STUDIA INFORMATICA 2009
Volume 30 Number 4 (87)

Krzysztof DOBOSZ, Arkadiusz LENART
Politechnika Śląska, Instytut Informatyki

PORÓWNANIE SZABLONÓW DLA APLIKACJI INTERNETOWYCH

Streszczenie. Artykuł opisuje porównanie pięciu szablonów projektowych, najczęściej stosowanych przez programistów. Porównano ich ogólne własności, możliwości konfiguracyjne, cechy warstwy prezentacji, obsługę formularzy i metody internacjonalizacji. Wskazano również wzorce projektowe najczęściej w nich wykorzystywane.

Słowa kluczowe: szablon aplikacji, wzorzec projektowy, aplikacja internetowa, warstwa prezentacji

COMPARING FRAMEWORKS FOR INTERNET APPLICATIONS

Summary. The article describes comparing five frameworks, most often applied by programmers. Their general properties, possible configurations, features of the presentation layer, the service of forms and methods of the internationalization are compared. Design patterns most often used in them are also shown.

Keywords: framework, design pattern, internet application, presentation layer

1. Wprowadzenie

Szablonem projektowym aplikacji (ang. *framework*) lub inaczej ramą projektową lub szkieletem nazywamy w programowaniu strukturę wspomagającą tworzenie, rozwój i testowanie powstającego oprogramowania. Z reguły na szablon składają się programy wspomagające, biblioteki kodu źródłowego i inne podobne narzędzia. Zadaniem szablonu jest stworzenie uogólnień wokół często powtarzanych zadań oraz mechanizmów i dostępu do nich. Jest to realizowane na dwa sposoby: przez automatyzację często powtarzanych, trywialnych zadań oraz przez dostarczanie eleganckiego rozwiązania architektonicznego dla powszechnie

stosowanych konstrukcji aplikacji internetowych. Zadaniami, które mogą być potencjalnie wykonywane automatycznie przez szablon, mogą być m.in.:

- sprawdzanie poprawności wprowadzanych danych,
- rozdzielenie zadań biznesowych od warstwy danych i warstwy prezentacji,
- internacjonalizacja,
- konwersja danych pomiędzy ich znakową reprezentacją w kodzie HTML a typami danych języka JAVA.

Bardziej istotnym zadaniem szkieletu projektowego, niż automatyzacja zadań, jest dostarczenie struktury, która pozwoli zapanować nad skomplikowanym charakterem aplikacji tworzonych dla platformy Java EE [1] i nie tylko, gdyż szablony mogą mieć zastosowanie także w platformie Java SE, a także innych środowiskach. Tworząc aplikacje sieciowe programista wcale nie jest zmuszony do wykorzystywania jakiegokolwiek szablonu. Dopóki aplikacja jest w miarę prosta, to takie rozwiązanie ma sens, gdyż nie pojawiają się większe problemy z architekturą aplikacji. Jednak gdy aplikacja staje się coraz bardziej złożona, użycie szablonu może być konieczne, tym bardziej że zazwyczaj przyspiesza to proces powstawania oprogramowania, a w perspektywie konieczności ukończenia projektu w określonym czasie jest to czynnik bardzo istotny.

2. Aplikacyjne szablony projektowe

Na rynku informatycznym znajduje się wiele szablonów aplikacyjnych. Stojąc przed koniecznością wyboru szablonu projektowego, z wykorzystaniem którego ma być implementowany nowy projekt, projektanci i programiści mają nie lada dylemat. Wiele z nich jest do siebie bardzo podobnych. Niektóre jednak dają możliwości nieudostępniane przez pozostałe szablony, a które czasem mogą być przydatne w procesie opracowywania określonych aplikacji. Tworzone aplikacje mają różne dziedziny zastosowania. Niektóre nie realizują skomplikowanych procesów biznesowych i koncentrują się na samym interfejsie użytkownika. Inne natomiast przeciwnie, wykonują bardzo skomplikowane procesy biznesowe, wymagające wykorzystania skomplikowanych mechanizmów, a sam interfejs użytkownika nie musi być bardzo rozbudowany. Wybór nie jest więc łatwy i zależy przede wszystkim od dziedziny zastosowania.

Najogólniej szablony projektowe możemy podzielić na zorientowane na akcjach oraz na komponentach. Szkielety zorientowane na akcjach powstały jako kombinacja pomysłów zaczerpniętych z technologii Java Servlets [2] oraz Java Server Pages [3]. Idea ta polega na rozdzieleniu przetwarzania żądania strony, którą widzi użytkownik w swojej przeglądarce internetowej, od części zarządzającej danymi i generującej wyniki, pozwalając każdej z nich

skupić się na zadaniach, które wykonuje najlepiej. W tym celu stosuje się wzorzec projektowy MVC (ang. *Model-View-Controler*), który zostanie przedstawiony w kolejnym rozdziale.

Odkąd aplikacje internetowe zaczęły stawać się coraz bardziej złożone, zdano sobie sprawę z tego, że strona internetowa nie jest już logicznie odseparowanym, niezależnym elementem. Strony zaczęły zawierać w swojej budowie wiele formularzy, odnośników do odświeżania zawartości i innych tym podobnych elementów, z których każdy wymagał wywołania fragmentu jakiegoś programu w celu wykonania ich zadań. Aby spełnić wymagania niesione przez tego typu aplikacje, powstały szablony zorientowane na komponentach i zaczęły one stawać się popularne wśród programistów aplikacji. Zapewniają one ścisłe powiązanie pomiędzy komponentami interfejsu użytkownika oraz obiektami, które reprezentują dane. Są one sterowane przez zdarzenia i bardziej zorientowane obiektowo od szablonów zorientowanych na akcjach. Dodatkową korzyścią, z tego typu szablonów jest to, że umożliwiają one ponowne wykorzystanie komponentów w wielu aplikacjach sieciowych dzięki wykorzystaniu standardu JavaBean.

Aby ułatwić zadanie wyboru szablonu dla aplikacji internetowej, w niniejszym artykule dokonano porównania rozwiązań najczęściej stosowanych, tj.: Java Server Faces [4], Jakarta Struts2 [5, 6], Spring [7, 8] oraz nieco mniej powszechnych, ale posiadających interesujące funkcjonalności: Tapestry [9] i JBoss Seam [10]. Porównań tych dokonano w opierając się na aplikacji sieciowej pozwalającej użytkownikowi korzystającemu z przeglądarki internetowej na dostęp do bazy danych, zaimplementowaną kilkukrotnie, za każdym razem z użyciem innego szablonu. Wykonane implementacje porównano z sobą. Szczegółowy opis badań został zawarty w pracy dyplomowej magisterskiej [11], a niniejszy artykuł prezentuje tylko najważniejsze wyniki.

3. Wzorce projektowe w szablonach aplikacyjnych

Tak jak wspomniano, wzorzec projektowy jest pomocny w opracowywaniu złożonych aplikacji internetowych. Każdy skomplikowany problem najlepiej jest rozwiązywać dokonując jego rozbioru na mniejsze zagadnienia, które z dużym prawdopodobieństwem będzie można łatwiej rozwiązać. Wzorce projektowe [12, 13] pomagają w rozwiązywaniu wielu powszechnie występujących, powtarzalnych problemów. Każdy z omawianych w pracy szablonów dla aplikacji internetowych stosuje kilka z powszechnie znanych wzorców projektowych, jednak na największą uwagę zasługują te wymienione w niniejszym rozdziale.

3.1. Wzorzec MVC

Każdy z omawianych w artykule szablonów aplikacyjnych dostarcza własną implementację wzorca projektowego MVC. Jeszcze parę lat temu powszechna była tendencja do tego, by oprogramowanie internetowe tworzone było zgodnie z takim modelem architektury oprogramowania, w którym serwlet bądź strona JSP odpowiadały praktycznie za całość prac niezbędnych w przetwarzaniu żądania. Nie dość, że komponenty te musiały wykonywać podstawowe swoje zadanie, tj. przygotować odpowiedź wysyłaną do użytkownika, to dokonywały także manipulacji danymi, sprawdzały ich poprawność i wykonywały algorytm aplikacji. Takie podejście miało jednak wiele wad i od pewnego momentu zaczęto stosować wzorzec MVC.

Wzorzec projektowy MVC to wzorzec strukturalny, którego zadaniem jest rozdzielenie funkcjonalności pomiędzy obiekty aplikacji w taki sposób, aby zminimalizować stopień powiązań między nimi. Aby to osiągnąć, aplikacja jest dzielona na trzy warstwy: modelu, widoku i kontrolera. Każda z nich ma za zadanie wykonywać określone dla siebie zadania i wywiązywać się z obowiązków wobec pozostałych warstw.

Model reprezentuje dane, razem z regułami biznesowymi oraz operacjami, które zarządzają dostępem do nich oraz pozwalają je modyfikować. Zadaniem modelu jest powiadamianie widoku o każdych zmianach, które w nim zachodzą, oraz zezwalanie widokowi na wykonywanie zapytań co do stanu modelu. Jego obowiązkiem jest także udostępnianie kontrolerowi dostępu do zawartych w modelu reguł biznesowych aplikacji.

Głównym zadaniem widoku jest wyświetlanie użytkownikowi zawartości modelu. Widok pobiera z niego dane oraz określa, w jaki sposób powinny być one reprezentowane. W przypadku gdy dane modelu ulegną zmianie, widok jest odpowiedzialny za odświeżenie swojej zawartości. Dodatkowym zadaniem widoku jest przekazywanie danych uzyskanych od użytkownika do kontrolera.

Kontroler definiuje zachowanie aplikacji. Przekazuje on żądania klienta i jest odpowiedzialny za wybór widoku do prezentacji klientowi. Przechwytuje dane wejściowe pochodzące od klienta i odwzorowuje je w obiekty tak, aby mogły być one przetwarzane przez model. W aplikacji internetowej danymi wejściowymi są żądania protokołu HTTP.

3.2. Wzorzec *Front Controller*

W czasie implementacji aplikacji internetowej może zaistnieć konieczność wplecenia takiego samego algorytmu sterującego w wielu miejscach tekstu źródłowego, np. w widokach. Przykładem takiej sytuacji może być konieczność zabezpieczenia dostępu do pewnej grupy stron WWW, do których dostęp powinni mieć tylko autoryzowani użytkownicy. Wstawianie takiego samego fragmentu oprogramowania w wielu miejscach znacznie utrudnia pielęgnację

kodu, a samo umieszczanie algorytmu sterującego we fragmencie oprogramowania, w którym tworzony jest widok, sprawia, że takie rozwiązanie jest mniej modułowe i mniej elastyczne.

Aby uniknąć opisywanych powyżej problemów, stosuje się wzorzec projektowy *Front Controller*. Wzorzec ten pozwala umieścić obsługę nadchodzących żądań w jednym, centralnym miejscu. Mimo że zalecana jest centralizacja obsługi, to wzorzec nie ogranicza liczby obiektów sterujących w systemie. Wykorzystująca go aplikacja może używać wielu takich obiektów, każdego z nich odwzorowując na różne usługi.

Zazwyczaj kontroler współpracuje z obiektem dyspozytora, który jest odpowiedzialny za zarządzanie widokami i nawigowaniem pomiędzy nimi. Dyspozytor może być zawarty wewnątrz kontrolera lub może być opracowany jako osobny komponent.

3.3. Wzorce IoC i DI

Wyjaśniając działanie wzorca odwracania sterowania IoC (ang. *Inversion of Controll*), przytaczana jest zazwyczaj tzw. zasada Hollywood: „nie dzwoń do nas, my zadzwonimy do Ciebie”. Zasada ta bardzo dobrze oddaje jego ideę. Zgodnie z nią biblioteka odwołuje się do funkcji zaimplementowanych przez programistę dopiero wtedy, gdy tylko zachodzi taka potrzeba, koordynując w ten sposób przepływ sterowania w aplikacji.

Jedną z odmian wzorca IoC jest wzorzec wstrzykiwania zależności DI (ang. *Dependency Injection*). Wstrzykiwanie zależności ma na celu usunięcie bezpośrednich zależności między poszczególnymi komponentami tworzącymi aplikację, na rzecz architektury opartej na wtyczkach. Wtyczką określamy element programowy, który spełniając określone wymagania może być wykorzystany w aplikacji w przeźroczysty sposób. W tym rozumieniu wtyczką będzie implementacja odpowiedniego interfejsu. We wzorcu tym tworzenie obiektów oraz ich konfigurację zleca się fabrykom. Są to metody lub klasy zawierające wiele metod, dzięki którym następuje konfiguracja określonych obiektów, z zapewnieniem spełnienia wszelkich koniecznych zależności.

4. Wyniki

Szablony aplikacyjne, porównywane w niniejszym artykule, są ciągle rozwijane, dlatego istotne jest wskazanie wersji, dla których wykonano porównania: JBoss Seam 2.0.2 GA, Spring 2.5.1, Jakarta Struts 2.0.11, Apache Tapestry 5.0.11, Java Server Faces 1.2.

W celu wykonania porównania przyjęto następujące kryteria porównawcze:

- ogólne własności szablonu,
- konfiguracja,
- cechy warstwy prezentacji, a wśród nich:
  - ogólne własności,
  - obsługa formularzy,
  - internacjonalizacja.

4.1. Ogólne własności szablonów

W tabeli 1 zostały zebrane ogólne właściwości każdego z omawianych szablonów.

Tabela 1

Porównanie ogólnych własności szablonów

	JSF	Struts	Spring	Tapestry	JBoss Seam
Rozmiar bibliotek w MB	4,82	5,21	3,45	2,39	3,44
Dostępność poza Java EE	–	–	+	–	–
Implementacja IoC	–	wewnętrzna	własna	własna	własna
Konfiguracja IoC poprzez adnotacje	–	zależna od implementacji	+	+	+
Front Controller zrealizowany jako...	serwlet	filtr	serwlet	filtr	serwlet i dodatkowa klasa
Wspierane języki skryptowe	–	dowolny język	JRuby, BeanShell, Groovy	dowolny język	Groovy
Wprowadzanie zmian bez restartu aplikacji	–	–	+	+	+
Dokumentacja	bardzo dobra	bardzo dobra	bardzo dobra	przeciętna	dobra

Wielkość podana w pierwszym wierszu tabeli 1 oznacza rozmiar minimalnego zestawu bibliotek, które są niezbędne do uruchomienia aplikacji z wykorzystaniem referencyjnej implementacji danego szablonu. Z każdym szablonem dostarczane są także dodatkowe, opcjonalne biblioteki, za pomocą których można rozszerzyć funkcjonalności aplikacji. Najwięcej opcjonalnych bibliotek dostarczają szkielety JBoss Seam oraz Spring. Rozmiary wszystkich bibliotek przekraczają w każdym z nich ponad 30 MB. Zaznaczyć też należy, że JSF to tylko specyfikacja, która może być implementowana przez różnych dostawców oprogramowania.

Wartość zamieszczona w tabeli jest rozmiarem kodu bibliotek dla implementacji Apache MyFaces.

W wierszu mówiącym o wspieranych językach skryptowych, określenie "dowolny język" oznacza wykorzystanie mechanizmu BSF (ang. *Bean Scripting Framework*), umożliwiającego wykorzystanie wielu języków skryptowych. Bez dodatkowych bibliotek wspiera on m.in. TCL, Python, JavaScript.

4.2. Konfiguracja szablonów

Tabela 2 przedstawia zestawienie kilku informacji dotyczących konfiguracji każdego ze szkieletów.

Tabela 2

Konfiguracja szablonów

	JSF	Struts	Spring	Tapestry	JBoss Seam
Minimalna liczba deskryptorów wdrożenia	1	2	2	0	3
Rozmiar deskryptorów w liniach	238	223	120	nie dotyczy	b.d.
Konfiguracja poprzez adnotacje	–	+/-	+/-	+	+

Niektóre szablony umożliwiają rozbicie konfiguracji na wiele deskryptorów w celu logicznego pogrupowania ustawień. W zestawieniu brane są pod uwagę wszystkie deskryptory konfiguracyjne. Dla każdej aplikacji należy dodać jeszcze dodatkowe deskryptory *web.xml* oraz *application.xml*, co wynika z potrzeb platformy Java EE. Porównanie liczby linii w uwzględnianych deskryptorach ma tylko charakter poglądowy, gdyż wiele zależy od sformatowania zawartości plików. Aby porównanie było bardziej miarodajne, każdy z deskryptorów został sformatowany za pomocą tego samego narzędzia.

4.3. Warstwa prezentacji

Tabela 3 zawiera ogólne informacje o właściwościach warstw prezentacji każdego z omawianych w pracy szkieletów.

Tabela 3

Ogólne własności warstwy prezentacji

	JSF	Struts	Spring	Tapestry	JBoss Seam
Technologie warstwy prezentacji	JSP	JSP Freemarker Velocity	JSP, Velocity, XSLT, PDF, Excel	własny format	JSP, PDF, RTF
Zorientowanie na...	komponenty	akcje	akcje	komponenty	komponenty
Sposób realizacji odwzorowywania żądania na obiekty obsługi	brak odwzorowań (ziarenka zarządzane dostępne na każdej stronie)	żądanie sprawdzane pod kątem występowania identyfikatorów pakietów i akcji	zależnie od ustalonego obiektu decydującego o wyborze obsługi	bezpośrednio na nazwę klasy i wzorca strony	jak w JSF
Sposób określania przejść między stronami	reguły nawigacyjne w odpowiednim deskryptorze	określenie wyników dla akcji w deskryptorze lub adnotacje dla klas akcji	w kontrolerze podanie łańcucha znaków w zwracanym wyniku	w klasie strony poprzez zwrócenie odpowiedniego obiektu	jak w JSF
Wsparcie dla technologii AJAX	poprzez zewnętrzne biblioteki	- specjalne znaczniki - wyniki generowane ze stron JSP - wtyczki	poprzez zewnętrzne biblioteki	- w każdym komponencie - odpowiednia adnotacja	poprzez zewnętrzne biblioteki
Dostępność elementów z JavaScipt	–	+	–	–	+/-
Sposób obsługi zdarzeń komponentów UI	dowolna bezparametrowa metoda komponentu zarządzanego	nie dotyczy, bo oparty na akcjach	nie dotyczy, bo oparty na akcjach	- dowolna metoda spełniająca konwencję nazewniczą - odpowiednia adnotacja	jak w JSF
Użycie złożonych komponentów	+	+	–	+	+

Wraz z implementacją szablonu JBoss Seam dostarczanych jest wiele dodatkowych bibliotek. Wśród nich jest biblioteka RichFaces, która dostarcza wielu funkcjonalnych komponentów.

Istotną częścią każdego szablonu projektowego aplikacji sieciowych jest obsługa formularzy. W tabeli 4 zestawiono podejścia do tego zagadnienia każdego ze szkieletów.

Tabela 4

Obsługa formularzy

	JSF	Struts	Spring	Tapestry	JBoss Seam
Miejsce określania zasad poprawności danych	kod strony	- deskryptor - odpowiednia metoda klasy akcji - adnotacje pól	- klasa implementująca odpowiedni interfejs	kod szablonu strony	jak w JSF oraz adnotacje w klasach
Podstawowe kryteria sprawdzania poprawności danych	- wymagane pole - długość łańcucha znaków - zakres liczb	- wymagane pole - zakres liczb - zakres dat - e-mail - url - długość łańcucha - wyrażenie regularne	- wymagane pole	- wymagane pole, - zakres liczb - min, maks. długość łańcucha - wyrażenie regularne	jak w JSF oraz: - zakres dat - e-mail - numer karty kredytowej - wyraż. regularne
Walidacja danych po stronie klienta	–	+	–	+	–
Ochrona przed kolejnym wysłaniem tego samego formularza	brak	dodanie obiektu przechwytującego oraz znacznika w formularzu	implementacja odpowiedniej metody	b. d.	b. d.
Wsparcie dla formularzy wielostronicowych	–	–	+	–	–

W przypadku szablonu JSF zasady poprawności określane są bezpośrednio w kodzie dokumentu JSP, a obiekty sprawdzające, opracowane przez użytkownika również, mogą być wykorzystywane na stronie po zarejestrowaniu w odpowiednim deskryptorze.

Warto zwrócić uwagę, że w szablonie Struts sprawdzanie poprawności danych można wykonać na trzy sposoby: w pliku deskryptora, który musi mieć z góry określoną nazwę, w odpowiedniej metodzie zdefiniowanej w klasie akcji lub stosując odpowiednie adnotacje.

W szablonie JBoss Seam mechanizm sprawdzania poprawności danych z wykorzystaniem adnotacji bazuje na mechanizmie z szablonu związanego z utrwalaniem danych: Hibernate Validation Framework.

W aplikacjach sieciowych bardzo istotną funkcją jest możliwość dostarczenia różnych wersji językowych, bez konieczności tworzenia osobnych stron w warstwie prezentacji. Tabela 5 zestawia podejście do internacjonalizacji zastosowane w każdym ze szkieletów.

Tabela 5

Internacjonalizacja

	JSF	Struts	Spring	Tapestry	JBoss Seam
Sposób definiowania	zdefiniowanie plików zasobów w deskryptorze	domyślnie aktywne	w obiekcie JavaBean	domyślnie aktywne	jak w JSF
Konieczność określenia dostępnych lokalizacji	+	–	–	+	+
Położenie plików zasobów	w ścieżce klas	w pakiecie języka Java z klasami akcji, lub w dowolnym miejscu	w ścieżce klas	w pakiecie języka Java z klasami stron, lub w katalogu WEB-INF	w ścieżce klas

Szablon Spring wyróżnia się sposobem definiowania ustawień międzynarodowych, który polega na zdefiniowaniu klasy zgodnej ze standardem JavaBean rozszerzającym odpowiednią klasę bazową. Następnie odpowiedzialny za ustawienia pakiet języka Java wraz z nazwami plików zasobów podajemy w parametrach jego konstruktora.

Poza zwykłą ścieżką klas pliki zasobów mogą być ulokowane w pakiecie klas akcji (Struts). Komunikaty zdefiniowane w ten sposób są określane dla poszczególnych akcji. Jeśli mają być dostępne jakiekolwiek komunikaty wspólne dla wielu akcji, to należy stworzyć dodatkowy plik własności i umieścić go w pakiecie z klasami akcji. Pliki zasobów mogą być również dostarczane oddzielnie dla każdej klasy strony (Tapestry) i wtedy muszą być ulokowane z nimi w tym samym katalogu. Szablon Struts pozwala w zasadzie umieszczać pliki zasobów w dowolnym miejscu, pod warunkiem podania ścieżek w odpowiednim pliku konfiguracyjnym.

Serwer aplikacji JBoss pozwala na umieszczanie wszelkich, wspólnych dla wszystkich aplikacji, zasobów w katalogu WEB-INF. Jednak wyłącznie w przypadku szablonu Tapestry udało się wykorzystać w ten sposób dostęp do zasobów z ustawieniami internacjonalizacji.

5. Uwagi końcowe

W celu wykonania porównań zostały praktycznie zaimplementowane aplikacje o funkcjonalności zawężonej do takiej, która pozwoliła na skupienie się na możliwościach samych szablonów. Aplikacje te zostały zrealizowane na podstawie najczęściej stosowanych szablonów. Wszystkie umożliwiają użytkownikowi na wykonywanie takich samych operacji i dostarczają mu prawie taki sam interfejs graficzny. Założono więc, że takie aplikacje pozwalają na wstępne poznanie każdego szablonu i porównanie go z pozostałymi. Należy mieć na uwadze, że w tabelach ujęto tylko wybrane cechy poszczególnych szablonów. Dokładna prezentacja wszystkich możliwości i mechanizmów udostępnianych przez zbadane szkielety znacznie przekraczałaby rozmiary niniejszego artykułu.

Zestawienie wybranych cech wszystkich szkieletów może pomóc czytelnikowi, na podstawie zamieszczonych tam kryteriów, dokonać wyboru, który szablon będzie najbardziej odpowiedni dla wymaganego przez niego zastosowania. Na podstawie przeprowadzonych badań można zasugerować dwa rozwiązania, zależne od dziedziny zastosowania.

Dla aplikacji, które przede wszystkim są zorientowane na interfejsie użytkownika tak, aby od strony programisty był on jak najprostszy w projektowaniu, można wybrać szablon Tapestry. Zastosowane tam podejście zorientowane na komponentach oraz wyrafinowana implementacja wzorca projektowego odwracania zależności pozwala na bardzo szybkie zbudowanie aplikacji, bez konieczności tworzenia szeregu deskryptorów wdrożenia. Niestety, wadą tego szablonu jest to, że dostarczana do niego dokumentacja jest niekompletna.

Gdyby głównym zadaniem tworzonej aplikacji sieciowej byłoby dostarczenie złożonej funkcjonalności, korzystającej z różnych mechanizmów biznesowych, należałoby wybrać szablon Spring. Szkielet ten dzięki swojej niezwykłej elastyczności pozwala wykorzystywać wiele mechanizmów, które w trakcie tworzenia aplikacji lub po jej wdrożeniu mogą ulec zmianie. Konieczność tworzenia deskryptorów wdrożenia wcale nie jest jego wadą. Pomimo możliwości stosowania adnotacji w celu konfiguracji różnych elementów aplikacji, to deskryptory wdrożenia nadal są jednak lepszym sposobem konfiguracji aplikacji. Umożliwiają umieszczenie konfiguracji różnych elementów tworzonego oprogramowania w scentralizowanym miejscu, w pliku tekstowym, czy to w formacie XML czy w formacie akceptowanym przez właściwości języka Java. Dzięki temu tak przygotowana konfiguracja jest nie tylko czytelna, ale także może być modyfikowana bez konieczności ponownej kompilacji aplikacji, czego wymaga mechanizm adnotacji.

Omawiane w artykule szablony programowe nie są jedynymi dostępnymi na rynku oprogramowania. Można korzystać z jeszcze wielu innych rozwiązań, jednak te zaprezentowane w pracy są najczęściej wykorzystywane. Porównanie omówionych szablonów z tymi mniej popularnymi na pewno pozwoliłoby uwypuklić ich zalety. Przyszłe prace w kierunku porów-

nania szablonów aplikacyjnych będą polegać na zaimplementowaniu aplikacji o większym stopniu komplikacji z wykorzystaniem bardziej złożonych mechanizmów. Jednym z bardziej interesujących zagadnień jest wykorzystanie transakcyjności aplikacyjnej, która pozwala na prowadzenie osobnej konwersacji użytkownika z aplikacją w różnych oknach tej samej przeglądarki internetowej.

BIBLIOGRAFIA

1. Mukhar K., Zelenak C., Weaver J. Crume J.: Begining Java EE from novice to professional. Apress, Berkeley 2006.
2. Hunter J., Crawford W.: Java Servlet, programowanie. Helion, Gliwice 2002.
3. Goodwill J.: Java Server Pages. Helion, Gliwice 2001.
4. Geary D., Horstmann C.: Java Server Faces. Helion, Gliwice 2008.
5. Roughley I.:Practical Apache Struts2 Web 2.0 Projects. Apress, Berkeley 2007.
6. Brown D., Davis C., Stanlick S.: Struts 2 in Action. Manning, Greenwich 2008.
7. Johnson R., Hoeller J., Arendsen A., Riseberg T., Samapalenu C.: Spring Framework. Profesjonalne tworzenie oprogramowania w Javie. Helion, Gliwice 2006.
8. Van de Velde T., Snyder B., Dupuis C., Li S., Horton A., Balani N.: Begining Spring Framework 2. Willey Publishing Inc., Indianapolis, USA, 2008
9. Kolesnikov A.: Tapestry 5 Building Web Applications. Packt Publishing, Birmingham 2007.
10. Nusairat J.: Begining JBoss Seam from novice to professional. Apress, Berkeley 2007.
11. Lenart A.: Porównanie szablonów projektowych do realizacji aplikacji sieciowych uruchamianych na serwerach internetowych. Praca dyplomowa magisterska, Instytut Informatyki, Politechnika Śląska, Gliwice 2008.
12. Alur D., Crupi J., Malks D.: J2EE Wzorce projektowe. Helion, Gliwice 2004.
13. Ganna E., Helm R., Johnson R., Vlissides J.: Wzorce projektowe. WNT, Warszawa 2005.

Recenzent: Dr inż Maciej Bargielski

Wpłynęło do Redakcji 14 lipca 2009 r.

Abstract

In software meaning, framework is a skeleton of an application that can be customized by an application developer. Software frameworks aid the software developer by containing source code that solves problems for a given domain and provides a simple API. Design patterns are most often used in them, i.e. Model-View-Controller, Front Controller, Inversion of Control, and Dependency Injection. They are shortly described in the article. Sometimes we have the possibility of selection from a few frameworks, but we don't know which to use, which is best for us.

The article concentrates on comparing five frameworks, most often applied by programmers. The most popular are Java Server Faces, Jakarta Struts and Spring. Less popular, but also interesting are Tapestry and JBoss Seam. The comparisons were done basing on the network application, which accessed database using an internet browser. The application was implemented five times, each time using the different framework. Next made implementations were compared with themselves. This article presents some results in several tables. Following features are compared: general properties (table 1), possible configurations (table 2), features of the presentation layer (table 3), forms processing (table 4) and methods of the internationalization (table 5).

Concluding, it is possible to make two choices, dependent on applying field. For applications, which are oriented on the user interface, we can choose the Tapestry framework. From the programmer point of view, it is the simplest in design. Used there component-oriented approach and special implementation of IoC design pattern, let for very fast building the application, without the need of making lot of descriptors.

However, if the goal is internet application supports special functionality using all sorts of business mechanisms, we should choose the Spring framework. This skeleton thanks of one's extraordinary elasticity, allows using many mechanisms, which in the process of creating the application or after implementing, still can be modified.

Adresy

Krzysztof DOBOSZ: Politechnika Śląska, Instytut Informatyki, ul. Akademicka 16, 44-101 Gliwice, Polska, krzysztof.dobosz@polsl.pl
Arkadiusz LENART: Politechnika Śląska, Instytut Informatyki, ul. Akademicka 16, 44-101 Gliwice, Polska, lenart.arkadiusz@gmail.com