

Budowa aplikacji w technologii Enterprise JavaBeans

Maciej Zakrzewicz

PLUG

Plan prezentacji

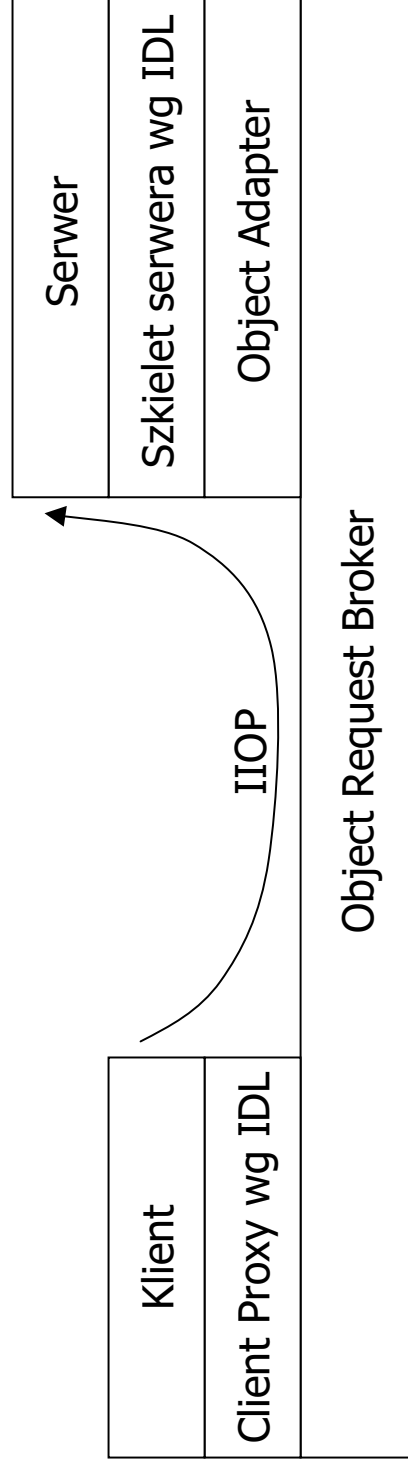
- Przegląd architektur aplikacji rozproszonych: CORBA, SOAP, EJB
- Wprowadzenie do Enterprise JavaBeans (EJB)
- Budowa komponentów sesyjnych EJB
- Budowa komponentów encyjnnych EJB
- Podsumowanie

Cele budowy aplikacji rozproszonych

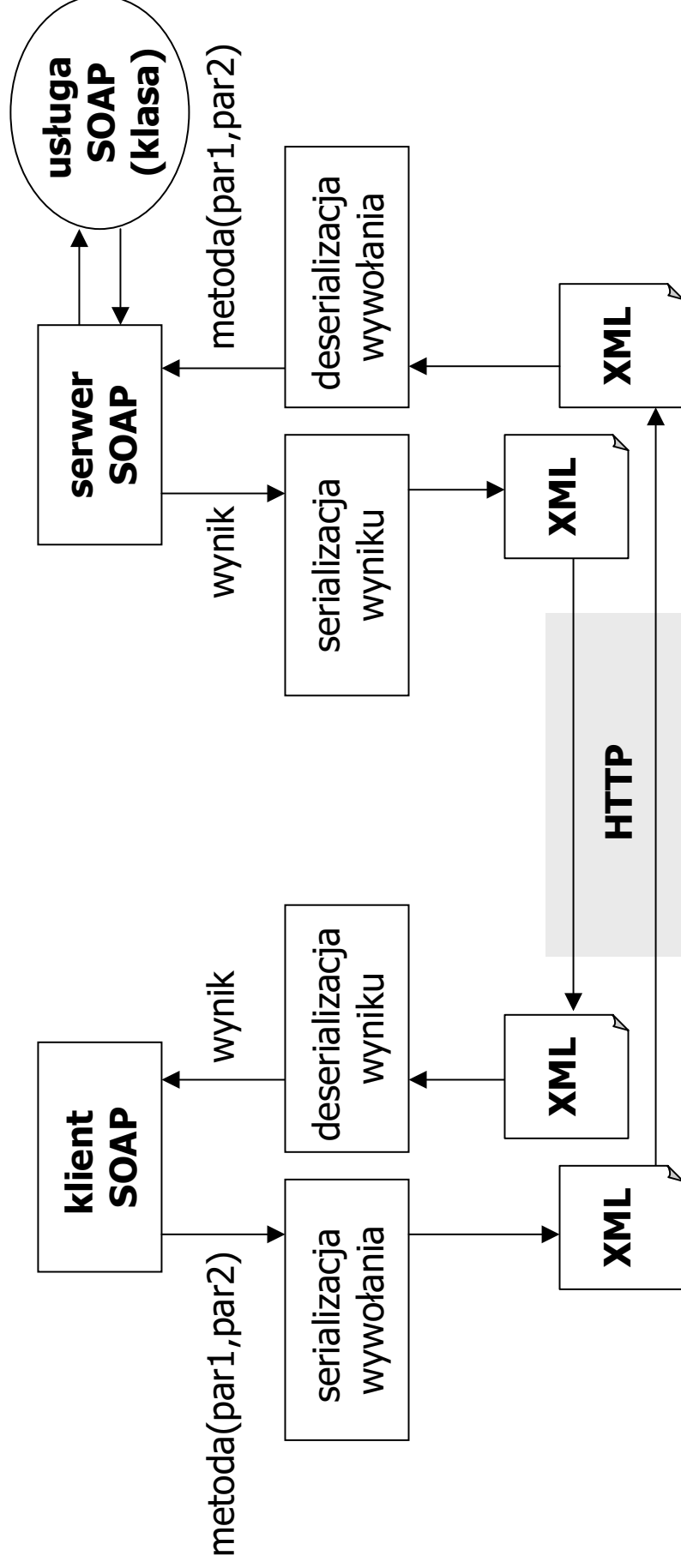
- Zwiększenie przepustowości systemu poprzez rozproszenie przetwarzania pomiędzy wiele komputerów
- Zwiększenie niezawodności systemu poprzez replikację komponentów w alternatywnych lokalizacjach
- Odciążenie komputera użytkownika końcowego dzięki przeniesieniu ciężkiego przetwarzania na stronę serwera aplikacji
- Współdzielenie komponentów przez różne aplikacje pracujące we wspólnych środowisku sieciowym

Przegląd architektury: CORBA

- Jedna z pierwszych architektur budowy heterogenicznych rozproszonych aplikacji komponentowych
- Specyfikacja obejmuje architekturę oraz język opisu interfejsów - IDL (Interface Description Language)
- Aplikacje CORBA składają się z programu klienta, współpracującego z programami serwerów poprzez warstwę komunikacyjną ORB (Object Request Broker)

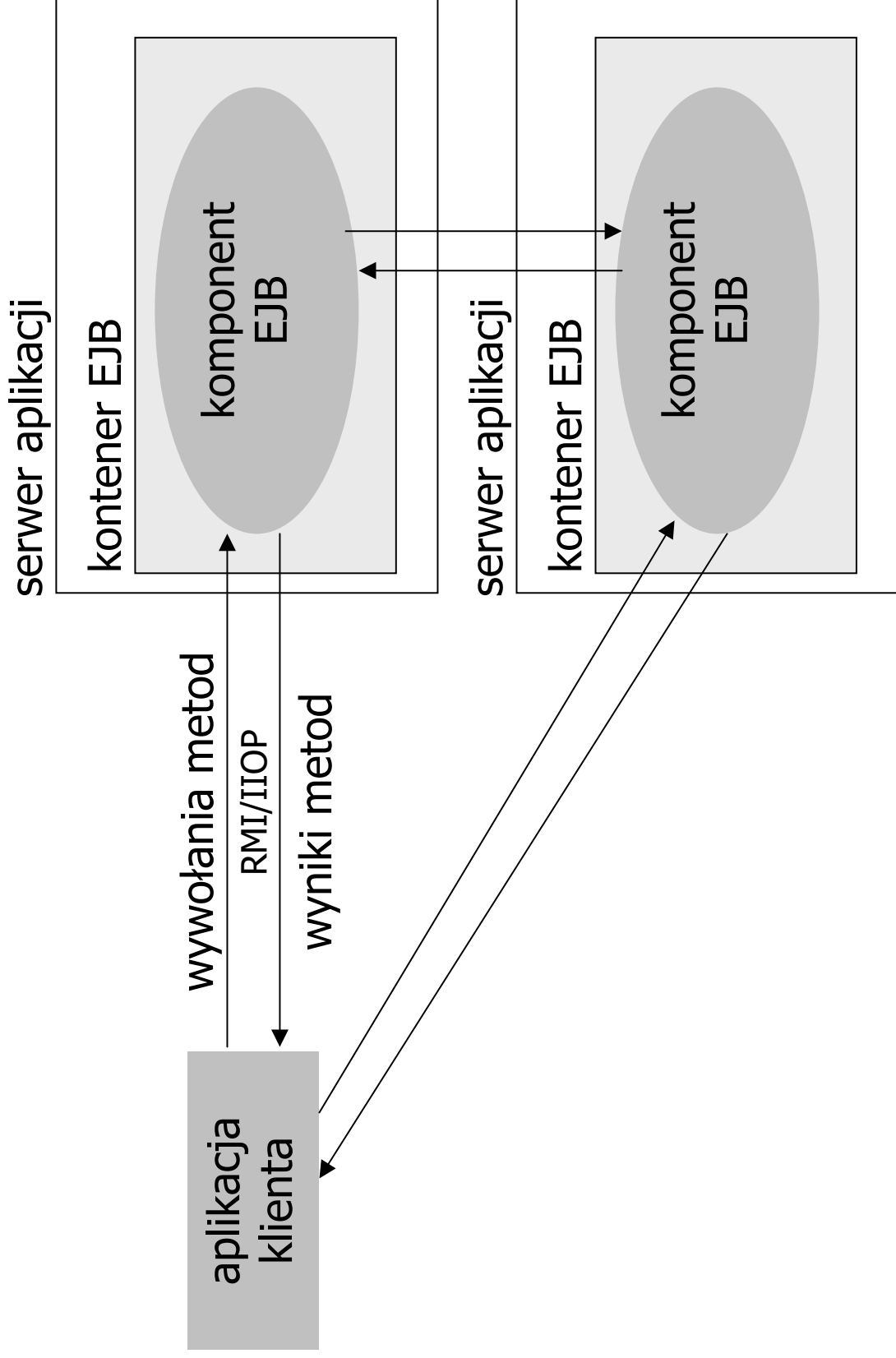


Przegląd architektury: SOAP



Wywołanie metody przez klienta jest konwertowane do dokumentu XML. Argumenty wywołania są serializowane (konwertowane do ciągu bajtów reprezentowanego heksadecymalnie). Dokument XML, nazywany komunikatem SOAP, jest przekazywany poprzez HTTP POST do usługi, która po dokonaniu deserializacji argumentów wywołania metody, uruchamia żadaną metodę i przekazuje zwrótnie jej rezultat, korzystając z analogicznego algorytmu.

Przegląd architektury: EJB

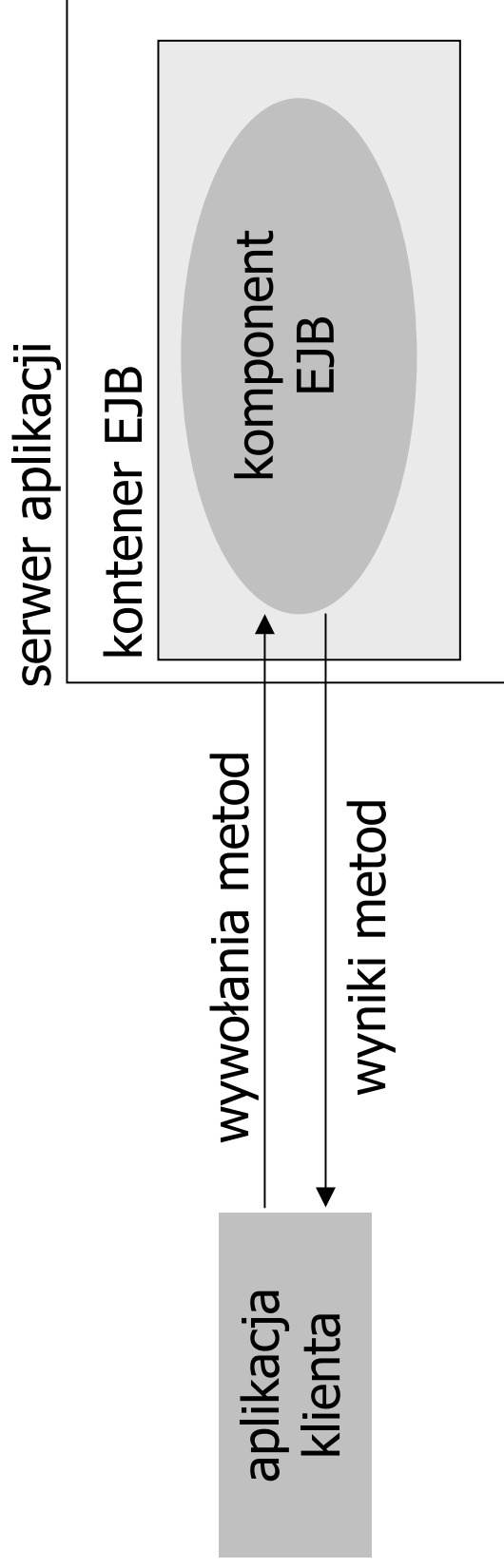


Wprowadzenie do Enterprise JavaBeans

Enterprise JavaBeans

- Specyfikacja Enterprise JavaBeans definiuje architekturę i metodę budowy rozproszonych komponentów obiektowych uruchamianych po stronie serwera aplikacji
- Komponenty EJB są wykorzystywane do budowy złożonych aplikacji rozproszonych na zasadzie „składania z klocków”

Ogólna architektura EJB



Program Java realizowany w technologii Enterprise JavaBeans składa się z dwóch rodzajów elementów składowych: lokalnej *aplikacji klienta* i zdalnych *komponentów przetwarzania danych*. Aplikacja klienta komunikuje się poprzez sieć komputerową z komponentami przetwarzania danych, przesyłając do nich żądania zdalnego wykonania metod. Wyniki działania tych metod są następnie tą samą drogą odsyłane do aplikacji klienta.

Kontener EJB

- Bezpośrednim środowiskiem uruchomieniowym dla każdego komponentu Enterprise JavaBeans jest kontener EJB
- Kontener EJB całkowicie pośredniczy w komunikacji pomiędzy komponentem EJB a światem zewnętrznym
- Kontener EJB oferuje komponentowi szereg usług o charakterze systemowym: ochrona dostępu, obsługa transakcji, itp.
- Komponent EJB może uzyskać dostęp do usług kontenera EJB korzystając z mechanizmu typu callback - w chwili powoływania do życia obiektu komponentu EJB, kontener EJB przekazuje komponentowi EJB pewien rodzaj uchwytu zwrotnego; za pomocą tego uchwytu obiekt komponentu EJB może wykonywać wywołania metod kontenera EJB

Typy komponentów EJB

- Specyfikacja Enterprise JavaBeans 2.0 definiuje trzy typy komponentów EJB:
 - **sesyjne** (Session Bean): sesyjny komponent EJB to krótkotrwały obiekt wykorzystywany przez pojedynczą aplikację klienta i nie współdzielony z innymi aplikacjami, stanowiący logiczne rozszerzenie kodu aplikacji klienta umieszczone po stronie serwera aplikacji
 - **encyjne** (Entity Bean): encyjny komponent EJB reprezentuje dane, które są trwale przechowywane w systemie bazy danych; cechuje się długim czasem życia, pozwala na atomowe, transakcyjne modyfikacje bazy danych, współdzielony przez wielu klientów, dostępny dla wielu sesji
 - **komunikatowe** (Message-Driven Bean): komunikatowy komponent EJB jest asynchronicznym konsumentem komunikatów JMS (Java Messaging Service) pochodzących ze środowisk kolejkowych; uruchamiany wtedy, kiedy nadchodzi komunikat od klienta, wykonywany asynchronicznie, może modyfikować zawartość bazy danych, bezstanowy

Ogólny proces tworzenia komponentu EJB

- Przygotowanie kodu źródłowego **klasy Java**, reprezentującej komponent EJB
- Utworzenie **dwóch interfejsów Java**, reprezentujących punkty kontaktu świata zewnętrznego z komponentem EJB:
 - interfejs *Home*, służący do zarządzania cyklem życia komponentu EJB
 - interfejs *Remote/Local*, służący do wywoływania metod logiki biznesowej komponentu EJB
- Przygotowanie XML-owego **pliku konfiguracyjnego** - *deskryptora instalacji* (Deployment Descriptor)
- Kompilacja całego przygotowanego kodu Java i utworzenie z niego pliku JAR/EAR o specjalnej wewnętrznej strukturze katalogów
- Umieszczenie przygotowanych plików w systemie plików serwera aplikacji; zarejestrowanie komponentu EJB

Sesyjne komponenty EJB

- Traktowane jako logiczna kontynuacja kodu aplikacji klienta, fizycznie zlokalizowana jednak po stronie serwera aplikacji
- Żyją jedynie przez czas trwania sesji aplikacji klienta
- Ich zastosowanie przypomina klasyczne rozwiązania mechanizmów zdalnego wołania procedury RPC
- Dwa rodzaje sesyjnych komponentów EJB:
 - *stanowe* (stateful) komponenty EJB są przez cały czas życia skojarzone z jedną i tą samą aplikacją klienta i pamiętają swój stan obiektu
 - *bezstanowe* (stateless) sesyjne komponenty EJB nie gwarantują pamiętania swojego stanu obiektu i mogą być przy każdym wywołaniu wykorzystywane przez inną aplikację klienta

Przykład

Tworzenie sesyjnego komponentu EJB (1/5)

Interfejs Remote

```
import javax.ejb.EJBObject;  
import java.rmi.RemoteException;  
  
public interface Multiplier1Remote extends EJBObject  
{  
    float multiply(float factor1, float factor2)  
        throws RemoteException;  
}
```

Interfejs Remote deklaruje metody komponentu EJB, które będą dostępne dla zdalnej aplikacji klienta. Metody te muszą zostać zaimplementowane przez programistę w klasie komponentu EJB.

Przykład

Tworzenie sesyjnego komponentu EJB (2/5)

Interfejs Home

```
import javax.ejb.EJBHome;
import java.rmi.RemoteException;
import javax.ejb.CreateException;

public interface Multiplier1Home extends EJBHome
{
    Multiplier1Remote create()
        throws RemoteException, CreateException;
}
```

Interfejs Home deklaruje metody zarządzania cyklem życia komponentu EJB, które będą dostępne dla zdalnej aplikacji klienta. Metody te nie będą implementowane przez programistę, lecz zostaną wygenerowane automatycznie.

Przykład

Tworzenie sesyjnego komponentu EJB (3/5)

Klasa komponentu EJB

```
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;

public class Multiplier1Bean implements SessionBean
{
    public void ejbCreate() {}
    public void ejbActivate() {}
    public void ejbPassivate() {}
    public void ejbRemove() {}
    public void setSessionContext(SessionContext ctx) {}
    public float multiply(float factor1, float factor2) {
        return factor1 * factor2;
    }
}
```

Przykład

Tworzenie sesyjnego komponentu EJB (4/5)

Plik deskryptora instalacji

```
<?xml version = '1.0' encoding = 'windows-1252'?>
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise
JavaBeans 1.1//EN" "http://java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd">
<ejb-jar>
  <enterprise-beans>
    <session>
      <description>Session Bean ( Stateless )</description>
      <display-name>Multiplier1</display-name>
      <ejb-name>Multiplier1</ejb-name>
      <home>Multiplier1Home</home>
      <remote>Multiplier1Remote</remote>
      <ejb-class>MultiplierBean</ejb-class>
      <session-type>Stateless</session-type>
      <transaction-type>Container</transaction-type>
    </session>
  </enterprise-beans>
</ejb-jar>
```

Przykład

Tworzenie sesyjnego komponentu EJB (5/5)

Fragment kodu aplikacji klienta EJB

```
Multiplier1Home multiplier1Home;
Multiplier1Remote multiplier1Remote;

Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
        "com.evermind.server.rmi.RMIInitialContextFactory");
env.put(Context.SECURITY_PRINCIPAL, "scott");
env.put(Context.SECURITY_CREDENTIALS, "tiger");
env.put(Context.PROVIDER_URL,
        "ormi://miner:1811/ejbapp1");
Context ctx = new InitialContext(env);

multiplier1Home = (Multiplier1Home)ctx.lookup("multiplier1");
multiplier1Remote = multiplier1Home.create();
System.out.println(multiplier1Remote.multiply(2,2));
```

Encyjne komponenty EJB

- Służą do uzyskania efektu trwałości obiektów programowych Java poprzez ich automatyczne fizyczne składowanie w bazie danych; żyją aż do momentu jawnego ich usunięcia
- Wykorzystywane do budowy warstwy komunikacji logiki biznesowej z bazą danych lub też do implementacji takiej logiki biznesowej, która intensywnie eksploatuje bazę danych
- Pojedynczy encyjny komponent EJB może być współdzielony przez wiele aplikacji klientów
- Dwa typy:
 - z *trwałością obsługiwaną przez komponent* (BMP - Bean-Managed Persistence): muszą same umieszczać swój stan w bazie danych, korzystając np. z biblioteki JDBC lub SQLJ
 - z *trwałością obsługiwaną przez kontener EJB* (CMP - Container-Managed Persistence): polegają na funkcjonalności kontenera EJB, którego zadaniem jest w tym przypadku automatyczne zapisywanie i odczytywanie stanu komponentu do/z bazy danych

Przykład

Tworzenie encyjnego komponentu EJB CMP (1/5)

Tabela Emp

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
...							

Tabela Emp będzie służyć do trwałego przechowywania obiektów EJB w bazie danych.
Każdy obiekt EJB będzie reprezentować jednego pracownika.

Przykład

Tworzenie encyjnego komponentu EJB CMP (2/5)

Interfejs Remote

```
import javax.ejb.EJBObject;
import java.rmi.RemoteException;

public interface Employee2Remote extends EJBObject
{
    long getEmpno() throws RemoteException;
    void setEmpno(long newEmpno) throws RemoteException;
    ...
    long getDeptno() throws RemoteException;
    void setDeptno(long newDeptno) throws RemoteException;
}
```

Interfejs Remote deklaruje metody komponentu EJB, które będą dostępne dla zdalnej aplikacji klienta. Metody te muszą zostać zaimplementowane przez programistę w klasie komponentu EJB. W powyższym przykładzie są to metody dostępu do obiektu pracownika.

Przykład

Tworzenie encyjnego komponentu EJB CMP (3/5)

Interfejs Home

```
import javax.ejb.EJBHome;  
import java.rmi.RemoteException;  
import javax.ejb.CreateException;  
import javax.ejb.FinderException;  
import java.util.Collection;
```

Interfejs Home deklaruje metody zarządzania cyklem życia i wyszukiwania komponentu EJB, które będą dostępne dla zdalnej aplikacji klienta. Metody te nie będą implementowane przez programistę, lecz zostaną wygenerowane automatycznie.

```
public interface Employee2Home extends EJBHome {  
    Employee2Remote create ()  
        throws RemoteException, CreateException;  
    Employee2Remote create (long empno)  
        throws RemoteException, CreateException;  
    Employee2Remote findByPrimaryKey (Employee2RemotePK primaryKey)  
        throws RemoteException, FinderException;  
    Collection findAll () throws RemoteException, FinderException;  
    Employee2Remote findByName (String val)  
        throws RemoteException, FinderException;}
```

Przykład

Tworzenie encyjnego komponentu EJB CMP (4/5)

Klasa komponentu EJB

```
import javax.ejb.*;

public class Employee2Bean implements EntityBean {
    public long empno; public String ename;
    public String job; public long mgr;
    public String hiredate; public long sal;
    public long comm; public long deptno;
    ...
    public long getEmpno() {return empno;}
    public void setEmpno(long newEmpno) {empno = newEmpno;}
    ...
    public long getDeptno() {return deptno;}
    public void setDeptno(long newDeptno) {deptno = newDeptno;}
}
```

Przykład

Tworzenie encyjnego komponentu EJB CMP (5/7)

```
<?xml version = '1.0' encoding = 'windows-1252'?>
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise
JavaBeans 1.1//EN" "http://java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd">
<ejb-jar><enterprise-beans> <entity>
<description>Encyjny komponent EJB - CMP</description>
<display-name>Employee</display-name>
<ejb-name>Employee</ejb-name>
<home>Employee2Home</home>
<remote>Employee2Remote</remote>
<ejb-class>Employee2Bean</ejb-class>
<persistence-type>Container</persistence-type>
<prim-key-class>Employee2RemotePK</prim-key-class>
<reentrant>False</reentrant>
<cmp-field><field-name>empno</field-name></cmp-field>
...
<cmp-field><field-name>deptno</field-name></cmp-field>
</entity></enterprise-beans></ejb-jar>
```

Plik deskryptora instalacji

Przykład

Tworzenie encyjnego komponentu EJB CMP (6/7)

```
...
<entity-deployment name="Employee" data-source="jdbc/Connection1DS" table="EMP">
  <primkey-mapping><cmp-field-mapping><fields>
    ...
    <cmp-field-mapping name="empno" persistence-name="EMPNO"
      persistence-type="NUMBER(4)"/>
    ...
    <cmp-field-mapping name="deptno" persistence-name="DEPTNO"
      persistence-type="NUMBER(2)"/>
  <finder-method partial="True" query="ename=$1"><method>
    <ejb-name>Employee</ejb-name>
    <method-name>findByName</method-name>
    <method-params>
      <method-param>java.lang.String</method-param>
    </method-params>
  </method></finder-method>
</entity-deployment></enterprise-beans></orion-ejb-jar>
```

Plik deskryptora instalacji

Przykład

Tworzenie encyjnego komponentu EJB CMP (7/7)

```
Employee2Home employee2Home;  
Employee2Remote e;  
  
Hashtable env = new Hashtable();  
env.put(Context.INITIAL_CONTEXT_FACTORY,  
        "com.evermind.server.rmi.RMIInitialContextFactory");  
env.put(Context.SECURITY_PRINCIPAL, "scott");  
env.put(Context.SECURITY_CREDENTIALS, "tiger");  
env.put(Context.PROVIDER_URL, "ormi://miner:1811/ejbapp2");  
Context ctx = new InitialContext(env);
```

Fragment kodu aplikacji klienta

```
employee2Home = (Employee2Home) ctx.lookup("Employee");  
e = (Employee2Remote) employee2Home.findByName("KING");  
  
System.out.println(e.getJob());  
System.out.println(e.getSal());  
e.setSal(550);
```

Podsumowanie

- Enterprise JavaBeans stanowią naturalny wybór architektury aplikacji rozproszonych dla programisty Java
- Ze względu na skomplikowaną procedurę przygotowywania wersji instalacyjnej komponentów EJB, zaleca się wykorzystywanie specjalizowanych narzędzi programistycznych (np. JDeveloper)
- W popularnych zastosowaniach, technologia EJB służy do realizacji wielowarstwowych aplikacji, w których interakcja z bazą danych obsługiwana jest przez komponenty encyjne, a przetwarzanie danych - przez komponenty sesyjne, będące klientami komponentów encyjnych

Podsumowanie - porównanie technologii

	CORBA	SOAP	EJB
język programowania	dowolny	dowolny	Java
protokół komunikacyjny	IIOP	HTTP	RMI
integracja z bazą danych	ręczna	ręczna	automatyczna
dostęp asynchroniczny (kolejkowy)	nie	tak	tak
obsługiwane przez J2EE	tak	tak	tak