Eclipse Web Tools Platform. Tworzenie aplikacji WWW w języku Java

Autor: Naci Dai, Lawrence Mandel, Arthur Ryman
Tłumaczenie: Przemysław Szeremiota
ISBN: 978-83-246-1391-5
Tytuł oryginału: COM+ Developer's Guide
Format: 168x237, stron: 744

Zwiększ swoją wydajność dzięki platformie WTP!

- Jak wykorzystać środowisko Eclipse do tworzenia aplikacji internetowych?
- W jaki sposób zorganizować projekt aplikacji?
- Jak przeprowadzić w aplikacjach testy jednostkowe?

Eclipse to zintegrowane środowisko programistyczne. Projekt został zapoczątkowany przez firmę IBM, natomiast aktualnie jest rozwijany przez Fundację Eclipse. Dzięki licznym dodatkom, pluginom i podprojektom zyskał on ogromną popularność i jest w tej chwili jednym z narzędzi najczęściej wybieranych do tworzenia aplikacji w języku Java – i nie tylko. Natomiast projekt WTP (skrót od ang. Web Tools Platform) dzięki swoim rozszerzeniom ułatwia tworzenie aplikacji WWW.

„Eclipse Web Tools Platform. Tworzenie aplikacji WWW w języku Java" jest długo oczekiwaną na polskim rynku książką, poświęconą tej właśnie platformie. Autorzy przedstawiają tu inspirującą historię tego rozwiązania, strukturę projektu oraz sposób konfiguracji Web Tools Platform. Kolejne rozdziały przybliżą Ci zagadnienia związane z warstwą prezentacji, logiki biznesowej i trwałości. Dowiesz się, w jaki sposób wykorzystać narzędzia dostarczane przez WTP do przyspieszenia prac w każdej z tych warstw. W zakresie podejmowanych zagadnień znajdują się również tematy związane z testami integracyjnymi i wydajnościowymi.

- Konfiguracja Web Tools Platform
- Architektura aplikacji WWW
- Podział projektu na warstwy
- Sposób wykorzystania narzędzia Maven
- Wykorzystanie usług WWW
- Testowanie aplikacji WWW
- Rozszerzanie WTP

Poznaj jedną z najlepszych platform do tworzenia aplikacji WWW i nie tylko!

Wydawnictwo Helion
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl

Spis treści

ROZDZIAŁ 5.

Architektura i projektowanie aplikacji WWW

Pomyłki są wrotami do odkryć.

— James Joyce

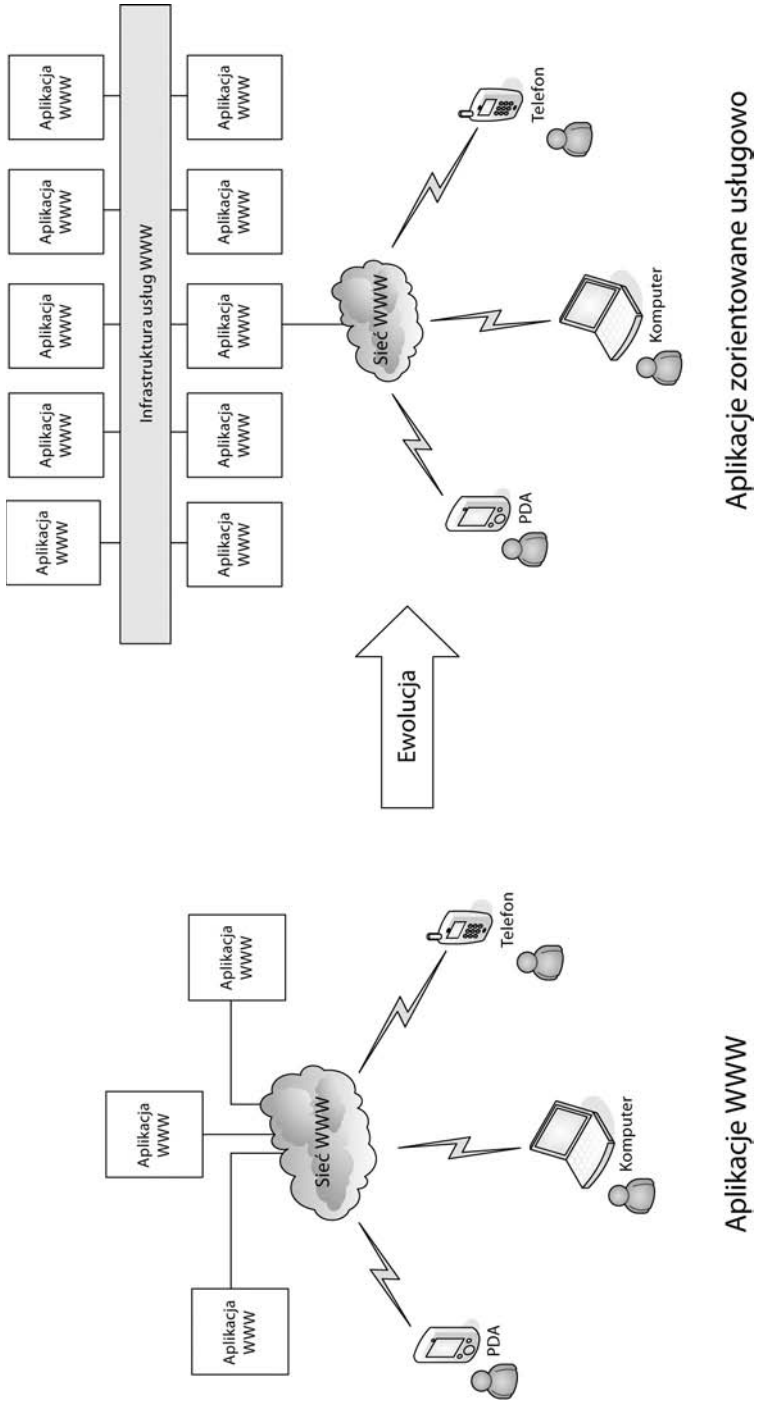
W tym rozdziale zajmiemy się opisem dwóch rodzajów systemów WWW — infrastruktury aplikacji i infrastruktury usługi. Wielu z nas tworzy aplikacje z interfejsem WWW. Owe interfejsy odwołują się do warstw biznesowych aplikacji i utrwalają pobrane dane w bazach danych. Dla takich rodzajów systemów architekturę warstwową udostępnia infrastruktura aplikacji. Z kolei w przypadku infrastruktury usługi mamy do czynienia ze współpracą poszczególnych usług za pośrednictwem WWW — bez udziału użytkowników. W tym przypadku mowa o architekturze zorientowanej na usługi — SOA (od *Service Oriented Architecture*).

Oba systemy mają cechy wspólne; w obu chodzi o zmontowanie rozległego i poprawnego pod względem struktury systemu WWW, opartego na prawidłach zasad obiektowości. Przypomnimy więc wiadomości z wykładów o projektowaniu obiektowym i zobaczymy, jak można je zastosować do WWW.

Krajobraz WWW

Sieć WWW ewoluuje od sieci stanowiącej nośnik informacji dla odbiorców-użytkowników, w kierunku środka komunikacji i współpracy pomiędzy ludźmi, ale i pomiędzy aplikacjami (zob. rysunek 5.1).

Sieć WWW działa w oparciu o standardowe i otwarte protokoły. Jest niejednorodna, rozproszona i szeroko dostępna. Sieć WWW jest więc niemal idealną platformą do wymiany informacji i koordynacji działań. Budowanie aplikacji WWW i integrowanie tych aplikacji nie



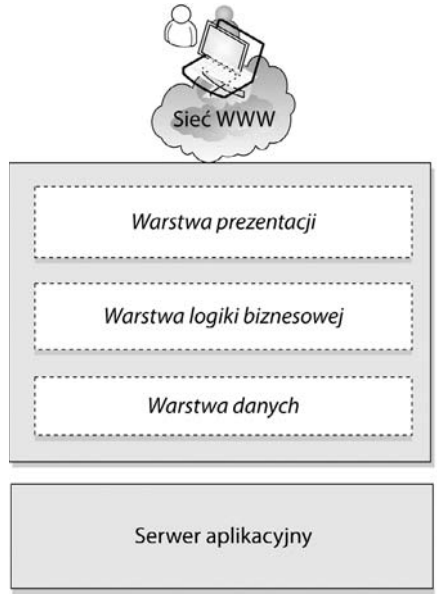
Rysunek 5.1. Aplikacje i usługi WWW

są już oddzielnymi zadaniami. Podstawowym założeniem SOA jest to, że systemy WWW powinny być montowane z usług eksponujących otoczeniu ściśle zdefiniowane interfejsy, które pozwalają na komunikację zarówno z użytkownikami, jak i aplikacjami.

W świecie ukierunkowanym na usługi mamy mnóstwo aplikacji sieciowych: aplikacje płatnicze uruchamiane na komputerach typu mainframe, drukarki fotograficzne drukujące zdjęcia z aparatu cyfrowego czy wsady świeżych wiadomości z praktycznie dowolnej dziedziny. Wszystkie te aplikacje są przykładami systemów świadczących pewne usługi. Każdy z takich „systemów-usługodawców" eksponuje swoje zasoby za pośrednictwem publicznego interfejsu definiującego jego usługę. Nowe aplikacje używają takich usług i montują z nich nowe aplikacje, które same mogą być udostępniane również jako osobne usługi. W tym prostym modelu usługodawcy i usługobiorcy można ująć tworzenie aplikacji WWW następnej generacji, z prawie nieograniczonymi możliwościami.

Aplikacje WWW

Prosta aplikacja WWW składa się z trzech logicznych warstw: warstwy prezentacji, warstwy logiki biznesowej i warstwy danych (zob. rysunek 5.2). To jedynie podstawowy podział ogólny, ale w faktycznej aplikacji można wyróżniać logicznie dodatkowe warstwy, reprezentujące i wyodrębniające poszczególne charakterystyczne elementy architektury aplikacji. Architektura fizyczna aplikacji jest tu nieistotna: wszystkie trzy warstwy mogą równie dobrze działać na pojedynczym serwerze aplikacyjnym i jednym komputerze albo na trzech i więcej osobnych serwerach aplikacyjnych. W J2EE architekturą fizyczną można zarządzać niezależnie od warstw logicznych aplikacji.



Rysunek 5.2. Aplikacje i usługi WWW

Najwyższa warstwa to warstwa prezentacji. Jest to warstwa interfejsu użytkownika, montowana zazwyczaj na bazie języka HTML. Modele RIA (*Rich Internet Application*) i AJAX wprowadzają zresztą do warstwy prezentacji nowsze technologie implementacji strony klienckiej, na przykład Flash czy JavaScript. Jeśli interfejs użytkownika nie wymaga do uruchomienia niczego poza przeglądarką WWW, określamy go mianem „cienkiego klienta" — ang. *thin client*.

Adobe Flash

Flash jest co prawda najpowszechniejszy właśnie w bogatym interfejsie użytkownika, ale często wykorzystuje się go również w aplikacjach wielowarstwowych. Flash ma własny obiektowy język programowania — ActionScript 2.0 — i komponenty umożliwiające odwoływanie się do usług WWW i baz danych. Więcej informacji o tej stronie platformy Flash można znaleźć pod adresem *http://www.adobe.com/platform*.

Warstwa środkowa to warstwa, w której realizuje się tak zwaną logikę biznesową aplikacji. W tej warstwie będą działać na przykład obiekty realizujące dodanie drużyny do ligi. Wydzielona warstwa logiki biznesowej nie jest zresztą związana wyłącznie z aplikacją WWW — porządne wyodrębnienie warstwy umożliwia wykorzystanie jej również w innych systemach.

Dolna warstwa to warstwa, gdzie realizowane jest zadanie przechowywania danych w sposób trwały. Najbardziej typowym źródłem trwałości jest baza danych, ale równie dobrze mogą to być pliki w systemie plików.

W dalszej części rozdziału zajmiemy się kwestiami dotyczącymi poszczególnych wyróżnionych tu warstw. W sieci WWW można znaleźć mnóstwo przykładów takich aplikacji; wszystkie udostępniają jakieś usługi warstwy biznesowej użytkownikom końcowym. Aplikacje te mogą być ze sobą skojarzone (na przykład odnośnikami hipertekstowymi), ale nie są faktycznie zintegrowane — stanowią raczej luźno powiązane „ekosystemy", w których czynnikiem wiążącym jest właśnie użytkownik końcowy. Tymczasem w systemie zorientowanym na usługi aplikacje są zintegrowane za pośrednictwem tychże usług; miejsce użytkowników w tych systemach zajmują inne, zewnętrzne aplikacje WWW, a warstwa prezentacji jest zastępowana warstwą usługi.

Aplikacje WWW w Javie

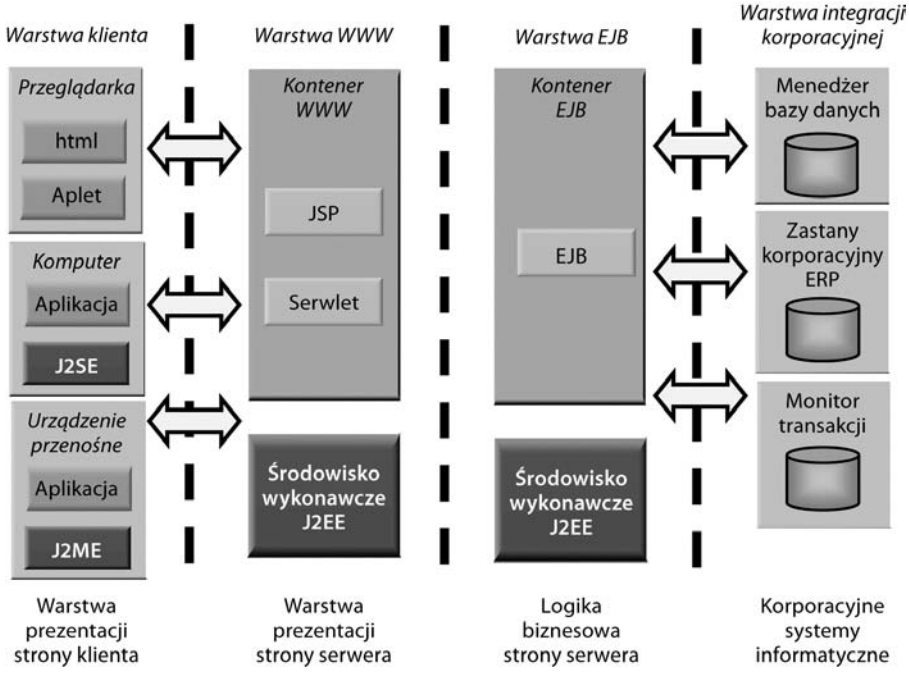
Aplikacje WWW w języku Java wykorzystują technologie opisane w specyfikacji J2EE i powszechne standardy ogólne, jak HTML, XMLO czy Web Service.

Koncepcja warstwowego podziału aplikacji i systemów klient-serwer jest w informatyce obecna znacznie dłużej niż technologie WWW i język Java. Chyba najcenniejszym wkładem ze strony J2EE w tej dziedzinie było udostępnienie praktycznej i ustandaryzowanej specyfikacji. Dzięki temu powstało wiele komercyjnych i niekomercyjnych serwerów aplikacyjnych obsługujących tak ustanowiony standard.

J2EE oferuje standardy dla modeli wykonawczych i programistycznych, wykorzystywanych w aplikacjach WWW. Są tu komponenty dla sesji klienckich, prezentacji, logiki biznesowej i logiki sterowania aplikacją, a także dla operacji biznesowych. Są tu też zdefiniowane usługi dotyczące rozproszenia, transakcyjności i zarządzania danymi, umożliwiające uruchamianie tych komponentów w środowiskach korporacyjnych.

Aplikacje WWW w technologii J2EE są przenośne pomiędzy zgodnymi ze specyfikacją serwerami aplikacyjnymi. Przenośność ta bazuje na dostępności kompletnego standardu regulującego przebieg interakcji klientów z systemem, implementację poszczególnych komponentów i sposób wykorzystania przez te komponenty interfejsów usług w celu integrowania ich z innymi systemami korporacyjnymi. Model J2EE dzieli warstwę prezentacji na warstwy prezentacji strony klienta i strony serwera. Do modelowania i implementowania logiki biznesowej służą tu komponenty EJB. Warstwa danych jest realizowana również za pomocą komponentów EJB, a także za pośrednictwem usług dających dostęp do korporacyjnych systemów informatycznych.

J2EE definiuje osobne kontenery dla komponentów WWW i komponentów biznesowych (EJB) — zob. rysunek 5.3. Kontenery udostępniają standardowe dla nich usługi, obsługując rozproszenie, sesyjność i transakcyjność. Klienty mogą być aplikacjami typu *thin client* (jak przeglądarka WWW) albo pełnoprawnymi i samodzielnymi aplikacjami. Wymagane do komunikacji protokoły WWW są obsługiwane przez środowiska wykonawcze po stronie serwera.



Rysunek 5.3. Kontenery J2EE

Kontener WWW obsługuje komponenty takie jak strony JSP i serwlety. Komponenty te są wykorzystywane powszechnie do realizacji warstwy prezentacji.

Kontener komponentów EJB udostępnia środowisko wykonawcze dla komponentów biznesowych i dostęp do korporacyjnych systemów informatycznych.

Środowisko wykonawcze składa się z licznych ustandaryzowanych usług, w tym usług łączności z bazami danych (JDBC), interfejsu transakcyjnego Java Transaction API i usług transakcyjności (JTA/JST), architektury J2CA (Java 2 Connector), usług uwierzytelniania

i autoryzacji (JAAS), usług interfejsu systemów nazw i katalogów (JNDI) i mnóstwa innych interfejsów realizujących komunikację wewnętrzną i zewnętrzną, zapewniających rozszerzalność systemów J2EE.

Projektowanie aplikacji WWW w języku Java

Czasami jesteśmy tak zaaferowani nowymi technologiami i systemami, że zapominamy nauki z przeszłości. Stosowanie zasad inżynierii oprogramowania, zwłaszcza zaś zasad obiektowości, jest dla nas w tych nowych technologiach wyzwaniem (Knight 2002). Owe nowe technologie bywają tak samo groźne, jak efektywne; mogą zachęcać do stosowania niekoniecznie najlepszych praktyk programistycznych. Na przykład programowanie stron JSP kusi możliwością bezpośredniego odwoływania się do baz danych, łatwo tu też zapomnieć o zdatności kodu do ponownego wykorzystania. Komponenty EJB jako składowe biznesowych komponentów aplikacji są krytykowane za nadmierną złożoność. XML promuje współdzielenie i wielokrotne wykorzystanie, ale bywa nadużywany.

Aplikacje WWW pisane w Javie składają się od strony warstwy prezentacji ze stron JSP i serwletów. Zarówno JSP, jak i serwlety można wykorzystać przy tworzeniu architektur z wyodrębnionymi warstwami. W ten sposób buduje się dwa najpopularniejsze typy aplikacji WWW w Javie, które będziemy oznaczać jako model 1. i model 2. Otóż wedle modelu 1. żądania klientów są przesyłane wprost do stron JSP, podczas gdy w modelu 2. są one kierowane do serwletu kontrolującego, który dopiero przekierowuje je do odpowiednich stron JSP. Obszerne opisy obu modeli można znaleźć w podrozdziale 4.4, *Web-Tier Application Framework Design*, w książce *Designing Enterprise Applications with the J2EE™ Platform* (Singh 2002). Model 1. jest wykorzystywany w bardzo prostych aplikacjach WWW. Z kolei model 2. stanowi adaptację wzorca projektowego MVC (*Model View Controller*) dla potrzeb aplikacji WWW. Wzorcem projektowym MVC zajmiemy się nieco później (w podrozdziale „Wzorzec MVC w aplikacji WWW").

Wedle modelu 1. najważniejszym celem jest przeniesienie możliwie dużej ilości kodu obsługującego prezentację z klas Java do stron JSP. Elementy JSP są popularne, ponieważ pozwalają na zarządzanie treścią (HTML, CSS, JavaScript czy XML) i na równoczesne stosowanie kodu w języku Java. W JSP można wygodnie przetwarzać żądania HTTP i generować odpowiedzi w postaci gotowych dokumentów HTML. Strony JSP dają się też łatwo zrozumieć projektantom i twórcom WWW (którzy niekoniecznie są wykwalifikowanymi programistami Java). W architekturze narzucanej przez model 1. cała aplikacja jest w zasadzie realizowana na bazie JSP. Niektórzy podnoszą, że mimo wszystko dochodzi do separacji treści i prezentacji, ponieważ wszystkie odwołania do danych są realizowane za pomocą komponentów języka Java. Architektura modelu 1. sprawdza się na szybko w niewielkich aplikacjach, ale dla wszystkich aplikacji obejmujących więcej niż kilka stron kodu trzeba ją uznać za zwyczajnie złą. JSP nie jest dobrym miejscem do realizowania logiki biznesowej i logiki sterowania aplikacją. Model 1. w miarę wzrostu rozmiaru aplikacji szybko degeneruje projekt, wymuszając zwielokrotnianie kodu i zwiększając złożoność. Wszystkie podstawowe warstwy logiczne typowej aplikacji są tu sprasowane do postaci pojedynczego komponentu.

Z punktu widzenia wyodrębniania różnych abstrakcji do różnych warstw i z punktu widzenia zasad obiektowości model 1. programowania z użyciem JSP jest najgorszy w tym, że w pojedynczym skrypcie grupuje się zadania należące do oddzielnych warstw. Wedle modelu 1. JSP musi:

1. Przyjmować dane wejściowe.
2. Obsługiwać logikę aplikacji (logikę biznesową i logikę przepływu sterowania).
3. Generować dane wyjściowe (obsługiwać logikę prezentacji).

Skoro wszystkie trzy warstwy zostały związane w pojedynczym komponencie, nie można żadnej z nich modyfikować ani testować z osobna. Do tego obsługa wszystkich tych zadań (opiszemy je osobno) również jest problematyczna. To samo dotyczyłoby wykorzystania samych serwletów (bo serwlet można uznać za skrypt z dodatkowymi osadzonymi elementami XML czy HTML); do tego mieszanie kodu z tekstem utrudnia zarządzanie kodem i diagnostykę kodu.

Przyjmowanie danych wejściowych

W ramach przyjmowania danych wejściowych skrypt otrzymuje do dyspozycji obiekt `HttpServletRequest`, będący reprezentacją strumienia wejściowego HTTP, minimalnie tylko przetworzoną. W HTTP wyróżniono trzy mechanizmy do przekazywania parametrów (metoda kodowania parametrów w URL, metoda parametrów zapytań i formularze) i we wszystkich dane są przekazywane jako zwyczajne ciągi znaków. Każdy ze skryptów składających się na aplikację musi więc „na własną rękę" określać sposób przekazywania parametrów, konwersję parametrów na wartości odpowiednich typów i weryfikację (walidację) tych wartości. Brak wspólnego, wyodrębnionego kodu obsługi danych wejściowych wymusza powielanie tego samego bądź podobnego kodu w wielu osobnych skryptach.

Obsługa logiki aplikacji

Kolejną kwestią problematyczną, dotyczącą zarówno obsługi wejścia, jak i logiki aplikacji, jest brak hermetyzacji informacji przy odwołaniach do danych żądania i danych sesji. Skrypt musi pobrać dane z żądania wejściowego po nazwie. Protokół HTTP jest protokołem bezstanowym, więc dane wykorzystywane na wielu osobnych stronach JSP w każdym skrypcie wymagającym tych danych muszą albo być zapisane w sesji skojarzonej z danym użytkownikiem, albo wielokrotnie odczytywane z zewnętrznego źródła danych.

Na przykład, jeśli skrypt przekazuje dane logowania użytkownika jako dane formularza, kod zapisujący te dane w sesji mógłby wyglądać tak:

Listing 5.1. Zapisywanie sesji z parametrami żądania HTTP

```
password = request.getParameter("passwordField");
decrypted = this.decode("password");
request.getSession().setAttribute("password", decrypted);
```

Zarówno zapis danych w atrybutach sesji, jak i zapis w zewnętrznym źródle danych to efektywny zapis danych w zasięgu globalnym, a aplikacja odwołuje się do takich danych jak do słownika, to znaczy poprzez ciągi z nazwami traktowanymi jako klucze wartości. Tak składowanych danych nie dotyczą zwyczajne mechanizmy kontrolowania czasu życia zmiennych i w każdym skrypcie czy też na każdej stronie korzystającej z danych trzeba „ujawnić"

stosowane nazewnictwo atrybutów-zmiennych. Nie można wtedy w prosty sposób wyszukać wszystkich odwołań do zmiennych za pośrednictwem zwyczajnych mechanizmów programistycznych, co z kolei utrudnia modyfikowanie odwołań do danych. A jeśli JSP nie hermetyzuje stosowanej konwencji nazewniczej, wiedza i niej i o niskopoziomowych przecież szczegółach implementacji protokołu HTTP musi być implementowana w całości aplikacji, co skutecznie niweczy jej zdatność do adaptacji do nowych zastosowań. Co więcej, mamy tu też potencjalne źródło błędów, które mogą powstać nie tylko wskutek zwyczajnej literówki w nazwie zmiennej, ale także w wyniku zastosowania takiej samej nazwy w różnych skryptach do różnych celów. W miarę zwiększania się liczby stron JSP budowanej w ten sposób aplikacji, problemy te mogą okazać się przytłaczające.

Kiedy do realizacji logiki aplikacji wykorzystuje się strony JSP, wprowadza się do nich potencjalnie znaczące ilości kodu. Tymczasem techniki zarządzania kodem w przypadku kodu osadzonego w JSP muszą z konieczności być ograniczone. Mieszanina kodu i tekstu na stronach JSP utrudnia też diagnostykę kodu. Co prawda w WTP znalazły się zarówno mechanizmy asysty przy wprowadzaniu kodu w JSP, jak i mechanizmy interaktywnego debugowania kodu, w przypadku prekompilowanych stron JSP trzeba będzie debugować złożony kod generowany przez serwer, co jest trudne. Z tych względów należałoby minimalizować ilość kodu w stronach JSP i unikać programowania tam logiki aplikacji.

Obsługa logiki biznesowej

W JSP cały kod ma strukturę monolityczną. Co prawda można w nim wydelegować zadania biznesowe do obiektów biznesowych, ale wciąż mamy do czynienia z mieszaniem logiki aplikacji i logiki biznesowej. Tymczasem nie da się skutecznie przetestować niezależnie od siebie poszczególnych fragmentów JSP, tak jak to jest możliwe w przypadku odrębnych klas języka Java (i testów jednostkowych). Do tego realizacja logiki biznesowej w JSP prowadzi do dalszego przeładowania strony kodem, co utrudnia zarządzanie tym kodem i konserwację projektu.

Generowanie danych wyjściowych

Przy generowaniu danych wyjściowych w prostym skrypcie miesza się treść HTML bądź XML z danymi dynamicznymi. Powoduje to wiązanie wyglądu strony wynikowej z pozostałymi warstwami aplikacji. Zmiana wyglądu strony WWW czy choćby przystosowanie aplikacji do potrzeb i ograniczeń różnych urządzeń docelowych są wtedy mocno utrudnione. A ta ostatnia trudność nabiera znaczenia, ponieważ sieć WWW nieustannie rozprzestrzenia się na urządzenia przenośne, w tym choćby telefony komórkowe. JSP pomaga w uporaniu się z tym problemem, pozwalając projektantom WWW tworzyć wygląd stron, podczas gdy programiści Java realizują logikę prezentacji w adnotacjach. Taki układ uważa się powszechnie za najbardziej odpowiednie zastosowanie stron JSP.

Projekt aplikacji WWW z podziałem na warstwy

W architekturze z wyodrębnionymi warstwami powstaje system składający się z kilku wyraźnie oddzielonych części (warstw), z możliwie ograniczonymi zależnościami i interakcjami pomiędzy tymi częściami. Taki system cechuje się dobrym podziałem problematyki aplikacji, co oznacza, że różne aspekty działania aplikacji można opracowywać niezależnie od siebie,

z minimalnym wpływem (a docelowo z brakiem takiego wpływu) na pozostałe części. Oddzielając od siebie poszczególne fragmenty systemu czynimy oprogramowanie wysoce adaptowalnym, łatwo dostosowującym się do zmieniających się w przyszłości wymagań. Warstwy obejmują logikę pobierania danych i ich wyprowadzania (warstwa prezentacji), logikę aplikacji, logikę biznesową i zagadnienia trwałości danych. Wszystkie te warstwy można wyprowadzić z opisywanego wcześniej trzywarstwowego modelu wzorcowego; warstwa wejścia to składowa warstwy prezentacji. Warstwa logiki aplikacji jest często podzielona na logikę przepływu sterowania w warstwie prezentacji i logikę biznesową oraz przepływy danych i procesów w warstwie logiki biznesowej. Logika utrwalania danych może stanowić osobną warstwę danych; elementy tej logiki mogą znajdować się w warstwie logiki biznesowej.

Warstwa danych wejściowych

Warstwa danych wejściowych albo też warstwa wejścia obejmuje kod zajmujący się przetwarzaniem i weryfikacją poprawności składniowej danych wejściowych, a więc strumieni SOAP, HTTP, SMPT i tak dalej; odpowiada też za wyłuskiwanie wartości parametrów z żądań. Wedle wzorca MVC odpowiada ona kontrolerowi wejścia.

Do zmontowania tej warstwy wykorzystujemy komponenty i interfejsy serwletów do obsługi protokołu HTTP. Sposobami użycia tych interfejsów i komponentów zajmiemy się w dalszej części rozdziału.

Logika aplikacji

Kod logiki aplikacji odpowiada za ogólny przepływ sterowania w aplikacji WWW. Często ta warstwa określana jest mianem warstwy sklejającej (ang. *glue layer*), oddzielającej warstwę logiki biznesowej od logiki danych wejściowych i wyjściowych i zarządzającej stykiem tych warstw. Wymaga to zaszycia tutaj pewnej wiedzy o obu tych warstwach. W tej warstwie będzie na przykład dochodzić do konwersji pomiędzy wejściem i wyjściem warstwy prezentacji w postaci ciągów znaków a komunikatami bądź wartościami obiektów biznesowych. W aplikacji WWW ta warstwa może również zarządzać interakcją użytkownika z wieloma stronami aplikacji jako sekwencją kroków (przejściami pomiędzy stronami WWW). W MVC odpowiada to roli kontrolera aplikacji.

Standard J2EE nie definiuje bezpośrednio komponentów do realizacji logiki aplikacji. Warstwa ta jest zazwyczaj implementowana w obrębie kontenera WWW J2EE i wykorzystuje podobne komponenty i interfejsy, jak te wykorzystywane w warstwie danych wejściowych. Sytuacja ta poprawia się wyraźnie wraz z dodaniem do Java EE 5 specyfikacji JSF (*JavaServer Faces*).

Logika biznesowa

Kod logiki biznesowej, implementujący tak zwane obiekty biznesowe, zajmuje się wyłącznie wewnętrznymi i właściwymi zadaniami aplikacji, czyli jej procesami biznesowymi. Kod ten powinien być kompletnie niezależny od warstw zewnętrznych (prezentacji). W złożonej aplikacji logika biznesowa będzie najpewniej najbardziej rozbudowanym komponentem, ściśle

związanym z kodem odwołującym się do systemów zewnętrznych, w tym baz danych, korporacyjnych systemów informacyjnych (EIS) takich jak ERP (*Enterprise Resource Planning*) czy CRM (*Client Relationship Management*) i innych powiązanych usług. We wzorcu MVC logice biznesowej odpowiada „model".

Obiekty realizujące logikę biznesową nie powinny być zależne od obiektów pozostałych warstw. W takim układzie można łatwo implementować rdzenne zadania biznesowe aplikacji i umożliwić realizację tych zadań nie tylko w obrębie aplikacji J2EE, ale również w innych systemach. Nasza rekomendacja dotycząca projektowania warstwy biznesowej jest prosta: warstwa ta powinna być możliwie uproszczona, implementowana w miarę możliwości za pomocą zwyczajnych obiektów Javy i całkowicie niezależna od pozostałych warstw architektury aplikacji. Odwoływanie się tu do komponentu takiego jak JSP albo któregoś z interfejsów J2EE typowego dla specyfiki WWW (np. interfejsu żądania HTTP) jest tu niepożądane. Tak samo niepożądane byłoby korzystanie tu bezpośrednio z interfejsów utrwalania danych wykorzystywanych w warstwie trwałości. Bo co stanie się z obiektami logiki biznesowej po późniejszej zmianie technologii utrwalania danych? Prawidłowe ograniczenie takich zależności eliminuje przyszłe problemy i umożliwia niezależne wprowadzanie zmian w innych warstwach.

Trwałość

Logika biznesowa implementowana w postaci obiektów języka Java potrzebuje jakiegoś narzędzia do trwałego składowania danych biznesowych. W większości aplikacji w tej roli występują relacyjne bazy danych. Można też wykorzystywać technologie alternatywne, jak bazy danych XML czy bazy obiektowe. Zadaniem warstwy trwałości jest udostępnienie tej funkcjonalności w aplikacji. Logika biznesowa nie powinna być zależna od warstwy trwałości, więc w modelu biznesowym nie należy odwoływać się wprost do interfejsów składowiska danych.

Utrwalanie obiektów realizuje się na różne sposoby, od obiektów DAO (*Data Access Object*), zależnych zazwyczaj od interfejsów dostępu do baz danych i języków zapytań takich jak SQL. Takie podejście jest odpowiednie w przypadku niewielkiego zestawu prostych obiektów, z zaletą zwiększonej elastyczności. W innych podejściach uwzględnia się interfejs trwałości Java Persistence API (JPA), wyrafinowane szkielety ORM (*Object-Relational Mapping*), jak Hibernate czy TOPLink, oraz podejścia angażujące obiektowe bazy danych. Utrwalanie obiektów było przedmiotem intensywnych prac i szczegółowe omawianie tego zagadnienia wykracza poza zakres tematyczny niniejszej książki.

Prezentacja wyników

Ta warstwa grupuje kod i zasoby niestanowiące kodu (pliki HTML, XML czy obrazki), lecz wykorzystywane do prezentowania wyników działania aplikacji użytkownikowi. Zazwyczaj składa się z niewielkiej ilości kodu, a tenże kod dotyczy wyłącznie formatowania i prezentowania danych. Na przykład strona JSP warstwy prezentacji wyników może zawierać fragmenty kodu w języku Java, wypisującego saldo konta na dynamicznie generowanej stronie WWW. We wzorcu MVC odpowiada to koncepcji widoku.

Standard J2EE udostępnia komponenty JSP i serwlety przewidziane do implementowania warstwy prezentacji. Komponenty te są obsługiwane poprzez bogaty zestaw interfejsów do przetwarzania HTML i XML, tworzenia obrazków, zarządzania adresami URL i ogólnie do obsługi wszystkich zadań związanych z budową interfejsu użytkownika poprzez WWW.

Wzorzec MVC w aplikacji WWW

Wzorzec MVC (*Model-Controller-View*) stanowi elegancki koncepcyjny model podziału zadań w serwerowej stronie aplikacji WWW. Aplikację implementuje się jako połączenie serwletów, stron JSP, usług i właściwego kodu w języku Java. Prezentowane i polecane tutaj podejście to jedno z możliwych podejść do podziału odpowiedzialności i wyeliminowania słabości tkwiących w wykorzystywanych technologiach. Koncepcja MVC wywodzi się z systemu Smalltalk-80 i promuje warstwowe podejście przy projektowaniu i tworzeniu graficznych interfejsów użytkownika. Oto podstawowe koncepcje uczestniczące w MVC:

- *model* obsługujący logikę aplikacji i logikę biznesową,
- *widok* obsługujący logikę prezentacji,
- *kontroler* przyjmujący i rozprowadzający wejście (z klawiatury i myszy).

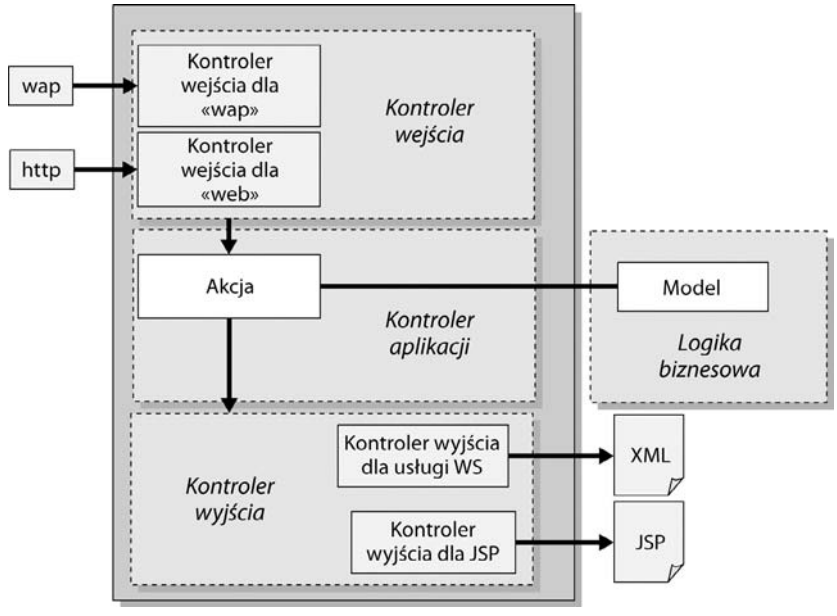
Wzorzec MVC powstał jako sposób oddzielenia kodu modelowego (czyli kodu niezwiązanego z interfejsem użytkownika) od kodu prezentacyjnego i sterującego. Kod modelowy nie powinien zawierać żadnej wiedzy o interfejsie, a jedynie rozgłaszać powiadomienia o wszelkich zmianach stanu do komponentów zależnych, którymi w klasycznym ujęciu są widoki.

Taki schemat zapewnia dobrą separację pomiędzy trzema wymienionymi warstwami, za to boryka się z dwoma wadami. Po pierwsze, postrzeganie modelu jest uproszczone i nie uwzględnia rozróżnienia pomiędzy logiką aplikacji (na przykład przepływem sterowania pomiędzy składowymi stronami WWW aplikacji) i logiką biznesową (czyli np. przetwarzaniem płatności). Po drugie, w większości systemów okienkowych i bibliotek funkcje kontrolera i widoku są łączone w pojedynczym elemencie interfejsu, co redukuje użyteczność koncepcyjnego podziału na kontroler i widok.

Pierwotne rozumienie architektury MVC ulegało więc ewolucji. Obecnie pojęcie *kontrolera* odnosi się do obiektu obsługującego logikę aplikacji, a pojęcie *modelu* zarezerwowano dla obiektów biznesowych. Będziemy więc tak określać obiekty biznesowe, a pojęcia *kontroler wejścia* i *kontroler aplikacji* odnosić do dwóch podstawowych typów kontrolerów aplikacji.

W popularnych ramach projektowych Javy, jak Struts, JSF czy Spring, koncepcje MVC są wcielane w życie jako połączenie kodu w języku Java, stron JSP, serwletów i zwyczajnych obiektów Javy realizujących zadania różnych komponentów. Owe ramy projektowe koncentrują się na oddzieleniu widoku od kontrolera, ale niekoniecznie sugerują sposoby oddzielenia kontrolerów, czyli logiki aplikacji, od logiki biznesowej. W kontekście WWW dwojakie stosowanie pojęcia *kontrolera* (jako kontrolera wejścia i kontrolera aplikacji) jest jak najbardziej poprawne. W przypadku aplikacji HTTP wejście i prezentacja są od siebie całkiem rozdzielone, pożądany jest więc kontroler wejścia oddzielony od widoku. A w przypadku aplikacji o jakiejkolwiek realnej złożoności trzeba jeszcze uwzględnić byt kontrolera aplikacji jako oddzielającego szczegóły przepływu sterowania w aplikacji od szczegółów implementacji logiki biznesowej.

W następnych podrozdziałach będziemy omawiać podstawową strukturę obiektów w ramach projektowych MVC dla WWW (zob. rysunek 5.4). Architektura ta jest implementowana przez liczne wymienione wcześniej szkielety aplikacyjne.



Rysunek 5.4. MVC dla aplikacji WWW

Kontroler wejścia

Centralnym elementem jest kontroler wejścia. W systemie działa jeden taki kontroler dla wszystkich stron WWW. Kontroler wejścia odbiera i analizuje dane wejściowe, wykrywa mechanizm przekazywania danych, wyłuskuje z żądania niezbędne informacje, we współpracy z kontrolerem aplikacji identyfikuje następną operację — określaną mianem *akcji* — i wywołuje ową akcję w odpowiednim kontekście (zmontowanym z zestawu danych wejściowych). Uruchomienie kontrolera wejścia jako pojedynczego komponentu pozwala na wyizolowanie w projekcie całości wiedzy związanej z obsługą protokołu HTTP i konwencji nazewniczych obowiązujących na poziomie żądań HTTP. W ten sposób eliminuje się powielanie kodu i zmniejsza łączny rozmiar kodu. Dzięki temu można też łatwiej modyfikować każdą z funkcji przetwarzających dane wejściowe, ponieważ modyfikacje są ograniczone do pojedynczego komponentu i nie wpływają na pozostałe komponenty aplikacji. Kontroler wejścia jest typowo realizowany za pośrednictwem serwletu; często wyróżnia się osobny serwlet, obsługujący żądania dostępu do aplikacji za pośrednictwem protokołu HTTP z poziomu zwyczajnej przeglądarki WWW, i drugi, obsługujący dostęp do aplikacji z poziomu urządzeń operujących protokołem i przeglądarką WAP.

Kontroler aplikacji

Kontroler aplikacji jest najczęściej zwyczajnym obiektem języka Java. Jego zadaniem jest koordynowanie działań związanych z przepływem sterowania w aplikacji, obsługa błędów, utrzymywanie informacji związanych ze stanem aplikacji (w tym referencji do obiektów biznesowych) i wybieranie odpowiedniego widoku do wyświetlenia. Kontroler aplikacji musi

„rozumieć" żądania i ich udział w przepływie sterowania w aplikacji oraz przekazywać te żądania w celu uzyskania odpowiedzi. Żądania WWW są kodowane w strumieniach tekstowego protokołu HTTP, a wartości parametrów mają w nich postać par klucz-wartość (oba elementy pary to ciągi znaków). Kontroler aplikacji musi dysponować mechanizmem odwzorowania takich par na obiekty aplikacji, które zarządzają przepływem danych. W większości szkieletów aplikacyjnych te odwzorowania są reprezentowane za pomocą rozbudowanych plików konfiguracyjnych w formacie XML, jak w przypadku pliku *struts-config.xml* wykorzystywanego w Struts. Na przykład kontroler aplikacji rozpoznaje ciągi URI takie jak poniższy:

```
/leagueplanet/addPlayer.do
```

Bazuje się tu na konwencji nazewniczej, co ma opisywane wcześniej wady, ale ponieważ jest to jedyny komponent wykorzystywany w taki sposób, skutki tych słabości są minimalizowane. W lepszym projekcie pojedynczy kontroler aplikacji jest zazwyczaj odpowiedzialny za wiele stron WWW i wiele operacji. W najprostszej aplikacji pojedynczy kontroler aplikacji może obsługiwać wszystkie strony. W aplikacji rozbudowanej wyznacza się zazwyczaj kilka kontrolerów aplikacji, każdy do innego obszaru jej działalności. Poprzez wykorzystywanie pojedynczego obiektu jako wspólnego, centralnego punktu odniesienia dla hermetyzacji informacji kontroler aplikacji rozwiązuje zagadnienia ukrywania informacji i konwencji nazewniczych. Zamiast przechowywać izolowane skrawki informacji w atrybutach sesji, składuje się je w obiektach biznesowych. Do śledzenia działania kontrolera aplikacji i obiektów biznesowych można wykorzystać klasyczne mechanizmy języka programowania, co bardzo ułatwia konserwację i modyfikowanie kodu. Do tego całość podlega statycznej kontroli typów, co jest niejako dodatkową metodą walidacji danych.

Kontrolery aplikacji konstruuje się rozmaicie. W szkielecie Struts reprezentuje się je jako *akcje*, podczas gdy w JSF określa się je mianem *managed backing beans*. W Struts przewiduje się obecność wielu akcji. Jeśli na przykład dana aplikacja ma dwa przypadki użycia — tworzenie drużyn i dodawanie graczy do drużyn — możemy zrealizować te przypadki w aplikacji za pomocą dwóch odpowiednich akcji. W aplikacji Struts klasy implementujące takie akcje mogłyby wyglądać tak, jak na listingu 5.2.

Listing 5.2. Przykładowy kod klasy akcji (Struts)

```
public class CreateTeamAction
{
  public void execute(...){}
}

public class AddPlayerAction
{
  public void execute(...){}
}
```

Akcje dotyczące drużyny i graczy są w oczywisty sposób powiązane. Choćby przez to, że graczy dodaje się do drużyn. Ale w Struts nie przewidziano mechanizmu do grupowania akcji. Nie można więc zgrupować wielu akcji będących częścią tego samego przepływu, jak na przykład przy procesie rejestracji konta online.

Te braki szkieletu aplikacyjnego Struts są wypełniane przez inne mechanizmy, bazujące na Struts, jak w projekcie Eclipse Pollinate, w którym kontroler aplikacji to obiekt Javy zwany *przepływem w obrębie strony* (ang. *page flow*). Jest to klasa ujmująca grupę akcji jako jej metody i definiująca strukturę opisującą przepływ pomiędzy poszczególnymi metodami. W Pollinate powyższe przypadki użycia zrealizowalibyśmy za pomocą pojedynczej klasy przepływu dla strony (klasy akcji i ich zachowanie z przykładowego listingu 5.2 zostałyby zaimplementowane jako metody klasy przepływu).

Możliwość grupowania akcji i kojarzenia ich z obiektem reprezentującym przepływ sterowania w obrębie strony ilustruje kod z listingu 5.3. Posiadanie wspólnego kontrolera aplikacji dla całej grupy akcji zwiększa elastyczność wyrażania logiki aplikacji. Do tego można zarządzać stanem przepływu w obrębie pojedynczego obiektu, a nie za pośrednictwem żądań HTTP i interfejsów obsługi sesji.

Listing 5.3. Przykładowy kod klasy przepływu page flow

```
public class LeacuePlanetPageFlow extends PageFlowController
{
  public Forward createTeam(){..}
  public Forward addPlayer(){..}
}
```

Dwie najpopularniejsze obecnie realizacje modelu MVC w programowaniu aplikacji WWW, czyli Struts i JSF, są w istocie bardzo podobnymi koncepcjami. Niektórzy uważają, że bliższy założeniom MVC jest JSF, a to ze względu na dostępność komponentów stanowych w warstwie widoku i obsługę modelu zdarzeniowego do obsługi interakcji (np. zdarzeń kliknięcia przycisku). JSF określa też rozbudowaną bibliotekę standardowych znaczników, które wydatnie zmniejszają ilość kodu Java w stronach JSP (zob. listing 5.4).

Listing 5.4. Znaczniki JSF w JSP

```
<h:panelGroup>
  <h:commandButton id="submitCreateTeam"
    action="#{JsfLeaguePlanetBean.createTeam}" value="Create Team" />
  <h:commandButton id="submitAddPlayer"
    action="#{JsfLeaguePlanetBean.addPlayer}" value="Add Player" />
</h:panelGroup>
```

Trzeba jednak zawsze pamiętać, że wszystkie te szkielety aplikacyjne bazują na bezstanowym protokole HTTP. W JSF występuje koncepcja kontrolera aplikacji w postaci komponentów *managed backing beans* (listing 5.5). Te komponenty mogą ujmować grupy powiązanych akcji, czego brakuje w Struts. Wreszcie koncepcja przepływu w obrębie strony nie jest implementowana ani w JSF, ani w Struts. Informacja o takim przepływie jest niejawnie zaszyta w kontrolerach i plikach konfiguracyjnych XML.

Listing 5.5. Komponent backing bean w JSF

```
public class JsfLeaguePlanetBean
{
  public String createTeam(...){}
  public String addPlayer(...){}
}
```

Kontroler wejścia dla każdego żądania wywoła jedną z wielu możliwych akcji. Jednym z jego zadań jest określenie właściwego kontekstu akcji przeznaczonej do wywołania. Wybór akcji zależny jest zarówno od danych wprowadzonych na wejście, jak i od bieżącego stanu aplikacji; decyzja należy więc do kontrolera aplikacji. Wynik tego procesu decyzyjnego jest reprezentowany poprzez obiekt `ApplicationController` (`ApplicationController` to implementacja wzorca projektowego Command, opisywanego w Gamma 1995).

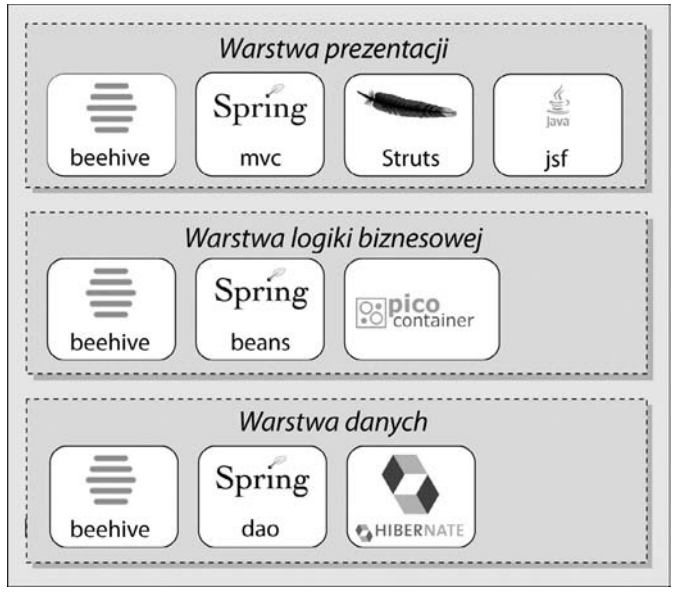
Obiekty biznesowe są zwyczajnymi obiektami Javy, zawierającymi wyłącznie logikę biznesową, bez jakichkolwiek informacji o warstwach sąsiadujących. Jedynym komponentem wyznaczonym do manipulowania obiektami biznesowymi jest zaś właśnie kontroler aplikacji (zob. rysunek 5.4). Dzięki temu znacznie łatwiej jest zarówno zaimplementować, jak i przetestować logikę biznesową, bo może się to odbywać w całkowitej izolacji od całej kłopotliwej infrastruktury WWW. Jeśli aplikacja jest prawidłowo zaprojektowana, obiekty biznesowe stanowią wyizolowaną warstwę, zdatną do wykorzystania zarówno w aplikacji WWW, jak i w aplikacjach zakładających dostęp z poziomu innych rodzajów klientów, a nawet w klasycznych aplikacjach stanowiskowych.

Widok

W aplikacji J2EE widoki są najczęściej stronami JSP, które odwołują się do kontrolera aplikacji i obiektów biznesowych. Widoki powinny zawierać możliwie mało kodu, a więc delegować większość swoich zadań do kontrolera aplikacji albo obiektów biznesowych. Na samej stronie powinien pozostać jedynie kod bezpośrednio związany z prezentacją bieżącej strony. Specyfikacja JSP udostępnia też biblioteki znaczników (ang. *taglibs*) do definiowania własnych znaczników, ujmujących bardziej rozbudowane zachowania widoku — w przypadku skomplikowanych widoków zaleca się przenoszenie kodu właśnie do samodzielnie definiowanych znaczników. Na rysunku 5.4 było widać dwa różne mechanizmy widoku. Kontroler wyjścia JSP wykorzystuje implementację JSP odpowiednią dla przeglądarki WWW albo WAP. Kontroler wyjścia WS reaguje na te same żądania, a w odpowiedzi generuje komunikaty XML zdatne do użytku w innych aplikacjach.

Szkielety aplikacyjne dla Javy

Programiści aplikacji WWW mogą przebierać pomiędzy rozmaitymi otwartymi i wolnymi (od „wolność") szkieletami aplikacyjnymi dla Javy — przyjrzymy się niektórym z nich (zob. rysunek 5.5). Szkielety te wydatnie upraszczają tworzenie aplikacji WWW w Javie. Udostępniają elementy znacznie polepszające możliwość testowania kodu, ułatwiają zarządzanie nim i samo programowanie. Izolują i implementują podstawową infrastrukturę aplikacji i dobrze integrują się z serwerami aplikacyjnymi.



Rysunek 5.5. Szkielety aplikacyjne dla Javy

OSGi

OSGi Alliance (dawniej Open Services Gateway Initiative) to ciało definiujące standard udostępniania platform usługowych na bazie Javy. Specyfikacja ta obejmuje szkielet aplikacyjny do modelowania cyklu życia aplikacji oraz rejestr usług.

Istniejące implementacje OSGi, jak Eclipse Equinox, Felix czy Knoplerfish, udostępniają kompletne i dynamiczne modele komponentowe, których brakowało w standardowych środowiskach wykonawczych dla Javy. Dzięki nim można instalować, uruchamiać, zatrzymywać, aktualizować i odinstalowywać aplikacje bądź komponenty nawet zdalnie.

OSGi zyskuje uznanie jako uogólniona platforma dla usług, ponieważ dobrze się skaluje, od urządzeń wbudowanych po systemy korporacyjne. Przykładem systemu bazującego na OSGi jest Eclipse (i co za tym idzie — WTP). Na bazie platform OSGi swoje serwery aplikacyjne następnej generacji konstruują takie firmy jak IBM i BEA. Co ciekawe, OSGi możemy też wykorzystać sami do budowania prostych komponentów biznesowych i usług, które w czasie wykonania są składane do postaci usługi biznesowej. W OSGi można na przykład uruchamiać aplikacje bazujące na Spring 2.0.

Apache Beehive

Projekt Beehive ma na celu udostępnienie szkieletu aplikacyjnego dla prostych komponentów sterowanych metadanymi, redukujących ilość kodu niezbędnego w aplikacjach J2EE. Rozwiązania z Beehive dotyczą wszystkich trzech warstw modelowej aplikacji WWW. Szkielet wykorzystuje adnotacje, zwłaszcza metadane według JSR 175 (JSR 175). Odwołuje się też

do innych projektów Apache, jak Struts i Axis. Oferuje NetUI do prezentacji, szkielet Controls do prostych komponentów i WSM (Web Service Metadata — implementacja JSR 181) oraz bazujący na adnotacjach model do konstruowania usług WS w Javie (JSR 181).

Apache Struts

Struts to szkielet aplikacyjny udostępniający warstwę kontrolną bazującą na standardowych technologiach WWW w Javie — wykorzystuje się tu JSP, serwlety oraz inne projekty Apache. Jest to swoista implementacja wzorca projektowego MVC.

JavaServer Faces

JSF to standard JSR 127 wywodzący się z JCP (*Java Community Process*) (JSR 127), definiujący zestaw znaczników JSP oraz klas Java upraszczających konstruowanie interfejsu użytkownika dla aplikacji WWW. Powszechnie uważa się, że JSF ma szansę zestandaryzować narzędzia i komponenty. JSF to szkielet dla warstwy prezentacji, z koncepcjami zapożyczonymi z MVC. JSF to oficjalny element specyfikacji J2EE 5.

Spring

Spring to szkielet aplikacyjny obejmujący komplet trzech warstw typowej aplikacji WWW w Javie. Szkielet dla wszystkich swoich komponentów implementuje wzorce projektowe Inversion of Control oraz Dependency Injection. Udostępnia też prosty kontener do implementowania logiki biznesowej za pomocą zwyczajnych obiektów Javy (POJO) i dzięki temu może wyeliminować potrzebę stosowania komponentów EJB. Do tego Spring udostępnia mechanizmy do zarządzania danymi aplikacji.

Pico Container

Pico Container jest prostym szkieletem aplikacyjnym, również bazującym na wzorcach Inversion of Control i Dependency Injection. Podobnie jak Spring, powstał jako odpowiedź na nadmierną złożoność tworzenia aplikacji J2EE, a szczególnie na trudności związane z programowaniem komponentów EJB.

Hibernate

Hibernate to szkielet relacyjnego odwzorowania obiektów — ORM (*Object Relational Mapping*). Pozwala programistom implementować relacyjne utrwalanie obiektów oraz odpytywać usługi o obiekty Javy bez konieczności modyfikowania ich kodu.

Specyfikacja komponentów encyjnych EJB3, a zwłaszcza specyfikacja interfejsu trwałości JPA (*Java Persistent API*), która z niej wyewoluowała, w obliczu złożoności dostępnych rozwiązań standardowych zwiększa jeszcze atrakcyjność Hibernate i podobnych szkieletów ORM.

Architektura usługowa SOA

SOA (*Service Oriented Architecture*) dotyczy wydzielania części właściwej działalności aplikacji do postaci istotnych jednostek, zwanych usługami, oraz montowania aplikacji poprzez integrowanie tychże usług. Ukierunkowanie na usługi oznacza ujmowanie poszczególnych elementów logiki biznesowej. Usługa to wcielenie fundamentalnej zasady programowania obiektowego, czyli oddzielenia implementacji od interfejsu. W połączeniu ze standardowymi językami opisu (jak XML), powszechnymi protokołami transportowymi (HTTP) oraz możliwością wyszukiwania i kojarzenia usługodawcy w czasie wykonania SOA okazuje się świetną technologią integrowania rozmaitych systemów. SOA i usługi WWW Web Service opierają się na licznych standardach, w tym XML, XML Schema, WSDL (*Web Service Description Language*), UDDI (*Universal Description, Discovery and Integration*), SOAP, JAX-RPC i wielu specyfikacjach WS-*. Szczegółowe omówienie SOA dalece wykracza poza tematykę niniejszej książki, zajmiemy się jednak tym, jak można w WTP montować aplikacje WWW ukierunkowane usługowo, głównie na bazie technologii Web Service.

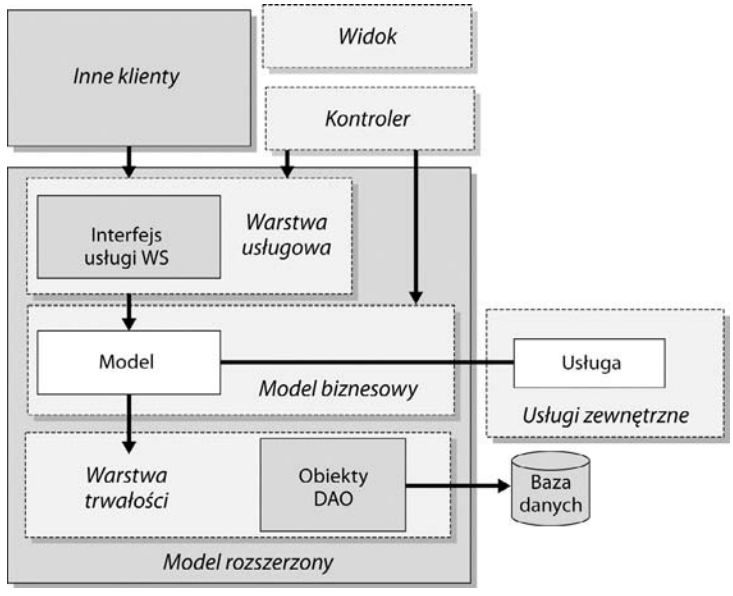
Udostępnianie usług. Warstwa usługowa

Zadaniem warstwy usługowej w aplikacji jest eksponowanie poszczególnych funkcji aplikacji i funkcji biznesowych jako wyodrębnionych usług. O przydatności czy ciekawości aplikacji świadczą przecież wyłącznie wykorzystujący ją klienci (jeśli w lesie przewróci się drzewo, a nikogo nie ma w pobliżu, to czy słychać trzask?).

Z usług, w przeciwieństwie do aplikacji, można korzystać rozmaicie:

- za pośrednictwem rozbudowanych aplikacji klienckich RCA (*Rich Client Application*), które konsumują usługi wielu usługodawców;
- w systemach osadzonych (telefony komórkowe);
- w portalach i aplikacjach WWW, od zdalnych portali i aplikacji WWW po warstwy sterujące systemu pionowego, w których warstwa usługowa wykorzystywana jest do odwołań do modelu biznesowego, w rozbudowanej implementacji MVC z warstwą usługową w postaci opisanej dalej;
- w projektach integracyjnych, za pośrednictwem języka BPEL (*Business Process Execution language*) automatyzującego procesy biznesowe;
- w systemach udostępniających usługi będące wynikiem zmontowania usług dostępnych skądinąd w nowym modelu biznesowym.

Uzupełnienie omówionej poprzednio architektury o warstwę usługową (jak na rysunku 5.6) pozwala na tworzenie aplikacji, która nie tylko jest użyteczna dla użytkowników końcowych, ale też może współdziałać z innymi systemami informatycznymi. Dodatkowa warstwa składa się z elementów takich jak usługi Web Service, rejestry usług (UDDI), kontrakty (WSDL) i elementy pośredniczące (proxy), które wiążą te usługi z naszymi aplikacjami. Warstwa usługowa eksponuje ściśle zdefiniowane interfejsy do identycznego jak poprzednio modelu biznesowego. Interfejs usługi jest określany przez kontrakt opisywany w języku



Rysunek 5.6. Model rozszerzony o warstwę usługową

WSDL. W danym modelu biznesowym można korzystać z procesów i logiki bazującej na usługach systemów zewnętrznych. Konsumentom usług nie są ujawniane szczegóły wewnętrznego modelu biznesowego. Wszystkie szczegóły techniczne niezbędne do konsumowania usług są opisywane przez kontrakt wyrażony w WSDL.

Gdzie pasuje SOA?

Warto byłoby się zastanowić, jak ma się SOA do tradycyjnego podziału warstwowego aplikacji. Nasuwają się od razu takie pytania: Czy SOA to część warstwy prezentacji? Czy SOA zastępuje warstwę prezentacji warstwą usługową? A co z logiką biznesową, gdzie ona teraz przynależy?

Otóż usługa nie ma widoku; skoro tak, nie ma potrzeby wydzielania warstwy prezentacji. Zastępuje ją zazwyczaj warstwa usługowa. Można więc powiedzieć, że warstwa usługowa odgrywa rolę warstwy prezentacji; usługa jest prezentacją, widokiem perspektywy biznesowej w formie dostępnej do zautomatyzowanego użycia przez inne aplikacje.

Trudniejsze jest pytanie o to, czy usługi wchodzą w skład modelu aplikacji i jej logiki biznesowej. Najprościej byłoby odpowiedzieć „nie", ale to nie wyczerpuje zagadnienia. Logika biznesowa nie jest zawsze i wyłącznie modelowana w formie, którą da się natychmiast i bezpośrednio udostępnić jako usługi. Dobry model obiektowy cechuje się wysoką ziarnistością i składa z wielu niewielkich, ale prostych do ogarnięcia obiektów z łatwymi do opanowania i wykorzystywania metodami. Obiekty te ujmują istotne koncepcje biznesowe i dane, ale nie są specjalnie przydatne do tworzenia usług. Usługi są zazwyczaj projektowane na bazie przypadków użycia; reprezentują zachowanie biznesowe jako ciąg zdarzeń, jak np. „płatność za rezerwację". Drobnoziarniste obiekty modelu pojedynczo nie ujmują takich transferów. Usługi stanowią więc mieszankę logiki biznesowej i logiki aplikacji. W naszej przykładowej aplikacji League Planet warstwa usługowa dysponowałaby obiektem obsługującym tworzenie nowej ligi. Warstwę logiki biznesowej omówimy w rozdziale 8.

Konsumowanie usług — zestawianie

Aplikacje konsumują usługi przez zestawianie i łączenie treści bądź usług udostępnianych przez usługodawców. Można więc zrealizować nowy proces biznesowy na bazie integracji usług świadczonych przez różnych usługodawców i uzupełnić całość o nową logikę biznesową, nowe reguły i możliwości. Udostępnione treści podlegają rekompozycji i odwzorowaniu do modelu biznesowego aplikacji klienckiej i są w niej prezentowane w nowy i unikatowy sposób, niekoniecznie dostępny od któregokolwiek z usługodawców usług składowych z osobna. To podstawowe założenie SOA.

Modelowanie procesu biznesowego i montowanie usług może konsumować usługi w sposób standaryzowany, za pośrednictwem mechanizmów usług WWW. Warstwa usługowa realizuje abstrakcję wielu różnych systemów biznesowych, a usługi tej warstwy można wykorzystywać do montowania nowych procesów biznesowych. W WTP nie mamy do dyspozycji narzędzi do zestawiania usług w taki sposób, ale producenci tacy jak IBM, Oracle czy BEA rozszerzyli WTP o możliwość projektowania i wykonywania takich procesów „składanych". Służy tam do tego język BPEL (*Business Process Execution Language*), standard OASIS będący odpowiednikiem języka XML dla dziedziny zestawiania i kompozycji usług w procesy.

Studium przypadku — League Planet

W tym podrozdziale spróbujemy opracować architekturę dla naszej fikcyjnej strony WWW League Planet (zob. podrozdział „Przedstawiamy League Planet" w rozdziale 1.). Z punktu widzenia architektury League Planet jest systemem wieloaspektowym, z różnymi profilami dla różnych użytkowników.

Po pierwsze, mamy wśród nich ludzi, którzy umieszczają treści w systemie. Są to osoby zainteresowane sportem i wykorzystujące League Planet do prowadzenia własnych amatorskich lig sportowych. Odwiedzają stronę WWW i tworzą nowe ligi, w ramach których mogą rejestrować drużyny, graczy, terminarze rozgrywek, miejsca rozgrywek, wyniki, statystyki i wiele rozmaitych innych informacji. League Planet ma dla nich udostępniać wysoce dynamiczny interfejs WWW, który pozwoli im wchodzić w bezpośrednie interakcje z systemem.

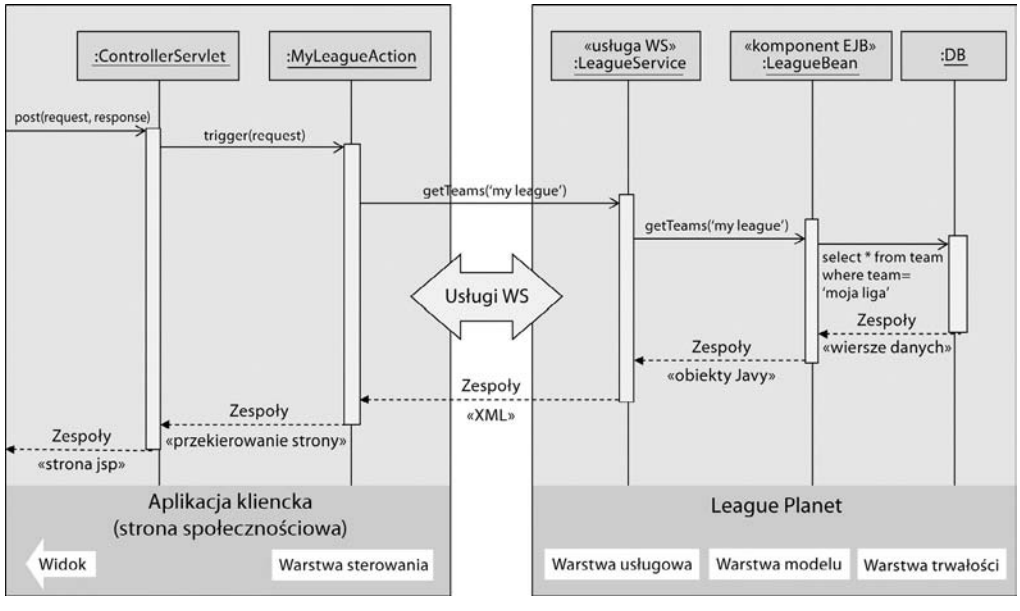
Korzystanie z systemu polega na przechodzeniu pomiędzy stronami WWW i wykonywaniu czynności takich, jak przeglądanie prezentowanych informacji, wypełnianie formularzy i zamawianie dóbr oferowanych przez partnerów biznesowych League Planet. Do obsługi takich użytkowników League Planet musi mieć warstwę prezentacji. Warstwa prezentacji będzie zrealizowana na bazie stron JSP. Aby ograniczyć ilość kodu opisującego interfejs, wykorzystamy bibliotekę znaczników dla Javy. Warstwa prezentacji będzie ograniczona do kodu wyświetlającego informacje i przyjmującego dane od użytkownika. Przepływ sterowania w aplikacji będzie należeć do zadań warstwy sterującej.

Warstwa kontroli jest odpowiedzialna za takie zadania, jak weryfikacja poprawności wprowadzanych danych, przepływ sterowania na stronie i pomiędzy nimi oraz współpraca z warstwą modelu biznesowego w celu realizacji zadań biznesowych i udostępniania treści dla warstwy widoku (prezentacji).

Kolejny profil dla League Planet to aplikacje zewnętrzne, np. aplikacje sponsorów, którzy potrzebują dostępu do pewnych usług. League Planet generuje sporą część swojego przychodu z reklam ogłoszeniodawców. Informacje o drużynach, graczach i odwiedzających oraz ich profilach mogą się przydać do przygotowywania skuteczniejszych reklam. Profile te służą do generowania banerów reklamowych i odnośników do stron odpowiednich partnerów biznesowych. Informacje te są udostępniane systemom partnerów League Planet za pośrednictwem zbioru usług, realizowanych w warstwie usługowej; samo wyświetlanie reklam również realizowane jest na bazie usług, tym razem zewnętrznych (udostępnianych w systemach partnerów biznesowych).

Wreszcie serwis League Planet wspiera też swoje organizacje partnerskie, oferując większość swoich treści i usług online dla zewnętrznych serwisów. Niektóre z nich to darmowe i abonamentowe informacje o ligach, graczach, drużynach, profilach odwiedzających, ogłoszeniach, wiadomości ze światka oraz doniesienia o najnowszych wynikach spotkań. Jako dostawca usług WWW League Planet jest źródłem unikatowych treści i usług, przeznaczonych do konsumpcji w aplikacjach zewnętrznych.

Aby zrealizować taki system, wykorzystamy architekturę opisaną w poprzednim podrozdziale (zob. rysunek 5.6). Aby zademonstrować planowany sposób działania systemu, weźmy scenariusz rozpisany na diagramie z rysunku 5.7.



Rysunek 5.7. Warstwa usługowa

Aplikacja kliencka, podobna do naszej własnej aplikacji WWW, konsumuje usługi udostępniane przez League Planet. Konsument usługi będzie wykorzystywał jedną (bądź wiele) z udostępnianych usług Web Service za pośrednictwem protokołu SOAP. Oba systemy mają kompletnie różne modele biznesowe i odmienną logikę aplikacji, ale będą zdolne do współdziałania na bazie architektury SOAP. Mianowicie, klient wysyła żądanie w celu pozyskania informacji o zespołach grających w danej lidze. Żądanie jest wysyłane za pośrednictwem

protokołu SOAP. Warstwa usługowa w League Planet odbiera żądanie; środowisko wykonawcze usług WWW rozpracowuje żądanie i odwzorowuje dane wejściowe opisane w przysłanym XML na odpowiadające im obiekty klas Javy. Kod Javy w `LeagueService` odbiera takie żądanie tak, jakby było zwyczajnym wywołaniem (komunikatem) z obiektu Javy. Warstwa usługowa wysyła do warstwy modelowej komunikaty Javy mające na celu pozyskanie z bazy danych informacji o zespołach w danej lidze; zespoły te są odwzorowywane na obiekty Javy i w takiej postaci przekazywane do warstwy usługowej. Tam wreszcie dochodzi do serializacji obiektów wynikowych, to jest konwersji obiektów Javy na ich reprezentację tekstową w języku XML, określoną poprzez typy danych odpowiedzi opisane WSDL. Odpowiedź jest potem zwracana do aplikacji klienckiej w formacie XML. Konsument usługi odbierający taką odpowiedź wykorzystuje podobne technologie do odwzorowania odpowiedzi na jego własne wewnętrzne obiekty modelu biznesowego i na przykład prezentuje pozyskane dane o zespołach na swojej stronie WWW.

Podsumowanie

Koncepcje omawiane w tym rozdziale powinny okazać się pomocne przy projektowaniu aplikacji WWW, które od początku mają mieć poprawną architekturę. Zaprezentowaliśmy szereg różnych podejść i omówiliśmy różne gotowe szkielety aplikacyjne wcielające te podejścia. Na bazie prezentowanych tu wzorców omówiliśmy zyski jakościowe, bo wzorce te narzucają i promują uznane zasady projektowania obiektowego. Omówione szkielety aplikacyjne są pomocą szczególnie istotną dla projektantów niedoświadczonych w metodologii obiektowej, bo dają im punkt wyjścia do projektu wysokiej jakości, bez konieczności zagłębiania się w szczegóły implementacji systemu, na którym ta jakość w dużej mierze bazuje. Szkielety wzorowane na MVC, omawiane w tym rozdziale, mogłyby zostać rozszerzone w kilku dziedzinach, uwzględniając na przykład różne charakterystyczne dla WWW wymagania, jak bezpieczeństwo, walidacja danych itp. Zalecamy własne eksperymenty z wymienionymi technologiami, bo to najlepiej pozwoli ogarnąć ich strukturę i poznać zastosowane kompromisy architekturalne; w ten sposób najlepiej można się też zorientować w zdatności danego rozwiązania w danej dziedzinie zastosowań.

Po wykładzie z projektowania możemy kontynuować naukę wykorzystywania WTP przy tworzeniu własnych aplikacji WWW. W rozdziale 6. zajmiemy się konkretnie różnymi stylami tworzenia aplikacji WWW w Javie i różnymi organizacjami projektu.