul. Toate referințele la acesta sunt eliberate când mediul de lucru *runtime* solicită ServletContext.

Numele ataşat unui obiect este unic pe tot timpul execuției, toate domeniile de vizibilitate comportându-se ca unul singur în cadrul unei secvențe cerere/ răspuns. Lipsa transmiterii variabilelor de stare prin HTTP este suplinită în mod automat de către motorul JSP prin cele două *modalități* cunoscute: *cookie*, respectiv *rescrierea* URL. Cât timp sunt făcute cereri procesate de către motorul JSP, rescrierea URL este făcută în mod automat.

Orice pagină JSP conține o serie de **obiecte create implicit**:

- *request* - cererea ce a solicitat pagina respectivă;
- *response* - răspunsul la cerere;
- *pageContext* - contextul paginii curente;
- *session* - obiect de tip sesiune pentru clientul solicitat (valabil doar pentru HTTP);
- *application* - contextul servlet-ului generat (getServletConfig().get Context();
- *config* - obiect de tip ServletConfig pentru pagina curentă;
- *page* - instanță a clasei create pentru pagina curentă (pentru Java: this);
- *exception* - excepția declanşată anterior putând fi folosită doar în pagina de eroare invocată;
- *config* - obiect de tip JspWriter ce scrie în şirul de ieşire.

Tipuri de aplicații **pentru JSP**

1. *Modelul de aplicație flexibilă folosind servlet-uri Java*

Un client Web poate face o *cerere direct către un servlet* Java, care generează conținutul dinamic, stochează rezultatul într-o componentă Bean şi invocă pagina JSP. Pagina JSP accesează conținutul dinamic din componenta Bean şi trimite răspunsul (HTML) browser-ului.

2. *Modelul de aplicație pe două straturi (two-tier)*

Navigatorul invocă în mod *direct pagina JSP* şi aceasta generează conţinutul solicitat (apelând eventual la JDBC pentru a obţine informaţia direct dintr-o baza de date). Pagina JSP poate apela JDBC sau o componentă JavaBean pentru a genera rezultatele dorite şi a crea HTML standard ce va fi trimis browser-ului.
Avantajele acestei abordări sunt uşurinţa de a programa şi posibilitatea autorului paginii de a genera conţinut dinamic bazat pe cererea clientului şi starea resursei solicitate

3. Modelul de aplicaţie pe mai multe straturi (N-tier)

Pagina JSP poate acţiona ca strat de mijloc (middle-tier) în cadrul unei arhitecturi de tip Enterprise JavaBeans (EJB). În acest caz, pagina JSP interacţionează cu anumite resurse aflate pe server printr-o componentă de tip Enterprise JavaBean. Aceasta controlează accesul la resursele de pe server, ceea ce furnizează performanţă scalabilă pentru un număr mare de utilizatori concurenţi. Pentru comerţ electronic sau alte tipuri de aplicaţii, EJB controlează tranzacţiile şi problemele de securitate aferente. Acest lucru simplifică pagina JSP. Modelul acesta va fi suportat de către platforma Java Enterprise Edition (JEE)

Cracteristicile principale ale tehnologiei JSP sunt:

- poate funcţiona pe *o multitudine* de servere;
- *separă* logica (conţinutul) aplicaţiei de aspectul (prezentarea) paginilor Web. Logica aplicaţiei se construieşte cu alte tehnologii (de exemplu: JDBC, JavaBean etc.) urmând ca prin JSP să se editeze aspectul paginilor Web din aplicaţie;
- permite *reutilizarea* unor componente generate cu alte tehnologii (de exemplu cu JavaBean) atât independent cât şi în cadrul unor interfeţe de dezvoltare a paginilor Web;
- este *independentă de* platforma pe care rulează, deoarece scripturile generate de JSP se bazează pe Java;
- pagina JSP *se interpune* între navigatorul (browser) clientului şi componentele pentru logica aplicaţiei (realizate în JavaBean, CORBA, JDBC etc.) care accesează o bază de date;
- în pagina JSP *se descrie* cum trebuie sa se creeze un obiect de răspuns pentru un anumit protocol de comunicaţie (HTTP etc.);
- tehnologia JSP se poate utiliza în *aplicaţii tip* multinivel (multi-tier);
- o pagină JSP *poate indica* modul de manipulare a unor evenimente;
- *execuţia* unei pagini JSP este realizată de către motorul JSP instalat pe un server de Web. Acesta procesează pagina JSP (este o pagină HTML cu elemente adiţionale) şi returnează o pagină standard HTML/XML;
- *implementarea* tehnologiei JSP se face în două etape:
  - traducerea, care se efectuează o singură dată şi are ca efect generarea unui mic server dedicat (servlet);
  - procesarea fiecăreicereri în parte de către micul server dedicat (servlet);
- o pagină JSP poate *conţine* următoarele elemente: marcatori (tags) HTML, marcatori de date, marcatori JSP, HTML predefinit, cod Java etc.

*Notă.*O altă tehnologie, similară cu cea de mai sus, este *ASP (Active Server Pages)* realizată de Microsoft în 1996. Aceasta însă: rulează doar pe platforma Windows, poate lucra doar cu SGBD care suportă ODBC (nu şi JDBC) etc.

Java DataBase Connectivity (JDBC)

JDBC *este un standard* realizat de *Sun Microsystems* pentru conectarea unei BD cu alte sisteme. El este o dezvoltare a tehnologiei ODBC (Open DataBase Connectivity) realizată de Microsoft, ţinând cont de cerinţele mediului Internet: interoperabilitate, integrare, deschidere, comunicare, portabilitate.

JDBC este constituită dintr-un set de clase şi interfeţe scrise în Java, furnizând mecanisme standard pentru proiectanţii aplicaţiilor de baze de date. Pachetul care oferă suport pentru lucrul cu baze de date este *java.sql*.

Folosind JDBC se pot *transmite cerei de regăsire* (secvenţe SQL) către baze de date relaţionale. Cu alte cuvinte, nu este necesar să scriem o secvenţă de comenzi pentru a accesa o baza de date Oracle, o altă secvenţă pentru a accesa o bază de date Sybase şi aşa mai departe. Este de ajuns să scriem o singură secvenţă de comenzi folosind interfaţa JDBC şi acesta va fi capabilă să trimită secvenţe SQL bazei de date dorite. Scriind codul sursă în Java, ne este asigurată portabilitatea programului. Tot ceea ce-i lipseşte este modalitatea prin care aplicaţiile Java pot comunica cu bazele de date. Aici intervine JDBC-ul care oferă acest mecanism.

Sintetic***, rolul*** intefeţei , JDBCeste:
1. stabileşte o conexiune (conectarea) cu o bază de date;
2. trimite secvenţe de comenzi SQL spre o bază de date relaţională;
3. prelucrează rezultatele

1.Conectarea la o bază de date Procesul de conectare la o bază de date implică *două operaţii*: încărcarea în memorie a unui driver corespunzător (JDBC), realizarea unei conexiuni propriu-zise

O *conexiune (sesiune)* la o bază de date reprezintă un context prin care sunt trimise secvenţe SQL şi primite rezultate. Într-o aplicaţie pot exista mai multe conexiuni simultan la baze de date diferite sau la aceeaşi bază.
Clasele şi interfeţele necesare pentru realizarea unei conexiuni sunt:
- clasa **DriverManager**, se ocupă cu înregistrarea driverelor ce vor fi folosite în aplicaţie;
- interfata **Driver**, trebuie să o implementeze orice clasă ce descrie un driver ;
- clasa **DriverPropertyInfo**, descrie proprietăţile driverului;
- interfaţa **Connection**, descrie obiectele ce modelează o conexiune propriu-zisă cu baza de date

Încărcarea în memorie a driver-ului necesar comunicării este primul lucru pe care trebuie să-l faca o aplicaţie în procesul de conectare la o bază de date. Acest lucru poate fi realizat prin mai multe *modalităţi:*

1. DriverManager.registerDriver(new sun.jdbc.odbc.JdbcOdbcDriver());
2. Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
3. System.setProperty("jdbc.drivers", "sun.jdbc.odbc.JdbcOdbcDriver");
4. java -Djdbc.drivers=sun.jdbc.odbc.JdbcOdbcDriver

Specificarea unei baze de date. O dată ce un driver JDBC a fost încărcat în memorie cu `DriverManager`, acesta poate fi folosit la stabilirea unei conexiuni cu o bază de date. Având în vedere faptul că pot exista mai multe drivere înregistrate în memorie, trebuie sa avem posibilitea de a specifica pe lânga identificatorul bazei de date şi driverul ce trebuie folosit la un moment dat. Acest lucru se realizează prin intermediul unei adrese specifice, numita JDBC URL, ce are următorul format**: `jdbc:sub-protocol:identificator_baza_de_date`**
Câmpul *sub-protocol* denumeşte tipul de driver ce trebuie folosit pentru realizarea conexiunii şi poate fi *odbc, oracle, sybase, db2* etc.. *Identificator_ baza_ de_ date* este un indicator specific fiecărui driver care specifică baza de date cu care aplicaţia doreşte să interacţioneze. În funcţie de tipul driver-ului acest identificator poate include numele unei maşini gazdă, un număr de port, numele unui fişier sau al unui director etc. (jdbc:odbc:testdb, jdbc:oracle:thin).La primirea unui JDBC URL, `DriverManager` va parcurge lista driverelor înregistrate în memorie, pâna când unul dintre ele va recunoaşte URL respectiv. Dacă nu există nici unul potrivit, atunci va fi lansată o excepţie de tipul `SQLException`, cu mesajul *`no suitable driver`*.

2.Trimiterea de secvenţe SQL. Metoda folosită pentru realizarea unei conexiuni este *getConnection* din clasa *DriverManager* şi poate avea mai multe forme:
- Connection c = DriverManager.getConnection(url);
- Connection c = DriverManager.getConnection(url, username, password);
- Connection c = DriverManager.getConnection(url, dbproperies);

O conexiune va fi folosita pentru: *crearea de secvenţe SQL* ce vor fi folosite pentru interogarea sau actualizarea bazei, aflarea unor informaţii legate de baza de date (meta-date)

Clasa Connection asigură suport pentru controlul tranzacţiilor din memoria internă către baza de date prin metodele *commit, rollback, setAutoCo`mmit` .*

O dată făcută conectarea cu *DriverManager.getConection(),* se poate folosi obiectul *Connection* rezultat pentru a se crea un obiect de tip *Statements*, cu ajutorul căruia putem trimite secvente SQL către baza de date
Exemplu: ResultSet r = s.executeQuery("SELECT * FROM tabela1 ORDER BY atr1");

3.Obţinerea şi prelucrarea rezultatelor se face cu ajutorul a două interfeţe: ResultSet şi ResultSetMetaData

Exemplu de utilizare JDBC.

```
import java.sql.*;
import java.io.*;

public class TestJDBC {
        public static void main (String[] args) {
                String dbUrl = "jdbc:odbc:test";
                String user = "dba";
                String password = "sql";
                try{
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                }
                catch(ClassNotFoundException e) {
```

```
                    e.printStackTrace();
                    System.out.println("Eroare incarcare driver!\n" + e);
                }
                try{
                    Connection c=DriverManager.getConnection(dbUrl, user,
password);
                    Statement s= c.createStatement();
                    ResultSet  r =
                        s.executeQuery(
                            " SELECT cod, nume FROM localitati"+
                            " ORDER BY nume");

                    while (r.next()) {
                        System.out.println (
                            r.getString ("cod") + "," +
                            r.getString ("nume") );
                    }
                    s.close();
                }
                catch(SQLException e) {
                    e.printStackTrace();
                }
        }
}
```

Avantajele oferite de produsul JDBC:

- asigură independența SGBD față de codul sursă (de exemplu scris în SQL);
- datele obținute pot fi accesate nu numai din SGBD-ul care le-a creat ci şi din alte sisteme prin interfețe de translatare tip API (Application Programming Interface).

Java Beans

Produsul Java Beans a fost construit pentru a *permite* realizarea unor componente software independente, din care utilizatorii să poată asambla apoi aplicații complexe. Componentele Java Beans pot fi utilizate de dezvoltatori fie pentru a le asambla, fie folosind tehnici de programare tradiționale, fie folosind interfețele grafice oferite de generatoarele din SGBD.

Utilizarea mai multor astfel de componente într-o concepție unitară a condus spre arhitectura EJB (Enterprise Java Beans).

Enterprise Java Beans (EJB)

Prezentare
EJB este o arhitectură pentru *dezvoltarea aplicațiilor distribuite*, tranzacționale şi bazate pe componente. Aplicațiile EJB pot fi dezvoltate complet în Java, prin obiecte tip container.

Specificațiile EJB detaliază nu numai formatul unei componente ci şi un *set de servicii* ce trebuie furnizate de container, reprezentând totodată *o metodologie* pentru construirea aplicațiilor distribuite.

Dezvoltatorul (componentei sau aplicației client) nu trebuie să se preocupe de detalii, cum ar fi: suportul pentru tranzacții, securitatea, accesul la distanță, toate acestea fiind servicii furnizate de serverul EJB şi de container.

Dezvoltarea componentelor EJB cuprinde *cinci activități*:
- *dezvoltare componentă* = scrierea codului pentru o componentă implică o experiență de programare în Java, SQl, JDBC sau SQLJ;
- *implementare componentă* = instalarea şi publicarea componentei şi scrierea fişierului descriptor, care specifică proprietățile fiecărei componente;
- *furnizare servicii server* = implementează cadrul în care rulează containerul componentei (exemplu, Oracle Server este un cadru care suportă containere EJB);
- *furnizare container* = implementează servicii care suportă EJB la momentul execuției;
- *dezvoltare aplicație client* = scrierea codului sursă care va apela metode ale componentelor de pe server.

Sunt *patru părți* din care este compusă o componentă EJB completă:
- *interfața locală* – specifică interfața unei clase instanțiată de container (exemplu, în Oracle modul de instanțiere este indicat prin metodele *create()*);
- *interfața externă* – specifică metodele care sunt implementate în componentă şi care se referă la regulile (logica) afacerii. Componenta va implementa şi servicii adiționale, care vor fi folosite de container;
- *implementarea componentei* – conține codul Java care se referă la interfața externă şi metodele specificate pentru container;
- *descriptorul de implementare* – specifică atributele componentei pentru implementarea şi încărcarea în baza de date.

Aplicația nu accesează componentele în mod direct. Containerul va genera pe server un obiect numit *EJBObject*. Acesta primeşte mesajele de la client, iar containerul îşi poate intercala propriile operații, înainte de a trimite mesajele către componentă.

Entity Beans (EB)
Obiectul EB reprezintă *date persistente* în BD, precum şi metode care operează asupra acestor date.

În contextul BDR există câte o *instanțiere* de EB pentru fiecare înregistrare dintr-o tabelă. O *cheie primară* unică leagă fiecare înregistrare de EB.

Crearea unui obiect EB se face prin apelul metodei *create()* cu parametru de cheie unică.

Ciclul de viață pentru o componentă EB nu este limitat la ciclul de viață al maşinii virtuale pe cae se execută. Dacă dintr-un anumit motiv maşina virtuală nu mai funcționează atunci componenta şi referința acesteia către clienți nu sunt şterse. Un client se poate reconecta la ceeaşi componentă folosind referința alocată anterior şi cheia primară.

O componentă îşi poate *gestiona persistența* singură (bean managed persistance) sau poate lăsa containerul să facă acest lucru (container managed persistence).

EB sunt *obiecte persistente*. Ele pot fi construite în memoria internă şi ştiu apoi să se mențină singure ca părți fizice, stocabile drept câmpuri (exemplu conturile unei bănci) într-o BD.

EB pot fi comparate cu *obiectele serializabile* din Java. Acestea pot fi grupate şi salvate singure, în mai multe moduri: serializare, mapare, obiecte în BD. Nu există specificare EJB care să impună un anumit mecanism de existență.

EB sunt complet diferite de *session beans*. Acestea sunt modele de procese. EB conțin datele şi nu efectuează sarcini complexe, ci sunt obiecte persistente car se asociază clientului. În general, EB sunt folosite pentru a modela datele iar *session beans* pentru a modela procesele. Dacă sunt bine definite atunci EB pot fi reutilizate oricând în cadrul proceselor.

Business Componente for Java (BC4J)

Componentele tip BC4J sunt module Java prefabricate care **pot fi utilizate** într-o aplicație multi-strat **pentru a realiza**:

- o interfață utilizator pe partea de client scrisă în Java, Java Script şi/sau *html*;
- una sau mai multe componente pe stratul de mijloc care se referă la "logica afacerii" şi la obiectele de tip "afacere";
- tabelele din baza de date ce conțin datele necesare aplicației.

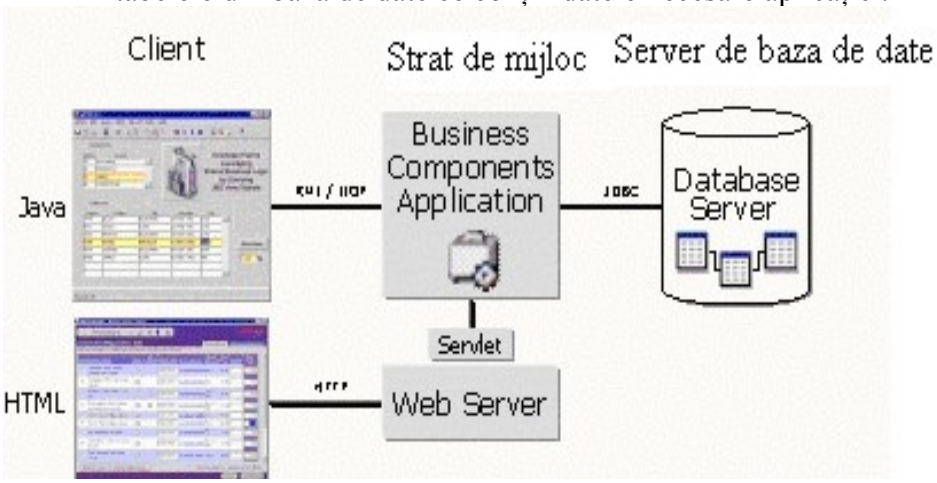


Fig. 1 Exemplu de configurație multi-nivel

Aceste componente **compun un mediu de lucru** (framework) şi **permit**:

- *reutilizarea* structurii logice prin viziuni (în SQL) proprii ale aplicației asupra datelor;
- *accesul* şi schimbarea datelor din viziuni prin intermediul unor produse Java (servlets, JSP, clienți Java);
- *adaptarea* funcționalității aplicației în straturi, fără a fi necesare schimbări ale aplicației deja livrate;
- *suport* pentru implementarea mesajelor *xml*;
- *distribuția* rapidă a componentelor pentru afaceri, sub formă de biblioteci încărcabile în baza de date;
- *actualizarea* tabelelor vizuale (viziunilor) cu produse Java (Servlet, JSP etc.);
- *reutilizarea* componentelor care au fost create deja, în aceeaşi aplicație sau în alta;
- *gestiunea tranzacțiilor* este inclusă automat în modulele BC4J astfel încât proiectantul nu trebuie să scrie rutine de acces la date.

Caracterizarea tehnologiei BC4J**.**

- Furnizează *componente reutilizabile*. După ce o componentă a fost creată, ea poate fi reutilizată atât în cadrul aceleiaşi aplicații cât şi în aplicații diferite.
- BC4J are incluse funcții de *gestiune a tranzacțiilor* la nivelul bazei de date, astfel încât programatorul nu trebuie să scrie rutine de acces la date.
- Foloseşte *viziuni* bazate pe fraze SQL. Aceastea oferă dezvoltatorului posibilitatea de a-şi defini propriile viziuni în funcție de necesitățile aplicației.

- O aplicație BC4J este *compusa* dintr-un număr de componente. Fiecare componentă creată este reprezentată printr-un fişier *xml* şi prin unul sau mai multe fişiere Java. Fişierul *xml* stochează metadate (informații descriptive despre setările şi trăsăturile aplicației), în timp ce obiectele Java stochează codul obiectelor care implementează comportamentul aplicației. *Tipurile de componente*, care au propriul rol într-o aplicație sunt: obiectele entitate, asocierile, obiectele viziune, modulele aplicație, domeniile.

- *Obiectele entitate* sunt primele componente create când se construieşte o aplicație cu BC4J. Fiecare obiect entitate reprezintă o entitate a aplicației.
De exemplu, pentru un sistem de comenzi *on-line* pot fi necesare entitățile "comanda" şi "client". Fiecare obiect entitate se mapează pe o sursa de date, de regulă o tabelă a bazei de date. Se poate spune că obiectele entitate “ascund” datele tabelelor.
Se crează câte un atribut în obiectul entitate pentru fiecare coloana a tabelei (fiecare atribut poate avea acelaşi nume cu cel al coloanei). Atributele definite în cadrul entității reflectă proprietățile coloanei corespunzătoare, incluzând tipul datelor şi restricțiile coloanei.
- *Asocierile* reprezintă acea componentă BC4J care defineşte legături de tip părinte-copil între două obiecte entitate.
- *Obiectele viziune* sunt clase ce permit definirea şi lucrul cu un set de înregistrări, adesea într-o interfață utilizator. De exemplu, o aplicație ar putea conține o formă pentru a vizualiza şi actualiza comenzile prioritare. Pentru aceasta, poate fi creat un obiect viziune "ComandaPrioritara" pe care să se bazeze forma. Fiecare obiect viziune conține o cerere SQL şi e legat de unul sau mai multe obiecte entitate. Astfel, viziunea "ComandaPrioritara" poate prelua date din entitățile "Clienți" şi "Comenzi".
Atributele unui obiect viziune pot fi de două tipuri: bazate pe atributele entităților şi bazate pe atribute calculate. Între doua viziuni pot fi stabilite legături părinte-copil cu ajutorul unor componente de tip asociere-viziune.
Pot fi definite viziuni multiple pentru un obiect entitate, iar un obiect viziune poate selecta date din mai multe obiecte entitate. Datele sunt ascunse la nivelul obiectului entitate şi toate obiectele viziune instanțiate în aceeaşi tranzacție partajează aceste date. Astfel, schimbările făcute într-o viziune sunt vizibile în celelalte viziuni în aceeaşi tranzacție.
- *Modulele aplicație* includ modelul de date precum şi codul scris pentru a implementa soluția. Pentru a crea modelul de date, modulul aplicație conține instanțe ale obiectelor viziune şi legătura între viziuni. Pot fi adăugate noi metode la un modul aplicație şi, în plus, modulele pot fi imbricate.
- *Domeniile de validare* sunt folosite în cazul unor validări complexe (de exemplu în cazul verificării validității unei adrese URL (Uniform Resource Locator), a unei adrese de *e-mail*, a formatului unui număr de telefon etc. Domeniile sunt componente de afaceri separate, nefiind legate de o anumită entiate sau atribut. .

Avantajele utilizării tehnologiei BC4j sunt:
- reducerea costurilor de dezvoltare a aplicației cu BD prin construcția rapidă şi posibilitatea reutilizării componentelor pentru afaceri;
- securitatea datelor ridicată prin utilizarea viziunilor;
- reducerea timpului de realizare al aplicației prin folosirea componentelor prefabricate;
- reutilizarea datelor şi programelor atât în cadrul aceleeaşi aplicații cât şi în alte aplicații.

Java Enterprise Edition (JEE)

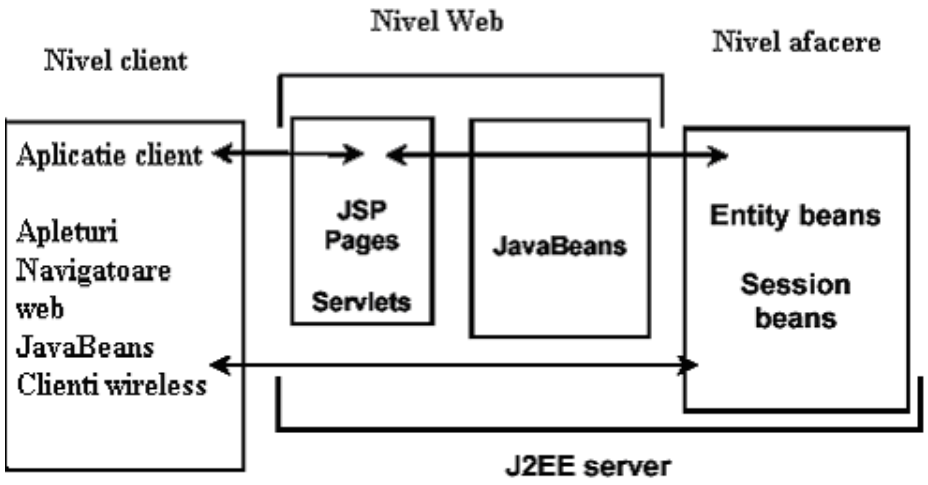
JEE este un mediu de dezvoltare care furnizează o *platformă de lucru bazat pe componente* şi presupune realizarea de aplicaţii multistrat. Astfel, aplicaţia poate fi segmentată pentru a rula pe mai multe dispozitive, aceste dispozitive fiind alese în funcţie de posibilitatea aplicaţiei de a realiza o anumită sarcină.
Obiectivele JEE:

- flexibilitatea ridicată prin introducerea profilului Web;
- extensibilitatea prin integrarea cu produse OpenSource;
- productivitatea ridicată pentru dezvoltarea aplicaţiilor;

Evoluţia: J2EE (2000) – JEE5 (2006) – JEE6 (2009) – **JEE7** (2012 va rula în mediul Cloud Computing).
Noutăţi JEE6: multe interfeţe tip API noi, profil Web, componente vechi actualizate (EJB, Servlets, JSP etc.), componente noi (CDI – Context and Dependency Injection, JAX – Java API for XML, Managed Beans etc), dispar multe fişiere de configurare – simplitate.

JEE presupune lucrul prin componente de *tip container*. Containerele sunt module care la momentul rulării furnizează anumite servicii. De exemplu, un container JEE Web trebuie să furnizeze, în momentul rulării aplicaţiei, suport pentru manevrarea cererilor clienţilor precum şi să returneze rezultate către clienţi.

Mediul JEE este **structurat** pe următoarele **3 niveluri** (staturi):



Nivelul client - Aplicaţiile de la nivelul client pot rula pe calculatoare, telefoane mobile sau dispozitive *wireless*. Aceste aplicaţii lansează o cerere către server la iniţiativa utilizatorului şi apoi îi transmit utilizatorului rezultatele obţinute. Cererea poate fi trimisă oricărui strat JEE, inclusiv unui strat Web sau afacere. Nivelul client poate fi: module Java, modul *applet*, un client *html* etc.
Nivelul web (prezentare) – este, în general, responsabil cu livrarea conţinutului dinamic către/dinspre cererea clientului. În cele mai multe cazuri acest lucru este realizat de Servlets şi JavaServerPages. Manevrarea conţinutului dinamic este, de asemenea, suportată şi de JavaBeans. Rolul nivelului *web* este de a evita accesul direct al utilizatorilor la date şi astfel permite crearea unor aplicaţii mult mai solide şi mai uşor de întreţinut.
Nivelul afacere - furnizează containere care găzduiesc obiecte ce administrează "logica afacerii". Acest nivel conţine componente Enterprise Java Beans (EJB). EJB reprezintă tehnologia ce se foloseşte pe partea de server în modelul multinivel JEE. "Componentele afacerii" dezvoltate în acest nivel sunt independente de celelalte straturi. Aceasta le face mai

sigure. Componentele EJB conţin reguli "de afaceri". Acestea pot fi schimbate fără a afecta nivelul client sau nivelul prezentare.

Arhitectura JEE cuprinde următoarele **componente:**
Container pentru aplicaţia client: gestionează execuţia componentelor client, furnizează servicii folosite pentru a se conecta la containerele altor obiecte. Aplicaţiile client şi containerul rulează pe maşina client.
Server JEE : este componenta ce rulează la momentul execuţiei, furnizează servicii pentru conectarea la containere a altor obiecte, furnizează containere EJB şi containere *web* (pentru gestiunea execuţiei *servlet*-urilor şi a paginilor JSP).

Concluzie. În aplicaţiile tradiţionale, programatorul trebuie să scrie cod pentru gestiunea tranzacţiilor, a resurselor, a firelor de execuţie etc. Acest mod de abordare este scump şi necesită timp. Arhitectura JEE furnizează aceste servicii şi multe altele la fiecare nivel al mediului multistrat.

SQL Java (SQLJ)

SQLJ *permite programatorilor* să includă comenzi SQL în codul sursă Java, adică într-un program Java se scriu comenzi SQL. Oracle suportă specificaţiile standardului ANSI de SQL, care acoperă doar operaţiile statice SQL, adică acelea care sunt redefinite şi nu se schimbă la momentul execuţiei. Oracle SQLJ oferă însă şi extensii pentru suportul operaţiilor SQL dinamice.

Componentele SQLJ
SQLJ este alcătuit dintr-un translator şi din componenta executabilă (run-time).

Translatorul este scris în întregime în Java şi suportă comenzile SQL şi instrucţiunile Java. Comenzile SQL, atât cele de declarare cât şi cele executabile sunt specificate în program (presursă) prin cuvântul rezervat *#sql* (exemplu: *#sql {INSERT INTO ang(nume, sal) VALUES ("Anda", 800)*) şi sunt intercalate printre instrucţiunile Java.
Translatorul produce un fişier tip *.java* (programul sursă) şi unul sau mai multe *profile SQLJ* cu informaţii despre comenzile SQL. După aceea, el apelează compilatorul de Java, care produce un fişier tip *.class* (programul obiect).
Translatorul este similar altor precompilatoare Oracle (C, Pascal, Cobol etc.) şi permite: verificarea sintaxei comenzilor şi instrucţiunilor, verificarea informaţiilor din dicţionarul BD verificarea compatibilităţii între tipurile date din SQL şi Java, înlocuirea comenzilor SQL cu apeluri către componenta executabilă (run-time).

Componenta executabilă (run-time) primeşte apelurile de la translator şi execută comenzile SQL aferente. În SQLJ standard execuţia se face prin intermediul driverului JDBC (este şi cazul Oracle). Când se rulează o aplicaţie SQLJ, componenta executabilă este apelată pentru a efectua operaţiile specificate prin comenzile SQL.

Profilele SQLJ
O componentă suplimentară a produsului SQLJ este *Customizer*, care adaptează profilel SQLJ pentru un SGBD specific.

Pentru modul standard de generare a codului SQLJ, *profilele sunt* resurse sau clase Java generate de translator, cu detalii despre comenzile SQL incluse în codul presursă.

Un profil *este format* dintr-o colecție de intrări, fiecare intrare fiind descrierea detaliată a unei comenzi SQL.
Translatorul *crează profile* şi le salvează sub formă de resurse binare sau de clase, în funcție de setări.

SQLJ *generează* un profil pentru fiecare conexiune la BD din aplicație, fiecare conexiune corespunzând unui set specific de entități SQL în baza de date. Standardul SQLJ impune formatul şi conținutul profilelor.

Personalizarea profilelor permite dezvoltatorilor să adauge informații suplimentare la profile, în funcți de SGBD ales.

Profilele SQLJ *au implicit* extensia *.ser* , extensie folosită pentru obiectele serializabile.
Un profil *nu este creat dacă* în codul presursă nu există comenzi executabile.

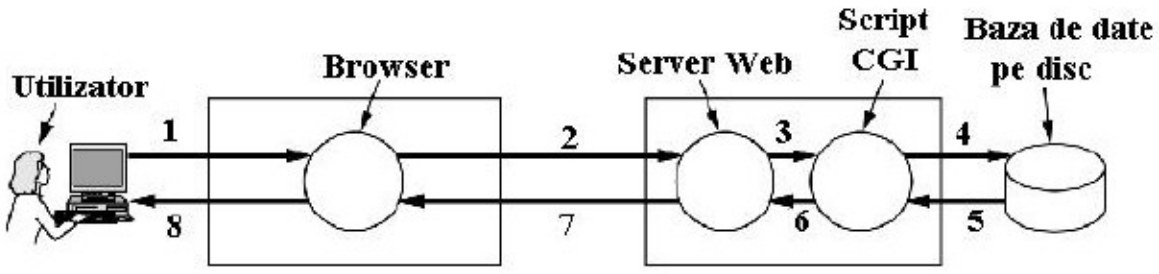
2.4. Tehnologia Web

Bazele tehnologiei Web

Deşi iniţial a avut un caracter academic, reţeaua Internet a devenit în scurt timp de la lansare un mediu deosebit de favorabil pentru afaceri. Acest lucru este posibil datorită dezvoltării rapide a reţelei globale de calculatoare, constituind nu numai o piaţă practic fără limite, dar oferind resurse imense de informaţie. Activităţile comerciale care pot fi derulate *on-line* sunt tot mai multe, mediul virtual Web creând noi posibilităţi pentru acestea, iar metodele de lucru sunt tot mai diversificate.
Tehnologia Web, prin caracteristica sa de mediu de lucru global, permite ca orice firmă să poată ajunge mult mai uşor la audienţa ţintă şi oferă posibilitatea unei legături directe vânzător – consumator, eliminând intermediarii. Pe Internet pot fi derulate afaceri de mare sau de mică anvergură, afaceri ale unor firme puternice sau a unor persoane fizice care îşi operează afacerea de acasă. Toate aceste lucruri pot fi realizate doar dacă "în spate" există o bază de date care oferă toate informaţiile necesare în timp util.
Elementele utilizate în cadrul tehnologiei Web pentru lucrul cu pagini dinamice în aplicaţii sunt date de următorul *flux de activităţi* (primele două sunt pentru partea de prezentare a aplicaţiei – interfaţa aplicaţiei), celelalte două pentru partea de prelucrare – logica aplicaţiei:

- utilizatorul accesează, de la un nod al reţelei, un navigator (browser) Web şi formulează cereri de regăsire;
- prin intermediul acestuia se apelează la serviciile unui Server Web;
- aceste servicii sunt oferite prin module de program rezidente pe Server Web şi realizate prin diferite tehnologii (module CGI, produse din platforma Java – servlets, JSP etc.);
- modulele de program rezidente sunt utilizate pentru accesul la baza de date şi pentru prelucrări asupra datelor, obţinându-se rezultate.

Prin intermediul aceloraşi elemente fluxul se derulează şi în sens invers, rezultatele trecând prin aceleaşi componente ajungând în final la utilizatorul care a formulat cererea de regăsire.



Pentru fluxul de activităţi de mai sus, tehnologia Web presupune utilizarea mai multor produse software. Dintre acestea, cele mai importante pentru proiectul nostru vor fi prezentate în continuare.
Module rezidente pe Server Web
Calea directă, şi prima ca apariţie istorică, către programarea Web o reprezintă *scripturile CGI - Common Gateway Interface*. Pentru a putea scrie astfel de module de program sunt necesare câteva cunoştinţe de bază privind protocolul HTTP si modul în care serverul Web

rulează scripturi CGI. Aceste programe care fac ca mediul Web sa fie dinamic sunt adesea numite "scripturi CGI" pentru că folosesc o interfață standard - CGI, care este suportată de toate serverele Web. CGI defineşte un mediu în care orice proces este rulat de către serverul de Web. Un proces rulat de server poate fi scris în orice limbaj de programare. Interfața CGI redirectează datele de intrare către un proces rulat de server şi interceptează datele de ieşire ale acestuia.
Alte tipuri de module rezidente pe server Web : servlets, JSP etc.
Protocolul HTTP
Protocolul HTTP - HyperText Transfer Protocol este metoda fundamentală – standardizată internațional - de comunicație prin Web. Toate sistemele care rulează pe Internet recunosc acest protocol, inclusiv Oracle 11g. Sarcina unui server Web este să răspundă la o cerere HTTP primită de la client. Dacă în urma cererii se accesează un fişier – care poate conține cod HTML sau date multimedia - atunci serverul va localiza fişierul şi îl va trimite către client prin intermediul unui navigator (browser) Web.
Navigatoare Web
În prezent, există numeroase navigatoare (browsers) Web cu utilizare largă: Microsoft Internet Explorer, Netscape Navigator, Opera etc. Un *browser* este un produs program care rulează pagini Web şi realizează navigarea pe diferite locații de pe Internet. Acesta are rolul de a aduce o copie a unei pagini Web pe calculatorul local şi de a o prezenta pe ecran pentru a fi vizualizată de utilizator.
Câteva *caracteristici ale unui browser* Web sunt: stabilirea unor repere pentru anumite site Web favorite; oferă posibilitatea prezentării mai multor ferestre de navigare; oferă cadre sau vederi multiple în interiorul unei ferestre; permite interogări – cereri de regăsire multiple către servere Web; permite utilizatorului să configureze anumiți parametri de lucru care privesc anumite caracteristici de afişare; asigură un anumit nivel de securitate a transmiterii datelor; asistă utilizatorul pentru noile tipuri de date multimedia, ce pot fi configurate de utilizator; oferă suport pentru limbaje de script.
Servere Web
Un Server Web este un sistem software specializat care este activ tot timpul, amplasat pe o maşină puternică, într-un mediu Internet şi care primeşte cereri Web pe care le rezolvă. În acest sens, serverul Web va verifica dacă un anumit modul de program activat este executabil şi, în caz afirmativ, îl va rula, pentru a obține rezultate corespunzatoare. Serverul Web poate fi configurat şi în aşa fel încât o cerere către un fişier aflat într-un director anume să se execute ca un script. Acest director special este numit de obicei *cgi-bin*. Serverele Web au adoptat aceste "directoare speciale" din motive de securitate.
Limbajele de scenarii (script)
Paginile Web conțin opțiuni privind modul de lucru. Navigatorul afişează o pagină Web citind nişte intrucțiuni dintr-un cod sursă de limbaj script. Primul, dar şi cel mai utilizat limbaj script este HTML, dar există şi altele destul de des folosite : JavaScript, BasicScript, PHP etc. Orice pagină Web are în spate un cod sursă de limbaj script.
HTML - HyperText Markup Language - nu este un limbaj de programare universal, ci unul special destinat realizării documentelor Web. Hypertextul, care apare într-un document Web sub forma unui text marcat luminos sau subliniat, permite utilizatorilor să realizeze legături cu alte documente sau să se deplaseze oriunde în cadrul aceluiaşi document. Navigatorul utilizează un URL asociat cu o hiperlegătură pentru a localiza documentul corespunzător. Documentele Web sunt legate sub forma unei "pânze de păianjen – World Wide Web" prin intermediul legăturilor. Elementele HTML specifică structura logică a unui document Web şi sugerează prezentarea virtuală a documentului. Prezentarea propriu-zisă a documentului este lăsată în sarcina aplicației responsabile cu redarea conținutului acestuia şi anume a navigatorului. Limbajul HTML furnizează două tipuri de elemente: etichete şi referințe de

entitate caracter. Etichetele permit definirea elementelor cheie, cum ar fi paragrafe, antete, legături, liste, marcaje luminoase etc. Referinţele de entitate caracter permit reprezentarea caracterelor speciale din cadrul unui document HTML pe care un browser sau alte programe client le pot interpreta în mod eronat.

Iniţiere în limbajul HTML

HTML – Hyper Text Marckup Language, este un *limbaj de scenarii* (script), standardizat internaţional, destinat construirii paginilor (documentelor) Web.

Acest limbaj este suportat de toate *navigatoarele* (browsers) de Internet (Netscape Navigator, Internet Explorer, Mozilla Firefox, Google Chrome, Opera etc.), dar şi de alte sisteme software care permit construirea de *site*-uri (editoarele de Web etc.).

HTML este *primul* limbaj de scenarii care a fost utilizat (primul standard în 1991), ulterior apărând şi altele (PHP, JavaScript, BasicScript etc.).

De la apariţie, HTML a evoluat în DHTML (dinamic) şi apoi spre XML (eXtensible Marckup Language) care are standard din 1998. Toate sistemele software care lucrează cu tehnologia *e-business* (inclusiv Oracle) acceptă XML.

Într-un program scris în HTML se *acceptă secvente scrise în alte limbaje de scenarii* (de exemplu, se poate scrie o secvenţă în Java Script între *<script language=”JavaScript”>* si *</script>*.

Programul sursă HTML este salvat într-un fişier cu *extensia .htm*.

Elemente de limbaj HTML

Programul sursă HTML este împărţit în zone numite *scenarii* /secţiuni.

O secţiune este delimitată printr-un *marcator* (tag) de început *<tag>* şi unul de sfârşit *</tag>*.

Structura unui program HTML este:

```
<html>          - început de program

<head>
        comenzi de declarare
</head>

<body>
        comenzi executabile
</body>

</html>         - sfârşit de program
```

Pentru *scrierea* pe ecran a unui text dorit, acesta apare ca atare pe o linie din program.

Comentariul într-un program HTML: pe o linie se scriu două caractere *slash* urmate de textul dorit (// text).

Câteva comenzi HTML

<title> - dă un *nume* paginii Web şi afişează o *legătură* cu altă pagină.

<img *argumente*> - afişează în pagină, o imagine specificată.

<form> ... </form> - permite construirea unui *videoformat* (formă) în pagina Web.

<input *argumente*> - construieşte diferite *obiecte* vizuale într-o formă (exemplu: type=”text” pentru obiect tip câmp-text; type=”button” pentru obiect tip butoane etc.)

<frameset> ... </frameset> - permite construirea unor *cadre* într-o fereastră de browser. Fiecare cadru va conține un document (de obicei HTML) care se încarcă dintr-un fişier.

<frame *argumente*> - încarcă un *document* în cadrul curent.

<layer> ... </layer> - defineşte un *strat* de un anumit ordin (numerotarea de la zero) pentru suprapunerea obiectelor (documentelor) pe ecran.
Notă. Analog există şi *ilayer* care se foloseşte dacă poziția stratului depinde de modul de afişare al documentului.

 ... - specificarea caracteristicilor pentru caracterele ce vor fi afişate.

Exemple

1. Să se construiască o pagină Web care să aibă numele *vax* şi să conțină textul *text de probă* precum şi o fotografie tip *bmp.*

```
<html>
<head>
<title> Pagina VAX
</title>
</head>

Text de proba

<body>
<img src="d:\man\pozamea.bmp" name="p1">
</body>

</html>
```

2. Să se construiască o pagină Web care să conțină:

- o legătură;

- o imagine;
- o formă cu două elemente: un câmp-text, un buton tip *push*
- un text afişat, două straturi: pe primul o imagine, pe al doilea un text.

```
<html>

// Definire si declarare
<head>
// legatura si denumirea
<title> Pagina speciala
</head>

//Executabil
<body>
<center>
<img src=”poza1.gif” name=”p1” width=200 height=100>
</center>

//forma
<form name=”forma1”>
Nume:
<input type=”text” name=”nume1” value=””> <br>
<input type=”button” name=”buton1” value=”apasa” onClick=”alert(‘Yo’)”>
</form>

//straturi
<layer name=pic z-index=0 left=200 top=100>
<img src=”poza1.gif” width=160 height=120>
</layer>

<layer name=txt z-index=1 left=200 top=100>
<font size=+4> <i>Exemplu</i></font>
</layer>

</html>
```

2.5. Tehnologia Grid Computing

2.5.1. Intranet

Definiții

Utilizarea tehnologiilor Internet pentru *legarea într-un tot unitar* a resurselor informaționale ale unei organizații: texte, baze de date, documente etc.

Implementarea tehnologiilor *Internet la nivelul unei organizații* (întreprinderi, instutuții).

Internet privat, care reprezintă rețeaua de transmisii de date privată a unei companii, deosebindu-se conceptual de *Extranet*, care este privit ca partea din rețeaua de transmisii de date comună cu partenerii agreați.

O rețea de calculatoare internă unei organizații, deci protejată de lumea exterioară, adică o insulă autonomă cu administrare internă, dar supusă protocoalelor de comunicare din Internet.

Componente

1. *Hardware:* servere Intranet (microcalculatoare, minicalculatoare), linii de comunicație (cabluri coaxiale, fibră optică, radio), arhitectură de rețea LAN (Ethernet etc.), echipamente de rutare (hub, switch etc.).

2. *Software:* sisteme de operare (Windows, OS2, Mac etc.), software de rețea (Unix, Windows xxServer, Novell etc.), protocoale de rețea (TCP/IP, DECnet etc), software specializat (SGBD, navigatoare, Platforma Java, CASE etc.).

Caracteristici

Intranet este un *concept flexibil* = nu are o rețetă de implementare prestabilită, ce trebuie respectată pas cu pas, ci poate fi modelat în funcție de nevoile organizației.

Intranet este realizat *pe baza protocoalelor Internet* = respectînd standardele din tehnologia Internet şi de aceea poate fi rapid actualizat (este deschis).

Intranet reprezintă soluția ideală pentru organizațiile cu un *număr de utilizatori mare* = se justifică soluția Intranet dacă sunt peste 1000 de utilizatori, iar localizarea lor distribută pe o suprafață geografică mare.

La baza tehnologiei Intranet este *serverul Web* = configurarea serverului Web se face pe o rețea locală, iar funcțiile sale pot fi extinse cu alte facilități: accesul la baze de date, apelul unor aplicații etc.

Intranet *nu implică în mod obligatoriu conectarea la Internet* , dar această posibilitate oferă o mulțime de avantaje.

Servicii

Rolul oricărui Intranet este de a oferi *anumite servicii* utilizatorilor.

1. *Serviciile de transport* permit transferul informației de la un punct la altul în Intranet:
- *transferul de date* într-o rețea locală, respectând protocoalelor de comunicație;
- *accesul de la distanță* la date şi la produse software, care aparțin Si;
- *accesul la Internet* pentru comunicarea cu exteriorul;

- *interconexiunea* între rețelele LAN şi WAN.

2. *Serviciile de administrare* a rețelei sunt operaționale sunt operaționale în număr de trei:
- *supervizarea* unui Intranet permite supravegherea tuturor componentelor hardware şi software ale ansamblului, având la bază standardul SNMP – Simple Network Management Protocol din Internet;
- *administrarea* Intranet este realizată de persoane specializate care au la dispoziție instrumente software specializate. Acestea asigură administratorilor de rețea o viziune unitară asupra întregii rețele a organizației şi le permite gestionarea eficientă a resurselor de calcul;
- *serverele de filtrare* (cache / proxy) au fost folosite mai întâi în Internet şi apoi în Intranet şi au rolul de a reduce fluxul informațional (traficul) prin sistemul de comunicație. Aceste servere dețin mecanisme ce le permite o selectare a cererilor şi a răspunsurilor.

3. *Serviciile de securitate* au rolul de a proteja Intranetul şi ele se referă la:
- *autentificarea* - este utilizată pentru a verifica dacă utilizatorul are drept de acces. Sistemele de autentificare dețin atât mecanisme simple (parole) cât şi mecanisme mai complexe (ActivCard, SecureID etc.);
- *criptarea* – are drept scop garantarea confidențialității fluxului de informații din Intranet;
- *filtrarea* – se realizează prin mecanisme de tip bariere (firewall), care autorizează anumite tipuri de pachete (servicii, adrese, conținut) conform setului de reguli stabilite prin păolitica de securitate a rețelei.

4. *Serviciile de partajare a datelor* din Intanet sunt destinate stocării şi interogării datelor pe care le memorează:
- *servicii de stocare şi acces* sunt asigurate de: *serverele de fişiere* (fişiere clasice ale S.O. utilizat, de exemplu protocoalele FTP, NFS din Internet), *serverele de date* (asigurate în special de SGBD relaționale şi este necesară o interfață între BD şi serverul Web pentru navigarea cu un *browser*), *serverele de documente* (sunt în general serverele Web şi permit utilizatorului să caute şi să consulte documentele produse în organizație);
- *servicii de producere şi publicare a informațiilor* (SGBD, clienți etc.).

5. *Serviciile de comunicare* între diferitele persoane din organizație sunt importante pentru succesul afacerii:
- *poşta electronică* are la bază mecanismele din Internet: SMTP – Simple Mail Transfer Protocol, POP3 – Post Office Protocol, Imap – Interactive Mail Access Protocol;
- *circulația documentelor* (workflow) este o extensie a poştei electronice şi permite circulația unui document după o schemă prestabilită;
- *videoconferințele* pentru care se folosesc produse software specializate;
- *lucrul în comun în timp partajat* este o extensie a circulației documentelor şi permite mai multor persoane, situate la distanță unele de altele, să vizualizeze (Net Meeting) sau să prelucreze împreună acelaşi document;
- *forumul* permite utilizatorilor de Intranet să schimbe idei, mesaje şi poate fi în timp real (Internet Relay Chat – IRC) sau în timp diferit.

6. *Serviciile de statistici* (anuar) sunt necesare pentru că în Intranet fiecare utilizator apelează la servicii diferite, pe calculatoare diferite. Tipurile de statistici sunt de: utilizatori, servere, servicii, aplicații, date etc.

7. *Serviciile pentru dezvoltarea aplicațiilor* se referă la tot felul de interfețe şi instrumente (limbaje de programare, generatoare, medii de dezvoltare etc.) care sunt adaptate mediului rețea şi sunt utilizate în ateliere de software pentru aplicații Intranet.

8. *Serviciile de acces* la informații şi la aplicațiile Intranet sunt incluse în navigatoare (Internet Explorer, Netscape Navigator, Opera etc.).

Organizarea Intranet

Sunt trei *modele de organizare* a Intranetului:
- modelul centralizat: toate resursele de calcul se găsesc într-o singură locație;
- modelul descentralizat: resursele de calcul sunt răspândite în organizație şi sunt dependente de departamentul în care se găsesc calculatoarele;
- modelul mixt: există un centru în care se stabilesc politici generale, iar departamentele lucrează în concordanță cu aceste politici (cel mai utilizat model).

Factorii de care depinde modul de alegere a unui model de Intranet sunt: controlul resurselor, responsabilitatea resurselor, mărimea organizației, numărul resurselor.

Proiectarea Intranet

Paşii (fazele) care se parcurg pentru realizarea unui Intranet:
1. *Obținerea acordului* la nivelul organizației, atât din partea conducerii cât şi din partea nivelurilor de execuție pentru: alocarea resurselor necesare, oganizarea proiectului, sprijinul pe toate planurile.
2. *Planificarea strategiei* de realizare a Intranetului presupune: identificarea structurii organizaționale existente, descrierea nivelului tehnic existent în organizație, definirea modelului de comunicație a informațiilor în organizație, avantajele şi dezavantajele, identificarea sistemelor existente în organizație.
3. *Stabilirea obiectivelor* proiectului înseamnă precizarea a ceea ce trebuie să facă Intranetul. În acest sens, se colectează informațiile de la angajați şi de la informaticieni şi se elaborează o listă a cerințelor şi specificațiilor de funcționare.
4. *Recrutarea* echipei de dezvoltare a Intranetului trebuie să vizeze reprezentanți ai diferitelor departamente ale organizației: specialişti în informatică (rețele, Si, SBD, programare, grafică, administrare), resurse umane, gestiune documente, organizare, serviciul vânzări etc.
5. *Dezvoltarea proiectului* de Intranet presupune gruparea conținutului pe subiecte şi categorii principale, după care se vor adăuga alte informații corelate după departamente. *Categoriile de informații* comune paginilor de Intranet sunt: noutăți, ştiri, prezentarea organizației, contracte, asistență tehnică, resusrse, vânzări, clienți, contact (telefon, e-mail etc.).
   După gruparea informațiilor, acestea sunt structurate pe niveluri conform cerunțelor utilizator şi într-o formă agreată (liniar, ierarhic, grilă etc.).

6. *Definirea interfeței cu utilizatorul* trebuie să țină cont de forma şi conținutul acesteia: prietenos, lizibil, uşor de utilizat, relevant, precis.

Accesarea informațiilor din Intranet

Documentele de pe serverul Web pot fi accesate prin intermediul *produselor de tip navigator* (browser – Internet Explorer, Netscape Navigator, Opera etc.). Pe lângă afişarea documentelor, el poate solicita în plus imagini, sunet, video şi aplicații de la server, dacă acestea nu se găsesc în document.

Caracteristica ce a condus la acceptarea acestei tehnologii, inițial în Internet şi apoi în Intranet, este aceea că permite ca *un document să fie alipit* (legat) la alt document. Acest lucru presupune că un anumit utilizator nu trebuie să cunoască localizarea unui anumit document sau sursele de informații, accesul fiind permis doar printr-o simplă selecție (click mouse).

Serverele HTTP sunt disponibile pentru toate tipurile de platforme hardware şi software. Practic, un server Web trece informația către o execuție exterioară. Acesta este modul în care se pot accesa aplicații exterioare, de exemplu baze de date. O astfel de procedură este larg răspândită şi permite relativ simplu să integreze un server Web cu orice aplicație cu baze de date. De aceea, mulți furnizori de aplicații cu baze de date încorporează această funcționalitate în produsele lor (SGBD, aplicații).

Securitatea Intranet

Securitatea în Intranet *presupune*: securitatea serverului Web, securitatea serviciilor TCP/IP, securitatea navigatoarelor client.

Metodele de securitate aplicate în Intranet se referă la autentificare: prin parolă, prin adresa IP de rețea, combinat.

Programarea în Intranet

Intranetul presupune *flexibilitate software*, adică oferă posibilitatea de ajustare şi de modificare în funcție de problema abordată. Acest lucru este realizabil prin completarea sistemelor software cu secvențe scrise în limbaje de scenarii (script), conform cerințelor utilizator.

Principalul serviciu care *necesită personalizare* este site-ul Web , celelalte servicii (poştă electronică, transfer de fişiere etc.) beneficiind de un software care acoperă toate problemele care pot apărea.

Serviciul de Web presupune două aspecte: interfața cu utilizatorul, interfața cu bazele de date.

Interfața cu utilizatorul presupune posibilitatea ca acesta să completeze formulare tip sau să selecteze opțiuni în timpul navigării.
Luând în considerare criteriul locului unde rezidă programul script, există *două variante* de implementare:

- pe server SSS (Server Side Script) (servlets): JavaScript, JSP, ASP, PHP. Serverul execută un anumit program script de la care preia ieşirile şi le trimite clientului;
- pe client CSS (Client Side Script) (applets): JavaScript, VBScript, applet Java, controale ActiveX.

Pentru utilizarea acestor tehnologii există *navigatoare specializate*: Internet Explorer (permite lucrul cu VBScript, controale ActiveX, applet Java, JScript (subset JavaScript)), Netscape Navigator (permite lucrul cu JavaScript, applet Java).

Interfața cu bazele de date presupune ca serviciul de Web să țină cont că indiferent de mărimea organizației, aceasta foloseşte una sau mai multe BD.
Acest lucru a dus la apariția unor *motoare de BD* (software optimizat care ştie să execute anumite comenzi asupra unui tip de BD) şi a unui standard privind limbajul de interogare a BDR (SQL). Aceste motoare de căutare respectă o serie de standarde, de exemplu ODBC, JDBC etc., ceea ce reprezintă un aspect important pentru programatori, deoarece acesta nu mai e obligat să învețe să lucreze cu fiecare dintre moroarele de baze de date utilizate în cadrul organizației.
În cadrul interfeței cu BD *administratorul BD* pune la punct modelul conceptual al BD, ajută la implementarea acestuia şi asigură apoi protecția datelor. Munca lui se complică dacă trebuie să țină cont de specificul interfeței Web şi de aspectul problemelor de securitate suplimentare care apar în acest caz.
Din punct de vedere al *programatorilor*, aceştia consideră că trebuie să se concentreze doar pe soluțiile de programare SSS (Server Side Script – partea de server), în timp ce partea de client este mai uşoară. Acest lucru este doar aparent, deoarece în realitate accesul la o BD mare necesită multe programe script pentru împărțirea sarcinilor şi apare astfel necesitatea lucrului în echipă. Produsele informatice pe care programatorii le pot utiliza sunt diverse: limbaje de programare universale (C, Pascal, Java etc.), interfețe standard (CGI, JSP, ASP etc.), produse tip bariere de acces - *gateway* (dbWeb, ColdFusion etc.), limbaje de script (HTML, JavaScript, PHP etc.).

Avantajele Intranet

- comunicațiile mai rapide între utilizatori şi între aplicații;
- managementul de rețea şi de BD se realizează mai simplu şi mai rapid;
- accesul la bazele de date este protejat şi simplificat la nivelul organizației;
- sursa de date este integrată şi gestionată în mod unic;
- productivitatea beneficiarilor de Intranet este crescută;
- se realizează economii băneşti majore (privind circulația informației, a documentelor);
- se realizează economie de timp privind documentarea şi fundamentarea deciziei.

Extranet

Conceptul de Extranet reprezintă al treilea val în evoluția rețelelor de calculatoare, după Internet şi Intranet, realizând legătura între acestea. *Noțiunea* a fost introdusă de Jim Barksdale (la Netscape Communications) pentru a descrie produsele software care facilitează colaborarea între organizații.

Extranet reprezintă utilizarea tehnologiilor Internet în vederea conectării resurselor informaționale a mai multor organisme, între care există legături de colaborare.

Extranetul poate fi privit ca:

- partea din Intranetul unei organizații care este accesibilă şi altor organizații;
- partea comună a două sau mai multe Intraneturi;
- o conexiune de colaborare pe Internet între două sau mai multe organizații.

În context de Extranet, *informația poate fi* accesată doar de părțile care colaborează sau poate fi publică.

Categoriile de aplicații care se pretează la Extranet sunt:

- grupuri de comunicare private, ale unor organizații, care au rolul de a difuza idei, experiențe etc.;
- grupuri de lucru ale unor organizații care dezvoltă proiecte comune;
- programe de instruire şi educaționale care sunt dezvoltate în mod partajat de mai multe organizații;
- cataloage cu produse accesibile doar pentru un anumit grup de clienți;
- managementul proiectelor care se desfăşoară în comun pentru mai multe organizații.

2.5.2 Inițiere în tehnologia Grid Computing (GC)

În lume, tehnica de calcul din organizații este *insuficient folosită*. Sistemele de calcul, de multe ori, sunt greu de schimbat, scump de întreținut şi de dezvoltat.

Schimbările în orice organizație apar în permanență, nevoia de informație este din ce în ce mai mare şi din aceste motive *adaptarea* trebuie să se facă cât mai repede, cu un efort cât mai mic pentru a ramâne competitive.

Cerințele pentru performanță cresc continuu, în timp ce *bugetele* pot rămâne neschimbate şi de aceea organizațiile, de multe ori, dezvoltă serverele (upgrade) sai achiziționează altele noi mai puternice.

Soluția la problemele de mai sus este un model nou de abordare, denumit Grid Computing – GC (Grilă de Calculatoare). Noua tehnologie are diferite alte denumiri: Oracle 11g, Adaptive Computing, Utility Computing, Organic Computing, Computing on Demand, Hosted Computing etc.

Pentru utilizatori şi aplicații, în tehnologia GC *nu mai contează*: unde sunt stocate datele, unde sunt stocate aplicațiile, ce calculatoare procesează cererea de regăsire, ce resurse sunt folosite în rețea.

GC reprezintă utilizarea coordonată a mai multor servere mici care acționează ca un singur sistem foarte puternic (Grilă de Calculatoare).

Istoric

GC a fost concepută ca *o tehnologie informatică* pentru:
- crearea unui mediu de tehnică de calcul dinamic care să împartă resursele şi rezultatele;
- mărirea capacității de stocare şi protecția acestor date în caz de incidente;
- menținerea unor costuri cât mai mici pentru exploatarea întregului sistem.

Prima aplicare ştiințifică a tehnologiei GC a fost în proiectul "Search for extraterestrial intelligence, seti@home". În acest proiect semnalele de la telescoape, de la receptoarele radio şi din alte surse care monitorizau spațiul au fost distribuite pe microcalculatoare *prin Internet*. Această rețea de microcalculatoare prelucrau informația primită căutând semnale despre viața extraterestră.

După *modelul Intranet*, organizațiile îşi vor dezvolta tehnologia GC pentru resursele lor de calcul, fără să se expună Internetului.

Pentru prima dată, tehnologia GC este *adaptată la baze de date* de firma Oracle, prin lansarea versiunii *Oracle 11g*. Principalele tehnologii informatice care, prin integrare, au făcut posibilă apariția noii versiuni de Oracle au fost: GC, Intranet, Internet, servere multiple (Mail, aplicații, rețea, date, SGBD etc.), arhitectura NC, Inteligența Afacerii, SBD distribuite.

Progresul tehnologic a putut avea loc datorită tehnologiilor informatice de mai sus, datorită apariției unor componente tot mai puternice şi mai ieftine şi datorită posuibilității integrării tehnologiilor (hardware, software, date, comunicații).

Rețeaua Grid Computing

Intr-o economie bazată pe cunoaştere, cum este cea actuală, avem nevoie de o rețea care să alimenteze uriaşa putere de calcul şi nevoia de informații în mod eficient. Aceasta este rețeaua Grid Computing sau Grila de Calculatoare – GC.

O nouă paradigmă de calcul

Rețea GC funcționează prin conectarea unei mari varietați de: calculatoare, baze de date, software, utilizatori, modele etc. Conectarea acestor resurse nu este nouă ea fiind deja prezentă în Internet, Intranet, Extranet. Însă, față de acestea rețeaua GC permite o serie de facilități suplimentare:

- se poate aloca putere de calcul dinamic de la diferitele procesoare din rețea care aparțin altor calculatoare decât cel la care se lucrează. În acest fel, utilizatorii vor putea folosi din rețea puterea de calcul a unui supercalculator doar atunci când au nevoie, fără să fie obligați să cumpere un supercalculator (computational grid);
- se poate împărți o sarcina primită de un calculator cu alte calculatoare conectate în rețea;
- calculatorul la care se lucrează devine o poartă spre un supercalculator, iar utilizatorul va plăti pentru puterea de calcul de care are nevoie, atunci când va avea nevoie;
- se pot accesa cantitați uriaşe de date, fie că sunt stocate pe echipamentele periferice ale diferitelor calculatoare conectate în rețea, fie că provin de la echipamente speciale de citire (data grid);
- se pot rula noi tipuri de aplicații informatice, care vor avea acces la cunoştințele din rețea şi vor putea construi răspunsuri ad-hoc (knowledge grid);
- se permite lucrul la proiecte în echipă compusă din membri de pe diferite continente, pentru diferite organizații. Acest Service Grid Se creeaza astfel, în mod dinamic, o organizație virtuala care lucrează în laboratoare virtuale (service grid).

Tehnologii şi aplicații revoluționare

Rețelele GC vor revoluționa informatica la fel de mult pe cât au revoluționat poşta electronică şi tehnologia Web comunicațiile.

Primele rețele GC au fost proiectate pentru Organizația Europeană pentru Cercetări Nucleare - laboratoarele de fizică, de unde a pornit şi realizarea WWW - World Wide Web.

Acum, rețelele GC pătrund în practică şi schimbă modalitatea de structurare a afacerilor şi felul în care oamenii trăiesc, învață sau lucrează. Furnizând utilizatorilor putere de calcul şi cantitați uriaşe de informații, rețelele GC vor îmbunătăți competitivitatea afacerilor existente şi vor deschide piețe şi servicii până acum crezute imposibile. Impactul lor asupra calității vieții noastre va fi profund, permițandu-ne să urmărim şi să modelăm mai bine orice, de la schimbările climaterice de pe Tera la evoluția inflației.

Rețelele GC vor permite cercetătorilor şi oamenilor de afaceri să împartă mai bine resursele de toate felurile şi cunoştințele pe care le dețin. Ele sunt tehnologii importante pentru realizarea "strategiei Lisabona", stabilită în anul 2000, care îşi propune să transforme Uniunea Europeană în cea mai competitivă şi informată economie din lume până în anul 2010. De aceea, tehnologia GC este un obiectiv strategic în cercetarea din societatea bazată pe cunoştințe, fondată de UE în al şaselea program cadru (FP6) pentru cercetare şi dezvoltare tehnologică.

Domenii de aplicabilitate

Rețele GC au fost concepute inițial pentru rezolvarea problemelor care includeau cantitați mari de informație şi calcule intense. De aceea, primele astfel de rețele au fost dezvoltate la nivel global pentru proiecte ştiințifice şi de cercetare, acestea necesitând puterea de calcul a unor supercalculatoare şi echipe de lucru cu o înaltă pregãtire tehnicã, echipe rãspândite în mai toate universitãțile lumii, centre de cercetare şi companii. Impactul tehnologiei GC asupra afacerilor este deja simțit iar în viitor va fi masiv. Unele sectoare de activitate cum ar fi: medicina, ingineria, divertismentul se confruntã de asemenea cu schimbãri radicale, care vor fi adaptate la tehnologia GC.

Avantajele

Pânã la rețelele GC, singurele organizații care-şi puteau permite supercalculatoare erau acelea cu bugete mari şi cu necesitați de tehnicã de calcul semnificative, permanente. Rețelele Grid schimbă radical situația: orice organizație conectată la o astfel de rețea îşi va putea permite utilizarea unui supercalculator virtual atunci când are nevoie, plătind o taxă de acces. De exemplu, o echipă de intervenție rapidã, ar putea sã se conecteze la o rețea GC pentru a analiza informații din mii de senzori montați în bazinele râurilor, în vârfurile munților şi chiar şi pe orbite. Se poate astfel analiza, modela şi compara aceste informații în timp util pentru a previziona catastrofele naturale: inundații, avalanşe, uragane, cutremure, erupții vulcanice.

Atunci când apare un dezastru, echipa poate folosi rețeaua GC pentru a prezice felul în care situația s-ar extinde şi pentru a optimiza alocarea resurselor. Acest lucru este posibil pentru cã rețeaua GC poate furniza oricând comunicarea şi puterea de procesare necesare precum şi informațiile dorite la prețurile pe care echipa de intervenție şi le poate permite.

Sectoarele de producție

Rețelele GC pe de o parte permit utilizatorului sã acceseze resursele de care are nevoie – date, aplicații, software, hardware - iar pe de altă parte dau voie utilizatorului sa-şi punã la dispoziție propriile resurse pentru alți utiliuatori. Acest lucru înseamnă cã utilizatorii pot lucra mai aproape unii de alții, chiar dacã sunt în diferite organizații, din diferite țări sau din diferite continente. Impactul ar putea semnificativ în sectoarele de producție europene, în care echipe din companii diferite vor putea lucra împreunã la realizarea proiectelor comune. De exemplu, pentru activitatea de proiectare a unui tip nou de masinã. Chiar dacã aceastã activitate este condusã de fabricantul de maşini, ea mai include proiectanți, economişti, ingineri, informaticieni, furnizori. Pãrțile diferite ale unui proiect sunt interdependente pentru cã fac parte dintr-un tot unitar care este maşina. Echipe diferite trebuie sã munceascã împreunã, integrând sistematic modelele şi informațiile în proiectul de ansamblu şi testând rezultatul obținut. Din punct de vedere al securitãții, managerul de proiect nu poate sã permitã tuturor membrilor acces la întregul proiect, deci se lucrează în achipă. Rețeaua GC poate ajuta permițând informațiilor şi modelelor sã fie integrate uşor, la timp, păstrându-se securitatea proiectului. Informațiile obținute în urma testelor de încercare la impact pot fi vãzute de fiecare furnizor, care poate folosi puterea de calcul a rețelei Grid pentru a analiza performanțele componentelor. Rezultatul este un proiect şi nişte produse, îmbunatațite, obținute cu un efort mai mic şi mai puțin costisitor. De aici rezultă o înaltã competitivitate, locuri de muncã noi, produse de calitate superioarã.

Alte sectoare de activitate

Avantajele tehnologiei GC nu se vor extinde practic la toate domeniile de activitate. Companii din sectoare diferite – afaceri, bănci, asigurări, media, servicii - vor putea folosi mediul de lucru virtual creat prin rețeaua GC pentru a-şi folosi eficient propriile resurse de calcul şi pentru a colabora mai eficient cu partenerii lor.

Iatã în continuare câteva exemple. Bibliotecile şi arhivele din presă, sunt în prezent indexate semantic în baze de date uriaşe localizate pe întregul glob. Cu tehnologia GC, redactorii vor putea sã caute în aceste arhive, sã extragã informații şi sã trimitã noul conținut într-un mod

rapid şi flexibil. De asemenea, moştenirea culturală a lumii, care este în prezent stocată în baze de date de pe întrega planetã, va fi accesibilã tuturor. Comerțul electronic, va putea sã se organizeze şi sã funcționeze mai bine din punct de vedere al aprovizionării cu produse, al relației privind clienții. Băncile şi asigurărilor vor putea realiza analize statistice şi demografice mult mai performante, care să le permită noi politici pentru a atrage cât mai mulți clienți. De altfel, aşa cum se ştie, marketingul actual este orientat pe client şi nu pe produs aşa cum era până acum. Acest lucru înseamnă că orice activitate care are printre actori clientul va profita din plin de rețeaua GC.

Sistemele de baze de date

Toate aspectele de mai sus care privesc tehnologia GC au ca rezultat, pentru utilizator, obținerea unor informații care să-l ajute să-şi fundamenteze deciziile operative sau de conducere şi să-şi desfăşoare activitățile curente. Ori această cantitate mare de informații nu poate fi stocată şi apoi accesată corespunzător decât dacă se lucrează cu un sistem de baze de date evoluat, care să funcționeze în arhitectură GC. Din acest motiv, SBD actuale au început să integreze tot ce este mai nou ca tehnologii informatice, inclusiv tehnologia GC. Primul SGBD care poate lucra cu o astfel de tehnologie este Oracle versiunea 11g (litera *g* vine de la Grid).

Enterprise Grid Computing (EGC)

EGC înseamnă procesul de punere la *lucru împreună* a mai multor calculatoare existente la nivelul unei organizații şi funcționarea ca un sistem integrat.

EGC presupune existența unui *produs software* care:

- face eficientă folosirea mai multor servere;
- asigură stocarea modulară a datelor;
- permite comutarea lucrului;
- permite suplimentarea capacității de lucru la cerere.

Aducând noi servere în această grilă de calculatoere, mai mari sau mai mici, obținem *performanță* la un preț scăzut, deoarece gestiunea resurselor de calcul se face unificat, simplu, eficient, ieftin.

Tratând orice GC ca un *serviciu virtual*, sistemele inteligente (business intelligence) pot optimiza utilizarea resurselor de calcul la nivelul întregii organizații.

EGC *combină două concepte* care sunt în lugătură unul cu celălalt:

- implementare de tip *unul din mulți* (one from many) = folosirea mai multor clustere (fizice) pentru a crea o singură entitate logică (exemplu pentru o bază de date). Deoarece o singură entitate logică este implementată peste mai multe calculatoare, se pot adăuga noi capacități în mod incremental (on-line) sau la cerere (on-demand), obținându-se astfel flexibilitate;
- administrare de tip *mulți ca unul* (many as one) = administrarea mai multor calculatoare ca un grup, adică o singură entitate logică (exemplu un grup de instanțe a unei BD se administrează ca o entitate logică – tabela).

Capabilitatea unui Server de aplicații de a *distribui sarcini* unui nou server, *fără a opri* activitatea sa necesită un *software inteligent*. În acest sens, Inteligența Afacerii ca tehnologie informatică a existat în SGBD înainte de GC şi a produs software inteligent.

În acest scop, organizația ar trebui să aibă în vedere *următoarele aspecte*:

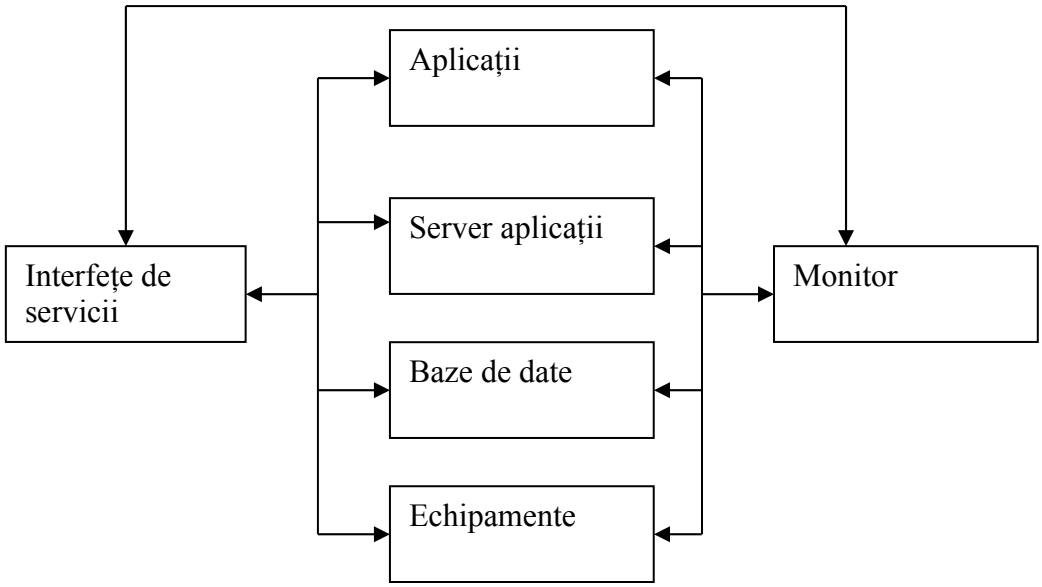
- *integrarea* informațiilor în centre de date, a datelor în BD, a aplicațiilor în Server de aplicații;
- *standardizarea* în două variante: standard de producător (promovează produse proprientar), standard internațional (promovează interoperabilitatea – open standard);
- *expertiza în Unix* car este un SO perfect pentru o configurație de GC.

Facilități Grid Computing

1. *Virtualizarea* (virtualization) este asigurată de un serviciu de introducere în GC a fiecărei entități logice sau fizice. Acest lucru permite componentelor GC (discuri, procesoare, BD, aplicații etc.) să se integreze firesc fără a crea fragilitate în sistem. În afară de faptul că stabileşte diferite legături între componentele GC, virtualizarea permite fiecărei componente GC să reacționeze mai repede la schimbarea contextului de lucru şi şă se adapteze repede dacă unele componente au căzut, fără a compromite performanțele sistemului în ansamblu.
2. *Furnizarea dinamică a resurselor* (dynamic provisioning) înseamnă distribuirea resurselor acolo unde este nevoie de ele. În contextul GC, resursele pot semnifica: cererile adresate serverului – care trebuie tratate, datele – care trebuie accesate şi prelucrate, calculele – care trebuie rezolvate. Există un serviciu de administrare (broker service grid) care cunoaşte necesitățile componentelor GC şi resursele disponibile, le asociază logic în mod dinamic, făcând astfel eficient întreg sistemul. Ulterior, el poate ajusta aceste asocieri la schimbările contextuale.
3. *Punerea în comun a resurselor* (resource pooling) este necesară în GC pentru a obține o mai bună utilizare a resurselor şi pentru a realiza costuri mici.Prin plasarea unor discuri sau unor servere într-un ansamblu comun, procesele executabile ale GC au o mare flexibilitate în optimizarea relației nevoi-resurse. Partajarea resurselor se realizează prin software.
4. *Adaptarea automată a sistemelor* (self adaptive systems) are ca efect automatizarea (reducerea) şi simplificarea sarcinilor administratorilor. O infrastructură GC ar fi impracticabilă dacă fiecare nod din grilă ar necesita intervenție manuală în mod constant. Multe din sarcinile care erau executate de administratori acum sunt tratate de software.
5. *Managementul unificat* (unified management) pentru întreținerea EGC semnifică faptul că sarcinile administratorilor sunt simplificate şi integrate *într-un singur instrument* care poate monitoriza, furniza şi administra fiecare componentă din grilă. Un astfel de instrument evaluează disponibilitatea şi performanțele fiecărui nod, astfel încât la orice gâtuire din sistem sau pentru orice componentă indisponibilă, se emite un mesaj. În GC administratorul tratează grupuri de componente ca pe o singură entitate logică, astfel încât sarcinile sunt definite o dată şi executate de mai multe ori. Managementu unificat implementează două concepte: *unul din mulți* (îmbină facilitățile: virtualizare, furnizare dinamică, punerea în comun a resurselor), *mulți ca unul* (Îmbină facilitățile: adaptarea automată a sistemelor, management unificat).

Arhitectura GC

Arhitectura GC evidențiază grafic diferitele componente şi legăturile dintre ele. Structurarea se face pe patru niveluri (aplicații, server de aplicații, baze de date, echipamente) conectate prin mecanisme de interfață de servicii şi monitorizare (control).



a) *Interfața de servicii/comunicare* (grid service interface). Conexiunea între straturile GC este dinamică, ajustându-se la momentul execuției, optimizând performanțele şi adaptându-se căderilor unor componente. Interfața presupune un mecanism de comunicare între toate componentele GC, care cunoaşte sau descoperă nevoile consumatorilor de resurse şi rezolvă aceste nevoi, asociindu-le cu furnizorii de resurse.
b) *Monitorul* (grid control). Monitorul este un produs de management puternic care permite administratorului să supravegheze toate activitățile din GC. El are un mecanism integrat care intervine atunci când este necesar, oriunde şi oricând în GC.
c) *Echipamentele* (hardware). Resursele hardware (discuri, procesoare, componente de rețea etc.) pot fi în mod dinamic alocate de către o BD conform anumitor politici şi programări , folosind un mecanism orientat pe servicii (service-oriented). Nivelurile înalte ale GC, ca de exemplu BD sau Servere de aplicați, nu trebuie să se preocupe de amănunte (unde se face stocarea, unde se face conectarea etc.). Interfața de comunicare tratează detaliile legate de conectarea serverelor şi BD cu echipamentele de care au nevoie.
d) *Baza de date* (database). BD sunt atât consumatori de resurse cât şi furnizori de resurse într-o arhitectură GC. Pentru resursele hardware, BD sunt consumatori de resurse, iar pentru serverele de aplicații sunt furnizoare de resurse. La nivelul BD din GC, punerea la un loc a resurselor duce la existența unei singure BD, la nivel logic, sau unui număr mic de BD, fiecare putând consta în mai multe instanțe care se execută pe clustere. Sarcinile sunt împărțite optim pe instanțele BD, separate pe calculatoare diferite. BD se adaptează în mod automat la căderea unei instanțe sau la adăugarea unei noi instanțe pe un cluster.
e) *Server de aplicații* (application server). Servere de aplicații sunt consumatoare de resurse hardware şi resurse pentru BD, furnizând mediul de execuție al aplicațiilor. În GC resursele de tip server de aplicații sunt puse în comun, similar cu BD. Un singur Server de aplicații constă dintr-o multitudine de instanțe care rulează pe calculatoare diferite, dar având lucrul coordonat şi folosind algoritmi specifici. Dacă o instanță cade atunci clientul este dirijat spre altă instanță disponibilă.
f) *Aplicații* (applications). Pentru a beneficia de tot avantajul tehnologiei GC, aplicațiile ar trebui să-şi facă cunoscut comportamentul lor către alte aplicații, prin intermediul interfețelor

standard, astfel încât funcționalitatea de la alte aplicații să se poată uşor combina pentru implementarea logicii afacerii. *Serviciul de Web* (web service) este termenul folosit pentru acest nivel, care permite folosirea coordonată a diferitelor aplicații.

2.5.3 Oracle 11g

Oracle este *prima companie* care a promovat tehnologia Grid Computing (GC), într-un SGBD, pentru întreprinderi. Astfel, Oracle oferă o infrastructură software completă şi comercială, proiectată special pentru GC cu baze de date. Oracle 11g permite *adaptarea sistemelor informatice* la modelul Enterprise Grid Computing (EGC), care foloseşte puterea de prelucrare a unui mare număr de calculatoare, puțin costisitoare, ce acționează ca un tot unitar. De la funcții de stocare a datelor, servere de BD, servere de aplicații (arhitectura NC), la administrarea datelor, *cele mai noi tehnologii Oracle* acceptă toate cerințele de calcul ale unui sistem GC.
Produsele: Oracle Database 11g, Oracle Developer Suite 11g, Oracle Application Server 11g, Oracle Enterprise Manager 11g etc. formează împreună prima platformă completă pentru infrastructura EGC.

Câteva ***atribute/facilități*** pe care le oferă sistemul Oracle 11g:
- gestionează *mai multe servere* ca pe o singură maşină;
- *întreținerea şi dezvoltarea* se realizează ca şi cum ar fi un singur sistem;
- *alocarea resurselor* se face în funcție de puterea de calcul solicitată de aplicații (acum, multe companii folosesc doar 50-60 la sută din resursele de calcul pe care le dețin, pentru că acestea nu sunt integrate);
- *creşte gradul de automatizare*, precum şi productivitatea muncii (se oferă valoare de utilizare tuturor resurselor de calcul);
- *administrarea* aplicațiilor este mult *mai simplă*, iar dezvoltarea acesteia este mult mai uşoară;
- se permite folosirea unei *infrastructuri grilă* (GC) pentru aplicații comerciale la nivelul întregii întreprinderi (pregătirea a fost făcută în Oracle 8i şi unde integrarea era cuvânt de ordine);
- *se reduc* mult (până la 50 %) *costurile de gestiune a BD* şi se oferă cea mai uşoară administrare a unei BD. Succint, administrarea unei BD Oracle cu Enterprise Manager 11g, se poate face: *realist* (ține cont de ceea ce există), *proactiv* (foloseşte integrat resursele de calcul), *inteligent* (rezolvă singur foarte multe situații);
- *foloseşte rațional* o infrastructură existentă de tehnică de calcul.

Implementarea tehnologiei GC în Oracle 11g presupune asigurarea de către sistem a următoarelor aspecte:

1. Virtualizare

Se permite adăugarea sau scoaterea discurilor de stocare a datelor, cu păstrarea *on-line* a aplicațiilor. Performanțele serverului sunt puse la dispoziția mai multor aplicații şi baze de date, într-o structură eficientă de tip *cluster*. Resursele serverelor se alocă în funcție de necesitățile utilizatorului, folosind tehnica *load balancing*.

2. Cost redus

Se poate rula pe platforme diverse, inclusiv servere şi echipamente de stocare a datelor cu un cost redus, oferind aceleaşi funcționalități indiferent de platforma aleasă. Stocarea datelor se realizează folosind componenta ASM - Automatic Storage Management, care permite utilizarea echipamentelor de cost redus. Administratorul BD alocă discurile de stocare componentei ASM care se ocupă de managementul ulterior, oferind performanțe optime, fără a necesita intervenția umană.

3. Scalare

Se poate porni de la o aplicație pe două sau trei straturi pe un grup de servere de cost redus, continuându-se apoi pe mai multe straturi (n-tired) prin includerea unor aplicații noi şi a unor echipamente noi.

4. Administrare GC

Oracle 11g are un produs special pentru administrarea GC, denumit Oracle Grid Control. Acesta poate fi utilizat de administratorul BD pentru monitorizarea şi actualizarea întregii infrastructuri de SBD, care include resurse eterogene, distribuite geografic. Resursele nu mai sunt administrate individual, aşa cum se realiza în mod tradițional, ci grupat, prin utilizarea unui navigator - *browser Web*. Astfel, se poate realiza managementul tuturor resurselor: servere de aplicații, servere de baze de date, servere de securitate - tip *firewall*, echipamente de stocare, echipamente de rețea.

5. Baze de date distribuite

Oracle 11g introduce conceptul de rulare a unei singure baze de date pe multiple servere folosind tehnologia Real Application Clusters - RAC. Acest lucru înseamnă că mai multe servere sunt folosite în mod optim de către mai multe aplicații, reducându-se numărul de echipamente şi licențe necesare. Tehnologia de tip cluster nu trebuie să fie achiziționată de la un producător diferit, introducându-se componenta Oracle Portable Clusterware care permite folosirea facilităților de tip cluster pe orice tip de hardware.

6. Sisteme informatice integrate

Componenta Oracle Streams 11g oferă funcționalități multiple pentru realizarea unei soluții integrate de sistem informatic, cu eliminarea reduntanței.

7. Securitate

Oracle Enterprise User Security este o componentă de management centralizat al drepturilor de acces ale utilizatorilor. Un utilizator este creat o singură dată, putând avea acces la multiple baze de date existente în GC.

Oracle Identity Management centralizează managementul autentificării şi autorizării utilizatorilor în cadrul unei soluții integrate, prin intermediul componentei Oracle Internet Directory.

Exemplu de implementare GC în Oracle 11g

Managementul *GC* este inclus în Oracle prin implementarea unei componente distincte denumită *Grid Control 11g Management Service*. De asemenea, *Oracle Entreprise Manager 11g Grid Control* monitorizează întregul sistem de baze de date integrat dezvoltat folosind platforma Oracle. În cele ce urmează se prezintă etapele care trebuie parcurse pentru lucrul în mediul GC în Oracle 11g.

Pregătirea mediului de lucru presupune parcurgerea următorilor **paşi** (fig.1)**:**

1. Crearea *aplicației* - *middle tier* pe o anumită maşină, ceea ce înseamnă instalarea:
   a. *Grid Control 11g Management Service*
   b. *Grid Control 11g Management Agent*
2. Crearea *bazei de date* - *repository* pe aceeaşi/altă maşină, ceea ce înseamnă instalarea:
   a. Bazei de date
   b. *Grid Control 11g Management Agent*
3. Crearea *agenților* prin instalarea pe fiecare maşină din sistem a următoarelor componente:
   a. *Additional Management Agent*

b. Enterprise Manager Patch

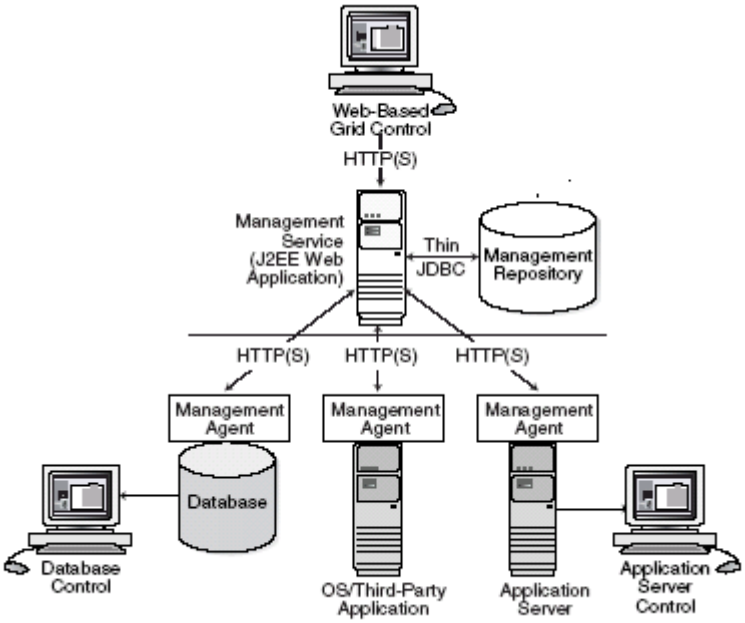


Figura 1. Arhitectura GC în Oracle 11g

Conectarea la *Oracle Entreprise Manager 11g* – User Name şi Password, presupune lansarea componentei Grid Control, ceea ce înseamnă utilizarea tehnologiei GC. Pe pagina principală care apare, se găseşte un meniu orizontal cu opţiuni care dau operaţiile ce pot fi executate sub produsul Oracle Entreprise Manager 11g. Câteva dintre aceste opţiuni sunt prezentate în continuare.

Managementul resurselor (fig. 2) presupune identificarea resurselor şi a performanţelor sistemului, a acţiunilor planificate, a eventualelor atenţionări - alerte. Se pot monitoriza diferite componente ale sistemului de baze de date: produse Java – BC4J, server http, fişiere de parametri - Listener, baze de date - Database, servere de aplicaţie – Application Server, aplicaţii Web – Web Application, agenţi - Agent.

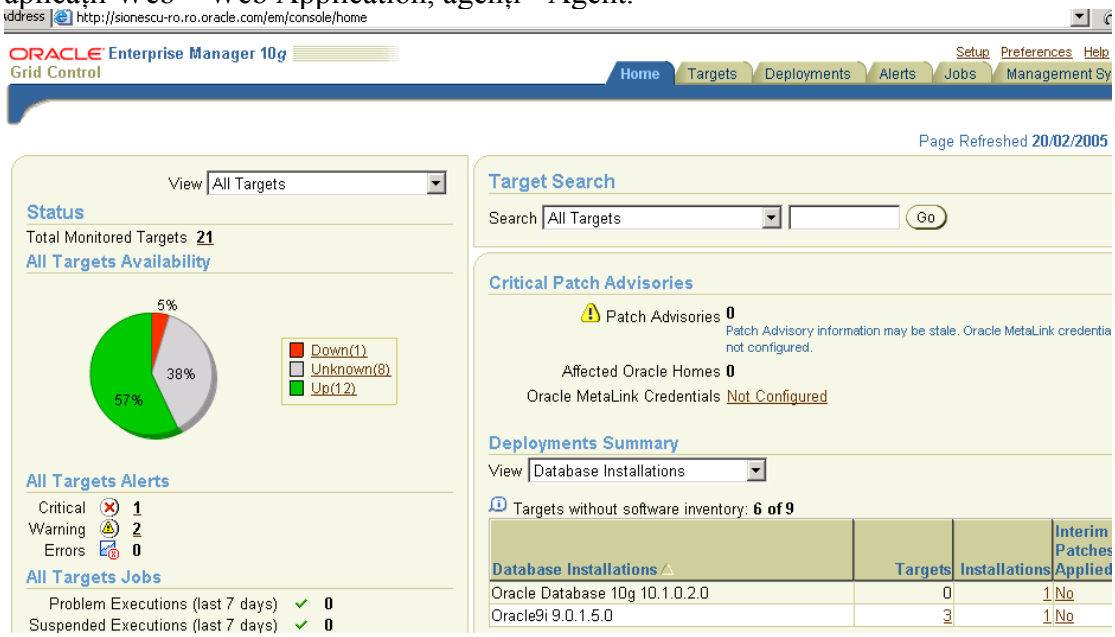


Figura 2. Situația curentă a resurselor sistemului

Gruparea resurselor – *Groups* are rolul de a se realiza un management unitar pentru toate componentele unui grup – de exemplu grupul facultăți conține CSIE, CIG, REI.

Atenționările - alertele pot fi de mai multe tipuri: resursele nedisponibile - cu precizarea momentului indisponibilizării şi mesajului de eroare aferent, erorile apărute, mesajele de avertizare etc.

Managementul sarcinilor - *jobs* implică următoarele operații: căutarea, adăugarea, modificarea, ştergerea, suspendarea unei anumite sarcini sau a unei anumite rulări a unei sarcini. Se pot crea – adăuga sarcini de tipul: aplicarea unui şablon - *patch*, clonarea unei aplicații, salvarea bazei sau aplicației - *backup*, rularea unui modul de program – *bloc PL/SQL, script* SQL. Acestea pot fi planificate să ruleze cu o anumită periodicitate, pe toate resursele sau pe un anumit grup de resurse.

Rolurile permit accesul anumitor utilizatori la anumite resurse şi descriu drepturile aferente. Cu ajutorul unor viziuni, care inclus şi o scurtă descriere, pot fi vizualizate toate aceste aspecte.

Monitorizarea evenimentelor de pe fiecare resursă presupune vizualizarea – în formă tabelară şi/sau grafică, a următoarelor informații:

- parametrii setați (listener): dacă sunt disponibili sau nu;
- problemele de rețea: dacă sunt şi care anume;
- evidență agenți: disponibilitate, agenți care nu mai răspund;
- monitorizare servere de baze de date: disponibilitate, procent de utilizare a spațiului, depăşiri ale spațiului utilizat de tabelele de spațiu (*tablespace*)
- monitorizare performanțe aplicații Web: starea, procent de disponibilitate, alerte, rezolvarea tranzacțiilor;

Regulile de notificare prin E-mail pot fi create şi vizualizate atunci când apar condiții critice pentru diferite componente introduse în sistemul integrat. De exemplu, se poate crea o notificare pentru o bază de date de câte ori aceasta se opreşte sau ori de câte ori anumiți indicatori devin critici - *Datafile Usage %, Archiver Hung Error Stack, Tablespace Space Used %.*

2.5.4 Baze de date în arhitectură Grid Computing (GC)

Tehnologia GC oferă servicii prin care se concentrează puterea de calcul de la nivelul unei organizații în mod eficient şi sigur, inclusiv stocarea datelor, în rețea de calculatoare conectată la Internet.

Tehnologia GC a apărut în USA la sfârşitul anilor ’90 şi a fost preluată în 2003 pentru baze de date, în infrastructura Oracle. Rolul acesteia este de a concentra la nivelul unei companii o putere de calcul foarte mare, pe care niciun calculator, oricât de puternic ar fi, nu o poate asigura. Mai mult, această putere de calcul este monitorizată eficient de infrastructura software pentru baze de date, evitându-se timpii morți şi sugrumarea traficului în rețea. În acest fel, resursele de calcul vor fi uriaşe, inclusiv spațiul de stocare pentru date, de ordinul petabytes (PB), gestionat eficient de SGBD.

Apariția şi dezvoltarea tehnologiei GC s-a bazat pe existența altor tehnologii informatice: bazele de date avansate, Internet, platforma Java, tehnologia orientată obiect etc.
GC oferă o infrastructură care presupune utilizarea integrată şi colaborativă a resurselor de calcul , a datelor, a rețelelor de calculatoare la nivelul organizației. Acest lucru presupune o investiție mare, iar astfel de proiecte sunt acum realitate în toată lumea, inclusiv în Europa. Aceste proiecte pot fi grupate în *două categorii*:

- dezvoltarea GC ca infrastructură cu hardware, software corespunzător, precum şi cu un mecanism administrativ corespunzător;
- cercetarea GC ca middleware ce investighează platformele ce pot monitoriza eficient resursele de calcul la nivelul organizației.

Rezultă că GC presupune, prin definiție, *integrare şi interoperabilitate* de tehnologii informatice. În aceste condiții, pentru ca tehnologia să funcționeze practic şi să se poată dezvolta ca un sistem deschis este necesară o cât mai mare standardizare. Informatica suferă încă de lipsa unor standarde şi de aceea noile tehnologii țin cont de acest lucru. Pentru tehnologia GC eforturile de *standardizare* sunt coordonate de organismul Global Grid Forum – GGF. In plus, marii furnizorii de aplicații informatice în arhitectură GC (Oracle, EMC, HP, Dell, Intel etc.) au inființat în 2004 *Enterprise Grid Alliance – EGA*, care promovează tehnologia GC dar şi emit standarde şi specificații în domeniu.
EGA se preocupă şi de *securitatea* aplicațiilor informatice în arhitectură GC formulând în acest sens un set de reguli pe care trebuie să le respecte producătorii şi utilizatorii acestor aplicații. O parte dintre reguli sunt valabile şi pentru aplicațiile în arhitectura client-server sau în arhitectura centralizată, dar problemele de securitate sunt mai numeroase pentru aplicațiile în arhitectură GC pentru că sunt mai multe posibilități de acces, din mai multe aplicații, la aceleaşi resurse.

Bazele de date distribuite, cu exemplul elocvent de implementare în Oracle 11g RAC (Real Application Cluster), se pare că vor evolua, alături de rețelele distribuite, prin integrarea tehnologiei GC.
Dacă tehnologia bazelor de date o integrează acum pe cea GC, aceasta integrează la rândul ei mai multe metode privind evoluția *tehnologiei calculului*:

- calculul distribuit – pentru a avea o putere de calcul suficient de mare se conectează resursele de calcul într-o rețea de calculatoare utilizată ca o resursă unică, integrată;
- metacalculul – legarea centrelor de supercalculatoare în rețele de mare viteză, ca un caz particular de calcul distribuit;
- calcul cluster – conectarea unor calculatoare heterogene între ele, obținând o putere de calcul mai mare decât a unui calculator mare (main-frame). Avantajul major este scalabilitatea: pot fi adăugate la un cluster existent alte calculatoare;
- calcul Peer to Peer – prin descărcarea unei aplicații dintr-o rețea pe calculatorul personal, un utilizator se va conecta cu toți utilizatorii care care au aceeaşi aplicație. Utilizatorii vor specifica ce informații sunt publice şi acestea vor fi partajate între ei. Astfel, se pot partaja informații în rețea între utilizatori fără să eiste un server central;
- calcul Internet – cu ajutorul Internetului se pot construi supercalculatoare virtuale care lucrează cu nodurile (grid) simultan pe mai multe părți ale problemei de rezolvat. Aceste noduri descarcă datele necesare de pe Internet, le procesează şi apoi le încarcă în sistemul central pentru post-procesare;
- grid computing local – un produs software de tip middleware, care nu este nici aplicație informatică nici sistem de operare ci ceva intermediar. Produsul verifică ce calculatoare din rețea sunt disponibile pentru rularea, în mod optim a unei aplicații. Şi aici există scalabilitate, adică se pot adăuga noi calculatoare.

Aplicațiile cu baze de date care integrează noi tehnologii informatice, inclusiv GC, vor implementa aspectele fundamentale din tehnologiile respective. In cazul GC *conceptele fundamentale* care sunt implementate în aplicațiile care sunt dezvoltate cu Oracle 11g sunt:

- partajarea resurselor – resursele aflate la distanță, necesare unei aplicații, sunt utilizate în comun (schimb de fişiere, acces direct la date, acces direct la alte aplicații etc.);
- accesul sigur – pentru a se avea acces partajat la anumite resurse se impun anumite restricții: politica de acces, autorizarea, autentificarea;
- accesul în timp util – indiferent de numărul de resurse existente utilizatorul va avea uneori de aşteptat. Mecanismele de alocare şi cele de acces au rolul de a rezolva eficient cozole de aşteptare. Experiența de la tehnologia bazelor de date este deosebit de utilă pentru acest aspect;
- dispariția distanțelor – performanțele mereu crescânde ale mediilor de comunicație în rețea fac acum posibilă conectarea unui număr mare de utilizatori şi transmisia unor volume uriaşe de date;
- standardele – pentru ca tehnologia GC să fie recunoscută şi utlizată la nivel global, în mod sigur, este nevoie să fie standardizate cât mai multe elemente. Există organisme de standardizare pentru tehnologia GC şi o propunere de arhitectură standard denumită Open Grid Services Architecture – OGSA;
- sistemul deschis – proiectele GC sunt construite pe baza unor protocoale şi servicii standarde , ca o infrastructură de sistem deschis (open-source), capabilă să preia din mers schimbările care apar.

2.6. Aplicabilitatea bazelor de date care integrează tehnologii informatice

Sistemele de baze de date actuale, prin integrarea celor mai noi tehnologii informatice, inclusive GC oferă soluții eficiente pentru dezvoltarea aplicațiilor informatice din toate domeniile de activitate. Dintre aceste domenii, cel economic se pretează cel mai bine pentru că el oferă cele mai multe tipuri de aplicații, de la dimensiuni mici până la dimensiuni foarte mari. Folosirea în comun a resurselor de calcul la nivelul unei companii, în mod eficient şi sigur, este o prioritate în condițiile globalizării economiei. În acest sens, în anii 2000 au dezvoltat proiecte de cercetare majoritatea marilor firme de informatică: Oracle, IBM, Microsoft etc.

Aplicațiile cu baze de date în arhitectură GC țin cont de următoarele *aspecte:*

- asigură un management distribuit eficient;
- menține controlul integrat asupra resurselor de calcul;
- asigură disponibilitatea comună a datelor;
- identifică soluțiile optime de acces la date;
- creşte productivitatea de lucru pentru beneficiar;
- asigură un mediu de lucru prietenos cu acces la numeroase facilități.

Domeniile de aplicabilitate ale aplicațiilor cu baze de date cu un mare grad de integrare a tehnologiilor informatice, aşa cum sunt şi cele de tip GC, sunt dintre cele mai numeroase. Acest lucru este posibil datorită interferenței tehnologiilor informatice care oferă aplicațiilor rezultate: o mare deschidere, o flexibilitate deosebită, interoperabilitate, eficiență în utilizarea şi gestionarea resurselor de calcul, o putere de calcul uriaşă. Astfel, marketingul orientat pe client are o influență deosebită privind dezvoltarea aplicațiilor informatice cu baze de date pentru afacerile organizațiilor.

In momentul actual al noului context economic, în lume sunt *două direcţii* de teorie şi practică de marketing şi economică, în general:

- scandinavă, care pune accentul pe metode calitative;
- americană, care pune accentul pe metode statistice superioare – Data Mining, Data Mart, Data Warehouse, OLAP etc.

În Europa se folosesc ambele variante, şi acelaşi lucru se poate spune despre ţara noastră, ca membru al Uniunii Europene. Specificul nostru constă în faptul că majoritatea firmelor fac parte din IMM – Întreprinderi Mici şi Mijlocii. Acest lucru înseamnă că: flexibilitatea economică este mare, concurența cu firme străine este slabă, pot apărea conflicte interumane. Soluțiile pentru informatizarea organizațiilor, în special pentru firmele mari, atât la noi în țară cât şi în lume, merg pe varianta americană.

Gradul de *răspândire* al tehnologiei GC în lume este aproximativ acelaşi pe zonele geografice America de nord, Europa, Asia Pacific şi anume în jur de 5, pe o scară de la 0 la 10. Studiul popularității tehnologiei GC a fost făcut de organismul Oracle Grid Index – OGI. Un rol important în răspândirea acestei tehnologii l-au avut aplicațiile cu baze de date suport pentru afaceri: Enterprise Resource Planning – ERP, Customer Relationship Management – CRM, Business Intelligence – BI etc.
Aşa cum arătam mai sus, Enterprise Grid Alliance – EGA este unul dintre organismele care promovează tehnologia GC. El sprijină *două clase de aplicații* cu baze de date în arhitectură GC:

- aplicații comerciale, care sunt fundamentale pentru informatizarea oricărei organizații (ERP, CRM, BI);
- aplicații specifice, care au un rol important privind eficiența organizației (analiza financiară, analiza datelor etc.).

Producătorii de aplicații în arhitectură GC, în special cei din grupul EGA (în frunte cu Oracle care a furnizat prin versiunea 11g primul SGBD ce funcționează şi în arhitectură GC), au *dezvoltat în comun proiecte* în acest sens, datorită interferenței de numeroase tehnologii informatice, dar şi datorită costurilor ridicate. Un astfel de proiect este MegaGrid în care fiecare companie participantă pune la dispoziție propriile tehnologii pentru construirea unei infrastructuri GC: Oracle oferă infrastructura Oracle 11g, Dell oferă servere, Intel oferă procesoare, EMC sisteme de stocare. Se pot astfel dezvolta şi rula aplicații informatice cu baze de date performante şi sigure în arhitectură GC.
În țara noastră sunt mai multe proiecte GC dezvoltate şi promovate în mediul universitar informatic şi sprijinite de CNCSIS. Scopul este de a se dezvolta şi exploata o infrastructură eScience în România, care să colaboreze cu infrastructurile similare din Uniunea Europeană.

Aplicabilitatea bazelor de date care integrează tehnologii informatice este influențată de noul contextul economic actual care, la rândul lui, este determinat de tehnologiile avansate. Acest lucru înseamnă că asistăm la schimbarea modului de afaceri, a relațiilor interumane, a strategiei de marketing etc. Actualele aplicații informatice cu baze de date trebuie să țină cont de aspectele care urmează.

Principalele *tehnologii avansate* care determină acum şi noul context economic sunt:

- echipamentele electronice complexe, inclusiv calculatoarele;
- asistenții de proiectare pentru realizarea a tot felul de produse;
- bazele de date de mărime şi complexitate diferite;
- serviciile de diferite tipuri – SOA (Service Oriented Architecture);
- tehnologia Web

- tehnologia Cloud Computing.

Ținând cont de noul context economic se poate construi noua *strategie de marketing* care se bazează pe orientarea pe client şi care are în vedere:
  - noi clienți de diferite tipuri;
  - noi produse şi servicii;
  - noi metode de analiză şi descoperire;
  - noi tehnici de prelucrarea a informației.

Pornind de această strategie de marketing, în contextul economic actual este de mare importanță ceea ce doreşte clientul. Clientul domină în acest moment viața de afaceri, iar aplicațiile cu baze de date țin cont de cerințele clientului şi sunt destinate acestuia. În sinteză, *dorințele clientului* sunt:
- recunoaşterea: clientul vrea să fie recunoscut ca individ, cu dorințele şi preferințele sale;
- serviciul: furnizarea spre client a unor servicii cunoscute şi dorite de el, prin acces la baza de date;
- accesibilitate: clientul este ocupat şi deci pentru afacere este bine să folosească doar un profil al său – nume, adresă, card bancar, Email etc. – recunoscut şi, de asemenea, să aibă un istoric al său;
- ajutorul total: orice îi va face viața mai uşoară va fi binevenit pentru client;
- informația: clientul foloseşte tot mai mult calculatorul şi Internetul. Acum informatica este poate chiar mai importantă pentru client decât produsul însuşi;
- identificarea: clientul trebuie să identifice produsele şi companiile. Acestea vor fi prezentate clienților cât mai prietenos, sugestiv şi apropiat de cerințele lor. Se ține cont de categoriile de clienți, de zona geografică etc.

Un alt aspect de care trebuie să țină cont aplicațiile cu baze de date în noul context economic este *importanța prețului*. În acest sens, sistemele de baze de date trebuie să asigure:
  - metodele pentru păstrarea şi atragerea clienților;
  - furnizarea informațiilor necesare prin servicii client;
  - produsele şi serviciile la modă destinate preferințelor individuale;
  - programele de marketing personalizate;
  - deschiderea companiei spre client;
  - purtarea unui dialog unu la unu cu fiecare client;
  - întocmirea unor liste de clienți loiali şi programe pentru atragerea altora;
  - clasificarea clienților după diferite criterii – interese, profitabilitate etc.;
  - recunoaşterea clienților după nume şi apelarea lor pentru socializare;
  - devize pentru programe de marketing pentru noi prospecțiuni.

La realizarea actualelor aplicații cu baze de date se va ține cont de următoarea *strategie*:
- atragerea şi menținerea clienților folosind metode, tehnici şi strategii specifice;
- centrarea întregii strategii pe client şi adaptarea rapidă a aplicației conform cerințelor acestuia;
- determinarea riscului pentru aplicația cu baze de date;
- acordarea unor reduceri clientului: ocazionale, la cerere, sistematic etc.;
- schimbarea aspectului şi conținutului aplicației (sistem de meniuri, Site Web, videoformate, rapoarte) ori de câte ori este necesar;
- se va ține cont tot timpul de drepturile consumatorului;

- crearea unor segmente de clienți după diferite criterii;
- se va ține cont de cerințele nivelurilor de conducere;
- se vor evidenția beneficiile clientului.

Dezvoltarea şi exploatarea aplicațiilor cu baze de date, în contextul economic actual, trebuie să țină cont şi de următoarele *aspecte metodologice*:

- valoarea ciclului de viață, care este valoarea netă prezentă a profitului care va fi realizată cu un număr mediu de clienți, pe parcursul unui număr de ani dat;
- succesul strategiei de dezvoltare ține cont de atributele clientului – profitabilitate, loialitate etc.;
- comunicarea cu clientul are rolul de a-i da acestuia satisfacție şi se poate face prin diferite metode - mesaje Email, telefon, puncte acumulate, gratuități, reduceri etc.;
- construirea profilului de client prin gruparea clienților cu atribute similare – demografice, zone geografice, stil de viață, profesii etc.

Metodologiile de realizare a sistemelor de baze de date - SBD care integrează noi tehnologii informatice trebuie să țină cont de aspectele prezentate, care rezultă din noul context economic actual.