Enterprise JavaBeans

Autorzy: Ed Roman, Scott W. Ambler, Tyler Jewell
Tłumaczenie: Paweł Gonera, Mikołaj Szczepaniak
ISBN: 83-7361-073-1
Tytuł oryginału: Mastering Enterprise JavaBeans, 2nd Edition
Format: B5, stron: 609

Technologia EJB stanowi podstawę platformy Java 2 Enterprise Edition. Umożliwia ona tworzenie komponentów programistycznych działających po stronie serwera, które mogą być instalowane i uruchamiane na serwerach aplikacyjnych różnych producentów, zgodnych ze standardem EJB. Technologia EJB umożliwia zakup gotowych komponentów od konkretnego sprzedawcy, połączenie z komponentami od innego sprzedawcy i uruchomienie ich na serwerze aplikacji dostarczonym przez jeszcze kogoś innego. EJB doczekało się już drugiej wersji specyfikacji. EJB 2.0 wprowadza wiele zmian, w tym nowy rodzaj komponentów: sterowane komunikatami.

Książka „Enterprise JavaBeans" opisuje EJB 2.0 prezentując zarówno podstawowe zagadnienia związane z komponentami EJB, jak i tematy najbardziej zaawansowane. Pokazuje w ten sposób zalety i wady budowania aplikacji opartych na tej technologii. Dzięki tej książce nauczysz się tak pasjonujących zagadnień, jak strategie projektowe EJB, relacje EJB oraz rozmaite spojrzenia na kwestię trwałości danych przetwarzanych przez komponenty. Do zrozumienia tej książki wymagana jest wyłącznie znajomość Javy.

Książka przedstawia:

- Podstawy tworzenia komponentów EJB
- Komponenty sesyjne, encyjne i sterowane komunikatami
- Zarządzanie trwałością, komponenty encyjne typu CMP
- Java Message Service (JMS)
- Relacje między komponentami encyjnymi
- Najlepsze wzorce tworzenia stałych komponentów
- Zarządzanie projektami wykorzystującymi EJB
- Różne serwery EJB i kryteria wyboru serwera
- Budowę aplikacji z wykorzystaniem EJB, serwletów i JSP

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

Rozdział 1.

Wstęp do EJB

Enterprise JavaBeans (EJB) jest komponentową architekturą aplikacji działających po stronie serwera, która znacznie upraszcza proces budowy, przeznaczonych dla przedsiębiorstw, rozproszonych aplikacji komponentowych w Javie. Za pomocą komponentów EJB możesz pisać skalowalne, niezawodne i bezpieczne aplikacje bez potrzeby tworzenia własnych skomplikowanych szkieletów systemów rozproszonych. Technologia EJB daje możliwość błyskawicznego opracowywania aplikacji działających po stronie serwera; możesz szybko i łatwo konstruować w Javie komponenty dla serwera w oparciu o napisaną wcześniej infrastrukturę rozproszoną dostarczoną z zewnątrz. EJB zostały zaprojektowane, by wspierać przenośność i możliwość ponownego wykorzystania aplikacji dla sprzedawców oprogramowania pośredniczącego dla przedsiębiorstw.

Jeśli pierwszy raz masz do czynienia z techniką komputerową przeznaczoną dla rozwiązań biznesowych, poniższe pojęcia powinny Ci pomóc zrozumieć istotę problemu. Technologia EJB jest tematem skomplikowanym i wymaga obszernych objaśnień. W tym rozdziale zaprezentujemy EJB, udzielając odpowiedzi na poniższe pytania:

- Jakich struktur potrzebujesz do zbudowania dobrego rozproszonego i obiektowego rozwiązania?
- Czym są komponenty EJB i co daje ich użycie?
- Jakie są najważniejsze elementy swoistego „ekosystemu EJB”?

Wyjaśnijmy sobie te zagadnienia, wykorzystując metodę burzy mózgów.

Przyczyny powstania EJB

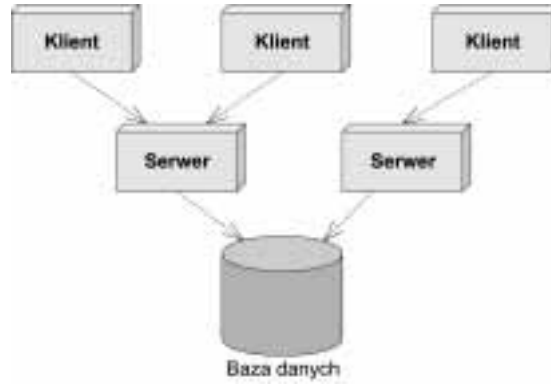
Rysunek 1.1 przedstawia schemat typowej aplikacji biznesowej. Taka aplikacja mogłaby działać w dowolnym przedsiębiorstwie i rozwiązywać dowolne problemy biznesowe. Oto kilka przykładów:

- system handlu papierami wartościowymi,
- aplikacja bankowa,

- centrum obsługi klienta,
- system obsługi zaopatrzenia,
- aplikacja analizy ryzyka ubezpieczeniowego.

Zwróć uwagę na fakt, że przedstawiona aplikacja jest **systemem rozproszonym**. Rozbiliśmy to, co normalnie byłoby dużym, jednolitym tworem, i oddzieliliśmy od siebie poszczególne obszary aplikacji tak, by powstałe warstwy były od siebie niezależne.

Spójrz na rysunek i zadaj sobie następujące pytanie: „Jeśli jednolitą aplikację rozbijemy na system rozproszony z wieloma klientami łączącymi się z wieloma serwerami i bazami danych za pośrednictwem sieci, co powinno być naszym głównym zmartwieniem w odniesieniu do nowego systemu (patrz rysunek 1.1)?".



Rysunek 1.1. *Standardowa architektura klient-serwer*

Zastanów się przez chwilę nad możliwie dużą liczbą zagadnień. Następnie przewróć stronę i porównaj swoją listę z naszą. Tylko nie oszukuj!

W przeszłości większość przedsiębiorstw budowało swoje własne oprogramowanie. Przykładowo, firma oferująca usługi finansowe mogła budować oprogramowanie pośredniczące, które wspomagało system handlu akcjami.

Obecnie przedsiębiorstwa, które same konstruują usługi pośredniczące, ponoszą spore ryzyko. Wykorzystywane obecnie oprogramowanie jest bardzo skomplikowane w budowie i konserwacji, wymaga wiedzy eksperckiej i jest całkowicie niezrozumiałe dla pracowników większości firm. Po co więc tworzyć coś, co można kupić?

Pomysł serwera aplikacji pojawił się, by umożliwić Ci kupowanie oprogramowania realizującego usługi pośredniczące, zamiast jego samodzielnego budowania. Serwery aplikacji udostępniają najbardziej popularne usługi, jak zarządzanie zasobami, obsługa sieci i wiele innych. Serwery takie pozwalają Ci skupić się na samej aplikacji bez potrzeby martwienia się o całe oprogramowanie pośredniczące potrzebne do stworzenia dobrego rozwiązania dla serwera. Piszesz więc kod dostosowany do potrzeb Twojego przedsiębiorstwa i umieszczasz go w odpowiednim środowisku wykonawczym serwera aplikacji. I w ten sposób, zgodnie z zasadą *dziel i zwyciężaj*, rozwiązujesz swój problem biznesowy.

Ważne problemy związane z budową dużych systemów biznesowych

Powinieneś mieć już przygotowaną skromną listę problemów, które musiałbyś mieć na uwadze podczas budowy dużego systemu biznesowego. Oto krótka lista poważnych problemów, z którymi musimy się zmierzyć. Nie przejmuj się, jeśli nie wszystkie na razie rozumiesz, wkrótce je poznasz:

- Zdalne wywoływanie metod. Potrzebujemy reguł opisujących proces łączenia klienta z serwerem przez sieć komputerową. Problem dotyczy wysyłania żądań wywołań metod, przesyłania parametrów i wiele innych.
- Zrównoważenie obciążeń. Klienty muszą być kierowane do serwera o najmniejszym obciążeniu. Jeśli przydzielony serwer jest przeładowany, natychmiast powinno nastąpić przekierowanie do innego serwera.
- Przezroczystość błędów. Czy w przypadku awarii serwera lub sieci można przekierować klienta do innego serwera bez przerywania usługi? Jeśli tak, to jak szybko można to zrobić? W ciągu kilku sekund, minut? Jakie rozwiązanie można zaakceptować w przypadku Twojego problemu biznesowego?
- Integracja warstw. Kod musi być tak napisany, by utrwalać dane biznesowe w bazach danych oraz by mógł współpracować z aktualnie działającym systemem.
- Transakcje. Co stanie się, jeśli klienty będą jednocześnie próbowały dostać się tego samego rekordu w bazie danych? Co stanie się, jeśli baza danych ulegnie awarii? Rozwiązaniem tych problemów jest system transakcji.
- Grupowanie. Co stanie się, jeśli serwer stanowy ulegnie awarii? Czy jego stan był zachowany na wszystkich pozostałych serwerach tak, by klient mógł kontynuować swoje działania na innym serwerze?
- Dynamiczne przegrupowania. Jak dokonać aktualizacji systemu w czasie jego działania? Czy musisz wyłączyć usługę czy może działać cały czas?
- Bezpieczne wyłączenie. Jeśli musisz przerwać działanie serwera, czy możesz to zrobić gładko, bez potrzeby przerywania usług aktualnie połączonych klientów?
- Zapisywanie w dzienniku i kontrolowanie. Jeśli coś nie działa poprawnie, czy mamy zapis wszystkich operacji, który umożliwia określenie przyczyny błędu? Dziennik mógłby nam pomóc wykryć i usunąć błąd tak, by więcej się nie powtórzył.
- Zarządzanie systemami. W przypadku katastrofalnego błędu, kto jest odpowiedzialny za monitorowanie naszego systemu? Chcielibyśmy dysponować oprogramowaniem monitorującym, które, w przypadku awarii, poinstruuje administratora systemu o kierunkach możliwych działań.
- Wątki. Obecnie działające systemy charakteryzują się dużą liczbą użytkowników łączących się z pojedynczym serwerem. Taki serwer musi być w stanie obsłużyć wiele żądań od klientów jednocześnie. Musi więc być zaprogramowany jako aplikacja wielowątkowa.
- Oprogramowanie oparte na przesyłaniu komunikatów. Pewne rodzaje żądań wymagają komunikacji opartej na komunikatach, która bardzo luźno wiąże klienta z serwerem. Potrzebujemy więc całej infrastruktury przeznaczonej do obsługi mechanizmów wymiany wiadomości.
- Cykl życia obiektu. Obiekty istniejący wewnątrz aplikacji serwera powinny być tworzone i usuwane w zależności od, odpowiednio, zwiększającego lub zmniejszającego się obciążenia serwera.

- ♦ Zarządzanie zasobami. Jeśli dany klient aktualnie nie korzysta z usług serwera, cenne zasoby serwera można zwrócić do puli dostępnych zasobów tak, by można z nich było skorzystać, jeśli do serwera podłączy się inny klient. Do takich zasobów zaliczamy gniazda (np. połączenia do bazy danych) oraz obiekty istniejące wewnątrz serwera.
- ♦ Bezpieczeństwo. Serwery i bazy danych muszą być zabezpieczone przed włamaniami. Uprawnieni użytkownicy powinni mieć dostęp jedynie do określonych operacji, które są niezbędne w ich pracy.
- ♦ Pamięć podręczna. Załóżmy, że mamy pewne dane w bazie danych, do których mają dostęp wszystkie klienty, jak np. wspólny katalog produktów. Po co serwer miałby wielokrotnie wyciągać te same dane z bazy danych. Można przecież przechowywać je w pamięci serwera i ograniczyć w ten sposób kosztowne operacje sieciowe i zapytania do bazy danych.
- ♦ I wiele, wiele innych zagadnień.

Każde z powyższych zagadnień wymaga osobnego obsłużenia i wykonania poważnych obliczeń po stronie aplikacji serwera. Obsługa wspomnianych problemów jest niezbędna w przypadku wszystkich rozwiązań biznesowych. Wymaga sporej wiedzy i stosowania dobrych struktur. Połączone usługi rozwiązujące wypunktowane powyżej kwestie nazywamy oprogramowaniem pośredniczącym.

Dziel i zwyciężaj

Omówiliśmy właśnie sposób tworzenia swojego oprogramowania pośredniczącego na bazie **serwera aplikacji**, co umożliwia nam skupienie się na samym problemie biznesowym. Mamy jednak jeszcze lepszą wiadomość: istnieje możliwość zakupu i wykorzystania częściowych rozwiązań tego problemu.

Aby osiągnąć swój cel, musisz zbudować aplikację, składając dostępne **komponenty**. Komponent jest kodem implementującym dobrze zdefiniowane interfejsy. Jest łatwym do zarządzania, odrębnym fragmentem logiki aplikacji. Komponenty same w sobie nie są aplikacjami — nie mogą działać samodzielnie. Mogą być raczej wykorzystane jak elementy układanki, która rozwiązuje szerszy problem.

Idea zastosowania komponentów programowych jest pomysłem dającym wielkie możliwości. Firma może kupić dobrze zdefiniowany moduł rozwiązujący dany problem i połączyć go z innymi komponentami rozwiązującymi większe problemy. Przykładowo, rozważ komponent programowy obliczający ceny towarów. Nazwijmy go *komponentem cenowym*. Przekazujesz temu komponentowi dane o zbiorze produktów i otrzymujesz od niego łączną wartość zamówienia.

Problem określania ceny może być całkiem skomplikowany. Przykładowo, załóżmy, że zamawiamy części komputerowe, jak pamięć czy dysk twardy. Komponent cenowy zwraca poprawną cenę w oparciu o *zasady wyceny*, które mogą zawierać:

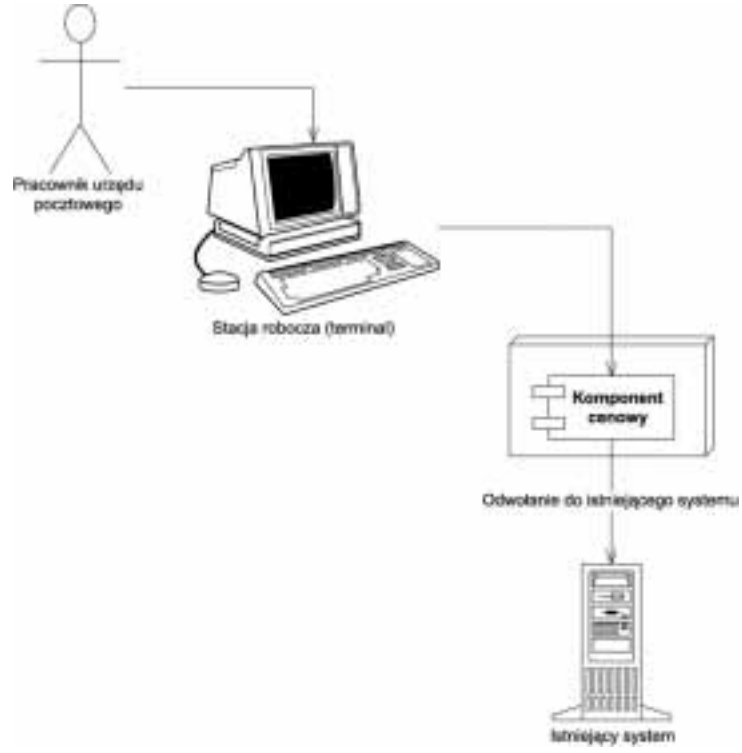
- ♦ **Cenę podstawową** pojedynczej kości pamięci lub dysku twardego.
- ♦ **Rabat ilościowy** przyznawany klientowi przy zakupie powyżej 10 modułów pamięci.

- **Rabat zestawowy** przyznawany klientowi za zakup zarówno pamięci, jak i dysku twardego.
- **Rabat stałego klienta**, który przyznaje się ważnym klientom.
- **Rabat miejscowy** zależny od miejsca zamieszkania klienta.
- **Dodatkowe koszty**, jak koszt przesyłki czy podatek.

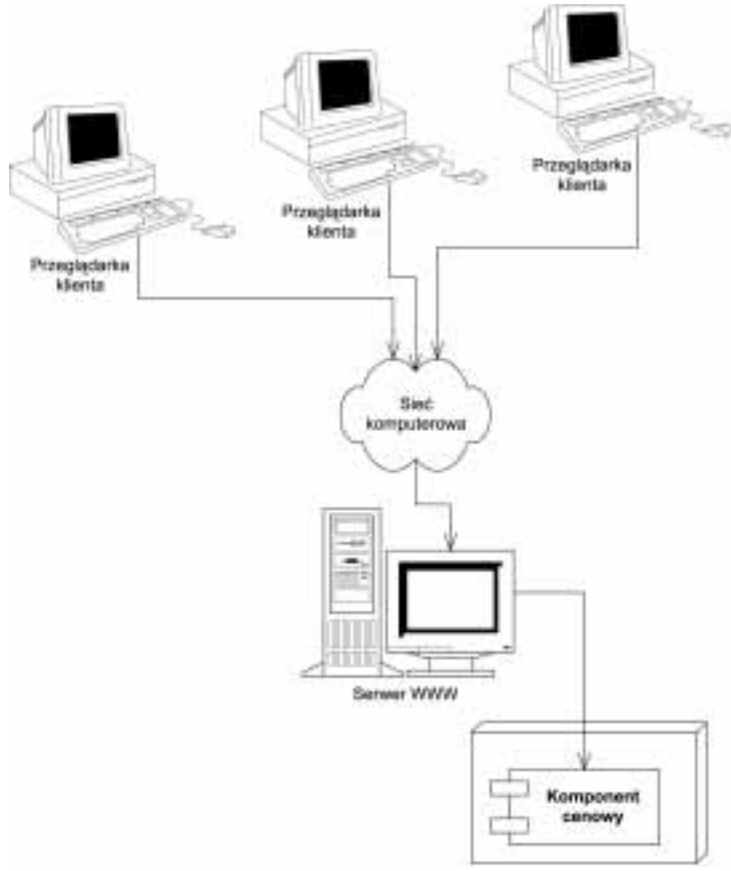
Powyższe zasady wyceny nie dotyczą oczywiście jedynie części komputerowych. W innych dziedzinach, takich jak sprzedaż urządzeń dla opieki zdrowotnej czy biletów lotniczych, wymagana jest podobna do opisanej powyżej funkcjonalność. Byłoby oczywiście wielkim marnotrawstwem środków, gdyby wszystkie przedsiębiorstwa potrzebujące skomplikowanego mechanizmu wyceny, musiały pisać własne wyszukane rozwiązania tego problemu. Sensowny staje się więc handel gotowymi, uogólnionymi wersjami komponentów obliczających wartości zamówień, które mogą być wielokrotnie wykorzystywane przez różne przedsiębiorstwa. Przykładowo:

1. Poczta Polska może wykorzystać komponent naliczania cen do wyznaczania kosztów wysyłania paczek. Prezentujemy to na rysunku 1.2.

Rysunek 1.2. *Ponowne wykorzystanie komponentu cenowego w urzędzie pocztowym*



2. Producent samochodów może używać tego komponentu do wyznaczania cen samochodów. Wystarczy, że umieści w Internecie witrynę umożliwiającą klientom firmy zapoznanie się z cenami samochodów w zależności, na przykład, od wyposażenia. Takie zastosowanie prezentujemy na rysunku 1.3.

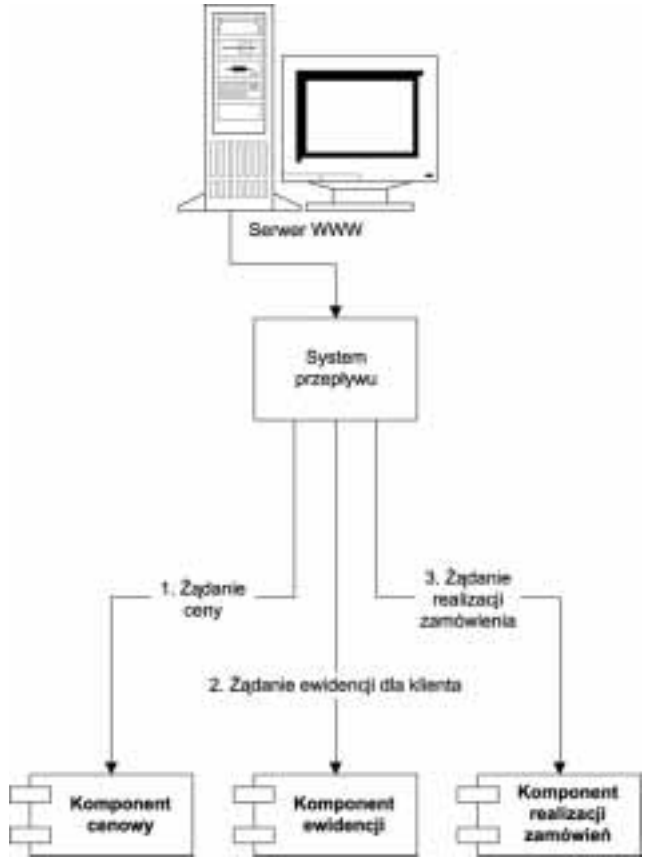


Rysunek 1.3. *Ponowne wykorzystanie komponentu cenowego do wyznaczania cen samochodów w zależności od wskazanego przez użytkownika wyposażenia*

3. Internetowy sklep spożywczy może wykorzystywać nasz komponent jako jedną z odrębnych części kompletnego systemu informatycznego. Gdy użytkownik kupuje artykuły spożywcze przez Internet, komponent cenowy oblicza najpierw wartość zamówienia, następnie inny komponent ewidencjonuje zamówienie z wygenerowaną ceną, a na końcu, trzeci komponent wypełnia zamówienie, ustalając konkretne towary ze sklepu przeznaczone dla użytkownika końcowego. Takie rozwiązanie zaprezentowaliśmy na rysunku 1.4.

Komponenty, które możemy wielokrotnie wykorzystywać, są rozwiązaniem bardzo kuszącym ze względu na możliwość błyskawicznego opracowywania aplikacji. Tworząc sklep internetowy, można znacznie szybciej zmontować kompletną aplikację z napisanych wcześniej komponentów, niż pisząc całą aplikację począwszy od szkicu. Oznacza to, że:

- **Stworzenie takiego sklepu wymaga znacznie mniej wiedzy specjalistycznej.** W naszym podejściu możemy postrzegać komponent cenowy jak czarną skrzynkę i nie potrzebujemy skomplikowanej wiedzy na temat algorytmów wyznaczania cen.



Rysunek 1.4. *Ponowne wykorzystanie komponentu cenowego jako części kompletnego rozwiązania realizującego problem handlu elektronicznego*

- **Aplikacja jest tworzona szybciej.** Dostawca komponentu napisał już część systemu, możemy więc wykorzystać włożoną przez niego pracę w celu ograniczenia czasu niezbędnego do realizacji całego projektu.
- **Koszt posiadania systemu jest mniejszy.** Źródłem utrzymania dostawcy naszego komponentu są właśnie sprzedawane komponenty. Jeśli więc dany producent komponentów chce utrzymać się na rynku, musi dostarczyć wyczerpującą dokumentację i pomoc techniczną. Ponieważ sprzedawca komponentów jest ekspertem w tej dziedzinie, dostarczony przez niego fragment oprogramowania ma mniej błędów i jest wydajniejszy niż oprogramowanie tworzone samodzielnie. Wszystkie te właściwości kupowanych komponentów sprawiają, że koszt utrzymania sklepu internetowego jest niższy.

Gdy określono już zasady pisania i wykorzystywania komponentów, narodził się rynek komponentów, na którym ich producenci mogą wielokrotnie sprzedawać firmom te same produkty. Komponenty umieszcza się na serwerach aplikacji, czego efektem jest kompletne oprogramowanie pośredniczące.

Czy rynek komponentów jest mitem?

Obecnie istniejący rynek komponentów jest bardzo mały. Przez całe lata mieliśmy nadzieję, że rynek ten w pewnym momencie eksploduje, ale wciąż pozostaje to w sferze planów. Istnieje wiele powodów, dla których niezależni sprzedawcy oprogramowania (ang. *Independent Software Vendors — ISVs*) nie udostępniają swoich komponentów:

- **Dojrzałość.** Ponieważ komponenty działają w serwerach aplikacji, serwery te muszą mieć ostateczny kształt, zanim będzie można tworzyć dla niech komponenty.
- **Polityka.** Wielu niezależnych twórców napisało swoje własne oprogramowanie dla serwerów aplikacji. Niektórzy z nich bezpodstawnie uważają to za źródło swojej wysokiej pozycji na rynku.
- **Niepewna wartość.** Większość wydawców oprogramowania uzależnia swoją produkcję od określonych potrzeb konkretnych klientów. Ponieważ, dla wielu klientów, komponenty są czymś nowym, nie zgłaszają producentom oprogramowania zapotrzebowania na tego typu produkty.

W naszej opinii rynek komponentów ostatecznie eksploduje, jest to jedynie kwestia czasu. Jeśli pracujesz dla niezależnego sprzedawcy oprogramowania, powstanie takiego rynku może być dla Ciebie wielką szansą.

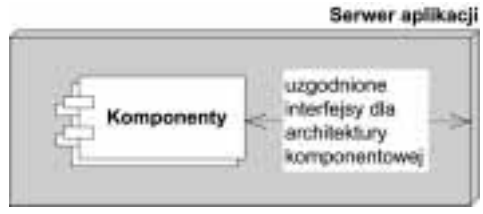
Dobrą wiadomością jest to, że taki rynek już zaczyna się wyłaniać. Większość dostawców rozwiązań dla handlu elektronicznego (jak Ariba, Broadvision, Vignette itp.) oferują lub ogłosili, że będą mieli w ofercie, obsługę serwerów w pracujących technologii opartej na Javie.

W międzyczasie będziesz musiał zbudować swoje własne komponenty na podstawie szkicu funkcjonowania Twojej firmy. Niektórzy z klientów firmy Middleware Company próbują robić to, wyznaczając grupy programistów, których zadaniem jest tworzenie komponentów dla innych działów. A zatem, zespoły te funkcjonują w rzeczywistości jak wewnętrzni niezależni sprzedawcy oprogramowania.

Architektura komponentowa

Minęło wiele lat, zanim przyjął się pomysł architektury wielowarstwowej dla serwerów. W tym czasie na rynku pojawiło się ponad pięćdziesiąt serwerów aplikacji. Początkowo, wszystkie one udostępniały usługi komponentowe w zastrzeżony dla pojedynczych rozwiązań, niestandardowy sposób. Działo się tak, ponieważ nie istniała uzgodniona wspólnie definicja komponentów. Efekt? Gdy raz zdecydowałeś się na konkretny serwer aplikacji, Twoje możliwości doboru modułów ograniczały się do oferty dostawcy serwera. To znacznie ograniczyło przenośność stosowanych rozwiązań i stało się źródłem sukcesu Javy, która promowała otwartość i właśnie przenośność. Sytuacja w świecie serwerów aplikacji była ogromną przeszkodą dla handlu komponentami, ponieważ ich sprzedawcy nie mogli łączyć komponentów napisanych dla jednego serwera aplikacji z komponentami napisanymi dla innego serwera.

Potrzebne jest więc porozumienie dotyczące zbioru interfejsów pomiędzy serwerami aplikacji a komponentami. Takie porozumienie pozwoliłoby na stosowanie dowolnych komponentów na dowolnych serwerach aplikacji. W takim układzie możliwe byłoby wymienianie komponentów w różnych serwerach aplikacji bez potrzeby zmiany kodu czy rekompilacji samych komponentów. Efektem takiego porozumienia jest architektura komponentowa (patrz rysunek 1.5).



Rysunek 1.5. *Architektura komponentowa*

Jeśli próbujesz wyjaśnić sobie pojęcie komponentów w języku nietechnicznym, proponujemy poniższe analogie:

- Każdy odtwarzacz CD może odtwarzać standardowe płyty kompaktowe. Możesz postrzegać serwer aplikacji jako odtwarzacz CD, zaś komponenty jako płyty.
- W Polsce każdy telewizor może wyświetlać dowolny nadawany w Europie program, ponieważ wykorzystywany jest jeden standard, PAL. Możesz postrzegać serwer aplikacji jako telewizor, zaś komponenty jako programy telewizyjne.

Wprowadzenie Enterprise JavaBeans

Standard Enterprise JavaBeans jest architekturą komponentową dla pisanych w Javie komponentów przeznaczonych dla serwerów. Zgodność komponentów i serwerów aplikacji umożliwia stosowanie dowolnych komponentów wewnątrz dowolnych serwerów aplikacji. Komponenty EJB są łatwe w stosowaniu i mogą być importowane i ładowane w serwerze aplikacji, który wiąże zawarte w sobie komponenty.

Poniżej przedstawiamy trzy najważniejsze zalety technologii EJB:

1. Jest efektem porozumienia pomiędzy producentami. Każdy użytkownik komponentów EJB może będzie w stanie wykorzystać ich rozpowszechnione zastosowania. Ponieważ programowanie komponentów EJB jest we wszystkich przypadkach podobne, łatwiej będzie w przyszłości zatrudnić pracowników, którzy zrozumieją nasz istniejący system (jeśli będą mieli jakieś wcześniejsze doświadczenia w programowaniu EJB), łatwiejsze też będzie poznanie najlepszych technik programowania w celu ulepszenia systemu (np. czytając książki podobne do tej), ułatwiona będzie współpraca z partnerami biznesowymi (ponieważ technologia EJB zapewnia zgodność) i, wreszcie, łatwiejsza będzie sprzedaż oprogramowania, ponieważ klienci będą skłonni zaakceptować nasze rozwiązania. Zasadnicze znaczenie ma tutaj podejście „naucz się raz, koduj gdziekolwiek".
2. Przenośność jest łatwiejsza. Specyfikacja EJB jest opublikowana i dostępna wszystkim za darmo. Ponieważ EJB jest standardem, nie musisz ryzykować stosowania opatentowanego oprogramowania pojedynczego producenta. Mimo że przywilej przenośności nigdy nie będzie darmowy, standardowe rozwiązania są dużo tańsze.

3. Błyskawiczne tworzenie aplikacji. Twoja aplikacja może być konstruowana szybciej, ponieważ otrzymujesz całe oprogramowanie pośredniczące razem z serwerem aplikacji. Ograniczasz również bałagan związany z konserwacją systemu.

Zauważ, że mimo iż komponenty EJB mają wiele zalet, istnieją przypadki, w których ich stosowanie nie jest dobrym rozwiązaniem. Rozdział 15. zawiera kompletną analizę sytuacji, w których należy (lub nie należy) stosować technologię EJB.

Fizycznie komponenty EJB są w rzeczywistości połączeniem dwóch elementów:

- **Specyfikacja.** Istnieje ponad 500-stronicowy plik Adobe Acrobat (PDF), dostępny za darmo w Internecie na stronie *http://java.sun.com*. Specyfikacja porządkuje zasady łączenia komponentów z serwerami aplikacji. Uściśla sposoby zalecane programowania w tej technologii.
- **Zbiór interfejsów Javy.** Komponenty i serwery aplikacji muszą być zgodne z tymi interfejsami. Ponieważ wszystkie komponenty są stworzone z myślą ojednym interfejsie, z punktu widzenia serwera aplikacji są identyczne. Serwer może więc swobodnie zarządzać dowolnymi komponentami. Wspomniane interfejsy możesz za darmo pobrać ze strony *http://java.sun.com*.

Dlaczego Java?

Komponenty EJB muszą być pisane wyłącznie w Javie. Także ich docelowym przeznaczeniem musi być Java. To poważne ograniczenie. Mamy jednak dobrą wiadomość, Java, z wielu powodów, jest idealnym językiem do budowy komponentów.

Rozdzielenie interfejsu od implementacji. Potrzebujemy jasnego podziału na interfejs i implementację komponentów. W końcu klient kupujący komponenty nie powinien się martwić o ich implementację. Ulepszenia i wsparcie oferowane przez dostawcę komponentów byłyby niemal niemożliwe do zrealizowania. Java umożliwia taki podział już na poziomie składni, za pomocą słów kluczowych `interface` i `class`.

Bezpieczeństwo. Struktura języka Java zapewnia znacznie większe bezpieczeństwo niż struktura innych języków. W Javie wątek jest likwidowany w momencie, gdy kończy się działanie programu. Wskaźniki także nie są już problemem. Znacznie rzadziej występują także problemy z pamięcią. Java udostępnia również bogaty zbiór bibliotek, nie jest więc tylko składnią, ale także całym zbiorem napisanych wcześniej i dokładnie przetestowanych bibliotek, które pozwalają programistom unikać pracy nad elementami dawno wymyślonymi. Oferowane przez Javę bezpieczeństwo ma zasadnicze znaczenie dla aplikacji krytycznych. Oczywiście udostępniana funkcjonalność, niezbędna do osiągnięcia pewnego poziomu bezpieczeństwa, może nieco ograniczać wydajność Twojej aplikacji, należy jednak pamiętać, że 90 procent wszystkich programistycznych rozwiązań biznesowych to po prostu osławiony graficzny interfejs użytkownika (ang. *Graphical User Interface — GUI*) do bazy danych. I właśnie baza danych, a nie Java, będzie wąskim gardłem numer jeden Twojego systemu.

Działanie na wielu platformach. Java działa na dowolnej platformie. Ponieważ EJB jest pewnym zastosowaniem tego języka, komponenty powinny dawać się łatwo uruchamiać na wszystkich platformach. Ma to zasadnicze znaczenie dla wielu klientów, którzy zainwestowali już w rozmaite platformy sprzętowe i programowe, jak Windows, UNIX czy komputery mainframe. Oczywiste jest, że takie firmy nie mają zamiaru rezygnować ze swoich dotychczasowych inwestycji.

Jeśli zdecydowałeś się stosować komponenty EJB, musisz dokonać jeszcze jednego wyboru pomiędzy: zarządzaniem komponentami za pomocą technologii .NET firmy Microsoft a Standardem CORBA (*Common Object Request Broker Architecture*) zaproponowanym przez konsorcjum OMG (*Object Management Group*).

Zauważ, że wiele spośród serwerów EJB bazuje na standardzie CORBA i może z tym standardem współpracować (patrz dodatek A, gdzie opisaliśmy sposoby uzyskiwania tego efektu).

EJB jako rozwiązanie biznesowe

EJB jest technologią stosowaną jako pomoc przy rozwiązywaniu problemów biznesowych. Komponenty EJB mogą wykonywać następujące czynności:

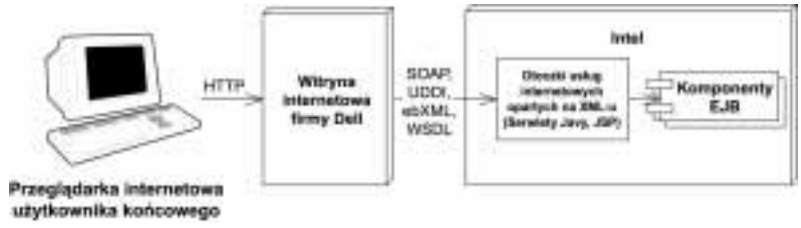
- **Realizacja zadania biznesowego.** Istnieją przykłady stosowania komponentów do obliczania podatków dla koszyka zakupów internetowych z możliwością zatwierdzania zamówienia przez menadżera lub wysyłania listu elektronicznego w celu jego potwierdzenia za pomocą interfejsu *JavaMail*.
- **Dostęp do bazy danych.** Mamy także przykłady wysyłania zamówienia na książki, transferu pieniędzy pomiędzy dwoma kontami bankowymi itp. Komponenty EJB realizują dostępy do bazy danych za pomocą interfejsu *Java Database Connectivity* (JDBC).
- **Dostęp do innego systemu.** Istnieją przykłady wywoływania wydajnych istniejących systemów napisanych w COBOL-u, które obliczają wskaźnik ryzyka dla nowego konta ubezpieczeniowego, wywołując, na przykład, system *SAP R/3*. Komponenty EJB wykorzystują do integracji z istniejącymi aplikacjami interfejs *Java Connector Architecture* (JCA).

EJB nie są komponentami graficznego interfejsu użytkownika; kryją się raczej za tym interfejsem i wykonują całą ciężką pracę systemu. Oto przykłady graficznych interfejsów użytkownika, które mogą być połączone z odpowiednimi komponentami EJB:

- **Pełny klient.** Pełny klient jest uruchamiany na komputerze użytkownika. Może się łączyć z komponentami EJB znajdującymi się na serwerze za pośrednictwem sieci komputerowej. Dopiero tak wywołane komponenty realizują opisane powyżej przykładowe zadania. Pełny klient to w rzeczywistości aplet lub aplikacja.

- **Dynamicznie generowane strony WWW.** Skomplikowane witryny internetowe muszą składać się ze stron, których treść zależy od określonego żądania. Przykładowo, strona główna *Amazon.com* jest całkowicie inna dla każdego użytkownika, jej wygląd zależy od jego profilu. Do generowania takich stron wykorzystujemy serwlety Javy i JavaServer Pages (JSP). Zarówno serwlety, jak i JSP działają na serwerze i mogą łączyć się z komponentami EJB w celu wygenerowania różnych stron w zależności od wartości otrzymanych z tych komponentów.
- **Otoczki usług internetowych opartych na XML-u.** Niektóre aplikacje biznesowe nie wymagają żadnego interfejsu użytkownika. Istnieją tylko po to, by łączyć się z innymi aplikacjami, które takie interfejsy udostępniają. Przykładowo, firma Dell Computer Corporation musi kupić procesory Intela do wyprodukowania swoich komputerów. Intel mógłby udostępnić serwis internetowy, który umożliwiłby oprogramowaniu firmy Dell łączenie się z jego systemem i zamawianie potrzebnej liczby procesorów. W takim przypadku system Intela nie musi zawierać żadnego interfejsu użytkownika, wystarczy, że będzie funkcjonował jako usługa internetowa. Można w takich sytuacjach stosować takie technologie jak SOAP, UDDI, ebXML czy WSDL. Pokazaliśmy to na rysunku 1.6.

Rysunek 1.6. *Komponenty EJB jako zaplecze usług internetowych*



Prawdziwą różnicą pomiędzy komponentami graficznego interfejsu użytkownika (pełne klienty, strony generowane dynamicznie, otoczki usług internetowych) a komponentami EJB jest domena, której interesujące nas komponenty są częścią. Komponenty interfejsu użytkownika są dostosowane do wykonywania takich operacji po stronie klienta jak generowanie elementów tego interfejsu (chociaż nie zawsze jest to konieczne), realizacja czynności związanych z prezentacją danych czy proste operacje biznesowe. Reprezentują system bezpośrednio przed użytkownikiem końcowym lub partnerem biznesowym.

Komponenty EJB nie są stworzone do działania po stronie klienta; są typowymi komponentami dla serwerów. Ich zadaniem jest wykonywanie operacji charakterystycznych dla serwerów, jak realizacja skomplikowanych algorytmów czy rozbudowanych transakcji biznesowych. Oprogramowanie serwera ma zupełnie inne potrzeby niż bogate środowisko interfejsów graficznych. Komponenty działające na serwerach muszą charakteryzować się wysoką dostępnością, odpornością na błędy, działaniem transakcyjnym oraz bezpiecznym środowiskiem dla wielu użytkowników. Serwer aplikacji udostępnia wysokiej jakości środowisko dla komponentów EJB oraz bezpieczną „obudowę" niezbędną do zarządzania tymi komponentami.

„Ekosystem" EJB

Aby uruchomić swoje rozwiązanie oparte na komponentach EJB, nie wystarczy serwer aplikacji i wspomniane komponenty. W rzeczywistości, stosowanie EJB wymusza współpracę *ponad sześciu różnych* elementów. Każdy z nich jest „ekspertem" w jednej dziedzinie i odpowiada za jeden z kluczowych fragmentów całego procesu pomyślnego uruchamiania systemu. Ponieważ każda część jest wyspecjalizowana, łączny czas budowy całego rozwiązania jest znacznie ograniczony. Razem, wspomniane elementy tworzą *ekosystem EJB*.

Zajmijmy się teraz elementami tego ekosystemu. W miarę czytania tego opisu, spróbuj pomyśleć o roli, jaką odgrywasz w modelu biznesowym Twojej firmy. Jeśli nie jesteś pewien, zadaj sobie pytanie, jaki nadrzędny cel realizujesz w swoim przedsiębiorstwie. Pomyśl także, jaką rolę możesz odgrywać w kolejnych projektach.

Ekosystem EJB nie jest przeznaczony dla każdego. W mojej firmie słyszałem okropne opowieści o przedsiębiorstwach, w których wybrano technologię EJB tylko dlatego, że korzystali z niej wszyscy wokół, lub dlatego, że była to technologia nowa i bardzo interesująca. To nie są najlepsze powody korzystania z komponentów EJB — efekty tak motywowanego wyboru mogą być fatalne. Kompletne rozważania na temat przypadków, w których stosowanie EJB jest (lub nie jest) pożądane omówimy w rozdziale 15.

JavaBeans. Enterprise JavaBeans

Mogłeś słyszeć o innym standardzie zwanym *JavaBeans*. Jest to zupełnie coś innego niż Enterprise JavaBeans.

Mówiąc w największym skrócie, JavaBeans to klasy Javy zawierające standardowy zestaw metod `get` i `set`. Są komponentami Javy wielokrotnego użytku zawierającymi własności, zdarzenia i metody (podobne do kontrolek ActiveX firmy Microsoft), które można łatwo wykorzystać w celu stworzenia (często wizualnych) aplikacji Javy.

JavaBeans są znacznie mniejsze od Enterprise JavaBeans. Możesz je wykorzystać do budowy większych komponentów lub całych aplikacji. JavaBeans nie są jednak komponentami przeznaczonymi do natychmiastowego stosowania, wymagają raczej pewnej obróbki. Nie umieszczasz przeważnie komponentu gotowego JavaBeans w swojej aplikacji, próbujesz go raczej wykorzystać do budowy większej aplikacji, którą już będzie można stosować. Ponieważ komponenty te nie mogą funkcjonować samodzielnie, nie wymagają specjalnego środowiska uruchamiania. Ponieważ JavaBeans są zwykłymi klasami Javy, nie potrzebują serwera aplikacji do inicjalizacji, niszczenia czy łączenia z innymi usługami. Sama aplikacja jest bowiem zbudowana z komponentów JavaBeans.

Dostawca komponentu

Dostawca komponentów oferuje nam komponenty biznesowe lub EJB. Komponenty EJB nie są kompletnymi aplikacjami, tylko gotowymi elementami, które możemy połączyć, tworząc kompletne rozwiązanie. Dostawcą komponentu może być niezależny sprzedawca oprogramowania lub wewnętrzny dział firmy udostępniający komponenty innym działom.

Wielu sprzedawców już dzisiaj oferuje swoje komponenty wielokrotnego użytku. Ich pełną listę możesz znaleźć na stronach *www.componentsource.com* lub *www.flashline.com*. W przyszłości tradycyjni producenci oprogramowania dla przedsiębiorstw (którzy dzisiaj oferują oprogramowanie automatyzujące, zarządzania zasobami, usług finansowych czy handlu elektronicznego) będą oferować swoje produkty w formie komponentów EJB lub przynajmniej połączenia do swoich aktualnych technologii.

Programista aplikacji

Programista aplikacji jest jednocześnie jej ogólnym architektem. Jego część pracy jest szczególnie ważna dla zrozumienia, jak poszczególne komponenty ze sobą współpracują. Efektem pracy programisty jest prawidłowe połączenie dostępnych komponentów. Taki architekt systemu może niekiedy nawet sam tworzyć część niezbędnych komponentów, jego praca polega na budowie aplikacji z komponentów, które można przecież ze sobą połączyć na wiele sposobów. Programista aplikacji jest klientem, który wykorzystuje komponenty dostarczone przez dostawcę komponentów.

Programista aplikacji może wykonywać niektóre lub wszystkie z poniższych zadań:

- Na podstawie swojej wiedzy o konkretnym problemie biznesowym decyduje, która z kombinacji dostępnych komponentów oraz nowych EJB będzie najlepsza do realizacji efektywnego rozwiązania; mówiąc najkrócej, planuje montaż.
- Zaopatruje system w interfejs użytkownika (przeważnie Swing, serwlet lub JSP, aplikacja lub aplet albo tylko usługa internetowa).
- Pisze własne komponenty EJB rozwiązujące problemy specyficzne dla danego problemu biznesowego.
- Pisze kod programu wywołującego komponenty dostarczone przez dostawcę komponentów.
- Pisze kod integrujący, który dokonuje odwzorowania danych otrzymywanych z komponentów dostarczonych przez różnych producentów. Komponenty nie będą przecież współpracować ze sobą w magiczny sposób w celu rozwiązania danego problemu biznesowego. Odpowiednie odwzorowanie danych jest szczególnie ważne, gdy komponenty pochodzą od różnych sprzedawców.

Programistą aplikacji może być integrator systemu, firma doradcza lub programista pracujący w swoim domu.

Wdrożeniowiec EJB

Po tym, gdy programista zbuduje aplikację, należy ją jeszcze wdrożyć (uruchomić) w działającym w przedsiębiorstwie środowisku. Oto niektóre z wyzwań stojących przed wdrożeniowcem takiej aplikacji:

- Zabezpieczenie wdrażanej aplikacji za pomocą zapory ogniowej (ang. *firewall*) i innych środków ochronnych.
- Integracja z serwerem LDAP w celu otrzymania list bezpieczeństwa (np. Lotus Notes lub Microsoft Active Desktop).

- Wybranie sprzętu umożliwiającego osiągnięcie wymaganego poziomu wydajności.
- Dostarczenie rezerwowego sprzętu i nadliczbowych zasobów w celu zapewnienia niezawodności i tolerancji dla błędów.
- Dostrojenie systemu w celu poprawy wydajności.

Zazwyczaj twórca aplikacji (który często jest programistą lub analitykiem systemowym) nie zna się na powyższych zagadnieniach. Z tego właśnie powodu niezbędne jest zaangażowanie wdrożeniowca EJB, który jest wyczulony na specyficzne wymagania środowiska operacyjnego i bez trudu jest w stanie wykonać powyższe czynności. Wdrożeniowcy wiedzą, jak należy umieszczać komponenty w serwerze i jak dostosowywać je do pracy w konkretnym środowisku. Mają wolną rękę w dostosowywaniu komponentów do środowiska, w którym system jest wdrażany.

Wdrożeniowiec EJB może być pracownikiem firmy, zewnętrznym konsultantem lub przedstawicielem dostawcy komponentów. Przykłady działania takich wdrożeniowców można znaleźć w firmach Loudcloud i *HostJ2EE.com*, które oferują kompleksowe rozwiązania związane z wdrożeniami rozwiązań opartych na komponentach EJB.

Administrator systemu

Kiedy system już zadziała, administrator rozpoczyna nadzorowanie jego stabilnego funkcjonowania. Administrator systemu jest odpowiedzialny za utrzymanie i monitorowanie uruchomionego systemu. Może do tego celu wykorzystywać narzędzie do monitorowania i zarządzania udostępniane przez serwer EJB.

Przykładowo, wyszukane serwery EJB mogą ostrzegać administratora systemu w przypadku wystąpienia błędu, który wymaga jego natychmiastowej interwencji. Niektóre serwery realizują tę usługę za pomocą odpowiednich interfejsów do profesjonalnych rozwiązań monitorujących, jak np. Tivoli czy Computer Associates. Inne udostępniają własne systemy zarządzania za pomocą modułu obsługi *Java Management Extension* (JMX).

Dostawca kontenera i serwera

Dostawca kontenera udostępnia *kontener EJB*, czyli serwer aplikacji. Jest to środowisko uruchamiania i działania komponentów EJB. Kontener zawiera usługi oprogramowania pośredniczącego, które umożliwiają działanie komponentów i zarządzanie nimi. Przykładem kontenera EJB jest WebLogic firmy BEA, iPlanet Application Server firmy iPlanet, WebSphere firmy IBM, Oracle 9 i firmy Oracle, JRun firmy Macromedia, PowerTier firmy Persistence, Gemstone/J firmy Brokat, Bluestone firmy HP, iPortal firmy IONA, AppServer firmy Borland czy dostępny za darmo kod źródłowy serwera aplikacji JBoss.

Dostawca serwera aplikacji jest w rzeczywistości dostawcą kontenera EJB. Firma Sun nie rozróżnia tych dwóch pojęć, będziemy więc stosować je w tej książce zamiennie.

Cechy usług związanych z EJB

Sposób monitorowania działania systemu opartego na komponentach EJB nie jest opisany w specyfikacji EJB. Jest opcjonalną usługą dostępną w zaawansowanych serwerach aplikacji. Oznacza to, że każdy serwer może realizować to zadanie w inny sposób.

Początkowo możesz myśleć, że taka usługa ogranicza przenośność aplikacji. W rzeczywistości jednak usługa ta powinna być całkowicie przezroczysta dla użytkownika, powinna działać na drugim planie i nie powinna mieć wpływu na kod aplikacji. Od specyfiki serwera zależy, czy usługa monitorowania znajduje się poniżej poziomu aplikacji, na poziomie systemu czy nie. Zmiana serwera aplikacji nie powinna jednak mieć wpływu na kod naszego programu.

Pozostałe, nie opisane w specyfikacji EJB, przezroczyste usługi to: równoważenie obciążeń, przezroczystość awarii, zapisywanie w pamięci podręcznej, grupowanie czy algorytmy zarządzania połączeniami.

Sprzedawcy narzędzi

Aby wspomóc proces tworzenia komponentów, powinna istnieć standardowa metoda budowy, zarządzania i konserwacji komponentów. W ekosystemie EJB istnieje wiele zintegrowanych środowisk programowania (ang. *Integrated Development Environments — IDEs*) wspomagających budowanie i testowanie komponentów. Przykładowe środowiska to Visual Cafe firmy Webgain, VisualAge for Java firmy IBM czy JBuilder firmy Borland.

Inne narzędzia umożliwiają modelowanie komponentów w zunifikowanym języku modelowania (ang. *Unified Modeling Language — UML*), który opiera się na diagramach i właśnie notację UML wykorzystujemy w tej książce. Możesz następnie, na podstawie diagramów UML, wygenerować automatycznie kod komponentów EJB. Przykładami produktów realizujących to zadanie są Together/J firmy Togethersoft i Rational Rose firmy Rational.

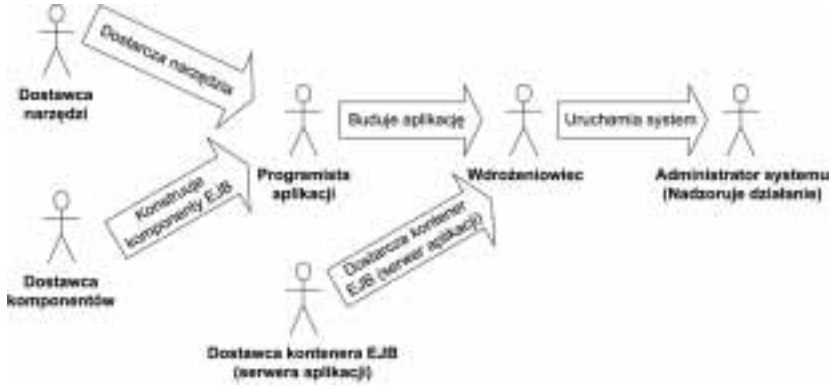
Istnieją także inne narzędzia, które umożliwiają organizację komponentów (Flashline, ComponentSource), testowanie (JUnit, RSW Software) czy ich budowę (Ant).

Podsumowanie ról

Rysunek 1.7 prezentuje podsumowanie zależności pomiędzy poszczególnymi rolami w procesie tworzenia aplikacji opartej na komponentach EJB.

Możesz się zastanawiać, dlaczego w tworzenie rozwiązań opartych na komponentach EJB zaangażowana jest tak duża liczba specjalistów. Dzieje się tak dlatego, że technologia EJB umożliwia stosującym ją firmom lub osobom kształcenie ekspertów do konkretnych ról, co sprawia, że możliwe jest stworzenie jasnego podziału odpowiedzialności w zaangażowanym w prace nad systemem zespole.

Specyfikacja EJB czyni każdą rolę jasną i różną od pozostałych, co umożliwia ekspertom z różnych dziedzin współpracę w projekcie, nie tracąc jednocześnie możliwości ich wymienialności. Zauważ także, że niektóre role można łączyć. Przykładowo, obecnie dostępne serwery EJB i kontenery EJB pochodzą od tych samych dostawców.



Rysunek 1.7. *Powiązania ról zaangażowanych w tworzenie aplikacji EJB*

W małych firmach dostawca komponentów, programista aplikacji i wdrożeniowiec może być tą samą osobą, która próbuje od podstaw zbudować rozwiązanie biznesowe oparte na komponentach EJB. Jaką rolę chciałbyś odgrywać?

Niektóre role jedynie sugerują możliwe zadania stawiane odpowiednim pracownikom, na przykład administrator systemu może nadzorować poprawne funkcjonowanie działającego systemu, ale może też mieć przypisane inne zadania. W innych przypadkach, na przykład w odniesieniu do dostawców komponentów czy kontenera, specyfikacja EJB ściśle określa zasady, którymi muszą się kierować, by uniknąć załamania całego ekosystemu EJB. Dzięki jasno zdefiniowanym rolom, technologia EJB jest solidną podstawą dla rozproszonej, skalowalnej architektury komponentowej, w której możliwa jest swobodna wymiana rozwiązań pochodzących od wielu producentów.

Przyszła specyfikacja EJB będzie definiowała nową rolę, nazwaną menadżerem trwałości, która będzie związana z działaniem serwera aplikacji. Twoje komponenty będą potrzebowały menadżera trwałości do odwzorowania danych biznesowych w pamięci, na przykład przekształcając obiekty w relacyjną bazę danych.

Zadaniem menadżera trwałości będzie utrwalanie danych biznesowych za pomocą odpowiednich rodzajów pamięci. Będzie mógł wykorzystywać płaski system plików, relacyjną bazę danych, obiektową bazę danych lub istniejący, specyficzny dla danej firmy, system.

Dostawcą menadżera trwałości może być dostawca kontenera (serwera) EJB, jest tak w przypadku rozwiązania WebSphere firmy IBM, które zawiera wbudowane narzędzia zarządzania trwałością. Przykładami osobnych produktów oferowanych przez niezależnych dostawców są TOPLink firmy WebGain oraz Cocobase firmy Thought Inc.

Niestety, rola dostawcy menadżera trwałości nie została zdefiniowana wprost w specyfikacji EJB 2.0. Z powodu ograniczeń czasowych standard wykorzystywania menadżera trwałości w serwerach aplikacji pozostaje nieokreślony aż do czasu ukazania się następnej wersji EJB. Dobrą wiadomością jest natomiast informacja o braku wpływu tego typu narzędzi na przenośność Twojego kodu, ponieważ na poziomie aplikacji nie musisz przejmować się sposobem utrwalania przetwarzanych danych (bezpośrednio przez kontener bądź za pomocą dodatkowego modułu zarządzania trwałością). Złą wiadomością jest natomiast konieczność opierania swoich rozwiązań na specyficznych układach pomiędzy producentami menadżerów trwałości a dostawcami serwerów aplikacji, co oznacza, że nie każdy menadżer trwałości będzie współpracował z dowolnym serwerem.

Platforma Java 2, Enterprise Edition (J2EE)

EJB jest tylko częścią większej technologii oferowanej przez firmę Sun Microsystems, zwanej platformą Java 2, Enterprise Edition (J2EE). Celem wydania J2EE jest dostarczenie niezależnego od platformy, przenośnego, wielowątkowego, bezpiecznego i standaryzowanego rozwiązania dla programowania serwerów w języku Java.

J2EE jest specyfikacją, a nie konkretnym produktem. Określa pewne zasady, do których należy się stosować podczas tworzenia oprogramowania dla przedsiębiorstw. Korzystając z tej specyfikacji, producenci mogą tworzyć konkretne rozwiązania zgodne z technologią J2EE.

Ponieważ J2EE jest jedynie specyfikacją (rozumianą, jako odpowiedź na potrzeby wielu przedsiębiorstw), nie wiąże się z żadnym konkretnym producentem oprogramowania. Co więcej, umożliwia programowanie dla wielu platform, co zachęca licznych producentów do tworzenia kompletnych, zaawansowanych rozwiązań w tej technologii. Specyfikacja J2EE ma też swoją ciemną stronę: pojawiające się niejednoznaczne definicje i chęć współzawodnictwa powodują niekiedy niezgodność rozwiązań pochodzących od różnych producentów oprogramowania.

J2EE to trzy różne platformy Javy. Każda z nich jest podzbiorem innej, większej platformy:

- **Platforma Java 2, Micro Edition (J2ME)** jest platformą programistyczną przeznaczoną dla urządzeń obsługujących Javę, jak palmtopy, pagery, zegarki itp. W przypadku tej platformy mamy do czynienia z bardzo uproszczonym językiem programowania, co wynika z oczywistych ograniczeń wydajnościowych i ilościowych w tego typu małych urządzeniach.
- **Platforma Java 2, Standard Edition (J2SE)** zawiera standardowe usługi Javy dla apletów i aplikacji, jak ułatwienia w operacjach wejścia-wyjścia, ułatwienia w tworzeniu graficznego interfejsu użytkownika itp. Platforma J2SE zawiera najczęściej wykorzystywany przez programistów zestaw narzędzi Javy (ang. *Java Development Kit — JDK*).
- **Platforma Java 2, Enterprise Edition (J2EE)** skupia w sobie interfejsy programowania Javy przeznaczone dla przedsiębiorstw i stanowi tym samym kompletną platformę programistyczną dla tworzonych w Javie rozwiązań dla serwerów korporacyjnych.

  Pojawienie się J2EE ma ogromne znaczenie, ponieważ nowa platforma unifikuje oprogramowanie serwerów pisane w Javie. J2EE składa się z następujących komponentów dostarczanych przez Sun Microsystems:

- **Specyfikacje.** Każdy interfejs programowy zawarty w J2EE ma własną specyfikację — plik PDF dostępny za darmo na witrynie *http://java.sun.com*. Za każdym razem, gdy pojawia się nowa wersja J2EE, firma Sun wiąże wersje wszystkich specyfikacji interfejsów programowych z wersją nowego

produktu. Takie rozwiązanie zwiększa przenośność kodu, ponieważ wszyscy producenci tworzą oprogramowanie dla dokładnie jednego wydania interfejsów. Podobnie postępuje firma Microsoft, która raz na kilka lat wypuszcza na rynek nową wersję systemu Windows. W takich sytuacjach, firma zawsze dostosowuje wersje pojawiających w tym czasie technologii do najnowszej wersji systemu operacyjnego i wydaje tak oznaczone produkty niemal równocześnie.

- **Pakiet testowy.** Sun udostępnia także pakiet testowy dla producentów serwerów J2EE, by mogli dokładnie przetestować swoje rozwiązania. Jeśli serwer pomyślnie przejdzie testy, firma Sun potwierdza zgodność systemu, co daje klientom pewność, że produkt jest całkowicie zgodny ze specyfikacją J2EE. Obecnie wielu producentów uzyskało już certyfikaty zgodności z technologią J2EE, opinie o ich produktach możesz za darmo znaleźć na stronach *TheServerSide.com*.
- **Przykładowa realizacja.** Aby umożliwić programistom pisanie kodu zgodnego z J2EE tak, jak programowali w JDK, firma Sun udostępniła swoją własną, darmową realizację specyfikacji J2EE. Sun określa swój produkt jako uproszczoną platformę i zastrzega, że nie jest ona przeznaczona do użytku komercyjnego.
- **Dokumentacja.** Każdy z interfejsów programowych zawartych w J2EE ma jasno określone przeznaczenie zdefiniowane przez firmę Sun (dokumenty *J2EE BluePrints*). Odpowiednia dokumentacja jest dostępna do pobrania w formacie pliku PDF, w którym twórca specyfikacji opisuje, jak należy łączyć ze sobą technologie J2EE.

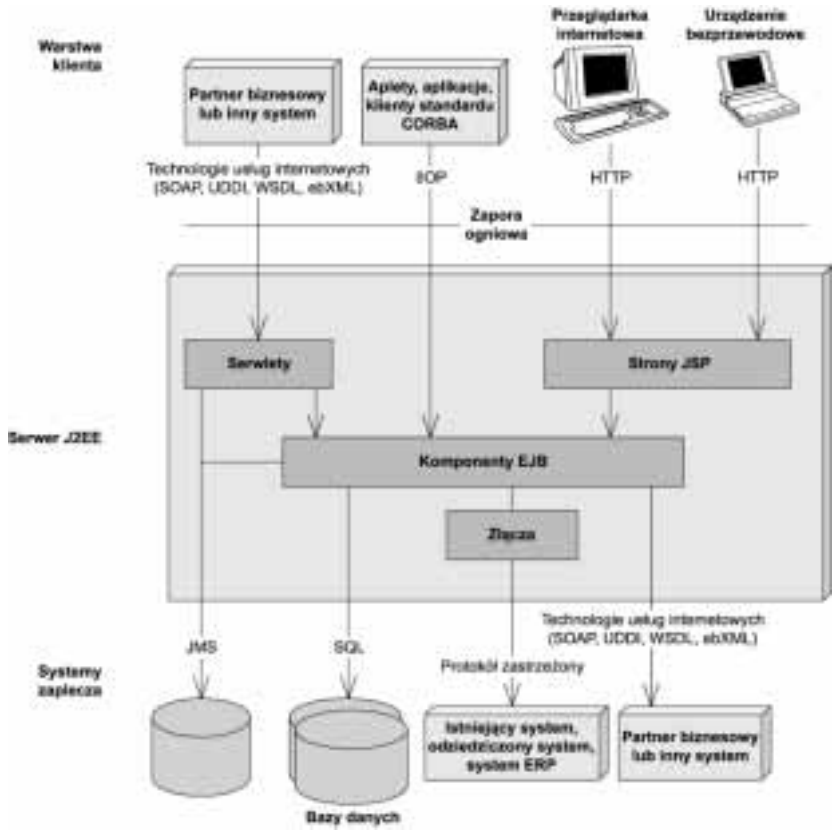
Technologie J2EE

Platforma Java 2, Enterprise Edition jest solidnym pakietem oprogramowania pośredniczącego, który znacznie ułatwia pracę programistów aplikacji dla serwerów. Platforma J2EE powstała na bazie istniejących technologii zawartych w platformie J2SE, która, poza podstawową obsługą Javy, umożliwia stosowanie różnych bibliotek (*.awt*, *.net*, *.io*) i budowę zarówno aplikacji, jak i apletów. Ponieważ J2EE zbudowano na bazie J2SE, produkty zgodne z J2EE muszą realizować zarówno wszystkie założenia J2EE, jak i J2SE. Oznacza to, że budowa rozwiązań zgodnych z platformą J2EE jest ogromnym wyzwaniem. To ograniczenie spowodowało znaczną konsolidację producentów oprogramowania z zakresu rozwiązań korporacyjnych pisanych w Javie. Obecnie na tym rynku pojawia się kilku wyraźnych liderów.

Omówimy J2EE w wersji 1.3, w której włączono obsługę EJB 2.0. Współpracę najważniejszych technologii zawartych w platformie J2EE przedstawiliśmy na rysunku 1.8.

Aby lepiej zrozumieć rzeczywistą wartość J2EE, przedstawiamy poniżej wszystkie interfejsy programowania oferowane przez rozwiązania zgodne ze specyfikacją J2EE 1.3:

- **Enterprise JavaBeans (EJB).** EJB definiuje sposoby pisania i udostępniania komponentów dla serwerów, określa także standard współpracy komponentów z zarządzającymi nimi serwerami aplikacji. Komponenty EJB są nieodłączną częścią platformy J2EE i wykorzystują wiele zawartych w niej technologii.



Rysunek 1.8. *Zastosowanie platformy Java 2, Enterprise Edition*

- **Zdalne wywoływanie metod Javy** (*Java Remote Method Invocation — RMI* oraz *RMI-IIOP*). RMI jest w języku Java naturalnym sposobem komunikowania się obiektów rozproszonych, takich jak dwa różne obiekty działające na dwóch różnych maszynach. RMI-IIOP jest rozszerzeniem RMI umożliwiającym integrację ze standardem CORBA. Oficjalnym interfejsem programowym dostarczanym przez J2EE jest RMI-IIOP (nie RMI) i właśnie ten interfejs omawiamy w dodatku A.
- **Interfejs nazewnictwa i katalogów Javy** (*Java Naming and Directory Interface — JNDI*). Interfejs JNDI jest stosowany do uzyskiwania dostępu z Javy do systemów nazewnictwa i katalogów. Możesz wykorzystywać JNDI w swojej aplikacji do rozmaitych celów, jak łączenie z komponentami EJB lub innymi zasobami za pośrednictwem sieci komputerowej, uzyskiwanie dostępu do danych przechowywanych w innych usługach jak Microsoft Exchange czy Lotus Notes. Interfejs JNDI również omówiony został w dodatku A.
- **Interfejs łączenia Javy z bazami danych** (*Java Database Connectivity — JDBC*). JDBC jest interfejsem programowym umożliwiającym uzyskiwanie dostępu do relacyjnych baz danych. Zaletą JDBC jest możliwość dostępu do dowolnej relacyjnej bazy danych za pomocą jednego interfejsu. JDBC omawiamy szczegółowo w rozdziale 6.

- **Interfejs transakcji Javy** (*Java Transactions API — JTA*). Specyfikacje JTA i JTS umożliwiają komponentom korzystanie ze wsparcia niezawodnego systemu transakcyjnego. Działanie JTA i JTS wyjaśniamy w rozdziale 10.
- **Usługa komunikatów Javy** (*Java Messaging Service — JMS*). JMS umożliwia Twoim rozwiązaniom, zgodnym z J2EE, komunikowanie się w oparciu o przesyłanie wiadomości. Możesz wykorzystywać ten rodzaj komunikacji zarówno wewnątrz swojego systemu, jak i poza nim. Przykładowo, możesz połączyć się z systemem pośredniczącym działającym w oparciu o komunikaty (ang. *Message-Oriented Middleware — MOM*), jak IBM MQSeries czy Microsoft Message Queue (MSMQ). Przesyłanie wiadomości jest rozwiązaniem alternatywnym dla RMI-IIOP i, jak każde tego typu rozwiązanie, ma swoje wady i zalety. Usługę JMS omówimy dokładniej w rozdziale 8.
- **Serwlety Javy**. Serwlety są komponentami sieciowymi, które możemy wykorzystywać do rozszerzania funkcjonalności naszych serwerów WWW. Działanie serwletów opiera się na obsłudze żądań i odpowiedzi, co oznacza, że przechwytują żądanie od pewnego węzła (najczęściej przeglądarki internetowej) i wysyłają do tego węzła odpowiedź. Takie działanie czyni z serwletów doskonałe narzędzie do zadań związanych z generowaniem dynamicznych stron WWW. Serwlety różnią się od komponentów EJB, które oferują wiele nieosiągalnych dla nich możliwości. Serwlety są za to znacznie lepiej przygotowane do obsługi prostych systemów żądanie-odpowiedź i nie wymagają wyszukanych metod zarządzania oferowanych przez serwery aplikacji. Używanie serwletów z komponentami EJB zaprezentujemy w rozdziale 17.
- **Java ServerPages (JSP)**. Technologia stron JSP jest bardzo podobna do serwletów. W rzeczywistości, skrypty JSP są kompilowane do postaci serwletów. Poważną różnicą pomiędzy skryptami JSP a serwletami jest sposób kodowania: strony JSP nie są czystym kodem Javy, koncentruje się raczej na konstrukcji interfejsu użytkownika. Stosowanie skryptów JSP jest szczególnie uzasadnione, jeśli chcesz w swoim rozwiązaniu fizycznie oddzielić łatwy w konserwacji interfejs użytkownika od reszty aplikacji. JSP jest w takich przypadkach idealnym rozwiązaniem, którym można łatwo zarządzać bez konieczności zatrudniania specjalistów od programowania w języku Java (skrypty JSP nie wymagają kompilatora Javy). Sposoby stosowania JSP i EJB prezentujemy w rozdziale 17.
- **Java IDL**. Java IDL jest stworzoną w Javie przez firmę Sun Microsystems implementacją standardu CORBA. Java IDL umożliwia integrację naszych rozwiązań z innymi językami. Pozwala także rozproszonym obiektom uzyskać pełny dostęp do usług standardu CORBA. J2EE jest więc całkowicie zgodne z tym standardem. Integrację naszych rozwiązań z technologią CORBA omówimy szczegółowo w dodatku B.
- **JavaMail**. Usługa JavaMail umożliwia nam wysyłanie listów elektronicznych z naszych napisanych w Javie programów niezależnie od platformy czy protokołu. Przykładowo, w oprogramowaniu serwera napisanym w J2EE możemy, korzystając z usługi JavaMail, potwierdzać zakupy dokonywane

przez Internet na witrynie handlu elektronicznego, za pomocą listu elektronicznego do klienta. Zwróć jednak uwagę na fakt, że usługa JavaMail jest uzależniona od JavaBeans Activation Framework (JAF), co powoduje, że również JAF musi być częścią J2EE. W książce nie będziemy się zajmować usługą JavaMail.

- **Architektura złączy J2EE** (*J2EE Connector Architecture — JCA*). Złącza umożliwiają uzyskiwanie dostępu do istniejących korporacyjnych systemów informacyjnych z poziomu naszego rozwiązania J2EE. Może to dotyczyć dowolnych istniejących rozwiązań, od systemów mainframe z wydajnymi systemami transakcyjnymi (np. CICS firmy IBM czy TUXEDO firmy BEA), przez systemy zarządzania zasobami przedsiębiorstwa (ang. *Enterprise Resource Planning — ERP*), do naszych systemów autorskich. Złącza są przydatne, ponieważ automatycznie zarządzają szczegółami związanymi z nawigacją pomiędzy oprogramowaniem a istniejącymi systemami takimi, jak obsługa transakcji czy problemy bezpieczeństwa. Kolejną zaletą architektury złączy jest możliwość jednorazowego napisania sterownika uzyskiwania dostępu do istniejącego systemu i umieszczania go na serwerze zgodnym z J2EE. Ta właściwość jest istotna, ponieważ wystarczy raz poznać sposób uzyskiwania dostępu do dowolnego istniejącego systemu. Co więcej, taki sterownik piszemy tylko raz i możemy wykorzystywać go wielokrotnie na dowolnym serwerze, co ma szczególne znaczenie dla niezależnych producentów oprogramowania (ISVs), którzy chcą umożliwić dostęp do swojego oprogramowania dla programów działających na serwerach aplikacji. Zamiast pisać sterowniki dostosowane do pojedynczych serwerów, można stworzyć jeden sterownik dla wszystkich. Problem integracji z istniejącymi systemami omówimy szerzej w rozdziałach 12. i 13.

- **API Javy dla przetwarzania XML-a** (*Java API for XML Parsing — JAXP*). Istnieje wiele aplikacji stworzonych w technologii J2EE i wykorzystujących format XML. Przykładowo, będziemy musieli przetwarzać dane w formacie XML, jeśli wykonujemy operacje na linii firma-firma (ang. *Business-to-Business — B2B*), na przykład w przypadku usług internetowych, jeśli próbujemy uzyskać dostęp do istniejących systemów i odwzorowujemy otrzymane z nich dane w formacie XML lub jeśli utrwalamy dokumenty XML w bazie danych. JAXP jest w istocie interfejsem programowym do przetwarzania dokumentów XML w aplikacjach J2EE i jest jedną z implementacji neutralnego interfejsu do analizatorów XML-a. Interfejs JAXP wykorzystuje się zazwyczaj w serwletach, skryptach JSP i komponentów EJB. Na stronach *TheServerSide.com* znajdziesz darmowe materiały opisujące sposoby budowy serwisów internetowych w środowisku J2EE.

- **Usługa identyfikacji i autoryzacji w Javie** (*Java Authentication and Authorization Service — JAAS*). JAAS jest standardowym interfejsem programowym dla operacji związanych z bezpieczeństwem programów napisanych w J2EE. W ogólności, JAAS umożliwia także włączanie całych systemów bezpieczeństwa do naszych rozwiązań. Więcej szczegółów na temat bezpieczeństwa i komponentów EJB znajdziesz w rozdziale 9.

Podsumowanie

Przedstawiliśmy w tym rozdziale bardzo dużo informacji. Najpierw opracowaliśmy listę problemów związanych z tworzeniem dużego, wielowarstwowego systemu. Następnie omówiliśmy komponentową architekturę serwerów, która umożliwia nam pisanie skomplikowanych aplikacji biznesowych bez konieczności zagłębiania się w usługi programów pośredniczących. Zajęliśmy się także standardem EJB i skupiliśmy się na jego szczególnych zaletach. Przeanalizowaliśmy role związane z tworzeniem rozwiązań opartych na komponentach EJB i rozważyliśmy technologie składające się na platformę J2EE.

Mamy dla Ciebie dobrą wiadomość: dopiero zaczynamy poznawanie wielu interesujących i zaawansowanych zagadnień. W następnym rozdziale zajmiemy się kwestiami związanymi z *przechwytywaniem żądań*, które będą miały zasadnicze znaczenie dla późniejszego zrozumienia komponentów EJB. Zaczynajmy!