Partner:

WSPÓŁCZESNE TECHNOLOGIE INFORMATYCZNE

TECHNOLOGIE MDE

W PROJEKTOWANIU APLIKACJI INTERNETOWYCH

Projekt Absolwent na miarę czasu współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego
Nr umowy UDA-POKL.04.01.01-00-421/10-01

Politechnika Lubelska
Wydział Elektrotechniki i Informatyki
ul. Nadbystrzycka 38A
20-618 Lublin

WSPÓŁCZESNE TECHNOLOGIE INFORMATYCZNE

TECHNOLOGIE MDE

W PROJEKTOWANIU APLIKACJI INTERNETOWYCH

Jacek Kęsik

Kamil Żyła

Politechnika Lubelska
Lublin 2011

Recenzenci:
dr inż. Marek Miłosz
dr inż. Piotr Kopniak

Skład komputerowy: Jacek Kęsik

Publikacja finansowana z projektu „Absolwent na miarę czasu”

Projekt „Absolwent na miarę czasu” współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego. Nr umowy UDA-POKL.04.01.01-00-421/10-01

Ta publikacja odzwierciedla jedynie stanowiska jej autorów, a Komisja Europejska nie ponosi odpowiedzialności za informacje w niej zawarte

Publikacja dystrybuowana bezpłatnie

Publikacja wydana za zgodą Rektora Politechniki Lubelskiej

ISBN: 978-83-62596-62-1

Wydawca: Politechnika Lubelska
ul. Nadbystrzycka 38D, 20-618 Lublin
Realizacja: Biblioteka Politechniki Lubelskiej
Ośrodek ds. Wydawnictw i Biblioteki Cyfrowej
ul. Nadbystrzycka 36A, 20-618 Lublin
tel. (81) 538-46-59, email: wydawca@pollub.pl
www.biblioteka.pollub.pl
Druk: ESUS Agencja Reklamowo-Wydawnicza Tomasz Przybylak
www.esus.pl

Elektroniczna wersja książki dostępna w Bibliotece Cyfrowej PL www.bc.pollub.pl
Nakad: 100 egz.

Spis treści

WSTĘP

Lata 90-te były zdominowane przez wykorzystanie aplikacji nie wymagających dostępu do sieci komputerowych. Edytory tekstu, narzędzia planistyczne ukierunkowano na przechowywanie danych i operowanie na nich w ramach komputera (hosta), na którym zostały zainstalowane. Bastionem oprogramowania rozproszonego były duże centra obliczeniowe usytuowane w ośrodkach badawczych i rządowych oraz bliższe statystycznemu Polakowi usługi świadczone przez firmy hostingowe, czyli dostęp do dynamicznych i statycznych stron internetowych oraz serwerów pocztowych. Sytuacja taka ukształtowała sięgłównie ze względuna ograniczoną dostępność Internetu, parametry i reguły dostępu do łącza internetowego, relatywnie niewielki wachlarz urządzeń dostępowych oraz nierozwiniętą świadomość potencjalnych użytkowników o możliwościach tego medium.

Współcześnie oprogramowanie działające w środowisku rozproszonym przeżywa prawdziwy renesans. Zasoby i infrastruktura Internetu służą odtwarzaniu i przechowywaniu materiałów audio-video, prezentowaniu i grupowaniu zdjęć, materiałów graficznych, filmów i muzyki. Powstają radia internetowe, a na popularności zyskuje usługa VoD (*ang. Video on Demand*) pozwalająca na odtwarzanie treści audio-wizualnych na żądanie. Wiodące firmy sektora IT oferują możliwość składowania dowolnych plików w przestrzeni dyskowej swoich serwerów,

co jest wykorzystywane do tworzenia kopii zapasowych plików, a nawet do tworzenia repozytoriów wiedzy i materiałów współdzielonych pomiędzy wieloma użytkownikami globalnej sieci.

Wzrasta znaczenie chmury i przetwarzania w chmurze (*ang. cloud-computing*), a co za tym idzie następuje przeniesienie działań, do tej pory wykonywanych na lokalnym komputerze, na zbiór serwerów, których lokalizacja nie jest istotna dla użytkownika z punktu widzenia świadczonych usług. Jej nieocenioną zaletą jest transparentne dla użytkownika zapewnienie bezpieczeństwa danych, a także ciągłościświadczenia usług oraz elastyczności co do miejsca i metody dostępu do nich.

Nawetnarzędzia programistyczne i pakiety biurowe są udostępniane w wersjach internetowych. Do pierwszej z grup można zaliczyć AppInventor, który służy do budowania aplikacji mobilnych z gotowych elementów na platformę Android.Do drugiej grupy należy Google docs, które aspiruje do statusu pełnoprawnego pakietu biurowego, oferując m.in. edytor tekstu, arkusz kalkulacyjny oraz narzędzie tworzenia prezentacji multimedialnych. Dodatkowo w wielu przypadkach zaimplementowano współbieżny dostęp do dokumentów orazmożliwość pracy (edycji, pobierania, itp.) w czasie rzeczywistym przez wiele osób.

Aplikacje internetowe znalazły profesjonalne zastosowania w kartografii, handlu, organizacji przepływu dokumentów oraz pracy organów państwowych. Dzisiejsze mapy internetowe nie tylko pozwalają sprawdzić lokalizację obiektu, ale oferują dostęp do dodatkowych informacji związanych z terenem – zdjęć lotniczych, informacji o natężeniu ruchu, informacji topograficznych, informacji o hot elach, gastronomii i innych usługach oraz położeniu wybranych osób (zazwyczaj wymaga to zgody tych osób lub nakazu sądowego). Geodeci mają możliwość oznaczania działek i umieszczania pomiarów w odpowiednich rejestrach na podstawie wskazań GPS lub ręcznie wprowadzonych współrzędnych. Rolnicy korzystając z kartografii internetowej mogą przypisywać informacje o typie zasiewów, oprysku, szkodach, sposobie nawożenia przy pomocy komputera osobistego lub telefonu komórkowego z modułem GPS. Pośród stron internetowych jest pełno ofert handlowych iporównywarek cen, które oprócz danych technicznych produktów udostępniają informacje o sklepach, ich stanach magazynowych oraz opiniach nabywców. Większość banków pozwala swoim

klientom zarządzać kontami bankowymi, wykonywać samodzielnie płatności, a nawet grać na giełdzie za pośrednictwem internetowych biur maklerskich.Potęgi globalnej sieci są świadomerównież organy rządowe, które korzystają z niej m.in. w celu publikacji dokumentów, komunikacji z obywatelami, kontroli obywateli (weryfikacji danych i uprawnień) oraz egzekwowania prawa. Wreszcie bogactwo kursów e-learningowych, tworzonych przez pasjonatów, uczelnie wyższe, szkoły oraz organy samorządowe, pozwala na podnoszenie kwalifikacji osobistych i zawodowych bez wychodzenia z domu.

W Internecie można kupić książki, muzykę, filmy, odnaleźć artykuły naukowe, narzędzia programistyczne, darmowe gry, wszelkiego rodzaju dokumentację oraz podręczniki użytkownika, a także skorzystać ze słowników iencyklopedii. Wszystko to wpływa również na modele biznesowe firm, które muszą dostosowywać się do dynamicznie zmieniającego się rynku i pojawiających się licencji.

Oblicze Internetu i świata kształtuje również ogromna siła i popularność serwisów społecznościowych, a także ich obecność w mediach publicznych. Wpływ tego trendu jest na tyle silny, że pojawia się w komercyjnych narzędziach do profesjonalnych zastosowań, np. w projektowaniu procesów biznesowych oraz tworzeniu oprogramowania. Niezliczona liczba blogów i prywatnych stron internetowych służy do głoszenia różnych przekonań, natomiastdzienniki w swoich serwisach integrują mechanizmy wymiany opinii ikomentarzy oraz wspierają dziennikarstwo obywatelskie. Serwisy społecznościowe służą nie tylko do wymiany zdjęć, poglądów i odnajdywania znajomych, ale również są poważnymi narzędziami w rękach policji i urzędów skarbowych.

Wraz z rozwojem Internetu i wykształceniem wspomnianych trendów pojawiło się wiele zagrożeń, głównie dotyczących poufności przechowywanych i przetwarzanych informacji, w tym problemu bezpieczeństwa danych osobowych oraz anonimowości w sieci. Ponadto integracja usług na niespotykaną dotąd skalę znacząco ułatwiła monitorowanie działań obywateli.

Podczas wpisywania treści każdego e-maila lub korzystania z chmury trzeba mieć świadomość, że do przesłanych danych na pewno ma dostęp administrator usługi i dodatkowo uprawnione organy rządowe (po uzyskaniu odpowiednich zgód lub dzięki uprawnieniom nadanym przez poszczególne ustawy). Istnieje również

niebezpieczeństwo przechwycenia danych przez nieuprawnione osoby trzecie np. wskutek: włamania, braku zabezpieczeń,nieumiejętnego usunięcia, przeniesienia lub przechowywania.

Profil obywatela, jego działania, nawyki, preferencje dotyczące produktów i konsumpcji, legalność przychodów i rozchodów można określić na podstawie: wyciągów bankowych (miejsca, daty, zakupione produkty i usługi, adresaci przelewów, bilans środków), transakcji z użyciem kart bankowych (również miejsca, daty, zakupione produkty i usługi), nadajników telefonii GSM oraz sieci Wi-Fi (trasy jakie przemierzał obywatel oraz daty z dokładnością co najmniej do minut, niekiedy typ urządzenia dostępowego). W dodatku dąży się do integracji wszystkich tych informacji i ich udostępnienia pewnym organom m.in. przy pomocy aplikacji w środowisku rozproszonym i z naciskiem na mobilność dostępu.

Producenci sprzętu i oprogramowania nagminnie zbierają informacje o swoich klientach, niekiedy niezależnie od ich woli. Zazwyczaj odbywa się to z zastrzeżeniem celu: dostosowania oferty i doskonalenia usług. W rzeczywistości, zależnie od typu oprogramowania, nie jest wykonalne odkrycie prawdziwego zastosowania tych informacji oraz typu i zakresu przesyłanych danych.

Wszystko to, wraz z powszechnością dostępu do Internetu,spowodowało wykształcenie postawy roszczeniowej i wymagań, których niespełnienie skutkuje frustracją użytkownika. Typowe wymagania stawiane przed oprogramowaniem w środowisku rozproszonym, to:

1. Wygoda i intuicyjność – zapewnienie ergonomicznego interfejsu użytkownika.Wymaga to uwzględnienia potrzeb zdrowych użytkowników, osób niedowidzących i z wadami wzroku (np. z upośledzeniem postrzegania barw), a także typu urządzenia dostępowego (współcześnie szczególnie istotne są mobilne wersje oprogramowania). W jej ramach wykształciło się też pojęcie dyskryminacji w dostępie do informacji.
2. Stabilność działania – otrzymywanie poprawnych rezultatów w rozsądnym czasie i zawsze wtedy, gdy są potrzebne, niezależnie od urządzenia i medium dostępowego.
3. Bezpieczeństwo i trwałość informacji – trwałe zapisanie lub usunięcie danych wprowadzonych lub powstałych wskutek przetwarzania danych już istniejących

oraz zapewnienie, przez dostawcę usługi, mechanizmów chroniącychskładowane informacje przed utratą lub uszkodzeniem,np. wskutek awarii sprzętowej.

4. Prywatność – wykorzystanie danych wyłącznie w celu świadczenia usług, a tam gdzie to możliwe zachowanie pełnej anonimowości. Dodatkowo ograniczenie dostępu do tych danych wyłącznie do właściciela lub wskazanych przez niego osób (administrator usługi i pracownicy rządowi zazwyczaj nie zaliczają się do tego grona).

W związku z dużym zainteresowaniem oprogramowaniem działającym w środowisku rozproszonym, w tym w pełni profesjonalnymi aplikacjami internetowymi, wzrasta potrzeba szybkiego projektowania i implementacji takiego oprogramowania z uwzględnieniem takich aspektów, jak bezpieczeństwo, skalowalność, wydajność oraz ergonomia. Wymaga to wykształcenia specjalistów w tym zakresie, co wpływa również na pracę szkół i podnoszenie kwalifikacji kadry (Żyła i Kęsik, 2010b).

Bogactwo środowisk deweloperskich, technologii oraz metodyk wytwarzania oprogramowania spowodowało wykształcenie dwóch głównych nurtów (szkół, metodyk)w tej dziedzinie – wytwarzania oprogramowania z punktu widzenia programisty (*ang. programmer-likeapproach*) i z punktu widzenia projektanta (*ang. designer-likeapproach*) (Żyła i Kęsik, 2010a).

Pierwsza ze szkół wykorzystuje gotowe biblioteki, wzorce projektowe, frameworki, szablony aplikacji, itp.. Niemniej wymagania stawiane osobie tworzącej oprogramowanie są dosyć wysokie – m.in. znajomość środowiska programistycznego i technologii, umiejętność implementacji założeń aplikacji przedstawionych przez analityków oraz umiejętność pisania schludnego i wydajnego kodu. Z drugiej strony programista ma pełną władzę nad kodem aplikacji, więc implementacja niestandardowej funkcjonalności nie jest szczególnie problematyczna (Żyła i Kęsik, 2010a).

Druga ze szkół wykorzystuje zalety Inżynierii Sterowanej Modelami. Różnorakie modele złożone z konfigurowalnych komponentów tworzą wyższą warstwę abstrakcji, pozwalającą skupić się na zasadzie działania aplikacji, jej funkcjonalności, układzie treści, sposobie nawigacji oraz przepływie informacji pomiędzy jej modułami. Zyskuje się dzięki temu oszczędność czasu oraz zmniejsza liczbę potencjalnych błędów, jakie mogłyby się pojawić podczas kodowania typowej funkcjonalności aplikacji.

Poszczególne komponenty mogą być traktowane jako „czarne skrzynki" udostępniające pewną funkcjonalność, a ukrywające jej implementację. Dodatkowo model aplikacji stanowi również jej aktualną dokumentację. Niestety wynikowy kod aplikacji zazwyczaj nie jest w pełni optymalny, a w przypadku niestandardowej funkcjonalności (nie przewidzianej przez użytą notację), następuje powrót do pierwszej z metodyk. Z drugiej strony, w zależności od potrzeb, do stworzenia w pełni funkcjonalnej aplikacji nie jest potrzebny programista technologii web(Żyła i Kęsik, 2010a).

Książka prezentuje technologie MDE z naciskiem na zastosowania internetowe, szczególną uwagę poświęcono językowi WebML. Nie jest to jednak pełnia wiedzy. Ograniczona objętość nie pozwoliła na wgłębianie się w niuanse tych technologii, z drugiej zaś strony jest to dziedzina dość młoda, nie do końca ustandaryzowana, przez co podlegająca szybkim zmianom. Czytelnik posiądzie wiedzę o istocie i podstawach wytwarzania aplikacji (internetowych) sterowanego modelami, co pozwoli mu na jej dalsze samodzielne zgłębianie, teoretyczne i praktyczne.

Autorzy pragną podziękować recenzentom za ich cenne uwagi, które niewątpliwie przyczyniły się do poprawy jakości tej książki

Autorzy

1

Standardy i technologie realizacji aplikacji internetowych

Cel

Przedstawienie architektury klient-serwer oraz podstawowych technologii wykorzystywanych w procesie tworzenia aplikacji internetowych, z podziałem na wykorzystywane po stronie klienta oraz po stronie serwera. Omówienie sposobu formułowania wymagań funkcjonalnych i niefunkcjonalnych aplikacji, w celu określenia wymagań dotyczących implementacji, architektury aplikacji internetowej oraz jej użyteczności.

Plan

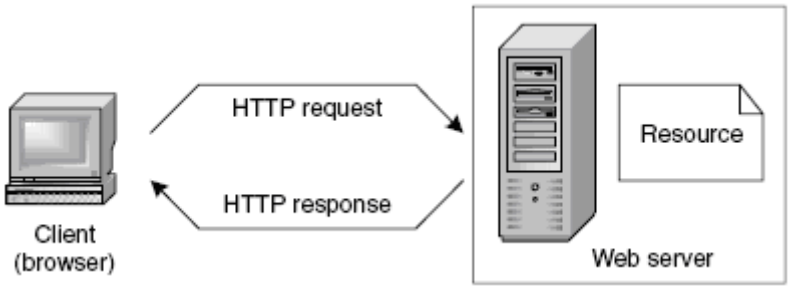
1. Architektura klient-serwer
2. Formułowanie wymagań wobec aplikacji internetowych
3. Wymagania stawiane współczesnym aplikacjom internetowym
4. Technologie w procesie wytwarzania aplikacji internetowych

1.1 ŚRODOWISKO ROZPROSZONE

Rozproszone środowisko przetwarzania, to środowisko złożone z infrastruktury technicznej (węzłów i łączy) oraz zbioru procesów wspólnie realizujących pewne cele. W skład infrastruktury technicznej wchodzą węzły (jednostki przetwarzające dane) oraz łącza komunikacyjne spinające węzły funkcjonujące w rozproszonym systemie informatycznym. Poprzez łącza odbywa się komunikacja pomiędzy węzłami. Przesyłane pakiety danych mogą stanowić dane bazowe, wyniki przetwarzania oraz informacje: o z mianie statusu węzła, o statusie przetwarzania, o realizowanym zadaniu lub synchronizujące przetwarzanie. Z kolei poszczególne węzły dysponują własnym procesorem, lokalną pamięcią oraz interfejsem komunikacyjnym(Tanenbaum i van Steen, 2002).

Zaletami systemów rozproszonych jest m.in.: wydajność, opłacalność, wykorzystanie zasobów, skalowalność i niezawodność. Niestety jednocześnie pojawia się problem: optymalnego zrównoleglenia przetwarzania, oceny poprawności i efektywności algorytmów, alokacji zasobów, synchronizacji procesów oraz zapewnienia bezpieczeństwa i niezawodności. Dąży się do tego, aby systemy rozproszone: umożliwiały rozproszenie geograficzne węzłów, były w stanie pracować w środowisku heterogenicznym (uwzględniając sprzęt, oprogramowanie oraz łącze), korzystały z uniwersalnych protokołów i interfejsów oraz były odporne na awarie(Tanenbaum i van Steen, 2002).

Z punktu widzenia użytkownika, upraszczając pewne kwestie, środowisko rozproszone funkcjonuje na zasadzie architektury klient – serwer, gdzie serwer jest siecią rozległą świadczącą pewne usługi, a klient jest aplikacją konkurującą z innymi o dostęp do pewnego typu usług (usługą może być np. zapisanie informacji w bazie danych). Serwer zapewnia usługi biernieoczekując na żądaniazgłaszane przez klientów. Po otrzymaniu żądania serwer wykonuje określone operacjei odsyła wynik do odpowiedniego klienta(rys 2.1). Z usług jednego serwera może korzystać wielu klientów, co nie przeszkadza, aby jednocześnie jeden klient korzystał z usług wielu serwerów oferujących zróżnicowane usługi.



Rys. 1.1. Architektura klient – serwer na przykładzie protokołu HTTP
Źródło: (Ceri i inni, 2003)

HTTP (*ang. HyperText Transfer Protocol*) jest bezstanowym protokołem klient – serwer stworzonym przez Tima Berners – Lee i Roberta Cailliau na początku lat 90 – tych XX wieku, umożliwiającym użytkownikom dostęp do zasobów znajdujących się na zdalnych serwerach. Protokół określa reguły, według których aplikacja kliencka komunikuje się z aplikacją serwerową, w celu uzyskania dostępu do zasobu (rys. 1.1) poprzez żądania (*ang. HTTP request*) i odpowiedzi (*ang. HTTP response*), odbywa sie to przy wykorzystaniu standardowego systemu adresowania zasobów URL (*ang. Uniform Resource Locators*)

http:// <host> [: <port>] [<path> [? <query>]]

gdzie:

- <*host*> - nazwa serwera lub jego adres IP,
- <*port*> - specyficzny port serwera,
- <*path*> - ścieżka dostępu do zasobu w systemie plików serwera,
- <*query*> - zestaw parametrów dodatkowych żądania.

Protokół HTTP jest bezstanowy, więc każde żądanie (realizowane metodą POST lub GET) jest traktowane przez serwer jako pojedyncze i niezależne. Nie ma różnicy pomiędzy sekwencją żądań od tego samego klienta lub różnych klientów, w związku z tym przechowywanie historii interakcji klienta z serwerem musi być realizowane poza protokołem HTTP. Ponadto interakcja serwera z klientem odbywa się wyłącznie w odpowiedzi na żądanie ze strony klienta(Ceri i inni, 2003).

1.2 SPECYFIKOWANIE WYMAGAŃ STAWIANYCH APLIKACJOM INTERNETOWYM

Proces specyfikacji wymagań składa się z dwóch etapów: zbierania informacji o aplikacji i ich analizy. Celem tego procesu jest określenie domeny oraz ogólnego obrazu aplikacji (funkcjonalność, układ stron, aktorzy itp.).

Niezależnie od metodyki wytwarzania aplikacji internetowych, w jej ramach wyróżnia się wejścia w postaci wymagań biznesowych i stałych środowiskowych oraz wyjścia, czyli gotową aplikację działającą na określonej platformie sprzętowej i programowej, razem z dokumentacją i instrukcją obsługi.

UCZESTNICY PROCESU TWORZENIA APLIKACJI INTERNETOWEJ

W ramach cyklu projektowania aplikacji internetowejmożna wyróżnić następujące role (Ceri i inni, 2003):

1. Analityk – jest odpowiedzialny za zbieranie wymagań biznesowych i ich wstępne opracowanie poprzez odpowiednią interpretację i opis celów długofalowych.
2. Projektant baz danych – jest odpowiedzialny za opracowanie modelu danych zaspokajającego potrzeby tworzonej aplikacji.
3. Architekt aplikacji – jest odpowiedzialny za implementację oferowanej przez aplikację funkcjonalności.
4. Grafik – jest odpowiedzialny za całość warstwy prezentacji projektowanej aplikacji.
5. Administrator – jest odpowiedzialny za wdrożenie aplikacji, czynności administracyjne i konserwację.

Bywa, że wszystkie te role są odgrywane przez jedną osobę, wtedy docenia się wygodę notacji oraz profesjonalizm IDE (*ang. Integrated Development Environment*) połączony z ułatwieniami przyspieszającymi tworzenie i wdrażanie systemów informatycznych.

FORMUŁOWANIE WYMAGAŃ FUNKCJONALNYCH

Wymagania funkcjonalneokreślają funkcjonalność aplikacji oraz reguły dostępu do niej. Środki wyrazu mogą być różne, począwszy od rysunków na kartce papieru, przez

półformalne notacje tekstowe, a skończywszy na w pełni formalnych diagramach w UML (*ang. UnifiedModeling Language*) lub innych notacjach. Zbiór środków wyrazu w zasadzie nie podlega większym ograniczeniom – dozwolone jest wszystko to, co pozwala dobrze opisać aplikację, ale w ilościach nie komplikujących niepotrzebnie jej obrazu. Zazwyczaj metodyki wytwarzania oprogramowania określają minimalny zestaw diagramów i opisów, włącznie ze sposobem ich konstruowania, wymagany do poprawnego przedstawienia aplikacji.

Wyniki procesu specyfikacji wymagań funkcjonalnych różnią się w zależności od metodyki, jednakże w większości przypadków zawierają(Ceri i inni, 2003):

1. Reguły dostępu do funkcjonalności aplikacji – opis aktorów oraz ich uprawnień, niekiedy uzupełniony rysunkiem przedstawiającym ich hierarchię. W skład opisu mogą wchodzić dane teleadresowe, dane osobowe, przypadki użycia, w których biorą udział aktorzy, itp..
2. Scenariusze interakcji aktorów z aplikacją – opis funkcjonalności aplikacji, sposobu jej realizacji, z uwzględnieniem podziału na moduły oraz znaczenia aktorów. Zazwyczaj reprezentowane przez diagram przypadków użycia oraz scenariusze użycia, będące składowymi UML.
3. Dystrybucję przypadków użycia – opis rozmieszczenia treści i funkcjonalności w ramach modułów, okien lub stron aplikacji. Szczególnie użyteczne okazują się notacje graficzne przedstawiające poszczególne komponenty aplikacji oraz przepływ sterowania pomiędzy nimi.
4. Spis obiektów w dziedzinie aplikacji – opis reprezentacji oraz charakterystyka istotnych obiektów świata rzeczywistego, na których operuje aplikacja. Ich wybór ściśle zależy od dziedziny aplikacji i jest bardzo intuicyjny.
5. Szkic interfejsu użytkownika– zestandaryzowane dla grup stron/okien aplikacji rozmieszczenie pewnych elementów interfejsu użytkownika. Może zawierać wytyczne względem m.in.: kolorystyki, położenia, zawartości i rozmiaru stopki, układu i położenia menu, rozmiaru i kolejności przycisków, itp..

FORMUŁOWANIE WYMAGAŃ NIEFUNKCJONALNYCH

Wymagania niefunkcjonalne opisują wszystkie kwestie implementacyjne dotyczące

tworzonej aplikacji, m.in. architekturę sprzętową i programistyczną niezbędną do działania aplikacji, rozmieszczenie modułów aplikacji oraz zagadnienia komunikacji (np. adresacja, wykorzystywane protokoły). Mechanizmy doboru środków wyrazu oraz ich mnogości są analogiczne do wymagań funkcjonalnych.

Wyniki procesu specyfikacji wymagań niefunkcjonalnych różnią się w zależności od metodyki, jednakże w większości przypadków poruszają zagadnienia związane z (Ceri i inni, 2003):

1. Użytecznością– spójność interfejsu aplikacji, przyjazność użytkownikowi pod kątem interfejsu, sposobu realizacji funkcjonalności oraz dokumentacji.
2. Wydajnością – sposób wykorzystania zasobów dostępnych dla aplikacji uwzględniający jej obciążenie, w tym złożoność wykonywanych zadań oraz jednoczesną ilość zadań do wykonania.
3. Dostępnością – sposób wykorzystania mechanizmów zapewniających ciągłość działania aplikacji uwzględniający analizę dopuszczalnejczęstości błędów i awarii wpływających na czas, przez który użytkownicy mogą korzystać z aplikacji.
4. Skalowalnością – sposób zwiększenia wydajności aplikacji w odpowiedzi na jej zwiększone obciążenie.
5. Bezpieczeństwem – przewidziane mechanizmy ochrony integralności, trwałości i poufności przetwarzanych informacji, w tym mechanizmy dostępu do informacji, zarządzania informacją oraz polityka jej udostępniania uprawnionym osobom trzecim.

1.3 TECHNOLOGIE WYKORZYSTYWANE W PROCESIE TWORZENIA APLIKACJI INTERNETOWYCH

Obecnie technologie wykorzystywane w procesie tworzenia aplikacji ulegają dynamicznym zmianom w związku z coraz szerszymi wymaganiami biznesowymi dla obszaru WEB. Niemniej istnieje ich pewien podzbiór typowy dla każdej aplikacji internetowej.

SGML i HTML

SGML (*ang.Standard GeneralizedMarkup Language*), to nadrzędny język definiowania zbiorów znaczników, opracowany jeszcze w latach 60 – tych XX wieku przez Charlesa Goldfarba, Edwarda Moshera i Raymonda Lorie. Służy przede wszystkim do precyzyjnego definiowania zbiorów znaczników konkretnego zastosowania np. język HTML (*ang. HyperTextMarkup Language*) oraz standaryzacji przepływu dokumentów w obrębie instytucji. W jego ramach wyróżnia się:

- deklarację dokumentu – definicja reguł stosowanych w zapisie dokumentu,
- definicję typu dokumentu – zestaw znaczników i reguł ich stosowania,
- właściwy dokument – treść dokumentu wraz ze znacznikami.

W związku ze złożonością standardu i trudnościami implementacyjnymi, wyodrębniono podzbiór reguł języka SGML, na podstawie którego powstał XML (*ang. eXtensibleMarkup Language*) (W3C, 2011a).

W oparciu o SGML pracownicy ośrodka naukowo – badawczego CERN Tim Berners – Lee i Robert Cailliau zdefiniowali HTML – niezależny od platformy sprzętowej i systemowej język tworzenia dokumentów w ramach sieci World Wide Web. Jego pierwsza publicznie dostępna specyfikacja ukazała się w Internecie w 1991 roku. Aktualna wersja języka, to HTML 5. Korzystając z zestawu znaczników zdefiniowanych w ramach języka HTML, można tworzyć dokumenty, przekazywane z serwerów WWW do przeglądarek za pomocą bezstanowego protokołu HTTP, zawierające m.in. metadane, tekst, hiperłącza, interaktywne formularze, obiekty multimedialne, statyczną grafikę oraz dynamiczne animacje. Możliwe jest również osadzanie ciągów instrukcji języków skryptowych(W3C, 2011b).

XML i XSL

XML (*ang. eXtensibleMarkup Language*), to metajęzyk służący do definicji zbiorów znaczników, powstały z podzbioru reguł języka SGML. Jego bazowa specyfikacja została opublikowana przez konsorcjum W3C w 1998 roku. Został zaprojektowany, aby zaspokoić potrzeby związane z publikacją dokumentów elektronicznych na szeroką skalę oraz wymianą różnorodnych danych w ramach sieci WWW (W3C,

2011c). Aktualnie jest również bardzo często stosowany w plikach konfiguracyjnych systemów operacyjnych i aplikacji działających w ich ramach.

Dokumenty XML, w zależności od swojej złożoności, mogą być trudne w odbiorze dla człowieka. W związku z tym stosuje się transformację takich dokumentów do postaci zakodowanej przy użyciu języka ukierunkowanego na prezentację danych. Przekształcenie takie wykonuje się wykorzystując język XSL (*ang. eXtensibleStylesheet Language*) i polega ono na odnalezieniu pewnego obszaru tłumaczonego dokumentu, a następnie przetworzeniu według dotyczących go reguł. W skład XSL wchodzą trzy języki (W3C, 2011d):

- XPath (*ang. XML Path*) – język służący znajdowaniu informacji oraz nawigacji po elementach i atrybutach w dokumencie XML,
- XSLT (*ang. XSL Transformation*) – język definiowania reguł transformacji dokumentów XML,
- XSL – FO (*ang. XSL Formatting Objects*) – język formatowania danych XML do postaci wyświetlanej na ekranie, drukowanej na papierze lub przesyłanej do innych mediów.

CSS

CSS (*ang. Cascading Style Sheets*), to tzw. kaskadowe arkusze stylów, będące prostym mechanizmem odpowiadającym za sposób wyświetlania elementów HTML strony internetowej. Zostały dołączone do specyfikacji języka HTML 4.0 przez konsorcjum W3C w odpowiedzi na problem oddzielenia kodu odpowiadającego za wygląd strony internetowej od treści, którą zawiera strona.

Kaskadowe arkusze stylów dzielą się na wewnętrzne oraz zewnętrzne. Wewnętrzne są umieszczane wewnątrz znacznika HEAD strony internetowej. Zewnętrzne arkusze stylów są umieszczane w plikach o rozszerzeniu *.css* i w przeciwieństwie do wewnętrznych mogą zostać użyte w więcej niż jednej stronie internetowej. Implikuje to, że edycja pojedynczego pliku *.css* wpływa na wygląd wszystkich stron internetowych, w których go użyto.

Kaskadowe arkusze stylów posiadają swoją hierarchię ważności, co rozwiązuje problem związany z definicją wielu arkuszy dotyczących tej samej strony internetowej.

W pierwszej kolejności za wygląd strony odpowiadają style zdefiniowane wewnątrz danego znacznika (elementu) języka HTML, następnie wewnętrzne arkusze stylów zdefiniowane w ramach sekcji HEAD strony internetowej, zewnętrzne arkusze stylów oraz domyślne style przeglądarki internetowej. Hierarchia powoduje m.in., że wygląd elementu strony określony *inline*, tzn. w ramach znacznika HTML, nie zostanie zmieniony przez pozostałe arkusze stylów, użyte w tej stronie (W3Schools, 2011a).

JAVASCRIPTI AJAX

JavaScript, to język skryptowy, działający po stronie aplikacji klienckiej, zaprojektowany przez Brendana Eicha i firmę Netscape, w celu dodania interaktywności do stron HTML. Skrypty napisane przy jego użyciu są interpretowane przez przeglądarki, tzn. ich wykonanie odbywa się bez prekompilacji. Po raz pierwszy jego obsługa pojawiła się w przeglądarce stron internetowych Netscape Navigator 2.0. Niedługo potem firma ECMA, na podstawie JavaScript firmy Netscape i JScript firmy Microsoft, opracowała standard ECMA – 262, który od 1998 roku stał się oficjalnym międzynarodowym standardem JavaScript (ISO/IEC 16262), a zdefiniowany w nim język nazwano ECMAScript(W3Schools, 2011d).

JavaScript jest obecnie najpopularniejszym językiem skryptowym, osadzanym wewnątrz dokumentów HTML, umożliwiającym m.in.:

- dynamiczne umieszczanie w kodzie HTML łańcuchów tekstowych,
- reakcję na zdarzenia,
- odczyt i modyfikację elementów HTML,
- walidację danych,
- tworzenie ciasteczek.

AJAX (*ang. AsynchronousJavaScript and XML*), to technika tworzenia aplikacji internetowych, w której interakcja użytkownika z serwerem odbywa się w sposób asynchroniczny, bez konieczności przeładowania całego dokumentu. Integruje w sobie technologie takie, jak:

- XHTML i CSS – odpowiadają za prezentację treści dokumentu,
- DOM (*ang. Document Object Model*) – odpowiada za logiczny podział, dynamiczne wyświetlanie i interakcję z danymi zawartymi na stronie www,

- XML i XSLT – odpowiadają za wymianę i manipulację danymi,
- XMLHttpRequest – odpowiada za asynchroniczne pozyskiwanie danych,
- JavaSript lub inny język skryptowy obsługujący model DOM – odpowiada za mediację pomiędzy wymienionymi technologiami.

Asynchroniczność modelu AJAX polega na uniezależnieniu interakcji użytkownika z aplikacją od komunikacji z serwerem. Pomiędzy GUI (*ang. Graphical User Interface*), a częścią serwerową znajduje się silnik AJAX, oparty o język skryptowy, który odpowiada za interfejs użytkownika oraz w razie potrzeby komunikuje się z serwerem. Umożliwia to wzbogacenie aplikacji internetowej m.in. o (W3Schools, 2011b):

- mechanizm przeciągnij i upuść (*ang. drag & drop*),
- selektywne odświeżanie zawartości strony,
- okresowe odświeżanie zawartości strony,
- warunkowe przeładowanie zawartości listy rozwijanej,
- zarządzanie zdarzeniami,
- podpowiedzi dotyczące możliwych wartości pól formularza,
- użycie okien modalnych.

TECHNOLOGIE DOSTĘPU DO DANYCH

SQL (*ang. Structured Query Language*), to jeden z najpopularniejszych języków służących do manipulacji danymi składowanymi w bazach danych. SQL został opracowany przez IBM Corporation Inc. w 1970 roku, w 1979 roku doczekał się swojej pierwszej implementacji w bazie danych Oracle, a aktualnie jest standardowym językiem zapytań relacyjnych baz danych. Celem języka było dostarczenie interfejsu pozwalającego na pracę z dużą ilością danych bez konieczności implementacji takich szczegółów, jak pozyskiwanie wierszy z tabeli z uwzględnieniem sposobu ich składowania i współbieżnego dostępu (W3Schools, 2011c).

W ramach SQL wyróżnia się 3 podstawowe języki:

1. DML (*ang. Data Manipulation Language*) – język manipulacji danymi, m.in. instrukcje INSERT, UPDATE, DELETE.
2. DDL (*ang. Data Definition Language*) – językdefiniowaniadanych, m.in.

instrukcjeCREATE, ALTER, DROP.

3. DCL (*ang. Data Control Language*) – język kontrolowania danych, m.in. instrukcje GRANT, REVOKE, DENY.

W zależności od typu relacyjnej bazy danych, dostępne są proceduralne rozszerzenia języka SQL. Ich integracja z językiem SQL pozwala na użycie tych samych instrukcji, funkcji, operatorów, pseudokolumn, typów danych i wykonywanie identycznych operacji na dan ych, co w SQL, ale poszerza możliwości o obsługę wyjątków oraz struktury sterowania. Proceduralna odmiana SQL znajduje zastosowanie przede wszystkim w procedurach i funkcjach składowanych.

W celu uzyskania dostępu do informacji składowanych w bazach danych można również wykorzystać specjalne biblioteki programistyczne(zbiory klas i interfejsów) dedykowane specyficznym wersjom i typom baz danych, często nazywane sterownikami, które pozwalają na połączenie aplikacji z bazą danych oraz manipulowanie nią bezpośrednio z poziomu języka programowania. Przykładem takich sterowników mogą być sterowniki ODBC (*ang. Open DataBase Connectivity*) lub JDBC (*ang. Java DataBase Connectivity*). Ich użycie pozwala na dostęp do bazy danych na niższym poziomie niż oferowany przez język SQL.Dla przykładu interfejs programistyczny JDBC od wersji 2.0zapewnia dodatkowo pełne wsparcie dla obiektowo – relacyjnych baz danych oraz umożliwia współpracę z plikami zawierającymi stabelaryzowane dane (tzw. *flatfiles* i arkusze kalkulacyjne), wykraczając tym samym poza granice języka SQL(Oracle, 2011b).

Zazwyczaj architektura sterowników zakłada istnienie sterownikaspecyficznego dla każdego typu, a nawet wersji,bazy danych oraz interfejsu programistycznego przeznaczonego dla twórców aplikacji w danym języku programowania. Taka struktura pozwala na opracowanie jednej aplikacji, komunikującej się z wieloma heterogenicznymi źródłami danych, przy użyciu specyficznych dla źródeł sterowników, co określa się mianem „Write Once, Run Anywhere” (*pol. „Napisz raz, uruchom wszędzie”*). Przetwarzanie w takiej architekturze przebiega w trzech etapach: nawiązanie połączenia ze źródłem danych, przesłanie zapytań do źródła oraz przetworzenie rezultatów.

JĘZYKI PROGRAMOWANIA DZIAŁAJĄCE PO STRONIE SERWERA

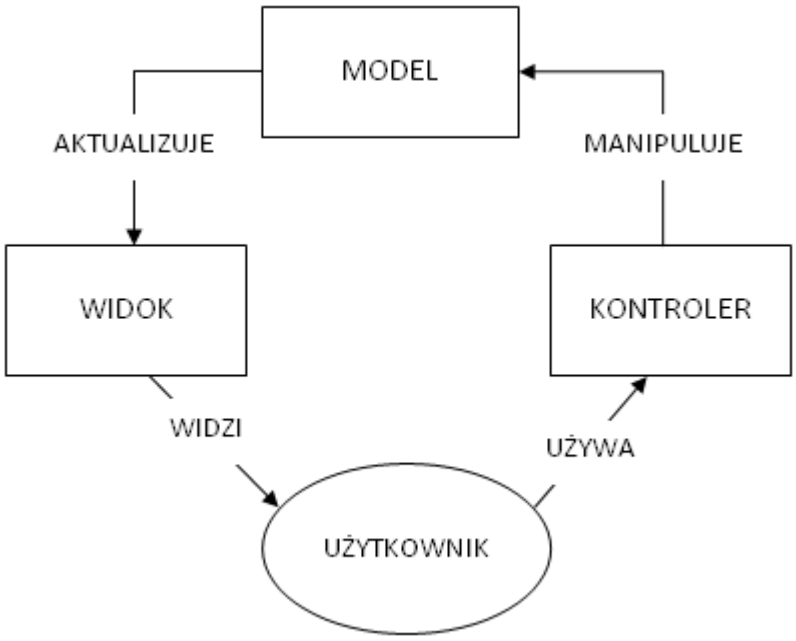
Języki programowania działające po stronie serwera kodują logikę biznesową aplikacji, która w reakcji na żądanie od klienta wykonuje zbiór instrukcji takiego języka, zwracając ich wynik klientowi. Obecnie tego typu języki są składowymi całych platform programistycznych, przy czym można wyróżnić dwie znaczące kategorie – platformy wymagające do swojego działania maszyny wirtualnej (np. Java) i platformy, które tego nie wymagają (np. PHP).

Dla przykładu, w ramach platformy Java Enterprise funkcjonuje technologia JSP (*ang. Java Server Pages*). Wspomaga ona tworzenie aplikacji internetowych opierając się na językach HTML, XML i ich pochodnych, wykorzystując je, a w szczególności zdefiniowane przy ich pomocy znaczniki, do w spółpracy z bibliotekami programistycznymi. Pozwala to na rozgraniczenie kodu odpowiedzialnego za prezentację strony od warstwy logiki biznesowej. Dodatkowo w kontekście tych technologii funkcjonuje specjalna warstwa mapująca, która w oparciu o sterowniki umożliwia odwzorowanie klas języka na tabele bazy danych (Lindenberg, 2003). Na analogicznych założeniach opiera się technologia ASP (*ang. Active Server Pages*) firmy Microsoft.

W ramach technologii podobnych do JSP funkcjonują specjalne kolekcje (biblioteki) znaczników umożliwiających realizację typowej funkcjonalności stron aplikacji internetowej, przy jednoczesnym zapewnieniu jednolitego interfejsu niezależnie od serwera, na którym odbywa się przetwarzanie. Określenie zakresu ich funkcjonalności zazwyczaj odbywa się na podstawie doświadczeń programistów oraz autorskich bibliotek znaczników. Główny obszar funkcjonalności takich bibliotek pokrywa zagadnienia związane ze: sterowaniem przebiegiem przetwarzania, śledzeniem sesji, przetwarzaniem i analizą składniową dokumentów, internacjonalizacją aplikacji i dostępem do bazy danych(Lindenberg, 2003).

W ramach platform programistycznych funkcjonują specjalne frameworki (zestawy gotowych narzędzi i wzorców) tworzone z myślą o przyspieszeniu procesu tworzenia aplikacji w oparciu o pewną architekturę, np. MVC (*ang. Model View Controller*) – inaczej Model Widok Kontroler, która obecnie stała się niemalże standardem (rys 2.2). W jej ramach Model odpowiada za przetwarzanie i dostęp do

danych, Widok odpowiada za sposób prezentacji danych oraz interfejsu użytkownika, a Kontroler odpowiada za koordynację pracy Widoku i Modelu(Żyła i Kęsik, 2010a).



Rys 2.2. Schemat architektury MVC
Źródło: opracowanie własne

Regułę działania architektury MVC, stosując pewne uproszczenia, można opisać następująco: w chwili zgłoszenia żądania przez klienta, Kontroler inicjuje pewne czynności (przetwarzanie) w ramach Modelu, w tym dostęp i operacje na bazie danych, jeśli zachodzi taka potrzeba. Wyniki tego przetwarzania są kierowane do Widoku, który umieszcza je w odpowiednich miejscach szablonów stron aplikacji internetowej. Następnie klient, w odpowiedzi na żądanie, otrzymuje stronę w języku HTML z ewentualnymi wstawkami z języków skryptowych zawierającymi instrukcje wykonywane przez przeglądarkę stron internetowych (Żyła i Kęsik, 2010a).

SOA

SOA (*ang. Service Oriented Architecture*) to architektura, w której komunikacja

pomiędzy systemami informatycznymi (również heterogenicznymi) odbywa się przy użyciu serwisów. Serwis jest definiowany przez konsorcjum W3C jako fragment oprogramowania wspierający współdziałanie systemów informatycznych w ramach sieci, posiadający odpowiednio opisany interfejs, z użyciem którego odbywa się interakcja systemów poprzez komunikaty SOAP (*ang. Simple Object Access Protocol*), przekazywane przy użyciu protokołu HTTP, w połączeniu z XML oraz innymi standardami obowiązującymi w sieci.Najważniejsze technologie funkcjonujące w ramach architektury SOA, to (W3C, 2004):

1. XML – dostarcza spójny i rygorystyczny zestaw definicji elementów dokumentów wymienianych w ramach architektury.
2. SOAP – dostarcza narzędzia do wymiany wiadomości XML przy użyciu protokołów sieciowych takich, jak np. HTTP, SMTP (*ang. Simple Mail Transfer Protocol*), FTP (*ang. File Transfer Protocol*). Definiuje trzy opcjonalne komponenty: zbiór reguł kodowania, konwencję reprezentowania wywołań RPC (*ang. Remote Procedure Call*) i ich odpowiedzi oraz zestaw reguł stosowania SOAP w połączeniu z protokołem HTTP.
3. WSDL (*ang. Web Services Description Language*) – język służący do kompleksowego opisu serwisów, wraz z wiadomościami wymienianymi pomiędzy dostawcą i odbiorcą usług świadczonych przez serwis. Wiadomości są opisywane przy użyciu XML, a następnie wiązane z konkretnym protokołem sieciowym i formatem komunikatu.

Architektura zorientowana na serwisy charakteryzuje sięprzede wszystkim następującymi właściwościami(W3C, 2004):

- zorientowanie na funkcjonalność – serwis jest abstrakcyjnym widokiem określonej funkcjonalności systemu informatycznego, spojrzeniem na to, co system robi,
- zorientowanie na wiadomości – serwis posiada definicję formalną opartą na wiadomościach wymienianych pomiędzy dostawcą usług, a ich odbiorcą; definicja ta jest niezależna od architektury oraz szczegółów implementacyjnych dostawcy i odbiorcy usług,
- zorientowanie na opis – serwis jest opisywany przez metadane przetwarzalne przez systemy informatyczne; opis serwisu powinien dotyczyć tylko szczegółów eksponowanych jako publiczne,

- zorientowanie na sieć – serwisy są zorientowane na pracę w środowisku rozproszonym,
- niezależność od platformy – wiadomości są przesyłane przez interfejsy w niezależnym od platformy ustandaryzowanym formacie, np. XML.

UML

UML, to zestandaryzowany przez OMG (*ang. Object Management Group*) wielozadaniowy język modelowania, używany w inżynierii oprogramowania do modelowania wizualnego. Notacja UML powstała z połączenia kluczowych dla niej technik:

- The Booch Method – zorientowana obiektowo metoda analizy i projektowania, autor: Grady Booch,
- OOSE (*ang. Object – Oriented Software Engineering*) – metoda skupiająca się na koncepcjach związanych z przypadkami użycia, autor: Ivar Jacobson,
- OMT (*ang. Object – ModelingTechnique*) – metoda skupiająca się na analizie procesów biznesowych i systemach przetwarzających duże ilości danych, autor: James Rumbaugh.

Konsolidację tych technik i rozszerzanie ich koncepcji rozpoczęła w 1993 roku firma Rational Software Corp. Od 1997 roku opieki nad UML podjęła się grupa OMG (Boggs i Boggs, 2002).

W ramach notacji UML wyróżniono grupy diagramów przedstawiające organizację, strukturę oraz zachowanie projektowanego systemu informatycznego, a wśród nich (Boggs i Boggs, 2002):

- diagramy przypadków użycia – obrazują funkcjonalność systemu i grupy użytkowników, które mają do niej dostęp; przedstawiają widok projektowanego systemu z punktu widzenia jego użytkowników,
- diagramy aktywności – obrazują przepływ funkcjonalności w projektowanym systemie,
- diagramy sekwencji – obrazują przepływ funkcjonalności w ramach przypadku użycia,
- diagramy współpracy – obrazują rozmieszczenie przetwarzania w ramach

przypadku użycia; pozwalają na wykrycie przeciążonych obiektów,
- diagramy klas – obrazują interakcje pomiędzy klasami w projektowanym systemie,
- diagramy stanów – obrazują możliwe stany obiektu w systemie wraz ze wskazaniem sposobu przejść pomiędzy tymi stanami,
- diagramy komponentów – obrazują komponenty projektowanego systemu i relacje pomiędzy nimi,
- diagramy wdrożeniowe – obrazują fizyczny rozkład komponentów systemu, uwzględniając architekturę sprzętową, protokoły komunikacyjne i inne wymagania niefunkcjonalne.

1.4 WSPÓŁCZESNE TRENDY W TWORZENIU APLIKACJI INTERNETOWYCH

Wraz ze wzrostem ilości i rodzajów usług dostępnych przez Internet, wzrasta również konkurencja na polu aplikacji internetowych. Użytkownicy porównują różne serwisy dostępne w sieci, a porównywanie to dawno już wykroczyło poza sam wygląd interfejsu. Wymusza to oferowanie innowacyjnych rozwiązań często uprzedzających działania konkurencji, a potrzeby biznesowe przenoszą się na coraz bardziej wygórowane wymagania stawiane przed współczesnymi aplikacjami internetowymi.

NACISK NA SZYBKOŚĆ APLIKACJI WEB

W dobie szerokopasmowego Internetu użytkownik oczekuje dostarczania informacji bez opóźnień, najlepiej w sposób jak najbardziej przypominający korzystanie z desktopowej aplikacji. Oczekiwanie to jest już dobrze zauważalne w biznesie – np. Google wykorzystuje informacje o szybkości aplikacji internetowej jako jeden z parametrów jej pozycjonowania na liście wyszukanych.

Szybkość aplikacji web zależy m.in. od czasu przesyłu informacji przez Internet, szybkości wygenerowania odpowiedzi przez serwer oraz szybkości renderowania strony internetowej przez przeglądarkę. O ile budowa aplikacji nie ma większego wpływu na szybkość transferu przez Internet, o tyle pozostałe dwa aspekty silnie od

niej zależą. W związku z tym zbiór najlepszych praktyk budowy aplikacji pod kątem optymalizacji szybkości ich działania dynamicznie się rozrasta.

NACISK NA NOWE TECHNOLOGIE

Standardy tworzenia aplikacji internetowych zmieniają się ze wzrastającą szybkością. Obecnym trendem jest wykorzystywanie HTML 5 i CSS 3. Udostępniają one m.in. następującą funkcjonalność(Adobe, 2011):

- natywna obsługa audio i video z wykorzystaniem własnych znaczników i możliwością dopasowania źródła do możliwości przeglądarki,
- tworzenie interaktywnych samodzielnie odświeżających dane tabel,
- osadzanie w kodzie wielu obiektów graficznych (np. w postaci skryptów),
- wykorzystanie kanwy do dynamicznego i interaktywnego zarządzania grafiką,
- animacje elementów DOM bez konieczności wykorzystywania skryptów.

Nazwy HTML 5 i CSS 3 oraz świadomość (przynajmniej częściowa) udostępnianych przez nie nowych możliwości dominują w biznesie. Naturalnym jest więc wymaganie ich wykorzystania towarzyszące zgłaszaniu zapotrzebowania na nowe aplikacje.

NACISK NA MOBILNOŚĆ

W dobie urządzeń mobilnych z dostępem do www, na porządku dziennym jest konieczność dopasowania interfejsu serwisu do szerokiej gamy rozdzielczości i rodzajów ekranów. Tworzenie specjalnych wersji serwisu dla pewnej grupy urządzeń wiąże się z wieloma niedogodnościami, jak powielanie informacji, czy różna adresacja tego samego serwisu (skutkująca pogorszeniem pozycjonowania w wyszukiwarkach).

Promowaną obecnie praktyką jest „reagujące projektowanie web” (*ang. responsive web design*), zakładające dopasowanie się wyglądu strony web do dowolnej rozdzielczości ekranu. Dopasowanie to polega nie tylko na skalowaniu, ale również na przemodelowaniu (włącznie z pominięciem mniej istotnych) elementów składowych strony tak, aby najlepiej pasowały do danego kształtu i rozmiarów.

W połączeniu z możliwościami HTML 5 związanymi z przechowywaniem danych bez dostępu do Internetu, istnieje możliwość (a więc i zapotrzebowanie) budowy

internetowych aplikacji mobilnych, posiadających zalety typowych bazodanowych aplikacji i omijających potrzebę tworzenia wersji na różne platformy.

W związku z tym biznes coraz częściej kwestionuje tworzenie aplikacji internetowych nie uwzględniających potrzeb i wymagań posiadaczy urządzeń mobilnych.

1.5 PYTANIA KONTROLNE

1. Jakie są podstawowe technologie wytwarzania oprogramowania działającego w środowisku rozproszonym?
2. Jakie są współczesne trendy i oczekiwania względem aplikacji internetowych?
3. Opisz inną architekturę niż MVC.
4. Na czym polega trójwarstwowa architektura klient-serwer?
5. Jakie technologie i dlaczego są w stanie zastąpić JSP i ASP?

2

Wytwarzanie oprogramowania sterowane modelami

Cel

Przedstawienie koncepcji oraz objaśnienie terminologii związanej z MDE (*ang. Model Driven Engineering*). Zapoznanie z jej głównymi nurtami: MDA (*ang. Model DrivenArchitecture*), MDSD (*ang. Model DrivenSoftware Development*). Zaprezentowanie wad i zalet projektowania sterowanego modelami.

Plan

1. Objaśnienie co to jest i jakie cele stawia przed sobą MDE
2. Przedstawienie MDA
3. Omówienie rodzajów modeli MDA
4. Przedstawienie procesu budowy MDA
5. Transformacje modeli
6. Przedstawienie i omówienie MDSD
7. Omówienie zalet i wad projektowania sterowanego modelami

2.1 MODEL DRIVEN ENGINEERING

Model Driven Engineering; projektowanie (aplikacji) sterowane modelami; ma na celu podniesienie poziomu abstrakcji oraz zwiększenie automatyzacji w wytwarzaniu oprogramowania.

Hasło przewodnie MDE: „wszystko jest modelem" nawiązuje do hasła „wszystko jest obiektem" programowania zorientowanego obiektowo, promując wykorzystanie modeli na różnych (możliwe wielu) poziomach abstrakcji tworzenia systemów. Zwiększa toogólny poziom abstrakcji opisu oprogramowania. Zwiększenie automatyzacji budowy oprogramowania jest uzyskiwane przez wykorzystanie (automatycznych) transformacji modeli wyższego rzędu do modeli rzędu niższego, aż do uzyskania wykonywalnego kodu, na drodze jego generacji lub interpretacji modelu odpowiednio niskiego poziomu (den Haan, 2008c).

Główną motywacją wykorzystywania MDE jest podniesienie produktywności. W czasach, gdy nie można już zbudować aplikacji kompatybilnej ze wszystkimi obecnymi i powstającymi systemami, ważne jest uzyskanie jak największego przychodu zinwestycji w jej wyprodukowanie. MDE przyczynia się do tego zarówno w krótkim okresie – przez wzrost funkcjonalności podstawowych cząstek oprogramowania, jak i w dłuższej perspektywie – przez zmniejszenie szybkości dezaktualizowania się tych cząstek.

Wiele technologii i narzędzi ma głównie na celu wzrost funkcjonalności podstawowych cząstek oprogramowania. Generowanie cząstek odbywa się na podstawie odpowiednich modeli, zwiększając produktywność konstruktorów oprogramowania. Wsparcie tego procesu nie jest jednak z reguły kontynuowane na dalszych etapach cyklu życia oprogramowania. Zmiany wprowadzane są w niektórych częściach modelu lub bezpośrednio w wygenerowanym kodzie źródłowym, powodując problemy z utrzymaniem synchronizacji między modelem arzeczywistą aplikacją (Schmidt, 2006).

W dobie szybkich zmian technologii i obszarów zapotrzebowań, możliwość utrzymania spójności między modelem a modernizowanym oprogramowaniem jest

czynnikiem o wzrastającym znaczeniu. Im dłużej zaprojektowana cząstka oprogramowania nadaje się do dalszego wykorzystywania, tym większy jest zwrot nakładów poniesionych na jej wytworzenie. Postulat podniesieniaodporności konkretnej cząstki oprogramowania na zmiany może być rozumiany w kilku aspektach (den Haan, 2008b):

1. **Aspekt ludzki.** Wiedza o budowie systemu lub jego cząstki nie może być przechowywana tylko w głowach jego projektantów. Łatwo można ją utracić w dobie częstych migracji pracowników. Informacja ta powinna więc być stosunkowo łatwo dostępna dla osób nie będących twórcami danej cząstki oprogramowania. Rozwiązaniem idealnym byłoby udostępnienie jej w formie zrozumiałej dla wszystkich zainteresowanych, w tym osób nie będących profesjonalistami w danej dziedzinie.
2. **Aspekt zmiany wymagań.**Coraz szybszy rozwój biznesu IT powoduje częste potrzeby zmian i dodawania funkcjonalności do istniejących już systemów. Wprowadzanie tych zmian nie może mieć znaczącego wpływu na funkcjonowanie systemu, w idealnym przypadku powinny być wprowadzane w sposób niezauważalny dla użytkowników.
3. **Różnorodność środowisk projektowych.** Możliwość wykorzystania danej cząstki oprogramowania w innym środowisku projektowym wiąże się z koniecznością umożliwienia przeniesienia opisującego ją modelu do tego środowiska. Powinien więc on być niezależny od środowisk, w których jest wykorzystywany.
4. **Zmienność środowisk uruchomieniowych.**Nowe wersje platform, serwerów aplikacji, itp. powstają w szybkim tempie. Aby zapewnić jak najdłuższą kompatybilność cząstki oprogramowania z kolejnymi wersjami środowiska, konieczne jest wprowadzenie ochrony przed wpływem zmian w tym środowisku na jej funkcjonowanie. Ten aspekt jest jednym z głównych zadań stawianych przez OMG przed MDA (*ang. Model Driven Architecture*).

Od strony technicznej postulaty stawiane przed MDE sprowadzają się więc do:

1. Przedstawienia cząstek oprogramowania za pomocą zwięzłej i dopasowanej do zadania notacji, często graficznej.

2. Umożliwienia dynamicznego dodawania nowych elementów w trakcie działania aplikacji.
3. Wprowadzenia uniwersalnych formatów opisu cząstek, niezależnych od narzędzi projektowych.
4. Daleko idącej automatyzacji uzyskiwania odpowiednich dla danej platformy cząstek oprogramowania z ich niezależnej od platformy definicji, wykorzystującej mapowania określane przez użytkownika.

Podsumowując, zadaniem MDE jest zwiększeniezarównokrótkoterminowej jak i długoterminowej produktywności w tworzeniu oprogramowania.

2.2 GŁÓWNE NURTY MDE

Technologia MDE nie jest określona żadnym ogólnym standardem. Istnieje kilka podejść mających na celu utworzenie takiego standardu. Dwa główne odłamy, to:

- MDA, zaproponowany i wspierany przez OMG,
- MDSD, będący mniej sformalizowanym, ukierunkowanym na szybkie wprowadzenie do użytkowania.

MDA

Architektura Sterowana Modelami (MDA)jest promowana przez OMG od 2 000 roku. Pierwsza w miarę pełna definicja standardu pojawiła się w roku 2003.

Generalnie MDA jest jednym z podejść do wykorzystywania modeli w produkcji oprogramowania. Standard ten opisuje, jakich rodzajów modeli należy używać, jak je należy przygotować oraz w jakich są między sobą relacjach. Dalekosiężnym celem konsorcjum OMG jest zmiana podejścia do budowy oprogramowania z zadania programistycznego na zadanie projektowe(OMG, 2003).

MDA uwzględnia dwie perspektywy opisu oprogramowania:

- funkcjonalną – opisującą funkcjonowanie i zachowanie systemu na zasadzie „czarnej skrzynki” (ukryta struktura wewnętrzna cząstki oprogramowania),

- konstrukcyjną – opisującą budowę i działanie systemu na zasadzie „białej skrzynki" (jawna struktura wewnętrzna cząstki oprogramowania).

Pierwsza perspektywa jest przydatna w użytkowaniu i kontroli systemu, druga może być wykorzystywana do budowy i wprowadzania zmian w systemie.

Podstawową koncepcją w MDA jest oddzielenie tego, jak działa system od detali związanych ze sposobem, w jaki system wykorzystuje możliwości platformy, na której jest zainstalowany. MDA jest podejściem, w którym systemy są definiowane niezależnie od platform, na których mogą funkcjonować. Jednocześnie pozwala na tworzenie definicji platform i automatyczną transformację niezależnej specyfikacji systemu na dopasowaną do wybranej platformy. Trzema podstawowymi celami stawianymi przed MDA są: przenośność (*ang. portability*), interoperacyjność (*ang. interoperability*) i wielokrotnego użytkowania (*ang. reusability*) dzięki rozdziałowi zainteresowań (*ang. separation of concerns*) na poziomie architektury – logicznemu podziałowi modelu na obszary rozumiane i edytowane przez różne grupy współtworzące oprogramowanie.

MDA określa 3 punkty widzenia systemu: niezależny od sposobu przetwarzania (*ang. computation independent*), niezależny od specyfiki platformy (*ang. platform independent*), uwzględniający specyfikę danej platformy (*ang. platform specific*) (OMG, 2003).

Punkt widzenia niezależny od sposobu przetwarzania skupia się na kontekście i wymaganiach stawianych przed systemem. Detale związane ze strukturą i sposobem przetwarzania danych w systemie są ukryte lub nie zdefiniowane na tym etapie.

Punkt widzenia niezależny od platformy skupia się na sposobie działania systemu, ukrywając detale związane z konkretnym wykonaniem danej operacji na określonej platformie. W tym widoku ukazana jest ta część specyfikacji systemu, która nie podlega zmianom przy przechodzeniu między różnymi platformami. Widok ten może wykorzystywać język ogólnego użytku lub dopasowany do specyfiki zadań, dla rozwiązywania których jest budowany system.

Punkt widzenia specyficzny dla danej platformy dodaje do widoku niezależnego informacje o sposobach wykorzystania funkcjonalnościspecyficznych dla platformy, na której będzie implementowany system(den Haan, 2008c).

Na podstawie tych punktów widzenia, MDA określa 3 podstawowe modele: CIM

(*ang. Computation Independent Model*), PIM (*ang. Platform Independent Model*), PSM (*ang.Platform Specific Model*). Zestawienie uzupełnia model platformy– PDM (*ang. Platform Description Model*)(Stahl i Völter, 2006), (den Haan, 2008c).

Model CIM

Model CIM, jest widokiem na system z punktu widzenia niezależnego od sposobu przetwarzania. Nie pokazuje szczegółów struktury systemu. Często jest nazywany modelem domeny, gdyż opisywany jest z reguły językiem zrozumiałym dla praktyków związanych z dziedziną, w której będzie wykorzystywany system. Użytkownik CIM-a nie musi posiadać wiedzy o sposobie modelowania funkcjonalności systemu, by go rozumieć.

CIM tworzy pomost między ekspertami z dziedziny, dla której jest tworzony system a ekspertami projektowania i budowy systemów IT. Ogranicza to problemy wynikłe ze wzajemnego niezrozumienia wymagań stawianych przed systemem i potencjalnych możliwości oferowanych przez technologię, w jakiej będzie wytworzony.

Model PIM

PIM jest widokiem na system z punktu widzenia niezależnego od platformy.Opisuje on budowę systemu na poziomie ontologicznym, czyli bez ujawniania szczegółów implementacji systemu na konkretnej platformie.

Model ontologiczny jest zdefiniowany jako posiadający następujące zbiory elementów:

- kompozycję - zestawienie elementów z jakiejś kategorii (np. biologicznej, socjalnej, muzycznej itp.),
- środowisko - zestawienie elementów z takiej samej kategorii;kompozycja i środowisko tworzą zbiory rozłączne,
- produkcję - zbiór elementów z kompozycji tworzących coś (produkty lub usługi), co jest dostarczane elementom ze zbioru środowisko,
- strukturę -zbiór więzów interakcji pomiędzy elementami z kompozycji oraz między nimi a elementami ze środowiska.

PIM cechuje kategoria – dziedzina, dla której jest przeznaczone oprogramowanie opisywane przez ten model.

Model PSM

PSM jest widokiem na system z punktu widzenia specyficznego dla danej platformy. Jest on uszczegółowioną wersją PIM, zawierającą elementy i parametry specyficzne dla danej platformy. Przed zdefiniowaniem PSM musi być znany model platformy docelowej.

Model PDM

PDM dostarcza informacji technicznych o częściach składowych platform i udostępnianych przez nie serwisach. Z jego pomocą można zdefiniować specyficzne funkcjonalności dla danej platformy i udostępnić je do wykorzystania w postaci elementów modelu.

Proces budowy MDA

Budowa aplikacji w oparciu o MDA nie sprowadza się tylko do definicji wykorzystywanych modeli. Równie ważna jest definicja metodyki przebiegu tworzenia aplikacji na podstawie poszczególnych modeli.

Proces budowy aplikacji (rys. 2.1) zaczyna się od zdefiniowania CIM,we współpracy z klientem, przez personel zajmujący się stroną biznesową przedsięwzięcia. CIM jest następnie przetwarzany na PIM przez zespół projektowy. W PIM zostaje zawarta wiedza o architekturze systemów bez szczegółów związanych z konkretną platformą. Aby dokończyć proces budowy,dla gotowego PIM trzeba zdefiniować platformę, cowymaga posiadania odpowiedniego modelu platformy (PDM). Transformacja PIM do P SM jest dokonywana pod n adzorem specjalisty od danej platformy. Jeżeli poszczególne modele pośrednie zawierały wszystkie niezbędne informacje, końcowy PSM może być gotową implementacją systemu na danej platformie (den Haan, 2008c).

W przypadku złożonych systemów, bezpośrednie przejście między poszczególnymi rodzajami modeli może nie być możliwe. MDA zakłada możliwość przechodzenia między modelami tego samego typu, różniącymi się poziomem abstrakcji. Ilość transformacji jest dobierana tak, aby umożliwić przejście między końcowymi modelami różnego typu. Ostre przejście między typami modeli jest tu zastąpione serią niewielkich zmian.

Rys. 2.1. Kolejność transformacji między modelami w MDA
Źródło: (MDA Guide V1.0.1, 2003)

W procesie budowy aplikacji z wykorzystaniem MDA każdy etap wiąże się z dodawaniem wiedzy przez inny rodzaj specjalistów. Głównym wyzwaniem technicznym jest tu przechodzenie między poszczególnymi rodzajami modeli. Automatyzacja tego procesu – to, co było dotąd praktycznie niemożliwe z powodu braku formalnych definicji poszczególnych modeli, ma być wg OMG możliwe dzięki odpowiednim mechanizmom definiowanym przez MDA.

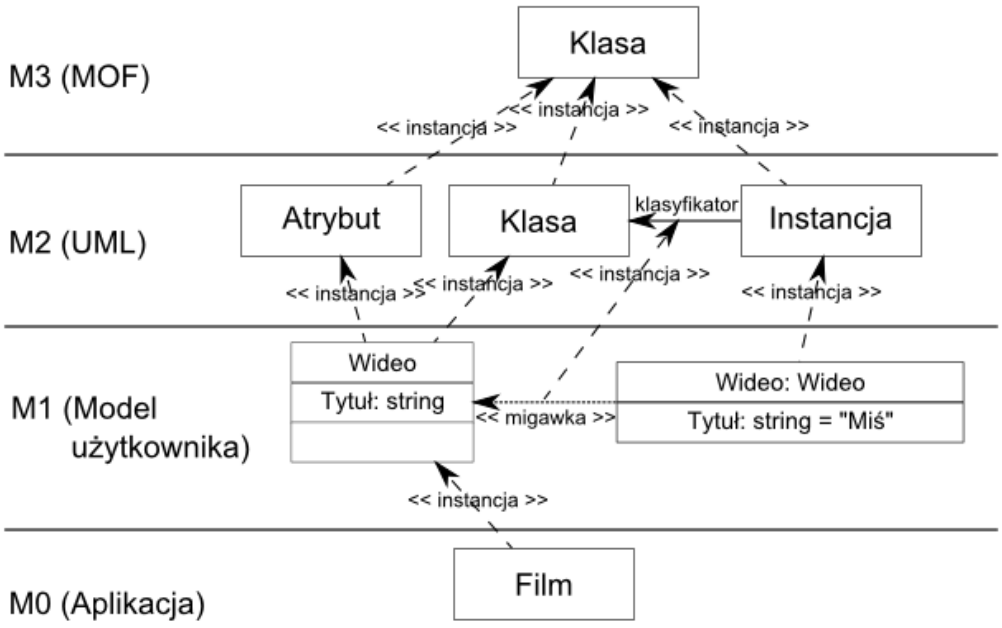
Metamodel

Podstawowym postulatem MDA jest formalizacja definiowania poszczególnych modeli. Proponowana struktura bazuje w dużej mierze na koncepcjach znanych z UML – flagowego projektu OMG.

Ta formalizacja narzuca istnienie ujednoliconego opisu sposobu tworzenia modelu. Taki „model modelu" jest określany nazwą metamodel. Jest on zbiorem koncepcji (elementów, procesów, itp.) w zakresie modelowania pewnej dziedziny. Jeżeli model jest abstrakcją pewnego zjawiska ze świata rzeczywistego, to meta-model jest abstrakcją ukazującą właściwości tego typu modeli, definiującą m.in.

podstawowe elementy, które mogą wystąpić w konkretnym modelu (Stahl i Völter, 2006).

Rys. 2.2 przedstawia czteropoziomową hierarchię UML. Każdy poziom niższy od najwyższego jest określany jako instancja poziomu wyższego. Poziom M0 zawiera rzeczywiste dane użytkownika, poziom M1 jest modelem tych danych, poziom M2 jest modelem określającym sposób budowania M1 (metamodelem). Poziom M3 określa sposób w jaki definiowany jest M2, czyli zawiera szablon (meta-metamodel) budowy M2(den Haan, 2008d).



Rys. 2.2. Hierarchia UML
Źródło: (OMG, 2007)

Pojęcie metamodelu może być rozumiane w dwóch kontekstach: lingwistycznym i ontologicznym (den Haan, 2008d).

Metamodel lingwistyczny A modelu B może być rozumiany jako model języka modelowania, wykorzystywanego w B. Elementy modelu B są lingwistycznymi instancjami elementów modelu A. Elementy modelu A, będące lingwistycznymi

typami, tworzą język definicji prawidłowych wyrażeń tego języka. Element modelu B, jest instancją lingwistycznego typu T, jeśli należy do zbioru możliwych rozszerzeń znaczenia T. Metamodel lingwistyczny definiuje również semantyki modelu: typów, statyczną i dynamiczną.

Przykładem metamodelu lingwistycznego jest specyfikacja BPMN (*ang. Business ProcessModelingNotation*) definiująca:

- różne rodzaje elementów i związków (składnia abstrakcyjna i semantyka typów),
- sposoby połączeń między elementami (semantyka statyczna),
- podstawowe zachowanie elementów, np. „zdarzenia oddziaływują na przepływ" (semantyka dynamiczna).

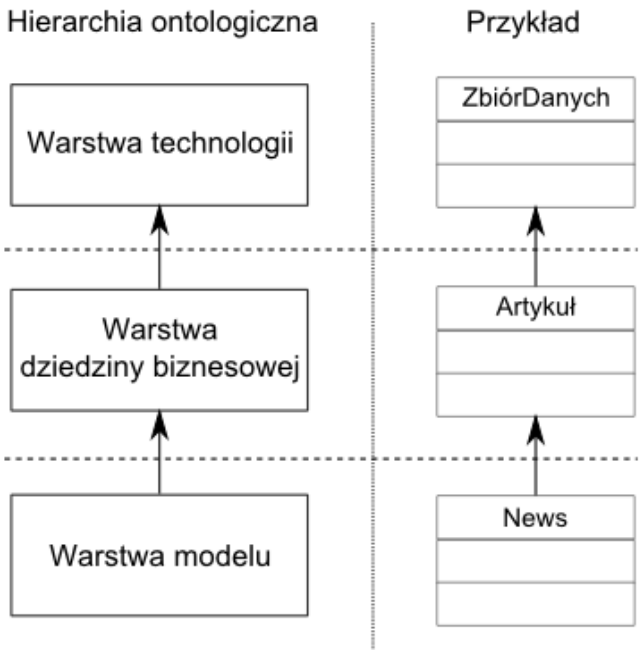
Metamodele ontologiczne, w przeciwieństwie do metamodeli lingwistycznych, definiują język modelu oraz określają jego naturalną semantykę. Naturalna (wrodzona) semantyka opisuje „wewnętrzne znaczenie" modelowanych zasobów i dostarcza podstaw do wnioskowania na temat pojęć. Przykładowo, taksówkarz jest człowiekiem, który może być kobietą lub mężczyzną oraz wozi pasażerów jakimś samochodem.

Ontologia jest systemem pojęć i ich relacji (często określanym nazwą „model domeny"), w którym wszystkie pojęcia są zdefiniowane w sposób deklaratywny. System definiuje słownik danej domeny oraz zbiór ograniczeń, w jaki sposób terminy ze słownika mogą być łączone by zamodelować domenę. Element może być ontologiczną instancją innego elementu tylko wtedy,gdy istnieje relacja między ich znaczeniami, np. „Szarik" jest ontologiczną instancją „Owczarka niemieckiego", który jest ontologiczną instancją „Rasy".

Metamodele ontologiczne mogą tworzyć hierarchie (rys. 2.3), w których najwyższy poziom opisuje abstrakcyjne pojęcia technologiczne, nie powiązane z żadną konkretną platformą (PIM). Przykładem może być *ZbiórDanych* określający trwałą (zapisaną w sposób nieulotny) cząstkę danych. Kolejnym poziomem jest warstwa biznesowa, gdzie instancją zbioru danych może być np. artykuł umieszczany w pewnym serwisie. Ta warstwa jest wykorzystywana jako specyficzny dla domeny język modelowania. Najniższy jest poziom modelu, gdzie instancją artykułu jest news umieszczany na stronie głównej serwisu.

Samo istnienie metamodelu(sformalizowanie sposobu tworzenia modelu) nie wystarcza do automatyzacji procesu transformacji pomiędzy modelami. Konieczne

jest również ujednoliceniei sformalizowanie sposobu definiowania różnych modeli. Opis sposobu definiowania modeli danego typu nazywany jest meta-metamodelem. OMG zaleca wykorzystywanie MOF (*ang. Meta Object Facility*).MOF jest określany jako język typu specyficznego dla domeny definiowania metamodeli (OMG, 2011).



Rys. 2.3. Hierarchia ontologiczna
Źródło: (Gitzel i Korthaus, 2004)

MOF wykorzystuje język XMI (*ang. XML MetadataInterchange*) do zapisu i wymiany informacji poprzez XML, co w założeniach OMG ma umożliwić transfer modeli i metamodeli między rożnymi narzędziami MDA.

Transformacje modeli

Podstawowym wyzwaniem w MDA jest transformacja między poszczególnymi poziomami modeli, co wiążesię ze zdefiniowaniemzestawu zasad transformacji. Metody definicji takich zasad mogą się od siebie różnić wnastępujących

aspektach(den Haan, 2008a):

1. Zasady transformacji. Opisują jak należy przetłumaczyć elementy modelu źródłowego na elementy modelu docelowego. Składają się z dwóch części: lewej, pobierającej elementy modelu źródłowego oraz prawej, generującej elementy modelu docelowego. Można je definiować za pomocą różnych środków,jak np. zapytania, zmienne, szablony itp.
2. Zasięg stosowalności zasady. Pozwala na określenie zasięgu, obszaru modelu podlegającego danej zasadzie transformacji. Ograniczenia mogą być narzucone na model źródłowy, jak i docelowy. Określanie zasięgu stosowalności zasady może być przydatne przy definiowaniu zaawansowanych struktur (zestawów reguł) transformacji.
3. Zasady przetwarzania modelu źródłowego i docelowego. W sytuacji, gdy transformacja nie polega na zbudowaniu od zera modelu docelowego na podstawie źródłowego, a np. definiuje metody propagacji zmian wprowadzonych w jednym z modeli, konieczne jest zdefiniowanie,czy zmiany mogą być destruktywne - powodować usunięcie elementów modelu docelowego.
4. Strategia wprowadzania zasady. Zasada musi dotyczyć konkretnego miejsca w swoim zasięgu stosowalności. Istnieje możliwość zaistnienia kilku potencjalnych miejsc stosowania zasady w zasięgu stosowalności dla modelu źródłowego. W takich przypadkach jest konieczne zdefiniowanie strategii wprowadzania zasady (kolejność, wzajemne wykluczenia, itp.).
5. Kolejkowanie zasad. W rozbudowanym systemie ilość zasad będzie znaczna. Może być konieczne wprowadzenie mechanizmu regulacji kolejności egzekwowania zasad. Algorytm kolejkowania może, ale nie musi uwzględniać wyborów użytkownika.
6. Schemat budowy zasad. Może się różnić w zależności od implementacji. Generalnie wyróżnia się trzy główne obszary zmienności:
   - modułowość – różne sposoby zbierania zasad w moduły,
   - mechanizmy wielokrotnego użycia – definicje zasad bazujące na innych zasadach,
   - struktura zorganizowania – zorganizowanie zasad bazujące na jednym z modeli, czy innym niezależnym układzie.

7. Zapisywanie działań. Dodatkowym rezultatem transformacji może być informacja o źródłowych i docelowych elementach uczestniczących w transformacji. Jest to przydatne w takich aspektach, jak utrzymanie synchronizacji między modelami, debugowanie na p odstawie modelu, czy po prostu odnajdywanie zmian wprowadzonych przez daną transformację.
8. Kierunkowość transformacji. W zależności od różnych podejść, transformacja może być jedno lub dwukierunkowa. Dwukierunkowa transformacja umożliwia zarówno generowanie modelu o niższym poziomie abstrakcji, jak i propagację zmian, które nastąpiły w modelu szczegółowym, na model bardziej abstrakcyjny. Takie „wstępowanie w górę" jest znacznie trudniejsze do zdefiniowania.

Tworzenie zasad transformacji zostało zdefiniowane w MDA jako kilka różnych podejść:

1. Wstawianie znaczników do modelu.
2. Tworzenie zasad transformacji na podstawie typów modeli.
3. Mapowanie wzorców.

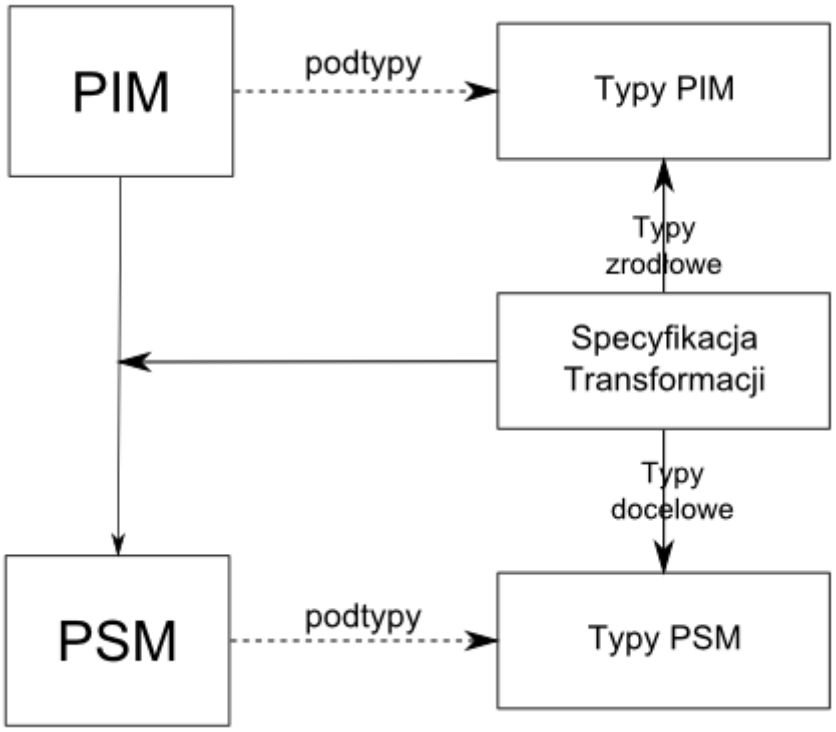
Wstawianieznaczników do modelu

Znacznikami nazywane są dodatkowe informacje wstawiane do modelu źródłowego, które kontrolują przebieg transformacji. Informacje te z reguły są wytwarzane na podstawie metamodelu docelowego. Model docelowy może oferować kilka rozwiązań realizacji elementu modelu źródłowego. Ich lista jest dostarczana przez metamodel docelowy. Wstawienie (przez projektanta lub automatycznie) do modelu źródłowego znacznika określającego wybrane rozwiązanie wymusza wybór konkretnej ścieżki podczas procesu transformacji.

Model źródłowy najczęściej nie jest oznaczany bezpośrednio, aby zachować niezależność parametrów specyficznych dla modelu docelowego. Tworzona jest kopia pośrednia modelu źródłowego, zawierająca znaczniki i odwołania do elementów oryginału. Taka konstrukcja upraszcza propagację zmian w modelu źródłowym, zmiany należy przenieść do kopii tylko wtedy, gdy dotyczą one oznaczonych elementów(Stahl i Völter, 2006).

Tworzenie zasad transformacji na podstawie typów modeli

Metoda ta bazuje na założeniu, że wszystkie elementy modelu źródłowego czy docelowego są podtypami typów określonych w modelu typów dla danego modelu. Zdefiniowany jest zestaw reguł mapujący bezpośrednio typ ze źródłowego modelu typów na typ z docelowego modelu typów. Mapowanie to może być proste (1:1) lub złożone (1:N).



Rys. 2.4. Metoda specyfikacji transformacji na podstawie typów elementów modeli
Źródło: (den Haan, 2008a)

Tylko w pierwszym przypadku można łatwo zdefiniować transformację dwukierunkową. Niezależnie od typu mapowania, dla każdego elementu modelu źródłowego jest: pobierana reguła pasująca do jego typu, odczytywany typ elementu docelowego, generowany element docelowy (den Haan, 2008a).

Model typów źródłowy i docelowy może być w ogólniejszym przypadku zastąpiony

odpowiednimi metamodelami. Wtedy zestaw reguł określa mapowania pojęć między metamodelami w bardziej ogólny sposób (dowolne kombinacje elementów, procesów, itd.), co znacząco zwiększa stopień komplikacji automatycznej realizacji transformacji.

Mapowanie wzorców

Mapowanie wzorców rozszerza metodętworzenia zasad transformacji na podstawie typów modeli o mapowanie specyficzne dla określonych grup elementów (pewnych typów) modelu źródłowego. Zmienia to zasięg reguł z całego modelu na obszary konkretnych wzorców elementów(den Haan, 2008a).

Tylko niewielka grupa narzędzi MDA w pełni implementuje transformacje między metamodelami. Pozostałe narzędzia stosują różnego rodzaju techniki bezpośredniego generowania kodu źródłowego aplikacji (np. na zasadzie szablonów) bez zaznajamiania generatora z metamodelem docelowego języka programowania. Do opisu szablonów stosowane są języki transformacji takie, jak:TCL, JSP, XSLT itp. (Stahl i Völter, 2006).

MDSD

W odróżnieniu od MDA, głównymi celami stawianymi przed MDSD jest zwiększenie efektywności tworzenia oprogramowania, jego jakości oraz możliwości ponownego użycia. Zastosowanie MDSD ma spowodować zwolnienie twórcy oprogramowania z wykonywania nudnej, podatnej na błędy, rutynowej pracy(Stahl i Völter, 2006).

Wymaga to dążenia do wykorzystywania sformalizowanych i dokładnie dopracowanych architektur oprogramowania. Im dokładniej architektura taka została opisana, tym bardziej schematyczny stanie się kod aplikacji wykorzystujący tę architekturę. Z drugiej strony, architektura stosująca niezależne od siebie opisy aspektów tworzenia aplikacji, może być wykorzystana do stworzenia tej samej aplikacji na rożne sposoby, dając w rezultacie zupełnie różne aplikacje (Schmidt, 2006).

Zakładając, że architekci oprogramowania stworzą zestaw referencyjnych implementacji – konkretnych realizacji najważniejszych aspektów danej architektury

oprogramowania, twórcy aplikacji mogą je użyć jako swego rodzaju implementacje wzorcowe wymagające jedynie drobnych modyfikacji. Te same realizacje techniczne powtarzają się w wielu miejscach i aplikacjach (np. użycie MVC). Posiadanie repozytorium rozwiązań właściwych dla danej architektury, umożliwia budowę aplikacji na podstawie gotowych szablonów oraz komponentów wielokrotnego użycia. Zwiększa się tym samym efektywność w stosunku do programowania „od podstaw".

Im dokładniej opisana zostanie dana architektura oprogramowania, tym bardziej schematyczne i powtarzalne stanie się programowanie z jej użyciem. Opiera się ono na metodzie kopiuj-wklej, poszerzonej o modyfikacje wynikające ze specyfiki danej domeny, dla której tworzona jest aplikacja. Z tego powodu nie wymaga dużego wkładu intelektualnego, ani posiadania szerokiej wiedzy o strukturze danej architektury. Co więcej, cały ten proces można zrzucić na barki generatora.Architektura specjalnie przystosowana do takiego zadania, opisująca wszystkie definicje swoich detali w formie programów, nosi nazwę generatywnej architektury oprogramowania (*ang. Generative Software Architecture*)(Völter, 2005).

MDSD zakłada tworzenie modelu CIM/PIM, ale w przeciwieństwie do MDA pomija pośrednią transformację do modelu PSM, przechodząc od razu do kodu aplikacji. Zabieg ten jest celowy i wynika z podejścia praktycznego, gdzie uzyskane w ten sposób uproszczenie całego procesu (nie trzeba kontrolować i manipulować wynikami pośrednich przejść) jest uznane za bardziej wartościowe od potencjalnie większej swobody uzyskanej dzięki PSM. Eliminacja modeli pośrednich zapobiega również potencjalnym problemom niespójności, gdy wprowadzona w modelu pośrednim zmiana nie może być automatycznie przeniesiona na wyższy poziom.

W praktycznym podejściu, prezentowanym przez MDSD, rezygnuje się z możliwości automatyzacji transformacji powrotnej z kodu do modelu abstrakcyjnego. Jest to skutkiem przekonania, że problem transformacji powrotnej jest nierozwiązywalny w ogólnym przypadku, gdyż dla arbitralnie wybranego kawałka kodu może nie istnieć odpowiadający mu element (bardziej abstrakcyjnego) modelu PIM. Stworzenie modelu PIM posiadającego elementy odpowiadające wszystkim dowolnie wybranym kawałkom kodu, sprowadziłoby go na ten sam poziom abstrakcji, co kod źródłowy.

MDSD prezentuje więc podejście inżynierii normalnej (*ang. forward engineering*),

gdzie wszystkie zmiany projektowe muszą być dokonywane w modelu, a nie w kodzie źródłowym. Nie oznacza to konieczności zaprojektowania od razu modelu całej aplikacji. Możliwe i preferowane jest podejście inkrementacyjne, wykonujące wiele iteracji: modyfikacja modelu + generacja kodu. Model przestał więc pełnić rolę wspierającą dla języka programowania, sam stał się takim językiem.

2.3 DSL

Cechą wspólną wszystkich podejść MDE jest konieczność opisu modelu za pomocą konkretnej notacji bądź języka modelowania. Języki te są z założenia dopasowywane do rozwiązywania określonej dziedziny problemów (występujących w danej domenie) oraz ich określonej reprezentacji.Tego typu „wyspecjalizowane" języki są nazywane językami dziedzinowymi - DSL (*ang. DomainSpecific Language*). Mogą one mieć formę zarówno tekstową, jak i graficzną. Przeciwieństwem języków dziedzinowych są języki programowania ogólnego zastosowania – GPL (*ang. General Purpose Language*).

Język dziedzinowy wykorzystuje do opisu problemu ontologię danej dziedziny - definiując aplikację internetową korzysta się z formularzy, list, menu, itp., a definiując system agrotechniczny używa się pojęć pole, uprawa, itd.. Mając model opisany w języku dziedzinowym, można zdefiniować sposób przekształcania tego opisu w domyśle na kod w języku GPL.Prawidłowo zaprojektowany język DSL dopuszcza istnienie składni roboczych (dialektów języka)oraz zawiera opisy mapowań i semantyki(znaczenia poszczególnych symboli oraz ich funkcji w programie)(Völter, 2005).

Model określony za pomocą DSL-a jest określony mianem DSM (*ang. DomainSpecific Model*). Rozbudowany system informatyczny wymaga z reguły wykorzystania kilku rodzajów DSM-ów do jego opisania, każdy zdefiniowany za pomocą innego języka DSL. Modele te odnoszą się wzajemnie do siebie i muszą być połączone w chwili generowania kodu wykonawczego aplikacji. Określenie odpowiedniej struktury przestrzeni modelowania ułatwia dobranie i zarządzanie kolekcją DSM-ów.

2.4 ZALETY I WADY PROJEKTOWANIA STEROWANEGO MODELAMI

MDE może wydawać się tylko kolejnym sposobem projektowania aplikacji, podobnym w swoich założeniach do UML. Na dodatek proponowany system wywraca porządek organizacyjny, przenosząc ciężar koncepcyjnej budowy aplikacji na pracowników nie koniecznie należących do grona ekspertów danej technologii. Rodzi to zrozumiałą niechęć tego grona do „oddawania pola" laikom.

W rzeczywistości zalety i wady podejścia MDE rozkładają się nieco inaczej. Johan den Haan, jeden z głównych promotorów i praktyków projektowania sterowanego modelami - MDD (*ang. Model Driven Development*), przedstawia w swoim blogu 15 powodów, dla których warto rozważyć wykorzystywanie MDD(den Haan, 2009b). Ich skrócona wersja jest przedstawiona poniżej:

1. MDD jest szybsze

   Wyższy poziom abstrakcji modelu powoduje, że pojedynczy element odpowiada kilku(nastu) linijkom kodu. Są one generowane automatycznie na podstawie modelu. Wykorzystując elementy modelu można więc szybciej zaprojektować daną funkcjonalność.
2. MDD obniża koszty

   Powodem jest po pierwsze krótszy czas dostarczenia aplikacji na rynek, po drugie obniżenie kosztów dzięki wykorzystaniu mniejszej ilości mniej wyspecjalizowanych pracowników (nie licząc początkowego kosztu wytworzenia środowiska MDD i zapoznania z nim personelu). Zmniejszają się również koszty wprowadzania zmian – łatwiej zrozumieć i modyfikować model o wyższym poziomie abstrakcji.
3. MDD sprzyja poprawie jakości

   Techniczna jakość aplikacji zależy od jakości generatora. Z założenia jego budową i poprawianiem zajmują się najlepsi specjaliści. Wprowadzone przez nich najlepsze praktyki budowy oprogramowania będą automatycznie powielane przez wszystkich projektantów aplikacji.
4. MDD jest mniej podatne na błędy

   Testowanie aplikacji sprowadza się do testowania jej funkcjonalności. Problemy techniczne są wychwytywane podczas budowy generatora. Przeoczony błąd techniczny wymaga poprawy generatora i ponownego wygenerowania aplikacji.

Nie ma potrzeby analizowania i poprawiania poszczególnych aplikacji.

5. MDD ułatwia walidację

   MDD umożliwia walidację na poziomie abstrakcji modelu. W jej ramach można wychwytywać błędy funkcjonalne i generować komunikaty operujące językiem danej dziedziny – funkcjonalność nieosiągalna w przypadku standardowego języka programowania.

6. Oprogramowanie stworzone w oparciu o MDD jest mniej zależne od zmian personalnych

   Zmiana specjalisty zajmującego się aplikacją jest łatwiejsza. Nie musi posiadać wysokich umiejętności technicznych, a poza tym łatwiej mu zrozumieć model aplikacji oznacznym poziomie abstrakcji.

7. MDD wykorzystuje wiedzę ekspertów z danej dziedziny

   MDD umożliwia ekspertom z danej dziedziny bezpośrednie włączenie się w tworzenie aplikacji. Mogą oni zaprojektować rozwiązanie programistyczne poprzez przeniesienie swojej wiedzy o domenie na model o odpowiednio wysokim poziomie abstrakcji. Możliwość budowania zaawansowanych aplikacji nie jest już ograniczona do „elitarnej" grupy programistów.

8. MDD docenia ekspertów technicznych

   Zaawansowani programiści mogą skupić się na wytwarzaniu unikalnych rozwiązań wykorzystywanych przez generator, pozostawiając powtarzalną pracę mniej doświadczonym członkom zespołu.

9. MDD jest pomostem między biznesem a dostawcami oprogramowania

   Synchronizacja między potrzebami biznesowymi a rozwiązaniami technicznymi jest zawsze ważnym aspektem wytwarzania aplikacji. MDD ułatwia utrzymanie tej synchronizacji: angażując ekspertów z danej dziedziny w proces budowy oprogramowania, przedstawiając modele zawierające koncepcje z danej dziedziny, wreszcie dzięki szybszemu generowaniu oprogramowania pozwalając na szybkie uzyskanie opinii zwrotnej użytkowników.

10. MDD tworzy oprogramowanie odporniejsze na zmiany wymagań biznesowych

    Współcześnie wymagania biznesowe wobec rozwiązań informatycznych zmieniają się w tempie szybszym niż są wprowadzane. MDD przynajmniej częściowo rozwiązuje ten problem, umożliwiając szybsze tworzenie oprogramowania oraz

ułatwiając modelowanie zmian. Wreszcie, gdy nowe wymagania wiążą się z konkretnymi zmianami modelu, możliwa jest automatyzacja procesu ich przeniesienia na aplikację.

11. MDD tworzy oprogramowanie odporniejsze na zmiany technologii
    Zmiany technologii związanych z wytwarzaniem oprogramowania następują z coraz większą częstotliwością. Struktura MDD zapewnia niezmienność modelu aplikacji w sytuacji zmian wykorzystywanych rozwiązań technologicznych. Zmiany są wprowadzane jedynie w generatorze kodu. Po ich wprowadzeniu wszystkie modele aplikacji mogą być przetworzone na kod dopasowany do nowej technologii.
12. MDD wymusza stosowanie konkretnej architektury oprogramowania
    Nawet, gdy przedsiębiorstwo przestrzega stosowania konkretnej architektury w wytwarzanym oprogramowaniu, nie ma całkowitej pewności jej wykorzystania we wszystkich cząstkach oprogramowania z powodu ręcznego tworzenia odpowiedniego kodu. MDD niejako gwarantuje zgodność wszystkich elementów z wybraną architekturą poprzez eliminację ręcznego wprowadzania/modyfikowania kodu elementu.
13. MDD umożliwia przechowywanie wiedzy specyficznej dla domeny
    Wiedza specyficzna dla domeny, dostępna najczęściej tylko ekspertom z danej dziedziny, jest zapisywana w postaci abstrakcyjnych modeli podczas projektowania aplikacji. Zapobiega to możliwości utratytejwiedzypozakończeniuprojektu.
14. MDD dostarcza aktualną dokumentację
    Model jest dokumentacją, co gwarantuje, że nigdy nie będzie ona przestarzała lub niekompletna. Odpowiednio wysoki poziom abstrakcji czyni ją czytelną dla ekspertów dziedzinowych oraz klientów.
15. MDD pozwala na skupienie się na problemach biznesowych
    MDD separuje sposób rozwiązania problemu biznesowego od technologii, w jakiej to rozwiązanie zostanie docelowo wykonane. Problemy doboru odpowiednich technologii nie dotyczą więc konkretnej aplikacji, a całej domeny, dla której zostało przygotowane narzędzie. Optymalny dobór jest automatycznie powielany we wszystkich aplikacjach dla tej domeny.

Powyższe zestawienie prezentuje wizję możliwości MDD brzmiącą jak prospekt reklamowy. Dla kontrastu przedstawiono poniżej, zdefiniowane również przezJohana den Haan, 8 niebezpieczeństw wynikających ze stosowania MDD(den Haan, 2009c):

1. MDD usztywnia sposób programowania
   W przeciwieństwie do klasycznych metod wytwarzania oprogramowania, MDD wymaga od projektanta stosowania się do sztywnych form i ograniczeń. Programowanie na wyższym poziomie abstrakcji pozwala na automatyczną generację, ale jednocześnie bardzo ogranicza możliwość dowolnej modyfikacji każdego detalu aplikacji. Oznacza to występowanie efektu sztampowości, np. interfejsu użytkownika.
2. ModeleMDD oferują tylko tyle elastyczności, ile jej zaprojektowano
   Pierwszym ograniczeniem jest używane narzędzie MDD,po drugie projektant ma tylko tyle elastyczności w tworzeniu modelu, ile przewidziano w wykorzystywanym języku DSL. Im wyższy poziom abstrakcji, tym więcej uogólnień. Elastyczność jest dostępna tylko tam, gdzie ją celowo wprowadzono – nie koniecznie tam, gdzie jest potrzebna.
3. MDD zmienia wymagania stawiane przed twórcami oprogramowania
   W przeciwieństwie do metod tradycyjnych, część programistyczna jest realizowana przez wąską grupę ekspertów IT („meta-team"). Budowa rozwiązania biznesowego jest natomiast wykonywana przez „biznes-inżynierów", a nie programistów. Te osoby muszą zarówno rozumieć specyfikę swojej domeny, jak i potrafić przedstawić ją za pomocą formalnego modelu. W zależności od dziedziny, może być trudno o osobę potrafiącą obie te rzeczy naraz.
4. Środowisko modelowania MDD najczęściej nie wspiera kontroli wersji
   Istniejące rozwiązania kontroli wersji radzą sobie świetnie z tekstowymi językami programowania. W przypadku języków graficznych brak jest jak dotąd dobrych rozwiązań kontroli wersji. W przypadku dużego zespołu projektowego może to się stać poważnym problemem.
5. Istnieje tendencja do wykorzystywania „prawie ukończonych" środowisk MDD
   Wykorzystywanie nowego, niesprawdzonego narzędzia w dużym projekcie wiąże się ze znacznym ryzykiem wystąpienia nieprzewidzianych komplikacji. Większość udostępnianych narzędzi MDD jest jeszcze niedojrzała i nie posiada wielu

pozytywnych referencji.

6. Osoby negocjujące z klientem zakres wymagań dla aplikacji muszą znać ograniczenia technologii

   W związku z ograniczeniami wykorzystywanego DSL, istnieje niebezpieczeństwo, że konsultant nieznający jego ograniczeń ustali z klientem rozwiązanie niewykonalne w danym narzędziu MDD.Rezultatem będzie konieczność znaczących ingerencji w wygenerowany kod i utrata większości zalet MDD.

7. Oferowanie rozwiązań MDD nie posiadając doświadczonego zespołu

   MDD staje się modne, istnieje więc pokusa oferowania tego typu rozwiązań bez odpowiedniego przygotowania zespołu. W rezultacie zespół realizujący projekt i jednocześnie uczący się technologii/narzędzia ma duże szanse nie dotrzymania terminów, a wykonana aplikacja może nie spełniać wszystkich wymagań klienta. Dodatkowo wprowadzanie zmian i poprawek może nie być takie łatwe z powodu niewłaściwego użycia narzędzia, czy braku znajomości jego ograniczeń.

8. Nowinki pochłaniają czas

   Zespół rozpoczynający użytkowanie nowego, innowacyjnego narzędzia ma tendencję do zajmowania się wyszukiwaniem i testowaniem wszystkich „fajnych" możliwości nowej technologii, odkładając na bok realizację zleconego projektu.

Technologie MDE są jeszcze we wczesnej fazie rozwoju, jednak widoczne są już trendy do ich wykorzystywania w nowoczesnych podejściach do budowy aplikacji. Narzędzia deweloperskie, implementujące te techniki, przechodzą powoli transformację z wczesnych niedojrzałych aplikacji do w pełni profesjonalnych rozwiązań oferujących wsparcie dla całego procesu budowy aplikacji w technologii MDE.

Mimo dużych oporów w środowisku IT, związanych z zupełną zmianąprzyzwyczajeń, technologiami tymi zajmuje się coraz większe grono instytucji odpowiedzialnych za wytyczanie kierunków rozwoju budowy i konserwacji oprogramowania.

2.5 PYTANIA KONTROLNE

1. Jakie są podstawowe cele, które stawia przed sobą technologia wytwarzania aplikacji sterowanego modelami?
2. Jakie są różnice między poszczególnymi typami modeli stosowanych w MDA?
3. Jakie znasz metody transformacji między modelami?
4. Wymień po 5 głównych zalet i wad stosowania technologii MDE.

3

WebML jako metodyka modelowania aplikacji internetowych

Cel

Opisanie genezy WebML oraz przykładowego cyklu życia aplikacji internetowej tworzonej przy pomocy narzędzi Inżynierii Sterowanej Modelami. Przedstawienie modelu danych, hipertekstu oraz warstwy prezentacji aplikacji wraz z ich taksonomią.

Plan

1. Wprowadzenie do WebML
2. Cykl życia aplikacji internetowej tworzonej w oparciu o MDE
3. Elementy specyfikacji wymagań wprowadzone przez WebML
4. Elementy modelu danych WebML
5. Elementy modelu hipertekstu WebML
6. Warstwa prezentacji aplikacji zarządzającej danymi

3.1 WPROWADZENIE DO MODELOWANIA

GENEZAWEBML

PoczątkiWebML sięgają 1996 roku, kiedy to pracownicy Politechniki w Mediolanie rozpoczęli pracę nad narzędziem CASE (*ang. Computer-Aided Software Engineering*)o nazwie AutoWeb. W 1998 roku WebML stał się częścią projektu W3I3 ESPIRIT, wspieranego przez EuropeanCommunity, w którym uczestniczyła Politechnika w Mediolanie, TXT e-solutions z Włoch, KPN Research z Holandii, Digia Inc. z Finlandi oraz Otto Versand z Niemiec.

WebML, to nie tylko notacja, czyli zbiór komponentów i modeli, ale również metodyka tworzenia aplikacji internetowych zarządzających dużymi ilościami danych, którapocząwszy od roku 1999 znalazła zastosowanie w procesie tworzenia przemysłowych aplikacji internetowych m.in. na mocy umów z firmami Microsoft, Cisco Systems, TXT e-solution i Acer Europe. W 2001 roku zespół twórców WebML założył firmę, której celem jest rozwój, dystrybucja i marketing środowiska IDE Webratio– następcy narzędzia AutoWeb(Kęsik i Żyła, 2008).

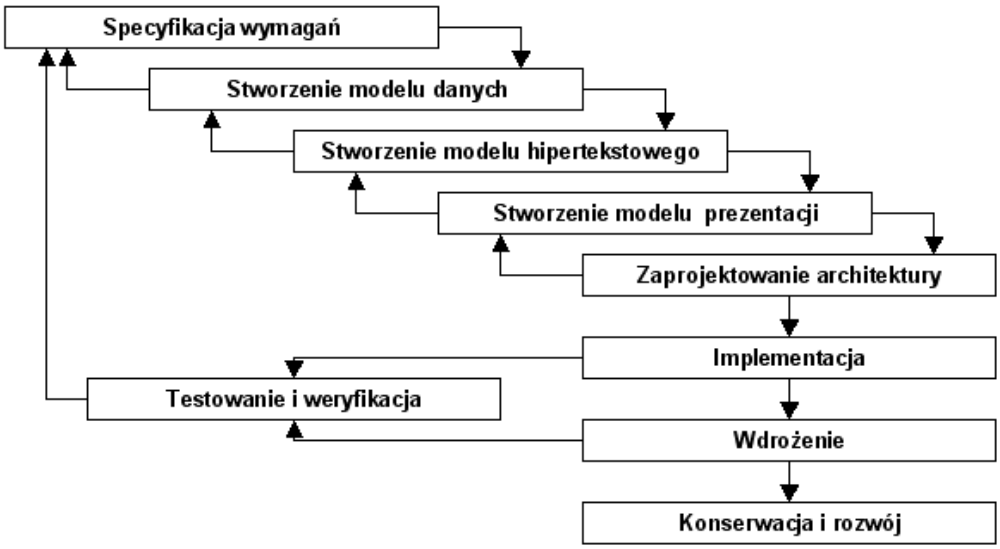
WebMLbywa również opisywany jako: graficzna notacja, służąca określaniu treści, zależności i nawigacji pomiędzy stronami aplikacji internetowej, składająca się z modelu danych, modelu hipertekstu oraz modelu prezentacji. Wymienione modele mogą być postrzegane jako wysokopoziomowa graficzna reprezentacja aplikacji internetowej w architekturze MVC (Żyła i Kęsik, 2010a).

CYKL ŻYCIA APLIKACJI INTERNETOWEJ WEDŁUG WEBML

MetodykaWebML nie ma na c elu zdefiniowania nowego cyklu życia aplikacji, lecz wykorzystuje rozwiązania już istniejące, dostosowując je do potrzeb aplikacji internetowych zarządzających dużymi ilościami danych. W ich ramach wyróżnia wejścia procesu projektowania w postaci wymagań biznesowych i stałych środowiskowych oraz wyjścia, czyli gotową aplikację działającą na określonej platformie sprzętowej i programowej, razem z dokumentacją i instrukcją obsługi.

Na rysunku 3.1 przedstawiono cykl życia aplikacji dostosowany do specyfiki

modelowania (przy pomocy WebML) aplikacji internetowych zarządzających dużymi ilościami danych. Proces tworzenia aplikacji odbywa się w postaci cyklów, których efektem jest opracowanie prototypu lub fragmentu aplikacji, który pozwala na przeprowadzenie testów oraz ewaluacji. Kolejne cykle następują po sobie w sposób iteracyjnyi inkrementacyjny(Strona domowa metodyki WebML, 2011).



Rys. 3.1. Propozycja cyklu życia aplikacji internetowej modelowanej przy pomocy WebML

Źródło: (Strona domowa metodyki WebML, 2011)

W ramach przedstawionego na rysunku 3.1 cyklu życia aplikacji internetowej wyróżnia się następujące etapy(Ceri i inni, 2003):

1. Specyfikacja wymagań –określenie wymagań funkcjonalnych i niefunkcjonalnych, jakie ma spełniać tworzona aplikacja.
2. Stworzenie modelu danych – zaprojektowanie warstwy danych, z której będzie korzystać aplikacja.
3. Stworzenie modelu hipertekstowego – zaprojektowanie: logiki biznesowej, rozmieszczenia treści oraz przepływu sterowaniapomiędzy stronami aplikacji.
4. Stworzenie modelu prezentacji – określenie sposobu prezentacji treści, innymi słowy określenie wyglądu stron aplikacji, które są prezentowane użytkownikowi.

5. Zaprojektowanie architektury–określenie uwarunkowań sprzętowych, sieciowych i programowych, które składają się na środowisko, w którym działa aplikacja.
6. Implementacja – odwzorowanie elementów warstwy danych w jednym lub więcej źródłach danych oraz wygenerowanie plików składających się na aplikację na podstawie jej modelu.
7. Testowanie i weryfikacja – sprawdzenie użyteczności, poprawności oraz wydajności aplikacji oraz jej zgodności z rezultatami/produktami specyfikacji wymagań.
8. Wdrożenie–umieszczenie aplikacji w ramach zaprojektowanej dla niej architektury, tj. przygotowanie bazy danych oraz umieszczenie plików aplikacji na serwerze aplikacji, w celu udostępnienia użytkownikom jej usług.
9. Konserwacja i rozwój – modyfikowanie wdrożonej aplikacji oraz opisującej ją dokumentacji w celu zapewnienia poprawności oraz ciągłości jej działania lub zmiany realizowanej przez nią funkcjonalności.

3.2 NOWE ELEMENTY SPECYFIKACJI WYMAGAŃ FUNKCJONALNYCH

Proces specyfikacji wymagań składa się z dwóch etapów: zbierania informacji o dziedzinie aplikacji i jej spodziewanej funkcjonalności oraz analizy zebranych informacji. Wejściamiprocesu są wymagania biznesowe oraz wszystkie dostępne informacje dotyczące: technologii, środowiska działania, przetwarzanych danych oraz specyfiki pracy jej użytkowników.Produktami zaś, czyli wyjściami procesu, są m.in.:

1. Specyfikacjagrupużytkowników.
2. Specyfikacjaprzypadkówużycia.
3. Specyfikacjasłownikaobiektów.
4. Specyfikacjawidoków witryny.
5. Szkic (zarys) wygląduaplikacji.
6. Specyfikacjawymagańniefunkcjonalnych.

Jedynie pierwsze cztery pozycje z przedstawionej listy zostały dostosowane do specyfiki wytwarzania oprogramowania według metodyki WebML. Pozostałe produkty są dobrze sprecyzowane przez inne istniejącemetodyki, które mogą być stosowane

niezależnie od wykorzystania inżynierii sterowanej modelami w wytwarzaniu oprogramowania. Ponadto nie ma przeciwskazań, aby produkty specyfikacji rozszerzyć o pozycje nie opisane przez metodykę WebML(Żyła i Kęsik, 2010).

SPECYFIKACJA GRUP UŻYTKOWNIKÓW

Specyfikacja grup użytkowników zawiera charakterystykę, wstępną hierarchię oraz uprawnienia grup użytkowników, które będą korzystały z aplikacji. Każda ze zidentyfikowanych grup jest opisywana,w sposób analogiczny do przedstawionego w tabeli 3.1, przez:

1. Nazwę – nazwa grupy użytkowników.
2. Opis – krótki opis grupy użytkowników.
3. Profil grupy – cechy charakteryzujące grupę użytkowników.
4. Nadgrupę – bezpośredni przodek grupy użytkowników.
5. Podgrupy – lista grup potomnych.
6. Przypadki użycia – lista przypadków użycia, w których uczestniczy opisywana grupa użytkowników.
7. Uprawnienia – opis uprawnień grupy użytkowników do obiektów przetwarzanych przez aplikację.

Dodatkowo specyfikacja może zostać rozszerzona m.in. o rysunek przedstawiający hierarchię grup użytkowników(Żyła i Kęsik, 2010).

Tabela 3.1. Przykładowa specyfikacja grupy użytkowników

Nazwa	Pracownik
Opis	Osoba pracująca w sklepie
Profil grupy	Nazwa, adres e-mail, hasło, (...)
Nadgrupa	Zarejestrowany użytkownik

Podgrupy	Dyrektor ds. marketingu, Dyrektor ds. administracyjnych, (...)
Przypadki użycia	Dodanie promocji, Usunięcie promocji, Wysłanie newslettera, Zalogowanie
Uprawnienia	Odczyt/zapis: promocja, produkt Odczyt: użytkownik

Źródło: opracowanie własne

SPECYFIKACJA PRZYPADKÓW UŻYCIA

Specyfikacja przypadków użycia zawiera informacje o najważniejszych przypadkach użycia obrazujących sposoby interakcji użytkowników z aplikacją. Jest złożona z diagramu przypadków użycia (wykonany w notacji UML) oraz scenariuszy (niewielkie modyfikacje względem UML). Każdyscenariusz jest opisywany,w sposób analogiczny do przedstawionego w tabeli 3.2, przez:

1. Nazwę – nazwa przypadku użycia.
2. Opis – krótki opis realizowanej funkcjonalności.
3. Warunki początkowe – wymagania do spełnienia przed realizacją przypadku użycia.
4. Warunki końcowe – efekt wykonania przypadku użycia.
5. Przebieg – uporządkowany ciąg czynności skutkujący pomyślną realizacją przypadku użycia.

Dodatkowo do specyfikacji można dołączyć diagramy aktywności (dla bardziej złożonych przypadków użycia) oraz wykonać bardziej szczegółowe scenariusze (dołączając np. sytuacje wyjątkowe i przebiegi alternatywne)(Żyła i Kęsik, 2010).

Tabela 3.2. Przykładowa specyfikacja scenariusza

Nazwa	Logowanie użytkownika należącego do wielu grup
Opis	Opisuje sposób logowania przez użytkownika pełniącego więcej niż jedną rolę w aplikacji

Warunki początkowe	Użytkownik należy do wielu grup i zdefiniowano odpowiednie widoki witryny dla tych grup
Warunki końcowe	Użytkownik zostaje zalogowany do aplikacji i uzyskuje dostęp do widoku witryny jednej z jego grup
Przebieg	1. Wyświetlenie formatki logowania 2. Wprowadzenie i potwierdzenie danych logowania 3. Jeśli podano poprawne dane, uwierzytelnienie użytkownika i wyświetlenie listy grup, do których należy 4. Wybranie grupy przez użytkownika 5. Wyświetlenie widoku witryny dla wybranej grupy

Źródło: opracowaniewłasne

SPECYFIKACJA SŁOWNIKA OBIEKTÓW

Specyfikacja słownika obiektów zawiera informacje o najważniejszych obiektach w dziedzinie tworzonej aplikacji. Każdy obiekt jest opisywany,w sposób analogiczny do przedstawionego w tabeli 3.3, przez:

1. Nazwę – nazwa obiektu.
2. Synonimy – określenia równoznaczne nazwie obiektu.
3. Opis – krótki opis roli obiektu.
4. Przykładowe instancje – przykładowe wystąpienia obiektu.
5. Właściwości – lista istotnych atrybutów obiektu.
6. Komponenty – lista istotnych obiektów wchodzących w skład opisywanego obiektu.
7. Relacje – lista istotnych powiązań z innymi obiektami.
8. Generalizację – koncept generalizujący wystąpienia obiektu.
9. Specjalizację – koncept specjalizujący wystąpienia obiektu.

Dodatkowo do specyfikacji można dołączyć diagram klas w notacji UML(Żyła i Kęsik, 2010).

Tabela 3.3. Przykładowa specyfikacja obiektu ze słownika obiektów

Nazwa	Promocja
Synonimy	Brak
Opis	Informacja o promocji na wybrany produkt
Przykładowe instancje	**Super promocja na telewizor 32''** *Od: 10.08.2011* *Do: 12.08.2011* *Tylko w lubelskich sklepach* Wybierz dowolny telewizor 32'' a dostaniesz niebieską pigułkę gratis.
Właściwości	• nagłówek – tytuł promocji • od – początek promocji • do – koniec promocji • miejsce – miejsce obowiązywania promocji • tekst – treść promocji
Komponenty	Brak
Relacje *PromocjaDoProdukt*	Opcjonalnie łączy produkt z promocją
Generalizacja	Brak
Specjalizacja	Jednodniowe promocje

Źródło: opracowanie własne

SPECYFIKACJAWIDOKÓWWITRYNY

Specyfikacja widoków witryny zawiera informacje o podziale aplikacji na części (obszary, strony) oraz realizowanej przez nie funkcjonalności. Każdy widok witryny jest opisywany,w sposób analogiczny do przedstawionego w tabeli 3.4, przez:

1. Nazwę – nazwa widoku witryny.

2. Opis – krótki opis funkcjonalności i zawartości widoku witryny.
3. Grupy użytkowników – grupy użytkowników, które mają dostęp do widoku witryny.
4. Przypadki użycia – przypadki użycia realizowane przez widok witryny.
5. Mapę widoku witryny – zwięzła charakterystyka obszarów składających się na widok witryny; obszary są opisywane przez:
   a. nazwę – nazwa obszaru,
   b. opis – krótki opis funkcjonalności i zawartości obszaru,
   c. obiekty – lista obiektów ze słownika obiektów obecnych w ramach obszaru,
   d. priorytet – umowne określenie istotności obszaru, wpływające na kolejność projektowania i implementacji.

Do opisu obszarów widoku witryny można dodać informację o nazwach przypadków użycia, zgodnie z którymi realizowana jest opisywana funkcjonalność(Żyła i Kęsik, 2010).

Tabela 3.4. Przykładowa specyfikacja widoku witryny

Nazwa	Zarządzanie promocjami		
Opis	Zawiera strony służące do edytowania, dodawania i usuwania promocji oraz rozsyłania informacji o ni ch do zapisanych odbiorców		
Grupy użytkowników	Pracownik, (...)		
Przypadki użycia	Dodanie promocji, Usunięcie promocji, Zmodyfikowanie promocji, Zalogowanie, Wysłanie newslettera		
Mapa widoku witryny			
Nazwa obszaru	**Opis obszaru**	**Obiekty**	**Priorytet**
Zarządzanie promocjami	Strona domyślna obszaru o nazwie „Promocje” przedstawia listę promocji z opcjami wyświetlenia szczegółów oraz usunięcia i dodania promocji. Wybranie pierwszej opcji przenosi na stronę o nazwie „Szczegóły promocji”. Wybranie	Promocja, newsletter	Wysoki

	drugiej opcji powoduje usunięcie promocji oraz przeładowanie bieżącej strony. Wybranie trzeciej opcji przenosi na stronę o nazwie „Nowa promocja". Strona obszaru o nazwie „Szczegóły promocji" wyświetla szczegóły promocji oraz umożliwia jej modyfikację. Zapisanie modyfikacji powoduje rozesłanie newslettera. (...)		

Źródło: opracowanie własne

3.3 MODEL DANYCH

Procestworzenia modelu danych ma na celu uzyskanie przejrzystej struktury obiektów aplikacji wymagających utrwalenia oraz zależności pomiędzy nimi. Wejściem procesu są produkty specyfikacji wymagań, a wyjściem model warstwy danych aplikacji, znajdujący odwzorowanie w rzeczywistych źródłach danych. Model danych bazuje na diagramie związków encji, inaczej ERD (*ang. EntityRelationship Diagram*), przy czym(Ceri i inni, 2003):

- operuje na pojęciach (relacje, role relacji, atrybuty, encje),
- dopuszcza relacje wiele-do-wielu,
- dopuszcza tworzenie hierarchii encji,
- dopuszcza wartości pochodne (encje, relacje, atrybuty),
- używa typów danych niezależnych od bazy danych,
- może być normalizowany.

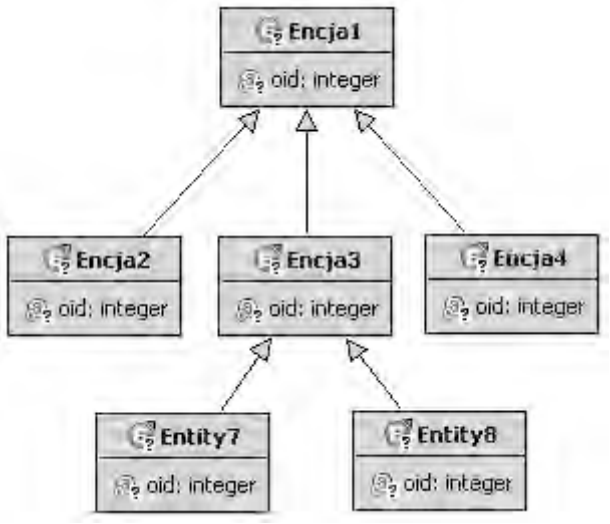
ENCJE

Encja jest opisem wspólnych cech grupy obiektów występujących w świecie rzeczywistym. Może być rozpatrywana jako odpowiednik klasy w obiektowych językach programowania – podobnie, jak klasa, posiada atrybuty o różnych typach danych. Zbiór instancji encji, czyli zbiór obiektów opisywanych przez encję jest

nazywany populacją encji. Populacja encji może być utożsamiana ze zbiorem rekordów w tabeli.

Każda instancja encji jest różna od pozostałych, więc projektant modelu musi zadbać o odnalezienie jednej lub więcej cech encji, które pozwolą na jednoznaczną identyfikację każdej z jej instancji. W tym kontekście pojawia się pojęcie klucza kandydującego i klucza głównego (podstawowego). Kluczem kandydującym jest każdy zestaw cech encji, który pozwala na j ednoznaczną identyfikację każdej z instancji encji. Kluczem głównym jest jeden wybrany klucz kandydujący. Niekiedy, dla wygody lub z braku naturalnych kluczy kandydujących, wprowadza się klucz główny będący np. kolejnymi liczbami całkowitymi numerującymi poszczególne instancje.

WebMLumożliwia organizowanie encji w hierarchię IS-A (rys. 3.2). Hierarchia encji może składać się z wielu poziomów, ale nie dopuszcza dziedziczenia wielokrotnego. Bazując na rysunku 3.2 można stwierdzić, że *Encja1* generalizuje, czyli jest koncepcją bardziej ogólną, encje o nazwach *Encja2*, *Encja3* i *Encja4*. Prawdziwe jest również stwierdzenie, że *Encja2*, podobnie jak *Encja3* oraz *Encja4*, specjalizuje, czyli jest koncepcją bardziej szczegółową, encję o nazwie *Encja1*. Analogicznie można rozpatrywać kolejne poziomy hierarchii.



Rys. 3.2. Przykładowa hierarchia encji w notacji WebML
Źródło: opracowanie własne

Mapowanie hierarchii na relacyjny model danych może odbywać się wertykalnie lub horyzontalnie. Mapowanie wertykalne polega na tym, że encja potomna oprócz swoich atrybutów otrzymuje atrybuty wszystkich swoich przodków. Mapowanie horyzontalne polega na tym, że encja będąca przodkiem oprócz swoich atrybutów otrzymuje atrybuty encji potomnych (Ceri i inni, 2003).

ATRYBUTY

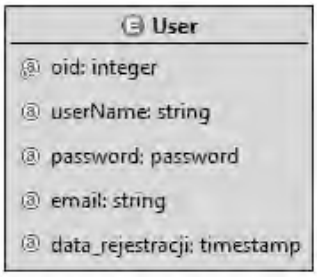
Atrybut encji, to cecha, właściwość grupy obiektów występujących w świecie rzeczywistym opisywanych przez encję. Wszystkie wartości przyjmowane przez atrybut mają wspólną dziedzinę (są tego samego typu) oraz są niepodzielne tj. atomowe. Atrybuty encji mogą być podstawą do tworzenia związków, czyli relacji.

WebML dopuszcza użycie następujących, powszechnie stosowanych w językach programowania i części baz danych, typów danych(Ceri i inni, 2003):

- BLOB – służy do przechowywania dużych obiektów binarnych,
- Boolean – służy do przechowywania reprezentacji wartości logicznej „True” lub „False”,
- Date – służy do przechowywania daty (dzień, miesiąc, rok),
- Decimal – służy do przechowywania liczb posiadających precyzję i skalę,
- Float – służy do przechowywania liczb rzeczywistych,
- Integer – służy do przechowywania liczb całkowitych,
- Password – służy do przechowywania haseł,
- String – służy do przechowywania łańcuchów znaków,
- Text – służy do przechowywania dużych ilości tekstu (odpowiednik CLOB),
- Time – służy do przechowywania godziny (godzina, minuta, sekunda),
- Timestamp – służy do przechowywania „odcisku czasu” (rok, miesiąc, dzień, godzina, minuta, sekunda),
- URL – służy do przechowywania adresów URL (uniwersalnych lokalizatorów zasobów).

Na rysunku 3.3 przedstawiono encję o nazwie *User* przechowującą dane użytkowników aplikacji. Kluczem głównym encji jest atrybut o nazwie *oid*, przyjmujący

wartości całkowite. Pozostałe atrybuty przechowują kolejno: nazwę, hasło, adres e-mail oraz dokładną datę i czas rejestracji użytkownika aplikacji.



Rys. 3.3. Przykład encji posiadającej atrybuty
Źródło: opracowanie własne

RELACJE

Relacje (związki), topołączenia pomiędzy encjami, które mogą być widziane z poziomu bazy danych jako klucze obce. Wyróżnia się relacje:

- binarne – łączą dwie encje,
- N–arne – angażują więcej niż dwie encje.

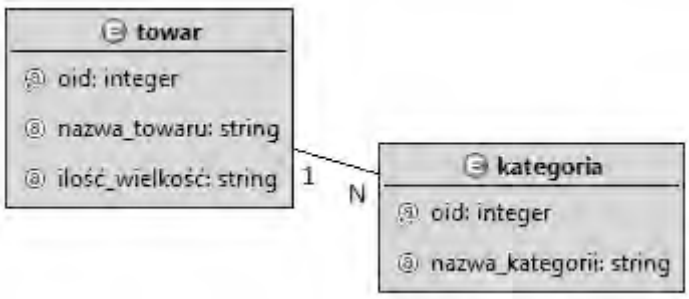
Relacje N–arne są dozwolone tylko na poziomie koncepcyjnym i odwzorowywane w źródłach danych przy użyciu encji i relacji binarnych.

Każda relacja binarna jest charakteryzowana przez dwie role relacji, które funkcjonują jako kierunki relacji oraz określają zależność między łączonymi encjami. Role posiadają atrybut liczności, czyli minimalną i maksymalną liczbę obiektów encji docelowej, z którymi może być powiązany obiekt encji źródłowej. Minimalna liczność roli, to wartość 1 lub 0, maksymalna to N lub 1. W zależności od liczności maksymalnej wyróżnia się następujące typy relacji:

- jeden–do–jednego (często oznaczana jako 1:1) – obydwie role relacji mają liczność maksymalną równą 1,
- jeden–do–wielu (często oznaczana jako 1:N) – jedna z ról ma liczność maksymalną równą 1, a druga równą N,
- wiele–do–wielu (często oznaczana jako N:M) – obydwie role mają liczność maksymalną większą od 1.

Relacje wiele–do–wielu są dopuszczone tylko na poziomie koncepcyjnym i odwzorowywane w relacyjnych źródłach danych przy pomocy tabeli pośredniczącej oraz relacji jeden–do–wielu(Ceri i inni, 2003).

Na rysunku 3.4 przedstawiono encje *towar* i *kategoria* połączone ze sobą relacją jeden-do-wielu. Oznacza ona, że pewien towar może należeć tylko do jednej kategorii (jednej instancji encji *towar* odpowiada jedna instancja encji *kategoria*), ale jednocześnie do jednej kategorii może należeć wiele towarów (jednej instancji encji *kategoria* odpowiada wiele instancji encji *towar*). Na modelu danych zaproponowanym przez WebML nie oznacza się kluczy obcych, są one tworzone automatycznie podczas mapowania modelu do rzeczywistego źródła danych.



Rys. 3.4. Przykład związku dwóch encji
Źródło: opracowanie własne

ELEMENTYPOCHODNE

Model danych można rozszerzyć o dodatkowe informacje powstałe wskutek przetworzenia danych już istniejących. Elementy pochodne (encje, atrybuty, relacje)mogą być definiowane przy pomocy dowolnego języka umożliwiającego przenoszalność relacji oraz budowę wyrażeń zawierających atrybuty encji, co udostępnia m.in. OCL (*ang. Object Constraint Language*; język zdefiniowany w ramach notacji UML) oraz OQL (*ang. Object Query Language*).

Encja może uzyskać status pochodnej, gdy specjalizuje inną encję, czyli reprezentuje podklasę jej instancji o specyficznych właściwościach. W związku z tym encję pochodną definiuje się jako perspektywę filtrującą rekordy nadencji, zawierającą

wszystkie jej atrybuty oraz współdzielącą wychodzące z niej relacje.

WebMLwyróżnia następujące typy relacji pochodnych:

1. Relacja konkatenowana (*ang. ImportedRelationship*) – relacja pomiędzy encjami zbudowana z istniejących relacji.
2. Relacja specjalizująca (*ang. New Relationship*) – relacja powstała poprzez uszczegółowienie relacji istniejącej.

Relację specjalizującą uzyskuje się dodając jeden lub więcej warunków logicznych identyfikujących instancje encji, których dotyczy relacja pochodna. Objęte nią rekordy stanowią podzbiór rekordów objętych relacją bazową(Ceri i inni, 2003).

WebML wyróżnia następujące typy atrybutów pochodnych:

1. Atrybut stały (*ang. ConstantAttribute*) – wartość atrybutu jest stałą.
2. Atrybut o prostym imporcie (*ang. Simple ImportedAttribute*) – wartość atrybutu jest kopią wartości innego atrybutu innej encji. Dotyczy relacji jeden-do-wielu i jeden-do-jednego.
3. Atrybut o złożonym imporcie (*ang. ComplexImportedAttribute*) –wartość atrybutu jest kopią wartości innego atrybutu osiągalnego z bieżącej encji relacją o dowolnej liczności i opcjonalnym warunku.
4. Atrybut obliczany (*ang. CalculatedAttribute*) – wartość atrybutu jest wynikiem formuły operującej na innych atrybutach oraz opcjonalnie zawierającej funkcje agregujące.

Na rysunku 3.5 przedstawiono encję o nazwie *promocja*zawierającą dwa atrybuty pochodne:

1. *średnia_ocena* – atrybut obliczany, który przyjmuje wartość będącą średnią ocen przyznanych każdej z promocji przez użytkowników aplikacji.
2. *ilość_ocen* – atrybut pochodny, który przyjmuje wartość będącą ilością ocen przyznanych każdej z promocji przez użytkowników aplikacji.

Wartość średnia jest wyznaczana poprzez wywołanie funkcji wyznaczającej wartość średnią (*avg*) wartości atrybutu *wartość* encji *ocena_promocji*. Odpowiedni atrybut został wskazany przez rolę relacji pomiędzy encją *promocja* a encją *ocena_promocji*. Wyrażenie wchodzące w skład atrybutu pochodnego jest następujące:

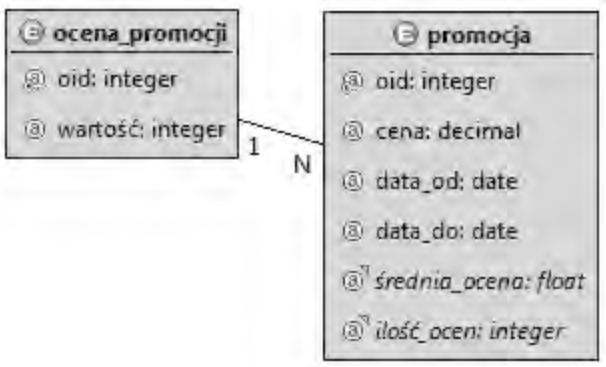
avg(Self.promocjaToocena_promocji(asVar1).wartość)

Ilość ocen jest wyznaczanapoprzez wywołanie funkcji zliczającej (*count*) liczbę

instancji encji *ocena_promocji*. Odpowiednia encja została wskazana przez rolę relacji pomiędzy encją *promocja* a encją *ocena_promocji*. Wyrażenie wchodzące w skład atrybutu pochodnego jest następujące:

count(Self.promocjaToocena_promocji(asVar1))

Treść przytoczonych wyrażeń może ulec zmianie w zależności od użytego języka.



Rys. 3.5. Przykład encji zawierającej atrybuty pochodne
Źródło: opracowanie własne

3.4 MODEL HIPERTEKSTU

Procestworzenia modelu hipertekstu polega na podziale aplikacji internetowej na bloki (widoki witryny, obszary i strony), rozmieszczeniu treści i funkcjonalności w ich ramach oraz ustaleniu przepływu sterowania pomiędzy elementami aplikacji. Wejściami są produkty specyfikacji wymagań oraz model danych, natomiast wyjściem jest model, który może być podstawą do wygenerowania kodu aplikacji.

Model hipertekstu może być postrzegany jako naturalne rozszerzenie modelu danych. Stanowi warstwę pośrednią pomiędzy warstwą danych a warstwą prezentacji. Pomimo niezależności od modelu danych wykorzystuje nazwy jego elementów. Na całość modelu hipertekstu składa się model kompozycji (określa rozmieszczenie treści i funkcjonalności) i model nawigacji (określa przepływ sterowania).

Model kompozycji bazuje na pojęciu komponentów – samodzielnych elementów

realizujących pewne zadanie. Komponent w WebML, oprócz określenia wykonywanego zadania, może składać się z:

- parametrów wymaganych na wejściu komponentu,
- parametrów dostarczanych na wyjście komponentu,
- nazwy encji, z którą jest połączony komponent,
- warunków ograniczających zbiór instancji encji, na których operuje komponent.

W ramach modelu kompozycji wyróżnia się następujące główne grupy komponentów (elementów notacji):

1. Komponenty prezentacji danych– odpowiadają za prezentację treści, rozmieszczane w ramach stron aplikacji.
2. Komponenty operacji – reprezentują czynności wykonywane przez aplikację w celu realizacji pewnych zadań, rozmieszczane poza stronami aplikacji.
3. Komponenty hybrydowe – w zależności od rozmieszczenia (w ramach strony lub poza stroną aplikacji) zachowują się jak komponenty danych lub komponenty operacji.
4. Komponenty serwisów – komponenty wykorzystywane wyłącznie w widoku serwisów.

W związku z dynamicznym rozwojem notacji, w celu dostosowywania do potrzeb nowoczesnych aplikacji internetowych, przedstawiony podział komponentów może ulec zmianie, podobnie jak zawartość ich palety(Ceri i inni, 2003).

3.4.1 Elementy modelu nawigacji

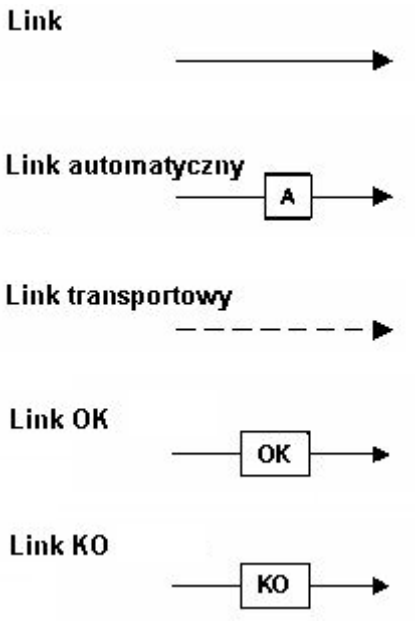
Model nawigacji definiuje połączenia pomiędzy powiązanymi ze sobą komponentami i stronami, umożliwiając przepływ sterowania oraz określając dostępne użytkownikom sposoby przemieszczania się pomiędzy stronami aplikacji. Centralnym pojęciem notacji jest link, który definiuje się jako wektor łączący pary komponentów lub stron.

WebML, ze względu na pełnione funkcje, wyróżnia następujące typy linków (Ceri i inni, 2003):

- link (*ang. Normal Link*) – widoczny dla użytkownika aplikacji, służy do nawigacji i aktywacji komponentu, może przenosić wartości,

- link automatyczny (*ang. Automatic Link*) – służy do automatycznego zainicjowania pewnej domyślnej akcji, kiedy z jakiegoś powodu brak akcji ze strony użytkownika,
- link transportowy (*ang. Transport Link*) – służy wyłącznie do przekazywania wartości pomiędzy komponentami, niewidoczny dla użytkownika aplikacji,
- link OK (*ang. OK Link*) – służy do inicjowania pewnej akcji, gdy operacja zakończy się powodzeniem,
- link KO (*ang. KO Link*) – służy do inicjowania pewnej akcji, gdy operacja zakończy się niepowodzeniem.

Na rysunku 3.6 przedstawiono jedną z graficznych reprezentacji poszczególnych typów linków. W zależności od implementacji notacji WebML ich wygląd może ulec zmianie. Bywa, że link OK jest zieloną strzałką pozbawioną etykiety „OK", a link KO jest czerwoną strzałką pozbawioną etykiety „KO". Ponadto link automatyczny jest niekiedy nie do odróżnienia od linku.



Rys. 3.6. Jedna z graficznych reprezentacji linków
Źródło: opracowanie własne

3.4.2 ELEMENTY MODELU KOMPOZYCJI

KOMPONENTDANYCH

Komponent Danych (*ang. Data Unit*) zalicza się do komponentów prezentacji danych, służy do publikacji na stronie dokładnie jednej instancji wybranej encji (rys. 3.7).Jest charakteryzowany przez następujące, istotne z punktu widzenia modelu, właściwości(WebRatio, 2009):

1. Nazwa – nazwa nadana przez projektanta.
2. Encja źródłowa – encja, której instancja jest prezentowana przez komponent.
3. Warunek wyboru – warunek wybierający dokładnie jedną instancję encji źródłowej.
4. Wyświetlane atrybuty – zbiór atrybutów encji, które zostaną wyświetlone na stronie w miejscu położenia komponentu.

Rys. 3.7. Reprezentacja graficzna Komponentu Danych
Źródło: opracowanie własne

KOMPONENTINDEKSU

Komponent Indeksu (*ang. Index Unit*) zalicza się do komponentów prezentacji danych, służy do publikacji wielu instancji wybranej encji w postaci listy (rys. 3.8).Jest charakteryzowany przez następujące, istotne z punktu widzenia modelu, właściwości(WebRatio, 2009):

1. Nazwa – nazwa nadana przez projektanta.
2. Encja źródłowa – encja, której instancje są prezentowane przez komponent.
3. Warunek wyboru – warunek wybierający zbiór instancji encji źródłowej.
4. Wyświetlane atrybuty – zbiór atrybutów encji źródłowej, które zostaną wyświetlone

na stronie w miejscu położenia komponentu.

5. Klauzula sortująca – porządek sortowania wyświetlanych instancji encji źródłowej.

Rys. 3.8. Reprezentacja graficzna Komponentu Indeksu
Źródło: opracowanie własne

KOMPONENTINDEKSUHIERARCHICZNEGO

Komponent Indeksu Hierarchicznego (*ang. Hierarchical Index Unit*) zalicza się do komponentów prezentacji danych i jest odmianą Komponentu Indeksu (charakteryzują go analogiczne cechy). Służy do wyświetlania wielopoziomowych list instancji encji, przy czym poszczególne poziomy są ze sobą powiązane oraz posiadają własne encje źródłowe (rys. 3.9). Przykładem obrazującym tą koncepcję jest np. menu wyświetlające kategorie (pierwszy poziom listy) oraz produkty przypisane do każdej z nich (drugi poziom listy)(WebRatio, 2009).

Rys. 3.9. Reprezentacja graficzna Komponentu Indeksu Hierarchicznego
Źródło: opracowanie własne

KOMPONENTWPROWADZANIA

Komponent Wprowadzania (*ang. Entry Unit*)zalicza się do komponentów prezentacji danych (rys. 3.10). Symbolizuje klasyczny formularzdostarczający wartości dla komponentów operacji lub służący jako formatka wyszukiwania. Jest

charakteryzowany przez następujące, istotne z punktu widzenia modelu, właściwości(WebRatio, 2009):

1. Nazwa – nazwa nadana przez projektanta.
2. Lista pól – zestaw pól służących do wprowadzania wartości.

Każde pole Komponentu Wprowadzania jest charakteryzowane przez następujące, istotne z punktu widzenia modelu, właściwości:

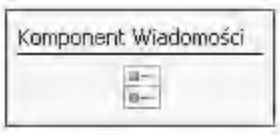
1. Nazwa – nazwa nadana przez projektanta.
2. Typ – typ wprowadzanej wartości.
3. Wartość początkowa – domyślna wartość pola proponowana użytkownikowi.
4. Podatność na modyfikacje – wartość logiczna określająca, czy wartość pola może być modyfikowana.
5. Reguły walidacyjne – zestaw wyrażeń sprawdzających poprawność wprowadzonej wartości pola.

Rys. 3.10. Reprezentacja graficzna Komponentu Wprowadzania
Źródło: opracowanie własne

KOMPONENTWIADOMOŚCI

Komponent Wiadomości (*ang. Multi Message Unit*) zalicza się do komponentów prezentacji danych, służy do publikacji na stronie określonej wiadomości lub komunikatu, np. komunikatu z błędem wykonania pewnej operacji (rys. 3.11).Jest charakteryzowany przez następujące, istotne z punktu widzenia modelu, właściwości(WebRatio, 2009):

1. Nazwa – nazwa nadana przez projektanta.
2. Wiadomość – tekst wiadomości do wyświetlenia.
3. Wiadomość domyślna – domyślna treść wiadomości, używana, gdy nie dostarczono treści w inny sposób.



Rys. 3.11. Reprezentacja graficzna Komponentu Wiadomości
Źródło: opracowanie własne

KOMPONENTSELEKCJI

Komponent Selekcji (*ang. Selector Unit*) zalicza się do komponentów hybrydowych, służy do wybierania instancji encji spełniających zadane warunki bez konieczności ich wyświetlenia na stronie (rys. 3.12).Jest charakteryzowany przez następujące, istotne z punktu widzenia modelu, właściwości(WebRatio, 2009):

1. Nazwa – nazwa nadana przez projektanta.
2. Encja źródłowa – encja, której instancje są wybierane przez komponent.
3. Warunek wyboru – warunek wybierający zbiór instancji encji źródłowej.
4. Klauzula sortująca – porządek sortowania wyświetlanych instancji encji źródłowej.

Rys. 3.12. Reprezentacja graficzna Komponentu Selekcji
Źródło: opracowanie własne

TIME UNIT – KOMPONENTCZASU

Komponent Czasu (*ang. Time Unit*) zalicza się do komponentów hybrydowych, służy do określania bieżącego czasu (rys. 3.13).Jest charakteryzowany przez następujące, istotne z punktu widzenia modelu, właściwości(WebRatio, 2009):

1. Nazwa – nazwa nadana przez projektanta.
2. Format czasu – wyrażenia opisujące sposób reprezentacji czasu.

Rys. 3.13. Reprezentacja graficzna Komponentu Czasu
Źródło: opracowanie własne

KOMPONENTTWORZENIAINSTANCJI

Komponent Tworzenia Instancji (*ang. Create Unit*) zalicza się do komponentów operacji, służy do tworzenia nowych instancji wskazanej encji (rys. 3.14).Jest charakteryzowany przez następujące, istotne z punktu widzenia modelu, właściwości(WebRatio, 2009):

1. Nazwa – nazwa nadana przez projektanta.
2. Encja – encja, której instancje są tworzone przez komponent.
3. Mapa przypisań – struktura powiązań atrybutów tworzonej instancji z dostarczonymi wartościami.

Rys. 3.14. Reprezentacja graficzna Komponentu Tworzenia Instancji
Źródło: opracowanie własne

KOMPONENTUSUWANIAINSTANCJI

Komponent Usuwania Instancji (*ang. Delete Unit*) zalicza się do komponentów operacji, służy do usuwania instancji encji spełniających zadane warunki (rys. 3.15).Jest charakteryzowany przez następujące, istotne z punktu widzenia modelu, właściwości(WebRatio, 2009):

1. Nazwa – nazwa nadana przez projektanta.
2. Encja – encja, której instancje są usuwane przez komponent.

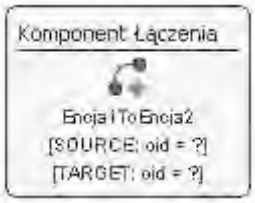
3. Warunek wyboru – warunek wybierający zbiór instancji encji do usunięcia.

Rys. 3.15. Reprezentacja graficzna Komponentu Usuwania Instancji
Źródło: opracowanie własne

KOMPONENTŁĄCZENIA

Komponent Łączenia (*ang. Connect Unit*)zalicza się do komponentów operacji, służy do tworzenia nowych instancji relacji, tzn. łączenia określonych instancji encji źródłowej z określonymi instancjami encji docelowej (rys. 3.16).Jest charakteryzowany przez następujące, istotne z punktu widzenia modelu, właściwości(WebRatio, 2009):

1. Nazwa – nazwa nadana przez projektanta.
2. Rola relacji źródłowej – rola, której dotyczy operacja.
3. Warunek wyboru instancji encji źródłowej – warunek wybierający zbiór instancji encji źródłowej do połączenia.
4. Warunek wyboru instancji encji docelowej – warunek wybierający zbiór instancji encji docelowej do połączenia.



Rys. 3.16. Reprezentacja graficzna Komponentu Łączenia
Źródło: opracowanie własne

KOMPONENTROZŁĄCZANIA

Komponent Rozłączania (*ang. Disconnect Unit*)zalicza się do komponentów operacji, służy do usuwania instancji relacji, tzn. usuwania połączeń pomiędzy instancjami encji źródłowej i docelowej (rys. 3.17).Jest charakteryzowany przez następujące, istotne z punktu widzenia modelu, właściwości(WebRatio, 2009):

1. Nazwa – nazwa nadana przez projektanta.
2. Rola relacji źródłowej – rola, której dotyczy operacja.
3. Warunek wyboru instancji encji źródłowej – warunek wybierający zbiór instancji encji źródłowej do rozłączenia.
4. Warunek wyboru instancji encji docelowej – warunek wybierający zbiór instancji encji docelowej do rozłączenia.

Rys. 3.17. Reprezentacja graficzna Komponentu Rozłączania
Źródło: opracowanie własne

KOMPONENTDECYZYJNY

Komponent Decyzyjny (*ang. Is Not Null Unit*)zalicza się do komponentów operacji,służy do sprawdzania, czy parametr wejściowy ma wartość *null* i w jego efekcie wskazywania kierunku przetwarzania (rys. 3.18).Jest charakteryzowany przez następujące, istotne z punktu widzenia modelu, właściwości(WebRatio, 2009):

1. Nazwa – nazwa nadana przez projektanta.
2. Akcja, gdy *not null* – akcja do podjęcia, gdy parametr wejściowy nie ma wartości *null*.
3. Akcja, gdy *null* – akcja do podjęcia, gdy parametr wejściowy ma wartość *null*.

Rys. 3.18. Reprezentacja graficzna Komponentu Decyzyjnego
Źródło: opracowanie własne

KOMPONENTPOCZTY

Komponent Poczty (*ang. Sendmail Unit*)zalicza się do komponentów operacji,służy dowysyłania wiadomości e-mail pod wskazane adresy (rys. 3.19).Jest charakteryzowany przez następujące, istotne z punktu widzenia modelu, właściwości(WebRatio, 2009):

1. Nazwa – nazwa nadana przez projektanta.
2. Treść – treść wiadomości e-mail.
3. Odbiorcy – lista adresów odbiorców wiadomości.
4. Nadawca – adres nadawcy wiadomości.
5. Temat – temat wiadomości.
6. Załączniki – załączniki dołączone do treści wiadomości.

Rys. 3.19. Reprezentacja graficzna Komponentu Poczty
Źródło: opracowanie własne

KOMPONENTMATEMATYCZNY

Komponent Matematyczny (*ang. Math Unit*)zalicza się do komponentów hybrydowych,służy dowyznaczania wartości wyrażenia matematycznego na podstawie dostarczonych wartości (rys. 3.20).Jest charakteryzowany przez następujące, istotne z punktu widzenia modelu, właściwości(WebRatio, 2009):

1. Nazwa – nazwa nadana przez projektanta.
2. Wyrażenie – wyrażenie matematyczne do obliczenia.
3. Domyślne wyrażenie – domyślne wyrażenie matematyczne obliczane, jeśli nie dostarczono wyrażenia.
4. Typ wyniku – typ wartości wynikowej.

Rys. 3.20. Reprezentacja graficzna Komponentu Matematycznego
Źródło: opracowanie własne

KOMPONENTLOGOWANIA

Komponent Logowania (*ang. Login Unit*)zalicza się do komponentów operacji, służy do uwierzytelnienia i autoryzacji użytkowników aplikacji (rys. 3.21).Jest charakteryzowany przez następujące, istotne z punktu widzenia modelu, właściwości(WebRatio, 2009):

1. Nazwa – nazwa nadana przez projektanta.
2. Nazwa użytkownika – nazwa konta użytkownika aplikacji.
3. Hasło – hasło do konta użytkownika aplikacji.

Rys. 3.21. Reprezentacja graficzna Komponentu Logowania
Źródło: opracowanie własne

KOMPONENTWYLOGOWANIA

Komponent Wylogowania (*ang. LogoutUnit*)zalicza się do komponentów operacji,

służy do wylogowania użytkownika aplikacji, co polega na usunięciu sesji zalogowanego użytkownika i przeniesieniu go do obszaru aplikacji przeznaczonego dla użytkowników niezalogowanych (rys. 3.22).Jest charakteryzowany przez następujące, istotne z punktu widzenia modelu, właściwości(WebRatio, 2009):

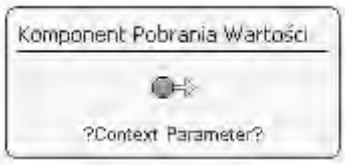
1. Nazwa – nazwa nadana przez projektanta.
2. Docelowy obszar – miejsce, do którego zostanie przeniesiony użytkownik aplikacji po wylogowaniu.

Rys. 3.22. Reprezentacja graficzna Komponentu Wylogowania
Źródło: opracowanie własne

KOMPONENTPOBRANIAWARTOŚCI

Komponent Pobrania Wartości (*ang. GetUnit*)zalicza się do komponentów operacji, służy do pobierania wartości parametrów globalnych lub lokalnych np. parametrów kontekstowych takich, jak identyfikator użytkownika lub identyfikator grupy użytkowników (rys. 3.23).Jest charakteryzowany przez następujące, istotne z punktu widzenia modelu, właściwości(WebRatio, 2009):

1. Nazwa – nazwa nadana przez projektanta.
2. Typ – typ zwracanego parametru.
3. Wartość – wartość zwracanego parametru.



Rys. 3.23. Reprezentacja graficzna Komponentu Pobrania Wartości
Źródło: opracowanie własne

KOMPONENTZADANIA

Komponent Zadania (*ang. Schedule Job Unit*)zalicza się do komponentów operacji, służy do określania chwili w czasie, kiedy ma zostać uruchomione określone zadanie zdefiniowane wcześniej w widoku serwisów (rys. 3.24).Jest charakteryzowany przez następujące, istotne z punktu widzenia modelu, właściwości(WebRatio, 2009):

1. Nazwa – nazwa nadana przez projektanta.
2. Zadanie – nazwa zadania uruchamianego przez komponent.
3. Harmonogram – czas, kiedy zadanie ma zostać uruchomione.

Rys. 3.24. Reprezentacja graficzna Komponentu Zadania
Źródło: opracowanie własne

KOMPONENTINICJACJIZADANIA

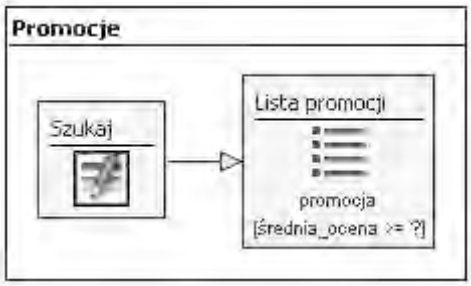
Komponent Inicjacji Zadania (*ang. Init JobUnit*)zalicza się do komponentów serwisów, służy jako punkt startowy zadania, jest początkiem łańcucha realizowanych przez zadanie operacji, przekazuje dane wejściowe do łańcucha operacji (rys. 3.25).Jest charakteryzowany przez następujące, istotne z punktu widzenia modelu, właściwości(WebRatio, 2009):

1. Nazwa – nazwa nadana przez projektanta.
2. Dane wejściowe – zbiór wartości do przekazania do inicjowanego łańcucha operacji.

Rys. 3.25. Reprezentacja graficzna Komponentu Inicjacji Zadania
Źródło: opracowanie własne

3.4.3 PROSTE PRZYKŁADY WYKORZYSTANIA KOMPONENTÓW MODELU HIPERTEKSTU

Na rysunku 3.26 przedstawiono model hipertekstu strony służącej do wyszukiwania promocji o średniej ocen wyższej niż wartość wprowadzona przez użytkownika w formularzu. Wprowadzona wartość jest przekazywana linkiem o nazwie *Wyszukaj* do selektora komponentu *Lista promocji*, który wybiera instancje encji *promocja* spełniające zadany warunek atrybutu.

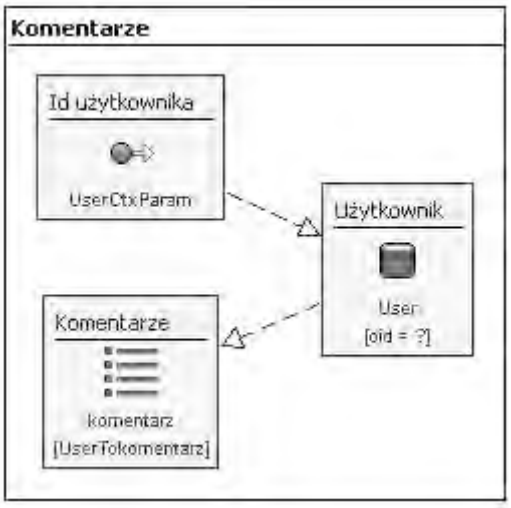


Rys. 3.26. Przykład wykorzystania komponentu wprowadzania do wyszukiwania
Źródło: opracowanie własne

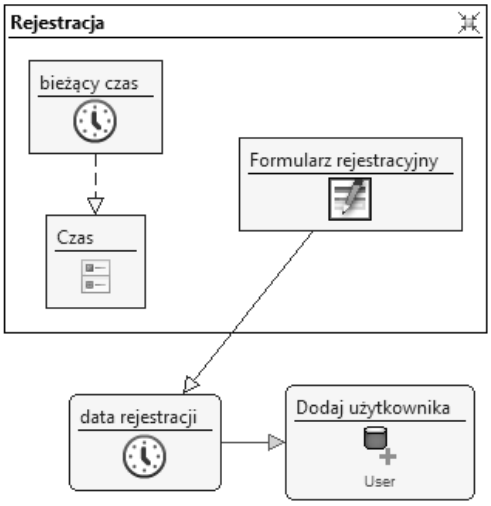
Na rysunku 3.27 przedstawiono model hipertekstu strony wyświetlającej informacje o zalogowanym użytkowniku oraz opublikowane przez niego komentarze do promocji. Komponent *Id użytkownika* przekazuje identyfikator zalogowanego użytkownika do selektora komponentu *Użytkownik*, w celu wyświetlenia danych użytkownika, np. imienia i nazwiska. Przetwarzanie strony kończy wyświetlenie wszystkich komentarzy do promocji, które były powiązane z zalogowanym użytkownikiem w sensie relacji *UserTokomentarz*.

Na rysunku 3.28 przedstawiono model hipertekstu strony umożliwiającej rejestrację nowego użytkownika. W ramach strony umieszczono formularz służący do wprowadzenia danych nowego użytkownika. Komponent *Dodaj użytkownika*nie jest aktywowany bezpośrednio po wysłaniu formularza, poprzedza go aktywacja Komponentu Czasu o na zwie *data rejestracji* (bieżący czas i data), generującego brakujący parametr czasu zapisu. Dopiero po dostarczeniu wszystkich wartości jest tworzona instancja encji *User*. Komponent Czasu umieszczono również wewnątrz

strony – efektem tego jest wyświetlenie w Komponencie Wiadomości czasu, kiedy otwarto stronę.



Rys. 3.27. Wyświetlanie informacji na podstawie parametru kontekstowego
Źródło: opracowanie własne



Rys. 3.28. Przykład wykorzystania komponentu hybrydowego
Źródło: opracowanie własne

3.5 MODEL PREZENTACJI

Elementy składowe WebML (komponenty, linki, parametry) są definiowane w modelu hipertekstu bez określania ich wyglądu czy ułożenia na stronie www. Układ elementów graficznych na stronie bywa bardzo różnorodny i jest kwestią określaną przez odpowiedni szablon wyglądu, definiowany przez grafików stron www.

Sposób i możliwości renderowania poszczególnych obiektów na stronie www podlegają ustawicznym zmianom, nie jest więc wskazane stałe przypisanie do komponentów WebML parametrów ich wyświetlania, właściwych tylko dla konkretnej technologii i układu struktury. Informacje te powinny być zawarte w osobnym modelu, definiującym sposób renderowania komponentów, właściwy dla wybranej technologii i szablonu układu. Z drugiej strony przydatne jest również uniezależnienie modelu prezentacji od modelu nadrzędnego tak, aby ewentualne zmiany w hipertekście nie wymagały korygowania przypisanego mu modelu prezentacji. Takie podejście owocuje również możliwością wykorzystania tego samego modelu prezentacji w wielu związanych ze sobą logicznie projektach (np. różne usługi oferowane przez konkretną firmę mogą posiadać ujednolicony szablon wyglądu).

WebML nie zakłada tworzenia konkretnego modelu (zdefiniowanego językiem DSL), mającego definiować prezentację aplikacji na poziomie konceptualnym. W zamian wykorzystuje standardowe podejścia, lepiej znane grafikom i twórcom interfejsów. Jako, że specyfikacja WebML może być reprezentowana za pomocą języka XML, model prezentacji jest uznawany za opis transformacji dokumentu, mapującej logiczny opis strony na stronę zapisaną w konkretnym języku budowy serwisów internetowych (np. JSP lub ASP.NET). W konsekwencji, prezentację definiuje się w WebML poprzez dołączanie arkuszy stylów XSL do widoków aplikacji, stron, komponentów i ich elementów składowych. Arkusz stylów XSL przyjmuje na wejściu specyfikacje WebML, zakodowane jako dokumenty XML zgodne z DTD (*ang. DocumentType Definition*) WebML. Na wyjściu zwraca szablony stron zawierające wymagane kody znaczników i zapytania pobierające dane. Implementacja WebML może zawierać kilka zdefiniowanych arkuszy stylów prezentacji wraz z komponentami po stronie serwera, obsługującymi zapytania pobierające dane, wymaganymi by wypełnić szablony stron wytworzone na podstawie arkuszy stylów XSL.

UKŁAD MODELU PREZENTACJI

Omawiany układ modelu prezentacji nie jest obligatoryjny. Jest to propozycja zoptymalizowana dla projektówWebML do budowy aplikacji operujących na znacznych ilościach informacji. Taki układ jest wykorzystywany w narzędziu WebRatio umożliwiającym budowę aplikacji internetowych wykorzystując WebML.

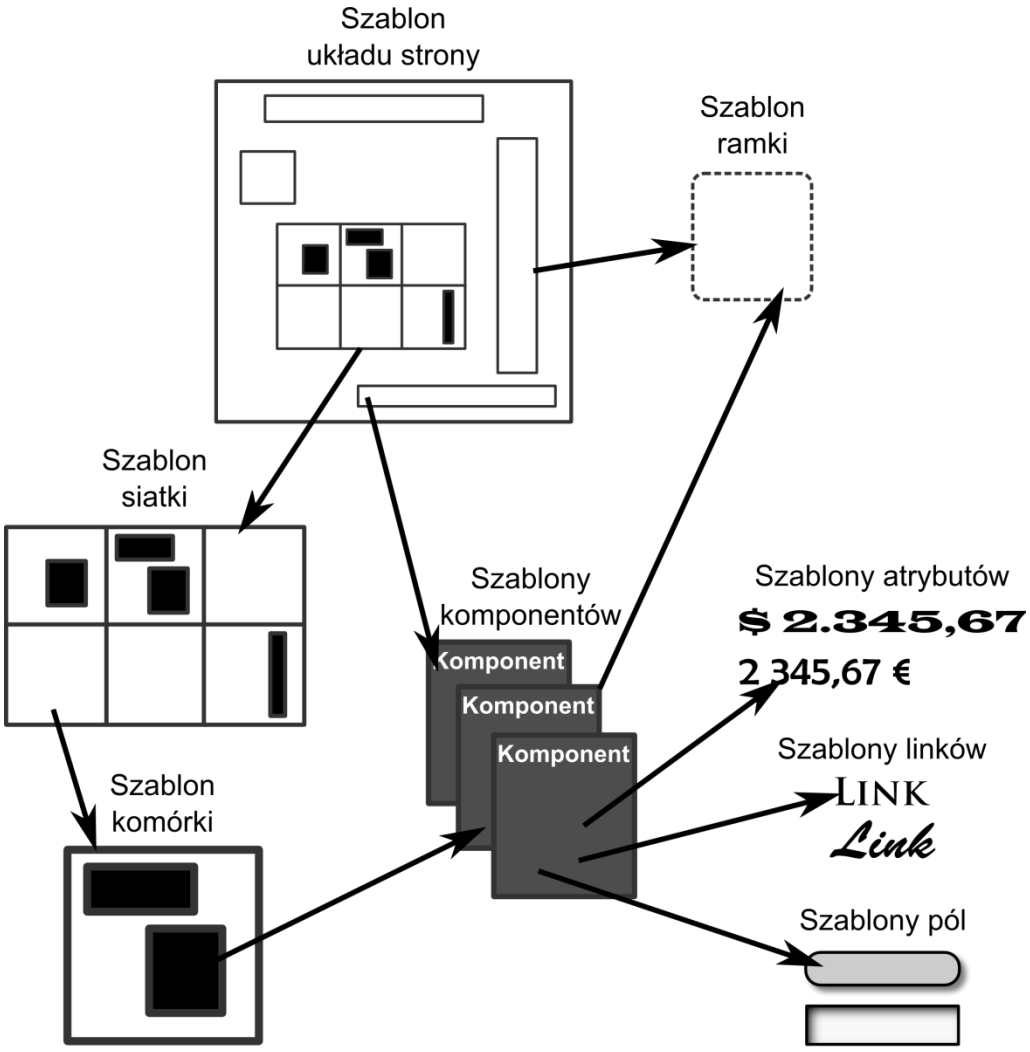
Układ ten zakłada istnienie szablonów stylu przypisywanych do całej aplikacji lub tylko do jej pewnego elementu (strony, wyskakującego okna, itp.). Szablon taki składa się z kilku rodzajów elementów połączonych w strukturę hierarchiczną.Struktura ta bazuje na definicji rodzajów elementów występujących na stronie:

- Siatka (*ang. grid*) – główny kontener organizujący komponenty prezentujące dane. Siatka grupuje komponenty w komórki w układzie tabelarycznym.
- Komórka (*ang. cell*) – pojedyncza komórka siatki. Zawiera dowolną ilość komponentów i/lub ich pojedynczych atrybutów. Czyli komórka może zawierać cały formularz rejestracyjny albo tylko pole wprowadzania nazwy użytkownika.
- Ramka (*ang. frame*) – obramowanie danego elementu. Odnosi się do dowolnego poziomu: siatki, komórki, komponentu, itp.
- Komponent (*ang. unit*) – pojedyncza instancja danego komponentu (np. Komponent Indeksu).
- Atrybut (*ang. attribute*) – jeden z atrybutów danego komponentu,np. kolumna *cena*wykazu produktów albo przycisk wysyłania w formularzu kontaktowym.
- Pole (*ang. field*) – pojedyncza instancja pola formularza generowanego przez Komponent Wprowadzania,np. pole do wpisywania adresu e-mail.
- Link – dowolny link wygenerowany przez jeden z komponentów.

Skutkiem tego podziału jest zdefiniowanie rodzajów szablonów, z jakich składa się model prezentacji (Rys. 3.29). Korzeniem tej struktury jest szablon układu strony (*ang. PageLayoutTemplate*). Pojedynczy model prezentacji może posiadać kilka różnych szablonów układu strony. Pozwala to na przypisywanie różnych układów różnym stronom modelu aplikacji.

Szablon układu strony jest makietą strony zapisaną w wybranym języku opisu stron (np. HTML). Jego struktura, oprócz elementów stałych, zawiera specjalne

znaczniki definiujące położenie i obszar stref (*ang. customlocation*), w których będą umieszczane elementy aplikacji.



Rys. 3.29. Hierarchiczna struktura modelu prezentacji

Źródło: opracowanie własne

Strefy (oprócz strefy *grid*) nie są przypisane do konkretnych komponentów. Przypisanie to odbywa się na etapie modelowania hipertekstu, gdzie udostępnione są nazwy poszczególnych stref wybranego szablonu. Strefa Siatki (*grid)* może być

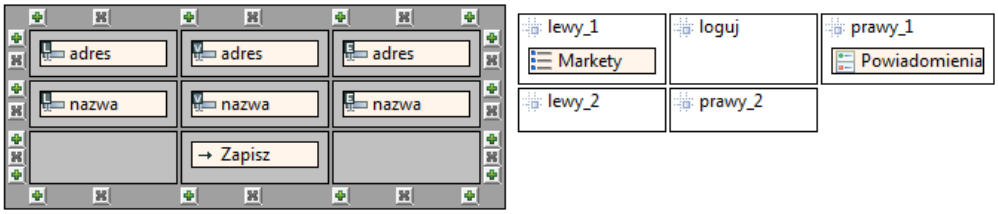
zdefiniowana tylko w jednym (najczęściej centralnym) miejscu strony i odnosi się do elementu *Siatka* modelu prezentacji. Listing 3.1 przedstawia przykładowy układ stref na stronie. Na górze strony znajduje się obszar przeznaczony na menu. Dalsza część jest podzielona na 3 kolumny – kolumny lewa i prawa zawierają po dwie strefy, podczas gdy kolumna środkowa zawiera strefę Siatki.

Listing 3.1. Szablon układu strony

```
<body>
<wr:PageFormfieldFocusScript="false">
<div id="loguj">
<wr:CustomLocation name="loguj"/>
</div>
<div id="menu">
<ul id="menu-gora-ul">
<wr:LandmarkMenu level="1">
<li><wr:Link/></li>
</wr:LandmarkMenu>
</ul>
</div>
<div id="lewy_panel">
      <div>
<wr:CustomLocation name="lewy_1"/>
      </div>
<div>
      <wr:CustomLocation name="lewy_2"/>
</div>
</div>
<div id="tresc">
<wr:Grid/>
</div>
<div id="prawy_panel">
<div>
<wr:CustomLocation name="prawy_1"/>
</div>
<div>
      <wr:CustomLocation name="prawy_2"/>
</div>
</div>
</wr:PageForm>
</body>
```

Rozkład komponentów między strefami jest definiowany na etapie modelowania

hipertekstu. Rys. 3.30 przedstawia przykład wykorzystania tych stref. Siatkazawiera atrybuty Komponentu Wprowadzania rozmieszczone w różnych komórkach. Jedna ze stref w kolumnie lewej zawiera Komponent Indeksu, a jedna ze stref kolumny prawej Komponent Wiadomości. Jak widać na rysunku, prezentowany rozkład nie uwzględnia graficznego rozmieszczenia stref na stronie – nie jest on istotny na tym etapie projektowania.



Rys. 3.30. Logiczny rozkład komponentów i atrybutów w szablonie
Źródło: opracowanie własne

Pozostałe szablony nie są bezpośrednio związane z konkretnym miejscem szablonu układu strony. Definiują one wyglądy pozostałych elementów, mogących wystąpić na stronie. Są to:

- Szablon układu siatki (*ang. GridLayoutTemplate*) – definiuje wygląd siatki elementów i ogólne parametry rozmieszczenia jej zawartości. Logiczny rozkład, między poszczególne komórki, komponentów umieszczonych w siatce jest definiowany na etapie modelowania hipertekstu.
- Szablon układu komórki (*ang. Cell LayoutTemplate*) – definiuje wygląd komórki siatki i ogólne parametry rozmieszczenia komponentów wewnątrz niej.
- Szablonukładuramki (*ang.FrameLayoutTemplate*) – definiuje wygląd obramowania dowolnego elementu składowego strony.
- Szablonukładukomponentu (*ang. Unit LayoutTemplate*) – definiuje wygląd danego typu komponentu,np. Komponent Kalendarza.
- Szablonukładuatrybutu (*ang.AttributeLayoutTemplate*) – definiuje wygląd atrybutu danego typu. Może być przypisany do konkretnego atrybutu prezentowanego przez dany komponent.

- Szablonukładupola (*ang.FieldLayoutTemplate*) - definiuje wygląd danego typupola w formularzu. Określa również układ ewentualnych elementów kontrolnych pola,np. komunikaty o błędach.
- Szablonukładulinku (*ang.LinkLayoutTemplate*) – definiuje wygląd linku danego typu. Może być przypisany np. do linków generowanych przez określony komponent.
- Szablonstronspecjalnych (*ang.SpecialPageTemplate*) – definiuje układ stron generowanych przez aplikację w szczególnych przypadkach,np. strona komunikatu o błędzie lub strona żądania zalogowania w przypadku wygaśnięcia sesji.

Każdy z tych szablonów składa się ze statycznych znaczników HTML/XHTML, określających wygląd danego elementu oraz części dynamicznej (będącej kodem w jednym z języków skryptowych, np. Groovy) realizującej wpasowanie danych dostarczanych przez ten element w szablon oraz pozwalającej na dobór i modyfikację wyglądu elementu w sposób przewidziany w szablonie. Innymi słowy, część dynamiczna realizuje rendering komponentu na kod znacznikowy strony, układając dane dostarczone przez komponent w sposób określony przez szablon. Sposób ułożenia danych może zależeć od parametrów zdefiniowanych w szablonie i ustawianych w trakcie modelowania aplikacji. Dzięki temu wzrasta uniwersalność danego szablonu.

Listing 3.2 przedstawia przykład wykorzystania parametru do zdefiniowania, czy Komponent Indeksu ma wyświetlać standardową informację w przypadku braku danych do zaprezentowania.

Listing 3.2. Przykład szablonu Komponentu Indeks

```
#?delimiters [%, %], [%=, %]
<wr:LayoutParameter label="Use Empty Unit Message" name="use-
empty-unit-message" type="boolean" default="false">
Określa czy komponent ma wyświetlać informacje o braku danych
- [Tak/Nie]
</wr:LayoutParameter>
<wr:LayoutParameter label="Empty Unit Message" name="empty-
unit-message" type="string" default="emptyUnitMessage">
Określa informację wyświetlaną w razie braku danych
</wr:LayoutParameter>
```

```
[%
import org.apache.commons.lang.StringUtils
import org.apache.commons.lang.math.NumberUtils
setHTMLOutput()
defunitId = unit["id"]
defuseEmptyUnitMessage = params["use-empty-unit-message"]
defemptyUnitMessage = params["empty-unit-message"]
def links = unit.selectNodes("layout:Link")
defunitLink = unit.selectSingleNode("layout:Link")
def link = unitLink?.valueOf("@link")
%]
[% if (useEmptyUnitMessage != "true") { %]
<c:if test="${not(empty <wr:UnitId/>)
and(<wr:UnitId/>.dataSizegt 0)}">
[% } else { %]
<c:choose>
<c:when test="${not(empty <wr:UnitId/>) and
(<wr:UnitId/>.dataSizegt 0)}">
[% } %]
[DEFINICJA SKRYPTU GENERUJĄCEGO POSZCZEGÓLNE WIERSZE INDEKSU]
[% if (useEmptyUnitMessage != "true") { %]
</c:if>
[% } else { %]
</c:when>
<c:otherwise>
<wr:Frame>
<div class="plain <wr:StyleClass/>">
<div class="plain IndexUnit">
<bean:message key="[%printJSPTagValue(emptyUnitMessage)%]"/>
</div>
</div>
</wr:Frame>
</c:otherwise>
</c:choose>
[% } %]
```

W szablonie tym wykorzystano zarówno język znacznikowy HTML, kod JET (*ang. Java Emitter Templates*) służący do generowania kodu JSP – platformy docelowej aplikacji, język Groovy obsługujący komunikację z komponentami oraz znaczniki specjalne definiujące parametry dostępne do modyfikacji z poziomu modelowania hipertekstu.

Parametry wykorzystywane w tym szablonie, to:

- *use-empty-unit-message* – parametr przyjmujący wartość true/false, odpowiadający za wybór, czy ma być prezentowany specjalny komunikat w razie braku danych do wyświetlenia (w przeciwnym wypadku szablon wygeneruje pusty obszar),
- *empty-unit-message*– parametr pozwalający na określenie treści komunikatu.

W zależności od parametru *use-empty-unit-message*jest wybierana konstrukcja *if* (wyświetlająca dane w pętli, jeżeli istnieją) albo konstrukcja *choose-when/otherwise* wyświetlająca albo dane w pętli, albo komunikat o ich braku. Należy tu zaznaczyć,że zarówno znaczniki specjalne, jak i kod Groovy oraz JET są przetwarzane tylko raz – w trakcie generowania aplikacji. Wynikowyfragment strony (w tym przypadku JSP) zawiera tylko kod własny oraz znaczniki HTML i skrypty Ajax interpretowane przez przeglądarkę.

Wynikowa strona zawierająca kod wykonywany po stronie serwera (server-side) jest więc generowana na podstawie zbioru komponentów (rozmieszczonych logicznie w modelu hipertekstu) wyrenderowanych przez hierarchicznie powiązane szablony. W przedstawionym na l istingu 3.2 przykładzie można zauważyć, że wynikowa struktura HTML bazuje na warstwach i szablonach stylów CSS. Odpowiednie klasy CSS są przypisywane do właściwych obszarów dokumentu wynikowego, pozwalając na dowolne dopasowanie jego wyglądu. Ważne jest, aby nazwy tych klas były wstawiane do szablonu w sposób ujednolicony, umożliwiający wykorzystanie danego arkusza stylów z dowolnym zestawem szablonów.

Konkretny model prezentacji może zawierać wiele grup szablonów, nazywanych szablonami układu. Szablon układu jest mieszaniną wielu skryptów: języków znacznikowych, kodu wykonywanego po stronie klienta, wyrażeń Groovy i JET, własnych znaczników, wreszcie znaczników XML łączących poszczególne komponenty i szablony. Każdy taki szablon układu może zawierać wiele szablonów danego typu, np. kilka szablonów układu strony – przeznaczonych dla różnych części serwisu. Posiada on również plik kontrolny, określający standardowe szablony dla poszczególnych elementów. Można więctaki szablon przypisać do konkretnego zestawu stron – i automatycznie odpowiednie szablony do ich wszystkich rodzajów elementów. Szablony układu można również przypisywać na niższych poziomach

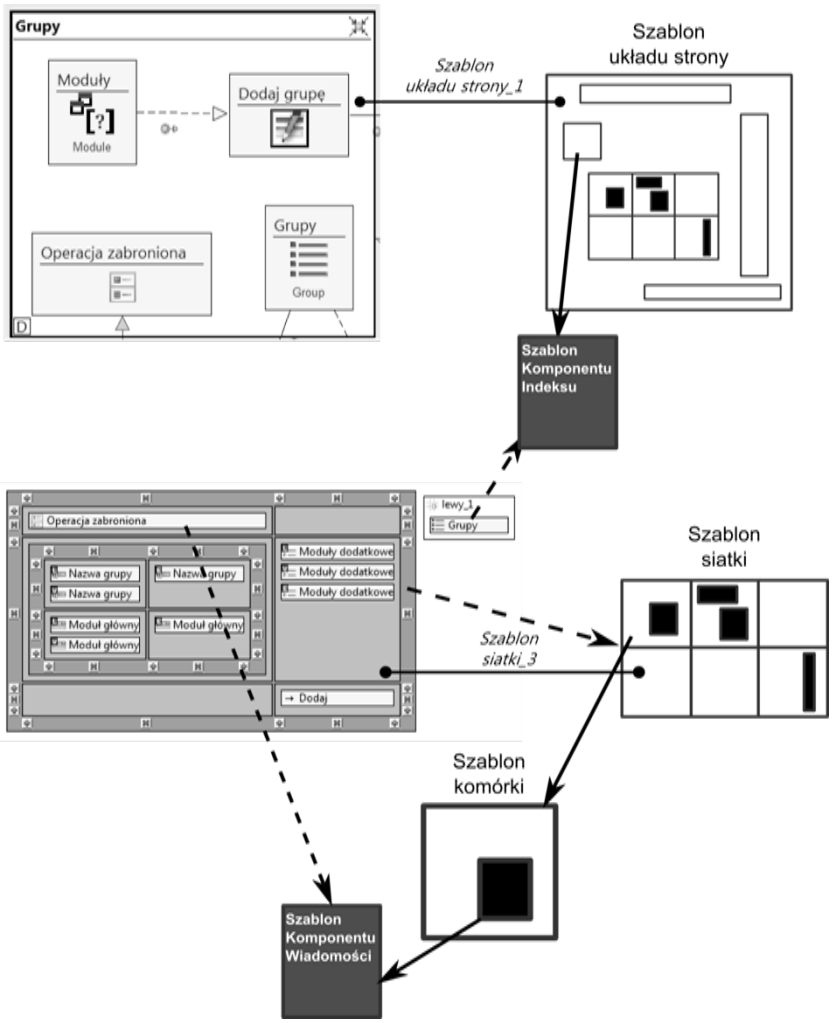
hierarchii układu elementów modelu hipertekstu, uzyskując zróżnicowanie wyglądu komponentów wewnątrz danego obszaru lub strony.

Rys. 3.31przedstawia powiązania między szablonami i komponentami na przykładzie strony umożliwiającej dodawanie grup użytkowników. Model hipertekstu tej strony zawiera 3 komponenty prezentujące dane: Komponent Indeksu – wyświetlający listę istniejących grup, Komponent Wprowadzania – udostępniający formularz dodawania nowej grupy, Komponent Wiadomości – wyświetlający komunikat w przypadku błędu utworzenia nowej grupy. Strona zawierająca te komponenty należy do grupy stron odpowiedzialnych za administrację systemem. Jednakże nie stosuje się ona do szablonu układu strony zdefiniowanego dla całej grupy – ma indywidualnie przypisany szablon o nazwie *Szablon układu strony_1,* co jest zobrazowane linią zakończoną kropkami. Komponenty Wprowadzania i Wiadomości zostały umieszczone w siatce podczas, gdy Komponent Indeksu jest umieszczony w obszarze *lewy_1*. Do wyrenderowania Komponentu Indeksu jest wykorzystywany szablon domyślny *Szablon układu strony_1*, (zobrazowany na rysunku strzałką ciągłą) oraz dane pochodzące z tego komponentu (zobrazowane na rysunku strzałką przerywaną).

Rozmieszczenie pozostałych dwóch komponentów w siatce zostało zdefiniowane w modelu hipertekstowym w sposób zróżnicowany. Komponent Wiadomości został umieszczony cały w jednej komórce siatki podczas, gdy atrybuty Komponentu Wprowadzania zostały rozmieszczone w wielu komórkach siatki.

Siatka również nie korzysta ze standardowego szablonu – został jej przypisany *Szablon siatki_3*. Szablon ten pobiera z siatki informacje o rozmieszczeniu komponentów i atrybutów. Przykładowo Komponent Wiadomości jest umieszczony w komórce korzystającej ze standardowego *Szablonu komórki*, sam również korzysta ze standardowego *Szablonu Komponentu Wiadomości*, dostarczając mu dane o swojej zawartości i parametrach sterowania szablonem.

Przedstawiony model prezentacji jest jednym z możliwych rozwiązań, wpasowujących się w strukturę WebML. Nie definiuje on modelu prezentacji w sposób sztywny, tłumacząc to udostępnieniem możliwości zaadoptowania ogólnie uznanych praktyk budowy interfejsów www.



Rys. 3.31. Ilustracja powiązań między szablonami i komponentami

Źródło: opracowanie własne

3.6 PYTANIA KONTROLNE

1. Opisz metody organizacji warstwy prezentacji aplikacji internetowej.
2. Jakie komponenty modelu hipertekstu mogą być używane w widoku serwisów?
3. Na jakiej podstawie można decydować o umieszczeniu obiektu w słowniku obiektów?
4. Wymień dwa cykle życia aplikacji.
5. Na czym polega idea szablonów?
6. W jaki sposób można mapować hierarchie encji na relacyjne bazy danych?

4

Organizacja modelu aplikacji internetowej

Cel

Omówienie organizacji modelu danych i modelu hipertekstu oraz ich reprezentacji w notacji UML. Przedstawienie zagadnień związanych z regułami przetwarzania hipertekstu oraz typowych błędów projektowych.

Plan

1. Podschematy modelu danych
2. Dekompozycja widoku witryny
3. Kolejność przetwarzania modelu hipertekstu
4. Typowe błędy w projektowaniu modelu hipertekstu
5. Reprezentacja modelu danych w notacji UML
6. Reprezentacja modelu hipertekstu w notacji UML

4.1 WZORCE W APLIKACJACH INTERNETOWYCH

WebMLwyróżnia pewne typowe wzorce (podschematy) w ramach modelu aplikacji internetowej, przekładające się na układ struktur w ramach modelu danych, a w konsekwencji również w modelu hipertekstu. Bazują one na następujących czterech klasach obiektów(Ceri i inni, 2003):

1. Obiekty szkieletowe (*ang. Core Objects*) – najważniejsze obiekty przetwarzane przez aplikację stanowiące o jej istocie/domenie.
2. Obiekty łączeniowe (*ang. Interconnection Objects*) – obiekty współtworzące powiązania, w sensie asocjacji, pomiędzy obiektami szkieletowymi.
3. Obiekty dostępowe (*ang. Access Objects*) – obiekty służące do klasyfikacji lub specjalizacji obiektów szkieletowych.
4. Obiekty personalizujące (*ang. Personalization Objects*) – obiekty służące do dostosowania aplikacji do potrzeb jej użytkowników.

PODSCHEMAT SZKIELETOWY

Podschematy szkieletowe (*ang. CoreSubschema*) skupiają się na najważniejszych obiektach zarządzanych przez aplikację, które zostały zidentyfikowane w trakcie analizy wymagań i umieszczone w słowniku obiektów. Każdy z nich, w związku z obecnością kompleksowych właściwości i wewnętrznych komponentów, może wymagać do swojej reprezentacji więcej niż jednej encji. Z tego powodu obiekty te są odwzorowywane w modelu danych w postaci całych podschematów (rys. 4.1).

Wskutek odwzorowania obiektów szkieletowych w modelu danych:

1. Obiekt ze słownika (*ang. coreconcept*) staje się encją szkieletową.
2. Właściwości obiektu, które są niepodzielne (atomowe), stają się atrybutami encji szkieletowej.
3. Właściwości obiektu o złożonej naturze, będące jego komponentami, stają się encjami połączonymi relacjami z encją szkieletową.

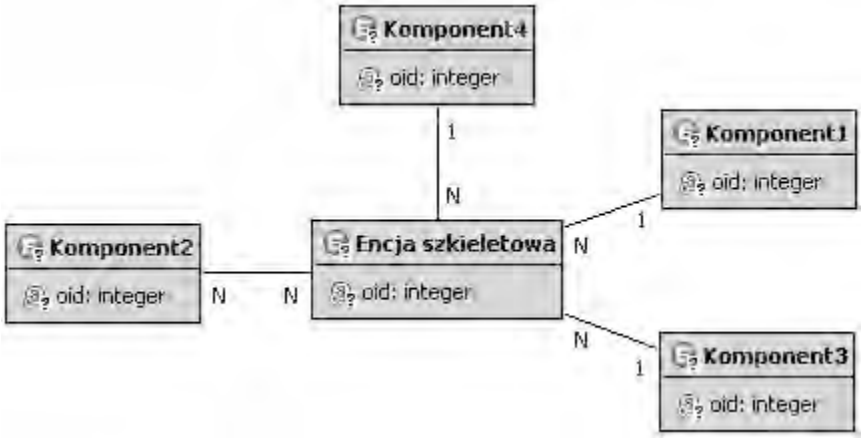
Komponenty mogą być powiązane z encją szkieletową relacjami opcjonalnymi o następującym typie:

1. jeden–do–wielu – instancja komponentu nie może istnieć bez obecności instancji

encji szkieletowej i nie może być współdzielona przez różne encje szkieletowe.

2. wiele–do–wielu – instancja komponentu może istnieć niezależnie od instancji encji szkieletowej i może być współdzielona przez różne encje szkieletowe.

Relacje wiele-do-wielu łączą komponenty, które nie są na tyle istotne, aby stać się niezależnymi wpisami w słowniku obiektów(Ceri i inni, 2003).



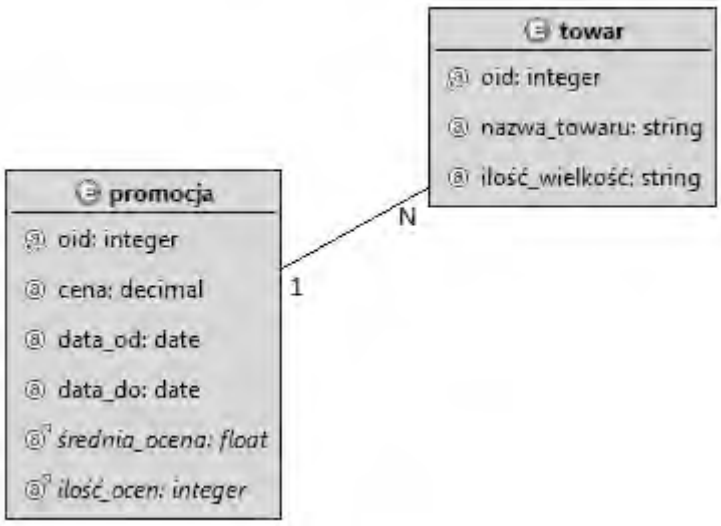
Rys. 4.1. Idea podschematu szkieletowego

Źródło: opracowanie własne

Przeniesieniem podschematu szkieletowego modelu danych (rys. 4.3) do modelu hipertekstu może być model strony internetowej wyświetlającej listę promocji (rys. 4.2).

Rys. 4.2. Model hipertekstu bazujący na podschemacie szkieletowym

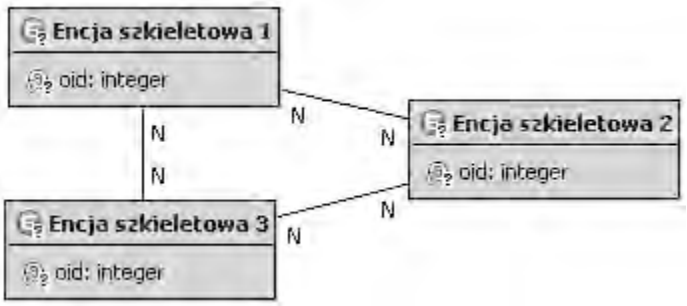
Źródło: opracowanie własne



Rys. 4.3. Przykład podschematu szkieletowego
Źródło: opracowanie własne

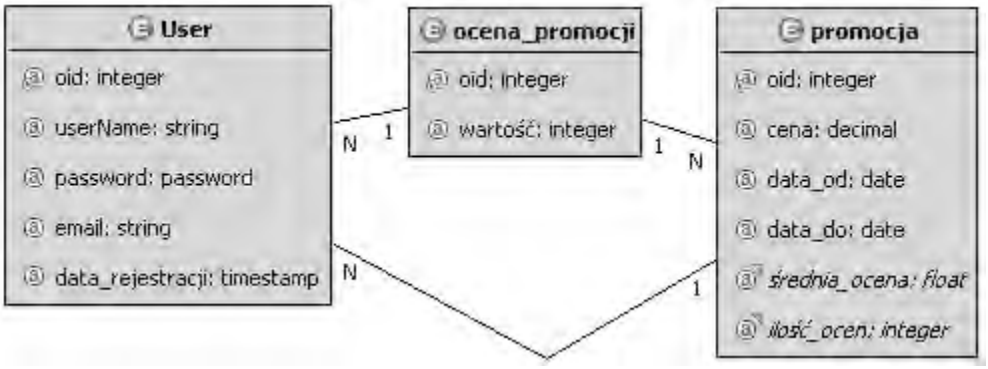
PODSCHEMATŁĄCZENIOWY

Podschemat łączeniowy (*ang. InterconnectionSubschema*) jest odwzorowaniem sieciasocjacjipomiędzy obiektami szkieletowymi, umożliwiającym nawigację pomiędzy powiązanymi ze sobą obiektami (rys. 4.4).Istnieją dwa skrajne przypadki, w których wszystkie encje szkieletowe są ze sobą połączone relacjami lub opisywany podschemat jest pusty, czyli encje szkieletowe są od siebie odizolowane(Ceri i inni, 2003).



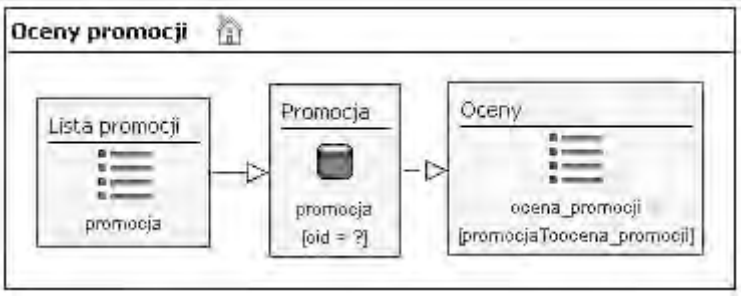
Rys. 4.4. Idea podschematu łączeniowego
Źródło: opracowanie własne

Przeniesieniem podschematu łączeniowego modelu danych (rys. 4.5) do modelu hipertekstu może być model strony internetowej wyświetlającej oceny promocji (rys. 4.6).



Rys. 4.5. Przykład podschematu łączeniowego

Źródło: opracowanie własne



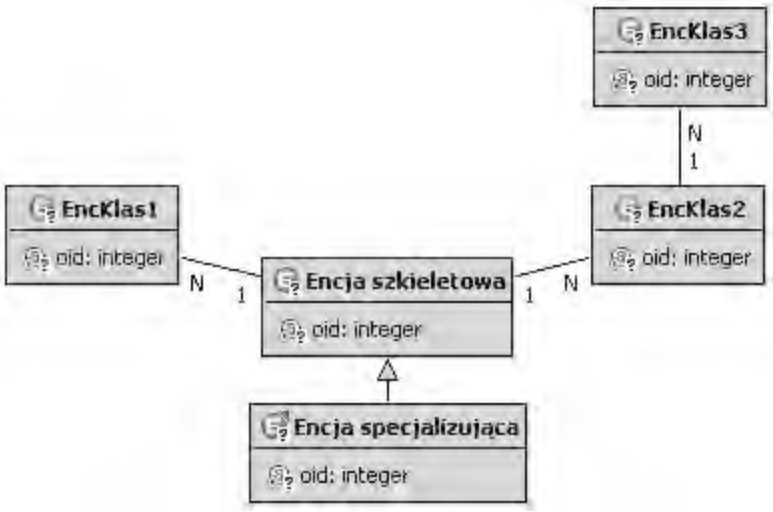
Rys. 4.6. Model hipertekstu bazujący na podschemacie łączeniowym

Źródło: opracowanie własne

PODSCHEMAT DOSTĘPOWY

Podschemat dostępowy (*ang. Access Subschema*) przedstawia obiekty pomocnicze, klasyfikujące lub specjalizująceobiekty szkieletowe, w celu zapewnienia dostępu do ich kolekcji (rys. 4.7). Identyfikacja obiektów dostępowych odbywa się głównie na podstawie specyfikacji przypadków użycia, ze zwróceniem uwagi na sposób lokalizacji obiektów przez użytkowników aplikacji oraz podział wystąpień obiektów szkieletowych

na kategorie lub kolekcje.



Rys. 4.7. Idea podschematu dostępowego
Źródło: opracowanie własne

Podschemat dostępowy może zawierać dwa typy encji dostępowych(Ceri i inni, 2003):

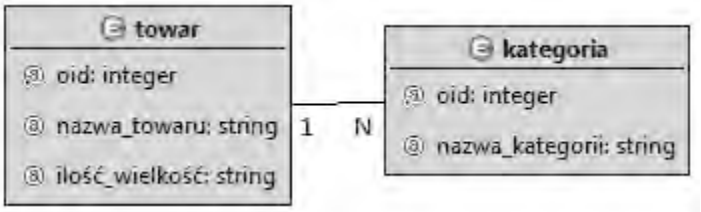
1. Encje klasyfikujące (*ang. CategorizingEntities*) – encje połączone relacjami z encjami szkieletowymi klasyfikujące lub grupujące instancje encji szkieletowych.
2. Encje specjalizujące (*ang. SpecializedSub-entities*) – encje potomne (w sensie hierarchi IS-A) encji szkieletowych, zbiór ich instancji jest podzbiorem instancji specjalizowanej encji szkieletowej.

Encje klasyfikujące mogą tworzyć hierarchię dostępu do instancji encji szkieletowych według następujących reguł(Ceri i inni, 2003):

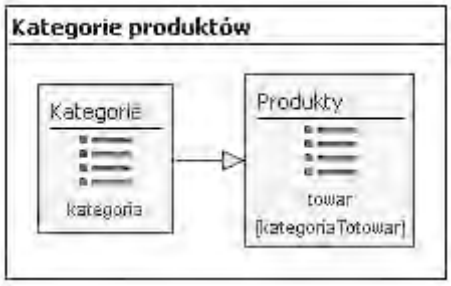
1. Encje klasyfikujące mogą być kategoryzowane (klasyfikowane).
2. Jedna encja szkieletowa może być przedmiotem więcej niż jednej kategoryzacji (klasyfikacji).
3. Jedna kategoryzacja (klasyfikacja) może dotyczyć więcej niż jednej encji szkieletowej.

Przeniesieniem podschematu dostępowego modelu danych (rys. 4.8) do modelu hipertekstu może być model strony internetowej wyświetlającej kategorie produktów

w postaci listy hierarchicznej (rys. 4.9).



Rys. 4.8. Przykład podschematu dostępowego
Źródło: opracowanie własne



Rys. 4.9. Model hipertekstu bazujący na podschemacie dostępowym
Źródło: opracowanie własne

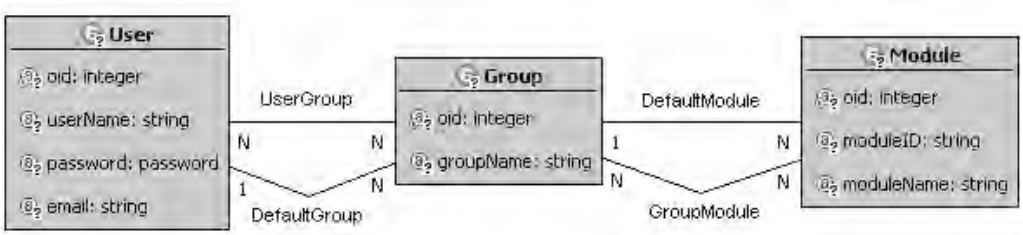
PODSCHEMAT PERSONALIZACJI

Podschemat personalizacji (*ang. PersonalizationSubschema*) jest podstawą zarządzania uprawnieniami dostępu do instancji encji szkieletowych. Zazwyczaj na jego strukturęskładają się (rys. 4.10):

1. Encja przechowująca dane użytkowników.
2. Encja przechowująca dane grup użytkowników.
3. Encja przechowująca dane chronionych stref aplikacji.
4. Relacje uwzględniające, że użytkownik ma swoją domyślną grupę użytkowników, a każda grupa użytkowników ma swoją domyślnąstrefę chronioną, do której uzyskuje dostęp.
5. Relacje uwzględniające, że użytkownik może należeć do wielu grup użytkowników

jednocześnie, a jedna grupa użytkowników może mieć dostęp do wielu chronionych stref aplikacji.

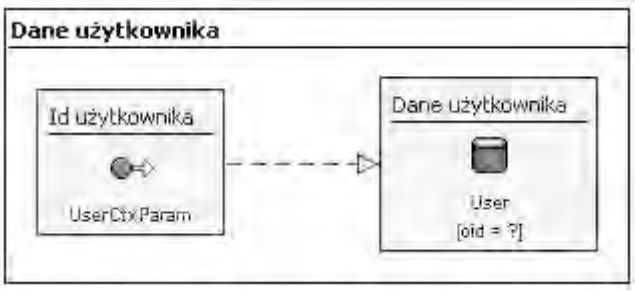
Typowy scenariusz uzyskiwania dostępu do strefy chronionej aplikacji (logowania do aplikacji) polega na uwierzytelnieniu użytkownika oraz autoryzacji na podstawie uprawnień jego domyślnej grupy, skutkującej wyświetleniem określonej strony strefy chronionej domyślnej dla tej grupy (Ceri i inni, 2003).



Rys. 4.10. Przykład podschematu personalizacji

Źródło: opracowanie własne

Przeniesieniem podschematu personalizacji modelu danych (rys. 4.10) do modelu hipertekstu może być model strony internetowej wyświetlającej dane zalogowanego użytkownika na podstawie jego identyfikatora będącego parametrem kontekstowym aplikacji (rys. 4.11).



Rys. 4.11. Model hipertekstu bazujący na podschemacie personalizacji

Źródło: opracowanie własne

ORGANIZACJA MODELU DANYCH BAZUJĄCA NA PODSCHEMATACH

W przypadku budowy modelu danych w oparciu o przedstawione w tym rozdziale podschematy, WebML zaleca procedurę składającą się z następujących etapów(Ceri i inni, 2003):

1. Utworzenie podschematu szkieletowego.
2. Dodanie podschematu łączeniowego poprzez zamodelowanie asocjacji pomiędzy encjami szkieletowymi.
3. Dodanie podschematu dostępowego poprzez zastosowanie przedstawionych mechanizmów klasyfikacji oraz specjalizacji.
4. Dodanie podschematu personalizacji poprzez wprowadzenie encji i relacji służących autoryzacji i uwierzytelnianiu użytkowników oraz ich powiązanie z encjami szkieletowymi.

Opisana wyżej sekwencja kroków nie jest obowiązkowa i może ulec pewnym modyfikacjom. Efektem końcowym jest powstanie modelu danych złożonego z czterech przedstawionych podschematów (przykład na rysunku 4.12).

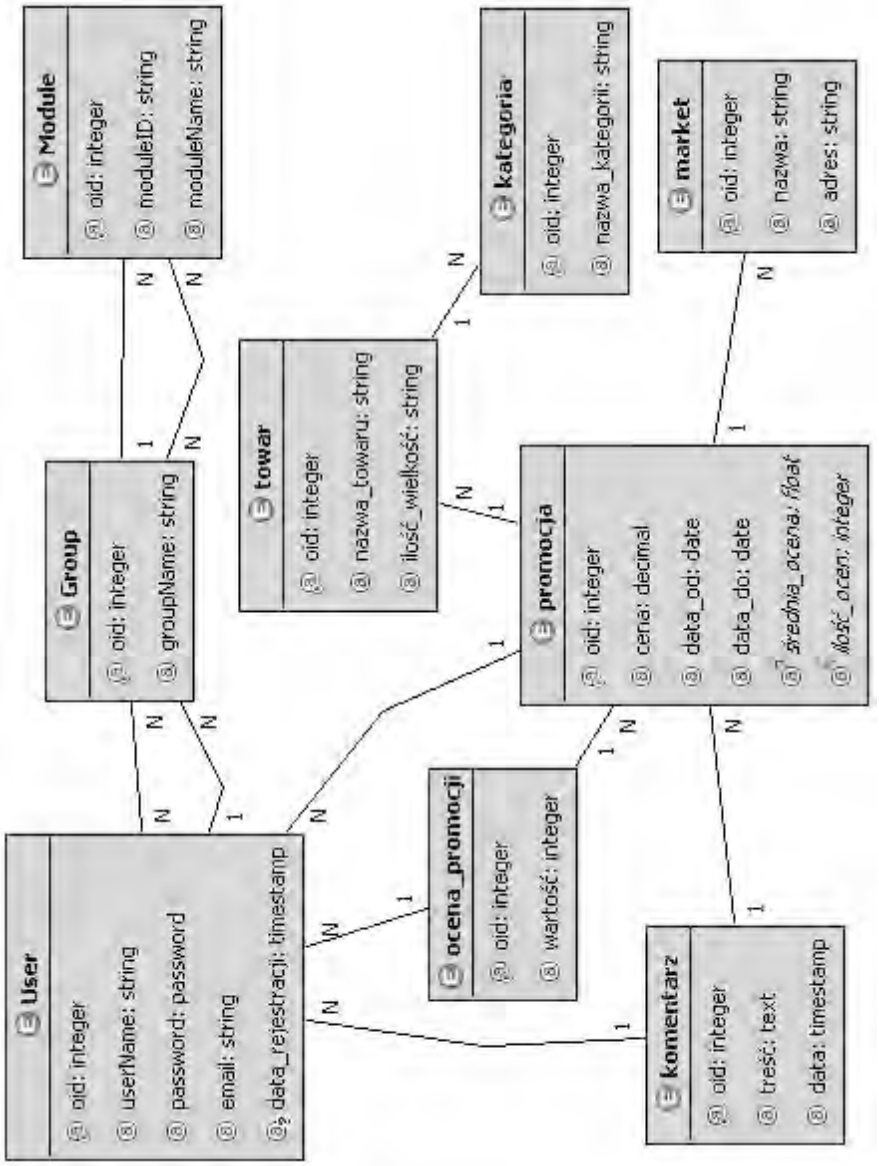
Niestety powyższa metodyka modelowania danych nie zawsze sprawdza się w przypadku warstw danych o dużej złożoności lub zawierających rozszerzenia istniejącej struktury bazy danych projektowanej zgodnie z inną metodyką. Model wynikowy bywa na tyle skomplikowany, że wyróżnienie tych wzorców jest jedynie zbędnym utrudnieniem.

4.2 ORGANIZACJA MODELU HIPERTEKSTU

IDEA DWUPOZIOMOWEJDEKOMPOZYCJIWIDOKUWITRYNY

Elementy modelu hipertekstu są grupowane w strukturach następujących typów:

1. Widok witryny (*ang. Site View*) – najogólniejsza struktura grupująca elementy modelu hipertekstu. Może zawierać obszary i strony. Biorąc pod uwagę specyfikę notacji WebML i jej elastyczność trudno scharakteryzować cechy wspólne funkcjonalności realizowanej przez pojedynczy widok.

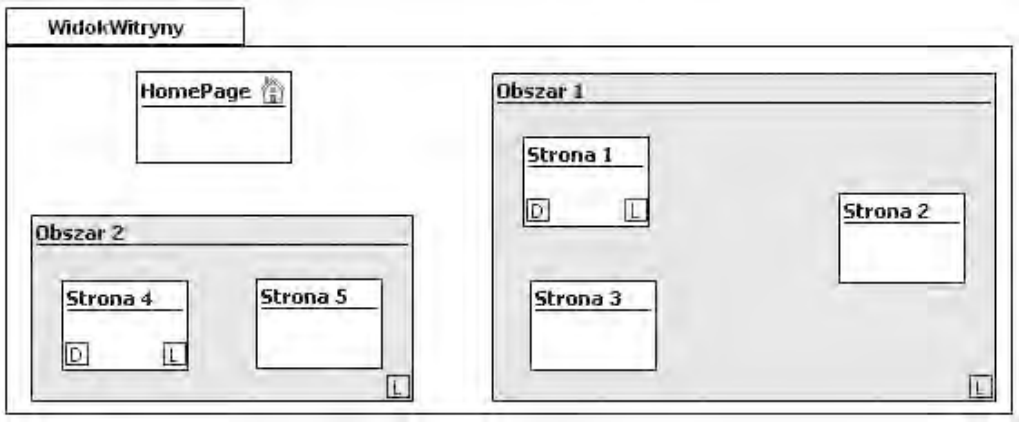


Rys. 4.12. Model hipertekstu bazujący na podschemacie personalizacji
Źródło: opracowanie własne

2. Widok serwisów (*ang. Service View*) – struktura grupująca wyłącznie komponenty operacji, hybrydowe i serwisów,w celu realizacji pewnych procesów lub zadań. Nie może zawierać w sobie obszarów i stron.

3. Obszar (*ang. Area*) – struktura wchodząca w skład widoku witryny grupująca elementy modelu hipertekstu realizujące powiązaną tematycznie funkcjonalność.
4. Strona(*ang. Page*) – najmniejsza struktura grupująca elementy modelu hipertekstu. Może być składową obszaru lub widoku witryny. Nie może zawierać obszarów, widoków witryny ani widoków serwisów. Symbolizuje pojedynczą stronę wyświetlaną w przeglądarce stron internetowych.
5. Transakcja (*ang. Transaction*) – struktura grupująca komponenty modelu hipertekstu, symbolizujące operacje, w sekwencję operacji, które muszą być wykonane jako całość albo w ogóle.

Organizacja modelu hipertekstu polega na identyfikacji struktur realizujących cel aplikacji, ich pogrupowaniu oraz reprezentacji przy pomocy komponentów składających się na notację WebML. Końcowym efektem tego procesu jest tzw. dwupoziomowa dekompozycja widoków witryny (rys. 4.13), w ramach której obszary składają się ze stron, a widoki witryny z obszarów i stron(Ceri i inni, 2003).



Rys. 4.13. Idea dwupoziomowej dekompozycji widoku witryny
Źródło: opracowanie własne

Symbolika określająca widoczność stron bazuje na oznaczeniach(Ceri i inni, 2003):

1. H (*ang. Home Page*) – strona domowa, pierwsza strona prezentowana użytkownikowi po wejściu do widoku witryny.
2. D (*ang. DefaultPage*) – strona domyślna obszaru, pierwsza strona prezentowana

użytkownikowi po przejściu do obszaru.

3. L (*ang. LandmarkPage*) – „strefa zrzutu", strona osiągalna ze wszystkich pozostałych stron wewnątrz zawierającego ją modułu (widoku witryny lub obszaru).

KOLEJNOŚĆ PRZETWARZANIA HIPERTEKSTU

Podstawową jednostką przetwarzania w notacji WebML jest strona. Dostęp do strony, bądź składających się na nią komponentów, można uzyskać przy pomocy linków pochodzących z: innych stron, komponentów składających się na stronę lub wskutek wykonania pewnych operacji. Zawartość strony jest przetwarzana, jeśli:

1. Strona została otworzona linkiem pochodzącym z innej strony.
2. Działania użytkownika doprowadziły do zmiany zawartości wyświetlonej strony.
3. Została wywołana operacja, której wykonanie prowadzi do miejsca wywołania.

Komponenty pod względem możliwości przetworzenia dzielą się na:

1. Bezkontekstowe (*ang. Context – freeUnits*) –mogą być zawsze przetworzone niezależnie od sposobu dostępu do strony.
2. Zewnętrznie zależne (*ang. Externally Dependent Units*) – ich przetworzenie zależy od wartości dostarczonej spoza strony lub jako wynik operacji.
3. Wewnętrznie zależne (*ang. Internally Dependent Units*) – ich przetworzenie zależy od wartości dostarczonej z obrębu strony, w ramach której się znajdują.

Komponent, aby został przetworzony, musi spełniać następujące warunki:

1. Dostarczono wartości dla wszystkich niezbędnych parametrów wejściowych.
2. Przetworzono wszystkie komponenty dostarczające wartości dla opcjonalnych parametrów wejściowych.

Algorytm przetwarzania strony (listing 4.1) w pierwszej kolejności oznacza jako zdolne do przetworzenia wszystkie komponenty bezkontekstowe oraz zależne zewnętrznie, posiadające wystarczający zestaw wartości wejściowych. Następnie, na podstawie sieci linków, wybiera spośród pozostałych komponentów, jeśli jakieś pozostały, komponenty spełniające wymagania po przetworzeniu poprzedniego zbioru komponentów. Algorytm kończy się w momencie, gdy zbiór nieprzetworzonych komponentów jest pusty(Ceri i inni, 2003).

Listing 4.1. Algorytm przetwarzania komponentów strony

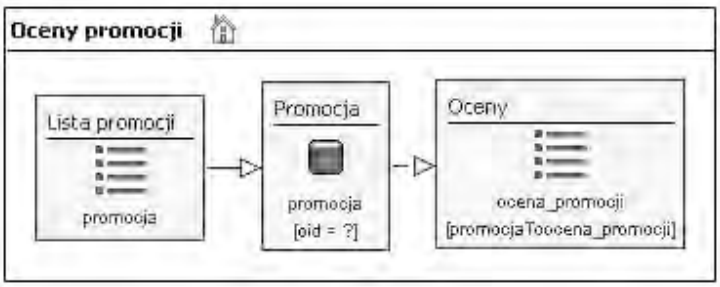
```
WEJŚCIE: zbiór komponentów do przetworzenia
WYJŚCIE: zbiór przetworzonych komponentów
PROCEDURA:
WHILE (istnieją komponenty do przetworzenia) DO
  IF istnieje komponent U taki, że
    (dostarczono wartości dla jego wszystkich wymaganych
     parametrów wejściowych
     AND
wszystkie komponenty dostarczające wartości dla jego
opcjonalnych parametrów wejściowych zostały przetworzone)
  THEN
    przypisz wartości do parametrów komponentu U;
    przetwórz komponent U;
  ELSE HALT;
END DO;
```

Źródło:(Ceri i inni, 2003)

Dla przykładu, przed wyświetleniem strony *Oceny promocji* (przedstawiona na rysunku 4.14) następuje przetworzenie wszystkich komponentów bezkontekstowych wchodzących w jej skład, czyli komponentu *Lista promocji* (nie dochodzą do niego linki, nie zależy od innych komponentów modelu hipertekstu).

Zakładając, że link wychodzący z komponentu *Lista promocji* nie jest automatyczny, strona wyświetlona użytkownikowi aplikacji będzie zawierać wyłącznie listę instancji encji *promocja*. W momencie, gdy użytkownik wybierze jedną z promocji, nastąpi ponowne przetworzenie zawartości strony – zostanie przekazana wartość dla wymaganego parametru *oid*komponentu *Promocja*, po którego przetworzeniu zestaw niezbędnych wartości zostanie przekazany linkiem do komponentu *Oceny*, który zostanie przetworzony jako ostatni na stronie. W efekcie na stronie pojawi się wybrana instancja encji *promocja* oraz lista, powiązanych z nią relacją, instancji encji *ocena_promocji*.

Zakładając, że link wychodzący z komponentu *Lista promocji* jest automatyczny, to zaraz po przetworzeniu komponentu *Lista promocji* link przekaże niezbędne wartości dla komponentu *Promocja*, po którego przetworzeniu wymagane wartości zostaną przekazane do komponentu *Oceny* powodując jego przetworzenie. W efekcie użytkownik za jednym kliknięciem otrzyma stronę zawierającą: listę instancji encji *promocja*, instancję encji *promocja* wskazaną przez link automatyczny oraz listę instancji encji *ocena_promocji* powiązanych relacją z wyświetloną instancją encji *promocja*.

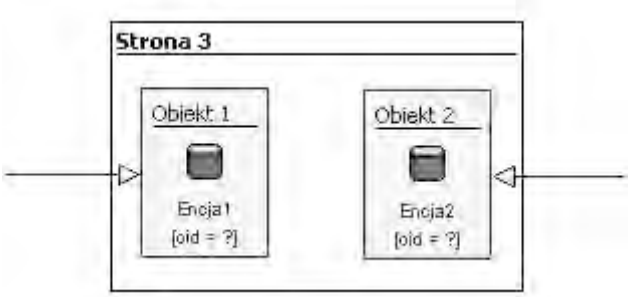


Rys. 4.14. Idea dwupoziomowej dekompozycji widoku witryny
Źródło: opracowanie własne

PRZYKŁADYBŁĘDÓW W PROJEKTOWANIUHIPERTEKSTU

Komponenty w modelu hipertekstu nie mogą być łączone przypadkowo. Fragmenty modelu, które nie przestrzegają reguł przetwarzania oraz poprawności, nie zostaną przetworzone. Najczęściej popełniane błędy są związane z niewłaściwym dostarczaniem wartości dla parametrów komponentów lub brakiem determinizmu.

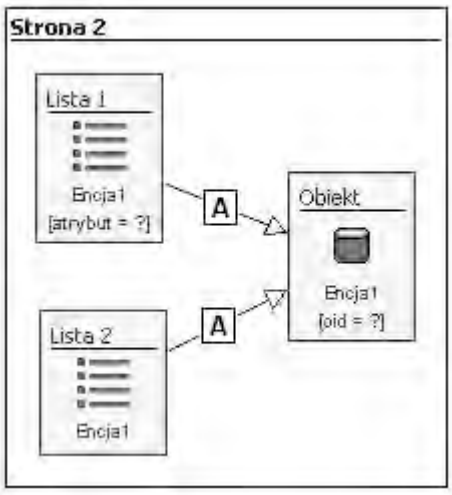
Na rysunku 4.15 przedstawiono model hipertekstu wyświetlający pojedyncze instancje dwóch różnych encji. Strona może zostać otworzona jednym z dwóch linków dostarczających wartości dla selektorów komponentów danych. Błąd polega na tym, że żaden z linków nie dostarcza wartości dla obydwu komponentów, a nie można otworzyć strony jednocześnie używając obydwu linków. W związku z tym niezależnie od sposobu przejścia do strony, jeden z komponentów danych nie otrzyma wartości wymaganej przez selektor.



Rys. 4.15. Przykład niewłaściwego dostarczenia wartości parametrów wejściowych komponentu

Źródło: (Ceri i inni, 2003)

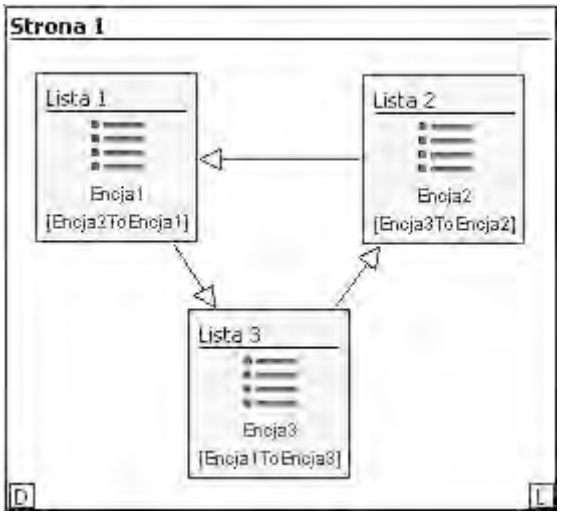
Na rysunku 4.16 przedstawiono model hipertekstu wyświetlającego dwie listy instancji encji oraz szczegóły jednej z instancji wskazanej przez link automatyczny. Model ten zawiera dwa błędy. Pierwszy z nich polega na nie dostarczeniu niezbędnej wartości dla selektora komponentu *Lista 1*. Drugi błąd jest związany z brakiem determinizmu, tzn. dwa linki automatycznie jednocześnie próbują przekazać identyfikator instancji encji *Encja 1* do wyświetlenia przez komponent *Obiekt*. Obydwa linki są równorzędne, więc nie można rozstrzygnąć, którą wartość wybrać.



Rys. 4.16. Przykład niedeterministycznego modelu hipertekstu

Źródło:(Ceri i inni, 2003)

Na rysunku 4.17 przedstawiono model hipertekstu wyświetlającego trzy listy instancji encji powiązanych ze sobą relacjami. Wszystkie komponenty indeksu posiadają selektory, dla których wymagane wartości są dostarczane przez linki. Pozornie model wydaje się być poprawny, jednakże stoi w sprzeczności z zasadą, że aby przetworzyć komponent, wszystkie komponenty, od których zależy, powinny być przetworzone jako pierwsze.Nie jest możliwe wyróżnienie komponentu, od którego powinien rozpocząć się proces przetwarzania.



Rys. 4.17. Przykład błędnych połączeń pomiędzy komponentami
Źródło: (Ceri i inni, 2003)

4.3 WEBML A UML

Ojcowie WebML opisując stworzoną przez siebie notację wspominają, że wyrasta ona z narzędzi do projektowania oprogramowania o ugruntowanej pozycji wśród deweloperów (**webml.org**). Razem z WebML lub jako towarzyszące metody opisu funkcjonują: notacja UML, diagramy związków encji, język OCL i jego pochodne oraz inne tekstowe, formalne i półformalne notacje.

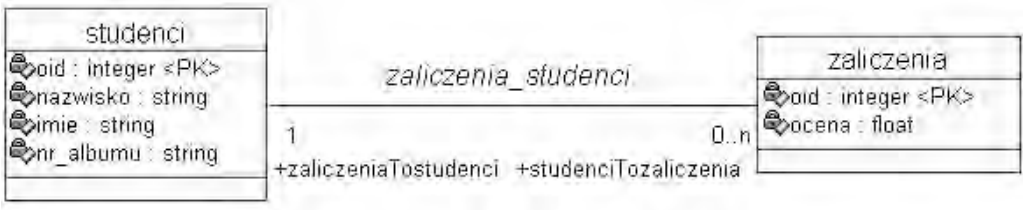
MODELOWANIE WARSTWY DANYCH W UML

Elementy modelu danych zaproponowane przez WebML mogą być równie dobrze reprezentowane w postaci diagramu klas w notacji UML (rys. 4.19) lub w postaci diagramu związków encji (rys. 4.18).Oczywiście UML ma sporą przewagę pod względem możliwości, jednakże należy liczyć się ze znaczącym stopniem skomplikowania modelu.

Semantyka na poziomie wystarczającym do modelowania warstwy danych jest bardzo zbliżona. Relacje WebML odpowiadają asocjacjom UML, obydwa koncepty są charakteryzowane przez role, nazwy ról i liczność. Koncept generalizacji i specjalizacji nie różni się pomiędzy notacjami. Największe różnice pojawiają się pomiędzy klasami a encjami. Koncept klasy generalizuje koncept encji, co pozwala na specyfikowanie nie tylko atrybutów, ale również metod operujących na instancjach klasy(Ceri i inni, 2003).

Rys. 4.18. Fragment modelu danych w notacji WebML

Źródło: opracowanie własne



Rys. 4.19. Fragment modelu danych w notacji UML

Źródło: opracowanie własne

MODELOWANIE PREZENTACJI TREŚCI W UML

Mapowanie prezentacji treści z notacji WebML na notację UML odbywa się zgodnie z następującymi wytycznymi(Ceri i inni, 2003):

1. Widoki witryny i obszary są reprezentowane jako zagnieżdżone pakiety.
2. Strony są reprezentowane jako klasyfikatory o stereotypie <<*page*>>.
3. Komponenty prezentacji danych są reprezentowane jako klasy umieszczane wewnątrz konstruktu strony i oznaczane stereotypem będącym nazwą typu komponentu WebML np. <<*data*>>, <<*index*>>, <<*hierarchicalindex*>>, etc..
4. Encja źródłowa i selektor komponentu są reprezentowane jako asocjacja, pomiędzy klasą reprezentującą komponent WebML a klasą reprezentującą encję źródłową, opisana przez wyrażenie OCL będące warunkiem selektora.
5. Linki pomiędzy stronami lub komponentami aplikacji są reprezentowane jako skierowane asocjacje oznaczone przez stereotypy <<*link*>>, <<*automatic*>> lub <<*transport*>> oraz opis przekazywanych parametrów.

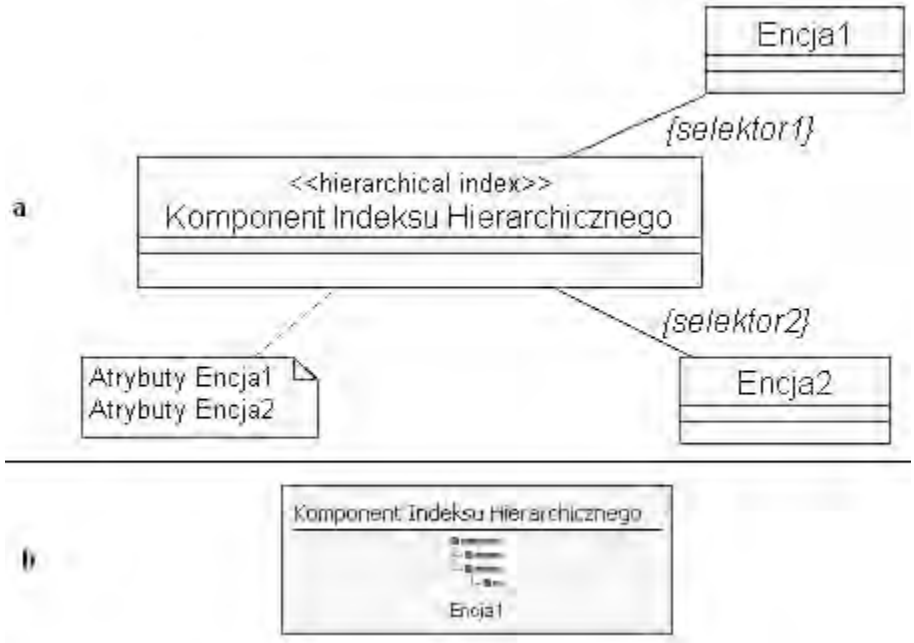
Dla przykładu odpowiednik komponentu danych oraz komponentu indeksu hierarchicznego w notacji UML przedstawiono odpowiednio na rysunkach 4.20 i 4.21.

MODELOWANIE OPERACJI W UML

Mapowanie operacji z notacji WebML na notację UML odbywa się zgodnie z następującymi wytycznymi (Ceri i inni, 2003):

1. Komponenty operacji są reprezentowane jako klasy,stanowiące interfejs pomiędzy komponentem inicjującym operację a encjami będącymi jej przedmiotem,oznaczane stereotypem będącym nazwą typu komponentu WebML np. <<*create*>>, <<*delete*>>, <<*connect*>>, etc..Klasa reprezentująca operację udostępnia na zewnątrz tylko jedną metodę – sposób jej wykonania jest nieistotny dla komponentu inicjującego operację.
2. Komponenty logowania, wylogowania i poczty są reprezentowane w postaci klas posiadających jedną metodę zgodną z nazwą komponentu.

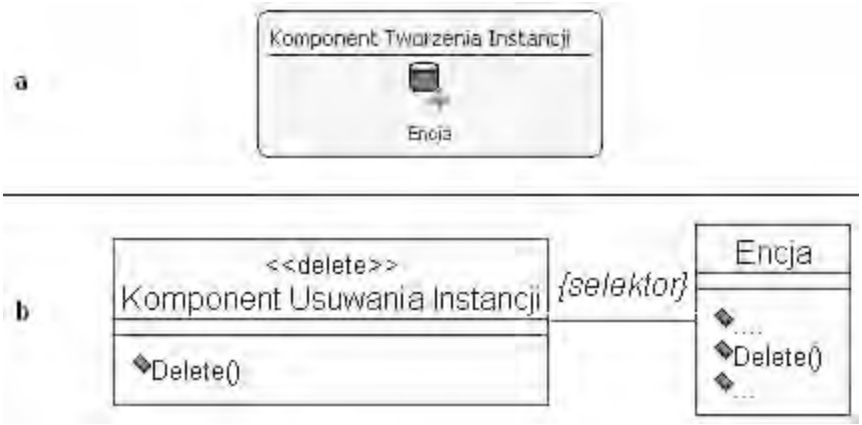
Rys. 4.20. Komponent danych w notacji WebML (a) i UML (b)
Źródło: opracowanie własne



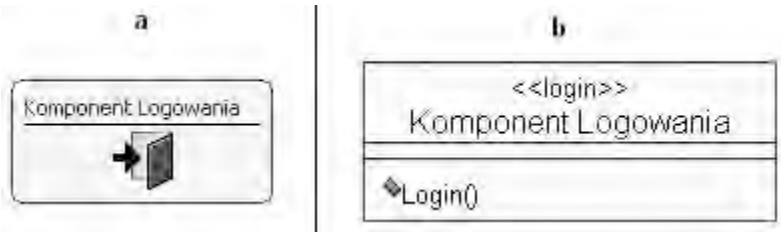
Rys. 4.21. Komponent Indeksu Hierarchicznego w notacji UML (a) i WebML (b)
Źródło: opracowanie własne

3. Linki wychodzące z komponentów operacji są reprezentowane jako skierowane asocjacje oznaczane przez stereotypy *<<OK>>,<<KO>>* lub *<<transport>>*i posiadające opis przekazywanych parametrów.
4. Transakcje są reprezentowane jako klasy posiadające metody *start()*, *commit()* i *abort()* oraz połączone asocjacjami (w sensie kompozycji) z klasami operacji.

Dla przykładu odpowiednik Komponentu Usuwania Instancji oraz Komponentu Logowania w notacji UML przedstawiono odpowiednio na rysunkach 4.22 i 4.23.



Rys. 4.22. Komponent Usuwania Instancji w notacji WebML (a) i UML (b)
Źródło: opracowanie własne

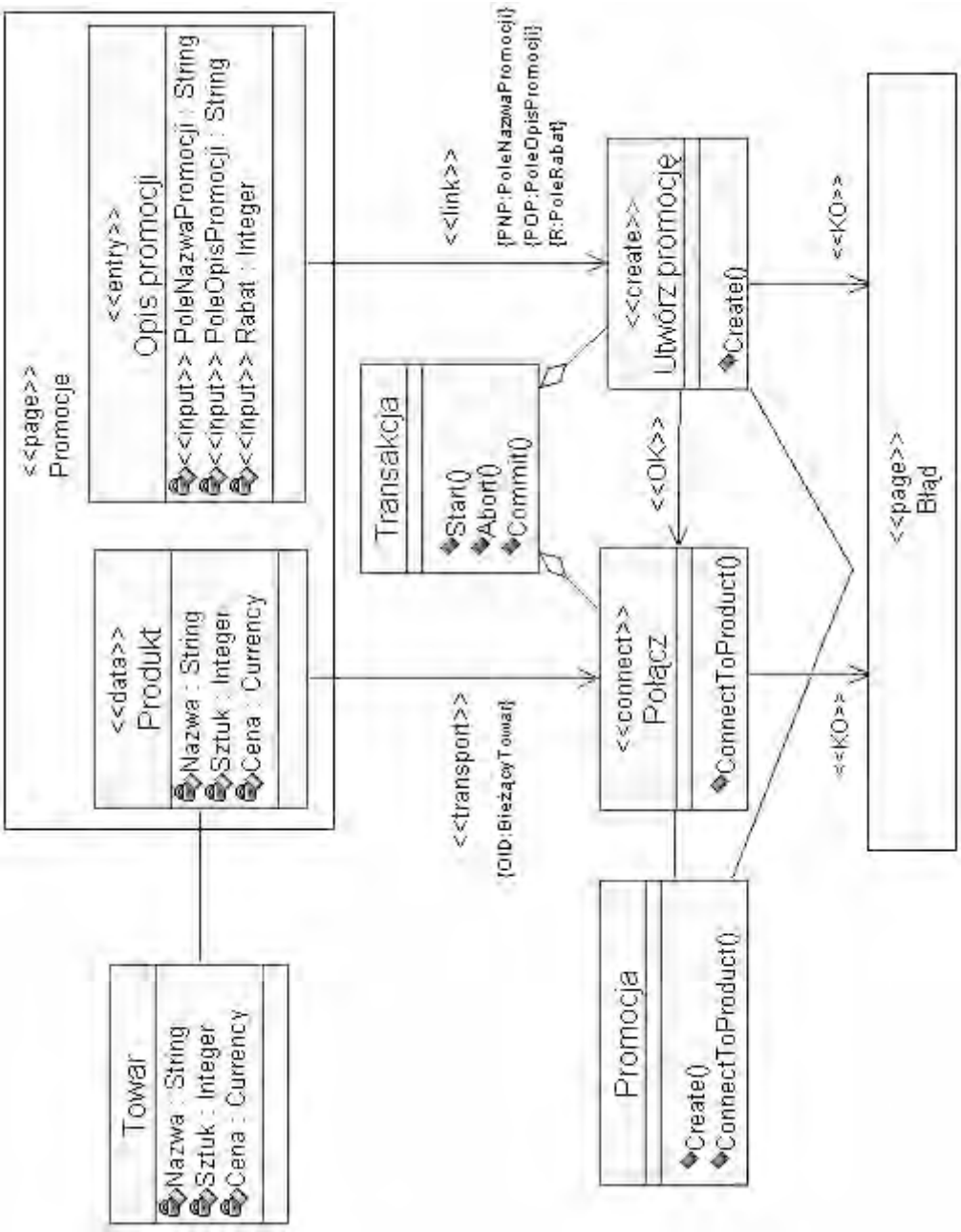


Rys. 4.23. Komponent Logowaniaw notacji WebML (a) i UML (b)
Źródło: opracowanie własne

MODELOWANIE WZORCA ZARZĄDZANIA TREŚCIĄ W UML

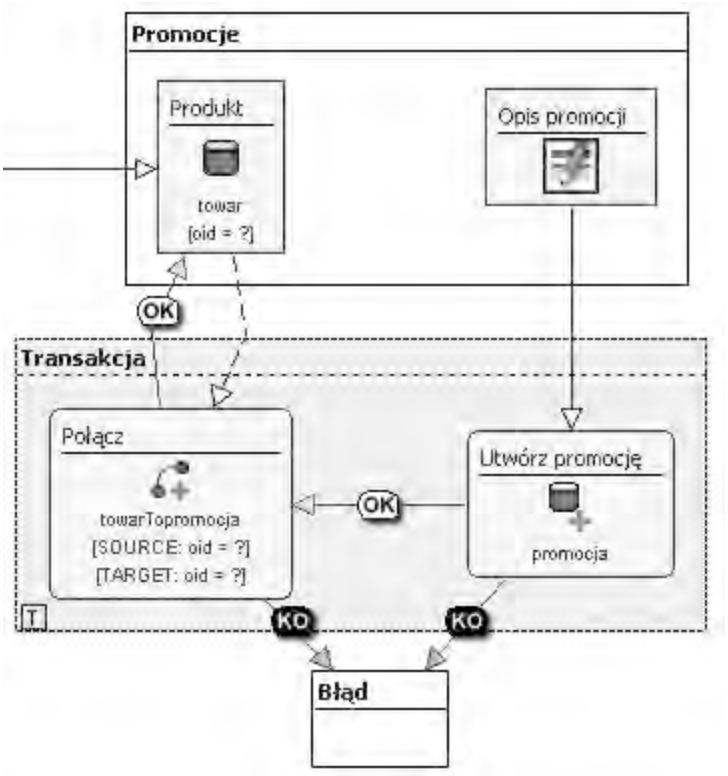
UML dzięki swoim rozbudowanym możliwościom pozwala na budowanie modeli dokładnie przedstawiających działanie i strukturę aplikacji, jednakże nie zawsze jest to niezbędne. Wybór pomiędzy UML a WebML, to wybór pomiędzy precyzją opisu a prezentacją ogólnej koncepcji działania aplikacji.

Dowodem tego może być reprezentacja wzorca zarządzania treścią w notacji WebML (rys. 4.25) i UML (rys. 4.24). Obydwa rysunki przedstawiają zapisywanie w bazie danych nowych promocji obejmujących wybrane produkty, przy czym ciąg operacji jest traktowany jako transakcja. W przypadku powodzenia transakcji jest wyświetlana strona *Promocje*, w przeciwnym razie strona *Błąd*.



Rys. 4.24. Przykład wzorca zarządzania treścią w notacji UML

Źródło: opracowanie własne



Rys. 4.25. Przykład wzorca zarządzania treścią w notacji WebML
Źródło: opracowanie własne

4.4 PYTANIA KONTROLNE

1. Na podstawie rysunku 4.12 spróbuj wyodrębnić podschematy modelu danych.
2. Zmapuj model WebML hipertekstu usuwającego promocje na UML.
3. Jak dzieli się komponenty modelu hipertekstu?
4. W jaki sposób jest zorganizowany model hipertekstu?
5. Podaj przykład błędnego modelu hipertekstu innego niż przedstawione w tym rozdziale.
6. Jaka jest kolejność przetwarzania elementów modelu hipertekstu?

5

Proces modelowania w WebML

Cel

Przedstawienie implementacji technologii WebML w narzędziu WebRatio. Przedstawienie typowych rozwiązań projektowania modelu hipertekstu uwzględniających specyfikę i ograniczenia narzucane przez wybrane narzędzie projektowania WebML.

Plan

1. Przedstawienie WebRatio jako narzędzia implementującego technologię WebML
2. Omówienie komponentów, struktury i zasad projektowania w WebRatio
3. Zapoznanie z modelem personalizacji narzucanym przez WebRatio
4. Przedstawienie procesu rozbudowy modelu hipertekstu

5.1 IMPLEMENTACJA TECHNOLOGII WEBML

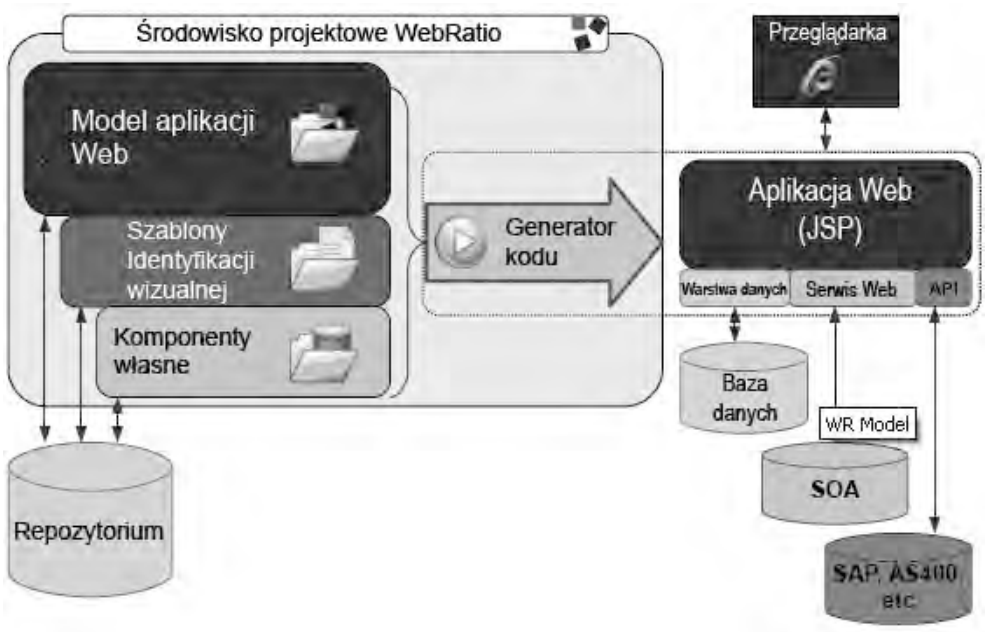
IDE WebRatio jest środowiskiem, opracowanym przez twórców metodyki WebML na bazie platformy Eclipse, określanym jako MDDE (*ang. Model Driven Development Environment*). Umożliwia wytwarzanie aplikacji internetowych na podstawie modelu (projektu) stworzonego przy użyciu notacji WebML(Rys. 5.1). Kod aplikacji jest generowany w języku JSP (*ang. Java Server Pages*), w związku z tym jest możliwe jego umieszczenie na dowolnym serwerze obsługującym ten język, m.in. Apache Tomcat, Jboss, Resi, IBM WebSphere. IDE WebRatio, dzięki wsparciu dla biblioteki Hibernate, pozwala na współpracę z większością oferowanych na rynku baz danych, dla których są dostępne sterowniki JDBC, włączając bazy danych Oracle 8i/9i/10g, DB2, MySQL, SQL Server i PostgreSQL (WebRatio, 2010), (Żyła i Kęsik, 2010b).Generowanie kodu aplikacji, zgodnie zpostulatem MDE dokonuje się automatycznie i nie wymaga ingerencji użytkownika.

Model danych jest zgodny z notacją WebML. Służy zarówno jako źródło dostępnych parametrów dla modelu hipertekstu, jak i wygenerowaniu struktury bazy danych.Możliwe jest przy tym wykorzystanie istniejącej już bazy danych i automatyczne dostosowanie do niej istniejącego modelu oraz podział modelu danych jednej aplikacji na kilka fizycznych baz (bazy danych mogą być różnego typu). Z wykorzystaniem kreatora synchronizacji można: zaimportować strukturę pustej lub wypełnionej bazy danych, dokonać modyfikacji struktury bazy, a także utworzyć od podstaw bazę danych zgodną z modelem.

Na podstawie modelu hipertekstu oraz prezentacji są generowane dynamiczne strony JSP z uwzględnieniem najbardziej powszechnych standardów HTML, CSS, Flash i AJAX. Rolą użytkownika IDE WebRatio jest budowa modelu danych i hipertekstu, ewentualna modyfikacja reguł generowania kodu (wraz z możliwościątworzenia własnych komponentów),wreszcie wybór i modyfikacja parametrów modelu prezentacji odpowiadającego za wygląd interfejsu aplikacji.

Warto wspomnieć, że twórcy IDE WebRatio twierdzą, że ich narzędzie można wykorzystać na każdym etapie cyklu życia aplikacji – od określenia wymagań, przez prototypowanie i wypuszczenie wersji finalnej, do korygowania błędów i rozbudowy aplikacji. Ponadto deklarują wsparcie dla interfejsu Web 2.0, aplikacji działających

w czasie rzeczywistym oraz możliwość projektowania dla Web, WAP i PDA w celu tworzenia aplikacji dostępnych dla wielu platform (WebRatio, 2010).



Rys. 5.1 Schemat środowiska programistycznego IDE WebRatio
Źródło: (WebRatio, 2010)

5.1.1 KOMPONENTY SKŁADOWE ŚRODOWISKA WEBRATIO

Kompleksowość środowiska wynika z wykorzystania wielu komponentów wspierających jego funkcjonowanie, wśród których warto wspomnieć o(Żyła i Kęsik, 2010b):

1. Apache Ant– narzędzie służące do zautomatyzowania procesu budowania oprogramowania. Zostało napisane w języku Java, co pozwoliło na jego wykorzystanie w wielu środowiskach programistycznych. Umożliwia:
   - definiowanie budowanych obszarów oraz zależności pomiędzy nimi,
   - automatyczne pozyskiwanie kodu źródłowego z repozytoriów,
   - budowanie aplikacji poprzez kompilację niezbędnych plików źródłowych w odpowiedniej kolejności.

Pliki konfiguracyjne narzędzia mają postać dokumentów XML (Apache Ant, 2011).

2. Apache POI– „Apache POI – Java API To Access Microsoft Format Files", zbiór interfejsów służących manipulowaniu plikami w formatach bazujących na Microsoft OLE 2 CompoundDocument oraz OpenXML przy użyciu języka Java. Obecnie wspierane są m.in. pliki: XLS, DOC, XLSX, DOCX, PPTX(Apache POI, 2011).
3. ApacheStruts – elastyczneśrodowisko bazujące na technologii Java, dokumentach XML oraz pakietach Apache Commons, składające się z komponentów odpowiadających strukturze MVC:
   - Controller,
   - Model – współpracuje z powszechnie używanymi technologiami dostępu do danych, takimi jak: JDBC, EJB, Hibernate, Object Relational Bridge,
   - View – współpracuje m.in. z: JSP, JSTL, JSF i XSLT (The apache software foundation, 2011).
4. Hibernate– narzędzie przeznaczone dla środowiska Java, wykorzystujące technikę ORM (*ang. Object/RelationalMapping*) polegającą na mapowaniu reprezentacji danych z modelu obiektowego na relacyjny model danych. Pozwala na mapowanie klas języka Java na tabele bazy danych, typów danych języka Java na typy danych charakterystyczne dla języka SQL oraz dostarcza narzędzia do dwukierunkowej komunikacji między bazą danych i aplikacją, z automatycznym użyciem sterowników JDBC (JBoss community, 2011).
5. Java Mail – niezależny od platformy i protokołu framework pozwalający na budowanie aplikacji w technologii Java zarządzających wiadomościami e-mail. Dostarcza narzędzia do odbierania i wysyłania wiadomości oraz definiowania dostawców usług (*ang. Service Providers*). Pozwala na obsługę serwerów IMAP, POP3 i SMTP (Oracle, 2011a).
6. Quartz–framework o otwartym kodzie, służący do tworzenia harmonogramów zadań dla systemów informatycznych zbudowanych w oparciu o J2EE lub J2SE, wspierający JTA (*ang. Java Transaction API*), czyli mechanizm transakcji rozproszonych. Może być użyty w dużych aplikacjach komercyjnych, w których zadania są zdefiniowane jako EJBs (*ang. Enterprise Java Beans*) (Quartz, 2011).

Bazujące na tych komponentach narzędzie cechuje się interfejsem typowym dla

rodziny wielu narzędzi budowy oprogramowania bazujących na Eclipse. Proces tworzenia i implementacji graficznego języka DSL w Eclipse został opisany w (Voelter, Kolb, Efftinge i Haase, 2006).

5.1.2 STRUKTURA ŚRODOWISKA WEBRATIO

Struktura folderów środowiska WebRatio przedstawia się następująco:

1. *Tomcat* – folder zawierający zainstalowany serwer aplikacji Apache Tomcat dołączony do IDE WebRatio. W podkatalogu *webapps* są umieszczane wygenerowane przez środowisko aplikacje.
2. *WebRatio* – katalog zawierający zainstalowane narzędzie WebRatio. Ponieważ opisywana wersja narzędzia bazuje na środowisku Eclipse, katalog ma strukturę charakterystyczną dla tego środowiska. Jego najważniejsze podkatalogi, to (Eclipse, 2011):
   - *configuration* – katalog zawierający informacje pozwalające na uruchomienie środowiska oraz informacje gromadzone w trakcie jego działania.
   - *drivers* – katalog zawierający sterowniki pozwalające na dostęp do źródeł danych w postaci baz danych lub plików o o dpowiedniej strukturze. Pakiet instalacyjny dostarcza tylko niektóre z nich, pozostałe należy pobrać samodzielnie.
   - *features* – katalog zawierający składniki środowiska, informacje owtyczkach realizujących wspólne funkcje.
   - *p2* – katalog wykorzystywany przez frameworkEquinox/P2, który odpowiada za aktualizację środowiska.
   - *plugins* – katalog zawierający wtyczki, czyli programy w języku Java rozszerzające funkcjonalność środowiska Eclipse.
3. *Workspace* – katalog zawierający podkatalogi z projektami tworzonymi w IDE WebRatio.

Katalog zawierający projekt aplikacji tworzonej w środowisku IDE WebRatioma następującą strukturę (Żyła i Kęsik, 2010b):

1. *Temp~* – katalog zawierający m.in. logi oraz informacje dotyczące synchronizacji modelu danych ze źródłami danych.

2. *DBScripts* – katalog zawierający skrypty SQL wygenerowane w trakcie mapowania elementów modelu danych do źródeł danych.
3. *WebContent* – katalog zawierający m.in. tekst i pliki multimedialne prezentowane w ramach stron aplikacji oraz pliki szablonów, np. wiadomości e-mail.
4. *Model.wr* – główny plik z projektem aplikacji.

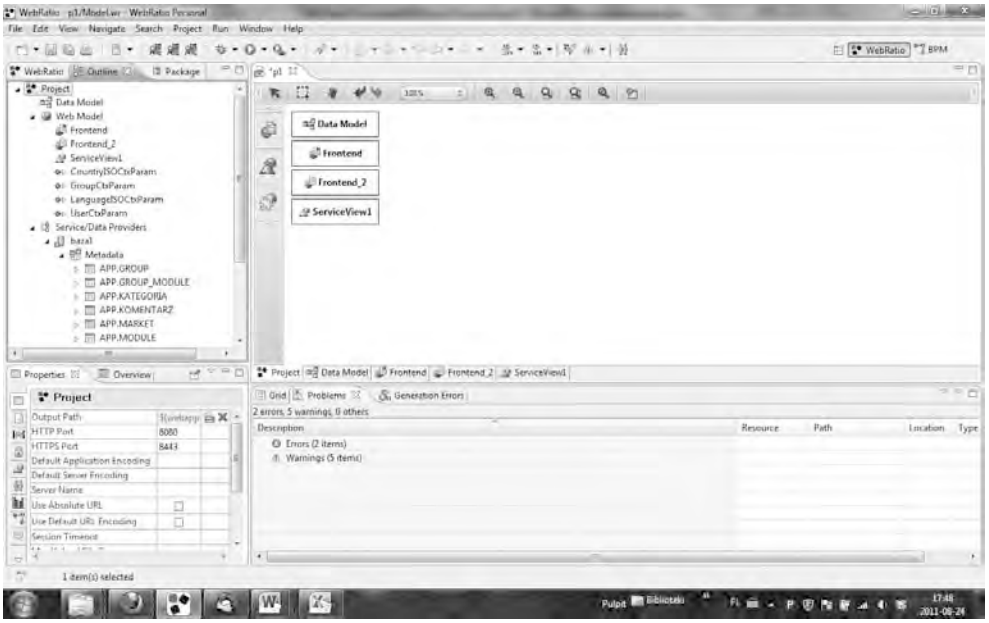
Katalog z wygenerowaną na podstawie projektu i umieszczoną na serwerze aplikacją ma strukturę typową dla technologii JSP(Oracle, 2010).

5.1.3 WPROWADZENIE DO NARZĘDZIA WEBRATIO

Graficzny interfejs użytkownika narzędzia WebRatioPersonal w wersji 6.1(Rys. 5.2) standardowo składa się z następujących sekcji(Żyła i Kęsik, 2010b), (WebRatio, 2009):

1. Eksplorator WebRatio (*ang. WebRatio Explorer*) – pokazuje poniższe elementy:
   - aktualnie otwarte projekty aplikacji internetowych,
   - aktualnie otwarte projekty stylów,
   - predefiniowane w WebRatio, nie podlegające modyfikacji, projekty stylów,
   - aktualnie otwarte projekty komponentów,
   - predefiniowane w WebRatio, nie podlegające modyfikacji, projekty komponentów.
2. Zarys (*ang. WebRatioOutline*) – przedstawia strukturę aktualnie aktywnego elementu projektu.
3. Podgląd projektu (*ang. Overview*) – widok obszaru projektu z dalekiej perspektywy.
4. Widok właściwości (*ang. PropertiesView*) – prezentuje i umożliwia modyfikację właściwości aktualnie aktywnego elementu projektu.
5. Widok siatki (*ang. GridView*) – pozwala na rozmieszczenie komponentów prezentacji danych w ramach strony.
6. Lista problemów (*ang. ProblemsView*) – zawiera wykaz błędów i ostrzeżeń wykrytych w trakcie analizy poprawności stworzonego modelu.
7. Widok logów (*ang. Log View*) – pozwala na otwieranie plików z logami wygenerowanej aplikacji.

8. Pomoc komponentu (*ang. Unit Reference*) – zawiera kompletny opis właściwości komponentu modelu.
9. Obszar edytora (*ang. Editor Area*) – obszar, w którym budowany jest projekt, umożliwiający również wyświetlenie oraz edycję plików tekstowych wchodzących w skład aplikacji.
10. Pomoc (*ang. Help*) – sekcja pozwalająca na korzystanie z dokumentacji dostarczonej wraz ze środowiskiem.



Rys. 5.2 Interfejs środowiska programistycznego WebRatio Personal 6.1
Źródło: opracowanie własne

Interfejs ten nie różni się znacząco od innych aplikacji programowania wizualnego. Za niedociągnięcie można uznać brak obsługi techniki typu „przeciągnij i upuść" w trakcie edycji modeli, co może powodować pewne zdezorientowanie przy próbach jej zastosowania.

Narzędzie WebRatio6.1 pozwala na tworzenie kilku typów projektów (WebRatio, 2010)

1. Aplikacji internetowej – projekt zawiera modele projektowanej aplikacji wykonane w notacji WebML. Model prezentacji aplikacji jest określany poprzez wybór wcześniej wykonanych projektów stylów.
2. Stylów – projekt określający warstwę prezentacji aplikacji internetowej. Zawiera kolekcję szablonów, na podstawie których są generowane strony HTML oraz grupuje zasoby takie, jak: arkusze CSS, pliki multimedialne, applety Java, kontrolki ActiveX oraz instrukcje języków skryptowych wykonywane po stronie klienta.
3. Komponentów – projekt zawierający zestaw dodatkowych, zdefiniowanych przez projektanta komponentów modelu hipertekstowego wraz z zasobami niezbędnymi w trybie generowania aplikacji.
4. BPM (*ang. Business Process Management*) – projekt aplikacji wspomagającej proces biznesowy opisany w notacji BPMN (*ang. Business ProcessModellingNotation*), służącej do opisywania procesów biznesowych. BPMN staje siępraktycznym standardem, zgodnym z koncepcją architektury SOA (model BPM i notacja BPMN nie znajdują się w zakresie materiału obejmowanym przez ten podręcznik). Model BPM jest nadrzędny w stosunku do modelu aplikacji internetowej, tzn. że narzędzie WebRatio umożliwia automatyczną transformację modelu BPM do modelu WebML.
5. BAM (*ang. Business Activity Monitor*)–projekt zawierający model struktury monitorującej stopień wykonania poszczególnych procesów modelu BPM przez aplikację wygenerowaną z wykorzystaniem modelu WebML odpowiadającego modelowi BPM.

5.1.4 Środowisko pracy wygenerowanych aplikacji

Jak już wcześniej wspomniano, w skład środowiska programistycznego wchodzi serwer aplikacji, na którym automatycznie są umieszczane aplikacje wygenerowane na podstawie opracowanych modeli. W przypadku IDE WebRatio6.1 jest to Apache Tomcat6.0.32, który jest implementacją technologii Java Servlet i JavaServerPages rozpowszechnianą na licencji Apache Software License i składa się z następujących komponentów (The Apache Software foundation, 2010):

1. Catalina – kontener servletów implementujący specyfikacje firmy Sun: Servleti JSP.
2. Coyote – konektor wspierający protokół HTTP. Nasłuchuje określony port TCP, przekazuje żądania do serwera Tomcat w celu ich przetworzenia oraz wysyła odpowiedzi do klientów.
3. Jasper – silnik JSP odpowiedzialny za analizę plików JSP i ich kompilację do servletów, które mogą zostać przetworzone przez Catalinę. W trakcie działania automatycznie wykrywa zmiany w plikach JSP i dokonuje ich rekompilacji.

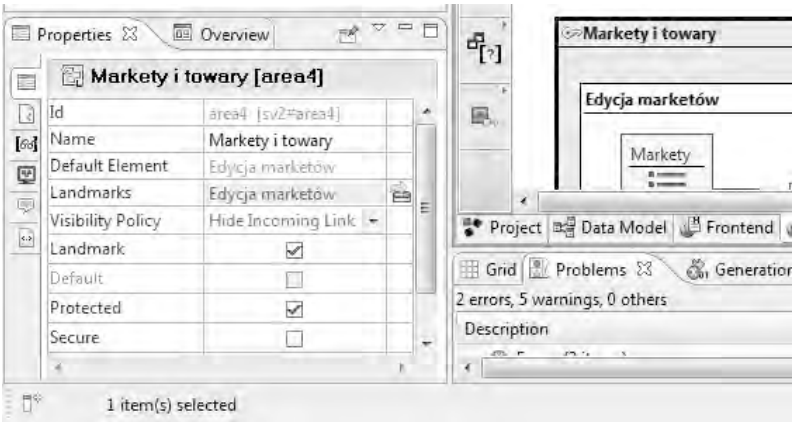
Wygenerowane aplikacje, poprzez ORM Hibernate, mogą współpracować z wieloma bazami danych, w tym z bazą danych Apache Derby dostarczoną w pakiecie instalacyjnym IDE WebRatio6.1. Jest ona w całości napisana w Javie, dostępna na licencji Apache License, Version 2.0. Jej kluczowe zalety, to (The Apache DB Project, 2011):

- niewielki rozmiar – silnik bazodanowy i sterownik JDBC zajmują około 2 MB,
- bazuje na standardach Java, JDBC i SQL,
- jest dostarczana z wbudowanym sterownikiem JDBC,
- dwa tryby działania embedded i server.

5.1.5 MODEL PERSONALIZACJI W WEBRATIO

Budowa aplikacji internetowej wiąże się z jej personalizacją i kontrolą dostępu użytkowników. WebML, jak przedstawiono w rozdziale 5.1, proponuje pewien standard realizacji takiej personalizacji. Jego skutkiem jest budowa aplikacji o różnych poziomach dostępu.

W narzędziu WebRatio zdecydowano się utrzymać proponowany przez WebML model personalizacji, narzucając projektantowi jego wykorzystanie. Użycie innego podejścia nie jest niemożliwe, wymaga jednak wielu zabiegów związanych z obchodzeniem narzuconych rozwiązań. Przedstawione dalej przykłady modelu hipertekstu opisują realizację zarządzania użytkownikami. Należy więc wpierw wyjaśnić model kontroli dostępu w WebRatio, z którego będą korzystać.

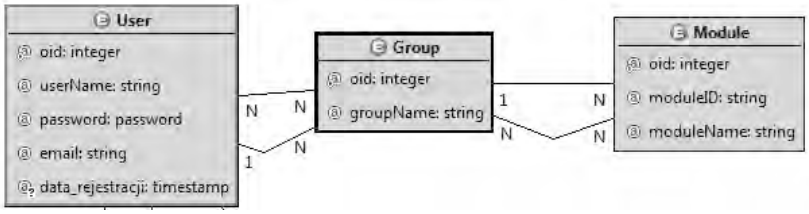


Rys. 5.3 Oznaczenie obszarów chronionych w WebRatio
Źródło: opracowanie własne

Struktura personalizacji i ograniczania dostępu do aplikacji, wykorzystywana w narzędziu WebRatio (które zostanie użyte do modelowania opisywanej funkcjonalności) wymaga podziału aplikacji na obszary, które można oznaczyć jako chronione. Obszar taki wydziela logicznie pewną część modelu hipertekstu aplikacji, którą może być zarówno zbiór stron, samodzielna strona, jak i pojedynczy komponent prezentacji danych lub operacji. Ustawienie parametru *protected*np. dla obszaru (Rys. 5.3) powoduje „wyjęcie go" z ogólnodostępnej części aplikacji. Użytkownicy nie mający uprawnień nie będą mogli uzyskać dostępu do zawartości tego obszaru – nawet linki nie będą widoczne w ich menu.

Rys. 5.4 przedstawia atrybuty i relacje między encjami modelu danych realizującego koncepcję dostępu do chronionych obszarów. Encja *Module* przechowuje informacje o wszystkich obszarach chronionych wyróżnionych w modelu hipertekstu. Dostęp do tych obszarów jest przydzielany grupom użytkowników, których dane są zapisane w encji *Group*. Jest ona połączona dwiema relacjami z encją *Module*. Pierwsza z nich (1:N) odpowiada za przypisanie grupie modułu głównego – obszaru, a raczej strony, do której jest przekierowywany użytkownik z tej grupy w chwili zalogowania. Druga relacja (N:N) odpowiada za przypisanie grupie zbioru innych modułów, do których grupa ma również dostęp. Encja *User*(przechowująca dane użytkowników systemu) jest połączona z encją *Group* również dwoma relacjami.

Relacja 1:N odpowiada za przypisanie użytkownikowi jego grupy głównej (tej, z którą startuje po zalogowaniu). Relacja N:N przypisuje użytkownikowi grupy dodatkowe, z których również może korzystać. Użytkownik nie ma jednak dostępu do modułów tych grup, może natomiast zmienić swoją grupę główną na jedną z nich – zmieniając tym samym swoją rolę w aplikacji.



Rys. 5.4 Model danych zarządzania użytkownikami
Źródło: opracowanie własne

Każdej strefie chronionej projektu odpowiada instancjaencji*Module*. WebRatio nie wykonuje jednak automatycznej synchronizacji encji *Module* z bieżącym stanem modelu. Strefy chronione można umieścić tam półautomatycznie za pomocą strony administracyjnej, aktywowanej z projektu. Stronę taką należy aktywować w oknie właściwości elementu *Project* (Rys. 5.5).

Uruchomiona strona administracyjna analizuje różnice między instancjami encji *Module*, a s tanem obecnym modelu hipertekstu i oferuje ewentualne korekty przez dodanie lub usunięcie odpowiednich instancji (Rys. 5.6).

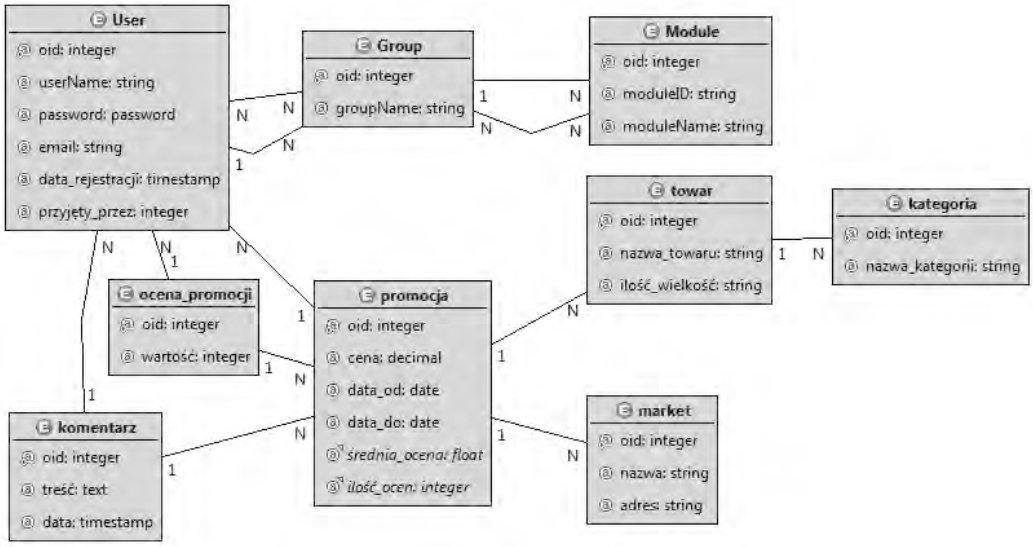
Rys. 5.5 Aktywowanie strony administracyjnej w WebRatio
Źródło: opracowanie własne

Rys. 5.6 Strona administracyjna aplikacji wygenerowanej przez WebRatio
Źródło: opracowanie własne

Aby użytkownik mógł uzyskać dostęp do konkretnego obszaru chronionego, musi należeć do grupy, która ma przypisany ten obszar jako moduł główny lub jeden z modułów dodatkowych. Realizacja funkcjonalności zarządzania użytkownikami powinna więc zacząć się od umożliwienia tworzenia grup z odpowiednio przypisanymi modułami.

5.2 ROZBUDOWA MODELU HIPERTEKSTOWEGO

Do przedstawienia procesu ewolucji modelu hipertekstu zostanie wykorzystana funkcjonalność zarządzania użytkownikami i ich grupami, związana z modelem dostępu użytkowników o różnych uprawnieniach do poszczególnych obszarów aplikacji. Niektóre aspekty modelowanej funkcjonalności zakładają konieczność wykorzystania danych zgromadzonych w innych encjach. Poniżej przedstawiono model danych, z którego będą korzystać poszczególne modele hipertekstu (Rys. 5.7).

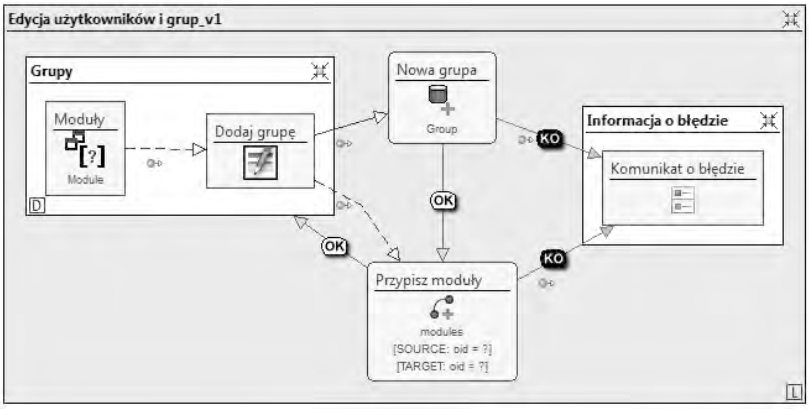


Rys. 5.7 Model danych aplikacji śledzenia promocji w marketach
Źródło: opracowanie własne

Przedstawiony na rysunku 5.7 model danych został opracowany z myślą o aplikacji internetowej do zbierania informacji o promocjach dostępnych w marketach. Promocja dotyczy danego towaru (z danej kategorii i w danym markecie). Użytkownik aplikacji może dodać informację o promocji, ocenić daną promocję lub wpisać komentarz o niej. Administratorzy (oprócz zarządzania użytkownikami) zapisują markety i towary oraz tworzą ich kategorie.

5.2.1 ZAPISYWANIE DANYCH

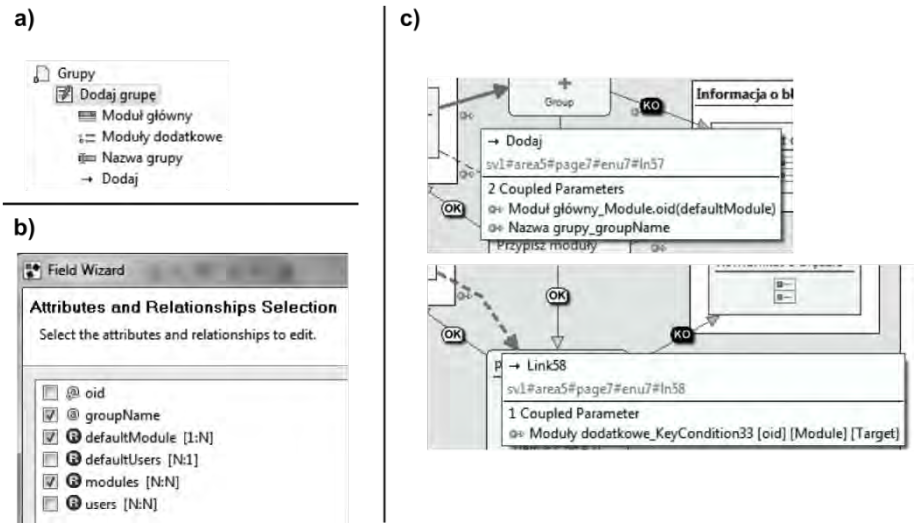
Rys. 5.8przedstawia prostą strukturę hipertekstową umożliwiającą dodawanie grup użytkowników. Zawarta jest ona na stronie o nazwie *Grupy,* będącej częścią obszaru(strefy) o nazwie *Edycja użytkowników i grup*. Jedynym elementem renderowanym na s tronie jest formularz reprezentowany przez Komponent Wprowadzania *Dodaj grupę.*



Rys. 5.8 Model hipertekstu dodawania grup użytkowników

Źródło: opracowanie własne

Z modelu danych przedstawionego na Rys. 5.4wynika, że dane wymagane przy tworzeniu nowej grupy, to: jej nazwa, identyfikator modułu głównego, lista identyfikatorów modułów dodatkowych. Komponent Wprowadzania wymaga więc zdefiniowania trzech pól: tekstowego, listy rozwijanej i pola wielokrotnego wyboru (Rys. 5.9a). IDE WebRatio udostępnia odpowiedni kreator (Rys. 5.9b) automatyzujący ten proces.

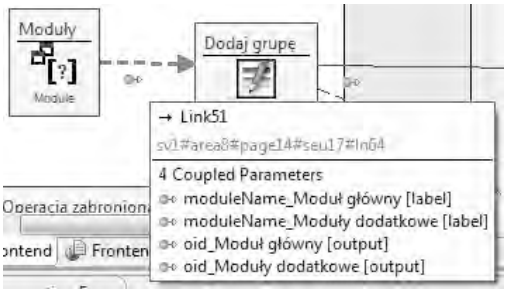


Rys. 5.9 Atrybuty i linki komponentu Dodaj grupę

Źródło: opracowanie własne

Zarówno lista rozwijana, jak i pole wielokrotnego wyboru wymaga informacji o nazwach i identyfikatorach modułów zapisanych w encji *Module*. Dostarczenie tych danych realizuje Komponent Selekcji *Moduły* za pomocą linku transportowego (Rys. 5.10).

Po wprowadzeniu w pola odpowiednich danych, link *Dodaj* aktywuje Komponent Tworzenia Instancji *Nowa grupa.* Realizuje on utworzenie instancji encji *Group* modelu danych.Za pomocą tego komponentu nie można jednak zrealizować zapisania listy modułów dodatkowych- odpowiadających relacji N:N między encją *Group* a *Module.* Instancje takich relacji są tworzone przez Komponent Łączenia.



Rys. 5.10 Modelowanie funkcji napełnienia danymi list wyboru w formularzu
Źródło: opracowanie własne

W opisywanym przykładzie taki komponent (o nazwie *Przypisz moduły*) jest aktywowany linkiem OK, po prawidłowym przetworzeniu komponentu *Nowa grupa.*

Oba komponenty wymagają otrzymania odpowiednich parametrów na wejściu:

- Komponent *Nowa grupa* potrzebuje nazwę grupy oraz identyfikator modułu głównego. Są one dostarczone linkiem *Dodaj* aktywującym operację.
- Komponent *Przypisz moduły* potrzebuje identyfikator utworzonej instancji grupy oraz listę identyfikatorów modułów dodatkowych. Oid (*ang. Object Identifier*) instancji grupy jest przekazywany wraz z linkiem OK. Lista identyfikatorów modułów dodatkowych jest dostarczana z Komponentu Wprowadzania linkiem transportowym.

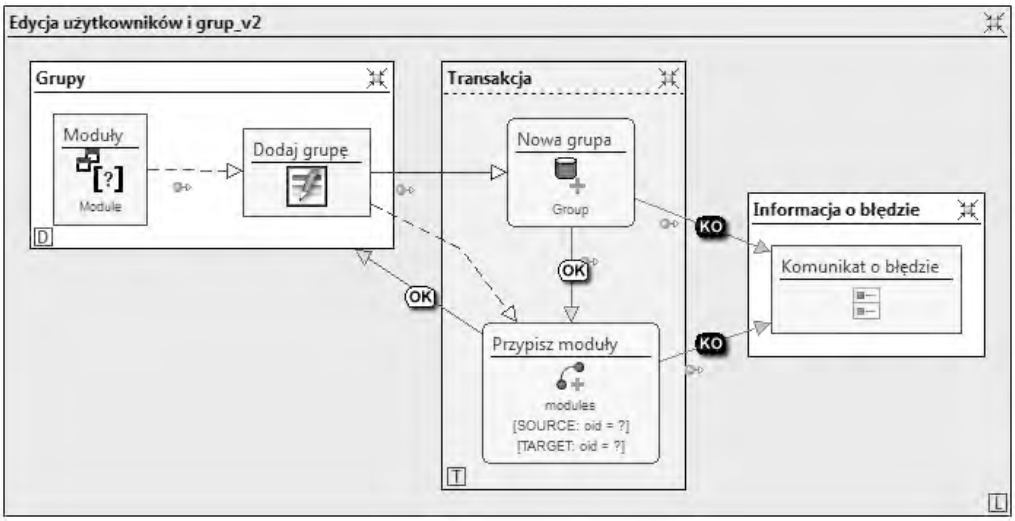
Rysunek5.9c przedstawia parametry przekazywane opisanymi wyżej linkami. Wyszczególnione są połączenia między parametrami wychodzącymi z komponentu źródłowego, a parametrami pobieranymi przez komponent docelowy.

Każdy komponent operacji może zakończyć swoje działanie powodzeniem lub porażką. W przypadku błędnego zakończenia operacji realizowanych przez komponenty *Nowa grupa*czy *Przypisz moduły*,należy powiadomić użytkownika poprzez wyświetlenie odpowiedniego komunikatu. W omawianym przykładzie komunikat ten jest wyświetlany przez Komponent Wiadomości *Komunikat o błędzie*, umieszczony na osobnej stronie www. Przejście do tej strony i wyświetlenie komunikatu jest realizowane w chwili aktywowania linku KO przez jeden z komponentów operacji.

Cechą tego prostego przykładu jest wykorzystanie linków OK komponentów operacji do zamodelowania sekwencji operacji wykonywanych w określonej kolejności.

5.2.2 ZAPISYWANIE DANYCH W TRANSAKCJI

Podstawową wadą modelu przedstawionego na rysunku 5.8 jest możliwość wystąpienia niepełnego zapisu danych w sytuacji błędnego wykonania się komponentu *Przypisz moduły.*



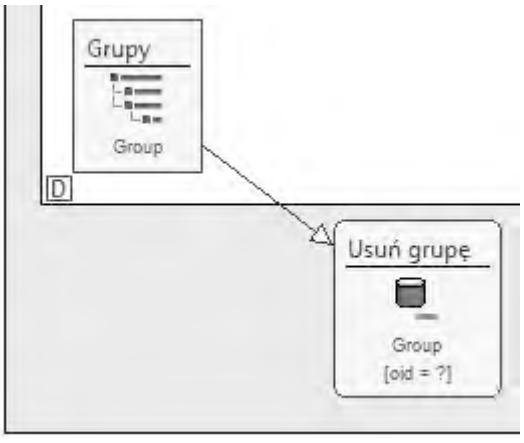
Rys. 5.11 Model hipertekstu dodawania grup użytkowników w transakcji
Źródło: opracowanie własne

Typowym rozwiązaniem jest wykonanie wszystkich operacji związanych z zapisem danych w transakcji, co umożliwia anulowanie wszystkich zmian w sytuacji wystąpienia błędu na dowolnym etapie. Rysunek 5.11 przedstawia modyfikację poprzedniego modelu uwzględniającą transakcję.

Jedyna konieczna zmiana, to utworzenie obszaru grupującego operacjeo nazwie *Transakcja*, umieszczenie w nim komponentów *Nowa grupa* i *Przypisz moduły*oraz ustawienie parametru *Transaction* tego obszaru na TRUE.

5.2.3 WARUNKOWE KASOWANIE DANYCH

Poprzednie modele umożliwiają jedynie dodawanie grup. Nie ma możliwości zobaczenia indeksu istniejących grup, ani skasowania którejkolwiek z nich. Dodanie tej funkcjonalności wymaga dodania dwóch komponentów do ostatniego modelu (rys. 5.12). Są to Komponent Indeksu Hierarchicznego oraz Komponent Usuwania Instancji.



Rys. 5.12 Dodatkowe komponenty realizujące prezentację listy i usuwanie grup
Źródło: opracowanie własne

Komponent Indeksu Hierarchicznego *Grupy*prezentuje listę istniejących grup. Dzięki temu, że komponent może prezentować dane powiązane hierarchicznie, jest

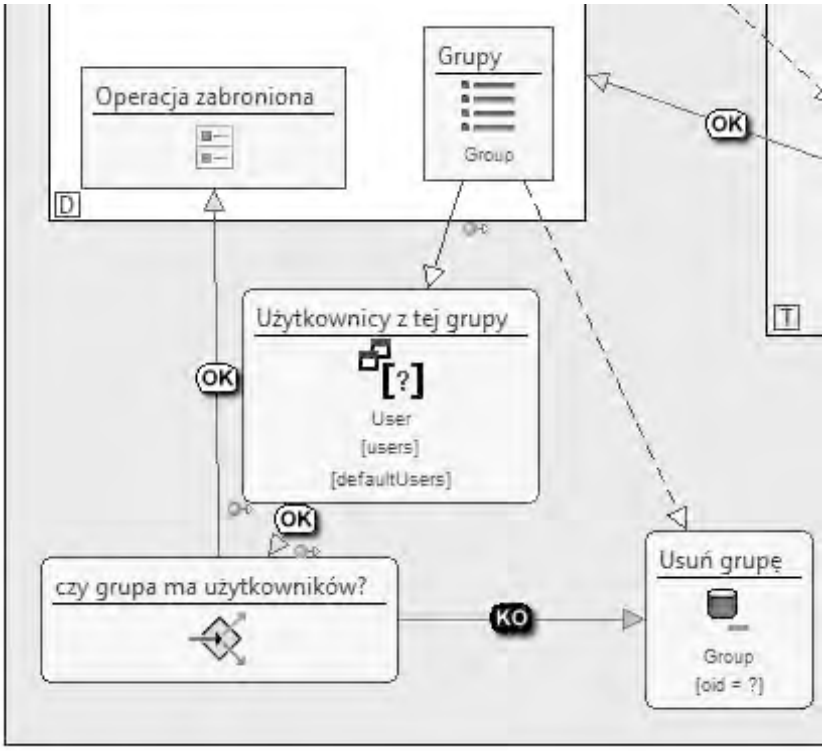
możliwe pokazanie nazwy domyślnego modułu chronionego dla każdej z wyświetlonych grup.

Komponent *Usuń grupę* wymaga podania w chwili aktywacji identyfikatora instancji encji do skasowania. Komponent *Grupy*może być źródłem tej wartości, zarazem dostarczając wygodnego interfejsu dla użytkownika – umożliwiając wybór grupy do skasowaniaz listy. Link pomiędzy tymi komponentami jest aktywatorem akcji i zarazem przenosi identyfikator(Nie ma wymogujawnego definiowania parametru identyfikatora przenoszonego przez link, dzięki wykorzystaniu domyślnych parametrów wyjściowych i wejściowych dla każdego komponentu)

Struktura *komponent indeksu -> komponent operacji* jest standardowym układem wykorzystywanym w modelowaniu hipertekstu. Taka konstrukcja zapewnia wygodę użytkownika i zapobiega możliwości skasowania niewłaściwej instancji przez uniemożliwienie użytkownikowi ingerencji w wartość identyfikatora wybranej instancji. Z drugiej strony jej główną wadą jest domyślny brak weryfikacji, czy dana instancja może być skasowana.

W kolejnym przykładzie zakładamy blokadę skasowania instancji encji *Group*, jeżeli istnieje choć jeden użytkownik do niej przypisany. Wymaga to wzbogacenia modyfikacji z rysunku 5.12 o kolejne komponenty (rys. 5.13).

Pierwszym etapem realizacji tej funkcjonalności jest zdefiniowanie warunków odrzucenia żądania skasowania wybranej grupy. Nastąpi ono w przypadku istnienia takiego użytkownika, który ma przypisaną ową grupę albo jako grupę główną, albo jako jedną z grup dodatkowych. Wyszukanie takich użytkowników jest realizowane przez Komponent Selekcji *Użytkownicy z tej grupy*, który pobiera instancje encji *User.* To, że wybrani zostaną tylko odpowiedni użytkownicy (a nie wszystkie instancje tej encji) zapewnia narzucony na ni ego warunek selekcji. Warunek ten składa się z dwóch członów złączonych warunkiem logicznym OR. Pierwszy człon *[users]*ogranicza zbiór do użytkowników posiadających kasowaną grupę wśród swoich grup dodatkowych, drugi człon *[defaultUsers]* ogranicza zbiór do użytkowników mających kasowaną grupę wybraną jako grupę główną. Komponent zwróci zbiór pusty w przypadku braku przeciwskazań do skasowania grupy albo listę identyfikatorów użytkowników powiązanych z tą grupą w przypadku przeciwnym.



Rys. 5.13 Modyfikacja realizująca warunkowe usuwanie grup
Źródło: opracowanie własne

Za uzależnienie operacji kasowania grupy od wyniku uzyskanego przez komponent *Użytkownicy z tej grupy*jest odpowiedzialny Komponent Decyzyjny *czy grupa ma użytkowników?*. Aktywuje on link KO nie tyle w przypadku wystąpienia błędu, co w przypadku braku parametru kontrolnego na wejściu. Parametrem kontrolnym jest wynik uzyskany przez komponent *Użytkownicy z tej grupy,* co powoduje (paradoksalnie) aktywację linku KO w przypadku braku przeciwskazań do skasowania grupy. Dalsza część modelu jest już analogiczna do wersji poprzedniej. Można tylko wspomnieć, że Komponent Wiadomości *Operacja zabronion*anie zawiera jakichkolwiek danych (czyli nie będzie renderowany), o ile nie zostanie aktywowany linkiem OK Komponentu Decyzyjnego.

5.2.4 DODAJ, USUŃ, MODYFIKUJ

Poprzednie modele nie przewidują możliwości zmiany danych już utworzonej grupy. Model hipertekstu uwzględniający tę możliwość został przedstawiony na rysunku 5.14. Bazuje on na szablonie udostępnionym w materiałach Wiki (WebRatio, 2009).

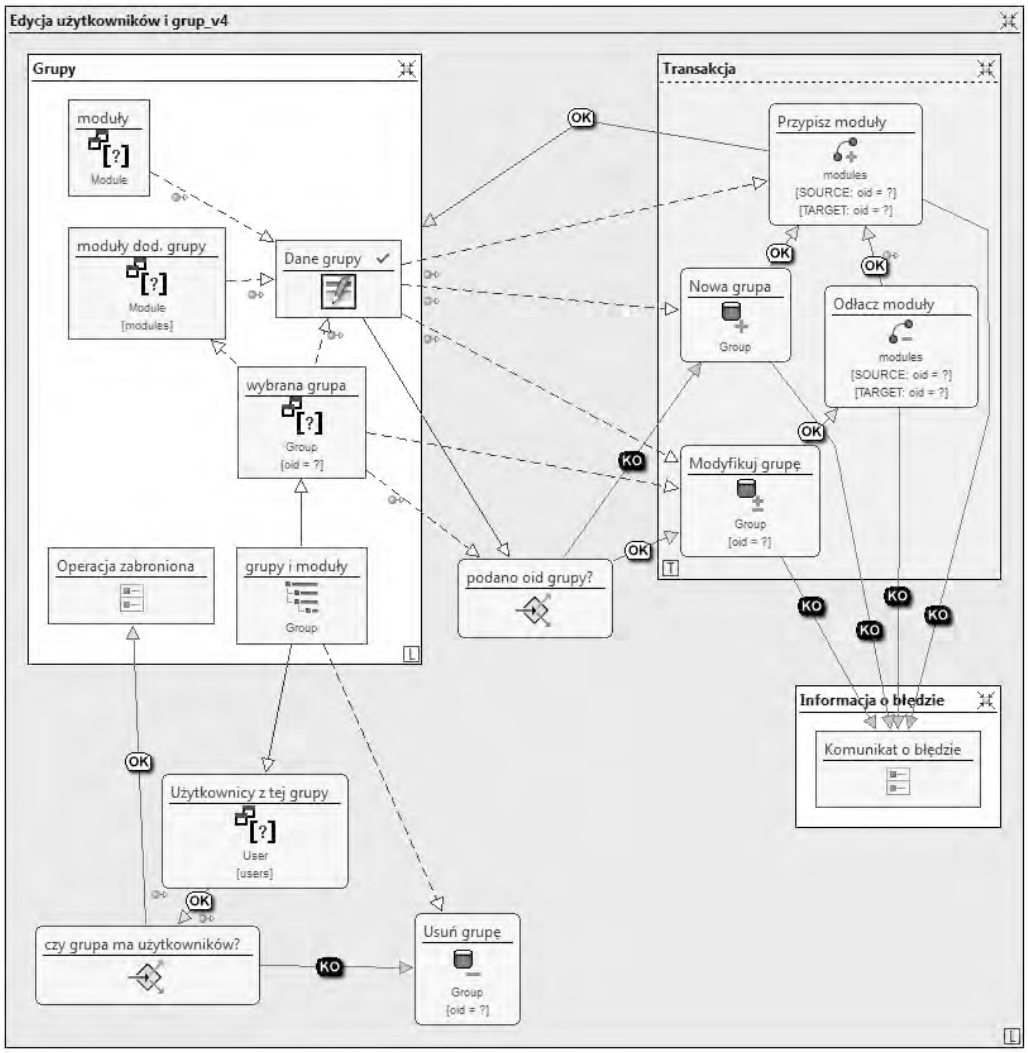
Zmiany w stosunku do poprzedniego modelu są związane z dwoma aspektami:

- koniecznością uzależnienia operacji od decyzji, czy grupa jest tworzona, czy modyfikowana,
- koniecznością pobrania danych grupy do formularza.

Pierwszy aspekt jest realizowany analogicznie jak w przypadku modyfikacji przedstawionej na rysunku 5.13. Komponent decyzyjny *podano oid grupy?* aktywuje link OK lub KO w zależności od tego, czy został aktywowany Komponent Selekcji *wybrana grupa*. Aktywacja ta następuje w chwili wybrania przez użytkownika jednej z grup z listy prezentowanej przez Komponent Indeksu *grupy i moduły*.

Ciekawym rozwiązaniem jest struktura umieszczona w strefie transakcji. Komponenty *nowa grupa* i *przypisz moduły* pochodzą z poprzedniej wersji modelu i są aktywowane linkiem KO w przypadku realizacji utworzenia nowej grupy.

W przypadku konieczności zapisania efektu modyfikacji danych grupy (link OK), kłopotliwym elementem jest pole wielokrotnego wyboru Komponentu Wprowadzania *Dane grupy*. Jest ono odpowiedzialne za wybór modułów dodatkowych dla grupy. Użytkownik może dowolnie zmienić ten wybór podczas edycji danych grupy. Rezultatem może być konieczność usunięcia części połączeń między grupą i modułami i/lub ustanowienia nowych. Aby uniknąć konieczności wyszukiwania, co należy dodać, a co usunąć, wykorzystana została sekwencja *Komponent Rozłączania* – *Komponent Łączenia* oraz właściwość Komponentu Rozłączania wykonująca rozłączenie konkretnej instancji encji źródłowej z wszystkimi powiązanymi z nią instancjami encji docelowej.



Rys. 5.14 Model realizujący pełną edycję grup

Źródło: (WebRatio, 2009)

Komponent Modyfikacji Instancji *Modyfikuj Grupę* zapisuje dane podstawowe instancji encji *Group* mającej identyfikator zgodny z dostarczonym przez komponent *Wybrana grupa*. Kolejnym komponentem w sekwencji jest Komponent Rozłączania *Odłącz moduły*. Aktywujący go link OK przenosi tylko domyślny parametr *oid* modyfikowanej grupy. Komponent *Odłącz moduły*odłącza wszystkie moduły od tej

grupy i przekazuje jej *oid* linkiem OK, aktywując następny w sekwencji Komponent Łączenia *Przypisz moduły*. W chwili aktywacji tego komponentu nie ma już żadnego połączenia między modyfikowaną grupą, a jakimkolwiek modułem. Komponent *Przypisz moduły*może więc wykonać połączenie modyfikowanej grupy z modułami, których identyfikatory otrzymał linkiem transportowym z komponentu *Dane grupy*.

Aspekt drugi, czyli pobranie danych edytowanej encji do formularza jest realizowany przez sekwencję Komponentów Selekcji. Komponent *wybrana grupa* jest aktywowany wybraniem nazwy grupy z listy dostarczanej przez komponent *grupy i moduły*. Narzucony na niego warunek klucza wymusza pobranie danych wyłącznie instancji, której *oid* został dostarczony wraz z aktywacją. Komponent ten dostarcza większość potrzebnych danych – wartości pól *Nazwa grupy* i *Moduł główny* (*oid* wartości, która ma być wybrana na początek). Brakuje jednak informacji o wybranych modułach dla pola *Moduły dodatkowe*. Informacje te dostarcza Komponent Selekcji *moduły dod. grupy*. Wyszukuje on moduły zgodne z narzuconym warunkiem *[modules]* istnienia relacji między modułem a grupą, której *oid* jest dostarczony linkiem transportowym przez komponent *wybrana grupa.*

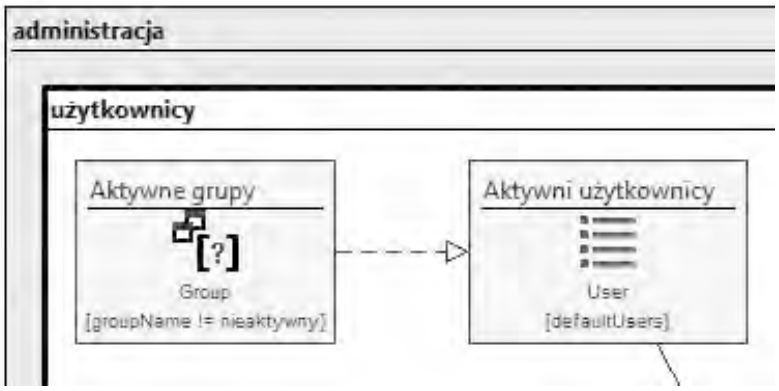
Sekwencje Komponentów Selekcji ograniczonych odpowiednimi warunkami trzech dostępnych typów: klucza, atrybutu i relacji, pozwalają na realizację dowolnego wyszukiwania danych. Są one zamieniane w czasie generacji aplikacji na odpowiednie zapytania SQL.

5.2.5 WARUNKOWE, KASKADOWE USUWANIE UŻYTKOWNIKA

Funkcjonalność zarządzania użytkownikami jest zbudowana analogicznie do zarządzania grupami, przedstawionego w rozdziale 5.2.4. Usuwanie użytkownika jest jednak bardziej złożone, gdyż wiąże się z danymi należącymi do tego użytkownika. Fizyczne usuwanie danych o użytkowniku jest w przypadku wielu systemów niewskazane w związku z niebezpieczeństwem utraty synchronizacji powiązań z innymi danymi związanymi z tym użytkownikiem. Stosuje się wtedy technikę ustawiania odpowiedniego statusu,czyniącego użytkownika nieaktywnym (w przypadku WebRatio przypisanie go do odpowiedniej grupy głównej). Wiąże się z tym konieczność sprawdzania statusu/grupy użytkownika podczas wyświetlania list

użytkowników dla celów administracji nimi.Rysunek 5.15 przedstawia fragment modelu realizujący taką funkcjonalność. Komponent Indeksu wyświetla listę użytkowników, których oid grupy głównej występuje na liście dostarczonej przez Komponent Selekcji wybierający wszystkie grupy, oprócz mającej nazwę „nieaktywny".

W przedstawionym przykładzie zakłada się możliwość skasowania użytkownika, o ile nie posiada aktywnych promocji. Jeżeli użytkownika można skasować, kasowane są również jego oceny i komentarze. Rysunek 5.16 przedstawia omawiany model hipertekstu.



Rys. 5.15 Wyświetlanie listy aktywnych użytkowników
Źródło: opracowanie własne

Nowymi aspektami są:

- wyszukanie aktywnych promocji użytkownika,
- zrealizowanie kaskadowego usunięcia jego ocen i komentarzy.

Aktywna promocja użytkownika, to taka, która została wpisana/utworzona przez tego użytkownika i jej data zakończenia nie jest wcześniejsza niż bieżąca. Wyszukanie takiej promocji może być zrealizowane przez Komponent Selekcji ograniczony odpowiednimi warunkami. Na rysunku 5.16 komponent *aktywne promocje użytkownika*ma przypisane dwa warunki selekcji:

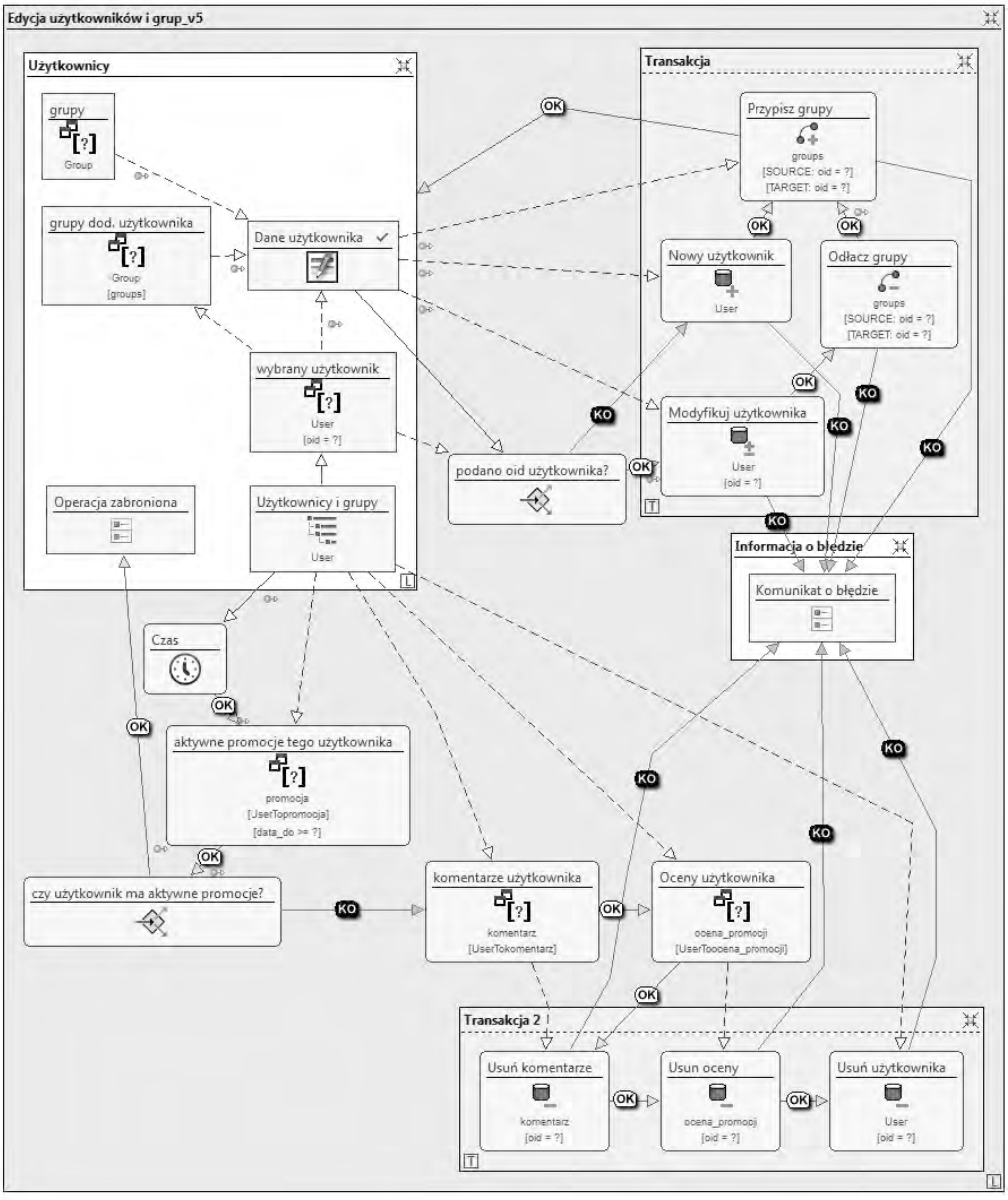
- warunek relacji – wybierający promocje będące w relacji z konkretnym użytkownikiem,
- warunek atrybutu – wybierający promocje o dacie zakończenia nie wcześniejszej, niż podana.

Komponent ten wymaga więc dostarczenia dwóch parametrów wejściowych: oid użytkownika oraz (bieżąca) data. Oid użytkownika wybranego do usunięcia dostarcza linkiem transportowym komponent *użytkownicy i grupy*, który udostępnia interfejs wyboru nazwy użytkownika. Link aktywujący nie jest jednak połączony bezpośrednio z Komponentem Selekcji, tylko aktywuje Komponent Czasu. Komponent *Czas* przekazuje aktywację linkiem OK do Komponentu Selekcji, dołączając parametr zawierający bieżącą (w chwili aktywacji) datę. Komponent Selekcji przekazuje aktywację oraz zbiór oid znalezionych promocji do Komponentu Decyzji *czy użytkownik ma aktywne promocje?.* W przypadku braku promocji Komponent Decyzji aktywuje link KO, uruchamiający operację skasowania danych użytkownika.

Operacja ta jest sekwencją komponentów. Można je podzielić na: grupę zbierającą dane (Komponenty Selekcji) oraz grupę usuwającą instancje (Komponenty Usuwania Instancji umieszczone w strefie transakcji).

Komponenty pierwszej grupy wykorzystują oid użytkownika (dostarczony linkiem transportowym z komponentu *użytkownicy i grupy*) do wyszukania komentarzy i ocen wystawionych przez tego użytkownika. Komponenty drugiej grupy wykonują w sekwencji kasowanie: komentarzy (których listę dostarcza komponent *komentarze użytkownika*), ocen (których listę dostarcza komponent *oceny użytkownika*) oraz użytkownika (którego oid dostarcza komponent *użytkownicy i grupy*).

Ustalona kolejność sekwencji kasowania nie daje możliwości skasowania danych użytkownika przed usunięciem danych odnoszących się do jego instancji. Ewentualne przerwanie wykonywania sekwencji nie spowodowałoby więc niespójności danych. Umieszczenie tej struktury w obszarze transakcji nie jest więc wymogiem koniecznym, a jedynie chęcią zapewnienia biznesowej atomizacji procesu.



Rys. 5.16 Model realizujący pełną edycję użytkowników
Źródło: opracowanie własne

W przykładzie tym ujawnia się jedna z wad związanych z wykorzystaniem technologii MDE, przedstawionych w rozdziale 2.4. Komponenty *Usuń komentarze,*

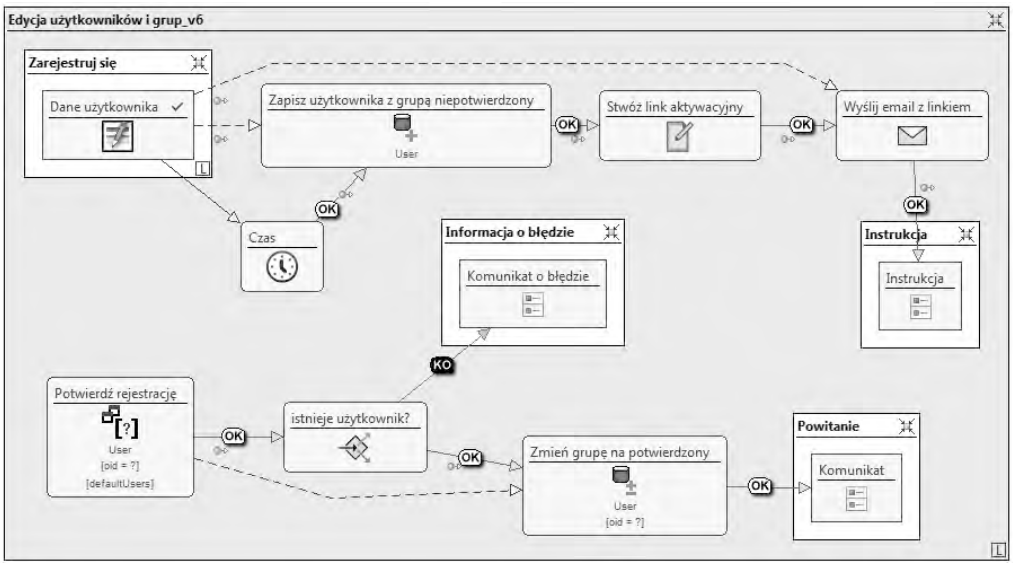
Usuń oceny i *Usuń użytkownika* nie muszą być wykonywane w sekwencji. Ich równoległe wykonywanie byłoby rozwiązaniem bardziej optymalnym w przypadku dużych ilości ocen i komentarzy wystawionych przez użytkownika. Wykorzystywane narzędzie WebRatio nie udostępnia jednak możliwości zamodelowania zrównoleglenia wykonywania operacji. Dostępną opcją byłoby zbudowanie niestandardowego komponentu, umożliwiającego kasowanie instancji z kilku encji jednocześnie. Wymaga to jednak od twórcy posiadania wiedzy technicznej z zakresu tworzenia oprogramowania na daną platformę. Nie jest to wiedza wymagana do projektowania modelu hipertekstu, nie należy więc jej oczekiwać od projektanta. Zamówienie wykonania takiego komponentu (lub w szczególnym przypadku wykonanie go samodzielnie) przekroczyłoby kosztownością zysk związany z optymalizacją rozwiązania pojedynczego przypadku.

5.2.6 REJESTRACJA UŻYTKOWNIKÓW W SERWISIE

Poprzednie przykłady przedstawiały zarządzanie grupami i użytkownikami przez administratora. Ręczne dodawanie użytkowników nie jest jednak zbyt optymalne w przypadku intensywnie wykorzystywanych aplikacji. Omówiony zostanie więc przypadek zarejestrowania się nowego użytkownika w serwisie.

Proces takiej rejestracji zakłada utworzenie użytkownika o statusie „niepotwierdzony” – nie mającego jeszcze dostępu do serwisu. Jednocześnie z utworzeniem, jest wysyłany do użytkownika e-mail zawierający link aktywujący konto. Link aktywacyjny musi być unikalny i trudny do odgadnięcia. Należy również zapobiegać zautomatyzowanej rejestracji kont, stosując test Captcha (*ang. Completely Automated Public Turing test to tell Computers and Humans Apart*).

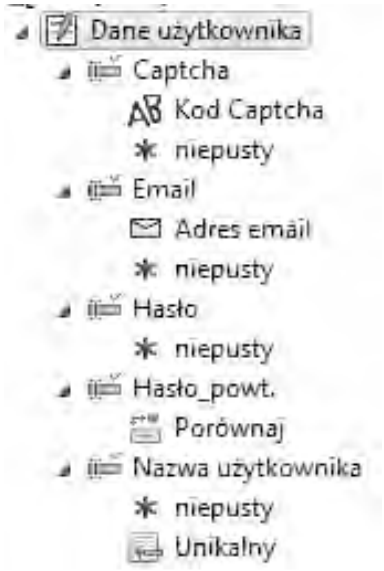
Model funkcjonalności rejestracji użytkowników, bazujący na rozwiązaniu przedstawionym w Wiki WebRatio(WebRatio, 2011)jest przedstawiony na rysunku 5.17.



Rys. 5.17 Model rejestracji użytkowników serwisu
Źródło: (WebRatio, 2011)

Jedynym komponentem renderowanym na s tronie jest Komponent Wprowadzania *Dane użytkownika*. Zawiera on pola formularza rejestracji nowego użytkownika, widoczne na rysunku 5.18. Tworzą one standardowy zestaw wymagany od rejestrujących się użytkowników: nazwa, hasło i powtórzenie hasła, adres e-mail i kod Captcha.

Na wszystkie pola zostały nałożone reguły walidacji wymuszające ich wypełnienie. Ponadto: pole *Email* otrzymało regułę sprawdzającą poprawność wprowadzonego adresu e-mail, pole *Hasło_powt* regułę sprawdzającą zgodność jego wartości z wartością pola *Hasło*, pole *Nazwa użytkownika* regułę testującą unikalność wprowadzonej nazwy użytkownika, pole *Captcha* regułę sprawdzającą „ludzkość" użytkownika. Chociaż złożoność reguł jest różnorodna (od prostego testu *Not Null* do generowania i analizy wyników zapytania SQL), sposób ich wprowadzania jest ujednolicony i wymaga co najwyżej podania kilku parametrów (rys. 5.19).



Rys. 5.18 Pola i reguły walidacji Komponentu Wprowadzania – Dane użytkownika
Źródło: opracowanie własne

a)

Porównaj [cmp2]

Id	Compare cmp2 [sv1#area11	
Name	Porównaj	
Predicate	Equal	
Ignore Case	☐	
Value Field	Hasło	
Value		
Error Message	Podane hasła nie są jednako	

b)

Unikalny [clc1]

Id	Collection clc1 [sv1#area11	
Name	Unikalny	
Predicate	Not in Collection	
Ignore Case	☐	
Values		
Entity	User	
Attribute	userName	
Error Message	Ta nazwa jest już używana	

Rys. 5.19 Parametry konfiguracji reguł walidacji: jednakowego hasła a) i unikalności nazwy użytkownika b)
Źródło: opracowanie własne

Komponent Wprowadzania modelu widocznego na rysunku 5.17 przesyła linkiem transportowym zwalidowane dane do Komponentu Tworzenia Instancji. Analogicznie do rozwiązania zastosowanego w modelu przedstawionym na rysunku 5.16,

aktywacja tego komponentu odbywa się poprzez Komponent Czasu, uzyskując dzięki temu marker czasu dokonania rejestracji przez użytkownika. Może on być wykorzystany do usunięcia użytkowników, którzy nie potwierdzili rejestracji w określonym czasie.

Kolejnym krokiem po zapisaniu danych użytkownika powinno być wygenerowanie linku aktywacyjnego. Podobnie, jak w przypadku równoległego wykonywania operacji usuwania z rozdziału 5.2.5, WebRatio nie udostępnia gotowego komponentu generującego taki link. Tym razem można jednak pokusić się o uzyskanie odpowiedniego efektu, wykorzystując Komponent Skryptu. Kod Groovy (listing 5.1), generujący potrzebny link, został pobrany z przykładu dostępnego na Wiki WebRatio(WebRatio, 2011).

Unikalny link wytworzony przez skrypt składa się z:

- adresu bezwzględnego aplikacji –*url*,
- adresu względnego operacji dokonującej potwierdzenia – *action*,
- ciągu parametrów identyfikujących użytkownika – *queryString*,
- ciągu zabezpieczającego link – *CRC*.

Ciąg CRC jest generowany na podstawie sesji użytkownika.

Listing 5.1. Kod Groovy generujący unikalny link aktywacji konta

```
#input String userOID

import com.webratio.rtx.RTXConstants
import com.webratio.struts.WRGlobals;
import com.webratio.struts.HttpRequestHelper
import com.webratio.rtx.core.BeanHelper

/*odczytujeżądanie HTTP*/
def request =
localContext.get(RTXConstants.HTTP_SERVLET_REQUEST_KEY)

/*odczytujesesję z żądania*/
def session = request.getSession().getServletContext();
defrequestHelper = (HttpRequestHelper)
session.getAttribute(WRGlobals.HTTP_REQUEST_HELPER_KEY);

/*odczytujeadres URL aplikacji*/
defurl = requestHelper.getAbsoluteURLContext(false, request)
```

```
/*definuje wywoływaną operację*/
def action = "/potwierdzenie.do"

/*tworzy ciąg parametrów identyfikujących użytkownika */
defqueryString = "kcond1.userOID=" + userOID +
"&rcond1.groupOID=1"

/* generuje kod CRC i tworzy adres URL linku aktywacyjnego*/
defconfirmationURL = url + action + "?" + queryString +
"&CRC=" + requestHelper.getCRC(queryString, request)

return confirmationURL
```

Źródło: (WebRatio, 2011)

Wytworzony link (ciąg znaków) jest przekazywany do Komponentu Poczty. Pozostałe potrzebne dane – nazwa i adres e-mail użytkownika, są dostarczane linkiem transportowym z Komponentu Wprowadzania. Komponent Poczty przesyła dane o wysłanym e-mailu wykorzystane przez Komponent Wiadomości wyświetlający informację z instrukcją dalszego postępowania dla użytkownika.

Druga część modelu odpowiada za obsługę żądania aktywacji, otrzymanego po kliknięciu przez użytkownika linku dostarczonego e-mailem. Konstrukcja nie posiada początkowej strony, zaczyna się od komponentu operacji. Parametr Komponentu Selekcji *Potwierdź rejestrację* o nazwie *Custom URL Name* zawiera nazwę wywołania operacji zapisaną w listingu 5.1. Dzięki niej żądanie wysłane po kliknięciu linku dociera bezpośrednio do tego komponentu. Zdefiniowane dla tego komponentu warunki selekcji wybierające użytkownika o określonym identyfikatorze i określonej grupie („niepotwierdzony”) pobierają parametry wejściowe z części *queryString* linku. Odszukany zostaje oid użytkownika w bazie, o ile jeszcze istnieje. Istniejący oid użytkownika zostaje przekazany do Komponentu Modyfikacji Instancji, który zmienia grupę główną użytkownika na „potwierdzony”.Zmiana ta aktywuje stronę wyświetlającą Komponent Wiadomości zawierający treść powitania.

5.2.7 AKCEPTOWANIE UŻYTKOWNIKA PRZEZ ADMINISTRATORA

Zajmując się nadal funkcjonalnością zarządzania użytkownikami należy rozważyćfunkcjonalność rejestrowania się użytkowników, ale od strony administratora.

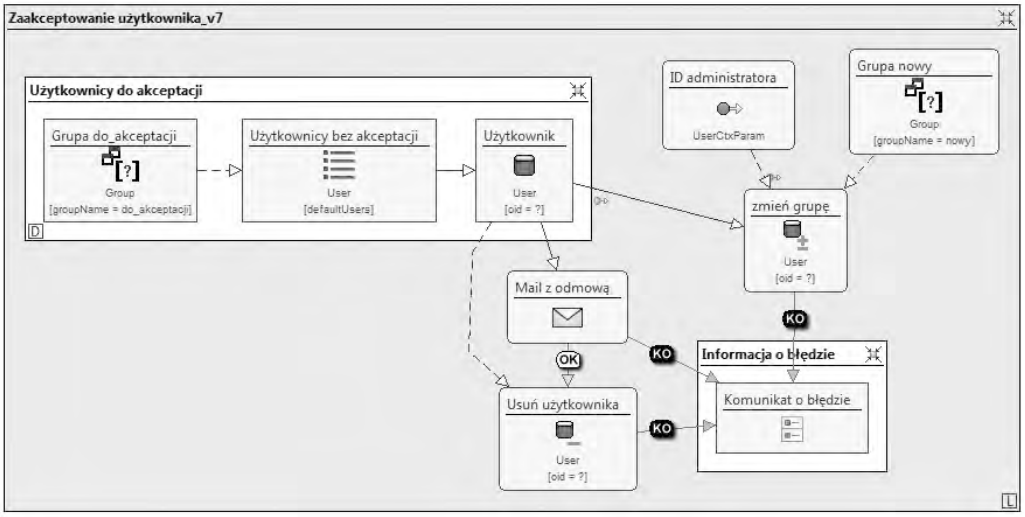
Nie każdy serwis zezwala na zupełnie automatyczne uzyskanie dostępu do niego przez nowo zarejestrowanego użytkownika. Omawiany serwis wymaga akceptacji jednego z administratorów do uzyskania pełnego dostępu.

Rysunek 5.20 przedstawia model funkcjonalności umożliwiającej administratorowi przyjęcie lub odrzucenie zarejestrowanego i potwierdzonego użytkownika. Komponent Indeksu *Użytkownicy bez akceptacji* ma nałożony warunek relacji, wymagający dostarczenia jako parametru wejściowego oid grupy. Ogranicza to listę tylko do użytkowników mających tę grupę określoną jako główną. Po tym parametrze można rozpoznać użytkowników jeszcze nie zaakceptowanych – mają oni ustawioną grupę główną o nazwie „do_akceptacji". Wpisywanie odpowiedniej wartości oid „na sztywno" mogłoby wywołać niepożądane skutki w przypadku zmian w bazie danych lub migracji na inną. Rozwiązaniem uniezależniającym od takich ewentualności jest wykorzystanie Komponentu Selekcji *Grupa do akceptacji*, wyszukującego grupę o nazwie „do_akceptacji" (odpowiedni warunek atrybutu nałożony na pole *groupName*). Taka nazwa jest zdefiniowana na p oziomie logiki biznesowej aplikacji i nie podlega zmianom wraz ze zmianami technologii implementujących aplikację.

Wybranie przez administratora jednego z użytkowników z listy powoduje załadowanie pełnych informacji o nim do Komponentu Danych *Użytkownik*. Komponent *Użytkownik* posiada dwa linki, umożliwiające administratorowi zaakceptowanie bądź odrzucenie użytkownika.

Wybranie linku *Akceptacja* powoduje aktywowanie Komponentu Modyfikacji Instancji *zmień grupę*i przekazanie muoid użytkownika do wprowadzenia zmian. Komponent *grupa nowy* realizuje analogiczny odczyt oid grupy *nowy*, jak komponent *Grupa do_akceptacji*.

Niewykorzystywanym dotychczas komponentem jest Komponent Pobrania Wartości*ID administratora*, odczytujący parametr globalny *UserCtxParam*, będący wartością oid użytkownika korzystającego z opisywanej funkcjonalności (zalogowanego).Oid ten jest zapisywany w polu *przyjęty_przez* instancji encji *User*.Za pomocą takiej konstrukcji aplikacja automatycznie rejestruje, kto (który administrator) dokonał akceptacji danego użytkownika. Umieszczenie Komponentu Pobrania Wartości poza stroną zabezpiecza jednocześnie przed ewentualną próbą podszycia się pod użytkownika o innym oid.

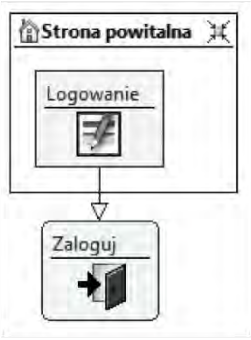


Rys. 5.20 Model hipertekstu funkcjonalności akceptowania użytkowników po rejestracji
Źródło: opracowanie własne

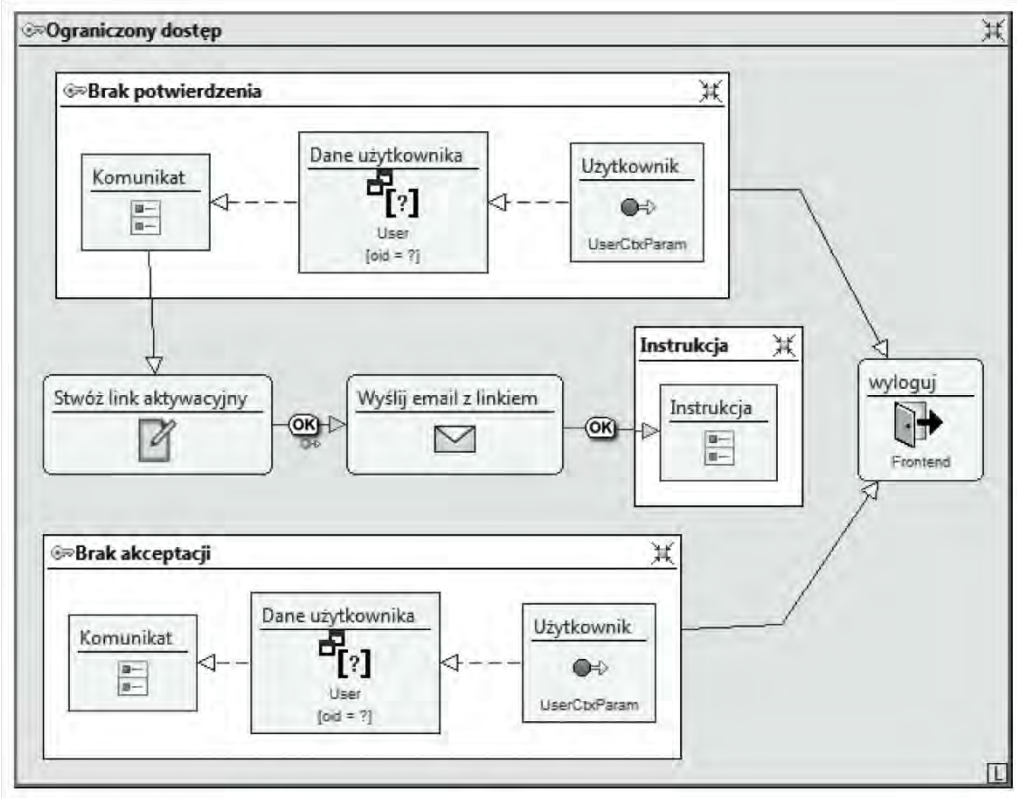
Wybranie przez administratora opcji odrzucenia użytkownika aktywuje sekwencję komponentów: Komponent Poczty wysyłający e-mail do użytkownika z powiadomieniem o tym fakcie, Komponent Usuwania Instancji kasujący dane o użytkowniku. Nie jest tu konieczne wprowadzenie warunków wykonania usunięcia, gdyż z założenia użytkownik niezaakceptowany nie mógł jeszcze wprowadzić żadnych danych do innych encji aplikacji, których synchronizację można by utracić po skasowaniu jego instancji.

Omawiana funkcjonalność zakłada istnienie użytkowników mających status *niepotwierdzony* lub *do_akceptacji*. Wspomniano również, że tacy użytkownicy mogą zalogować się w systemie, ale nie mają uprawnień, aby z niego korzystać. Próba zalogowania takiego użytkownika ma skutkować jedynie wyświetleniem odpowiedniego komunikatu (w przypadku użytkownika niepotwierdzonego możliwe ma być również powtórne wysłanie e-maila aktywacyjnego).

a)



b)



Rys. 5.21 Model hipertekstulogowania i ograniczonego dostępu użytkowników

Źródło: opracowanie własne

Rysunek 5.21a przedstawia realizację umożliwienia logowania ze strony powitalnej aplikacji. Komponent Wprowadzania pobiera od użytkownika login i hasło, a następnie przesyła je do Komponentu Logowania. Komponent Logowania

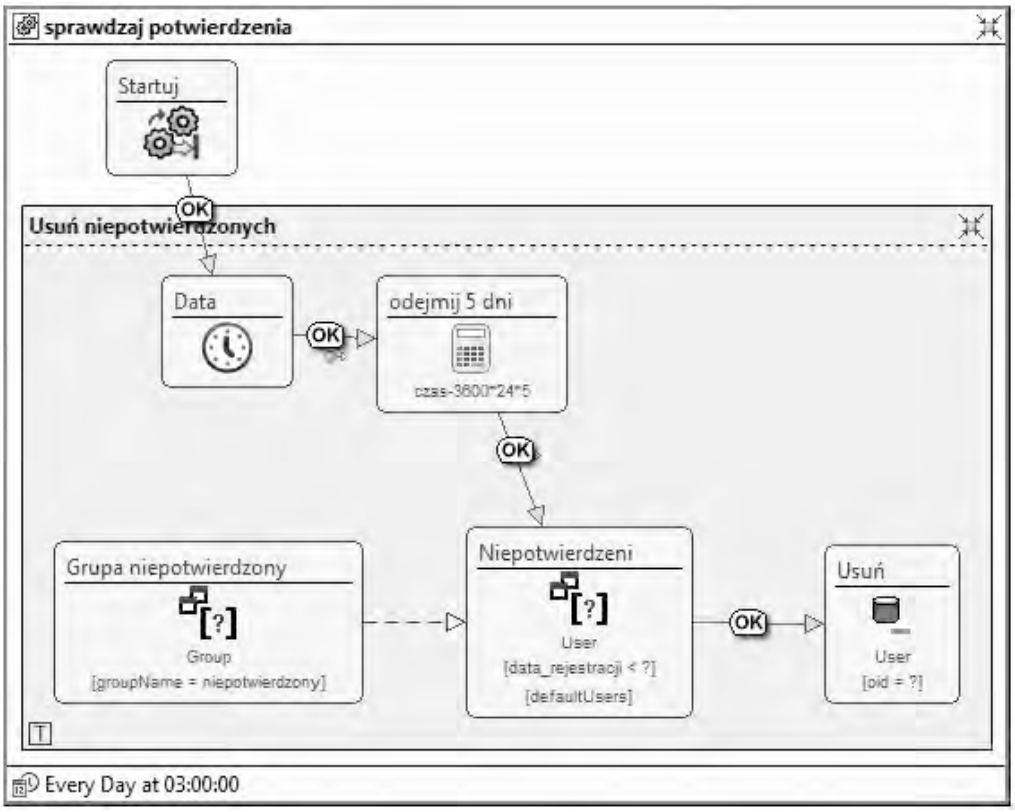
(w przypadku podania prawidłowych danych) przenosi użytkownika na domyślną stronę modułu głównego grupy głównej użytkownika. Generowane przez aplikację menu udostępnia linki do stron znajdujących się w modułach dostępnych dla grupy głównej użytkownika.

Rysunek 5.21b przedstawia strukturę modelu hipertekstu realizującą ograniczony dostęp dla nie w pełni zatwierdzonych użytkowników. Strona *Brak potwierdzenia* jest oznaczona jako chroniona i zapisana jako moduł główny grupy *niepotwierdzony*. Strona *Brak akceptacji* jest również oznaczona jako chroniona i zapisana jako moduł główny grupy *do_akceptacji*. Strefa *Ograniczony dostęp* jest przypisana obu grupom jako ich jedyny moduł dodatkowy. Strefa ta i odpowiednia ze stron będzie dostępna w menu użytkownika jednej z tych grup. Poza tym menu będzie pokazywało tylko linki do ogólnodostępnych części aplikacji. Inni użytkownicy aplikacji nie zobaczą tego działu w menu, gdyż nie będzie on przypisany do modułów dodatkowych ich grup.

Obie strony realizują wyświetlenie spersonalizowanego komunikatu, zawierającego informacje o użytkowniku. Strona *Brak potwierdzenia* udostępnia dodatkowo link aktywujący sekwencję opisaną w rozdziale 5.2.6. Sekwencja nie zawiera Komponentu Modyfikacji Instancji, gdyż nie ma potrzeby ponownego zapisu danych podstawowych użytkownika.

5.2.8 Okresowe usuwanie niepotwierdzonych użytkowników

Ostatnią omawianą funkcjonalnością modelu zarządzania użytkownikami jest automatyczne usuwanie użytkowników, którzy nie potwierdzili swojej rejestracji przez czas dłuższy niż 5 dni od chwili rejestracji. Operacja taka mogłaby być aktywowana przez administratora w dowolnej, dogodnej dla niego chwili. Obciążałoby to jednak administratora koniecznością pamiętania o regularnym jej aktywowaniu.Rozwiązaniem zwiększającym użyteczność aplikacji jest przeniesienie tej funkcjonalności do specjalnego Widoku Serwisów, pozwalającego na niezależne od użytkownika aktywowanie zadania.



Rys. 5.22 Model hipertekstuzadania usunięcia niepotwierdzonych użytkowników
Źródło: opracowanie własne

Przedstawiona na rysunku 5.22 realizacja zadania usunięcia niepotwierdzonych użytkowników jest typową strukturą definicji zadania realizowanego po stronie serwera, niezależnie od obsługi żądań HTTP. Każde zadanie jest definiowane wewnątrz własnej strefy, która może zawierać wiele grup operacji. Każda grupa operacji może zawierać komponenty operacji oraz hybrydowe. Komponenty prezentacji danych nie mogą być wykorzystane z powodu braku strony, na której mogłyby wyświetlać wynik działania. Widoczna na rysunku 5.22 grupa operacji rozpoczyna się od Komponentu Czasu, który dostarcza wartość bieżącego czasu i daty (*ang. timestamp*) do Komponentu Matematycznego*odejmij 5 dni*, odejmującego 5 dni (w sekundach) od dostarczonej wartości. Komponent ten aktywuje następnie Komponent Selekcji *Niepotwierdzeni*, dostarczając mu obliczoną wartość. Jest ona

wykorzystywana w warunku atrybutu wyszukującym wartości pola *data_rejestracji* mniejsze od dos tarczonej. Selekcję dopełnia warunek relacji wybierający użytkowników, których grupa główna ma nazwę „niepotwierdzony". Warunki są połączone logicznym AND, zwracając zbiór użytkowników spełniający oba z nich jednocześnie. Wyszukani użytkownicy są przekazywani wraz z aktywacją linkiem OK do Komponentu Usuwania Instancji encji*User*. Wykonanie się tego komponentu kończy sekwencję.

Komponentem specyficznym dla widoku serwisów jest Komponent Inicjacji Zadania. Zapoczątkowuje on wykonanie sekwencji operacji w chwili zadziałania jednego z wyzwalaczy zadania. W omawianym modelu komponent *Startuj*wysyła aktywację linkiem OK do Komponentu Czasu rozpoczynając wykonanie zadania. Przedstawiane zadanie ma zdefiniowany tylko jeden wyzwalacz czasowy, aktywujący komponent *Startuj* codziennie o godzinie 3 w nocy (rys. 5.22).

5.3 PYTANIA KONTROLNE

1. W jaki sposób w WebML jest zrealizowane zapisywanie instancji relacji N:N?
2. Podaj domyślne parametry Komponentu Indeksu.
3. Czy i w jaki sposób można zapewnić wykonanie sekwencji operacjijako niepodzielnej całości?
4. Jakiego parametru wejściowego wymaga Komponent Decyzyjny? W jaki sposób przekazuje aktywację dalej?
5. Opisz strukturę realizującą kasowanie użytkownika pod warunkiem, że nie wprowadził żadnych komentarzy.
6. Jak można zrealizować dodawanie lub modyfikację komentarza w jednym formularzu?
7. Jak należy zrealizować wybór ocen wystawionych wcześniej niż przed dwoma dniami?
8. Jak znaleźć i wyświetlić grupy dodatkowe użytkownika wybranego z listy?
9. Czy za pomocą WebRatio można zrealizować równoległe wykonanie zapisu do kilku encji?

10. Czy Komponent Operacji można uruchomić bezpośrednio odpowiednim linkiem?Jeśli tak, to w jaki sposób?
11. Jak zrealizować walidację formularza oceniania promocji w skali od 1 do 5? Czy zamiast pola tekstowego można użyć innej opcji?
12. Jak można zrealizować zapisanie czasu i danych moderatora modyfikującego treść komentarza?
13. W jakim widoku można zdefiniować ciąg operacji wykonywanych niezależnie od użytkownika aplikacji? W jaki sposób można zdefiniować wyzwalacz takiej operacji?
14. Jakie pamiętasz wzorce typowych zadań prezentacji danych, tworzonych w modelu hipertekstu?
15. Jakie pamiętasz wzorce typowych zadań operujących na danych, tworzonych w modelu hipertekstu?

6

WebML a projektowanie aplikacji internetowych

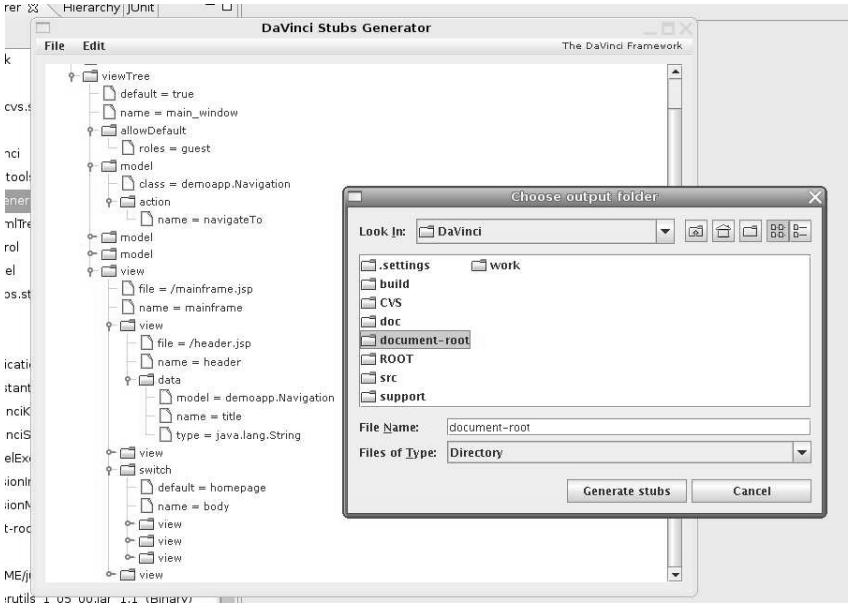
Cel

Podsumowanie zalet i wad metodyki WebML i jej implementacji w narzędziu WebRatio. Wyszczególnienie dostępnych narzędzi modelowania WebML i przedstawienie możliwości obecnej wersji narzędzia WebRatio. Omówienie dostępnych materiałów edukacyjnych oraz projektów realizowanych przez Politechnikę Lubelską, dotyczących tej technologii.

Plan

1. Zalety i wady implementacjiWebML wWebratio w projektowaniu aplikacji internetowych
2. Narzędzia implementujące WebML
3. Dostępność materiałów edukacyjnych

WebML, z racji stosunkowo długiego stażu oraz kompletności podejścia może być prezentowany jako (całkiem udany) przykład implementacji technologii MDE w domenie budowy aplikacji internetowych. Jest to jednocześnie jedyna technologia MDWE (*ang Model Driven Web Engineering*) mogąca pochwalić się narzędziem wykorzystywanym przez znaczące firmy z różnych branż (Web Models s.r.l., 2010). Co więcej, można pokusić się o stwierdzenie, że WebRatio jest jak dotąd jedynym naprawdę działającym narzędziem MDWE. Wśród wzrastającej liczby różnych narzędzi MDA, MDE, itp. (Software Pointers, 2011), (Seignard, 2010), można wyróżnić jeszcze jedno przeznaczone stricte do budowy aplikacji internetowych:DaVinciframework (Langegger A. , 2006)–otwarte środowisko MDWE dla Javy. Jest ono jednak we wczesnej fazie rozwoju, bezcelowym więc byłoby porównywanie obu rozwiązań.



Rys. 6.1 Fragment interfejsu aplikacji DaVinci

Źródło: (Langegger, Palkoska i Wagner, 2006)

6.1 Zalety i wady stosowania WebML w projektowaniu aplikacji internetowych

Niewątpliwą zaletą WebML jest jego skupienie na domenie web. Dostępne języki modelowania (głównie UML) są językami ogólnego użytku, co implikuje niezbyt dobre dopasowanie do specyfiki budowy aplikacji internetowych – różniącej się pod wieloma względami od typowych aplikacji desktopowych. Trudno więc o alternatywną technologię, co daje WebML monopolistyczną pozycję w obecnym czasie. WebML powiela również, co jest oczywiste, zalety i wady technologii MDE, na której bazuje.

Poza brakiem konkurencyjnych rozwiązań, do zalet technologii WebML można zaliczyć:

- Sposób określania wymagań funkcjonalnych aplikacji w WebML jest skierowany na użytkownika. Efektem takiego podejściajest model łatwy do objaśnienia klientowi – laikowi w dziedzinie budowy aplikacji web. Z drugiej strony projektanci i programiści nie mają problemów z ogarnięciem stawianych przed nimi zadań.
- WebML zakłada współpracę z wykorzystywanymi już metodykami budowy aplikacji internetowych. Programiści web nie powinni mieć większych kłopotów z przestawieniem się z budowania jednorazowych (choć typowych) rozwiązań dla danej aplikacji, na tworzenie i modyfikowanie (w miarę zmian technologicznych) komponentów wielokrotnego użytku. Wykorzystywanie gotowych szablonów jest i tak częstą praktyką.
- WebML zawiera dobrze dopasowaną metodykę specyfikacji wymagań. Pozwala ona na precyzyjne określenie grup użytkowników i ich ról w aplikacji, jaki zgodny z założeniami podział widoku witryny już na etapie specyfikacji wymagań. Jest to dobra podstawa do budowy modelu hipertekstu, pozwalająca również na zrozumiały dla laików opis obszarów funkcjonalności projektowanej aplikacji.
- Istnienie w pełni funkcjonalnej aplikacji deweloperskiej dowodzi sprawdzaniu się tej technologii w konkretnych rozwiązaniach (Web Models s.r.l., 2010). To z kolei sprzyja zwiększeniu przekonania klientów do proponowanej technologii.
- Model hipertekstu jest zarówno źródłem aplikacji, jak i jej zawsze aktualną dokumentacją. Wykorzystanie XML do zapisu struktury modelu dodatkowo ułatwia

wszelkie konwersje na inne formy dokumentacji, jak i potencjalne przejście na inną metodykę.

Technologia WebML nie jest również wolna od wad. Wynikają one głównie z ograniczeń narzucanych przez MDE, opisanych w rozdziale 2. Pozostałe są związane ze sposobem jej implementacji w narzędziu WebRatio:

- Etap specyfikacji wymagań i tworzenia zarysów stron nie jest wspierany przez żadne dedykowane narzędzia. Jest to niewielka wada, gdyż łatwo można wykorzystać inne, ogólnego przeznaczenia narzędzia do modelowania. Eliminuje to jednak możliwość zaimportowania zdefiniowanej struktury do dalszych etapów.
- Wbudowany mechanizm analizy błędów w modelu nie uwzględnia wszystkich możliwości wykorzystania niektórych komponentów. Powoduje to zgłaszanie błędów w poprawnych modelach, nawet tych prezentowanych w materiałach szkoleniowych producenta aplikacji. Może to powodować dezorientację u początkującego projektanta.
- Realizacja modelu prezentacji z jednej strony wymaga posiadania wiedzy w zakresie budowy szablonów stron internetowych, z drugiej strony praktycznie zmusza projektanta do stosowania układu tabelarycznego, zgodnego z siatką (grid). Całkowite wyzwolenie z tabelek jest bardzo uciążliwe od strony stworzenia odpowiedniego szablonu, jak i modelowania układu komponentów. Model ten nie jest więc zupełnie zgodny z zasadami MDE, gdzie budowa układu prezentacji komponentów powinna być niezależna od konkretnego sposobu renderowania tego układu na stronie www.

6.2 NARZĘDZIAWEBML

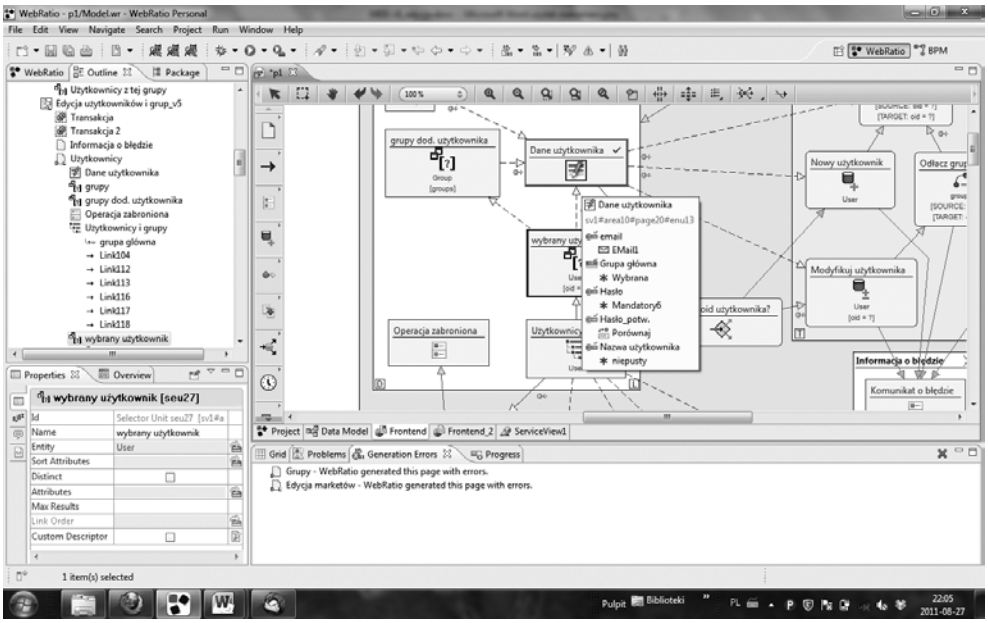
WEBRATIO

W chwili obecnej, jedynym narzędziem WebMLspełniającym postulat MDE – generacji kodu z modelu jest WebRatio. Jest to podyktowane faktem, że praktycznie ten sam zespół osób zajmował się definiowaniem technologii WebML, jak i budową

wykorzystującej jej aplikacji.

Bieżącą wersją narzędzia jest 6.1 (rys. 6.2). Udostępnia ono, oprócz możliwości budowy modeli danych i hipertekstu, następującą główną funkcjonalność (WebRatio, 2010):

- walidacja poprawności modelu,
- generowanie aplikacji do kodu JSP,
- debugowanie wygenerowanej aplikacji,
- automatyczne generowanie dokumentacji.



Rys. 6.2 Interfejs narzędzia WebRatio 6.1

Źródło: opracowanie własne

W stosunku do wcześniejszych wersji:

- poprawiono mało czytelny sposób zgłaszania błędów w wygenerowanej aplikacji,
- dodano debugowanie aplikacji,
- dodano integrację z przeglądarką Firefox – umożliwiającą inspekcję fragmentu modelu generującego zaznaczony fragment strony,

- dodano możliwość tworzenia profili generacji aplikacji, umożliwiając wygenerowanie z tego samego modelu aplikacji przeznaczonych dla różnych klientów/działów firmy.

Ogólnie, zdaniem autorów, aplikacja WebRatio jest implementacją godną uwagi i jakkolwiek nie bezbłędną, to dającą dużą pewność uzyskania poprawnie działającej aplikacji. Generowanie do kodu JSP, jednego z głównych języków budowy aplikacji web, zapewnia możliwość modyfikowania aplikacji nawetbez wykorzystywania modeli WebML. Jest to jednak dość uciążliwe z powodu niezbyt dobrej czytelności kodu.

Z punktu widzenia użytkownika, WebRatio podlega tendencji obowiązującej ogólnie dla aplikacji typu MDSD:

- Użytkownik aplikacji zajmujący się na co dzień programowaniem w językach obiektowych budowy stron www czuje się zagubiony i ograniczany zasadami modelowania WebML. Rodzi to chęć odrzucenia tej technologii jako „nieefektywnej" i „niepraktycznej". Nastawienie to zmienia się, gdy użytkownik zacznie tworzyć i wykorzystywać własne komponenty.
- Użytkownik WebRatio, który jest zainteresowany budową stron WWW ze względu na ich potencjalną użyteczność, a sam nie posiada zaawansowanej wiedzy w dziedzinie ich tworzenia, jest zachwycony łatwością i szybkością, z jaką może uzyskać aplikację realizującą jego wymagania. Jest on gotowy „przymknąć oko" na niedo końca oczekiwany wygląd lub mniejszą niż oczekiwana użyteczność wygenerowanej strony www. Po pewnym czasie zaczyna jednak odczuwać brak komponentów wykonujących specyficzne, potrzebne mu działania.

Jednakże prawdziwa zaleta technologii objawia się, gdy użytkownicy obu typów potrafią połączyć siły, każdy zajmując się swoim obszarem.

INNE MOŻLIWOŚCI MODELOWANIA WEBML

Profile UML

WebML bazuje w dużej mierze na UML, jego twórcy prezentują nawetmożliwość wykorzystania profili UML do zapisania funkcjonalności komponentów WebML(Ceri i

inni, 2003). Dowolne narzędzie UML potrafiące wykorzystać taki profil, mogłoby służyć do modelowania WebML. Profil taki nie doczekał się jednak wykorzystania, gdyż również zdaniem twórców WebML(Cabot i Butti, 2011)nie wystarczało to do reprezentacji wszystkich aspektów aplikacji internetowej. Dodatkowo notacja UML okazała się nieporęczna przy próbach specyfikacji aplikacji internetowych z uwzględnieniem znacznej ilości detali.Nie bez znaczenia jest również trudność, jaką UML wykazuje przy podejmowaniu prób generowania kodu wykonawczego aplikacji.

Komponenty Visio

W oficjalnym serwisie internetowym technologii WebML można pobrać zestaw komponentów MS Visio, odpowiadających komponentom WebML. Dostępne są wersje: kolorowa i czarno-biała.Umożliwiają one jedynie wizualizację koncepcji konkretnego modelu hipertekstu, bez możliwości jego walidacji, czy generacji kodu.

6.3 DOSTĘPNOŚĆ MATERIAŁÓW EDUKACYJNYCH

Podstawowym źródłem informacji o WebML jest książka napisana przez jego twórców „Designing Data-Intensive Web Applications” (Ceri i inni, 2003) wydana po raz pierwszy w 2002r.Zawiera ona zarówno objaśnienie koncepcji języka modelowania dla web, metodyki, poszczególnych modeli, jak i przykładowe rozwiązania modelowane przy pomocyWebML.

Wśród źródeł elektronicznych można wyróżnić:

- **webml.org**
  Strona domowa technologii WebML. Zawiera podstawowe informacje o technologii, również w języku polskim. Umieszczone są tam materiały edukacyjne prezentujące podstawowe zasady modelowania WebML.
- **wiki.webratio.com**
  Baza wiedzy o aplikacji WebRatio, zawierająca zbiór artykułów prezentujących różne aspekty budowania modeli danych, hipertekstu i prezentacji oraz opis pozostałej funkcjonalności aplikacji.

- **learnmde.org**

  Strona domowa projektu koordynowanego przez Politechnikę Lubelską, skupiającego naukowców zajmujących się technologiami MDE z Włoch (twórcy WebML), Francji, Hiszpanii i Polski. Projekt ma na celu wypracowanie jednolitych metod i materiałów nauczania technologii MDE.

- **wrportal.pl**

  Strona prowadzona przez inż. Wojciecha Guzowskiego, absolwenta Państwowej Wyższej Szkoły Zawodowej w Nysie. Strona jest poświęcona tworzeniu aplikacji internetowych w WebRatio. Zawiera wiele artykułów opisujących praktyczne rozwiązania modelowania w WebRatio.

Ponadto Politechnika Lubelska prowadzi następujące projekty obejmujące technologie MDE – WebML:

1. **Absolwent na miarę czasu**

   Projekt ten obejmuje stworzenie materiałów do nauczania nowatorskich dziedzin IT. Wśród nich powstaje zbiór materiałów dotyczących technologii WebML. Materiały te są prekursorskie i pierwsze w Polsce o tak kompleksowym podejściu do tematu. Składa się na nie komplet w postaci: wykładów, materiałów laboratoryjnych (z zadaniami do wykorzystania przez prowadzących przedmiot), materiałów e-learningowych (pozwalających na utrwalenie wiadomości ogólnych i rozszerzenie wiedzy o specyficzne zagadnienia związane z tematem) oraz niniejszy podręcznik będący wykładnią teoretyczną.

2. **„MDE Expertise - Exchanging knowledge, techniques and experiences around Model Driven Engineering education”**

   Projekt jest współfinansowany ze środków UE w ramach programu "Uczenie się przez całe życie" (LIFELONG LEARNING PROGRAMME) Leonardo da Vinci – Projekty Partnerskie (LdVPartnerships). Projekt skupia się na wspólnych badaniach, wymianie wiedzy i tworzeniu materiałów edukacyjnych w obszarze projektowania MDE. Celem projektu jest ujednolicenie wiedzy i technik nauczania związanego z MDE na europejskich uczelniach. W projekcie uczestniczą partnerzy z: Włoch – Politechnika w Mediolanie (Politecnico di Milano), Hiszpanii – Uniwersytet w Alicante (University of Alicante), Francji – (Ecole des Mines de Nantes) i Polski – Politechnika Lubelska.

Literatura

Adobe. (2011). The Expressive Web. Pobrano 8 25, 2011 z lokalizacji http://beta.theexpressiveweb.com/

Apache Ant. (2011). Apache Ant™ 1.8.2 Manual. Pobrano 08 24, 2011 z lokalizacji The apache ant project: http://ant.apache.org/manual/index.html

Apache POI. (2011). Apache POI - the Java API for Microsoft Documents. Pobrano 8 24, 2011 z lokalizacji The apache POI project: http://poi.apache.org/

Boggs, W. i Boggs, M. (2002). Mastering UML with Rational Rose 2002. Sybex.

Cabot, J. iButti, S. (2011). WebRatio – MDD tool for building Web/SOA business applications. Pobrano 08 25, 2011 z lokalizacji MoOdelingLAnguages: http://modeling-languages.com/coffee-stefano-butti-webratio-mdd-tool-building-websoa-business-applications

Ceri, S. iinni. (2003). Designing Data – Intensive Web Applications. Elsevier Science.

den Haan, J. (2008a). MDA and Model Transformation. Pobrano 8 23, 2011 z lokalizacji The Enterprise Architect: http://www.theenterprisearchitect.eu/archive/2008/02/18/mda-and-model-transformation

den Haan, J. (2008b). Model Driven Engineering. Pobrano 08 23, 2011 z lokalizacji The Enterprise architect: http://www.theenterprisearchitect.eu/archive/2008/03/14/model-driven-engineering

den Haan, J. (2008c). MDA, Model Driven Architecture, basic concepts. Pobrano 08 23, 2011 z lokalizacji The Enterprise architect: http://www.theenterprisearchitect.eu/archive/2008/01/16/mda_model_driven_architecture_

den Haan, J. (2008d). Combining general purpose languages and domain specific languages for Model Driven Engineering. Pobrano 08 23, 2011 z lokalizacji The Enterprise architect: http://www.theenterprisearchitect.eu/archive/2008/04/15/combining-general-purpose-languages-and-domain-specific-languages-for-model-driven-engineering#metamodel

den Haan, J. (2009a). MDE - Model Driven Engineering - reference guide. Pobrano 08 23, 2011 z lokalizacji The Enterprise architect: http://www.theenterprisearchitect.eu/archive/2009/01/15/mde---model-driven-engineering----reference-guide

den Haan, J. (2009b). 15 reasons why you should start using Model Driven Development. Pobrano 08 23, 2011 z lokalizacji The Enterprise architect: http://www.theenterprisearchitect.eu/archive/2009/11/25/15-reasons-why-you-should-start-using-model-driven-development

den Haan, J. (2009c). 8 reasons why Model-Driven Development is dangerous. Pobrano 08 23, 2011 z lokalizacji The Enterprise architect: http://www.theenterprisearchitect.eu/archive/2009/06/25/8-reasons-why-model-driven-development-is-dangerous

Eclipse. (2011). Eclipse documentation. Pobrano 8 24, 2011 z lokalizacji Eclipse: http://help.eclipse.org/indigo/index.jsp

Gitzel, R. i Korthaus, A. (2004). The Role of Metamodeling in Model-Driven Development. Proceeedings of the 8th World Multi-Conference on Systemics, Cybernetics and Informatics. Orlando.

JBoss community. (2011). Hibernate. Pobrano 08 24, 2011 z lokalizacji http://hibernate.org/

Kęsik, J. i Żyła, K. (2008). Implementacja języka WebML w WebRatio na przykładzie prostego forum remontowego. VariaInformatica. Katowice: PTI.

Langegger, A. (2006). DaVinci - A Model-driven Web Engineering Framework for Java. Pobrano 08 25, 2011 z lokalizacji SourceForge: http://davinci4j.sourceforge.net/#dal

Langegger, A., Palkoska, J. i Wagner, R. (2006). DaVinci - A Model-driven Web Engineering Framework. International Journal of Web Information Systems, vol. 2, strony 119-132.

Lindenberg, N. (2003). Developing Multilingual Web Applications Using JavaServer Pages Technology. Pobrano 8 23, 2011 z lokalizacji Oracle: http://java.sun.com/developer/technicalArticles/Intl/MultilingualJSP/

OMG. (2003). MDA Guide Version 1.0.1. Pobrano 08 23, 2011 z lokalizacji OMG Model Driven Architecture: http://www.omg.org/cgi-bin/doc?omg/03-06-01

OMG. (2007). OMG Unified Modeling Language (OMG UML), Infrastructure, V2.1.2. OMG Document Number: formal/2007-11-04.

OMG. (2011). Meta Object Facility (MOF) Core specification. Pobrano 08 23, 2011 z lokalizacji Meta Object Facility (MOF™) Core: http://www.omg.org/spec/MOF/2.4/Beta2/PDF/

Oracle. (2010). Java Server Pages Technology. Pobrano 08 20, 2011 z lokalizacji Oracle: http://www.oracle.com/technetwork/java/javaee/jsp/index.html

Oracle. (2011a). JavaMail. Pobrano 08 24, 2011 z lokalizacji http://www.oracle.com/technetwork/java/javamail/index.html

Oracle. (2011b). Java SE Technologies - Database. Pobrano 8 24, 2011 z lokalizacji http://www.oracle.com/technetwork/java/javase/jdbc/index.html

Quartz. (2011). Quartz Overview. Pobrano 08 24, 2011 z lokalizacji http://www.quartz-scheduler.org/overview

Schmidt, D. C. (2006). Guest Editor's Introduction: Model-Driven Engineering. Computer, 2(39), strony 25-31.

Seignard, X. (2010). Model driven tools : the big list! Pobrano 8 23, 2011 z lokalizacji MDA, MDSD, MDE, MDWhatever!: http://mdwhatever.free.fr/index.php/2010/06/model-driven-tools-the-big-list/

Stahl, T. iVölter, M. (2006). Model-Driven Software Development: Technology, Engineering, Management. Wiley.

Strona domowa metodyki WebML. (2011, sierpień). Pobrano z lokalizacji http://webml.org

Tanenbaum, A. i van Steen, M. (2002). Distributed Systems. Principles and Paradigms. Prentice Hall.

The Apache DB Project. (2011). Apache Derby. Pobrano 08 24, 2011 z lokalizacji http://db.apache.org/derby/

The Apache Software foundation. (2010). Apache Tomcat 6 documentation. Pobrano 8 23, 2011 z lokalizacji Apache Tomcat: http://tomcat.apache.org/tomcat-6.0-doc/index.html

The apache software foundation. (2011). STRUTS 2. Pobrano 8 24, 2011 z lokalizacji http://struts.apache.org/2.2.3/index.html

Voelter, M., Kolb, B., Efftinge, S. iHaase, A. (2006). From Front End To Code - MDSD in Practice. Pobrano 08 24, 2011 z lokalizacji eclipse.org: http://www.eclipse.org/articles/Article-FromFrontendToCode-MDSDInPractice/article.html

Völter, M. (2005). Graphical DSLs. Pobrano 8 10, 2011 z lokalizacji voelter - ingenieurbüro für softwaretechnologie: http://www.voelter.de/data/presentations/mdsd-tutorial/05_GraphicalDSLs.pdf

Völter, M. (2005). Model-Driven Software Development Tutorial. Pobrano 8 23, 2011 z lokalizacji voelter - ingenieurbüro für softwaretechnologie: http://www.voelter.de/data/presentations/mdsd-tutorial/01_Introduction.pdf

W3C. (2004). Web Services Architecture. Pobrano 8 23, 2011 z lokalizacji W3C Working Group Note: http://www.w3.org/TR/ws-arch/

W3C. (2011a). SGML. Pobrano 8 15 , 2011 z lokalizacji http://www.w3.org/MarkUp/SGML/

W3C. (2011b). HTML. Pobrano 8 23, 2011 z lokalizacji http://www.w3.org/html/

W3C. (2011c). XML. Pobrano 8 20, 2011 z lokalizacji http://www.w3.org/XML/

W3C. (2011d). The Extensible Stylesheet Language Family (XSL). Pobrano 8 25, 2011 z lokalizacji http://www.w3.org/Style/XSL/

W3Schools. (2011a). CSS Tutorial. Pobrano 8 25 , 2011 z lokalizacji http://www.w3schools.com/Css/

W3Schools. (2011b). AJAX Introduction. Pobrano 8 23, 2011 z lokalizacji http://www.w3schools.com/Ajax/ajax_intro.asp

W3Schools. (2011c). SQL Tutorial. Pobrano 8 23 , 2011 z lokalizacji http://www.w3schools.com/sql/default.asp

W3schools. (2011d). JavaScript Introduction. Pobrano 8 23, 2011 z lokalizacji http://www.w3schools.com/JS/js_intro.asp

W3Schools. (2011d). JavaScript Introduction. Pobrano 8 23, 2011 z lokalizacji http://www.w3schools.com/JS/js_intro.asp

Web Models s.r.l. (2010). A tool for the most innovative companies. Pobrano 8 24 , 2011 z lokalizacji WebRatio: http://www.webratio.com/portal/contentPage/en/Customers

WebRatio. (2009). How to manage users and groups. Pobrano 8 23, 2011 z lokalizacji Wiki WebRatio: http://wiki.webratio.com/index.php/How_to_manage_users_and_groups

WebRatio. (2009). Webratio User Guide. WebRatio.

WebRatio. (2010). Web Model. Pobrano 8 24, 2011 z lokalizacji WebRatio WebML Wiki: http://wiki.webratio.com/index.php/Category:Web_Model

WebRatio. (2011). How to dinamically create URLs. Pobrano 8 2011, 2011 z lokalizacji Wiki WebRatio: http://wiki.webratio.com/index.php/How_to_dinamically_create_URLs

Żyła, K. i Kęsik, J. (2010). Creation of the user-oriented requirements specification of the data intensive web application basing on WebML. InformatykZakładowy, (strony 203-218).

Żyła, K. iKęsik, J. (2010a). Usability comparison of WebRatio and symfony for educational purposes. Prace Instytutu Elektrotechniki, zeszyt 247, (strony 223-240). Warszawa.

Żyła, K. i Kęsik, J. (2010b). Zintegrowane środowisko programistyczne IDE WebRatio 5.1 wspomagające nauczanie MDE. Logistyka 6/2010, (strony 3931-3941).

Indeks