Zagadnienia konstrukcji oprogramowania komponentowego

Krzysztof Zieliński
Katedra Informatyki, Wydział EAIiE, Akademia Górniczo-Hutnicza
kz@ics.agh.edu.pl

Streszczenie

Rozdział omawia podstawowe problemy konstrukcji, uruchamiania i testowania oprogramowania w technologii komponentowej mającej ważne znaczenie dla tworzenia systemów informatycznych udostępniania usług sieciowych. Na wstępie przedstawiono krótki przegląd środowisk programowych wspierających budowę aplikacji w technologii komponentowej takich jak CCM, EJB, oraz .Net. Mówiono szczegółowo wielowarstwowy model tworzenia tych aplikacji zwracając uwagę na problemy konstrukcji i ograniczenia skalowalności każdej z warstw. W tym aspekcie omówiono użycie typowych wzorców programistycznych oraz dobre praktyki wspierające proces tworzenia tych aplikacji. Odrębną cześć artykułu poświęcono zagadnieniom i konfigurowania i uruchamiania aplikacji, co wiąże się z procesem deklarowania sposobu użycia serwisów zapewnianych przez kontenery serwerów aplikacji. W celu domknięcia całości prezentacji zagadnień budowy aplikacji komponentowych bardzo krótko przedstawiono zagadnienia testowania ich wydajności i profilowania. Omówiono przykładowe środowiska wspierające tą fazę rozwoju oprogramowania.

1. Podstawy programowania komponentowego

Programowanie komponentowe nabiera coraz większego znaczenia w zakresie budowy nowoczesnego oprogramowania. W sytuacji rosnącej złożoności systemów informatycznych, konieczności skrócenia czasu ich budowy, potrzeby klarownej strukturalizacji procesu wytwarzania oprogramowania oraz poniesienia jego jakości komponentowa budowa oprogramowania wydaje się być obecnie najbardziej obiecującym podejściem. Nie oznacza to jednak, że programowanie komponentowe rozwiązuje wszystkie problemy budowy oprogramowania i nie wnosi nowych złożonych zagadnień mających istotny wpływ na jego proces wytwarzania, uruchamiania oraz końcowe własności eksploatacyjne.

Programowanie komponentowe jest wynikiem ewolucji obiektowego podejścia do projektowania i implementacji aplikacji, polegającym na wyposażeniu obiektów aplikacji w predefiniowane usługi zapewniające realizacje szeregu standardowych funkcjonal-

ności jak np.: możliwość zdalnej komunikacji, transakcyjność, bezpieczeństwo, trwałość danych, mechanizmy łączenia, samotestowanie, samoinstalacja, etc. Podejście to charakteryzuje zatem dążenie do odseparowania usług, które można uznać za systemowe od funkcjonalności danej aplikacji. Osiąga się w ten sposób znaczącą redukcję złożoności budowy aplikacji. Wymaga to jednak opracowania interfejsów łączenia funkcjonalność aplikacji z częścią systemową stanowiącą środowisko wykonania komponentów.

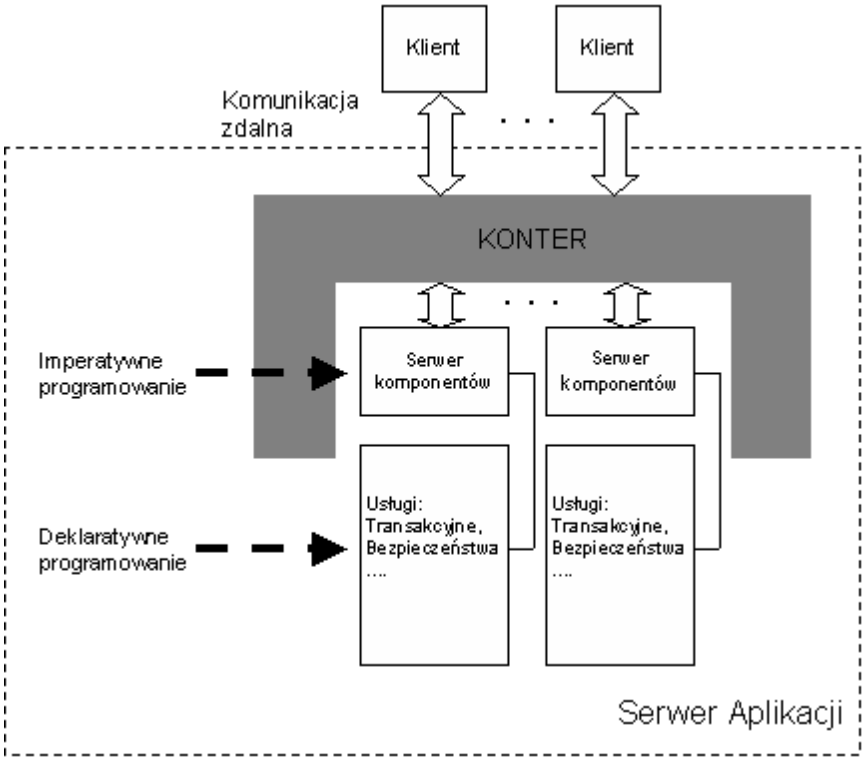
Cechą charakterystyczną środowisk komponentowych jest fakt, że budowa aplikacji obejmuje zarówno imperatywne programowanie, związane z implementacją danej funkcjonalności użytkowej, oraz deklaratywne skonfigurowanie środowiska wykonania zgodnie z wymaganiami aplikacji [ORF1996]. Wobec złożoności usług systemowych faza deklaratywna budowy aplikacji komponentowej może być bardzo trudna i wpływać znacząco na efektywność działania systemu informatycznego.

Współczesne systemy oprogramowania mają zazwyczaj strukturę wielowarstwową złożoną z co najmniej trzech głównych warstw: prezentacji, logiki biznesowej, oraz źródeł danych. Każda z tych warstw wykształciła odpowiednie do pełnionej funkcji środowisko systemowe realizacji przetwarzania. Środowisko systemowe w modelu komponentowym określa się zazwyczaj jako kontener, w którym wykonuje się kod napisany przez użytkownika. Zadaniem kontenera jest wyposażenie obiektów aplikacji w usługi systemowe skonfigurowane zgodnie z deklaratywnym opisem ich użycia. Kontenery wykonują się w ramach serwerów aplikacji stanowiących procesy danego systemu operacyjnego.

Obecne systemy informatyczne są w przeważającej większości systemami rozproszonymi, pracującymi w sieciach komputerowych. Do tej klasy systemów należą w szczególności systemy udostępniania usług w sieci Internet. Systemy rozproszone są znacznie trudniejsze w implementacji od systemów scentralizowanych, a ich środowisko systemowe jest bardzo złożone w implementacji. W tej sytuacji szczególnie duże wymagania stawiane są kontenerom i serwerom aplikacji dedykowanym warstwie logiki biznesowej, która obsługuje stronę serwerową tych systemów. Typowymi ogólnymi wymaganiami stawianymi w tym zakresie są [ORF1996]:

- Przeźroczysta, niezależna od platformy oprogramowania zdalna komunikacja.
- Efektywne i zróżnicowane strategie aktywacji zdalnych komponentów.
- Mechanizmy odszukiwania i tworzenia zdalnych komponentów.
- Procedury utrzymania spójnego trwałego stanu systemu w warunkach realizacji współbieżnych operacji. Wymaganie to sprowadza się do konieczności zapewnienia transakcyjności przetwarzania i współpracy z bazą danych.
- Zapewnienie bezpieczeństwa przetwarzania w sensie autentykacji i autoryzacji wykonania operacji przez klienta. Wymaganie to może być poszerzone o konieczność zapewnienia poufności danych.

Do wymagań tych należy dodać jeszcze wymagania niefunkcjonalne takie jak skalowalność systemu oraz zapewnienie określonej jakości usług (ang. *QoS*). Koncepcję budowy serwera aplikacji przedstawiono na rysunku 1.



Rys. 1. Schematyczna budowa serwera aplikacji

Systemy komponentowe strony serwerowej powstały w wyniku uogólnienia systemów opartych na koncepcji standardu CORBA [ORF1996] oraz usług systemowych opracowanych w ramach architektury OMA zwanych *Common CORBA Services*. Systemy te opracowano na początku lat dziewięćdziesiątych. Po stronie klienta zasadniczy wpływ na ukształtowanie się komponentów warstwy prezentacji posiadały technologie związane z tworzeniem stron WWW oraz budową web serwerów.

2. Przegląd środowisk programowania komponentowego

Środowiska komponentowe wspierające budowę warstwy przetwarzania biznesowego obejmują trzy podstawowe technologie: CCM (*CORBA Component Mode*l), EJB (*Enterprice Java Beans*), oraz COM+ (*Common Component Model*). Środowiska te różnią się wieloma szczegółami, jednakże ich architektura jest bardzo podobna i odpowiada schematowi budowy omówionemu w poprzednim punkcie. Najbardziej dopracowaną i ogólna jest technologia CCM, lecz pozostaje ona etapie specyfikacji i stopień jej rozpowszechnienia jest stosunkowo niewielki. Podstawowe cechy technologii CMM zostały przez nią odziedziczone z dopracowanych i dobrze rozwiniętych systemów opartych na standardzie CORBA [ORF1996].

Tabela 1. Porównanie technologii komponentowych

Cecha	CCM	EJB	.Net
Firma wiodąca	OMG	SUN	Microsoft

Wparcie dla języków programowania	wielu	Java	wielu
System operacyjny	wiele	JVM	Windows
Koncepcja architektury	Kontener & interceptor	Kontener & interceptor	Kontener & interceptor
Kategorie kompomentów	Session, Service, Process, Entity	Session, Entity, Message-Driven	Session, ADO
Aktywacja	Kontener, POA	Kontener	Kontener
Komunikacja	IIOP	IIOP/RMI	DCE/SOAP
Transakcje	OTS	JTS/OTS	MTS
Bezpieczeństwo	CORBA Sec.	Security Pack.	COM+Sec.
Asyn. komunikacja	AMI/TII	JMS	Queued C.
Zdarzenia	Notifcation Service	brak	COM+Events
Usługa katalogowa	CosNaming	JNDI	ADSI
Sposób konfiguracji	XML	XML	XML
Skalowalność	++	+	+

Należą do nich niezależność od języka programowania, bardzo dobrze rozwinięty złożony mechanizm aktywacji i zarządzania przetwarzaniem operacji po stronie serwera w postaci POA (*Portable Object Adapter*), efektywny i sprawdzony serwis transakcyjny OTS (*Object Transaction System*), rozbudowany mechanizm obsługi zdarzeń w postaci serwisu notyfikacji (*Event Notification*). Podstawowe cechy technologii CCM i jej porównanie z innymi platformami komponentowymi przedstawiono w tabeli 1.

Najpopularniejszym środowiskiem programowania komponentowego jest obecnie EJB. Niewątpliwie wynika to ze znaczenia języka Java oraz faktu, że firma SUN dołożyła wielu starań, aby wdrożyć i rozpowszechnić tą technologię. Jest to także związane z rozwojem aplikacji internetowych i takich jak portale, których zasadniczą część stanowią serwery aplikacji pośredniczące w dostępie do warstwy bazodanowej. Technologia EJB jest znacznie łatwiejsza w użyciu niż CMM, a na rynku oprogramowania jest obecnie dziesiątki serwerów aplikacji zgodnych z wersją EJB 2.0, która jest częścią technologii J2EE (*Java 2 Enterprice Edition*). Nowa wersja standardu EJB 2.1 czeka jeszcze na rozpowszechnienie. Podstawowy problem dotyczący EJB to skalowalność, która, zwłaszcza na poziomie dostępu do bazy danych, jest stosunkowo ograniczona. Pewnym rozwiązaniem tego problemu jest klasteryzacja serwerów aplikacji, która polega na uruchomieniu grupy serwerów dzielących proces przetwarzania.

Technologia .NET wspiera budowę szerokiego spektrum aplikacji: od klasycznych monolitycznych, poprzez aplikacje klient serwer, aplikacje trójwarstwowe, skończywszy na aplikacjach wielowarstwowych oraz usługach sieciowych. Architektura aplikacji wielowarstwowych w technologii .NET bazuje na modelu DNA, zdefiniowanym wraz z pojawieniem się modelu COM+ oraz Windows 2000. Architektura DNA to klasyczny trójwarstwowy model budowy oprogramowania. W zakresie organizacji dostępu do baz danych technologia .NET wykorzystuje ADO.NET.

W dalszym ciągu tego rozdziału skoncentrujemy się na budowie aplikacji w technologii J2EE jako najbardziej reprezentatywnej dla współczesnego rozwoju środowisk programowania komponentowego.

3. Warstwowa budowa aplikacji komponentowychi wzorce projektowe

Projektując system w technologii J2EE nie jesteśmy na szczęście skazani na samych siebie. Mamy bowiem do dyspozycji sprawdzone i pewne rozwiązania opisane w postaci wzorców, języków wzorców oraz szkieletów aplikacji [ALU2001, MAR2002, GAM1995].

Wzorzec, (ang. *pattern*) to zwyczajowo przyjęte rozwiązanie typowego, powtarzającego się problemu w danym kontekście.

Podział wzorców opierać można na różnych kryteriach [LAR2002]. Najbardziej popularny podział wzorców nosi nazwę kryterium ogólności. Kryterium to mówi, że wzorce działają na różnych poziomach abstrakcji – od pojedynczej klasy do poziomu systemu jako całości i dzieli je na:

- wzorce architektoniczne (ang. *architectural patterns*) są to wzorce wysokiego poziomu określające strukturę i zachowanie systemu jako całości,
- wzorce projektowe (ang. *design patterns*) to wzorce pośredniego poziomu, w ogólności określające strukturę i zachowanie komponentów oraz zestawów klas systemu.
- wzorce programowania, czyli idiomy (ang. *programming patterns, idioms*) są to wzorce niskiego poziomu podsuwające rozwiązania dla konkretnych problemów implementacyjnych.

Innym kryterium podziału wzorców jest kryterium technologii, w której są wykorzystywane. Stosując taki podział wyróżnia się wzorce:

- wzorce CORBA,
- wzorce EJB,
- wzorce J2EE,
- wzorce Struts,
- inne.

Projektanci i programiści tworzący system informatyczny muszą zrozumieć nie tylko pojedyncze wzorce mające zastosowanie w jego budowie. Potrzebują również wiedzy

na temat ich wzajemnych zależności, sposobów współpracy, metod budowy większych rozwiązań tworzonych z wykorzystaniem wielu połączonych i współpracujących ze sobą wzorców. Rozwiązaniem tego problemu jest zastosowanie języka wzorców. Język wzorców (ang. *pattern language*) opisuje relacje pomiędzy wzorcami oraz sposoby ich współpracy w ramach tworzenia złożonego rozwiązania projektowego [ALU2001, MAR2002].

Tematem dalszych rozważań będą tylko wzorce EJB oraz J2EE.

3.1. Architektura logiczna systemu J2EE

Jedną z pierwszych faz projektowania rozproszonej aplikacji internetowej jest zdefiniowanie jej architektury. Przy jej tworzeniu stosować należy wzorce architektoniczne określające strukturę i zachowanie systemu jako całości [LAR2002]. Najbardziej popularne wzorce architektoniczne to wzorce rodziny POSA, które dzielą się na cztery kategorie:

- Wzorce porządkujące system (ang. *From Mud to Structure Patterns*). Zadaniem wzorców tej kategorii jest podzielenie całości złożonego systemu na mniejsze, współpracujące ze sobą jednostki. Przedstawicielami tej kategorii są wzorce Layers, Tiers, Pipes and Filters oraz Blackboard.
- Wzorce rozproszenia systemu (ang. *Distributed Systems Patterns*)
- Wzorce tej grupy odpowiadają za architekturę komunikacyjną aplikacji rozproszonych. Do kategorii tej należy wzorzec Broker.
- Wzorce interakcji systemu (ang. *Interactive Systems Patterns*)
- Wzorce strukturalizują system w taki sposób, aby umożliwić jego użytkownikowi pozostawanie z nim w bezustannej interakcji. Do kategorii tej należą wzorce Model View Controller (MVC) i Presentation Abstraction Control (PAC).
- Wzorce przystosowania systemu (ang. *Adaptable Systems Patterns*)
- Wzorce sugerują takie rozwiązania w budowie systemu, które umożliwią w przyszłości jego łatwe rozszerzanie, modyfikowanie oraz przystosowywanie do nowych technologii. Do kategorii tej należą wzorce Reflection i Microkernel.

Z punktu widzenia aplikacji budowanej w oparciu o technologię J2EE najistotniejsze wydają się wzorce Layers, Tiers oraz MVC.

Najczęściej stosowanym wzorcem architektonicznym jest wzorzec Layers, który opisuje podział złożonego systemu na warstwy logiczne, zwany podziałem pionowym systemu (ang. *vertical approach*). W ramach każdej z warstw logicznych systemu możliwy jest również dodatkowy podział systemu na podsystemy, zwany podziałem poziomym systemu (ang. *horizontal approach*).

Liczba i rodzaj warstw, na które dzieli się system jest zależna od domeny zastosowań systemu. Aplikacje budowane w oparciu o technologię J2EE (a właściwie to niemal każda aplikacja rozproszona) składają się z pięciu warstw logicznych [ALU2001, MAR2002, WWW2002]. Są nimi mianowicie:

- Warstwa prezentacji (ang. *presentation layer)* – warstwa interfejsu graficznego

użytkownika. W przypadku aplikacji tworzonej w technologii J2EE stanowią ją wszystkie technologie budowy interfejsu graficznego aplikacji internetowej takie jak HTML, JSP, Flash.

- Warstwa aplikacji (ang. *application layer*) – obsługuje przepływ żądań i sterowania między elementami warstwy interfejsu graficznego użytkownika a warstwą biznesową aplikacji. Warstwa ta odpowiedzialna jest za utrzymywanie stanu sesji klienta, dokonywanie walidacji syntaktycznej wykonania logiki aplikacji.
- Warstwa biznesowa (ang. *business layer*) – nazywana jest również warstwą usług (ang. *services layer*) oraz warstwą procesów i przepływu sterowania (ang. *process and workflow layer*). Stanowi ona miejsce dostępu do logiki biznesowej systemu. Warstwa aplikacji chcąc zrealizować pewien przypadek użycia wywołuje metody biznesowe zawarte w obiektach tej właśnie warstwy. W przypadku aplikacji tworzonej w technologii J2EE warstwa biznesowa zaimplementowana jest zgodnie ze wzorcem projektowym Session Facade/Message Facade. Stanowią ją więc komponenty session bean oraz message-driven bean. Do głównych zadań tej warstwy należy wykonywanie logiki biznesowej aplikacji poprzez odwoływanie się do obiektów warstwy domeny systemu, kontrolowanie przebiegu transakcji oraz zarządzanie bezpieczeństwem wykonywanych przetwarzań biznesowych.
- Warstwa domeny (ang. *domain layer*) – jest miejscem, w którym rezydują obiekty domeny systemu, zidentyfikowane podczas etapu jego analizy obiektowej. W przypadku aplikacji opartej o technologię J2EE warstwę tą stanowią komponenty entity bean. Warstwa biznesowa realizuje swoją funkcję dostawcy logiki biznesowej i danych dla strony klienta poprzez odwoływanie się do obiektów warstwy domeny. Warstwa domeny jest najczęściej na tyle ogólna, że stanowi zbiór obiektów niezależnych od wykorzystującej ją aplikacji i w związku z tym może zostać ponownie użyta przy tworzeniu innych systemów.
- Warstwa integracji (ang. *integration layer*) – nazywana jest również warstwą źródła danych (ang. *data source layer*), warstwą dostępu do danych (ang. *data access layer*) lub po prostu warstwą danych (ang. *data layer*). Zawiera ona skomplikowaną logikę pozwalającą na pobieranie i umieszczanie obiektów domeny w bazie danych. Jeśli aplikacja tworzona w technologii J2EE, w warstwie domeny, wykorzystuje komponenty CMP (*Container Managed Persistence*) entity bean, wówczas programista nie musi martwić się o implementację tej warstwy systemu. Zadba o to w jego imieniu kontener EJB. Jeśli natomiast warstwa domeny systemu opartego o technologię J2EE bazuje na komponentach BMP (*Bean Managed Persitence*) entity bean, wówczas programista sam musi stworzyć kod tej warstwy wykorzystując interfejs programistyczny JDBC oraz wzorce projektowe DAO (*Data Access Object*) lub DACB (*Data Access Command Bean*).
- Warstwa podstawowa (ang. *foundation layer*) – nazywana jest również warstwą techniczną (ang. *technical layer*), warstwą infrastruktury (ang. *infrastructure layer*) lub warstwą wspólną (ang. *common layer*). W jej skład wchodzi kod obsługi błędów systemu, jego konfiguracji, logowania pracy, a także klasy pomocnicze (ang. *utils*) systemu. Wszystkie te serwisy używane są w całej aplikacji przekraczając granice warstw logicznych. Dlatego też bardzo często klasy te nazywane są kodem niezależ-

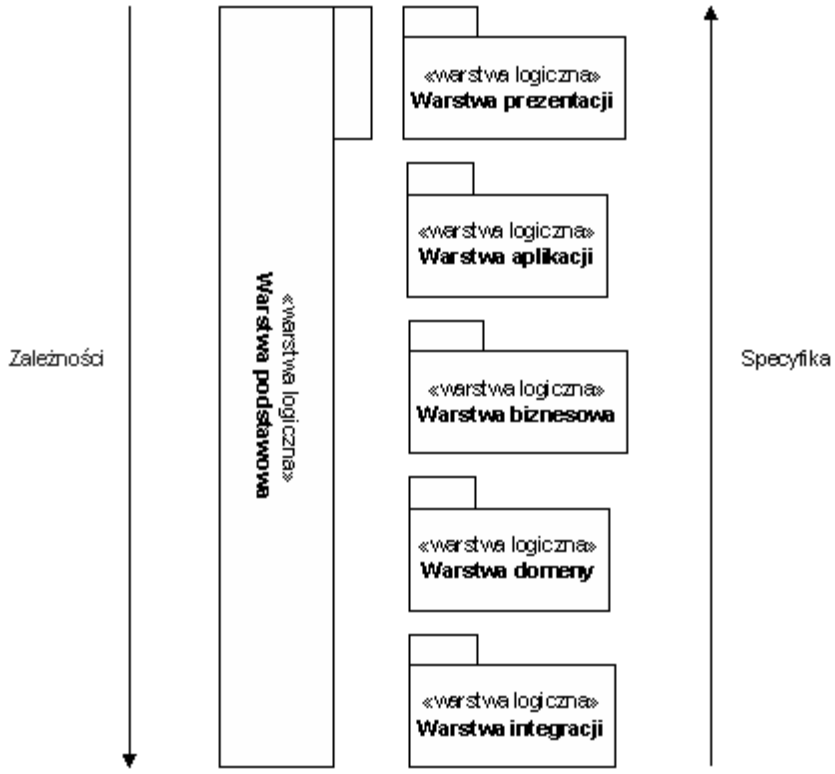
nym od warstw.

Warstwy aplikacji i prezentacji noszą wspólną nazwę warstwy interfejsu użytkownika (ang. *UI layer)* lub warstwy klienta (ang. *client łyaer*). Model warstwowy aplikacji J2EE przedstawiono na rysunku 2.

Z użyciem wzorca Layers wiążą się dwie przeciwstawne zasady. Pierwsza z nich mówi o tym, iż każdą funkcjonalność może na dodać przez wprowadzenie dodatkowej warstwy pośredniej. Druga stwierdza natomiast, że każda nowa warstwa wprowadzona do systemu obniża szybkość i sprawność jego pracy. Prawdziwa sztuka polega na właściwym wyważeniu obu czynników i zaprojektowaniu systemu o akceptowalnej funkcjonalności i wydajności.

3.2. Architektura fizyczna systemu J2EE

Architektura fizyczna (ang. *physical architecture*) dzieli system na warstwy fizyczne. Opisują one fizyczne środowisko uruchamiania i pracy systemu,

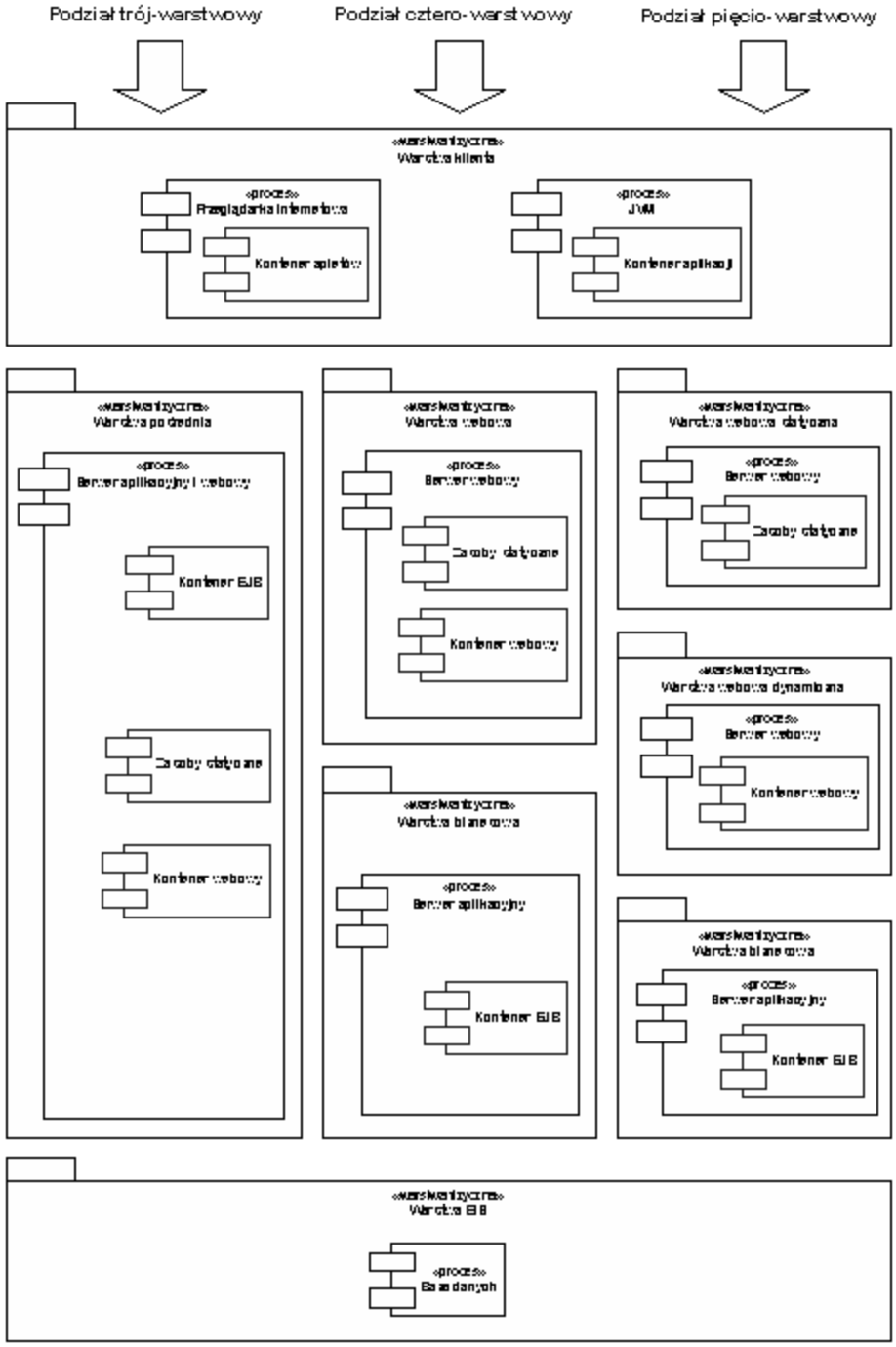


Rys. 2. Podział aplikacji opartej na technologii J2EE na warstwy

przedstawiając jego różne konfiguracje. Środowisko takie składa się z węzłów fizycznych, komunikujących się ze sobą za pomocą różnych protokołów transmisyjnych. Architektura fizyczna określa także sposób odwzorowywania oprogramowania potrzebnego do uruchomienia systemu, składającego się z procesów czyli węzłów logicznych, na węzły fizyczne. Opisuje ona również rozmieszczenie (ang. *Deployment*) aplikacji stanowiącej system na węzłach logicznych. Architektura ta budowana jest w oparciu o wymagania niefunkcjonalne systemu i bazuje na wzorcu architektonicznym Tiers. W

technologii J2EE można wyróżnić następujące warstwy fizyczne:

- Warstwa klienta (ang. *client tier*). Warstwę klienta stanowią dwa rodzaje kontenerów J2EE:
  - Przeglądarka internetowa z kontenerem apletów. Specyfikacja J2EE przewiduje kontenery, w których mogą być uruchamiane komponenty zwane apletami. Aplety najczęściej stosowane są do tworzenia zaawansowanego graficznego interfejsu użytkownika na stronach WWW. Aplety pobierane są z serwera, a następnie uruchomiane na maszynie klienta wewnątrz przeglądarki internetowej. Najczęściej kontener apletów jest częścią przeglądarki internetowej.
  - Kontener aplikacji. Typowo jest to J2SE (Java 2 Standard Edition) dostarczająca kontenera zdolnego do uruchamiania aplikacji, napisanych w języku Java, wykorzystujących takie interfejsy programistyczne jak Swing czy AWT.



Rys. 3. Architektura trój-, cztero- oraz pięcio-warstwowa aplikacji opartej o technologię J2EE

- Warstwa pośrednia (ang. *middle tier*) nazywana również warstwą serwera (ang. *server tier*). Warstwę tę obsługują:
  - serwery aplikacyjne (serwery J2EE) oferujące kontener EJB, w którym pracują komponenty EJB zawierające logikę biznesową aplikacji oraz zarządzające danymi.

- serwery webowe zaopatrzone w kontener webowy, stanowiący środowisko zarządzania komponentami servlet oraz JSP odpowiadającymi za dostarczanie dynamicznej zawartości klientom. Serwer webowy pracuje również w charakterze dostawcy dokumentów statycznych (strony HTML, pliki graficzne, itp.).

Możliwe jest również rozbicie warstwy pośredniej na dwie lub trzy podwarstwy. Warstwy te naszą wówczas nazwy odpowiednio warstwy biznesowej (ang. *business tier*) oraz statycznej i dynamicznej warstwy webowej (ang. *static, dynamic web tier*). W ten sposób otrzymuje się architekturę cztero- lub pięciowarstwową. Rysunek 3. przedstawia architekturę trój-, cztero- oraz pięcio-warstwową z uwzględnieniem różnych typów kontenerów rozmieszczonych w różnych warstwach fizycznych.

- Warstwa EIS (ang. *enterprise information systems tier*) nazywana również warstwą informacyjną (ang. *information tier*) lub warstwą zasobów (ang. *resource tier*). Warstwa ta najczęściej realizowana jest przez serwery baz danych oraz inne systemy EIS.

3.3. Wzorce projektowe aplikacji J2EE

Wzorce projektowe, to wzorce pośredniego poziomu abstrakcji, których działanie obejmuje klasy, obiekty oraz komponenty tworzonego systemu. Wśród wzorców projektowych wyróżniamy wzorce ogólne (niezależne od technologii), które znajdują zastosowanie w budowie każdego dobrze zaprojektowanego systemu informatycznego oraz wzorce specyficzne dla technologii zastosowanej przy tworzeniu aplikacji. Tą pierwszą grupę wzorców projektowych stanowią powszechnie znane wzorce GoF (*Gang of Four*), drugą np. wzorce projektowe technologii J2EE [ALU2001, MAR2002, GAM1995]. Przy czym wzorce specyficzne dla technologii w swoim działaniu opierają się i wykorzystują wiele wzorców ogólnych GoF.

Wzorce GoF, ze względu na ich przeznaczenie, zostały podzielone na trzy grupy:

- Wzorce tworzenia (ang. creational design patterns). Wzorce tej grupy opisują proces tworzenia obiektów. Należą tutaj np. wzorce Factory Method, Abstract Factory, Singleton.
- Wzorce strukturalne (ang. structural patterns). Opisują one sposoby kompozycji klas i obiektów celem tworzenia większych struktur. Przykładami wzorców tej grupy są Adapter, Facade, Proxy.
- Wzorce zachowania (ang. behavioral patterns). Stanowią one opis interakcji i współpracy obiektów i klas. Zaliczamy tutaj np. wzorce Command, State, Iterator.

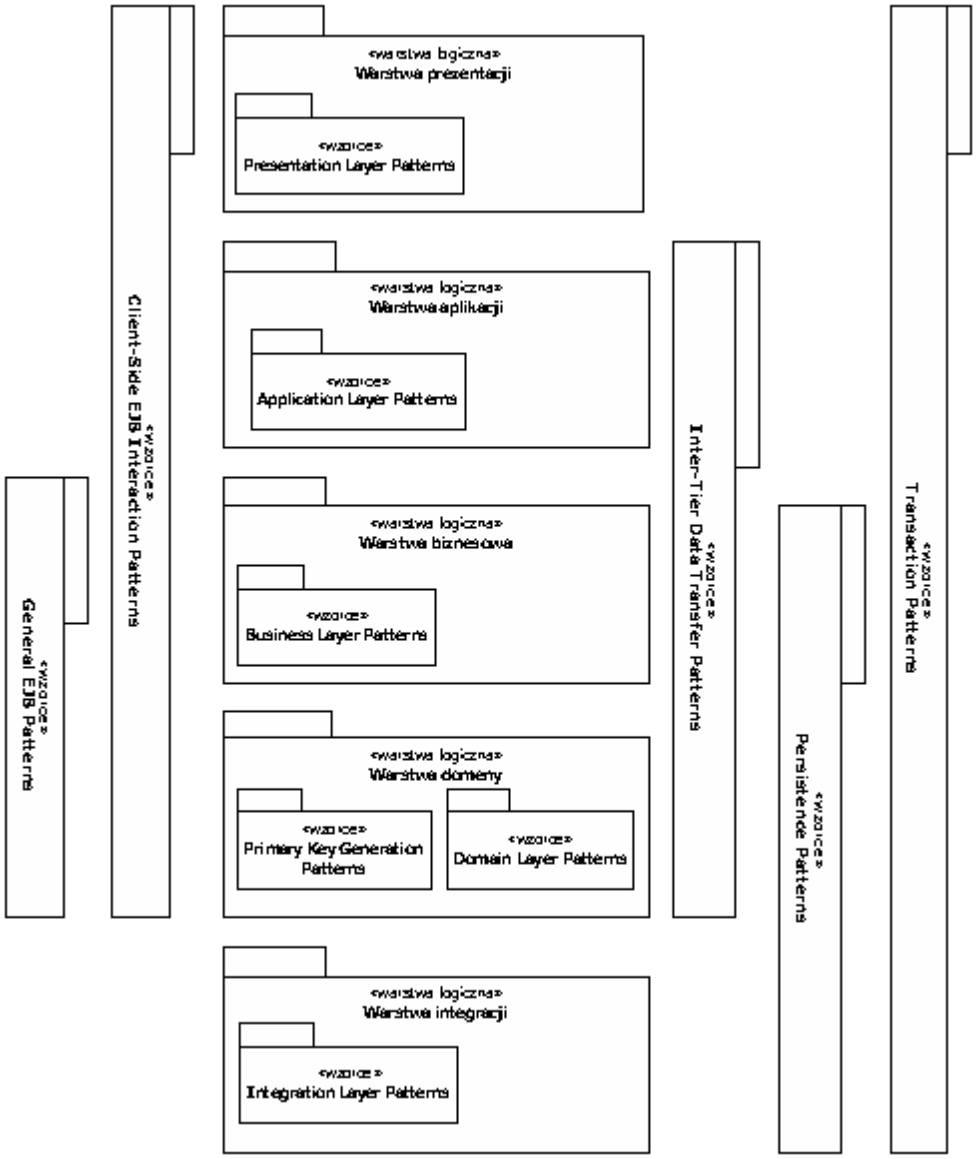
Podział wzorców projektowych technologii J2EE oraz ich umiejscowienie w ramach warstw logicznych systemu opartego o tę technologię przedstawia rysunek 4.

Wzorce technologii J2EE możemy podzielić na następujące grupy:

- General EJB. Pracują w warstwach Business Layer oraz Domain Layer aplikacji J2EE. Są to ogólne wzorce znajdujące zastosowanie w budowie komponentów EJB. Zaliczamy do nich wzorce Business Interface oraz EJB Layer Supertype.
- Client-Side EJB Interaction. Wzorce projektowe zgrupowane w tej kategorii pracują

w warstwach Application, Business oraz Domain Layer aplikacji opartej o technologię J2EE. Odpowiadają na pytanie jak należy budować kod klienta, aby jego komunikacja z komponentami EJB była wydajna a sam kod aplikacji przejrzysty oraz łatwy w utrzymaniu i rozwoju. Zaliczamy do nich wzorce Service Locator oraz Business Delegate.

- Inter-Tier Data Transfer. Wzorce tej grupy odpowiadają na pytanie, w jaki sposób należy wymieniać dane, w sposób efektywny oraz wydajny, pomiędzy stronami serwera i klienta aplikacji opartej o technologię J2EE. Zaliczamy do nich wzorce Data Transfer Object oraz Data Transfer Object Factory.
- Transaction. Wzorce tej grupy pracują w każdej z warstw aplikacji opartej o technologię J2EE. Mają za zadanie zarządzać transakcjami aplikacji. Ich przedstawicielem jest wzorzec projektowy Version Number.
- Primary Key Generation. Wzorce tej grupy odpowiedzialne są za generowania kluczy podstawowych. Zaliczamy do nich wzorce projektowe: Sequence Blocks, UUID for EJB Pattern oraz Stored Procedures for Autogenerated Keys.
- Presentation Layer. Wzorce te wykorzystywane są do tworzenia graficznego interfejsu użytkownika. Zaliczamy do nich wzorce projektowe Composite View oraz View Helper.
- Application Layer. Wzorce tej grupy odpowiadają na pytanie, w jaki sposób realizować obsługę żądań przychodzących od klienta oraz przepływ sterowania w aplikacji. Odpowiadają one również za delegowanie żądań do warstwy biznesowej. Zaliczamy do nich Front Controller, Intercepting Filter oraz Business Delegate.



Rys. 4. Wzorce projektowe J2EE a warstwy logiczne aplikacji opartej o technologię J2EE

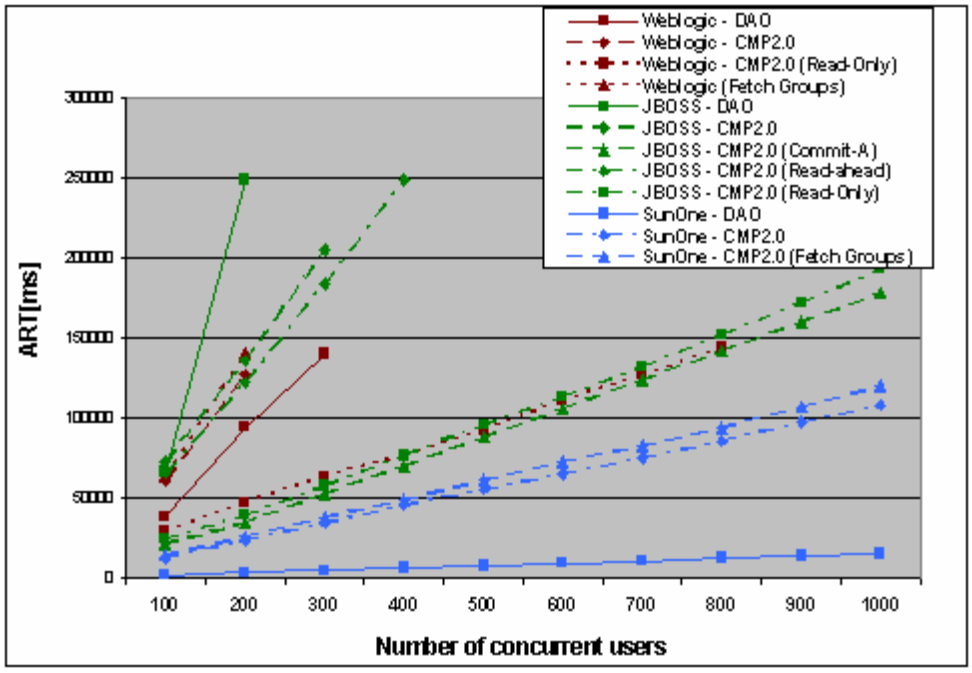
- Business Layer. Wzorce tej grupy odpowiadają za udostępnianie logiki biznesowej związanej z przypadkami użycia aplikacji. Ich głównymi przedstawicielami są wzorce projektowe Session Facade oraz Message Facade.
- Domain Layer. Wzorce te rozwiązują problem komunikacji z warstwą biznesową. Pomagają również w identyfikowaniu logiki domenowej aplikacji. Zaliczamy do nich wzorce projektowe Local Entity Bean oraz Data-Oriented Entity Bean.
- Integration Layer. Wzorce tej grupy odpowiadają za realizację wydajnego oraz niezawodnego mechanizmu obsługi interakcji ze źródłami danych. Zaliczamy do nich, między innymi, wzorzec Connection Pool.

4. Konfigurowanie i uruchamianie aplikacji komponentowych

Użycie wzorców programistycznych jest zaledwie warunkiem wstępnym poprawności budowy aplikacji i nie zapewnia osiągnięcia wymaganej efektywności działania systemu w sensie gwarantowanego maksymalnego czasu odpowiedzi systemu czy liczby przetwarzanych transakcji w jednostce czasu. Zależy to od wielu czynników, do których należy także wybór i sposób skonfigurowania serwera aplikacji. W celu skonfigurowania serwera aplikacji wykorzystuje się programowanie deklaratywne. Użytkownik ma do dyspozycji opracowany przez producenta zbiór parametrów danego serwera aplikacji, który może skonfigurować tworząc plik konfiguracyjny mający zazwyczaj postać pliku XML. Liczba parametrów, jakie można skonfigurować jest zazwyczaj bardzo duża i jest to zadanie wymagające sporej wiedzy.

Dalsze rozważania ograniczono do serwera aplikacji EJB. W przypadku tej technologii użytkownik stoi przed wyborem między innymi: rodzaju organizacji współpracy z bazą danych BMP lub CMP, sposobu obsługi transakcji BMT lub CMT, poziomu izolacji transakcji oraz warunków ich tworzenia, wielkości puli beanów sesyjnych oraz entity beanów, wyboru interfejsów lokalnych lub zdalnych, sposobu użycia mechanizmów autoryzacji. Oprócz decyzji sprawdzających się do zadeklarowania określonych wartości parametrów, twórca oprogramowania może rozważać wybór sposobu realizacji określonej warstwy systemu. Przykładowo entity beany w aplikacji EJB stosowane do budowy warstwy integracji są często postrzegane jako element ograniczający skalowalność systemu. Jedną z alternatyw jest zastosowanie do budowy tej warstwy technologii DAO (*Data Access Object*). Omawiane zagadnienia były przedmiotem szczegółowych badań [JAR2003]. Przykładowe wyniki zamieszczono na rysunku 5.

Do badań wybrano trzy najpopularniejsze serwery aplikacji: WebLogic, SunOne 7.0, oraz JBOSS. Przedmiotem badań była efektywność realizacji warstwy integracji systemu. Badano średni czas odpowiedzi systemu w funkcji liczby użytkowników wykonujących współbieżnie operacje odczytu danych ich zapisu oraz usuwania.



Rys. 5. Średni czas wykonania operacji odczytu

Przeprowadzone testy należały do kategorii obciążeniowych [ZAD2002] i zakładały jednoczesne zgłoszenie się określonej liczby użytkowników systemu.

Zamieszczony przykład uwidacznia także istotny fakt, że każdy z rozważanych serwerów aplikacji posiada swoje charakterystyczne opcje konfiguracyjne. Serwery SunOne oraz Weblogic posiadają opcje Fetch Groups polegającą na możliwości załadowania z bazy danych w jednej operacji grupy danych. Serwery Weblogic i JBOSS mają opcje Read-ahead, czyli czytania danych z bazy danych z wyprzedzeniem. Opcji tej nie posiada SunOne. Opcja Commit-A jest dostępna tylko w serwerze JBOSS i polega na przechowywaniu w pamięci podręcznej pomiędzy transakcjami stanu entity beanów. Opcja ta sprawdza się w sytuacji, gdy kontener jest jedynym użytkownikiem bazy danych. Zamieszczone na rysunku 5. wyniki dobrze ilustrują wpływ poszczególnych opcji i zastosowanych rozwiązań na średni czas realizacji odczytu.

Testowanie aplikacji komponentowych [ZAD2002, WIL2000, SHI2000] stanowi osobny trudny problem techniczny. Testy te można podzielić na: testy poprawności funkcjonalnej, obciążeniowe testy wydajnościowe, oraz profilowanie, czyli określanie jak rozkłada się czas wykonania w poszczególnych modułach, czy warstwach aplikacji. Obecnie istnieje cała gama dostępnych narzędzi dla realizacji tych testów dostępnych bezpłatnie, jak np.: Ginder, JProbe, czy WebLoad. Do najbardziej znanych producentów tego typu narzędzi należą: Mercury Interactive, Rational, Compuware, Empirix, Seque i RadView. Większość z tych narzędzi pozwala na zebranie obciążenia z pracującej aplikacji a następnie na wygenerowanie na tej podstawie odpowiedniego obciążenia testowego. Pozwala to na realizacje testowania w warunkach zbliżonych do tych, które występują w trakcie normalnej eksploatacji systemu.

5. Podsumowanie

Budowa oprogramowania komponentowego stanowi kolejny etap w dążeniu do automatyzacji procesu wytwarzania oprogramowania. Podejście to w porównaniu z programowaniem obiektowym wnosi dalszą strukturalizację kodu i eliminuje w dużym stopniu potrzebę implementacji funkcji systemowych przez programistę.

Wzorce programowe pomagają rozwiązać typowe problemy programistyczne i uniknąć rozwiązań mogących prowadzić do powstania nieefektywnego kodu. Jednakże samo ich użycie nie gwarantuje, że powstała aplikacji będzie spełniać wymagania efektywnościowe.

Aplikacje komponentowe są złożonymi wielowarstwowymi systemami. Ich testowanie wymaga specjalistycznego oprogramowania i jest ważne z punktu widzenia oceny przydatności funkcjonalnej aplikacji dla końcowego użytkownika, poprawności jej skonfigurowania, a także oszacowania potrzebnych zasobów sprzętowych dla jego eksploatacji w różnych warunkach obciążenia

Bibliografia

[ALU2001] D. Alur, D. Crupi i D. Malks, *Core J2EE Patterns. Best Practices and Design Strategies*, Prentice Hall, 2001.

[BOO2001] G. Booch, G. Rumbaugh i G. Jacobson, *UML przewodnik użytkownika*, Wydawnictwo Naukowo-Techniczne, 2001.

[GAB2002] K. Gabrick i K. Weiss, *J2EE and XML Development*, Manning Publications Co., 2002.

[GAM1995] E. Gamma, E. Helm, E. Johnson i E. Vlissides, *Design Patterns. Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[BEC2001] K. Beck, *Extreme Programming Explained.*, Addison-Wesley, 2000.

[JAR2003] M. Jastrząb i M. Zieliński, *Database Access with EJB Application Servers Performance Study*, Submitted for publication 2003.

[JEF2001] R. Jeffries, R. Anderson i R. Hendrickson, *Extreme Programming Installed*, Addison-Wesley, 2001.

[FLA1999] D. Flanagan, D. Farley, D. Crawford i D. Magnusson, *Java Enterprise in a Nutshell. First Edition*, O'Reilly, 1999.

[LAR2002] C. Larman, *Applying UML and Patterns. An Introduction to Object-Oriented Analysis and Design and the Unified Process*, Prentice Hall, Inc., 2002.

[MAR2002] F. Marinescu, *EJB Design Patterns. Advanced Patterns, Processes and Idioms*, John Wiley & Sons, Inc., 2002.

[MON2001] R. Monson-Haefel, *Enterprise JavaBeans. Third Edition*, O'Reilly, 2001.

[ORF1996] R. Orfali, R. Harkey i R. Edwards, *The Essential Distributed Objects Survival Guide*, John Willey, 1996.

[SHI2000] J. Shirazi, *Java Performance Tuning*, O'Reilly, 2000.

[ROM2002] E. Roman, E. Ambler i E. Jewell, *Enterprise JavaBeans. Second Edition*, John Wiley & Sons, Inc., 2002.

[TAT2002] B. Tate, *Bitter Java*, Manning Publications Co., 2002.

[WIL2000] S. Wilson i S. Kesselman, *Java Platform Performance, Strategies and Tactics*, Addison Wesley, 2000.

[WWW2003] *www.precisejava.com*.

[WWW2003a] *www.onjava.com*.

[WWW2002] *www.theserverside.com*.

[WWW2002a] *jakarta.apache.org*.

[ZAD2002] P. Zadrozny, *Performance Testing*, Expert Press, 2002.