Zastosowanie JSP jako techniki generatywnej do realizacji części systemów w ramach linii produktów

Silva Robak

Politechnika Zielonogórska, Instytut Informatyki i Zarządzania, 65-246 Zielona Góra, ul. Podgórna 50, tel. 68 3275401, e-mail: Robak@pz.zgora.pl

Roman Stryjski; Bogdan Franczyk
Wyższa Szkoła Pedagogiczna, Instytut Techniki, 65-762 Zielona Góra al. Wojska Polskiego 69,
e-mail: stryjski@post.pl;

Streszczenie

W artykule przedstawione zostało nowatorskie podejście pokazujące możliwości wykorzystania nowoczesnych technologii tworzenia aplikacji dla sieci WWW, w procesie rozwoju oraz podczas ewolucji systemów oprogramowania wchodzących w skład linii produktów. Budowane z komponentów systemy mogą być rozwijane z uwzględnieniem zasad ponownego wykorzystywania poprzez odpowiednie dostosowywanie starannie wypracowanej i specyficznej dla danej dziedziny infrastruktury w postaci architektury generycznej. Podejście takie wydaje się stanowić najbardziej obiecującą drogę prowadzącą do wzrostu jakości i produktywności w dziedzinie tworzenia oprogramowania dla rodzin systemów. Do realizacji i dopasowywania komponentów infrastruktury dla linii produktów w artykule pokazano możliwości i ograniczenia oraz przykład zastosowania technologii wchodzącej w skład platformy programowej Java, tzw. JSP (Java Server Pages).

1. Zastosowanie techniki generatywnych w rozwoju rodzin systemów

Rozwijanie rodziny podobnych systemów zamiast tworzenia kolejnych (wersji) pojedynczych systemów może przynieść wiele korzyści takich jak zwiększenie wydajności i jakości oprogramowania oraz skrócenie czasu dostarczania gotowych produktów na rynek (*time-to-market*) [2]. O ile właściwości wspólne dla rodziny systemów są raczej natury technicznej, to wspólne właściwości konkretnych produktów (jako wystąpień pewnej linii produktów) mają służyć zaspakajaniu specyficznych wymagań związanych z wybranym rynkiem lub przedsięwzięciem. Zebranie informacji o wspólnych właściwościach systemów pozwala na rozwijanie ponownie używalnych podstawowych aktywów. Informacji o częściach wspólnych i różnicach występujących pomiędzy członkami rodziny [1, 3] mogą zostać użyte do projektowania i rozwoju wspólnej generycznej architektury dla linii produktów wywodzącej się z jednej (lub wielu) rodzin systemów [1]. Podczas konfiguracji na podstawie architektury tworzone są egzemplarze systemów. Plan produkcji opisujący sposób, w jaki konkretne systemy mogą być tworzone na podstawie komponentów wchodzących w skład generycznej (lub wysoce elastycznej) architektury może uwzględniać trzy stopnie automatyzacji [4]:

- Montaż manualny – architektura i komponenty uzupełniane są odpowiednią dokumentacją dla użytkownika, a aplikacje tworzone są manualnie na bazie istniejących komponentów Jest to obecny stan w obiektowej inżynierii oprogramowania.
- Montaż wspomagający częściową automatyzację – montaż komponentów jest wspomagany różnymi narzędziami np. do przeglądania i wyszukiwania komponentów, czy generatorami automatyzującymi wybrane aspekty rozwoju aplikacji [5]. Przykładem aplikacji reprezentującej ten poziom jest SanFrancisco – IBM [7].
- Montaż zautomatyzowany – zbiór narzędzi wspomaga proces zamawiania aplikacji przez klienta, a cała aplikacja może zostać w większości wygenerowana na podstawie zamówienia. Jest to podejście zgodne z zasadami programowania generatywnego.

Celem podejścia generatywnego jest zapewnienie odpowiednich środków do wyboru i tworzenia konkretnych członków rodziny na podstawie wyspecyfikowanych właściwości w możliwie zautomatyzowany sposób. Zakłada to miedzy innymi modelowanie wiedzy konfiguracyjnej do odpowiedniego poziomu szczegółowości umożliwiającego dalszą automatyzację (implementację z udziałem generatorów). Konkretne systemy mogą być specyfikowane za pomocą

tekstowych lub graficznych języków specyficznych dla dziedziny DSLs [11]. Zasady leżące u podstaw programowania generatywnego to oddzielanie wątków (*Separation of concerns* - SoC), parametryzacja różnic, analiza i modelowanie zależności i współpracy oraz oddzielenie przestrzeni problemu od przestrzeni rozwiązania. Do odwzorowań pomiędzy przestrzenią problemu, a przestrzenią rozwiązania stosowana jest wiedza konfiguracyjna. Przestrzenią problemu określany jest przy tym zbiór wszystkich istotnych kombinacji właściwości w pewnej dziedzinie (tj. istotnych specyfikacji), natomiast zbiór wszystkich konkretnych komponentów występujących w tej dziedzinie, wraz z ich możliwymi konfiguracjami nazywany jest przestrzenią rozwiązania.

Proces (częściowego) generowania aplikacji może być wspomagany przez interaktywne (możliwie graficzne) narzędzia pozwalające użytkownikowi na unikanie (wzgl. szybkie korygowanie) błędów i uwzględniające możliwie wszystkie elementy modelowania wiedzy konfiguracyjnej tj. ograniczenia kombinacji właściwości, ustawienia standardowe, standardowe zależności, reguły kompozycji i optymalizacje [6].

W artykule pokazane zostanie podejście generatywne mające na celu automatyczne tworzenie wielu podobnych systemów (wariantów systemów) na podstawie generycznego kodu. W tym celu autorzy proponują wykorzystanie ogólnie dostępnych mechanizmów generacyjnych przeznaczonych pierwotnie do tworzenia aplikacji internetowych. W punkcie 2 przedstawione zostaną podstawowe technologie umożliwiające tworzenie aplikacji internetowych. W punkcie 3 pokazane zostanie na przykładach jak strony Java Server Pages (JSP) [10] używane standardowo do dynamicznego generowania stron HTML mogą zostać wykorzystane w celu generowania programów Javy [8] oraz jak parametryzować fragmenty generowanego kodu poprzez ustawianie odpowiednich właściwości komponentów JavaBeans.

2. Technologie stosowane przy tworzeniu aplikacji internetowych

Języki i techniki stosowane do tworzenia klasycznych witryn oraz aplikacji internetowych mogą być stosowane zarówno po stronie serwera (*server-side*), jak i po stronie klienta (*client-side*). Technikami stosowanymi po stronie klienta są np. aplety odpowiadające za prezentację danych w przeglądarce oraz interakcję z użytkownikiem. Po stronie serwera stosowane są np. serwlety, podczas wykonywania których na serwerze tworzone są dynamicznie strony zawierające jedynie stosownie sformatowany kod HTML. Strony te są następnie przenoszone do przeglądarki i tam prezentowane. Przetwarzanie serwletów odbywa się w

sposób bardziej wydajny niż np. skryptów CGI (wymagających rozpoczęcia osobnego procesu dla każdego zapytania), co w pełni rekompensuje wolniejsze wykonywanie kodu Javy niż np. kodu języka C/C++. Problemem związanym z przetwarzaniem skryptów wg protokołu CGI jest też konieczność restartowania serwera, za każdym razem, gdy kompilowany moduł ulega zmianie.

Koncepcje oparte na kompilowanych modułach i skryptach umożliwiają aplikacjom internetowym reagowanie na dane wprowadzane przez użytkownika pozwalając m.in. na dostęp do baz danych. Interpretowane języki skryptowe charakteryzują się prostotą i łatwością użycia pomijając aspekty sprawdzania poprawności składniowej - zgodności typów itp. Skrypty po stronie klienta są wbudowane w stronę WWW i wykonywane przez przeglądarkę. Pozwalają one jedynie na wstępną weryfikację danych (np. sprawdzenie, czy wypełniono wszystkie obowiązkowe pola formularza) wysyłanych następnie do serwera. Ich zaletą jest możliwość odciążenia serwera. Stosowane po stronie klienta JavaScript i Visual Basic Script bazują one na odpowiednich językach programowania Java (Sun) i Visual Basic (Microsoft).

Po stronie serwera do przetwarzania żądań klienta stosowane są serwlety, Java Server Pages JSP i Active Server Pages ASP oraz Cold Fusion CFA. Serwlety zaliczane są zwyczajowo do skryptów, mimo iż są to małe, niezależne od platformy klasy Javy skompilowane do poziomu neutralnego kodu bajtowego (*byte code*), który może być dynamicznie ładowany i wykonywany przez serwer WWW [9]. JSP (Sun) i ASP (Microsoft) i CFA (Allaire) to analogiczne technologie konkurujących ze sobą firm. JSP jako technologia wywodząca się z Javy pozwala na zachowanie niezależności od platformy i serwera oraz wykorzystywanie dalszych zalet Javy takich jak możliwość szybkiego tworzenia aplikacji, ich bezpieczne przetwarzanie w środowiskach rozproszonych oraz wysoka pielęgnowalność aplikacji.

3. Zastosowanie JSP do generowania kodu Javy

Serwlet jest komponentem WWW zarządzanym przez kontener tzw. Servlet Engine generujący dynamiczną treść. Serwlety komunikują się z klientami poprzez kontener serwletów zgodnie z protokołem HTTP [12]. Serwlety realizowane są przy użyciu biblioteki Servlet-API.

Java Server Pages JSP [10] jest alternatywnym rozwinięciem i uzupełnieniem technologii serwletów. Strona JSP jest plikiem tekstowym zawierającym specjalne znaczniki JSP (<% ... %>) zawierające wewnątrz odwołania do kodu Javy - obok znaczników HTML lub XML [13]. Fragmenty kodu zawarte wewnątrz znaczników JSP stanowią dynamiczną część kodu,

podczas gdy część HTML zawiera jego statyczną część. Zapytanie klienta żądające strony JSP powoduje przetłumaczenie jej na serwlet (o ile nie była uprzednio przetłumaczona) przez tzw. JSP-Engine, a następnie wykonanie serwletu i zwrot wyniku do klienta.

Do podstawowych elementów będących znacznikami JSP należą dyrektywy (<%@ *directive* %>) ustalające globalne informacje dla całej strony np. <%@ page language= ”java” %> ustalająca, że językiem do którego odnoszą się odwołania będzie Java. Deklaracje: <%! *declaration* %> (np. <%! String HelloWorld; %>) obowiązują dla zmiennych i operacji w obrębie całej strony. Skryplety zawierają kod źródłowy Javy (<% *code fragment* %>), a wyrażenia (<%= *expression* %>) zawierają zmienne, których wartości po konwersji na stringi zostaną wyprowadzone na wyjście przez JSP. Komentarze możliwe są na kilku poziomach widoczności związanej z procesem przetwarzania strony: komentarze wyprowadzane na wyjście (<!- *comment* ->), komentarze widoczne tylko w kodzie JSP (<%- *comment* -&>), oraz komentarze Javy, które są usuwane podczas kompilacji .

W skrypletach można używać wielu rodzajów predefiniowanych obiektów. W procesie generowania kodu Javy przydatne są m.in. obiekty:

- request z podklasy javax.servlet.ServletRequest zawierający informacje dotyczące zapytania (źródło, parametry, cookies itd.),
- response z podklasy javax.servlet.ServletResponse,
- out z Jsp.Writer - strumień wyjściowy, do którego generowana jest strona,
- session z klasy javax.servlet.http.HttpSession,
- exception z klasy java.lang.Throwable, itd.,

a pochodzących z klas związanych z technologią serwletów. Skryplety dołączane są do operacji _jspService Servletu, a zmienne i operacje wymienione w deklaracjach są tłumaczone na zmienne i operacje serwletu.

Wydruk 1 zawiera prostą stronę JSP przeznaczoną do wygenerowania strony HTML zawierającej tekst “Hello World” oraz wartość ew. przekazywanego parametru. Stosowana jest przy tym jest większość z opisanych wyżej elementów strony JSP. Do wyprowadzania wartości parametru można było również użyć wyrażenia tak jak w przypadku stringu "Hallo World", wzgl. komunikatu `print` do obiektu `out`.

Wydruk 2 zawiera stronę JSP wykorzystaną do wygenerowania kodu klasy Javy o takiej samej funkcjonalności jak kod na Wydruku 1. Jak pokazano, możliwe jest więc wykorzystanie techniki JSP do generowania nie tylko stron HTML, ale również np. programów Javy. Generowane programy Javy mogą ponadto zawierać parametryzację potrzebną następnie do realizacji konkretnych wariantów kodu. W tym celu można wykorzystywać możliwości samego języka

Java, jak i stosować podejście komponentowe. Główną zaletą JSP jest możliwość przenoszenia części kodu źródłowego do innych komponentów integrowanych w strony JSP, tzn. do JavaBeans. Za pomocą odpowiednich znaczników takich jak <jsp:useBean> możliwy jest dostęp do istniejącego obiektu komponentu typu JavaBean wewnątrz strony JSP (lub utworzenie nowego obiektu). Znacznik <jsp:plugin> umożliwia załadowanie JavaBean (lub appletu) do systemu komputerowego klienta, zaś <jsp:setProperty> służy do ustawiania wartości właściwości egzemplarzy JavaBean. Dołączanie do strony JSP danych z zewnętrznego pliku umożliwia <jsp:include> (ale dane takie nie są potem sprawdzane przez parser).

W kolejnym przykładzie pokazujemy jak parametryzować fragmenty generowanego kodu przez ustawianie odpowiednich właściwości komponentów JavaBeans przez strony JSP (Wydruk 3, Wydruk 4 i Wydruk 5). Wyświetlanie zawartości niektórych witryn wymaga autoryzacji, która jest przeprowadzana za pomocą uprzednio przygotowanego okna logowania. W tym przykładzie od użytkownika wymagane jest jedynie podanie hasła, które dla uproszczenia kodu zostało umieszczone wewnątrz komponentu LogowanieBean.Java (Wydruk 4). Strona JSP o nazwie Witaj.jsp (zob. Wydruk 3) wyświetla stosowny formularz zawierający pole edycyjne typu tekstowego i poprzez atrybut `value` znacznika `input` odczytuje wpisane do formularza hasło. Po przyciśnięciu przycisku "Zaloguj" wywoływana jest strona Logowanie.jsp (zob. Wydruk 5), a odczytane hasło przekazywane jest do niej metodą `post`. Strona Logowanie.jsp aktywuje komponent JavaBean LogowanieBean (<jsp:useBean> ustala, że do egzemplarza komponentu JavaBean plik JSP odwołuje się za pomocą nazwy `log`). Strony JSP i komponent `log` (JavaBean) wymieniają ze sobą dane: wpisane przez użytkownika hasło nazwane jest w tym przykładzie `haslo`, a poprzez zastosowanie <jsp:setProperty> przekazywane jest ono atrybutowi `wpis` komponentu `log`. Komponent sprawdza poprawność hasła i poprzez metodę `getPrawidlowyWpis()` zwraca stosowny wynik. Strona Logowanie.jsp (Wydruk 5) dynamicznie generuje kod HTML w zależności od zwróconego wyniku. W przypadku podania poprawnego hasła wita użytkownika (tzn. generowana jest strona zawierająca napis "Witam!), lub gdy jest ono niepoprawne następuje powrót do strony Witaj.jsp (Wydruk 3) i wyświetlany jest napis "Spróbuj ponownie".

Pokazane mechanizmy mogą zostać wykorzystane nie tylko do generowania stron HTML, ale jak to pokazano w przykładzie z Wydruk 2 również do generowania różnych wariantów kodu Javy.

4. Wnioski

W pracy zostały pokazane i zbadane możliwości wykorzystania technologii serwletów i JSP do dynamicznego generowania kodu Javy. Strony JSP zawierające mieszaninę HTML (wzgl. XML) i języka skryptowego mogą wydawać się nieprzejrzyste i trudne w śledzeniu dla człowieka, ale nie ma to znaczenia, jeśli JSP zastosowane zostanie jako baza służąca do automatycznego generowania programów (np. kodu Javy). Zastosowanie języka XML będącego szeroko uznanym standardem do wymiany danych umożliwiłoby także korzystanie z wielu dostępnych standardowych narzędzi (edytorów, parserów itd.).

W komponentowym podejściu do tworzenia aplikacji strony JSP mogą również pełnić rolę elementu spajającego poszczególne jednostki funkcyjne.

Techniki generatywne wykorzystujące JSP mogą umożliwić projektantom dostosowywanie kodu stanowiącego generyczną architekturę dla linii produktów dzięki temu, że poprzez generowanie kodu pozwalają na automatycznie umieszczanie wybranych wariantów w konkretnym egzemplarzu. Ponadto dodatkową zaletą wynikającą z zastosowania JSP do generowania kodu jest możliwość przekazywania plików poprzez internet. Natomiast wadą takiego rozwiązania jest brak możliwości stosowania przetwarzania rekurencyjnego.

Zaproponowane przez autorów podejście proponujące zastosowanie technologii JSP do celów generacyjnych pozwala na zachowanie niezależności od platformy i serwera oraz wykorzystywanie dalszych zalet Javy takich jak możliwość szybkiego tworzenia aplikacji, ich bezpieczne przetwarzanie w środowiskach rozproszonych oraz wysoka pielęgnowalność aplikacji. To wszystko przyczynia się do spełniania specyficznych wymagań stawianych przez nowoczesne techniki projektowania, platformy komponentowe (Enterprice JavaBeans) i nowoczesne dziedziny aplikacji takie jak systemy rozproszone lub handel w internecie.

• Bibliografia

[1]. J. Bosch. *Design and Use of Software Architectures*. Adopting and evolving product-line approach. Addison-Wesley, 2000.

[2]. P. Clements, L.M. Northrop. A Framework for Software Product Line Practice - Version 2.0 (online). Pittsburgh, PA: *Software Engineering Institute, Carnegie Mellon University*, July 1999. http://www.sei.cmu.edu/plp/framework.html

[3]. J. Coplien, D. Hoffman and D. Weiss. Commonality and variability in Software Engineering. *IEEE Soft*ware, November/December, pp. 37-45,1998.

[4]. K.Czarnecki, U. Eisenecker. *Generative Programming Methods, Tools and Applications*. Addison-Wesley, 2000.

[5]. S. Robak, B. Franczyk. Obiektowe ewolucyjne podejście do tworzenia rodzin systemów z zastosowaniem struktur aplikacji. *Studia z Automatyki i Informatyki* Tom 26 – 2001, 2001.

[6]. S. Robak, B. Franczyk. Feature Interaction and Composition Problems in Software Productlines. Position Paper in Workshop on Feature Interaction in Composed Systems. ECOOP (*European Conference on Object-Oriented Programming*) Budapest Hungary, 2001.

[7]. IBM SanFrancisco http://www-4.ibm.com/software/ad/sanfrancisco/

[8]. Sun Microsystems: Java Specification, http://www.java.sun.com

[9]. Sun Microsystems: Java Servlet Specification, http://www.java.sun.com/products/servlet/index.html

[10].Sun Microsystems: Java Server Pages Specification, http://www.java.sun.com/products/servlet

[11].A. Van Deursen Domain-Specific Languages vs. Object-Oriented Frameworks: A Financial Engineering Case Study", STJA'97 Conference Proceedings, Technical University of Ilmenau, s. 35-39, 1977.

[12].Available WWW: <URL: http://www.w3.org>

[13].Extensible Markup Language (XML) 1.0 REC-xml-19980210, February 1998.

Wydruki:

```
<HTML><HEAD> <TITLE> Przykład Hello World </TITLE> </HEAD>
<BODY>
<!- Globalne ustawienia dla strony ->
<%@ page language= „java" %>
<!- Deklaracja stringu HelloWorld ->
<%! String HelloWorld; %>
<!- skryplet w Javie ->
<% HelloWorld = << Hello World>>; %>
<!- wypisanie stringu na stronie HTML ->
<%= HelloWorld %>
```

```
<% String n = request.getParameter(“n”);
        if (x==null) {
%>
         Bez parametru
        <% } else  { %>
        Parametr n wynosi  <% =n%>
<%  } %>
</BODY> </HTML>
```

Wydruk 1

```
class GenerowanyProgram {
        public static main (String[] args) {
                System.out.println("HelloWorld ");
<% String n = request.getParameter(“n”);
        if (x==null) {
%>
        System.out.println("Bez parametru");
<% } else  { %>
        String n = "<%=n%>;
        System.out.println("Parametr n wynosi  " +n);
<%  } %>
}
```

Wydruk 2

```
<jsp:useBean id="log" scope="session" class="demo.LogowanieBean"/>
<html>
<form action="Logowanie.jsp" method=post>
Podaj haslo <input type=text name="haslo" size=20><p>
            <input type=submit name="Zaloguj" value="Zaloguj"><br>
</form>
</body>
</html>
```

Wydruk 3

```
package demo;
import java.io.*;
import java.util.*;
import javax.servlet.http.*;
public class LogowanieBean {
        private boolean prawidlowyWpis = false;
        private String prawidloweHaslo;
        private String wpis;
        public LogowanieBean(){
          prawidloweHaslo = "java";
          wpis = null;
        }
        private String getWpis(){
          return wpis;
        }
        public void setWpis(String wpis){
          this.wpis = wpis;
        }
        public boolean setPrawidlowyWpis(){
          try{if (getWpis().equals(prawidloweHaslo))
                    prawidlowyWpis = true;
          else
                    prawidlowyWpis = false;
           }
           catch(Exception e) {}
          return prawidlowyWpis;
        }
        public boolean getPrawidlowyWpis(){
           return prawidlowyWpis;
        }
}
```

Wydruk 4

```
<jsp:useBean id="log" scope="session" class="demo.LogowanieBean"/>
<jsp:setProperty name="log" property="wpis" param="haslo"/>
</jsp:useBean>

<html>
<head> <title>Logowanie</title> </head>
```

```
<%  log.setPrawidlowyWpis();
      if (log.getPrawidlowyWpis())
      {     %> Witam! <%
      }
      else
      {     %>Haslo jest nieprawidlowe.<br>
           <a href ="Witaj.jsp"> Sprobuj ponownie</a> <%
      }
%>
</body>
</html>
```

Wydruk 5