

XI Konferencja PLOUG
Kościelisko
Październik 2005

Projektowanie aplikacji J2EE w architekturze Model-View-Controller

Maciej Zakrzewicz

PLOUG, Instytut Informatyki Politechniki Poznańskiej

mzakrz@cs.put.poznan.pl

Streszczenie

Projektowanie modułowych aplikacji J2EE składających się z serwletów Java, stron JavaServer Pages i komponentów Enterprise JavaBeans jest zadaniem trudnym ze względu na liczbę występujących składników oraz złożone interakcje zachodzące pomiędzy nimi. Jednym z rozwiązań ułatwiających koordynację prac projektowych oraz pielęgnację wytworzonego oprogramowania jest filozofia architektury Model-View-Controller, zakładającej specyficzną funkcjonalną specjalizację modułów. Referat przedstawia podstawowe założenia architektury Model-View-Controller, a na tym tle omawia narzędzia projektowania aplikacji J2EE wchodzące w skład pakietu Oracle JDeveloper 10g.

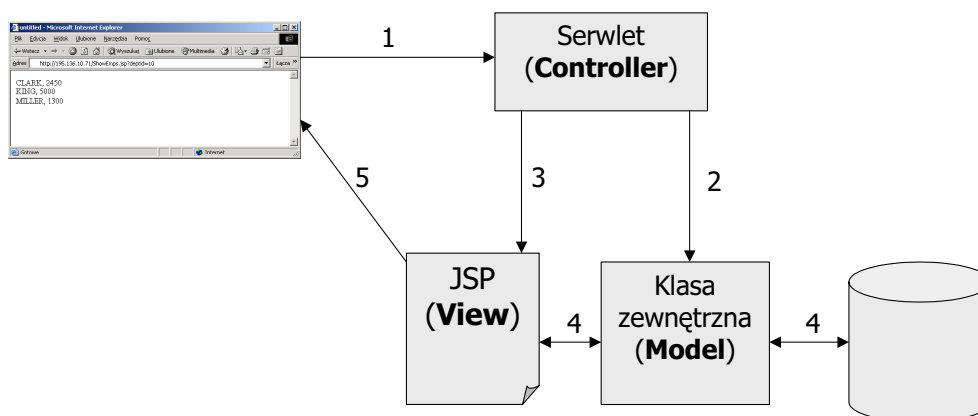
1. Wprowadzenie

Wzrastająca popularność języka Java sprawia, że coraz większa liczba projektantów i programistów aplikacji biznesowych zwraca swoją uwagę ku technologii J2EE. Takie walory, jak przenoszalność kodu, bogactwo dostępnych bibliotek oraz duża wydajność pracy aplikacji, powodują, że technologia J2EE zaczyna być postrzegana jako alternatywa dla innych popularnych platform rozwoju oprogramowania, m.in. Oracle Forms. Niestety, wciąż znaczącą barierą dla projektantów jest wysoki koszt tworzenia i pielęgnacji oprogramowania realizowanego w technologii J2EE. Ostatnie lata przyniosły jednak wiele nowych ciekawych rozwiązań, które, z jednej strony, skracają czas tworzenia oprogramowania poprzez automatyczną generację kodu, a z drugiej strony, ułatwiają pielęgnację oprogramowania poprzez wymuszanie modułowości tworzonych aplikacji. Jednym z rozwiązań godnych uwagi jest koncepcja architektury Model-View-Controller (MVC), która nakłania programistę do wyraźnego separowania funkcjonalności komponentów modułowej aplikacji, a także oferuje interesujący sposób podziału ról tych komponentów.

2. Architektura Model-View-Controller

Architektura MVC zakłada podział komponentów systemu aplikacyjnego na trzy kategorie: (1) komponenty typu Model, to komponenty reprezentujące dane i funkcje logiki biznesowej, na których operują aplikacje, (2) komponenty typu View, to komponenty reprezentujące wizualizację (prezentację) danych dla użytkownika – pobierają dane od komponentów typu Model, a następnie wyświetlają je na ekranie użytkownika, (3) komponenty typu Controller, to komponenty przechwytyjące żądania użytkowników i odwzorowujące je w wywołania metod komponentów typu Model – następnie komponenty typu Controller przekazują sterowanie do komponentów typu View. W języku polskim wspomniane trzy typy komponentów składowych bywają nazywane odpowiednio: Model – komponenty modelu, View – komponenty widoku lub prezentacji, Controller – komponenty sterujące lub kontrolera.

Realizacja architektury MVC na platformie J2EE angażuje zwykle trzy technologie implementacyjne: komponenty typu Controller są budowane jako serwlety Java, komponenty typu View – jako strony JavaServer Pages, a komponenty typu Model – jako moduły JavaBeans, Enterprise JavaBeans lub podobne. Schemat wewnętrzny przykładowej aplikacji przedstawiono na rys. 1. Przepływ sterowania w przykładowej aplikacji jest następujący. Przeglądarka WWW wysyła sparametryzowane żądanie uruchomienia serwletu Controller (1). Po wywołaniu, serwlet Controller analizuje żądanie i tworzy obiekty wymagane do dalszego przetwarzania żądania – m.in. obiekty klas zewnętrznych JavaBeans realizujących funkcję Model. W obiektach tych wywoływane są funkcje logiki biznesowej (2). Następnie, serwlet Controller przekazuje żądanie do odpowiedniej strony JavaServer Pages, realizującej funkcję View (3). Komponent View pobiera wyniki pracy funkcji logiki biznesowej z obiektów Model (4), a następnie generuje wynikowy dokument dla użytkownika (5).



Rys. 1. Schemat przykładowej aplikacji w architekturze MVC

Przeanalizujemy kod źródłowy przykładowej aplikacji z rys. 1. Poniżej przedstawiono komponent JavaBean realizujący funkcję Model. Komponent został zaimplementowany w postaci klasy Java o nazwie DBBean, posiadającej trzy metody. Metoda „connect()” służy do nawiązania połączenia z bazą danych. Metoda „setDeptNo()” umożliwia wskazanie departamentu, którego dotyczyć będą pobierane dane pracowników. Metoda „getEmps()” pobiera z bazy danych tablicę obiektów String opisujących pracowników wcześniej wskazanego departamentu.

```

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;
    }
}
  
```

Kolejnym komponentem jest serwlet Controller, odpowiadający za odbiór żądań użytkowników. Poniżej przedstawiono kod źródłowy serwletu Java, nazwanego „Controller”. Po uruchomieniu, serwlet tworzy obiekt klasy DBBean i wywołuje w nim metody „connect()” i „setDeptNo()”. W ten sposób obiekt Model jest przygotowany do udostępnienia danych o pracownikach. W ostatnim wierszu następuje przekierowanie do komponentu View, odpowiadającego za prezentację wyników.

```
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");
    }
}
```

Komponent View odpowiada za wizualizację danych przygotowanych przez obiekt Model. Poniżej przedstawiono kod źródłowy strony JavaServer Pages realizującej funkcję View. W wierszu 9. pobierany jest wcześniej utworzony obiekt Model, na którym następnie wywoływana jest metoda „getEmps()”, zwracająca listę pracowników zatrudnionych w wybranym departamencie. Dane pobranych pracowników są wyświetlane na ekranie przeglądarki WWW użytkownika końcowego.

```
<%@ page contentType="text/html; charset=windows-1250" %>
<%@ page import="java.util.Vector;" %>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html">
<title>JSP - View Component</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>
```


2.1. Zalety i wady architektury Model-View-Controller

Architektura MVC umożliwia łatwą zmianę interfejsu użytkownika w istniejącej aplikacji. Operacja taka wymaga wyłącznie wymiany składników typu View i/lub Controller, natomiast komponenty typu Model pozostają niezmienione. W ten sposób interfejs użytkownika może być także wzbogacany o alternatywne formy prezentacji danych i interakcji z użytkownikiem końcowym.

Warto wspomnieć o łatwiejszym testowaniu logiki biznesowej aplikacji zbudowanej zgodnie z założeniami architektury MVC. Możliwe jest np. skonstruowanie środowiska symulatorów testujących, które bezpośrednio wywołują funkcje komponentów typu Model, pomijając warstwy związane z interfejsem użytkownika.

Istotna jest także dostępność wielu bibliotek i środowisk wspomagających dziś realizację tego typu aplikacji. Programista, który wybiera architekturę MVC, może korzystać z gotowych, konfigurowalnych komponentów typu Controller, co powoduje skrócenie czasu implementacji systemu i podniesienie jego niezawodności.

Istnieją jednak również pewne wady omawianego rozwiązania. Stosowanie architektury MVC zwiększa złożoność projektu, zwielfokrotniając liczbę komponentów, które muszą zostać zaimplementowane. Należy też pamiętać o tym, że ze względu na ścisłe sprzężenie komponentów na poziomie interfejsów funkcyjnych wszelkie zmiany takich interfejsów w komponentach typu Model pociągają za sobą konieczność modyfikacji kodu komponentów typu View i Controller.

3. Apache Struts

Budowę aplikacji w architekturze MVC ułatwiają istniejące biblioteki i środowiska typu „framework”. Najpopularniejszymi z nich są: Apache Struts, J2EE BluePrints Web Application Framework, JavaServer Faces. Od kilku lat największym uznaniem programistów cieszy się Apache Struts. Temu środowisku poświęcono dalszą część rozdziału.

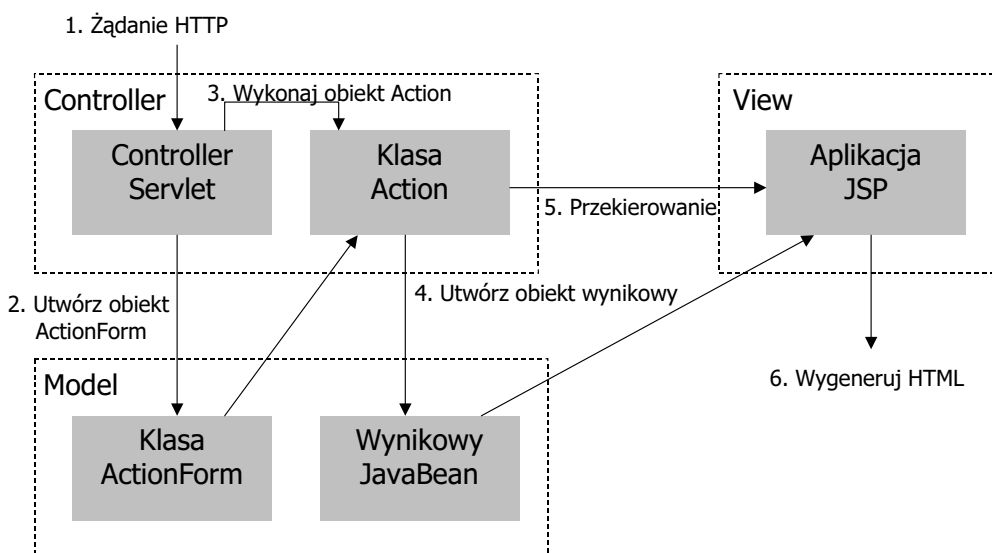
Apache Struts jest szkieletową implementacją aplikacji J2EE, zgodną z architekturą MVC. Apache Struts zostały opracowane przez Craiga McClanahana, a w roku 2000 подарowane Apache Software Foundation. Na uwagę zasługuje fakt, że Apache Struts stanowią integralną część standardowych bibliotek i narzędzi Oracle JDeveloper 10g. W skład środowiska wchodzi m.in.: konfigurowalny serwlet Controller, klasy biblioteczne Java do wykorzystania 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. Zarys architektury aplikacji Apache Struts przedstawiono na rys. 2.

Centralnym składnikiem Apache Struts jest serwlet ActionServlet, stanowiący pojedynczy punkt wejścia do całego systemu aplikacyjnego. Serwlet ActionServlet korzysta z pliku konfiguracyjnego „struts-config.xml”, w którym zapisane są ścieżki nawigacyjne oraz referencje do wszystkich klas Action i ActionForm. Na podstawie definicji z pliku „web.xml”, serwlet ActionServlet reaguje na wszystkie żądania HTTP, których adres URL posiada rozszerzenie „.do”.

Zadaniem programisty jest przygotowanie klas obsługi żądań, dziedziczących z klasy bibliotecznej Action. Gdy serwlet ActionServlet odbiera żądanie użytkownika, tworzy obiekt klasy obsługi żądań i wywołuje jego metodę „execute()” lub „perform()”. Zadaniem programisty jest implementacja jednej z tych metod w taki sposób, aby zwracały one obiekt klasy ActionForward, wskazujący stronę JSP, do której powinno zostać przekazane sterowanie. Implementując klasę obsługi żądań, programista posiada dostęp do: nagłówka żądania HTTP, parametrów wywołania, obiektów JavaBeans o zasięgu application/session/request, obiektu ActionForm opcjonalnie przekazanego przez serwlet ActionServlet, obiektu HttpServletResponse.

W celu propagacji parametrów wywołania aplikacji, programista tworzy klasy parametrów, dziedziczące z klasy bibliotecznej ActionForm. Obiekty klas parametrów są automatycznie wypełniane parametrami wywołania pochodzącymi od użytkownika. Każdy parametr jest zapisywany

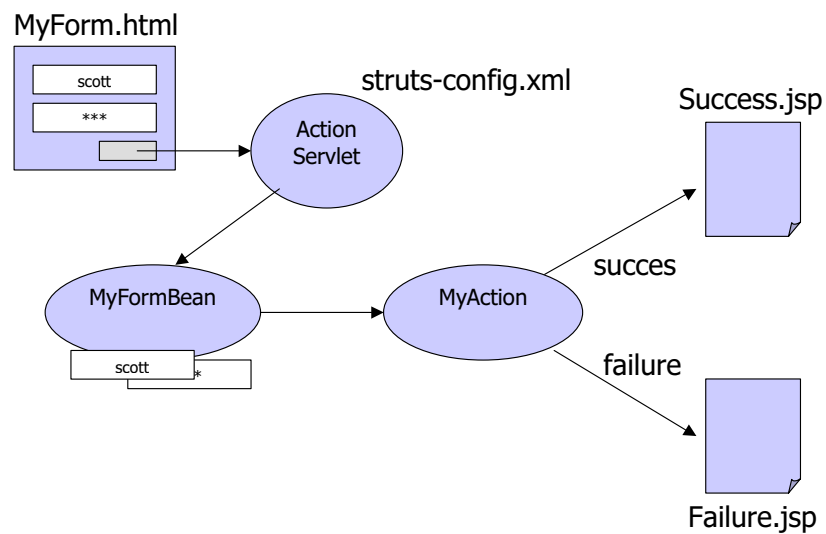
w obiekcie `ActionForm` za pomocą metody „`setXXX()`”, gdzie „`XXX`” to nazwa parametru. Implementacja takich metod jest zadaniem dla programisty. Obiekt `ActionForm` jest tworzony przez `ActionServlet`, a po wypełnieniu danymi jest przekazywany do obiektu obsługi żądań (metoda „`execute()`”). Klasa parametrów nie zawiera kodu przetwarzania danych, a jedynie wspomniane metody „`setXXX()`” oraz metody „`getXXX()`”, służące do odczytu odebranych parametrów wywołania. Klasy parametrów umożliwiają także łatwą walidację.



Rys. 2. Architektura aplikacji Apache Struts

4. Aplikacja przykładowa

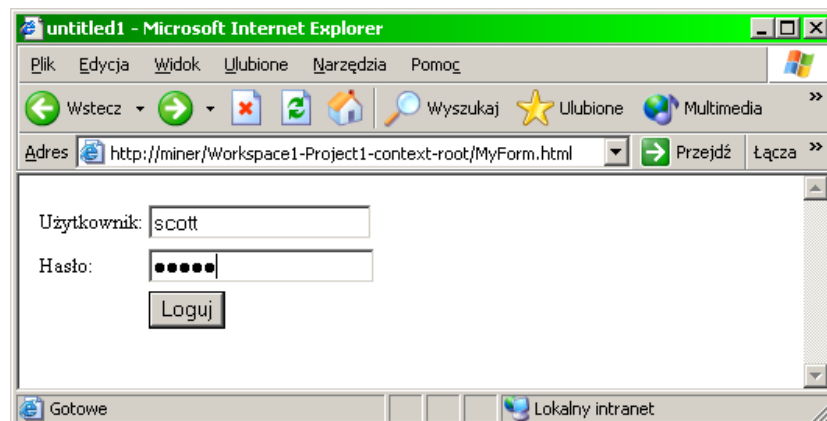
Rozważmy następujący przykład prostej aplikacji Apache Struts. Aplikacja posłuży do pobrania nazwy i hasła od użytkownika, zweryfikowania poprawności wprowadzonych danych, wreszcie do wyświetlenia jednego z dwóch komunikatów – o pomyślnym zalogowaniu lub o błędzie logowania. Schemat aplikacji przedstawiono na rys. 3. Aplikacja składa się z formularza HTML („`MyForm.html`”), za którego obsługę odpowiada serwlet `ActionServlet`. Serwlet `ActionServlet` zapisuje otrzymane parametry wywołania w obiekcie parametrów („`MyFormBean`”), który z kolei przekazuje obiektowi obsługi żądań („`MyAction`”). Obiekt obsługi żądań dokonuje weryfikacji nazwy i hasła użytkownika, a następnie przekazuje sterowanie do strony `JavaServer Pages` oświadczającej o pomyślnym zalogowaniu („`Success.jsp`”) lub o niepowodzeniu („`Failure.jsp`”).



Rys. 3. Schemat przykładowej aplikacji

Poniżej przedstawiono kod źródłowy formularza „MyForm.html”. Formularz zawiera dwa pola, „user” i „pass”.

```
<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>
```



Kolejnym składnikiem jest klasa parametrów „MyFormBean”. Jej kod źródłowy przedstawiono poniżej. Klasa posiada dwie metody zapisu i dwie metody odczytu parametrów wywołania pochodzących od wcześniej omówionego formularza.

```
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; }
}

```

Poniżej przedstawiono kod źródłowy klasy obsługi żądań „MyAction”. Klasa ta zawiera metodę „execute()”, która zostanie wywołana przez serwlet ActionServlet. Metoda „execute()” dokonuje prostej weryfikacji poprawności nazwy i hasła użytkownika.

```

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");
    }
}

```

Klasa obsługi żądań przekazuje sterowanie do jednej z dwóch stron JavaServer Pages, „Success.jsp” lub „Failure.jsp”. Kod źródłowy obu stron zamieszczono poniżej. Zauważmy, w jaki sposób strona JavaServer Pages uzyskuje dostęp do obiektów przekazywanych przez obiekt obsługi żądań.

Success.jsp

```

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

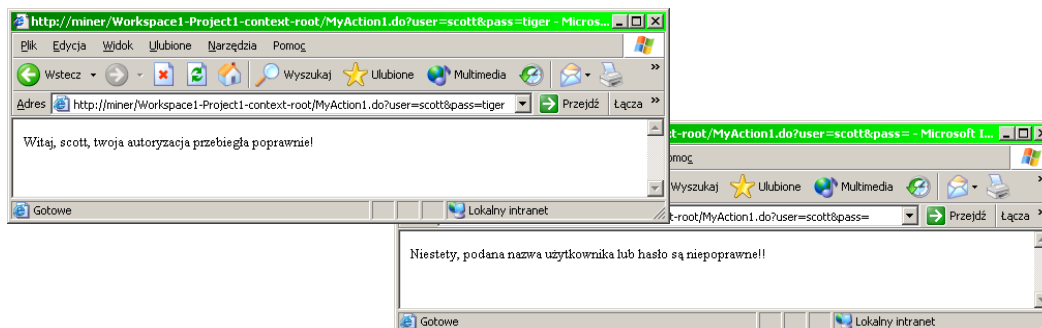
```

Failure.jsp

```

Niestety, podana nazwa użytkownika lub hasło są niepoprawne!!

```



Ostatnim składnikiem przykładowej aplikacji jest plik konfiguracyjny „struts-config.xml”, który steruje pracą serwletu Controller. W pliku tym znajduje się m.in. opis nawigacji – powiązanie etykiet „success” i „failure” wykorzystywanych przez klasę obsługi żądań z konkretnymi stronami

JavaServer Pages. Ponadto, zawiera on deklaracje interakcji z użytkownikiem końcowym. Plik „struts-config.xml” został przedstawiony poniżej.

```
<?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>
```

5. Podsumowanie

W artykule przedstawiono zarys koncepcji architektury Model-View-Controller, która coraz częściej bywa doceniana przez twórców aplikacji J2EE. Wpływ na powodzenie takiego podejścia do budowy systemów ma niewątpliwie dostępność bibliotek i szkieletowych środowisk dla programistów, np. Apache Struts. Twórcy aplikacji powinni jednak w pełni rozumieć zarówno zalety, jak i wady wynikające ze stosowania przedstawionych zaleceń projektowych.

6. Literatura

- [1] Alur, D., Crupi, J., Malks, D., Core J2EE Patterns: Best Practices and Design Strategies, Prentice Hall, 2003,
- [2] Burbeck, S., Application Programming in Smalltalk-80: How to use Model-View-Controller (MVC), <http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html>
- [3] Seshadri, G., Understanding JavaServer Pages Model 2 Architecture, <http://www.javaworld.com/javaworld/jw-12-1999/jw-12-ssj-jspmvc.html>