Proiectarea Sistemelor Software Complexe

Curs 6 –Servere de Aplicații

Serverele de aplicații sunt servere bazate pe componente care se găsesc de obicei în mijlocul unei arhitecturi formată din N niveluri (N-tier architecture) și oferă servicii precum comunicare, securitate, tranzacții și persistență.

În Fig. 6.1 este prezentată arhitectura unei aplicații Web dezvoltată pe N niveluri. Acest tip de arhitectură este unul destul de frecvent întâlnit la aplicațiile Web.

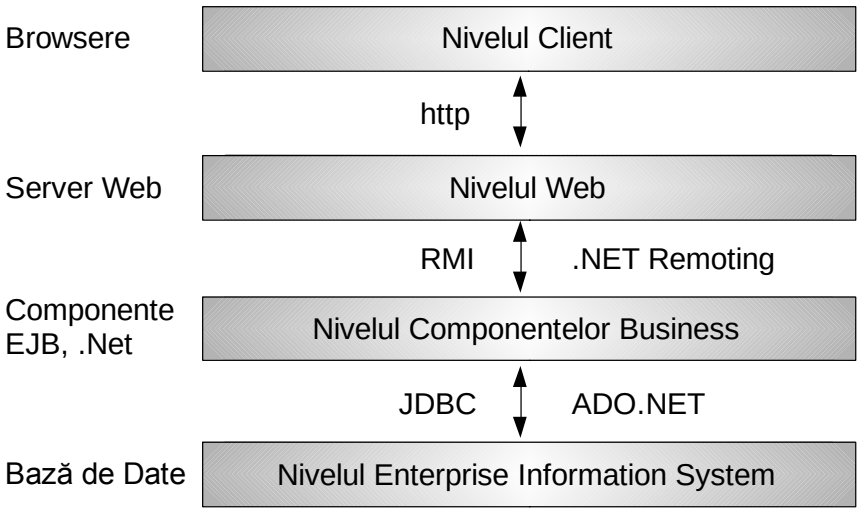

Fig. 6.1. Aplicație Web pe N niveluri.

În continuare sunt explicate cele 4 niveluri care apar în Fig. 6.1:

- Nivelul Client – într-o aplicație Web nivelul client este reprezentat de un browser Internet care trimite cereri HTTP către serverul Web și descarcă pagini HTML.
- Nivelul Web – acest nivel rulează pe un server Web, scopul lui fiind acela de a aștepta și a răspunde la cereri venite din partea clienților. Atunci când apare o cerere serverul Web invocă anumite componente găzduite de server pentru a răspunde la cerere; aceste componente diferă în funcție de tipul de server, de exemplu: pentru un server Web Java se poate vorbi de servlets și Java Server Pages (JSP), iar pentru un server .NET componentele sunt denumite Active Server Pages (ASP). Identificarea componentei care trebuie invocată se face pe baza informației conținută în cerere. Componenta identificată va procesa parametrii cererii și îi va folosi pentru a apela nivelul business logic pentru a obține informațiile necesare pentru a rezolva cererea. În final componenta va formata rezultatele în format HTML pentru a trimite răspunsul la client prin intermediul serverului Web.
- Nivelul Componentelor Business – acest nivel conține partea de business logic a aplicației. Componentele business sunt reprezentate fie de componente Enterprise JavaBeans (EJB) pentru J2EE, componente .NET sau obiecte CORBA. Componentele business primesc cereri de la nivelul

Web, rezolvă cererile interogând una sau mai multe baze de date și trimit răspunsul către nivelul web. Aceste componente rulează într-un container care oferă o serie de servicii. Tipul de servicii oferite de container componentelor pe care le găzduiește depinde de tipul acestuia (EJB, .NET, CORBA), dar include: suport pentru tranzacții, managementul ciclului de viață al componentelor, managementul stării, securitate, suport pentru multithreading și resource pooling. Componentele specifică prin fișiere de configurare comportamentul pe care îl doresc, iar containerul trebuie să garanteze îndeplinirea acestor cerințe. Acest mod de a specifica comportamentul unei componente prin fișiere de configurare este foarte avantajos întrucât programatorul nu trebuie să adauge cod care să trateze aspecte care sunt specifice mediului în care componenta va fi rulată.

- Nivelul Enterprise Information System – este reprezentat de una sau mai multe baze de date sau aplicații back-end pe care nivelul business trebuie să le interogheze pentru a rezolva cererile primite de la clienți.

Nucleul unui server de aplicații este reprezentat de containerul care găzduiește componentele business. În continuare va fi analizat ca și studiu de caz modelul EJB suportat de J2EE.

6.1 Enterprise JavaBeans

Enterprise JavaBeans reprezintă un model de programare standard pentru dezvoltarea de aplicații server utilizând limbajul de programare Java. Un server de aplicații compatibil J2EE trebuie să conțină un container EJB care să gestioneze execuția componentelor aplicației. Fig. 6.2 redă relațiile dintre serverul de aplicații container și serviciile oferite. Când un client EJB invocă o componentă de pe server, containerul alocă automat un fir de execuție și invocă o instanță a componentei EJB. Containerul gestionează toate resursele folosite de componentă și mediază interacțiunile dintre componentă și mediul exterior.

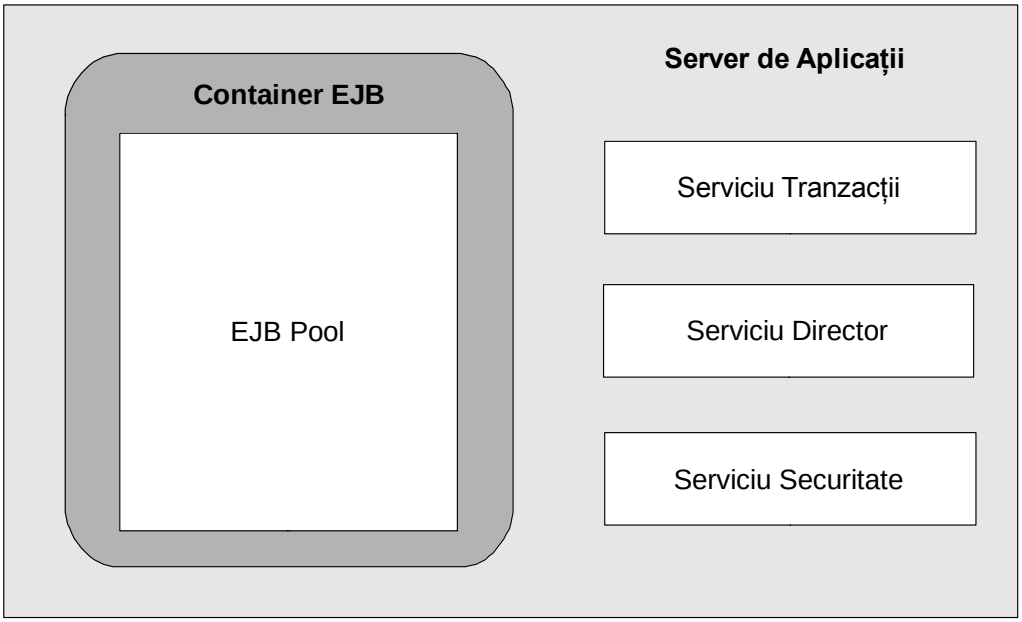

Fig. 6.2. Serverul de aplicații J2EE, containerul EJB și serviciile asociate.

6.2 Modelul unei Componente EJB

Modelul unei componente EJB definește arhitectura unei componente EJB. Specifică interfața componentei precum și modul în care ea interacționează cu containerul și cu alte componente. Specificațiile pentru EJB versiunea 1.1 definesc două tipuri principale de componente EJB: *session beans* și *entity beans*. Componentele de tipul *session beans* pot fi folosite pentru a executa partea de business logic a unei aplicații și de a oferi servicii care pot fi apelate de clienți. Într-o arhitectură de tipul model-view-controller un *session bean* corespunde nivelului controller (încapsulează partea de business logic a unei aplicații pe trei niveluri). Un *entity bean* pe de altă parte este folosit pentru a reprezenta o structură de date. Membrii unui entity bean corespund câmpurilor stocate într-o baza de date. Un *entity bean* este folosit (accesat) de către un *session bean* pentru a implementa un anumit serviciu.

De asemenea există două tipuri de *session beans* și anume: *session bean fără stare* și *session bean cu stare*. În Fig. 6.3 este ilustrată diferența dintre cele două tipuri de *session beans*. Un bean fără stare se caracterizează prin faptul că nu stochează nici o informație despre un client între două apeluri succesive. Un client obține o referință către o astfel de componentă după care poate să facă mai multe apeluri către diferite operații ale componentei. Totuși, de la un apel la altul se poate ca, containerul EJB să delege o altă componentă pentru a deservi apelul. Alocarea componentelor beans fără stare se realizează în funcție de necesități, de aceea un client nu poate fi sigur cu ce componentă comunică. Acest lucru face inutilă stocarea unei stări.

Un session bean cu stare se caracterizează prin faptul că poate stoca informații despre comunicarea cu un clientul. Odată ce un client a obținut o referință către o referință toate apelurile realizate de client vor fi direcționate către aceeași componentă. Containerul va crea câte o componentă bean pentru fiecare client. Clientul poate stoca orice fel de informație în starea componentei pe care o va putea regăsi la următorul apel.

Ciclul de viață al unei componente cu stare este gestionat de containerul EJB. Containerul va salva starea componentei pe disc dacă acesta nu a fost folosită pentru o anumită perioadă de timp și o va reîncărca automat atunci când apare un nou apel din partea clientului. De asemenea un container mai poate fi configurat să șteargă o componentă cu stare dacă acesta nu a fost folosită pentru o anumită perioadă de timp.

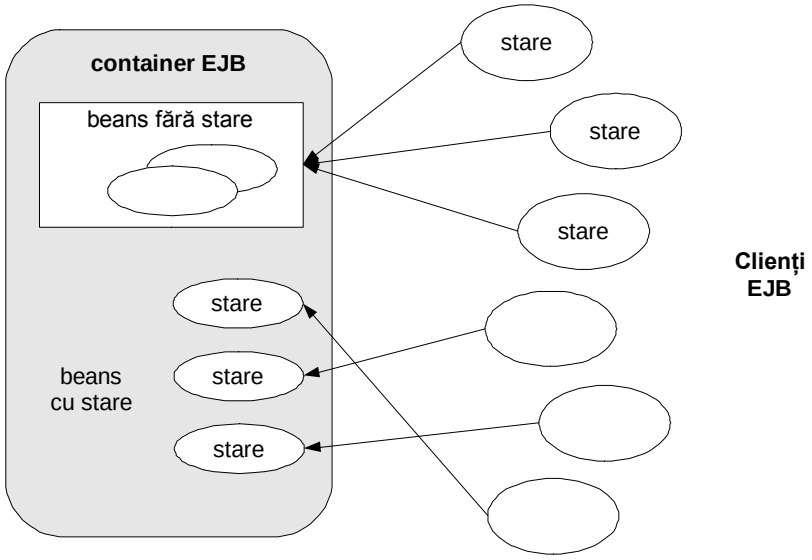

Fig. 6.3. Componente *session beans* cu stare și fără stare.

Componentele entity bean sunt și ele de două tipuri: *Container Managed Persistence* (CMP) entity bean și *Bean Managed Persistence* (BMP) entity bean. În acest context prin persistență se înțelege modul în care datele (de cele mai multe ori reprezintă un rând într-o bază de date relațională) asociate cu un entity bean sunt citite respectiv scrise.

În cazul componentelor CMP responsabilitatea de a încărca datele în bean, respective de a scrie modificările la momentele potrivite (sfârșitul unei tranzacții) înapoi în baza de date revin containerului. Componentele CMP se bazează pe servicii oferite de container și nu necesită cod suplimentar întrucât containerul se ocupă de tot. Componentele CMP sunt ușor de implementat și de întreținut pentru baze de date relaționate.

În cazul componentelor de tipul BMP, sarcina de a accesa și salva data reprezentată de componentă revine chiar codului componentei. Acest lucru se poate realiza utilizând apeluri JDBC sau apeluri către o baza de date dedicată sau apeluri API. Componentele BMP oferă programatorilor flexibilitatea de a implementa metode de persistență care sunt prea complicat de generat de către container sau care folosesc baze de date care nu sunt suportate de container (ex.: o bază de date dezvoltată peste FTP). Deși componentele BMP necesită un efort de programare mai mare ele se pot dovedi utile mai ales atunci când este nevoie de performanțe ridicate.

6.3 Programarea EJB

O componentă EJB depinde în totalitate de containerul EJB pentru tot ce face. Dacă o componentă EJB trebuie să folosească o conexiune JDBC sau o altă componentă EJB, ea va apela la serviciile puse la dispoziție de containerul EJB pentru a face acest lucru.

Pentru a crea o componentă EJB trebuie implementate două interfețe: una definește partea de business a componentei, iar cealaltă reprezintă metodele care gestionează ciclul de viață al componentei. Cele două interfețe sunt interfețele *remote* și *home*.

Interfața *home* conține metodele care definesc ciclul de viața al unei componente EJB. Acest metode oferă clienților servicii precum crearea, ștergerea și găsirea de instanțe de componente. Pe de altă parte interfața remote conține metode care reprezintă partea de business a componentei. Aceste metode sunt specifice sistemului software. Pentru a putea invoca metode din interfața *remote* un client trebuie mai întâi să obțină o referință la o componentă EJB utilizând metode din interfața *home*.

Un client EJB poate fi o aplicație Java de sine stătătoare, un servlet, un applet sau chiar o altă componentă EJB. Toți clienți vor folosi interfața home a componentei pentru a obține referințe la componentă. Referința obținută în acest mod va fi asociată cu o instanță a unei clase ce implementează interfața remote a componentei. Astfel, clientul va interacționa cu componenta EJB prin intermediul metodelor din interfața remote.

6.4 Deployment Descriptors

Unul din principalele avantaje ale componentelor EJB este reprezentat de modul în care o astfel de componentă separă responsabilitățile, și anume, ea separă partea de business logic de partea de infrastructură. Astfel o componentă EJB conține doar cod specific părții de business a unui sistem software. Rezolvarea problemelor specifice platformei și infrastructurii (ex.: tranzacții, ciclul de viață al componentei sau securitatea) este în totalitate sarcina containerului EJB. Această separare a responsabilităților duce la crearea unor componente relative simple fără a mai fi nevoie să se gestioneze complexitatea suplimentară introdusă de gestionarea infrastructurii.

O componentă EJB informează containerul ce servicii are nevoie prin intermediul așa numitului *deployment descriptor*. Un deployment descriptor este un document XML asociat cu o componentă EJB. Atunci când o componentă EJB este instalată într-un container, acesta din urmă va citi documentul XML asociat pentru a determina modul în care vor fi gestionate operații precum tranzacții, persistență etc. Așadar un descriptor reprezintă un mecanism declarativ prin care se descrie modul în care vor fi gestionate problemele legate de infrastructură.

Avantajul acestui mecanism constă în faptul că aceeași componentă EJB poate fi instalată utilizând diferiți descriptori fiecare fiind specific unui anumit tip de aplicație. De exemplu, dacă este nevoie de securitate se poate specifica modul în care componenta poate fi accesată, dacă nu este nevoie de securitate atunci se specifică acest lucru în descriptor. Avantajul este acela că în ambele situații din punctul de vedere al ingineriei software componenta va fi aceeași.

6.5 Responsabilitățile Containerului EJB

Containerul EJB este o unitate software foarte complexă. Serviciile pe care el trebuie să le ofere sunt:

- Gestionarea ciclului de viață și a instanțelor componentelor EJB, și anume: crearea, activarea, dezactivarea și ștergerea.
- Interceptează apelurile clienților către metode ale componentelor EJB pentru a garanta tranzacțiile și securitatea. De asemenea asigură apeluri înapoi la începutul și sfârșitul unei tranzacții.
- Gestionează persistența câmpurilor unei componente CMP.

Un container EJB oferă și o serie de alte facilități vitale:

- **Threading** – o componentă EJB nu trebuie să creeze și să manipuleze explicit un fir de execuție Java, ci trebuie să se bazeze pe container pentru a aloca un fir de execuție unei componente bean active pentru a mări gradul de concurență și performanțele. Acest lucru simplifică sarcina programatorului care nu mai trebuie să se preocupe de tratarea cererilor concurente.
- **Caching** – containerul EJB poate să gestioneze un cache de instanțe entity bean; dimensiunea cache-ului se poate specifica în descriptor.
- **Connection Pooling** – containerul poate să gestioneze un set de conexiuni la baza de date ceea ce duce la o creștere a eficienței accesului la resurse externe.

Bibliografie

[1] Ian Gorton. Essential Software Architecture. Editura Springer. 2006.
[2] http://java.sun.com/javaee/5/docs/tutorial/doc/