# The Reuse of the Components on the Java 2 Enterprise Edition Platform

Lect.dr. Marian CRISTESCU
Catedra de Informatică Economică, Universitatea "Lucian Blaga" Sibiu

*Reusable components are simply pre-built pieces of programming code designed to perform a specific function. When designing an application in a visual environment, controls can be quickly dropped into the design, and modified to fit the task at hand. Most of the controls you'll find are designed to handle such tasks as pushbuttons, menus, text labels, and so forth. As a developer, you only need to write code to "glue" them into your application, and develop the interactions between controls. Beans are "capsules" of code, each designed for a specific purpose. The advantage of Java Beans over standard programming controls is that Beans are independent. They are not specific to operating systems or development environments. A Bean created in one development environment can be easily copied and modified by another. This allows Java Beans greater flexibility in enterprise computing, as components are easily shared between developers.*
***Keywords:*** *Enterprise JavaBeans, EJB Object, Remote Interface, Home Interface, Home Object, Local Interface, Deployment Descriptor.*

## 1 Introducere

Platforma Java2 Enterprise Edition (J2EE), foloseşte ca model pentru aplicaţiile distribuite modelul ”multi-tier. Logica aplicaţiei este divizată în componente, iar diferitele componente ale aplicaţiei, care formează împreună o aplicaţie J2EE, sunt instalate pe maşini de calcul diferite în funcţie de nivelul de care aparţine respectiva componentă a aplicaţiei.

Astfel vom vorbi despre:

- *Client-tier* - componente care rulează pe maşina client (aplicaţii client şi apleturi);
- *Web-tier* - componente care rulează pe server-ul J2EE (Java Servlet şi JavaServer Pages);
- *Business-tier* - componente care rulează pe server-ul J2EE (Enterprise JavaBeans);
- *Enterprise information system (EIS)-tier* - software care rulează pe un server EIS, [DALE01]

În lucrarea de faţă sunt tratate componentele nivelului business-tier.

Obiectivele urmărite sunt:

- prezentarea părţilor componente ale unui Enterprise JavaBean şi interacţiunile dintre acestea;
- evidenţierea caracteristici de reutilizarea a componentelor pe partea de server.

## 2. EJB

Tehnologia EJB permite o abordare simplificată a dezvoltării aplicaţiilor multi-nivel, mascând complexitatea aplicaţiei şi oferă posibilitatea «component developerului» de a se concentra asupra business logicului aplicaţiei.

Tehnologia EJB oferă o serie de tipuri de componente EJB: session beans, entity beans, message-driven beans.

Componentele session beans reprezintă comportamentul asociat cu sesiunile client – de exemplu, un user care derulează o tranzacţie - *Entity beans* - reprezintă colecţii de date – ca de exemplu liniile într-o bază de date relaţională – şi încapsulează operaţii asupra datelor pe care le reprezintă. *Message-driven beans* permit aplicaţiilor J2EE să proceseze mesaje în mod asincron.

## 3. Enterprise bean class

Reprezintă implementarea propriu-zisă a bean-ului, şi este o clasă java care se conformează unei interfeţe bine definită şi respectă o serie de reguli precise necesare pentru ca bean-ul să poată rula în orice container EJB.

Implementările celor trei tipuri de bean-uri sunt diferite astfel:

▪ *pentru session beans* EB Class conține, de obicei, implementarea business logicului, de exemplu: calcularea prețurilor, transferul de fonduri între conturi bancare etc.;
▪ *pentru entity beans* EB Class implementează operațiunile pe/cu date, spre exemplu: schimbarea numelui unui client, modificarea unei comenzi etc.;
▪ *pentru message-driven beans* EB Class implementează operațiile legate de mesaje, cum ar fi de exemplu: recepționarea unui mesaj de transfer de fonduri între două conturi şi apelarea session bean-ului care ştie să execute operația de transfer de fonduri între două conturi.

Specificația EJB definește o serie de interfețe standard pe care clasa bean le poate implementa. Acestea obligă clasa bean-ului să expună anumite metode, pe care toate bean-urile trebuie să le ofere, aşa cum este definit de modelul componentelor EJB. Interfața de bază pe care toate tipurile de beans-uri trebuie să o implementeze este *javax.ejb.EnterpriseBean* :

```
public class FirstBean implements
javax.ejb.SessionBean{
        private SessionContext ctx ;
// o serie de metode care sunt cerute de interfața
        public void ejbCreate(){}
        public void ejbRemove(){}
        public void ejbActivate(){}
        public void ejbPassivate(){}
        public void
setSessionContext(javax.ejb.SessionContex
ctx){
                this.ctx=ctx ;
}
// metodele care definesc business logic-ul
bean-ului
        public String
FirstBeanBussniesMethod{
        //codul metodei
}
}
```

Această interfață este ca o interfață «marker»; implemetarea acestei interfețe indică faptul că orice clasă care o implementează este o clasă enterprise bean. Ficare tip de EJB în parte are câte o interfață specifică care extinde interfața *javax.ejb.EnterpriseBean*:
▪ *javax.ejb.SessionBean;*
▪ *javax.ejb.EntityBean;*
▪ *javax.ejb.MessageDrivenBean.*

## 4. EJB object

Enterpsise beans nu sunt obiecte remote pe deplin. Un client care vrea sa folosească o instanță a unei clase ejb, nu invocă metoda direct, ci invocarea este interceptată de către containerul EJB care ulterior o trimite instanței ejb respective, prin urmare containerul acționează ca un strat intermediar între codul client si bean. Acest strat intermediar se comportă ca un singur obiect numit EJB object, şi care poate fi considerat ca parte fizică a containerului. Toate obiectele EJB au cod specific containerului implementat, de aceea containerul generează fişierul *class* pentru obiectul EJB automat.

Fiecare container EJB are o serie de unelte numite «glue-code». Aceste unelte au menirea să integreze beans-urile în mediul containerului EJB, ele generează clasele *stub*, *skeleton*, clase de acces la date, şi alte clase de care fiecare container în parte are nevoie.

Uneltele containerului transformă un enterprise bean într-o componentă distribuită pe partea de server; acest fapt implică algoritmi care să suporte managementul resurselor, ciclu de viață, managementul stărilor, tranzacții, securitate, şi multe alte servicii. Codul generat tratează aceste servicii într-un mod particular fiecărui container în parte.

## 5. Remote interface

Interfața remote este o interfață specială, care reproduce toate metodele pe care clasa bean corespunzătoare le expune, astfel uneltele care autogenerează obiectele EJB ştiu ce metode să ”cloneze”. Interfețele Remote trebuie să se supună unor reguli speciale pe care le definește specificația EJB.

De exemplu, toate interfețele remote trebuie să derive dintr-o interfață comună oferită de Sun Microsystems. Această interfață este

numită j*avax.ejb.EJBObject*.

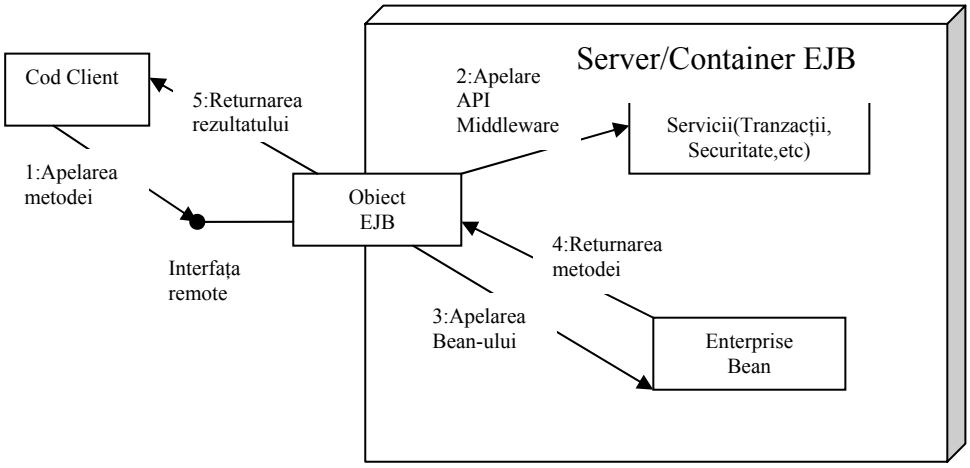


**Fig.1.** Apelarea unei metode a bean-ului

*public interface javax.ejb.EJBObject extends java.rmi.Remote*
*{*
*//o serie de metode care nu trebuie implementate*
*//ele sunt implementate de container*
*}*

Pe lângă metodele listate în cod sursă, interfaţa remote dubleză metodele bean-ului. Atunci când clientul bean-ului respectiv invocă oricare din metodele expuse (de către bean), obiectul EJB deleagă metoda implementării corespunzătoare care există în bean.

**6. Home Object**

Clientul nu poate instanţia un obiect EJB direct deoarece obiectele pot exista pe o staţie diferită decât cea pe care se află clientul. Pentru a obţine o referinţă la un obiect EJB, codul client cere un obiect EJB dintr-o "fabrică" de obiecte EJB, care este responsabilă cu instanţierea şi distrugerea obiectelor EJB. Specificaţia EJB numeşte această "fabrică" *home object*.

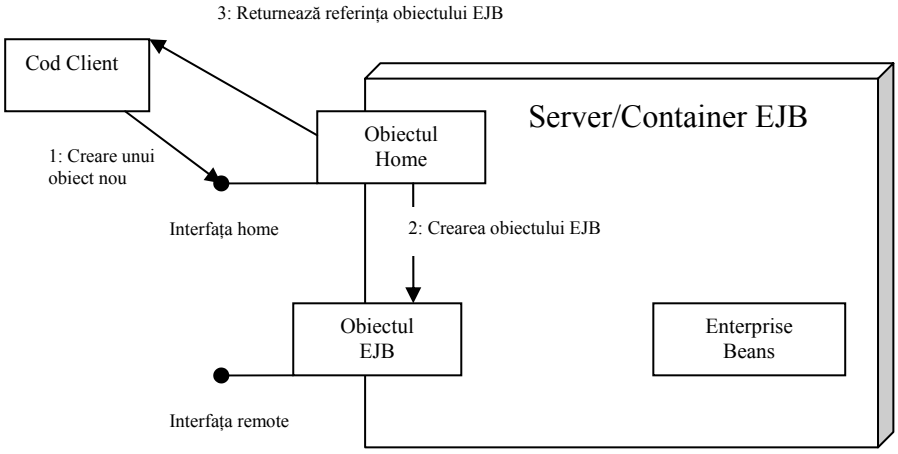


**Fig.2.** Crearea unui obiect EJB

Principalele responsabilităţi ale obiectelor home sunt:

- crearea obiectelor EJB;
- găsirea obiectelor EJB existente;
- distrugerea obiectelor EJB.

La fel ca şi obiectele EJB, obiectele home sunt particulare şi specifice fiecărui container EJB şi sunt autogenerate de către uneltele containerului.

**7. Home interface**

Interfețele home definesc metodele pentru crearea, distrugerea şi găsirea obiectelor EJB. Obiectul home al containerului implementează interfața home definită.
La fel ca la celelate interfețe, EJB definește o serie de metode pe care toate interfețele home trebuie să le suporte. Aceste metode sunt definite în interfața *javax.ejb.EJBHome*, interfață pe care trebuie să o extindă interfața home propriu-zisă.

```
public interface javax.ejb.EJBHome extends
java.rmi.Remote
{
//o serie de metode care nu trebuie imple-
mentate
//ele sunt implementate de container
}
```

**8. Local interface**

Crearea beans-urilor prin interfața home se face destul de încet. De asemenea apelarea unui bean prin interfața remote este un proces care consumă destul de mult timp. Pentru realizarea acestei operații se parcurg următorii paşi:
1.Clientul apelează stub-ul local;
2.Stub-ul organizează parametrii într-o formă specifică nivelului rețea;
3.Stubul trimite parametrii prin rețea la skeleton;
4.Skeletonul preia parametri în forma specifică rețelei şi îi transformă într-o formă specifică pentru Java;
5.Skeletonul apelează obiectul EJB;
6.Obiectul EJB execută operațiile specifice nivelului middleware, cum ar fi servicii pentru tranzacții, securitate si ciclul de viață al beans-urilor;
7.După ce obiectul EJB apelează instanța bean-ului, şi bean-ul execută operațiile specifice, iar paşii prezentați anterior trebuie parcurşi pentru returnarea rezultatului.
Începând cu versiunea 2.0 beans-urile enterprise pot fi apelate într-un mod mai eficient şi mai rapid prin intermediul unor obiecte locale astfel:
1.Clientul apelează obiectul local;
2.Obiectul local execută operațiile specifice nivelului de mijloc;
3.După ce instanța enterprise bean-ului execută operațiile specifice, returnează controlul obiectului local, care apoi predă controlul clientului.
Prin aceste obiecte locale se evită paşii 2-5 (în ambele sensuri). De asemenea pentru crearea, într-un mod mai rapid, se poate apela o interfață specială - local home interface - care este implementată de către container ca şi obiectul local home aferent.
Aceste interfețe locale sunt opționale, ele pot fi complementare sau înlocuitoare ale interfețelor remote. Scrierea interfețelor locale implică extinderea interfeței javax.ejb.EJBLocalObject pentru interfața locală şi extinderea interfeței javax.ejb.EJBLocalHome.
Interfețele locale au şi două dezavantaje:

- ✓ pot fi folosite numai la apelurile de bean-uri care se află în acelaşi proces (de exemplu: dacă avem un session bean – CeckAccount - care apelează un entity bean – BankAccount - ambele bean-uri aflându-se pe acelaşi server de aplicație). Nu se poate apela un bean remote dacă codul se bazează pe interfețe locale;
- ✓ interfețele locale transmit parametri prin referință şi nu prin valoare.

**9. Deployment Descriptors**

Descriptorul de deployment este un fişier XML care descrie serviciile middleware pe care bean-ul le necesită. Containerul examinează acest fisier XML şi îndeplineşte cerințele enunțate în fişier.
De exemplu, se foloseşte un deployment descriptor pentru a specifica următoarele cerințe ale unui bean:

- *cerințe legate de managementul şi ciclul de viață al unui bean* - se specifică numele clasei bean-ului, dacă este session, entity sau message-driven bean, şi interfața home care generează bean-ul;
- *cerințe legate de tranzacții* - aceste setări specifică necesitățile bean-ului care rulează într-o tranzacție, cum ar fi: tranzacția trebuie să înceapă de fiecare dată când cineva apelează acest bean şi tranzacția trebuie să se termine după ce bean-ul termină apelarea metodei;
- *cerințe legate de securitate* – aceşti de-

scriptori conţin «*access control entries*» pe care bean-urile şi containerul le folosesc pentru a accesa anumite operaţii; se poate specifica cui îi este permis să folosească anumite bean-uri sau chiar anumite metode ale unui bean anume; de asemenea, poate fi specificată securitatea mediului în care rulează bean-ul, lucru folositor dacă bean-ul trebuie să execute operaţii securizate (de exemplu: doar directorul executiv al unei bănci poate apela metoda care creează un nou cont bancar).

**10. Reutilizarea componentelor**

Generând clasele bean, interfeţele remote, home şi local dar şi deployment descriptorul, respectiv «împachetându-le» într-un fişier EJB-jar, se obţine enterprise bean(-ul/-urile) în forma finală. Acest fişier este apoi desfăşurat în serverul de aplicaţii. Serverul de aplicaţii despacheteză fişierul EJB-jar şi încarcă bean(-ul/-urile).

Caracteristicile bean-ului care determină reutilizarea componentelor EJB sunt:

- separarea business logicului bean-ului de restul codului;
- folosirea deployment descriptorului pentru particularizarea bean-ului fără «atingerea» codului;
- EJB oferă o interfaţă pentru reutilizare (interfaţa remote/local).

Accesarea metodelor beans-urilor de către alte componente prin intermediul acestor interfeţe permit reutilizarea unei bucăţi din business logic în detrimentul extinderii şi particularizării acesteia.

Reutilizarea automată la nivel funcţional a sistemelor, componentelor şi aplicaţiilor eterogene este o problemă complexă din cel puţin două puncte de vedere:

- *tehnologic* - "ascunderea" detaliilor de implementare a unor componente scrise în limbaje de programare diferite, rulând poate sub sisteme de operare diferite, eventual chiar distribuit, pe sisteme de calcul diferite;
- *conceptual* - pentru reutilizarea unor componente având funcţionalităţi complexe şi modele de date diferite (din punct de vedere sintactic dar şi semantic) nu este suficientă încapsularea acestora utilizând facilităţi de tip CORBA, fiind necesară construcţia unui *model conceptual comun* al componentelor şi utilizarea explicită a acestuia în procesul de raspuns la interogări şi în general la exploatarea sistemului integrat.

**Concluzii**

Programarea orientată pe componente este un mijloc eficient pentru atingerea unui deziderat universal al programatorilor - reutilizarea funcţionalităţii existente.

Redusă la esenţial, această tehnică înseamnă interconectarea unor componente de bază pentru realizarea unor componente mai complexe; un exemplu concludent pentru această abordare îl reprezintă combinarea de componente EJB pentru realizarea business logicului aplicaţiilor distribuite pe platforma J2EE.

Reutilizarea de componente, permite industrializarea şi eficientizarea produselor software, reducând costurile de proiectare şi întreţinere ale acestora .

**Bibliografie**

[DALE01] - Dale G., Bodoff S., Jendrock E., Pawlan M., Stearns B., „The J2EE Tutorial”, Sun Microsystems 2001;

[MARI02] - Marinescu F., „EJBDesignPatterns", Wiley Computer Publishing, 2002;

[ROMA01] - Roman E., Ambler S., Jewell T., “Mastering Enterprise Java Beans, (de la www.TheServerSide.com), 2001;

[****a] - http://www.w3schools.com/sql/default.asp;

[****b] - http://java.sun.com/features/01/wora.html;

[****c] - http://www.fawcette.com/javapro/archives/jm0102/default.asp;

[****d] - http://localhost:8080/jdbc-simple;

[****e] - http://www.coreservlets.com;

[****f] - http://www.onjava.com/pub/a/onjava/2002/07/24/antauto.html.