POLSKO-JAPOŃSKA WYŻSZA SZKOŁA TECHNIK KOMPUTEROWYCH

PRACA MAGISTERSKA

Nr .....

Tworzenie modularnych aplikacji na podstawie struktury bazy danych

Student/studentka	**Wojciech Pragacz**
Nr albumu	4981
Promotor	dr inż. Mariusz Trzaska
Specjalność	Inżynieria Oprogramowania i Baz Danych
Katedra	Systemów Informacyjnych
Data zatwierdzenia tematu	29.02.2008
Data zakończenia pracy	31.01.2010

Podpis promotora pracy .....

Podpis kierownika katedry .....

Streszczenie

Celem pracy jest stworzenie narzędzie dla programistów, który odciąży ich od najbardziej trywialnych i podstawowych prac podczas wykonywanych przez nich obowiązków. Przedmiotem pracy będzie generator kodu, tworzący gotową implementację aplikacji w języku C#, ASP.NET oraz XML na podstawie struktury bazy danych. Taki automatyczny generator kodu jest podstawą dla programistów piszących aplikacje biznesowe w środowisku korporacyjnym. Pozwala on być bardziej efektywnym i nie tracić czasu na powtarzalne zadania, które stanowią sporą cześć pracy.

Na rynku istnieje wiele rozwiązań pozwalających tworzyć automatycznie kod programów lub aplikacje na podstawie metadanych pobieranych ze wskazanych baz danych, jednak nie ma żadnych standardów w tym zakresie i każdy producent stosuje własne wbudowane rozwiązania języki lub API. Oznacza to, że trzeba ponieść znaczne nakłady czasowe na początku, aby w ogóle móc pracować z danym narzędziem. Poza tym bardzo często są to rozwiązania komercyjne i to dosyć drogie.

Głównym założeniem pracy jest stworzenie takiego narzędzia, które pozwoli programiście od razu pracować, bez konieczności nauki nowych rzeczy. Chodzi przecież o to, żeby przyśpieszyć pracę, a nie uczyć się od podstaw nowego narzędzia tracąc czas, który ma być zaoszczędzony. Aplikacja ta jest skierowana szczególnie do programistów znających język XSLT. Osoby te odniosą największy zysk czasowy, ponieważ nie będą się musiały niczego nowego uczyć, tylko przystąpią od razu do pracy z aplikacją.

W pracy opisano również problemy, jakie napotkano podczas implementacji oraz możliwe dalsze zastosowania.

Podziękowania

Autor pracy dziękuje kierownictwu firmy KPMG, w szczególności Managerowi IT Jarosławowi Bielanowi, za umożliwienie wykorzystania infrastruktury komputerowej w celu zaimplementowania i przetestowania niniejszej pracy dyplomowej.

Spis treści

1. Wstęp

Praca w zawodzie programisty w dużych firmach, a zwłaszcza w korporacjach, wiąże się z kilkoma podstawowymi wymogami, m.in. z wysoką produktywnością. Szefowie oczekują od programistów realizacji powierzonych zadań w jak najkrótszych terminach. Dodatkowo, tworzone przez nich rozwiązania powinny mieć ciekawą i bogatą funkcjonalność z jednoczesnym nastawieniem na intuicyjną obsługę. Biorąc pod uwagę te wszystkie czynniki, a zwłaszcza wymóg dostarczenia aplikacji w jak najszybszym czasie, czyli tzw. „time to market", liczy się każdy sposób, aby przyspieszyć tworzenie oprogramowania oraz zdążyć z wykonaniem produktu przed konkurencją. Powstało i nadal powstaje coraz więcej rozwiązań ułatwiających życie fachowcom zajmującym się programowaniem. Do takich rozwiązań można zaliczyć różnego rodzaju biblioteki gotowych komponentów, kontrolek UI, biblioteki wykorzystujące mechanizm adnotacji oraz generatory kodu. Oczywiście każde z tych rozwiązań ma zarówno swoje plusy, jak i minusy.

1.1. Cel pracy

Celem niniejszej pracy jest stworzenie i przedstawienie narzędzia umożliwiającego programistom automatyczne generowanie kodu aplikacji lub nawet działającego programu na podstawie metadanych wskazanej przez użytkownika bazy danych. Dotychczas dostępne na rynku rozwiązania wymagały od programisty nauczenia się wbudowanych w nie języków lub API stworzonych przez producentów tych programów, przez co globalny zysk z zastosowania tych rozwiązań nie był już tak duży. Nasze narzędzie ma być stworzone w taki sposób, aby pomagać w codziennej pracy, jednocześnie biorąc pod uwagę obecne wymagania rynku komercyjnego – zwłaszcza oszczędność czasu i automatyzację pracy. Środkiem do uzyskania tego celu będzie prosty i przejrzysty interfejs graficzny oraz system szablonów bazujących na powszechnie znanych standardach.

1.2. Rezultat pracy

Rezultatem pracy jest zaimplementowany program - generator kodu. Aplikacja ta pobiera z określonej bazy danych jej strukturę i pozwala przy pomocy prostego interfejsu użytkownika utworzyć aplikację WWW, w konwencji CRUD, podzieloną na moduły odpowiadające kolejno wybranym tabelom. Użytkownik dodając kolejne tabele do rozwiązania, ustala, które kolumny i w jakim sposób mają się pokazywać na liście rekordów oraz na formatce do wprowadzania i edycji danych.

Jednocześnie w tle tworzone są klasy niższych warstw programu, czyli obiekty warstwy Service Level, obiekty POCO oraz pliki konfiguracyjne NHibernate.

W rezultacie wykreowane projekty można zapisywać i otwierać w celu późniejszych zmian. Dodatkowo dodano możliwość uruchamiania wygenerowanych modułów w celu natychmiastowego podglądu utworzonego rozwiązania. Dzięki tej funkcjonalności możemy proste aplikacje natychmiast opublikować, jeżeli nie wymagają one dodatkowych prac programistycznych (np. wprowadzanie autoryzacji).

1.3. Organizacja pracy

Praca zostanie napisana w następujących etapach/rozdziałach:

- zaczniemy od opisania stanu sztuki, czyli przegląd generatorów kodu dostępnych na rynku zarówno komercyjnym jak i bezpłatnych;

- następnie scharakteryzujemy, czym się powinien charakteryzować idealne narzędzie tego typu. Omówimy również zagadnienia architektoniczne związane z budową aplikacji 3 warstwowych;

- w kolejnym rozdziale przedstawimy platformy i programy wykorzystane przy tworzeniuniniejszej pracy dyplomowej;

- w opisie implementacji znajdą się informacje na temat sposobów pozyskiwania struktury baz danych, wykorzystywania szablonów XML/XSL, projektu interfejsu użytkownika oraz przykład pracy z samym programem;

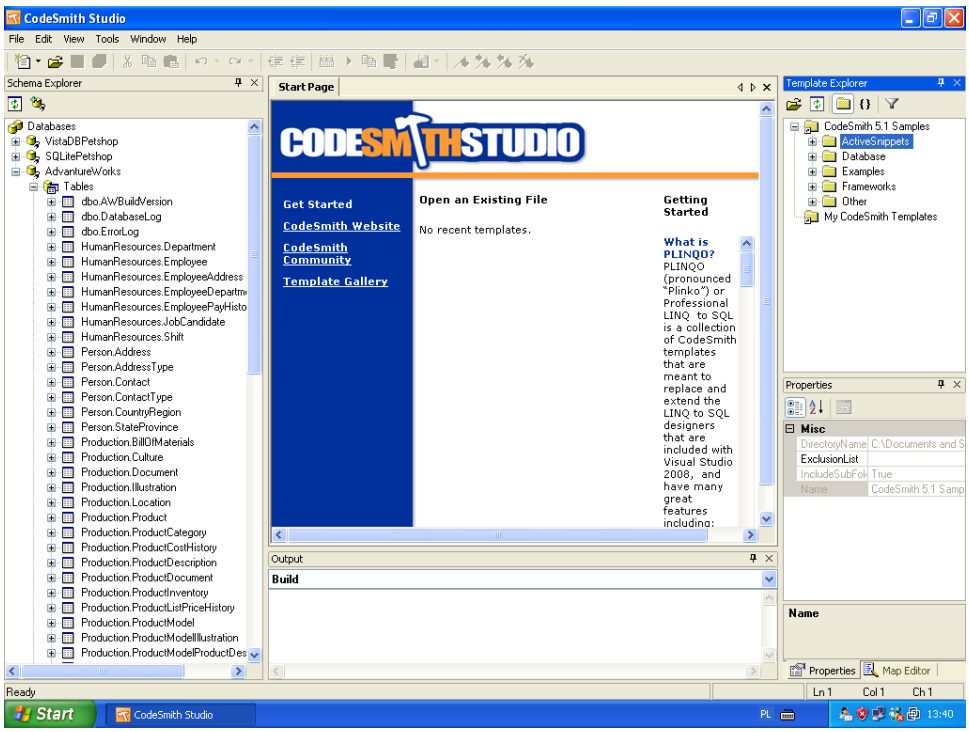
- na zakończenie przyjrzymy się problemom, jakie napotkał Autor pracy, a także ich rozwiązaniom oraz możliwościach dalszego rozwoju oprogramowania.

2. Opis stanu sztuki – przegląd dostępnych rozwiązań na rynku

Na rynku istnieje już kilka rozwiązań pozwalających generować kod na postawie struktury bazy danych. Wybrano najpopularniejsze z nich, aby przedstawić je w tej pracy. Autor postara się uwypuklić ich zarówno mocne strony, a także wady, które stały się inspiracją do zajęcia się tym tematem.

2.1. CodeSmith

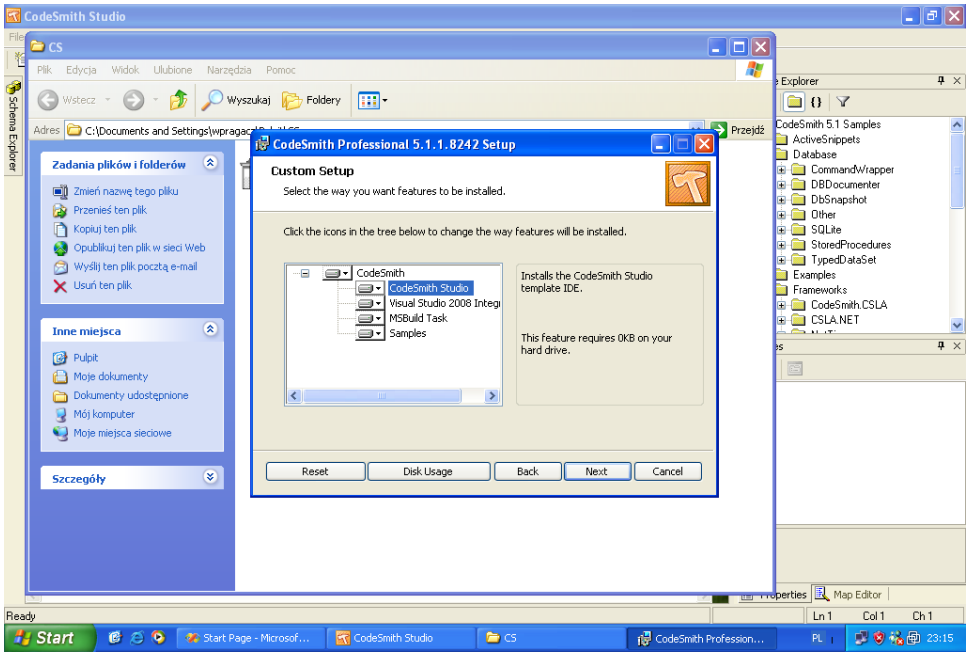
Pierwszym z opisanych produktów będzie CodeSmith [1]. Jest to najpopularniejsze rozwiązanie do generacji kodu, szczególnie dla osób tworzących oprogramowanie pod systemy Microsoft Windows. Aplikacja ta pozwala na podstawie szablonów dostarczonych z programem oraz metadanych pobieranych z baz Microsoft SQL Server lub Oracle generować kod w językach C# lub Visual Basic .NET, który później jest importowany bezpośrednio do projektów w Microsoft Visual Studio.



Rysunek 1. CodeSmith - okno główne

System dostępny jest tylko w wersji komercyjnej. Osiągalne są dwie edycje Standard Edition oraz Professional Edition, kosztujące odpowiednio 99$ lub 399$. Wydanie Standard jest mocno okrojone, dlatego zaleca się bardziej zakup pełnego wariantu Professional, który pozwala na wydajne korzystanie z tego narzędzia. Opcja Professional posiada dodatkowo miedzy innymi: CodeSmith Studio IDE (Rys. 1), integracja z Visual Studio czy na przykład wsparcie dla MSBuild.

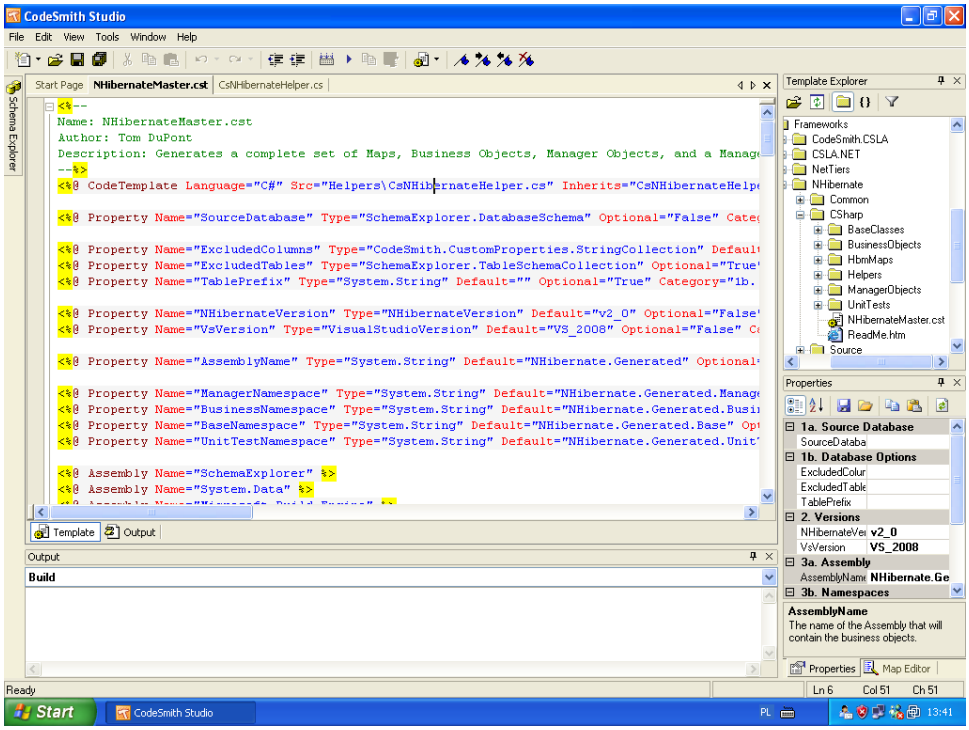
Instalacja produktu jest całkiem prosta i intuicyjna. Produkt zapisuje się przy pomocy dostarczonego instalatora, pobieranego w formie pliku *.exe. Podczas instalacji jest możliwość wybrania, które elementy CodeSmith mają być dodane. Możemy wybrać opcje (Rys. 2): CodeSmith Studio, Visual Studio 2008 integration, MSbuild Task, Samples. Przy pierwszym uruchomieniu należy podać klucz licencyjny lub rozpocząć 30-dniowy okres testowy.



Rysunek 2. Instalacja CodeSmith

Głównym narzędziem w pakiecie jest CodeSmith Studio IDE. Produkt ten przypomina z wyglądu starą wersję Microsoft Visual Studio 2003. Ekran składa się z głównego okna – edytora kodu oraz okien narzędziowych po lewej i prawej stronie (Rys. 1). Po lewej stronie znajduje się Schema Explorer. W okienku tym możemy przeglądać strukturę podłączonych baz danych oraz możemy podłączyć nową bazą danych. Z prawej strony znajdują się domyślnie dwa okna. Template Explorer przedstawia w strukturze drzewiastej szablony, z jakich może skorzystać użytkownik, natomiast na dole znajduje się okno właściwości, które pozwala podglądać i edytować właściwości templatów w zależności od kontekstu (Rys. 3).

Okno właściwości jest tworzone dynamicznie, na podstawie zadeklarowanych w pliku szablonu zmiennych. W oknie Template Explorer można przeglądać strukturę poszczególnych templatów i powiązanych plików. Użytkownik ma możliwość dodawania, modyfikowania i usuwania plików. W dolnej części ekranu znajduje się Output window, dzięki któremu programista ma możliwość śledzenia komunikatów o błędach podczas kompilacji.



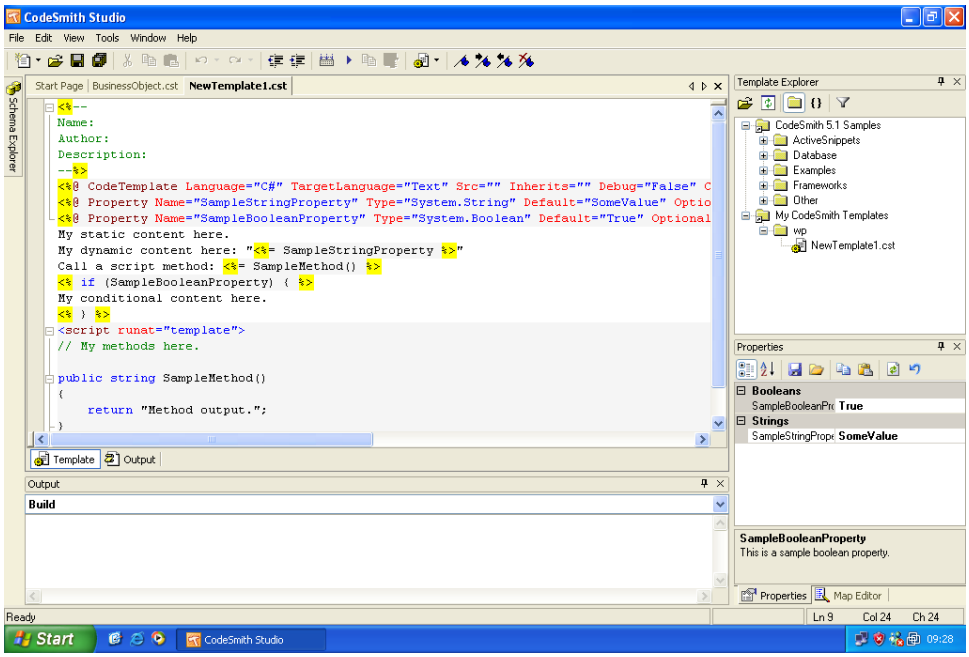
Rysunek 3. Przykładowy plik z definicją właściwości

CodeSmith jest dostarczony wraz ze zbiorem gotowych szablonów do generowania kodu. Oprócz tego użytkownik może dodawać nowe, własne templaty. Są one tworzone w głównych językach - .NET Framework czyli C#, Visual Basic .NET oraz JScript .NET.

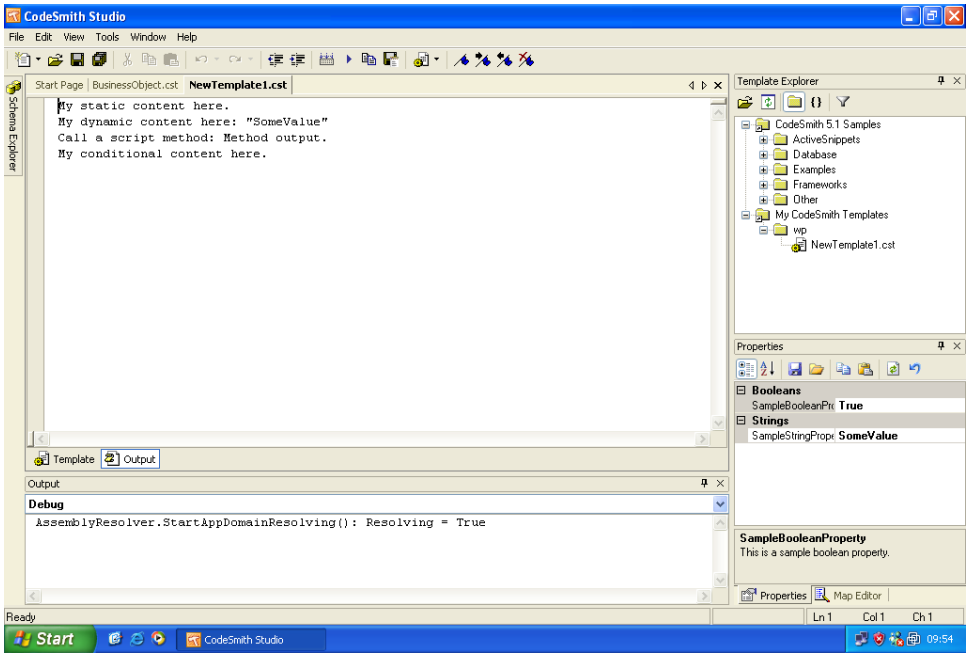
Nowy szablon jest sporządzany od razu z przykładowym kodem, gdzie mamy przedstawione podstawowe elementy potrzebne do jego generowania. Mamy przykład zadeklarowania dwóch parametrów SampleStringProperty i SampleBooleanProperty (Rys. 4 i 5). Dzięki nim widzimy, w jaki sposób możemy przekazywać parametry wejściowe przed uruchomieniem generatora. Funkcja ta przydaje się na przykład do wpisania przestrzeni nazw wykorzystywanej w naszym projekcie tak, aby nie trzeba był później już edytować nowo utworzonych plików. Jest to również przykład metody zaimplementowanej w C# oraz przykład użycia jej bezpośrednio w szablonie - szczególnie przydatne do tzw. zadań specjalnych.

Tworzenie szablonów budzi skojarzenia z tworzeniem stron w ASP.NET. Mamy tutaj wyrażenia znane nam właśnie z tej technologii <% %> lub <%= %>. Kod C#, Visual Basic .NET lub JScript .NET może być umieszczany wewnątrz Tagu <SCRIPT RUNAT=”template”>. Do templatów można przypisywać pliki code-behind, w których umieszcza się bardziej skomplikowaną logikę biznesową.

Głównym zadaniem CodeSmith jest generowanie kodu na podstawie struktury bazy danych. W tym celu należy wykorzystać załączona bibliotekę Schema Explorer Schema Discovery API. Pozwala on na odczytanie struktury i metadanych we wskazanej bazie danych. Należy do tego użyć wybrany na początku język programowania i odpowiednio zaimplementować wykorzystanie API.

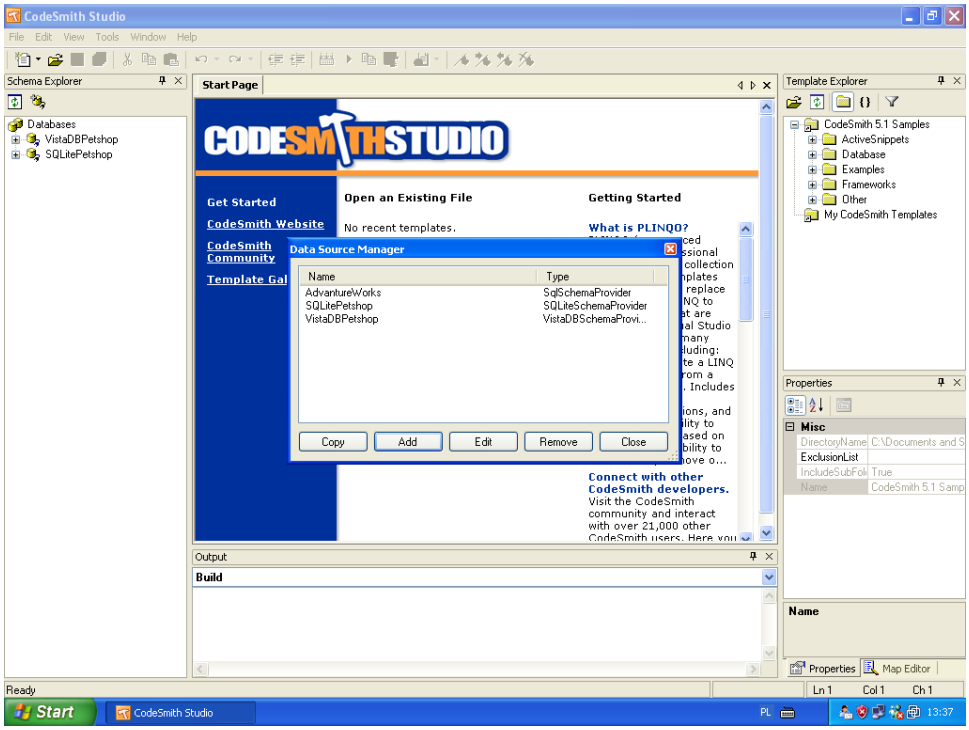


Rysunek 4. Standardowy szablon CodeSmith

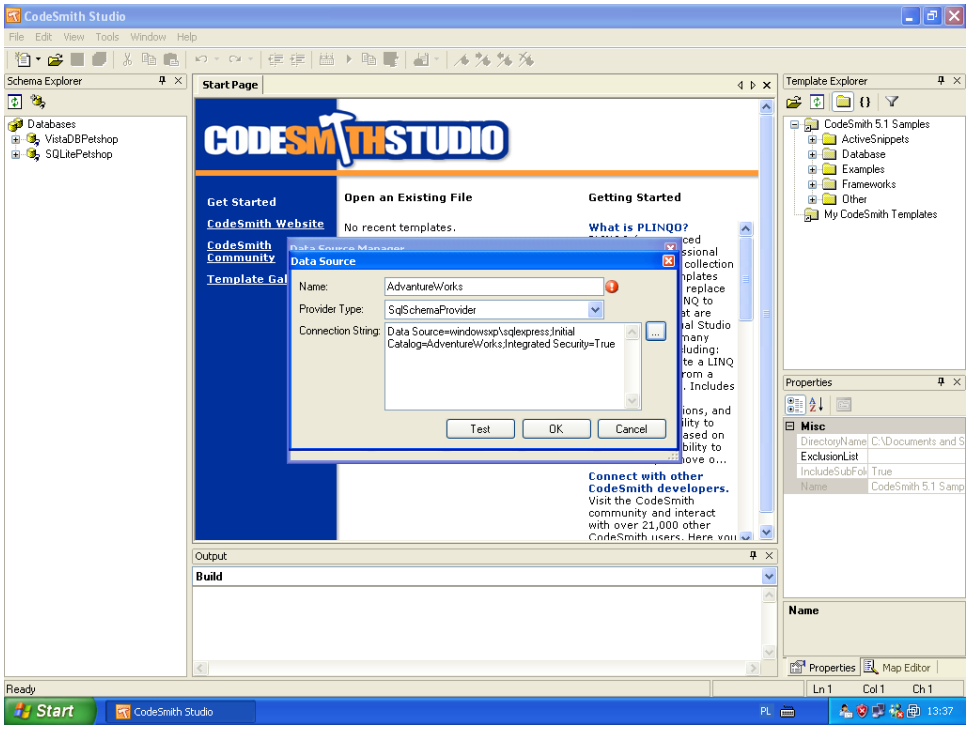


Rysunek 5. Standardowy szablon CodeSmith - wynik

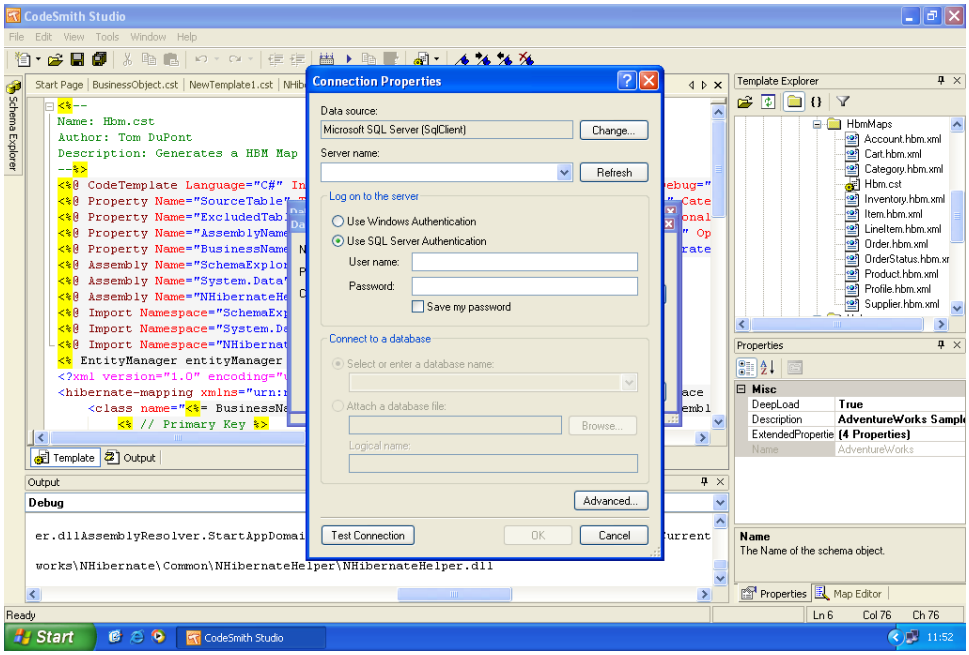
Podłączenie do bazy danych następuje poprzez okienko Data Source Manager (Rys. 6). Użytkownik ma możliwość zarządzania listą połączeń: może dodawać, modyfikować oraz usuwać połączenia. Istnieje nawet możliwość utworzenia nowego połączenia na podstawie starego połączenia. Robi się to poprzez wejście w edycje istniejącego połączenia i zapisanie go jako nowego, po uprzedniej modyfikacji. Aby utworzyć nowe połączenie należy wprowadzić: nazwę połączenia, typ providera oraz tzw. connection string (Rys. 7). W zależności od wybranego providera istnieje możliwość skorzystania z formularza pomocniczego do utworzenia connection string. Z takiego ułatwienia można skorzystać przy SQlSchemaProvider oraz OracleSchemaProvider. Po wciśnięciu przycisku połączenia otwiera się formularz znany z Microsoft Visual Studio (Rys. 8). Najwyraźniej autorzy CodeSmith korzystają z tej samej biblioteki. Dla pozostałych providerów, w tym providerów do baz opensourcowych (np. MySQLSchemaProvider i PostgreSQlSchemaProvider) nie ma takiego ułatwienia i trzeba connection string utworzyć ręcznie.



Rysunek 6. Lista połączeń bazodanowych



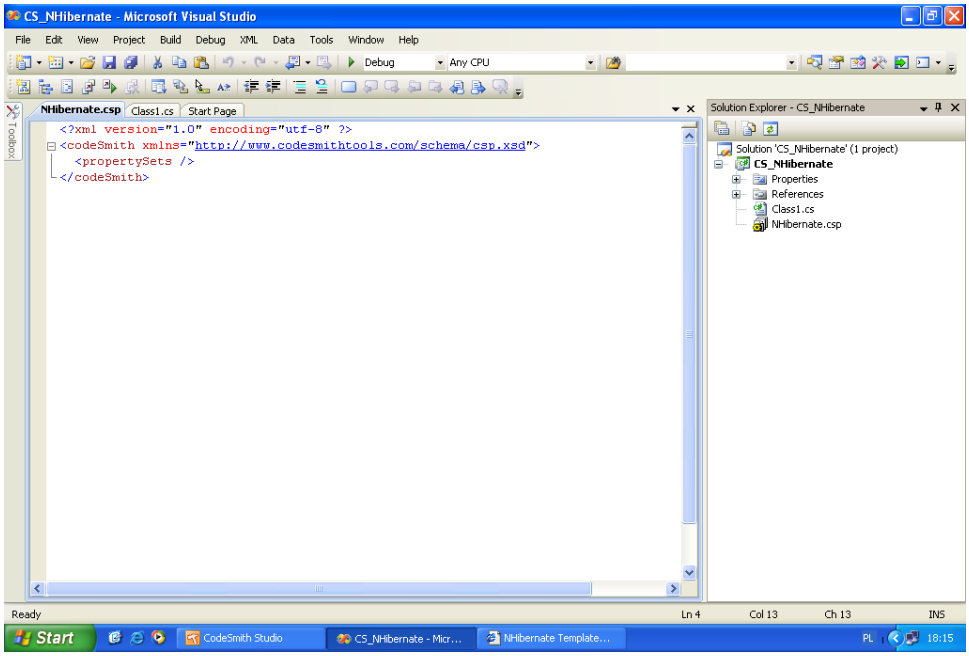
Rysunek 7. Okno ustawień połączenia bazodanowego



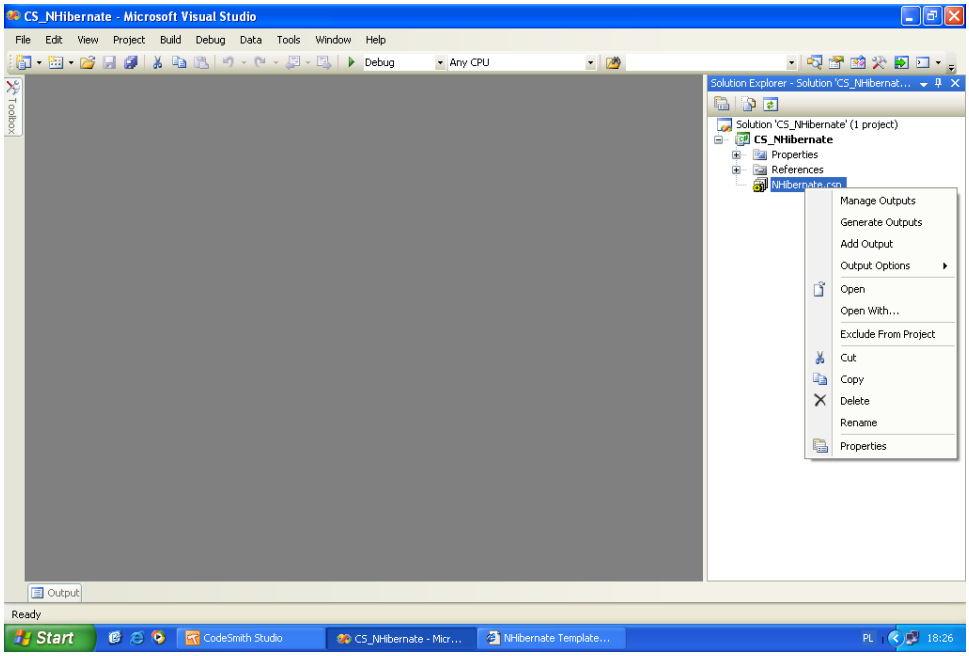
Rysunek 8. Wizard dla połączeń OLEDB

Standardowo CodeSmith wspiera następujące bazy danych: Ado, MySQL, Oracle, PostgreSQL, SQLCompact, SQLLite, SQL Server oraz VistaDB

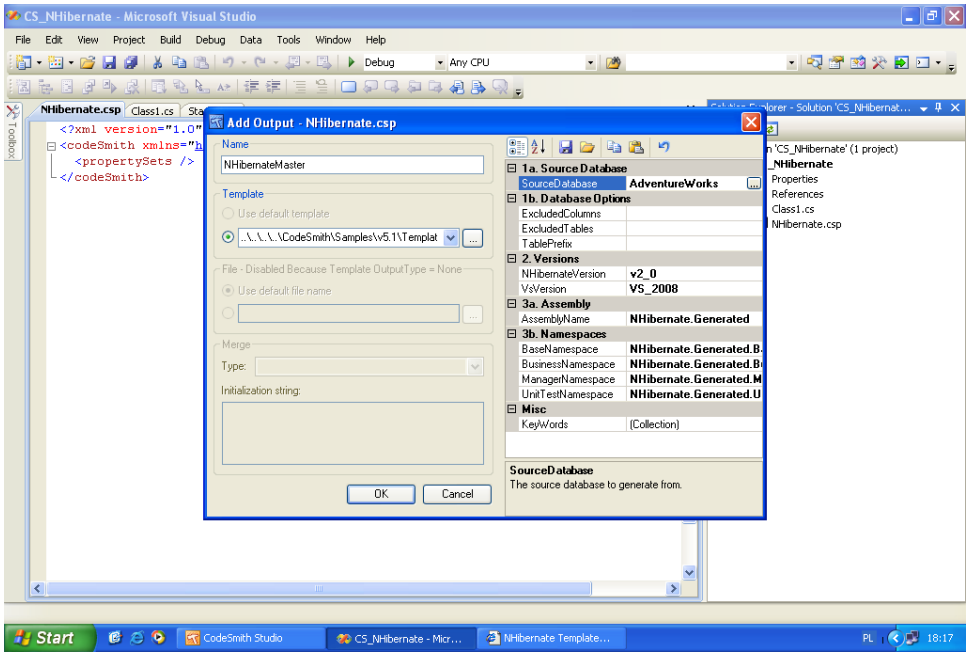
Samo tworzenie kodu najwygodniej jest przeprowadzić z poziomu Microsoft Visual Studio. Dzięki temu w przypadku generowania pliku do projektu, co najczęściej jest robione, pliki te są od razu dodawane i rejestrowane w środowisku. Generowanie należy rozpocząć od utworzenia nowego projektu w środowisku Microsoft Visual Studio. Później należy dodać nowy element CodeSmith Project (Rys.9). Następnie konfigurujemy ten element wybierając opcję „Add Output" (Rys. 10). W tym przypadku wskazujemy template z którego będziemy korzystali. Po wybraniu templatu mamy możliwość ustawić właściwości tworzenia kodu. Te konkretne właściwości są wyspecyfikowane w pliku szablonu (Rys. 4). Jest to bardzo ciekawe rozwiązanie. Z jednej strony pozwala na elastyczne definiowanie danych wejściowych, z drugiej strony przedstawia przejrzysty i łatwy do wprowadzania interfejs użytkownika. W tym przypadku mamy możliwość wybrania wcześniej zarejestrowanej bazy danych, wersji NHibernate do której ma być generowany kod wraz z plikami mapującymi, nazwy assembly oraz nazwy poszczególnych przestrzeni nazw (namespaces) (Rys.11). Na koniec pozostaje nam stworzenie Projektu. Z menu kontekstowego wybieramy „Generate Outputs" i obserwujemy, jak do projektu dodawane są automatycznie kolejne pliki.



Rysunek 9. Nowy projekt Visual Studio z dodanym projektem CodeSmith

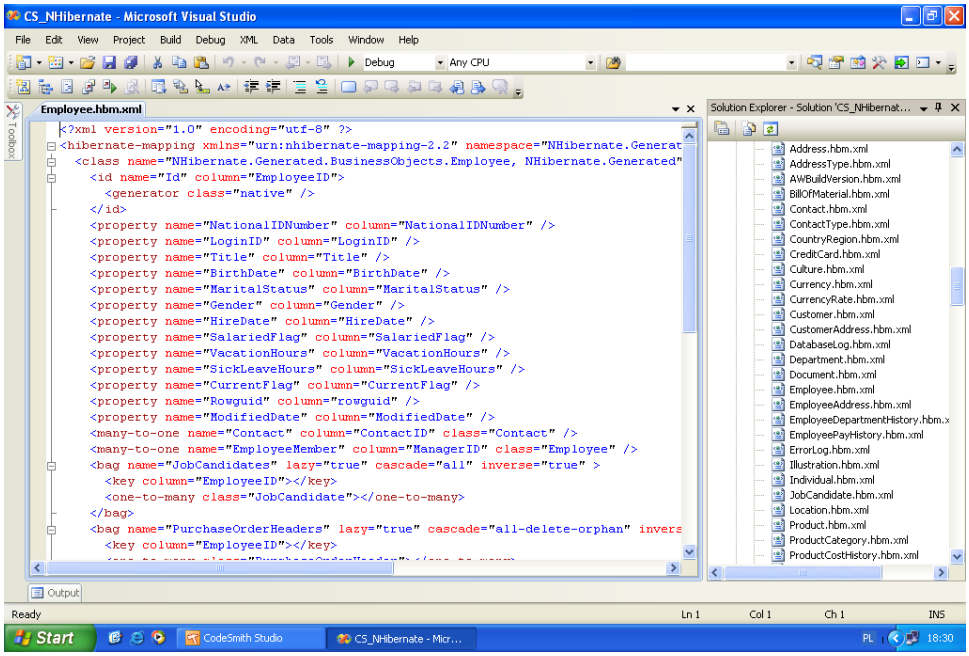


Rysunek 10. Opcje z menu kontekstowego projektu CodeSmith

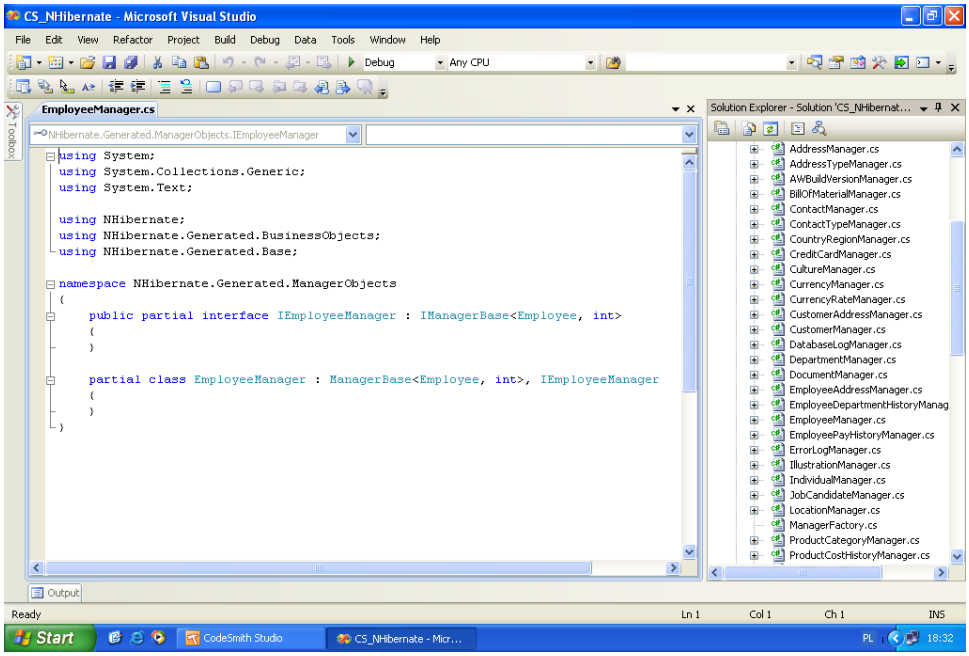


Rysunek 11. Interfejs wprowadzania parametrów generowania

W załączonym przykładzie wygenerowaliśmy zarówno pliki C# (*.cs) jak i XML (*.hbm.xml) (Rys. 12 i 13).



Rysunek 12. Przykładowo wygenerowany zestaw plików (1)



Rysunek 13. . Przykładowo wygenerowany zestaw plików (2)

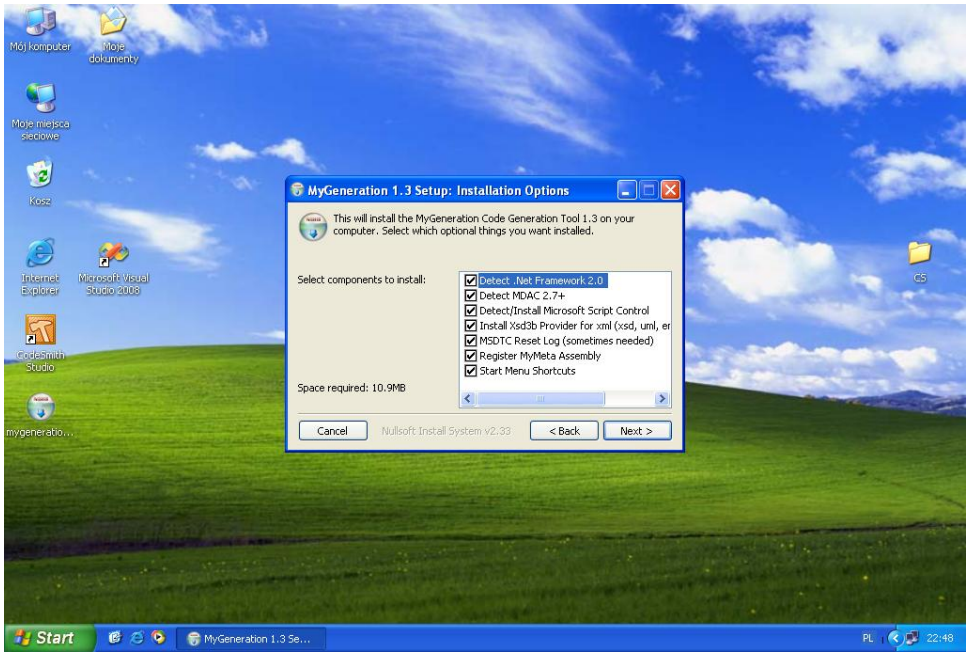
Warto zwrócić również uwagę na dobrą dokumentację CodeSmith. Na stronie internetowej możemy znaleźć rozbudowaną dokumentację składającą się z tutoriali, quickstartów, filmów oraz udokumentowanego API. Ponadto do poszczególnych templatów dołączony jest plik README ze zrzutami ekranów, który wyjaśnia krok po kroku jak skorzystać z tego szablonu.

2.2. MyGeneration

MyGeneration [2] jest darmowym produktem. Licencja pozwala na wykorzystywanie narzędzia do pracy z dowolnymi projektami, włączając w to projekty komercyjne. Produkt ten był kiedyś narzędziem płatnym, jednak w 2005 roku właściciele MyGeneration postanowili udostępnić go szerszej rzeszy odbiorców za darmo, na tzw. licencji freeware. Twórcy tej aplikacji, na swojej stronie WWW, podkreślają, że potencjalny użytkownik dostaje gratis program o wartości 199$. Aplikacja ta pozwala na podstawie szablonów dostarczonych z programem oraz metadanych pobieranych ze wskazanej bazy danych generować kod dla dowolnej platformy. MyGeneration nie jest związany z żadnym producentem oprogramowania. Aplikacja nie integruje się w żaden sposób z popularnymi na rynku IDE.

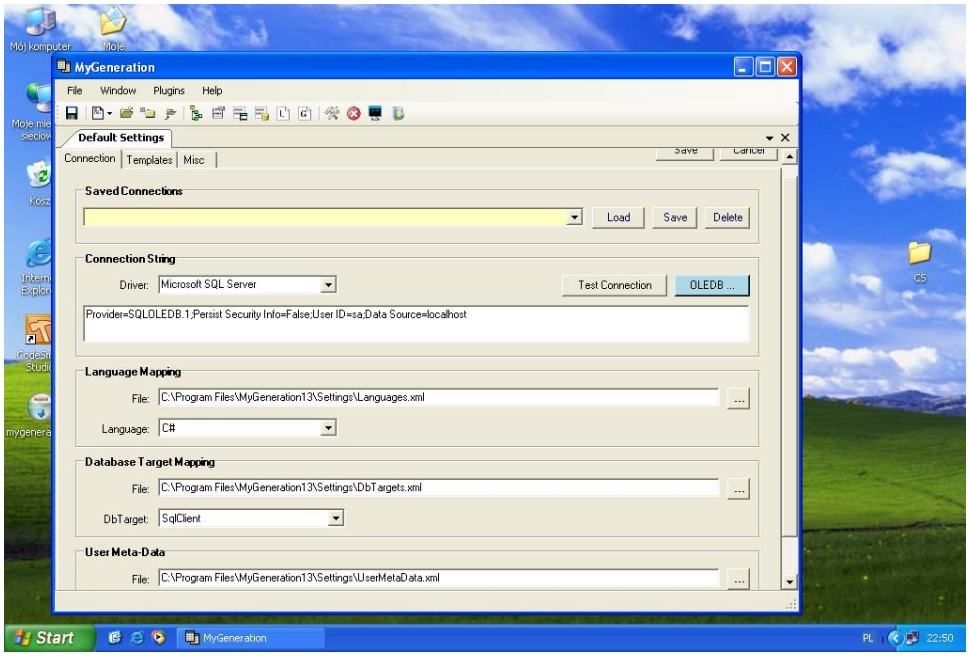
Instalacja MyGeneration jest prostym bardzo prostym procesem, spełniającym wymóg intuicyjnej obsługi. Wystarczy tylko pobrać ze strony instalator w formie pliku *.exe, uruchomić, a nastepnie przejść standardowy wizard instalacyjny. Na jednym z ekranów (Rys. 14) mamy do wyboru

opcje: Detect .Net Framework 2.0, Detect MDAC 2.7+, Detect/Install Microsoft Script Control, Install Xsd3b Provider for xml, MSDTC Reset Log, Register MyMeta Assembly, Start Menu Shortcuts. Można odznaczyć pojedyncze opcje, jednak każdy z tych komponentów jest wymagany do poprawnego działania programu. Wydaje mi się, iż ta funkcjonalność instalatora może przydać się bardziej przy naprawianiu lub upgradowaniu programu.

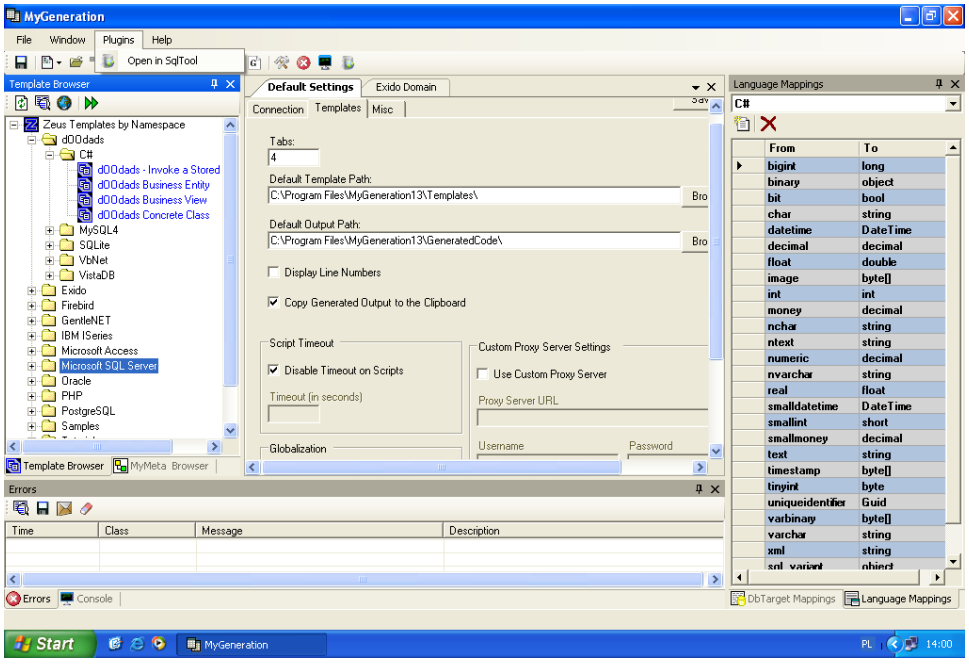


Rysunek 14. Instalacja MyGeneration

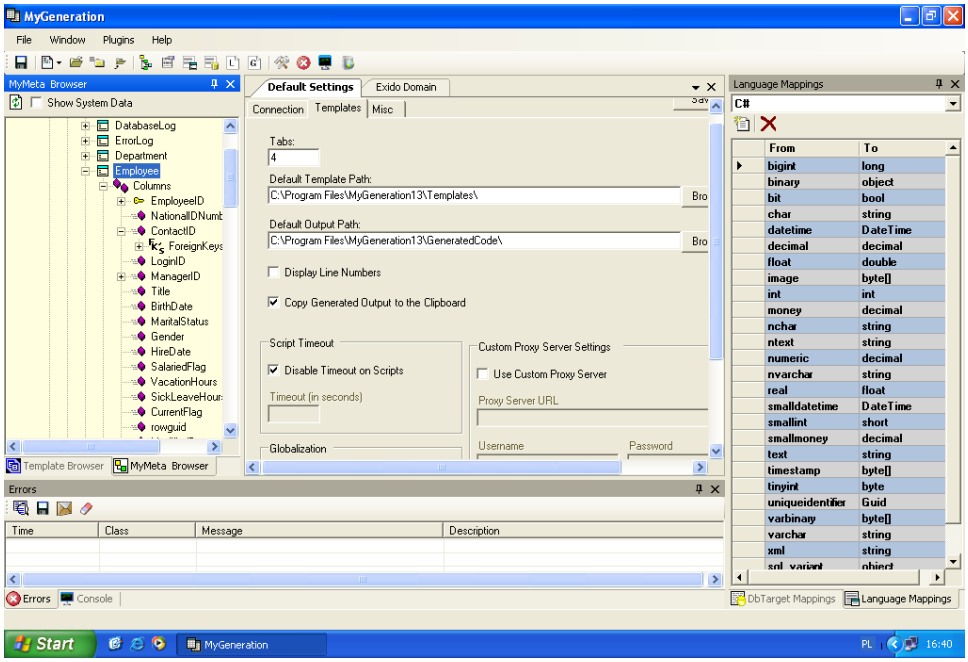
Po uruchomieniu aplikacji pokazuje się główne okno konfiguracyjne (Rys. 15). W tym miejscu możemy ustawić połączenie do bazy danych oraz pliki konfiguracyjne określające w jaki sposób mapowane są typy danych z bazy danych do typów zmiennych .NET Framework. Takie funkcjonalności, jak układ okien, możliwość przesuwania, dokowania okien przypominają starsze wersje Microsoft Visual Studio. Standardowo po lewej stronie znajdują się okna Template Browser (Rys. 16) oraz MyMeta Browser (Rys. 17). W oknie Template Browser znajdziemy repozytorium templatów gotowych do wykorzystania, natomiast MyMeta Browser tak jaka sama nazwa wskazuje, służy do podglądu metadanych bazy danych, która została podłączona we wcześniejszym kroku. Możemy tutaj podejrzeć strukturę bazy danych: tabelę, widoki oraz procedury składowane, wraz z ich kolumnami czy parametrami (w przypadku procedur składowanych). Po prawej stronie znajdują się okna Lanuage Mappings oraz DbTarget Mappings. Można w nich zmieniać sposób konwersji ze z typów bazodanowych na typy zmiennych używane w danym języku programowania (np. C#). W dolnej części aplikacji znajduje się okno Errors, w którym pokazywane są błędy podczas kompilacji templatu oraz okno Console zawierające podgląd na kolejno wykonywane akcje.



Rysunek 15. Ustawienia programu



Rysunek 16. Template Browser

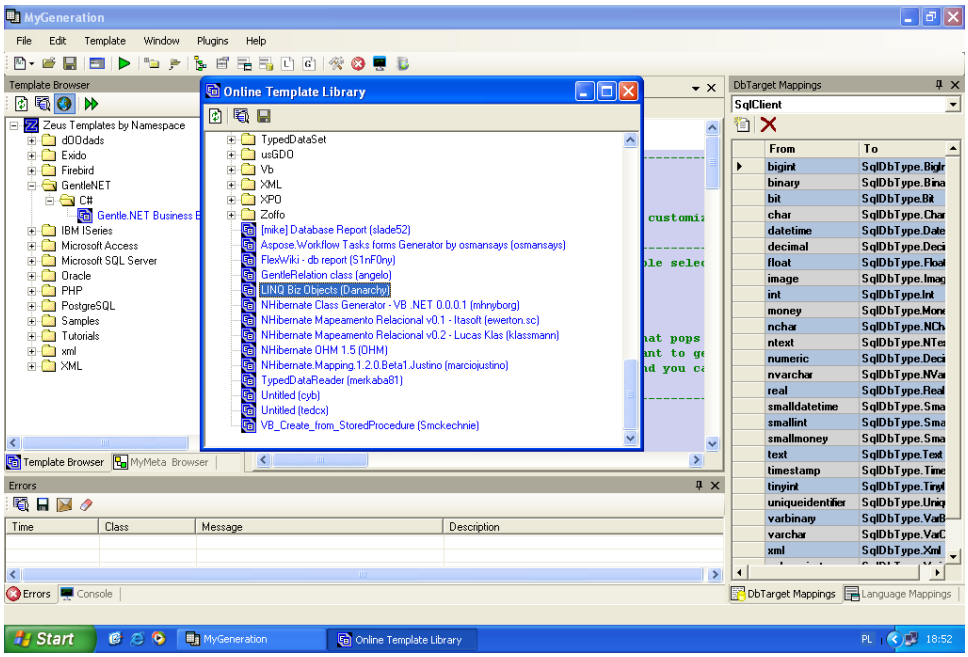


Rysunek 17. MyMeta Browser

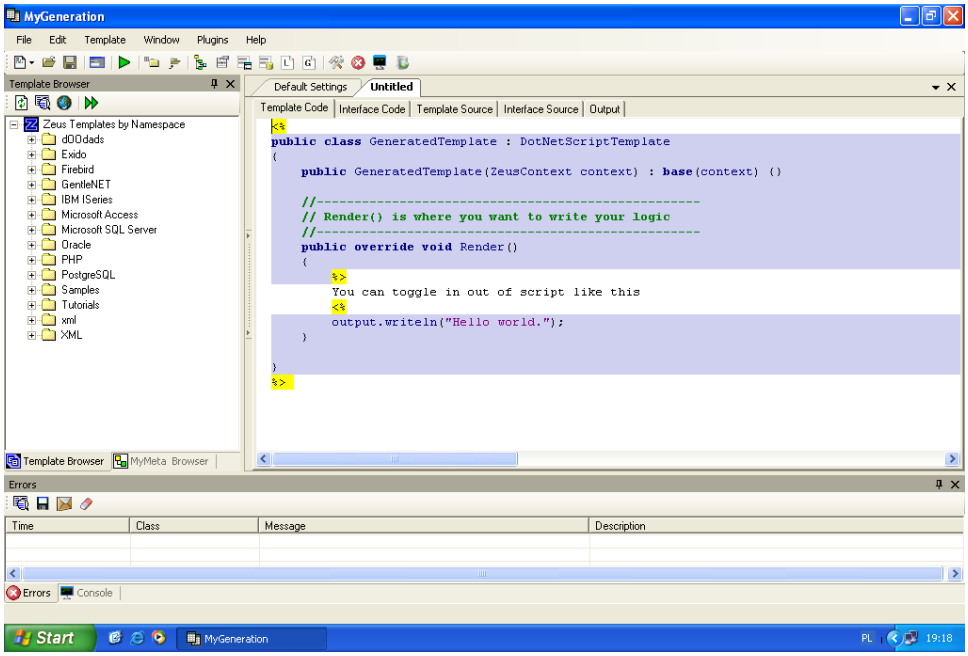
MyGeneration jest dostarczane z zestawem gotowych templatów. Można je przeglądać korzytajac z okienka Template Browser. Szablony są pogrupowane według tematyki. Templaty znajdujące się w węzłach oznaczonych nazwą bazy danych (np. PostgreSQL, Oracle) generują kod związany z obsługa tych baz, czyli są to najczęściej jakieś procedury składowane, natomiast węzły zawierające w sobie nazwę jakiegoś języka (np. C#) zawierają szablony tworzące kod w danym języku. Niestety, nie jest to rozwiązanie przejrzyste i utrudnia zrozumienie działania MyGeneration początkujących użytkownikom. Jeżeli w podstawowym zestawie templatów nie znajdziemy tego czego szukaliśmy, mamy jeszcze możliwość podłączenia się do onlinowej bazy MyGeneration i pobrania stamtąd określonego szablonu i zapisania go na nasz dysk (Rys. 18). MyGeneration ma dosyć silną społeczność i wiele osób udostępnia tam nieodpłatnie swoją pracę.

Jeżeli nie znajdziemy odpowiedniego szablonu, możemy sami utworzyć nowy wzorzec według własnych potrzeb. Osoba tworząca nowy template ma do dyspozycji 4 języki: C#, Visual Basic.NET, JScript, VBScript. Okno edytora szablonu składa się z 5 zakładek. W dwóch pierwszych można pisać kod, natomiast 3 pozostałe służą tylko do podglądu. W zakładce Template Code (Rys. 19) tworzymy szablon, który będzie przetwarzany i parsowany razem z metadanymi. Możemy tutaj umieścić zarówno kod statyczny jak i dynamiczny. Kod dynamiczny umieszczamy wewnątrz wyrażenia <% oraz %>, co budzi naturalne skojarzenia z ASP.NET. Kod do automatycznego generowania powinien być umieszczony wewnątrz utworzonej już metody Render. Jak widać na rysunku, możemy tam umieszczać zarówno treści statyczne jak i dynamiczne, a także dodawać pomocniczo własne metody

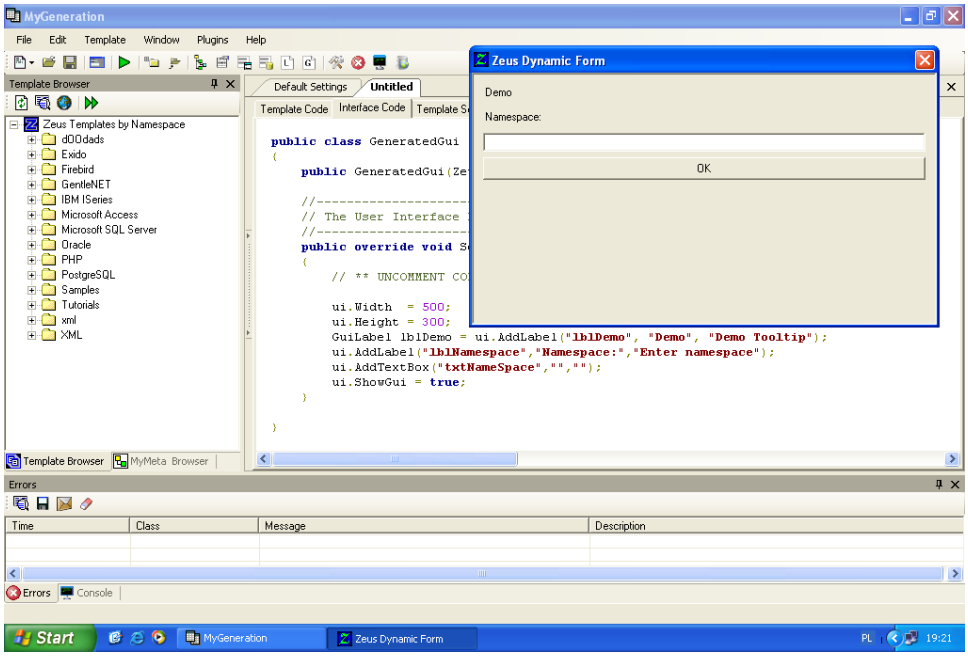
wewnątrz klasy, aby ułatwić sobie generowanie kodu. Cały kod musi się znajdować w jednym pliku, nie ma możliwości przerzucania części akcji do tzw. pliku code-behind. W drugim oknie Interface Code (Rys. 20) programista ma możliwość stworzenia interfejsu użytkownika do pobierania parametrów generowania. Można tutaj dodać pole tekstowe, pole pojedynczego lub wielokrotnego wyboru, pola checkbox lub radio. Dzięki temu osoba generująca kod ma możliwość wprowadzenia własnych ustawień, jak na przykład nazwa przestrzeni nazw oraz wybrane tabele, na podstawie których ma być generowany kod. Template Source i Interface Source są podglądem na kod wygenerowany na podstawie dwóch pierwszych okien. Ostatnie zrzut Output (Rys. 21) służy jako strumień wyjściowy z generatora. O ile w szablonie nie jest zaimplementowane zapisywanie do pliku, wynik generowanego kodu pojawia się właśnie w oknie Output. Templaty można zapisać jako pliki o rozszerzeniu *.zeus



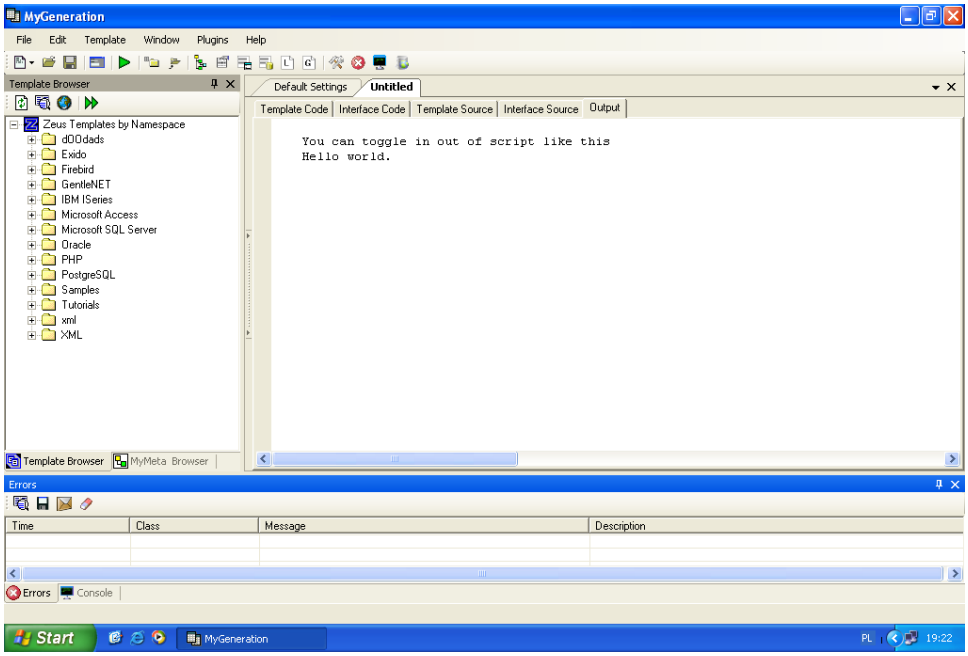
Rysunek 18. Onlinowa baza MyGeneration



Rysunek 19. Zakładka Template Code

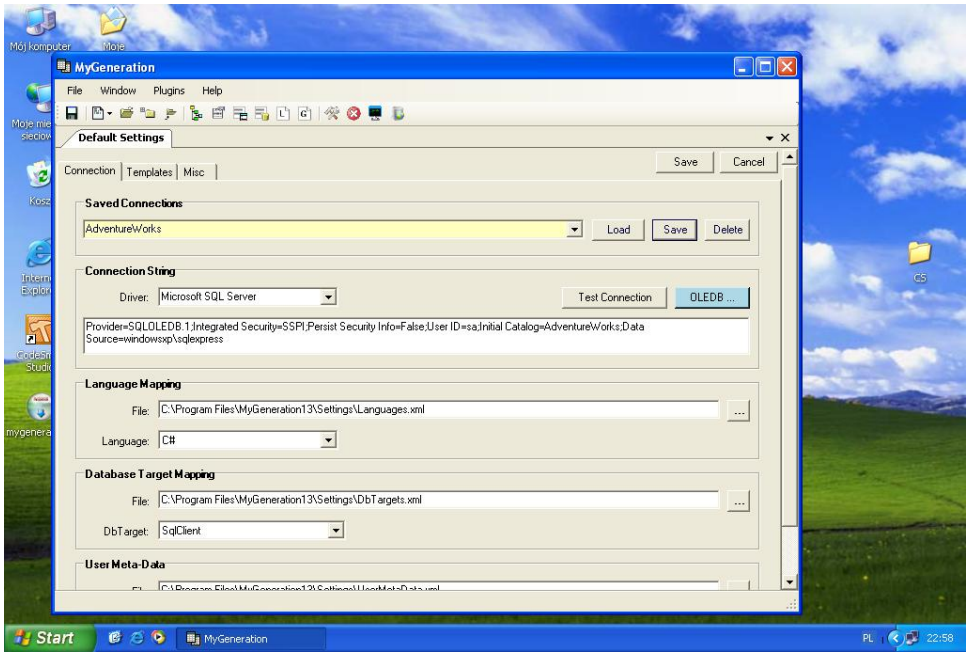


Rysunek 20. Zakładka Interface Code

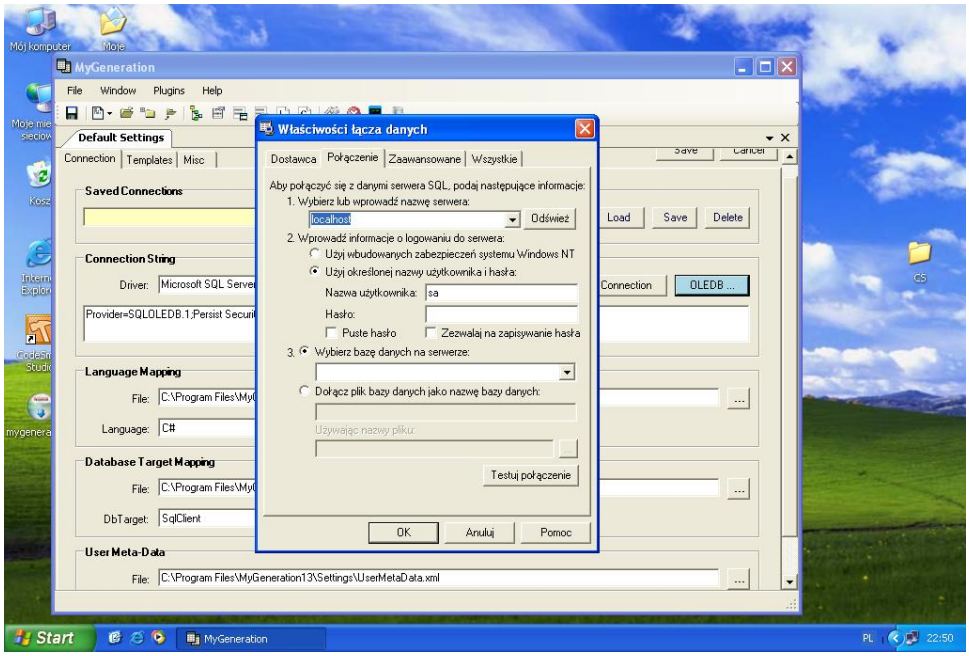


Rysunek 21. Przykładowy wynik działania programu

Bazę danych podłącza się poprzez okno ustawień Default Settigs (Rys. 22). W przypadku baz nie obsługiwanych prze OLEDB, czyli np., MySQL czy PostgreSQL, użytkownik musi wpisać connection string ręcznie. Dla baz danych obsługujących OLEDB możemy skorzystać z wizarda zaczerpniętego z Microsoft Visual Studio i wygenerować connection string automatycznie (Rys. 23). Jest to analogiczne rozwiązanie jak w przypadku pierwszego programu CodeSmith. Wpisane połączenie można zapisać w celu użycia go w przeszłości. Po uruchomieniu aplikacji nawiązuje ona połączenie z ostatnio używaną baza danych.



Rysunek 22. Ekran ustawień do wpisania connection string



Rysunek 23. Wizard dla baz OLEDB

Aktualnie MyGeneration wspiera aż 12 silników baz danych: Microsoft SQL, Oracle, IBM DB2, PostgreSQL, Microsoft Access, FireBird, Interbase, VistaDB, SQLite, MySQL, Advantage oraz Pervasive.

Standardowym wyjściem programu jest zakładka Output (Rys. 21). Zostanie na niej wygenerowany kod według zaimplementowanego algorytmu. Dodatkowo programista ma możliwość zapisania generowanego kodu do pliku lub kolekcji plików. Ponieważ trzeba to zrobić programistycznie, możemy podczas generowania stworzyć dowolną strukturę katalogów i używać aktualnej daty i czasu oraz nazw bazy danych, tabel przy nadawaniu nazw utworzonym plikom.

MyGeneration jest dobrym rozwiązaniem dla osób, które chcą za darmo dostać narzędzie do generowania kodu na podstawie struktury bazy danych. Dostajemy do dyspozycji narzędzie, dzięki któremu, poprzez automatyczne tworzenie kodu, mamy możliwość realnych oszczędności czasowych. Dokumentacja projektu pozostawia wiele do życzenia, ale jest to domena opensourceowych/darmowych programów. O ile udokumentowanie API jest na dosyć dobrym poziomie, o tyle brakuje dobrych tutoriali, quickstartów i filmów szkoleniowych.

Myślę, że właśnie w tym tkwi problem małej popularności MyGeneration. Śledząc fora internetowe możemy dojść do wniosku, iż jest wiele osób, które ma właśnie największe problemy z rozpoczęciem pracy z MyGeneration i zrozumieniem filozofii jego działania. Większość szybko się zraża i rezygnuje.

3. Generowanie kodu na podstawie meta danych

W tym rozdziale zostaną przytoczone argumenty za i przeciw takiemu podejściu do automatyzacji wytwarzania oprogramowania, zostaną przedstawione założenia budowy aplikacji trój warstwowych oraz zostaną przedstawione założenia odnośnie implementacji wykonanej w ramach pracy dyplomowej.

3.1. Dlaczego generowanie kodu?

Przytoczmy tutaj szereg argumentów, które znajdują się w książce „Code generation In Action", Jacka Herringtona [3]

Generatory pomagają przede wszystkim utrzymać odpowiednią jakość kodu. Kod tworzony przez nie jest spójny i ujednolicony. Dzięki temu, że nazwy klas, zmiennych i metod są sporządzane przez automat, przyjęte będą te same założenia względem nazewnictwa w całym projekcie. Generatory kodu pozwalają narzucić architekturę rozwiązania, nałożyć z góry ramy, w których porusza się programista. Należy pamiętać, że wszystko, co może zrobić maszyna/automat będzie lepiej zrobione niż miałby to robić człowiek mający swoje lepsze i gorsze dni. Z doświadczenia Autor wie, że przeważenie programiści wykazują albo zbyt dużą aktywność i próbują eksperymentować i szukać lepszych rozwiązań lub chcą sobie ułatwić życie idąc na skróty. Obydwa rozwiązania prowadzą do dużej rozbieżności jakości kodu, co przekłada się ujemnie na cały projekt. Dodatkowo niestabilność kodu może wprowadzić bardzo duże zakłócenia w harmonogramie projektu.

Generatory kodu pozwalają również tworzyć oprogramowanie zgodnie z wymogami TDD (Test Driver Development). Podczas wytwarzania kodu można od razu tworzyć testy jednostkowe, a później dowolnie je modyfikować, rozszerzać w zależności od tego, jak będzie rozbudowywany kod naszego projektu. Podobnie jak testy, można automatycznie tworzyć również dokumentacje produktu. Generator tworzy pliki dokumentacji opisujące klasy aplikacji, wyszczególniając metody, atrybuty, a rola dokumentalisty ogranicza się do wpisania w odpowiednie pola opisu zrozumiałego dla wykonawcy projektu. Uzyskujemy, więc nie tylko oszczędności na etapie tworzenia oprogramowania, ale także przy testowaniu oprogramowania i tworzeniu jego dokumentacji. Firmy, które rezygnowały z przyzwoitej dokumentacji i testów ze względu na koszty mogą ponownie rozważyć taką opcję.

Kolejna zaleta generatorów kodu pozwala na podniesienie morale zespołu. Ta teza może się wydawać śmieszna na pierwszy rzut oka, ale dzięki automatyzacji mniej skomplikowanych procesów pracownicy nie muszą wykonywać rutynowej, nużącej pracy. Zadowolenie z wykonywanego projektu

jest szczególnie ważne w dużych i długich projektach. Programiści, są to z reguły osoby lubiące wyzwania i niechętnie zajmują się monotonnym pisaniem kodu według z góry ustalonych zasad i reguł. Używając generatory kodu pracownicy skupiają się na naprawdę istotnych zadaniach i problemach, które są dla nich wyzwaniem i motywacją do dalszej pracy.

Z osobistych doświadczeń Autora wynika, iż spora część systemów informatycznych używanych w biznesie, tworzonych jest według pewnych założonych z góry schematów. Najczęściej z nich spotykanym jest CRUD (ang. Create/Read/Update/Delete). W tej konwencji użytkownik ma możliwość zarządzania obiektami jednej tabeli lub obiektami tej samej klasy w przypadku OOP. Możemy wyszczególnić tutaj następujące operacje: przeglądanie listy, filtrowanie, sortowanie rekordów, dodawanie nowego rekordu, edycja rekordu, podgląd rekordu (z wydrukiem) oraz usuwanie rekordów. Są to dosyć proste operacje niewymagające złożonych obliczeń, jednak dosyć pracochłonne, jeżeli chodzi o implementację.

Bardzo często moduły oprogramowania w CRUD znajdują się w systemach backofficowych, w panelach konfiguracyjnych. Często nie są używane na co dzień, a co za tym idzie budżety przeznaczone na zaprojektowanie, zaimplementowanie i w wdrożenie tego typu oprogramowanie są niskie.

Autor często spotyka się z potrzebą szybkiego generowania (ad-hoc) kolejnych modułów aplikacji lub paneli konfiguracyjnych bez potrzeby dużego zaangażowania czasowego, tak abym mógł się skupić na rzeczach unikalnych i ważnych w tym module aplikacji.

Narzędzie do stworzenia kodu powinno być na tyle elastyczne, aby móc generować kod w różnych językach, w dowolnej notacji. Kod powinien być generowany na podstawie szablonów wykonanych przez użytkownika generatora lub wyznaczoną do tego osobę w zespole, a także powinien być tworzony w architekturze 3 warstwowej:

- Warstwy dostępu do danych (Data Source Layer)
- Warstwy logiki biznesowej (Service Layer)
- Warstwy prezentacji (User Interface)

  oraz klas będących odwzorowaniem encji w bazie danych (Domain model).

Poszczególne warstwy zostaną opisane w dalszej części tego rozdziału.

3.2. Alternatywy dla generowania kodu

Alternatywą dla opisanych wcześniej generatorów kodu jest tworzenie programów, które działają na zasadzie metadanych, na przykład - przy wyświetlaniu formularza jest robiona refleksja na obiekt i na podstawie jego pól/atrybutów i ich typów tworzona jest formatka[4]. Ma to oczywiście

swoje dobre i złe strony. Jeżeli postanowimy jakoś inaczej odzwierciedlić typ (np. chcemy wydłużyć pole tekstowe i zmienić jego kolor), to wprowadzanie jeden raz tej zmiany spowoduje zmianę wyglądu w już wytworzonych modułach, jak i w tych, które dopiero powstaną. Z drugiej strony, w przypadku generatorów „statycznego kodu", narzędzia typu Microsoft Visual Studio .NET czy Eclipse maja bardzo dobre narzędzia do refaktoringu kodu, dzięki czemu wprowadzenie tej zmiany w już wygenerowanych modułach nie powinno stanowić problemu. Przy takim podejściu mamy również możliwość nie wprowadzania takiej zmiany wstecz, tylko generowanie nowych modułów z ową zmianą.

Należy zaznaczyć, że kod generowany jest o wiele bardziej wydajny niż programy działające na podstawie metadanych. Czytanie metadanych, czyli tzw. refleksja, jest bardzo kosztownym procesem i należy go unikać, jeżeli zależy nam na jak największej wydajności. Przykładem może być prosta strona profilu użytkownika w bardzo popularnym portalu internetowym. Jeżeli chcielibyśmy utworzyć formatkę z danymi użytkownika na podstawie metadanych trzeba by było oprócz zapytania bazy danych o dane użytkownika, wykonać iterację po polach (kolumnach) zwróconego zapytania i zinterpretować każdy ze zwracanych typów. Przy pojedynczych żądaniach HTTP różnica jest pewnie niezauważalna, ale przy obciążeniu kilkuset żądań HTTP na sekundę, takie rozwiązanie mogłoby się nie sprawdzić.

Rozwiązania działające dynamicznie na podstawie metadanych są o wiele trudniejsze do przetestowania. Są gotowe metodyki i narzędzia do testowania oprogramowania, jednak w przypadku programów generujących „ w locie" inne programy, testowanie jednostkowe byłoby bardzo trudne, jeżeli w ogóle możliwe.

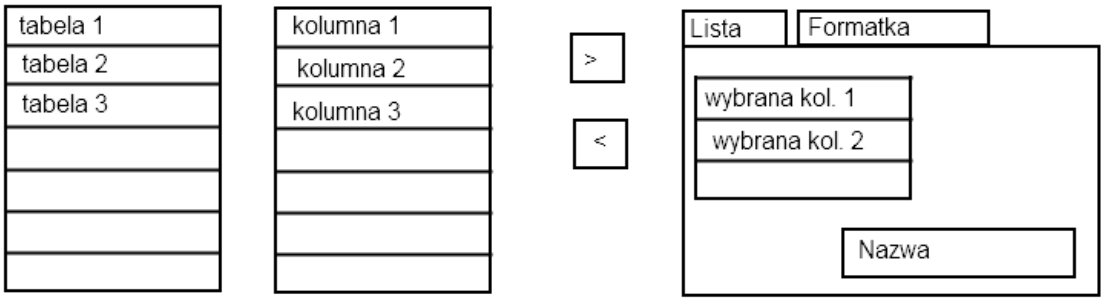
Klienci lubią wyjątki. Niestety, dla programistów każda firma jest inna. Jest to istota wolnego rynku. Firmy budują swój kapitał na wyprzedzaniu konkurencji, byciu bardziej innowacyjnym. Z racji tego, iż większość firm wykorzystuje różnego rodzaju systemy do wspomagania zarządzania przedsiębiorstwem, systemy te musza być dopasowane ściśle do procesów przebiegających na jego terenie. Niestety, trudno jest wymyślić uniwersalny system, który zadowoli każdego z klientów. Generowanie kodu ma tą przewagę nad programowaniem z wykorzystaniem metadanych, że możemy rezultat dowolnie modyfikować, rozszerzać lub okrajać. O wiele łatwiej jest wprowadzić różnego rodzaju wyjątki od reguł do wygenerowanego kodu, niż do jakiegoś uniwersalnego silnika, który po paru takich zmianach robi się kompletnie nieczytelny i staje się wolniejszy.

3.3. Czym powinien się charakteryzować projektowany generator

Na tym etapie należałoby określić, czym ma się charakteryzować nasz generator. Jako narzędzie wspierające pracę programisty powinien pomagać, a nie utrudniać codzienną pracę. Teza jest z pozoru oczywista, jednak przyglądając się konkurencyjnym rozwiązaniom, rzeczywistość wygląda nieco odmiennie. Producenci Codesmith[1] Czy MyGenration[2] wymagają od użytkowników ich programów dogłębnego zapoznania się z ich aplikacjami, przed wygenerowaniem najprostszego przykładu typu Hello World. Nie ma problemu, jeżeli producent korzysta ze znanego języka, na przykład C# i nauka ogranicza się jedynie do poznania nowego API. Pamiętajmy jednak, że głównym założeniem wdrożenia takiego narzędzia miało być zminimalizowanie czasu. Niestety, praca przy dużych projektach to nieustanna gonitwa z rzeczywistością i bardzo często nie można poświęcić nawet kilku dni na wdrożenie takiego rozwiązania.

Głównymi założeniami powinna być więc prostota, a aplikacja powinna dać możliwość szybkiego wystartowania z generacją. Nasze narzędzie nie ma rozwiązywać wszystkich problemów informatycznych, lecz odciążyć programistę od najbardziej banalnych zadań. Aby narzędzie było łatwo i szybko wdrożyć do działania, musi się ono opierać na powszechnie panujących standardach. Biorąc pod uwagę mechanizmy łączenia danych z szablonami, to takim standardem jest na pewno XML/XSL. Dowodem na to jest fakt, że każda licząca się platforma czy język programowania ma zaimplementowana obsługę transformacji. Szukając przykładów w literaturze możemy odnieść się do książki Kathleen Dollard „Code Generation in Microsoft .NET”[5], w której autorka przedstawiła przykład opierający się na transformacji XSLT w celu automatycznego wytwarzania oprogramowania.

Ponieważ ciężko by było przygotować generator do wszystkich możliwych scenariuszy pracy z danymi skupimy się tutaj na najpopularniejszym modelu CRUD. Obejmuje on operacje dodawania, czytania, modyfikacji i usuwania wybranych rekordów. Interfejs użytkownika będzie więc obejmował dwa ekrany: listę rekordów oraz formatkę danych.



Rysunek 24. Projekt interfejsu do projektowania modułu

Program musi umożliwiać wprowadzanie parametrów połączenia do bazy źródłowej w celu zaczytania jej struktury. Na rysunku Rys. 24 przedstawiono projekt interfejsu. Informacje wyświetlamy w taki sposób, że użytkownik może wybrać tabelę, i dla dostępnych kolumn może wybrać te, które chce widzieć na liście rekordów lub samej formatce

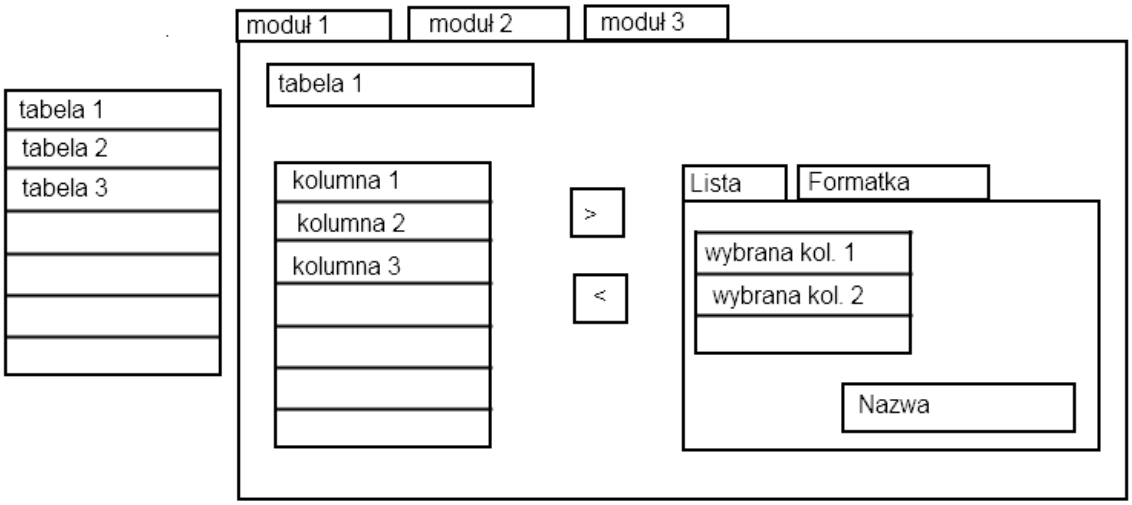
Musimy wprowadzić podział pomiędzy listę rekordów, a formatkę kodu, ponieważ w wielu przypadkach będą wyświetlane inne kolumny na liście, a inne na samej formatce. Programista powinien mieć możliwość zmiany dla poszczególnych pól i kolumn wartości labelek z domyślnych odpowiadających nazwie kolumny tabeli na własne. Aby uprościć trochę rozwiązanie przyjmiemy, że pola będą przyjmowały typ w zależności od rodzaju źródłowej kolumny, W poniższej tabelce przedstawiono sposób transformacji

Typ SQL	**Typ C#**	**Kontrolka ASP.NET**
narchar, nvarchar	string	<asp:textbox />
char,char	string	<asp:textbox />
int,bigint, smallin	int	<asp:textbox />
Datetime, smalldatetime	DateTime	<asp:calendar />
Float	double	<asp:textbox />
Bit	bool	<asp:checkbox />
pozostałe typy	string	<asp:textbox />

Jeżeli pole jest kluczem obcym innej tabeli należałoby wyświetlić listę rozwijaną z możliwością wyboru konkretnego rekordu. Do rozwiązania pozostaje kwestia, jakie informacje powinny być wyświetlane. Mogą to być na przykład trzy pierwsze pola tekstowe

Aby uatrakcyjnić rozwiązanie możemy dodać mechanizmy sortowania i wyszukiwania tak. Należałoby dodać dwa pola wyboru (checkbox) przy każdej z wybranych kolumn, określające czy pole to ma być brane pod uwagę przy wyszukiwaniu oraz czy umożliwić sortowanie dla tego pola. W przypadku formatki można by było dodać umożliwić wybór, czy pole ma być edytowalne, czy tylko do odczytu. Przydatną funkcją byłaby opcja zmiany kolejności pól na liście lub formatce.

Aplikacja powinna mieć możliwość zapisywania wielu takich scenariuszy w jeden projekt. Odbywało by się to przez dodawanie poszczególnych pół do zakładek - tabel, gdzie z kolei znajdował by się wcześniej opisany panel konfiguracyjny dla modułu (Rys.25).



Rysunek 25. Projekt interfejsu do projektowania modułu -wersja rozszerzona

Warto by było również wprowadzić możliwość wyboru templatów XSL. Na tym etapie będzie umożliwiało zaimplementowanie różnych wersji scenariusza CRUD. Można tutaj przytoczyć przykład strony ASP.NET, która nie będzie korzystała ze standardowych kontrolek dostarczonych przez Visual Studio, tylko z alternatywnych dostawców takich jak Telerik i DevExpress. W przyszłości ten mechanizm może być wykorzystany do przełączania pomiędzy odmiennymi technologiami jak Java czy PHP.

Nasza aplikacja powinna posiadać mechanizm standardowego zapisywania ustawień i ich późniejszej modyfikacji. Dzięki takiemu rozwiązaniu, jeżeli będziemy chcieli dodać dodatkową

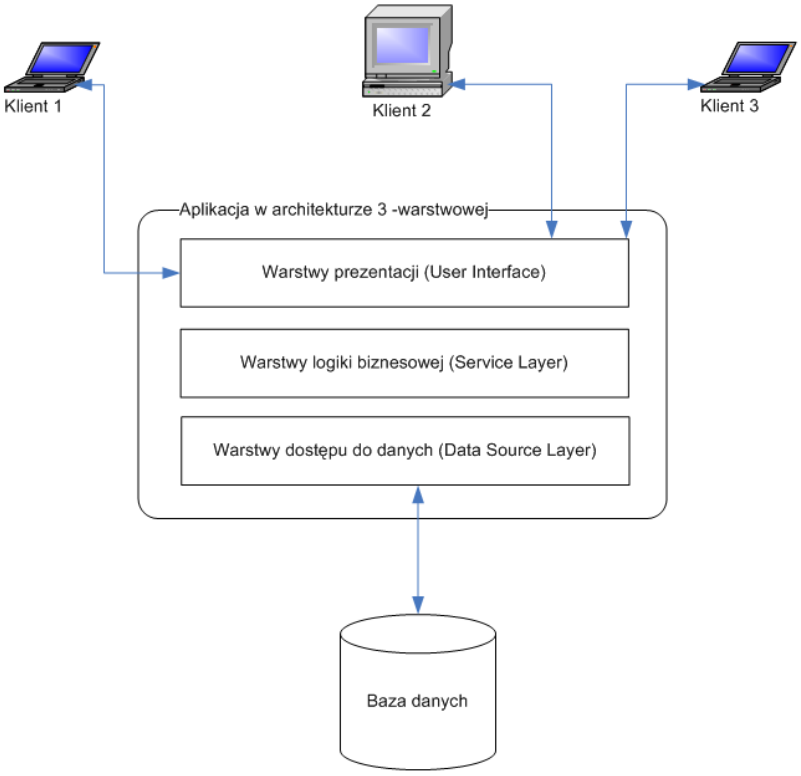
kolumnę do tworzonego modułu, nie musimy definiować wszystkiego od początku. Zostanie utworzony w tym celu plik *.cgn. Rozszerzenie to powinno zostać skojarzone z programem, aby domyślną operacją (dwukilk) było otworzenie nowej instancji aplikacji.

Program powinien mieć dwie funkcje generowanie kodu, które można załączyć do już istniejącego projektu oraz generowanie działającej niezależnie aplikacji. Druga operacja wykorzystywałaby mechanizmy pierwszej, ale dodatkowo kopiowałaby poszczególne pliki do struktury aplikacji i uruchamiała proces kompilacji. Ciekawą możliwością może być opcjonalne załączanie pliku własnego pliku CSS, co pozwoliłoby w prosty sposób customizować wytworzone oprogramowanie.

Do każdego modułu powinien zostać wygenerowany kod logiki biznesowej, oraz kod dostępu do danych. Zagadnienia te zostaną omówione w następnym podrozdziale.

3.4. Budowa wielowarstwowych aplikacji biznesowych

System informatyczny powinien być skalowalny i mieć możliwość łatwej rozbudowy wraz z powiększaniem się firmy, wzrostem liczby klientów oraz przeprowadzanych w systemie transakcji. Skalowalność oznacza, że w przypadku zwiększania ilości przetwarzanych danych, należy jedynie zwiększyć liczbę procesorów, rozmiar pamięci RAM oraz wielkości dysków. Nie będzie natomiast potrzeby modyfikacji systemu (kodu).



Rysunek 26. Architektura 3-warstwowa

Równie istotna jest taka implementacja systemu, aby można było szybko i łatwo wprowadzać jego modyfikacje, dodatkowo nie komplikując go podczas kolejnych zmian. Jest to szczególnie ważne w naszym kraju, gdzie bardzo często modyfikowane jest prawo oraz system podatkowy. Nie bez znaczenia jest również fakt, że życie produktów jest coraz krótsze, a tym samym trzeba coraz częściej dopasowywać systemy sprzedaży do nowych produktów, usług oraz promocji.

W odpowiedzi na powyższe wymagania powstał warstwowy model budowy aplikacji. Ostatnio najczęściej spotykaną architektura jest architektura 3 – warstwowa. Możemy w niej wyodrębnić warstwę dostępu do danych, warstwę logiki biznesowej oraz warstwę prezentacji (Rys. 26). W przeciwieństwie do poprzedniego modelu 2 - warstwowego (klient-serwer), logika biznesowa jest umieszczona na osobnym poziomie, dzięki czemu aplikacje są bardziej elastyczne. Jeżeli zmienimy dostawcę baz danych, wystarczy zmodyfikować warstwę dostępu do danych, natomiast jeżeli będziemy chcieli zamienić interfejs GUI na WWW wystarczy podmienić tylko tą dotyczącą prezentacji danych.

Możemy również zwiększyć skalowalność naszego rozwiązania umieszczając każdą z warstw na osobnych serwerach. Dodatkowo możemy tworzyć farmy serwerów obsługujące równolegle operacje użytkowników. Każdy z serwerów powinien być ulokowany w osobnej strefie DMZ

oddzielonej firewallem. Dzięki temu włamanie do systemu z poziomu aplikacji może być niemal niemożliwe.

3.4.1 Warstwa dostępu do danych (Data Source Layer)

Data Source Layer odpowiedzialna jest za pobieranie i zapisywanie danych do stałego źródła, jakim jest najczęściej baza danych. Zadaniem warstwy dostępu do danych jest otwieranie zamykanie połączeń, obsługa transakcji oraz obsługa wyjątków. Powinna być ona zaimplementowana w taki sposób, aby w przypadku zmiany bazy danych (producent, typ, wersja), wszelkie modyfikacje można było wykonać jedynie dokładnie w tej warstwie. Programista musi poświęcić szczególną uwagę na problem SQL injection. Atak ten umożliwia przekazanie do zapytania, tak skonstruowanego ciągu znaków, który pozwoli na wykonanie dowolnego zapytania, w tym np. skasuje pewne rekordy. Większość producentów oprogramowania udostępnia najlepsze praktyki, jak należy zabezpieczyć się przed tego typu zdarzeniami.

Ze względu na sposób zaimplementowania warstwy dostępu do danych możemy wyróżnić:

- Kod wywołujący procedury składowane
- Kod uruchamiający zapytania SQL
- Biblioteki ORM.

Data Source Layer oparta na procedurach składowanych jest obecnie uważana za najbezpieczniejsze rozwiązanie problemu dostępu do danych. Procedury są wywoływane przy użyciu standardowych bibliotek ADO.NET, JDBC, ODBC. Dane przekazywane są do procedury w postaci parametrów a informacje są czytane również w takiej postaci. Jeżeli mamy do wczytania listę rekordów możemy posłużyć się kursorem. W zależności od producenta oprogramowania trzeba się posłużyć jawnie np. kursorem referencyjnym (PL\SQL), lub dzieje się to w „tle” niezauważalne dla programisty (T-SQL)

Niestety, procedury składowane mają również swoje wady. Są one ściśle związane z producentem bazy danych (Microsoft T-SQL, Oracle PL/SQL), a co za tym idzie nie ma możliwości ich migracji. Pojawia się także problem podczas uruchamiania różnych wersji oprogramowania, gdy wymagane są odmienne typy procedury o tej samej nazwie, ale zmodyfikowanej funkcjonalności i liczbie parametrów.

Coraz częściej spotyka się lżejsze podejścia do tematu warstwy dostępu do danych, czyli zaszywanie zapytań SQL w aplikacji. W bazie Microsoft SQL 2005 zapytania uruchomione z poziomu procedury składowanej i jako normalne zapytanie umieszczone w aplikacji mają zbliżony czas wykonania. Dzieje się tak dlatego, że również „zwykłe” zapytania są kompilowane i przetrzymywane w cache -u. Zapytania takie powinny być wywoływane ze względu na bezpieczeństwo (SQL injection) oraz wydajność (cache), jako zapytania tzw. sparametryzowane. Oznacza to, że zapytanie nie jest

budowane dynamicznie poprzez złączanie łańcuchów znaków, ale że w nim na stale umieszczony jest parametr, który ustawiany jest niezależnie od zapytania, a całość łączona jest dopiero przez serwer SQL.

W przypadku implementacji to podejście jest zbliżone do wywoływania procedury składowanej, należy jedynie podmienić wywołanie procedury na sparametryzowane zapytanie SQL. Ustawianie wartości parametrów oraz czytanie danych z wywołanego zapytania pozostaje takie samo.

Bardzo modnym ostatnio rozwiązaniem są - ORM (Object-Relational Mapping). Tego typu technologie same w sobie stanowią warstwę dostępu do danych i można się poprzez nią odwoływać do bazy danych, bez konieczności implementacji własnej warstwy. Są one również odpowiedzią na problem występowania dwóch światów - obiektowego, w którym pisane jest dzisiaj oprogramowanie, oraz świata relacyjnego, w którym są pobierane i zapisywane dane. Mimo prób popularyzacji obiektowych baz danych nie cieszą się one zbytnią popularnością. Aby skorzystać z ORM należy stworzyć klasę posiadającą pola, które będziemy chcieli odczytać (lub zapisać) z tabeli. Następnie należy stworzyć plik konfiguracyjny, w którym zmapujemy utworzoną klasę na tabelę w bazie danych oraz poszczególne pola na kolumny. Istnieją narzędzia pomagające w stworzeniu takiego pliku konfiguracyjnego.

Narzędzia ORM obsługują wiele scenariuszy dostępu do danych w tym połączenia (many –one, one -many, many – many) oraz np. pola tylko do odczytu.

Należy pamiętać, że narzędzia tego typu są wydajne jedynie przy pracy z pojedynczymi obiektami. Jeżeli chcemy zmodyfikować wartość kolumny w większej liczbie rekordów, lepiej zrobić to przy pomocy jednego polecenia SQL.

3.4.2 Warstwa logiki biznesowej (Service Layer)

W warstwie tej umieszczona jest implementacja reguł biznesowych, czyli określa ona funkcjonalność i sposób działania aplikacji.

Jej obiekty wywołują obiekty warstwy dostępu do danych, a same z kolei są wywoływane przez warstwę prezentacji. Stałymi elementami powinny być obsługa logowania wydarzeń w aplikacji, autoryzacja dostępu użytkowników do zasobów aplikacji oraz obsługa wyjątków.

3.4.3 Warstwa prezentacji danych (User Interface)

Warstwa prezentacji jest to interfejs użytkownika, której zadaniem jest prezentacja danych użytkownikowi systemu, umożliwianie wprowadzania danych oraz sprawdzanie ich poprawności. Warstwa prezentacji może być jako gruby klient, czyli aplikacja GUI, lub jako cienki klient, czyli aplikacją www. W ostatnich latach modne stały się aplikacje biznesowe oparte o cienkiego klienta.

Powstało dużo nowych bibliotek do tworzenia takich aplikacji. Najpopularniejszymi są ASP.NET wchodzące w skład Microsoft .NET oraz zaproponowane przez SUN –a JSF (JSR 127), które jest standardem i doczekało się Open Source –owej implementacji Apache MyFaces. Producenci oprogramowania udostępniają już rozbudowane IDE umożliwiające graficzne budowanie stron WWW, bez potrzeby ręcznego tworzenia kodu WWW. Dzięki temu aplikacje webowe buduje się obecnie o wiele szybciej i wygodniej, niż miało to miejsce kilka lat temu.

Ostatnio promowana jest technologia AJAX. Jest to połączenie JavaScript, XML –a i Web Services. Umożliwia ona interakcję tylko wybranych fragmentów strony www, bez potrzeby przeładowywania i pobierania wszystkich danych na stronie.

4. Opis narzędzi zastosowanych w pracy

Podczas pisania mojej pracy dyplomowej posłużyłem się szeregiem użytecznych narzędzi, które chciałbym przedstawić w kolejnych podrozdziałach.

4.1. Microsoft Visual Studio

Microsoft Visual Studio jest jednym z najpopularniejszych środowisk programistycznych IDE. Na pewno muszą je znać osoby tworzące aplikacje opierające się na rozwiązaniach Microsoftu. Microsoft Visual Studio występuje w 4 różnych wersjach:

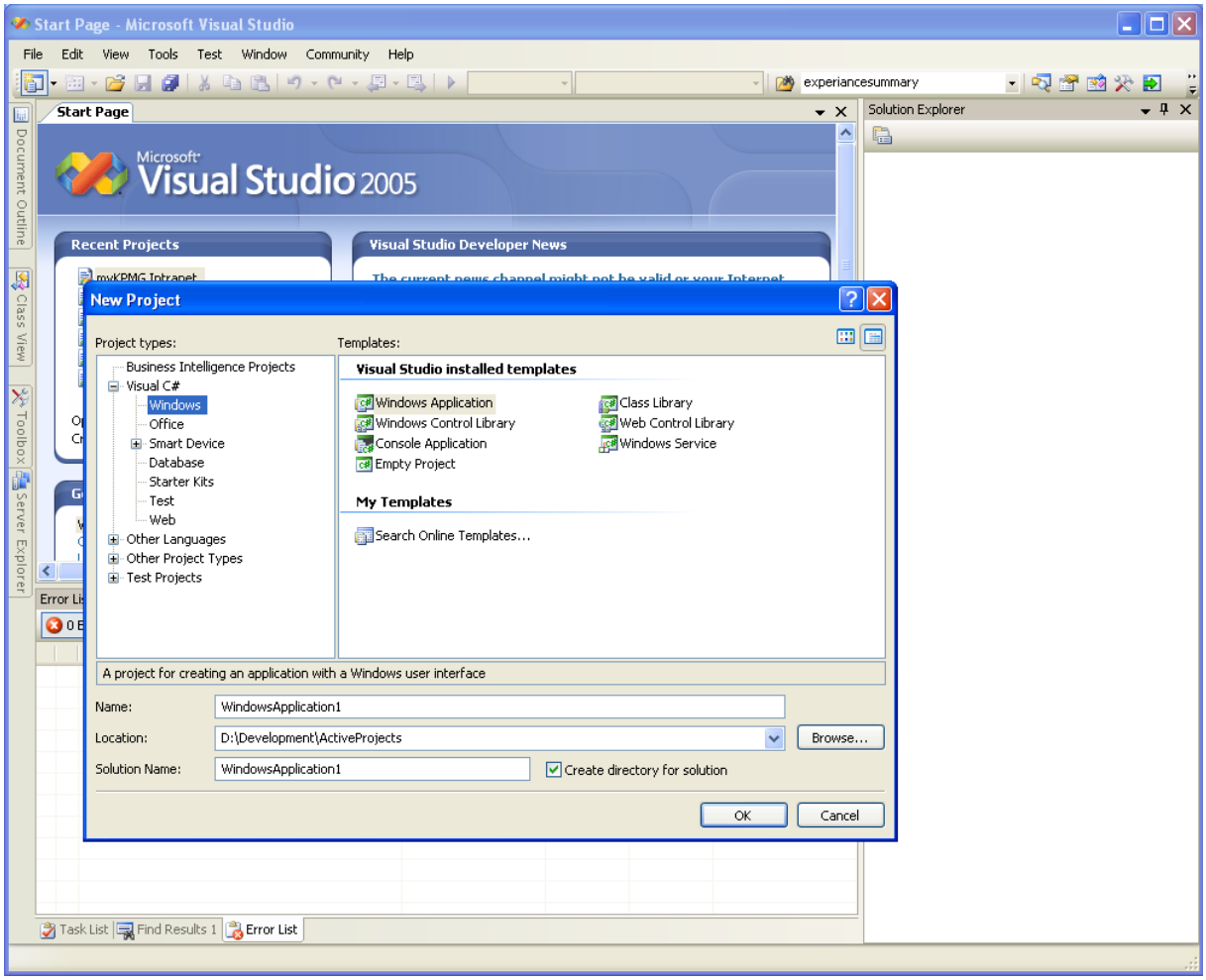
- Team Edition,
- Professional,
- Standard,
- Express Edition.

Wersja Microsoft Visual Studio Team System została specjalnie stworzona dla dużych zespołów programistycznych. Składa się z całego ekosystemu do wytwarzania oprogramowania. Znajdują się tam narzędzia do zbierania wymogów funkcjonalnych, projektowania aplikacji, tworzenia i na koniec do jej testowania (Rys. 27). Dla wszystkich uczestników procesu tworzenia, udostępniony został Team Foundation Server, który służy, jako repozytorium kodu, system do zarządzania przydziałem prac i monitorowania ich postępów. Jest również możliwość tworzenia stron WWW dla poszczególnych zespołów przy użyciu technologii portalowej Windows SharePoint Services (WSS) oraz raportów przy użyciu SQL Server Reporting Services.

Rysunek 27 Team Foundation Server

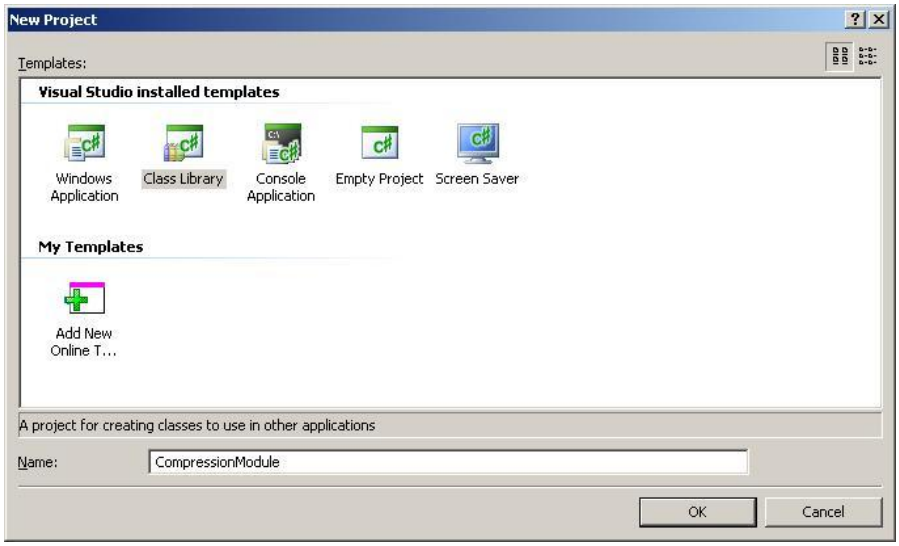
Wersje Professional (Rys. 28) oraz Standard służą pracującym samodzielnie programistom lub bardzo małym zespołom do wytwarzania kodu. Wersja Standard jest opcją uboższą w funkcjonalność, a co się z tym wiąże, również tańszą. Ma ona również bardziej uproszczony interfejs użytkownika. Z kluczowych różnic pomiędzy tymi dwoma wersjami należy wymieć następujące funkcje: brak edytora XSLT, brak wsparcia dla debugowania zdalnego. W przeciwieństwo do wersji Professional, w wersji Standard mamy niestety możliwość pracy tylko z jednym wybranym językiem .NET Framework, w związku z tym wybór języka należy dokonać już w momencie zakupu licencji.

Porównując wersje Professional/Standard do Team Edition można zauważyć następujące braki: wsparcia dla 64-bitowego kompilatora, profilera kodu (Code Profiling), analizy statycznej (Static Analysis), testów jednostkowych (Unit Testing), pokrycia kodu (Code Coverage), zarządzania projektem (Project Management) oraz zarządzania przypadkami testowymi (Test Case Management). Pozbawione jest oczywiście również możliwości połączenia do Team Foundation Server.



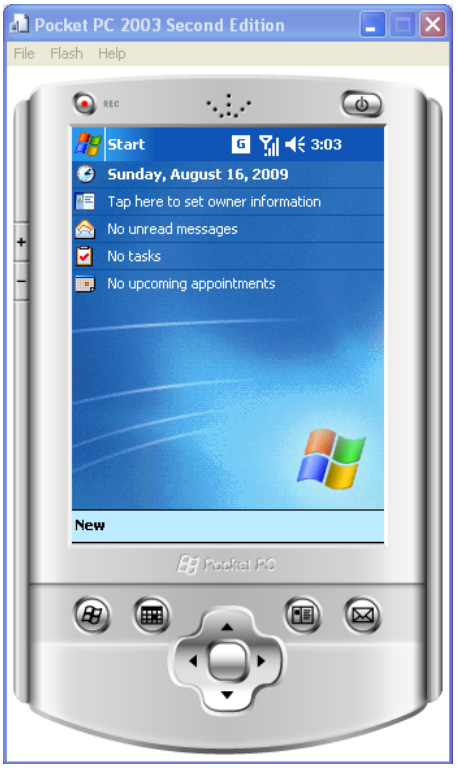
Rysunek 28 Microsoft Visual Studio

Na rynku dostępna jest jeszcze bardzo popularna seria Express. Jest to wersja dla tzw. hobbystów, czyli jak się można domyślić osób uczących się oraz „bawiących się" pisaniem oprogramowania. Licencja serii Express pozwala używać tych produktów do komercyjnych projektów, jednak jak to bywa w tego typu wersjach programów występujące spore ograniczenia uniemożliwiają lub mocno utrudniają używanie tego oprogramowania na co dzień (Rys. 29).



Rysunek 29 Bezpłatna Microsoft Visual Studio Express Edition

Dzięki Microsoft Visual Studio programiści mogą tworzyć programy typu: aplikacje desktop, aplikacje webowe, aplikacje zintegrowane z pakietem Office oraz aplikacje na urządzania mobilne (Rys. 30). Te ostatnie można tworzyć nie posiadając nawet takiego urządzenia. Microsoft Visual Studio posiada wbudowany symulator urządzeń mobilnych (Rys. 30). Ułatwia to bardzo tworzenie takich aplikacji.



Rysunek 30 Emulator Pocket PC

Jednym z najbardziej charakterystycznych i najlepszych elementów Microsoft Visual Studio jest debuger. Umożliwia on dokładne śledzenie stanu aplikacji a także dzięki temu wyłapywanie występujących błędów. Debuger ten jest na tyle rozwinięty, że pozwala na podmianę wartości zmiennych w czasie działania aplikacji. Pozwala on również na zdalne debugowanie. Możemy na przykład badać stan aplikacji Web działającej na zdalnym serwerze.

Microsoft Visual Studio obsługuje główne języki .NET Framework czyli: C#, Visual Basic .NET , C++ i J#. Istnieje oczywiście możliwość instalowania pojedynczych rozwiązań czyli na przykład samego C#. Produkt Microsoft Visual Studio for Office Tools umożliwia tworzenie aplikacji zintegrowanej z narzędziami Office, Jest to bardzo ciekawe rozwiązanie dla osób, które chcą rozbudować arkusz Excel, nie mogą tego zrobić przez mechanizm makr, a nie chcą przenosić całej funkcjonalności arkusza, włącznie z funkcjonalnością Excela do nowej aplikacji. Takim przypadku głównym interfejsem użytkownika pozostałe Excel i może on korzystać z funkcjonalności napisanych na przykład w C# lub Visual Basic .NET.

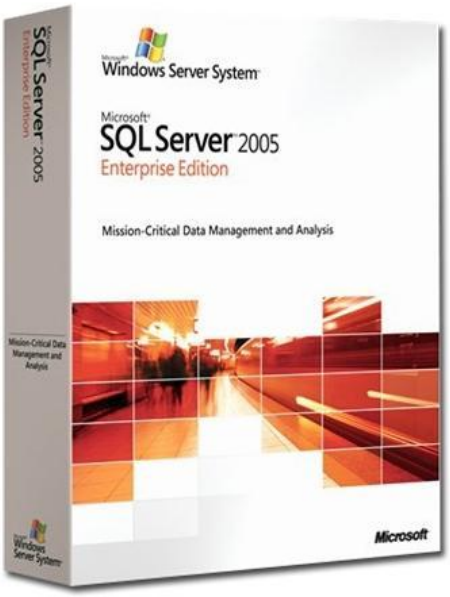
Najnowsza wersja Microsoft Visual Studio to Microsoft Visual Studio 2008. Jest to już 9 wersja tego środowiska. Do napisania pracy magisterskiej używałem wersji starszej Microsoft Visual Studio 2005. Wersja Microsoft Visual Studio 2008 została wzbogacona o obsługę nowych elementów .NET

Framework 3.5: Linq, Windows Communication Foundation, Windows Presentation Foundation oraz Windows Workflow Foundation.

4.2. Microsoft SQL Server 2005

Microsoft SQL Server [6] w ostatnim czasie szybko zdobywa udziały w rynku. Według specjalistów daleko mu do nr 1, czyli do Oracle, ale wraz z rozwojem kolejnych wersji ten odstęp się zmniejsza. Microsoft SQL Server jest bardzo dobrym rozwiązaniem dla firm, które już zainwestowały w infrastrukturę IT opartą o rozwiązania Microsoftu i chcą uniknąć bardzo drogiego rozwiązania bazodanowego jakim jest serwer Oracle.

Microsoft SQL Server jest głównym produktem – bazą danych Microsoft. Jest to relacyjna baza danych działająca w architekturze klient – serwer. Baza jest zgodna ze standardami ANSI/ISO. Wersja użyta do napisania pracy magisterskiej to 9 odsłona Microsoft SQL Server o nazwie kodowej Yukon. Została opublikowana jesienią 2005 r., gdzie wbudowany został język TRANSACT-SQL. Umożliwia on pisanie procedur składowanych, dzięki którym operator bazy danych ma możliwość wykorzystania bardziej zaawansowanych operacji niż SELECT, INSERT, UPDATE i DELETE.



Rysunek 31 Microsoft SQL Server

Microsoft udostępnił również wersję Microsoft SQL Server Express. Jest ona przeznaczona dla mniejszych rozwiązań, na przykład: system zarządzania treścią (CMS), silnik do pisania bloga lub aplikacji desktopowych. W przeciwieństwie do poprzednich wersji nie ma ograniczenia na ilość

połączeń równoległych. Ograniczona jest jedynie ilość obsługiwanych procesorów – jeden, ilość pamięci RAM - 1GB oraz ograniczenia wielkości bazy danych do 4GB danych. Warto zauważyć, że wersja SQL Server Express jest również udostępniona w wersji SQL Server Express with Advanced Services. Zawiera ona opcję zainstalowania Reporting Services oraz Full-Text Search. Konkurencja (Oracle Express, Sybase Express) nie udostępnia tak bogatej funkcjonalności w darmowych pakietach.

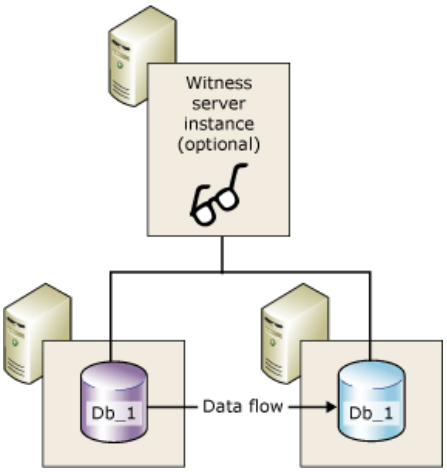
W skład Microsoft SQL Server wchodzą następujące komponenty/usługi: silnik baz danych, SQL Server Integration Sernice (ETL), Reporting Services, Analysis Services (OLAP), Service Broker, Notyfication Services.

W bazie danych utworzono nowy rodzaj danych: XML (Rys. 32). Teraz można wykorzystywać SQL nie tylko do przetrzymywania danych XML w polach typu Blob lub Text, ale również do aktywnego odpytywania. Korzystając z wbudowanego procesora XPath. Dzięki temu nie ma potrzeby pobierania zawartości dokumentu XML z pola bazy danych i jego dalszej obróbki w zewnętrznym procesorze XPath. Można również ograniczać ilość wyświetlanych rekordów właśnie korzystając wyrażeń XPath. Idąc dalej Microsoft wykorzystał powyższą funkcjonalność i dodał możliwość tworzenia tzw. endpointów Web Service. Jest to oczywiście mocno kontrowersyjny sposób udostępniania danych, ale na pewno i dla niego znajdzie się odpowiedni scenariusz.

Column Name	Data Type	Allow Nulls
EmplID	int	☐
Name	varchar(200)	☐
Email	varchar(200)	☐
XMLProfile	xml	☐

Rysunek 32 Dane XML

W wersji Microsoft SQL Server 2005 poprawiono również algorytmy indeksujące oraz system odzyskiwania danych po awarii. Dodano także obsługę mirroringu (Rys. 33) oraz klastrowania. Dzięki temu produkt ten jest bardziej niezawodny, podczas awarii jednego serwera automatycznie uruchamia się drugi i wskakuje na jego miejsce. Z innych nowości, pojawiła się opcja partycjonowania tabel i ulepszono mechanizm replikacji.



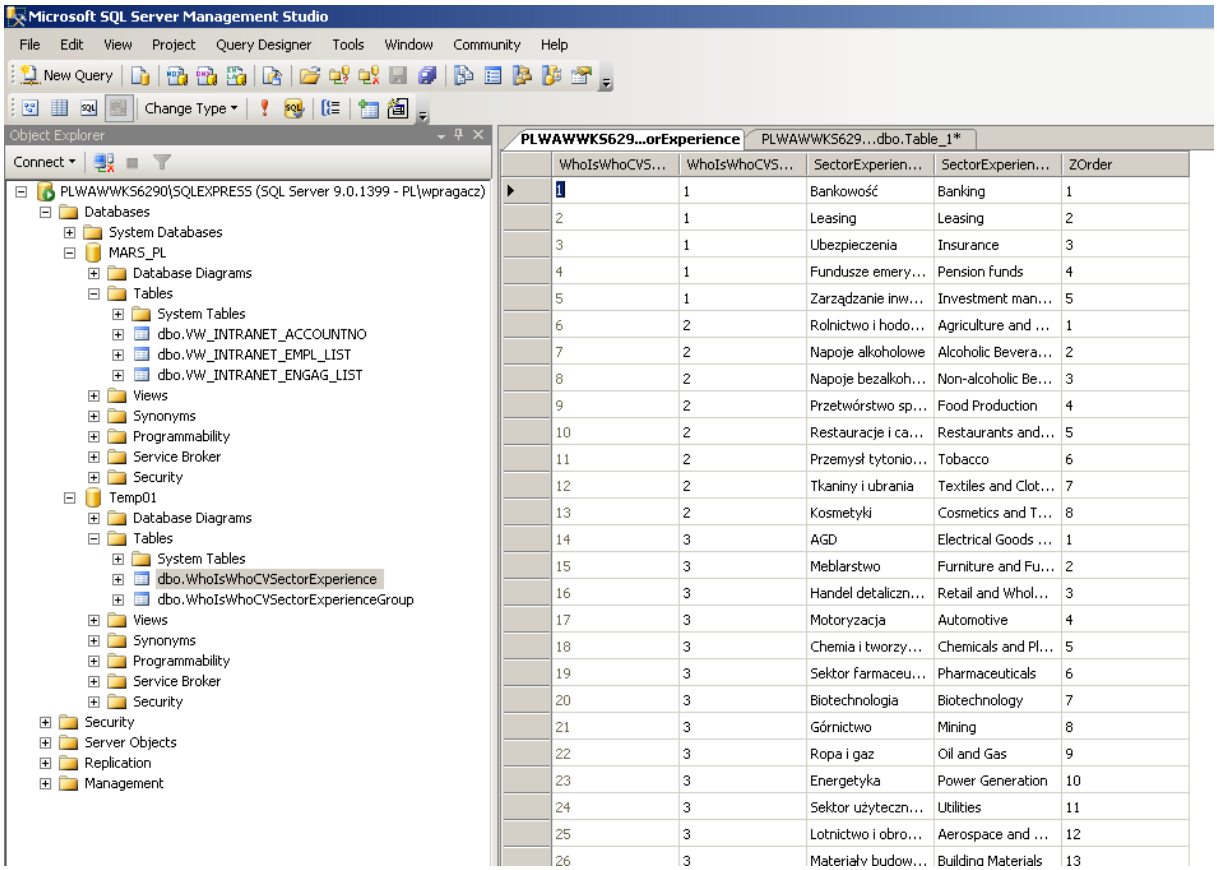
Rysunek 33 Mirroring

Dzięki temu coraz więcej firm uznało i uznaje rozwiązanie Microsoftu, jako wystarczająco stabilne do budowania swoich aplikacji i biznesu na nim. Po głośnym ataku wirusa z robakiem Slammer postawiono na zasadę „secure by default" domyślnie wyłączając wszystkie zbędne usługi Microsoft SQL Server i metody (porty) komunikacji (Rys. 34). Yukon jest pierwszym wydaniem, Microsoft SQL Server, które w swoim głównym wydaniu obsługiwało architekturę procesora x64.

Rysunek 34 Surface Area Configuration

Microsoft SQL Server 2005 jest pierwszą wersją, w której pojawiła się również możliwość integracji z Common Language Runtime (CLR). Dzięki temu do pisania procedur możemy wykorzystywać C# lub Visual Basic .NET oprócz standardowo zaimplementowanego Transact-SQL.

Zmianie uległy narzędzia. Administratorzy zyskali nowe Microsoft SQL Server Management Studio (Rys. 35). W programie zaszły rewolucyjne zmiany w stosunku do poprzedniej wersji. Tyczy się to głównie interfejsu użytkownika. Serwery i bazy danych SA teraz eksponowane w drzewie przy pomocy osobnego okienka, które można w zależności od upodobać umieścić po prawej lub lewej stronie głównego okna. Czy zmiany te były na lepsze czy gorsze zdania są już podzielone?



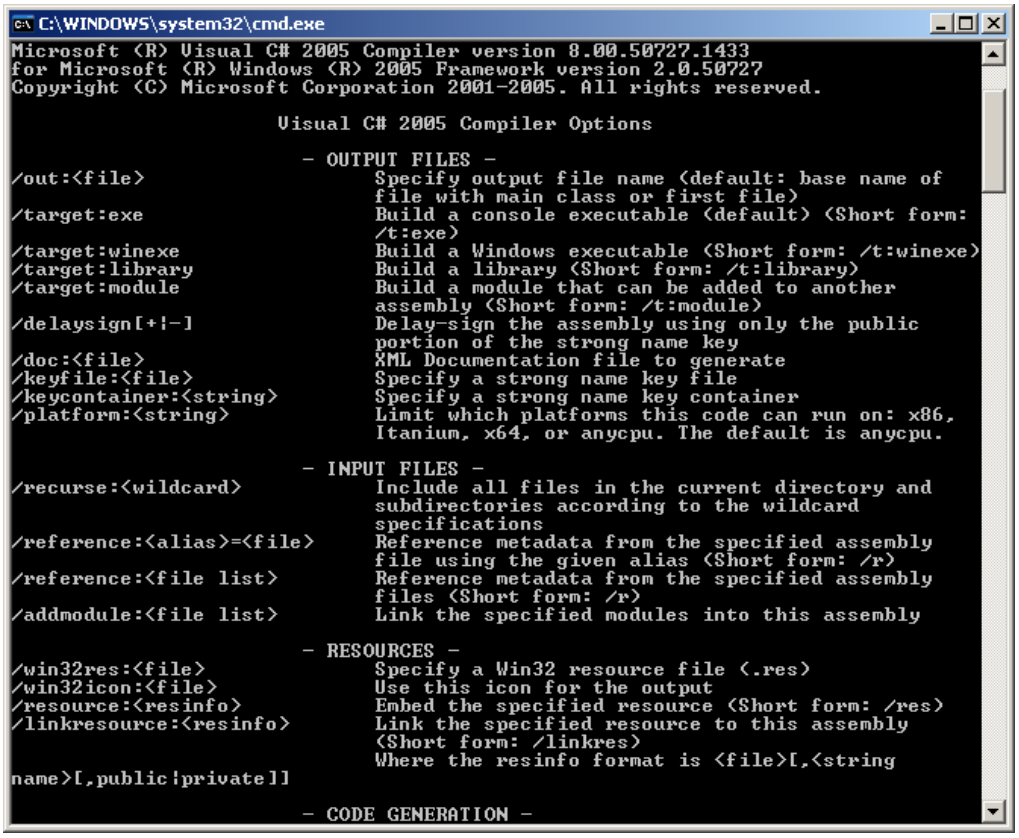
Rysunek 35 Microsoft SQL Server Management Studio R33

Programiści i analitycy otrzymali również zupełnie nowy SQL Server Business Intelligence Development Studio. Produkt ten jest oparty o Microsoft Visual Studio. Microsoft udostępnił wersję Microsoft SQL Server Compact. Jest to darmowy dodatek do głównej bazy danych typu embeded. Jest ona uruchamiana w procesie wykorzystującej jej aplikacji, a nie jako osobny proces. Na raz może z niej korzystać tylko jedna aplikacja. Microsoft SQL Server posiada mechanizmy, dzięki którym można szybko zsynchronizować bazę „matkę" z tą kompaktową na przenośnym urządzaniu, które tylko od czasu do czasu ma łączność z głównym systemem teleinformatycznym w firmie. W 2008 wyszła 10 wersja SQL Servera – MS SQL Server 2008 o nazwie kodowej Katmai.

4.3. Kompilator CSC.EXE

Narzędzie CSC.EXE służy do kompilowania kodu napisanego w języku C#. Jest on częścią .NET Framework. Każda z wersji zarówno 1.1, 2.0, 3.0, jak i 3.5, dysponują osobną wersją tego programu. Jest to aplikacja konsolowa (Rys. 36). Poprzez przekazywanie parametrów wywołania

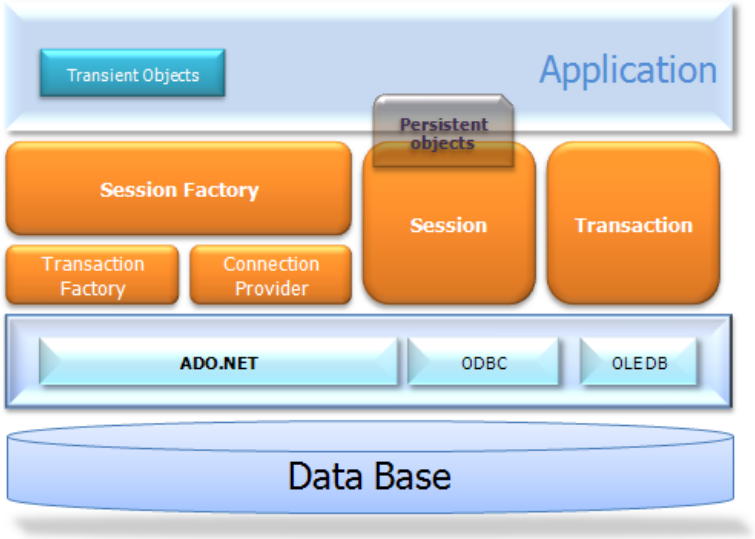
wskazujemy pliki C# lub inne (tzw. resources) do utworzenia pliku wykonywalnego lub biblioteki. Kompilator wspiera szereg różnych platform: x86, Itanium oraz x64. Dodatkowo pozwala również generować pliki dokumentacji w formacie XML. W tym celu trzeba oczywiście wpierw odpowiednio udokumentować kod w plikach *.cs poprzez dodanie odpowiednich komentarzy.



Rysunek 36 Kompilator CSC.EXE R34

4.4. NHibernate

NHibernate jest biblioteką ORM (Object Relation Mapper). Jak wskazuje sama nazwa umożliwia on mapowanie struktury relacyjnej bazy danych do świata obiektowego. NHibernate jest tzw. portem znanego Java –owego narzędzia Hibernate. Oba projekty są utrzymywane przez spółkę RedHat-a JBoss. Są to projekty open sourceowe udostępniane na licencji GNU LGPL. Wersja NHibernate 1.0 odpowiada wersji Hibernata 2.1 z drobnymi elementami Hibernata 3.0.



Rysunek 37 Architektura NHibernate

Podstawową zaletą bibliotek ORM jest odciążenie programistów od pisania warstwy dostępu do bazy danych (Rys. 37). Musi on jedynie określić sposób mapowania poszczególnych kolumn z tabeli na odpowiednie definicje klas i ich atrybutów (Rys. 38). NHibernate obsługuje referencje pomiędzy obiektami, czyli relacje klucza obcego pomiędzy tabelami w bazie. Wykorzystuje przy tym tzw. lazy loading. Obiekty, do których chce dostać dostęp program są ładowane dopiero na konkretne żądanie, a nie przy uruchomieniu pierwotnego zapytania. NHibernate potrzebuje do działania obiektów POCO. Jest to jedna z głównych cech charakterystycznych w stosunku do konkurencyjnych produktów. POCO (Plain Old CLR Objects) są używane w modelu tzw. anemicznym [7]. Takie obiekty z samymi atrybutami do przechowywania danych, nadają się o wiele bardziej do ich przesyłania pomiędzy różnymi warstwami aplikacji a dzięki temu można je używać w szerszym kontekście. Nie zawierają one w sobie żadnej funkcjonalności oraz nie odwołują się do źródeł danych. API NHibernate pozwala na obsłużenie większości scenariuszy dostępu do danych: dodawanie, aktualizacja, usuwanie, pobieranie pojedynczego obiektu, pobieranie kolekcji obiektów. Dodatkowo w NHibernate został wbudowany język HQL. Jest to język obiektowych zapytań pozwalający budować bardziej skomplikowane zapytania.

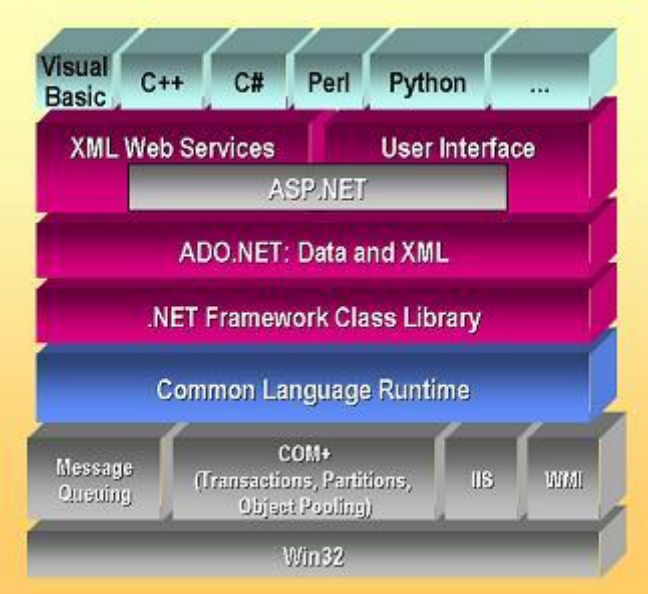
```xml
<?xml version="1.0" encoding="utf-8" ?>

<hibernate-mapping xmlns="urn:nhibernate-mapping-2.2">
    <class name="Kpmg.myKPMG.Core.Domain.SyslogEntry, Kpmg.myKPMG.Core" schema="dbo" table="[Syslog]" lazy="false">
        <id name="SyslogID" type="Int32" >
            <generator class="identity" />
        </id>
        <property name="InsertDate" type="DateTime" />
        <property name="EventType" type="String" />
        <property name="EventName" type="String" />
        <property name="URL" type="String" />
        <property name="Module" type="String" />
        <property name="Description" type="String" />
    </class>
</hibernate-mapping>
```

Rysunek 38 Przykładowy plik konfiguracyjny HBM

4.5. .Net 2.0

.Net Framework [8, 9] jest platforma programistyczną utworzona przez firmę Microsoft. Składa się z Common Language Runtime (CLR), Base Classes Library (BCL), Common Intermediate Language (CIL), Common Language Infrastructure (CLI) oraz Common Type System (CTS) (Rys. 39).

W przeciwieństwie do Javy obsługuje wiele różnych języków w tym C#, Visual Basic.NET, J#, C++, Delphi.NET. Przy pomocy tej platformy możemy tworzyć oprogramowanie, które może pracować jako samodzielne aplikacje konsolowe, aplikacje desktopowe, usługi Windows, czy strony WWW. Cechą charakterystyczną jest to, że kod źródłowy tworzonych aplikacji jest kompilowany do kodu pośredniego Common Intermediate Language. Później ten kod pośredni jest tworzony przez Common Language Runtime na komputerze, na którym jest uruchamiany program do kodu maszynowego (Rys. 40). Dzięki temu jest możliwa przenaszalność kodu pomiędzy różnymi platformami. Pierwotnie Microsoft miał na myśli przenoszenie pomiędzy różnymi wersjami systemu Windows. Dzięki implementacji opensourceowej Mono mamy teraz możliwość korzystać z przenoszalności kodu pomiędzy Windowsem, a Linuxem.

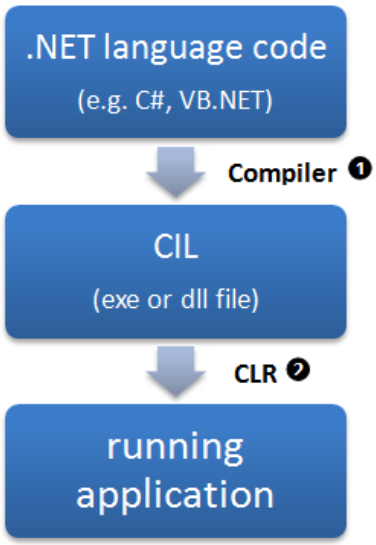


Rysunek 39 Microsoft .NET Framework

.NET Framework miał w założeniach projektantów rozwiązać problem tzw. „DLL hell". W przeszłości technologia DLL (Dynamic Link Libraries) cierpiała z braku mechanizmów do wersjonowania bibliotek. Bardzo często okazywało się, że nowa aplikacja nadpisywała dla innej wspólne biblioteki. Przy usuwaniu aplikacji mogło dojść do usunięcia współdzielonych bibliotek, dlatego pojawiał się błąd o braku potrzebnej biblioteki lub braku określonej metody. Skutkowało to niezrozumiałymi dla końcowego użytkownika komunikatami i jego duża frustracją. Lekarstwem na ten problem miał być element .NET Framework Global Assembly Cache. Jest to katalog zarejestrowanych bibliotek współdzielonych przez wiele aplikacji. GAC obsługuje wersjonowanie dzięki temu możemy wymusić odwołanie nie tylko do konkretnej biblioteki, ale także wymusić odwołanie do wersji nie starszej niż (Version Policy). Jest to możliwe dzięki Common Language Runtime zarządzającemu naszym uruchomionym programem.

Common Language Runtime dostarcza takie elementy funkcjonalne jak bezpieczeństwo – Code Access Security (CAS), zarządzanie pamięcią (Garbage Collector) oraz obsługę wyjątków (Exceptions). Code Access Security działa w oparciu o źródło uruchamiania kodu i umożliwia tworzenie polityk (Policy) dla niego. Dzięki temu mamy możliwość ograniczania funkcjonalności aplikacji bez ingerencji w jej kod. Możemy na przykład ograniczyć użytkownikom jakiejś grupie security dostęp do zapisu na dysku. .NET Framework jest dostarczany domyślnie z politykami dla następujących podmiotów: organizacji (enterprise), komputera (machine) oraz konkretnego użytkownika (user). Garbate Collector zajmuje się jak sama nazwa wskazuje odśmiecaniem pamięci.

Jednym z założeń Common Intermediate Language jest to, że programista operuje na referencjach do obiektu a nie na samych obiektach. Wirtualna maszyna kontroluje stan aplikacji i powiązania referencji z obiektami po usunięciu ostatniej referencji, na przykład poprzez jej nadpisanie lub zakończenie metody, pętli, lub bloku kodu, oznacza obiekt jako gotowy do usunięcia. Można ręcznie oznaczyć obiekt poprzez wywołanie metody Dispose, którą każdy z obiektów powinien mieć zaimplementowaną. Co pewien czas jest uruchamiany Garbadge Collector i usuwa on z pamięci obiekty wcześniej oznaczone do usunięcia. Całym „urokiem" tego rozwiązania jest to, że dzieje się to niezależnie od programisty. Podczas usuwania obiektu jest wywoływana metoda Finalize. Nie powinno się jej jednak jawnie wywoływać z kodu, stąd wzięło się właśnie pojecie kodu zarządzanego. Kod jest wykonywany pod kontrolą Common Language Runtime, więc jest możliwość kontrolowania, zarządzania nim. Takiej możliwość nie ma w przypadku aplikacji napisanych w starszych technologiach takich jak C++ czy Visual Basic.



Rysunek 40 Common Intermediate Language

Podstawową jednostką w kontekście, której uruchamiany jest kod jest assembly, w którego skład wchodzą źródła skompilowane do kodu pośredniego - Common Intermediate Language, metadane opisujące ten kod znajdujący się w assembly oraz na przykład elementy graficzne używane przez aplikację lub pliki XML.

Assemblies mogą mieć rozszerzenie dll lub exe w zależności od jego typu, czyli czy ma być uruchamiany samodzielnie, czy jako część jakiegoś kontenera (ASP.NET, usługa Windows).

W pliku metadanych AssemblyInfo.cs znajdują się informacje zawierające wersje, autora, firmę oraz lokalizację (Culture). Dzięki temu możemy kontrolować format wyświetlania daty, symbol użyty do separowania, części dziesiętnej od całkowitych pól.

Kolejnym elementem .NET Framework jest Base Class Library (BCL) (Rys. 41). Base Class Library zawiera zestaw bibliotek do obsługi interfejsu dostępu do danych, obsługi systemu plików, obsługi wejścia wyjścia, łączności z bazami danych, kryptografii, budowania aplikacji web oraz komunikacji sieciowej zarówno natywnej jak i Web Services.

Rysunek 41 Base Class Library

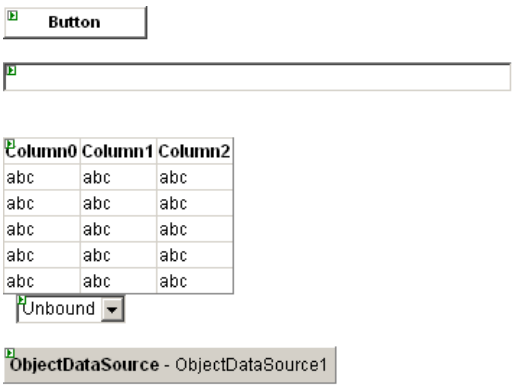
Kod aplikacji jest podzielony na tzw. przestrzenie nazw (namespaces). Dzięki temu struktura kodu aplikacji jest bardziej przejrzysta. Można dzielić kod na w logiczne grupy niezależnie od jego lokalizacji w assemblies.

W skład .Net Framework wchodzą frameworki ASP.NET oraz ADO.NET. Są to integralne części .NET Framework i w pliku instalacyjnym .NET Framework są wszystkie potrzebne pliki do obsługi tych technologii. Jest to bardziej historyczne i marketingowe podejście. Do starych technologii ASP i ADO dodano końcówkę .NET. Miało to na celu podkreślenie, że są one obecnie wspierane przez .NET Framework. ASP.NET jest technologia umożliwiającą pisanie interaktywnych stron WWW, czyli tzw. aplikacji Web. W modelu tym interfejs użytkownika jest oparty o przeglądarkę WWW, logika aplikacji znajduje się natomiast na serwerze. Technologia zostanie szerzej opisane poniżej w osobnym podrozdziale.ADO.NET jest natomiast technologią dostępu do danych. Zawiera ona wsparcie dla baz danych Microsoft poprzez natywne sterowniki do SQL Servera lub technologię OLEDB opierającą się o COM. ADO.NET wspiera również dostęp do bazy danych Oracle.

4.6. ASP.NET

ASP.NET jest technologią do budowania aplikacji Web. W tym wypadku przeglądarka WWW służy, jako interfejs użytkownika. Tego typu aplikacje nazywamy „cienkim klientem” ze względu na to, że na komputerze użytkownika renderowany jest jedynie sam interfejs użytkownika. Cała logika aplikacji znajduje się po stronie serwera w jednej centralnej lokalizacji. To podejście ma wiele zalet - przede wszystkim łatwość wprowadzania zmian i modyfikacji. Nową wersję oprogramowania wystarczy wgrać do jednej lokalizacji na serwerze. Nie ma potrzeby aktualizowania za każdym razem oprogramowania na każdym komputerze użytkownika. Ze względu na to, że za każdym razem trzeba przesłać cały interfejs użytkownika, technika ta sprawdza się bardziej do przesyłania danych tekstowych, niż danych binarnych jak na przykład obrazy. Technologia ta jest więc idealnie wykorzystywana w dużych środowiskach, jak na przykład korporacje z setkami stacji roboczych do pisania aplikacji wspierających zarządzanie danymi – przykładowo w firmie ubezpieczeniowej czy finansowej.

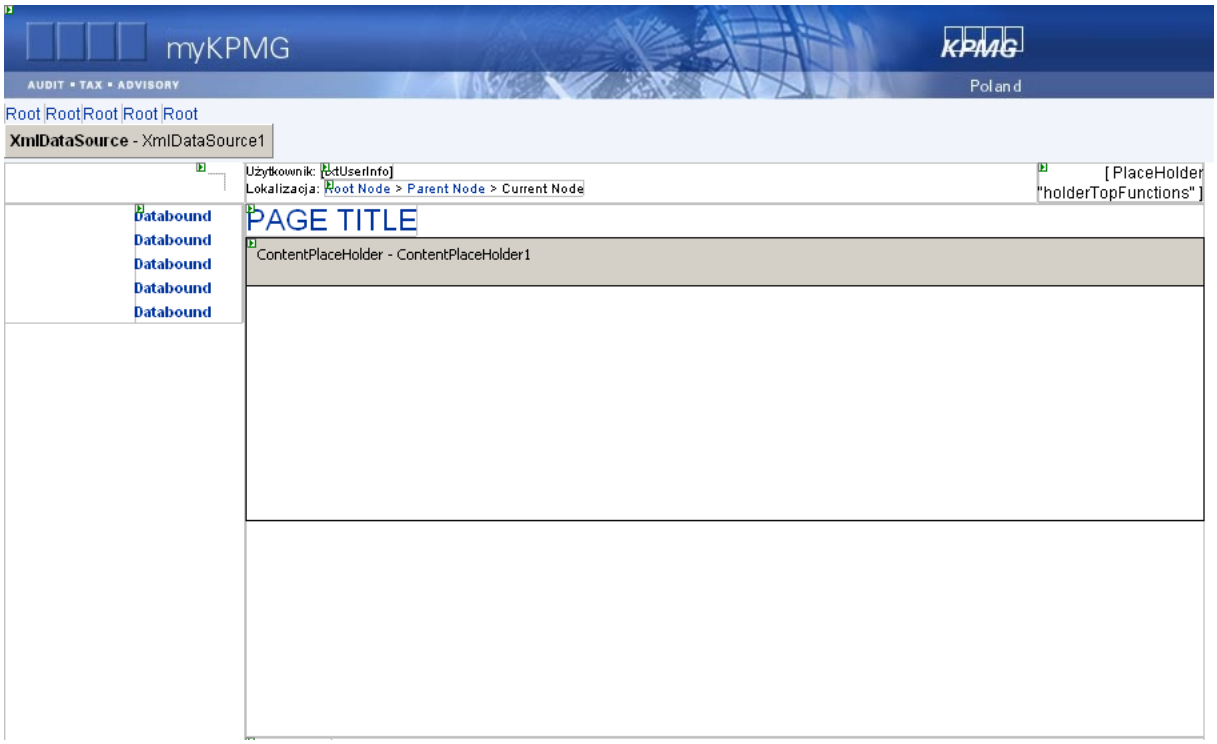
Jednym z założeń ASP.NET było umożliwienie pisania aplikacji web osobom, które nie znają specyfiki pracy z bezstanowym protokołem HTTP oraz językiem HTML. Funkcjonalność ta była adresowane szczególnie do programistów Visual Basic. Przyzwyczajeni oni byli do wizualnego tworzenia aplikacji w oparciu o kontrolki przeciągane na formularz (Rys. 42). Dlatego w ASP.NET mamy kontrolki serwerowe, które pozwalają „wyklikać” wizualnie nową aplikacje.



Rysunek 42 Kontrolki ASP.NET

ASP.NET ma również przewagę nad starym ASP.NET w rozdzieleniu kodu HTML wysyłanego do klienta z kodem wykonywanym po stronie serwera. Ta koncepcja nazywa się code behind. W plikach ASPX lub ASMX znajduje się wskazanie na konkretny assembly, oraz nazwę klasy wraz z pełnym namespacem. Dzięki temu unikamy bałaganu zwanego „spaghetti code”. Użytkownika ma

również możliwość samodzielnego tworzenia kontrolek serwerowych, które można oddzielnie dystrybuować oraz na przykład sprzedawać. W ASP.NET bardzo dobrze rozwiązano mechanizm szablonów stron. W celu zachowania zgodności każdej ze stron z odgórnymi wymogami, wskazane jest umieszczenie kodu HTML, ASP.NET i powtarzających się na wielu stronach treści w jednym miejscu. Takimi mechanizmami w ASP.NET są Master Pages oraz User Controls. Pierwsze rozwiązanie jest typowym mechanizmem szablonów. Programista tworzy na początku szablon strony: nagłówek, stopkę, górne menu, lewe menu oraz określa w CSS wygląd strony (Rys. 43). W samym środku strony jest umieszczany specjalny kontener, w którym będzie się wyświetlała właściwa treść strony WWW czy aplikacji web. Przy tworzeniu nowej strony (ASPX) użytkownik może wybrać szablon dla tworzonej strony i wówczas treść oblewa stworzony wcześniej szablon. Drugim podejściem jest tworzenie tzw. User Controls, które pozwalają na utworzenie jakiś treści lub funkcjonalności, które można umieścić w wielu miejscach na stronie. Zmiana wewnątrz samej User Control powoduje zmianę w każdym z miejsc, gdzie wykorzystana jest ta kontrolka.



Rysunek 43 Przykładowy szablon Master Page

Cechą charakterystyczna protokołu HTTP jest bezstanowość. Oznacza to, że użytkownik korzystając z przeglądarki internetowej wysyła komunikaty tzw. żądania do serwera WWW, a serwer WWW odpowiada użytkownikowi nagłówkiem oraz treścią najczęściej w formacie HTML.

Bezstanowość polega na tym, że przeglądarka nie pamięta, iż użytkownik wpisał tekst lub zaznaczył pole typu checkbox na poprzedniej stronie. Taka informacja musi być przekazana jawnie do serwera, jako parametr żądania, który następnie musi być uwzględniony przy tworzeniu odpowiedzi, generowaniu HTML nowej strony. Obecnie w każdej szanującej się technologii web zaimplementowana został obsługa mechanizmu Cookie lub bazując na niej obsługa sesji użytkownika. W ASP.NET pojawiło się nowe podejście w walce z bezstanowością jest to tzw. Viewstate. Technologia ta polega na zapisaniu w ukrytym polu HTML zaenkryptowanej wartości pól na stronie. Dzięki temu po wciśnięciu na przykład przycisku zapisującego jakieś informacje, silnik ASP.NET pamięta wpisaną wartość w polu i nie ma potrzeby ręcznego przekazywania zawartości pola, jako parametru zadania, a później przepisywania tego parametru do wyświetlanych ponownie pól. Technologia ta może prowadzić, do bardzo dużego rozrostu strony, dlatego należy się z nią obchodzić bardzo ostrożnie przy aplikacjach, które działają na wolniejszym łączu lub są użytkowane przez setki lub tysiące osób.

Innymi sposobami przetrzymywania danych w bezstanowym środowisku ASP.NET są mechanizmy Session, Cache oraz Application State. Session jest rozwiązaniem opierającym się o Cookie, który pozwala przypisywać konkretnemu użytkownikowi, gościowi strony wartości widoczne z poziomu aplikacji tylko dla niego. Jest to idealne miejsce do przechowywania informacji o profilu osoby, która porusza się po spersonalizowanych stronach. Drugą metodą jest Ciche -technologia, która pozwala przechowywać jakieś serializowalne treści dostępne dla wszystkich użytkowników. Cechą charakterystyczna Cache jest to, że dane te wygasają do określonym czasie. Mamy możliwość dokładnego sterowania tym wygasaniem, na przykład możemy ustawić, że konkretna dana dodana do Cache ma zostań zeń usunięta 10 min. po ostatnim użyciu lub dane mają być usunięte dokładnie po 15 minutach od ich zapisania. Jest to dobre rozwiązanie do buforowania danych z wolniejszych źródeł, na przykład jakaś starsza baza danych, plik tekstowy lub źródło danych w odległej lokalizacji. Ostatnią omawianą technologią do buforowania danych jest Application State. Pozwala on na przetrzymywanie zserializowanych danych w podobny sposób jak to wygląda w Cache, natomiast nie mamy możliwości wpływania na okres trwania danych. Są one przechowywane tak długo, jak długo pracuje obecny watek. Po ponownym uruchomieniu procesu ASP.NET, czyli pooli aplikacyjnej dane są tracone. Application State różni się również tym, że przypadku bardziej skomplikowanych systemów posiadających kilka serwerów web w swoim rozwiązaniu. Dane z Cache mogą być współdzielone pomiędzy różnymi serwerami WWW. W przypadku Application State odwołujemy się zawsze do lokalnej maszyny i nie ma możliwości współdzielenia danych.

Microsoft dostrzegł „nowy" trend budowy aplikacji Web w oparciu o interaktywny JavaScript. W tym celu do ASP.NET dodano wsparcie dla technologii AJAX. Głównym elementem jest komponent o nazwie UpdatePanel (Rys. 44). Dzięki niemu można wydzielić na stronie WWW część,

która będzie się aktualizowała niezależnie od reszty strony. Taki kawałek strony może się odświeżać co pewien czas automatycznie lub akcja ta może zostać wywołana przez zdarzenie (np. kliknięcie) w innej części strony. Jest to bardzo praktyczne rozwiązanie, ponieważ można wykorzystać już istniejące kontrolki serwerowe, a w rezultacie nie trzeba od nowa pisać ich funkcjonalności wzbogaconej o AJAX. Microsoft starał się maksymalnie ułatwić pracę programiście, na czym ucierpiała wydajność całego rozwiązania. Przy odświerzaniu UpdatePanel wysyłany jest bowiem cały kod HTML kontrolki. Kontrolki pisane od początku z myślą o AJAX potrafią samodzielnie komunikować się z serwerem i wysyłać/odbierać dane, które faktycznie podlegają zmianie, a nie całość kodu HTML reprezentującego dany fragment strony. Dodatkowo oprócz UpdatePanel pojawiło się rozbudowane API w JavaScript, które umożliwia bardziej zaawansowanym programistom pisanie aplikacji wykorzystujących AJAX bardziej finezyjnie niż poprzez odświeżanie całego kawałka strony.

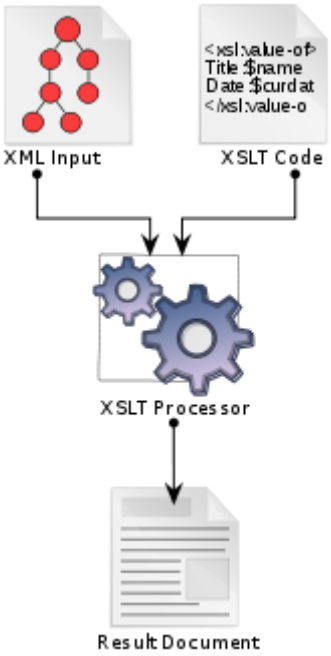
Rysunek 44 ASP.NET AJAX UpdatePanel

Wersji ASP.NET 3.5 wprowadzono możliwość programowania zgodnego z MVC. Rozwiązanie to ucieszyło na pewno grupę osób, które były nie do końca zachwycone z obecnego sposobu tworzenia stron w ASP.NET. Silnik ASP.NET sam dodaje potrzebny kod HTML i JavaScript do stron wysyłanych z serwera do przeglądarki klienta. Prowadzi to do niekontrolowanego rozrostu stron i do spowolnienia całej aplikacji. Dzięki ASP.NET MVC aplikacje są tworzone w prosty i przejrzysty sposób zgodnie ze wzorcem Model View Controller. Programista ma całkowitą kontrole nad każdym znakiem wysłanym z serwera i dzięki temu można pracować nad optymalizacją kodu HTML. Każdy kij ma dwa końce i tym razem nie jest inaczej. Całkowita kontrola nad kodem oznacza również ręcznie implementowanie funkcjonalności, które są dostępne w klasycznym ASP.NET Webforms na przykład: kalendarzyk.

4.7. XSLT

XSLT (Extensible Stylesheet Language Transformations) [10, 11] jest funkcyjnym językiem programowania. Jego zadaniem jest opisanie, w jaki sposób ma zostać przetworzony dostarczony

dokument XML. XSLT pozwala na tłumaczenie z jednego formatu na drugi. Możemy np. pobrać dokument zgodny ze standardem xml i zamienić go na inny rodzaj dokumentu np. xhtml lub plik tekstowy (Rys. 45). Tworzy on wraz z innymi technologiami XPath (XML Path Language) oraz XSL-FO (XSL Formatting Objects) większa rodzinę technologii do przetwarzania XML znaną jako XSL. Jednym z popularniejszych zastosowań XSLT to generowanie stron WWW w serwisach internetowych. Na podstawie parametrów żądania HTTP oraz pobieranych danych z bazy danych tworzony jest dokument XML zawierający surowe dane. Ten dokument łączony jest z szablonem XSLT i generowany jest dokument wynikowy. Wewnątrz szablonu można się odwoływać do pól pliku danych (XML) poprzez wygodny język zapytań zbudowany na technologii XPath. W rezultacie programista może pobierać pojedyncze wartości pól lub może na przykład sczytywać dane w pętli (np. lista rekordów). XSLT daje również możliwość importowania innych plików XSL. Jest to naprawdę bogate w funkcjonalności rozwiązanie. Przykładowo istnieje możliwość wywoływania rekurencyjnych zapytań.

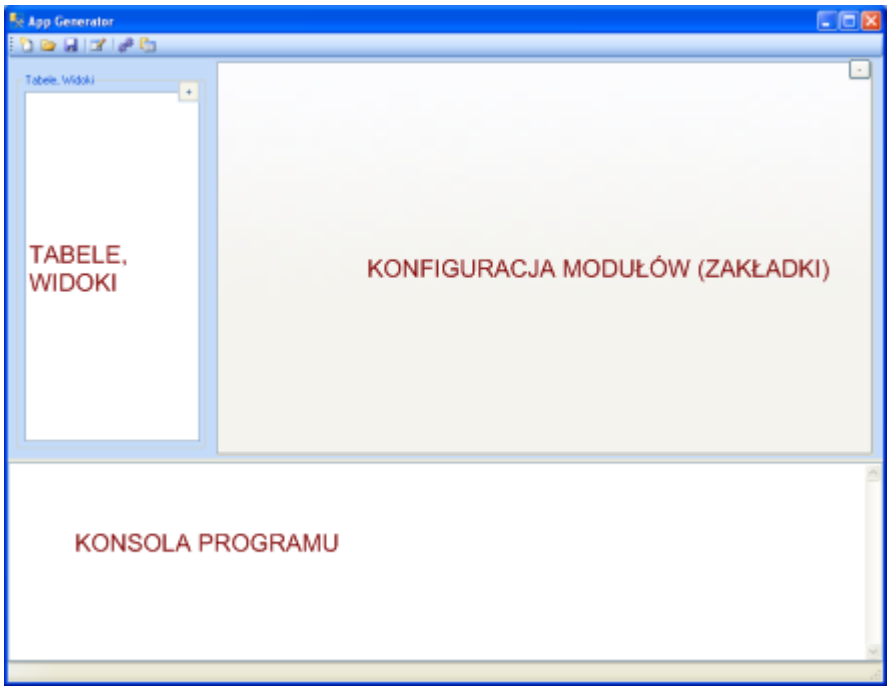


Rysunek 45 XSLT

5. Opis implementacji generatora kodu

5.1. Opis interfejsu użytkownika

App Generator jest aplikacją desktopową. Interfejs użytkownika (Rys. 46) jest podzielony na 3 główne części. Po lewej stronie znajduje się lista tabel oraz widoków SQL z podłączonej bazy danych, na podstawie których można wygenerować kod aplikacji. Po prawej stronie mamy okno zakładek. Każda zakładka odpowiada konfiguracji jednego tworzonego modułu. W dolnej części jest konsola programu dająca możliwość podglądu na aktualnie działające zadania. Na samej górze znajduje się pasek ikonek umożliwiających otwieranie projektu, zapisywanie projektu, edycje ustawień projektu, generowanie samego kodu lub generowanie pełnej aplikacji.

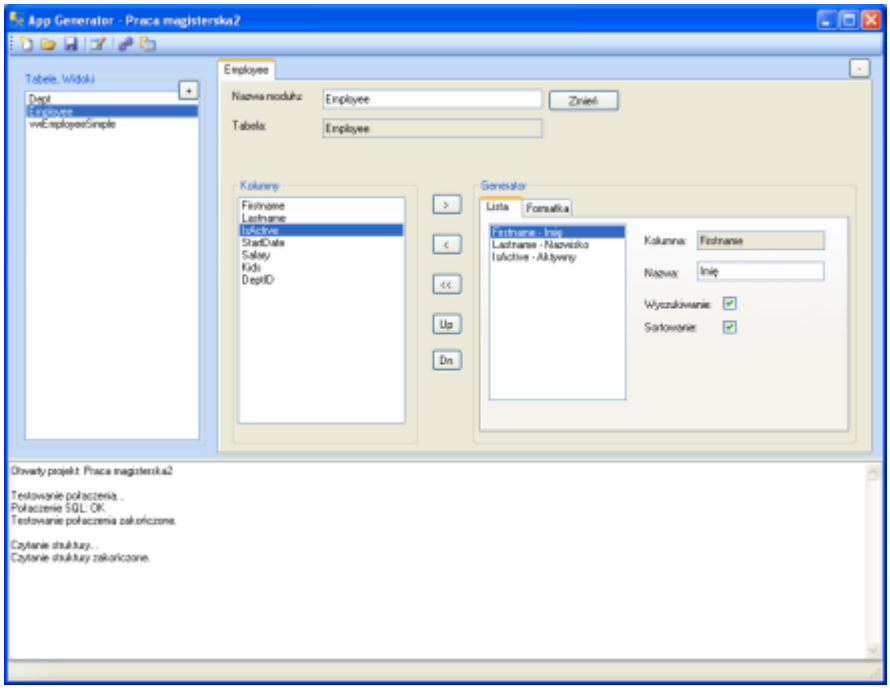


Rysunek 46. Głowne okno App Generatora

Okno ustawień (Rys. 47) składa się z 4 pól tekstowych: nazwa profilu, parametry połączenia do bazy danych (ConnectionString), katalog wyjściowy (w którym zostanie umieszczone są wygenerowane źródła lub aplikacja gotowa do uruchomienia), przestrzeń nazw dla utworzonego kodu oraz pole wyboru zestawów szablonów.

Rysunek 47. Okno ustawień

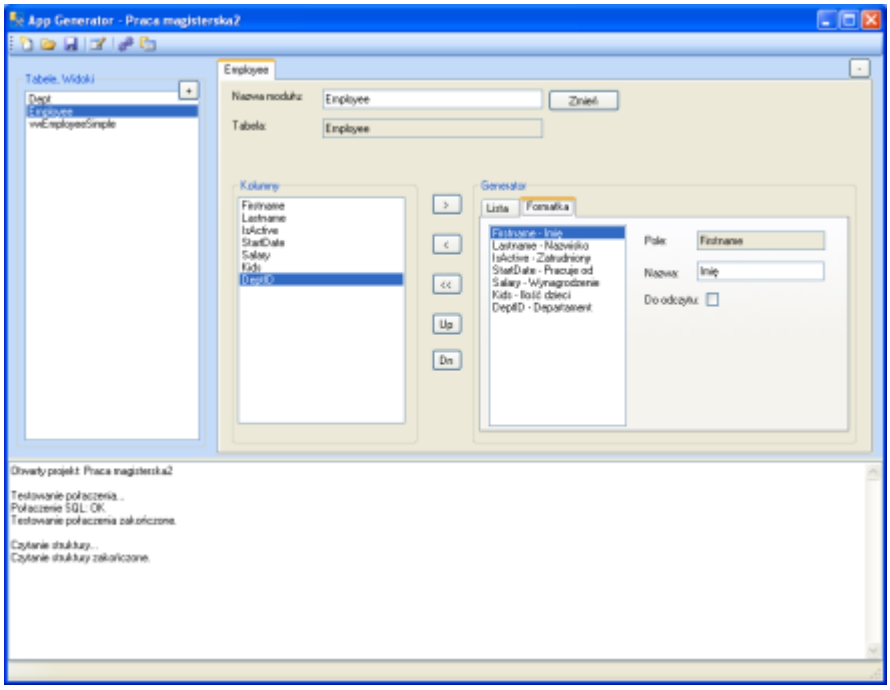
Każda z zakładek konfiguracji modułu składa się z jego nazwy, która domyślnie jest ustawiana na taka samą jak nazwa obiektu (tabeli lub widoku). W jej skład wchodzi również nazwa wspomnianego obiektu źródłowego – pole tylko do odczytu. Pod spodem mamy interfejs graficzny do tworzenia warstwy prezentacji powstającego programu. Po lewej stronie mamy listę dostępnych kolumn na podstawie których będziemy generować nasze rozwiązanie. Na prawo znajdują się dwie zakładki Lista i Formatka. Są one odpowiednio odpowiedzialne za wygląd ekranu listy rekordów i ekranu edycji.



Rysunek 48. Zakładka konfiguracyjna listy rekordów

Zakładka Lista (Rys. 48) składa się z listy wyselekcjonowanych przez użytkownika kolumn do tworzonej aplikacji oraz ustawień dotyczących aktualnie podświetlanego wiersza w tej liście. Składają się one z nazwy oraz dwuch pól typu checkbox. Pola określają czy wybrana kolumna powinna brać udział w wyszukiwaniu oraz czy można po niej sortować listę.

Zakładka Formatka (Rys. 49) jest zbudowana podobnie do wcześniej opisana fragment ekranu. Posiada ona listę kolumn tabel, które zostaną przetransformowane do pól formatki, nazwę aktywnego pola oraz pole wyboru czy wybrany element ma być generowany tylko do odczytu.



Rysunek 49. Zakładka konfiguracyjna formularza

Pomiędzy zakładkami, a lista kolumn z bazowej tabeli umieszczono zestaw przycisków, Pozwalają one dodawać lub usuwać poszczególne kolumny

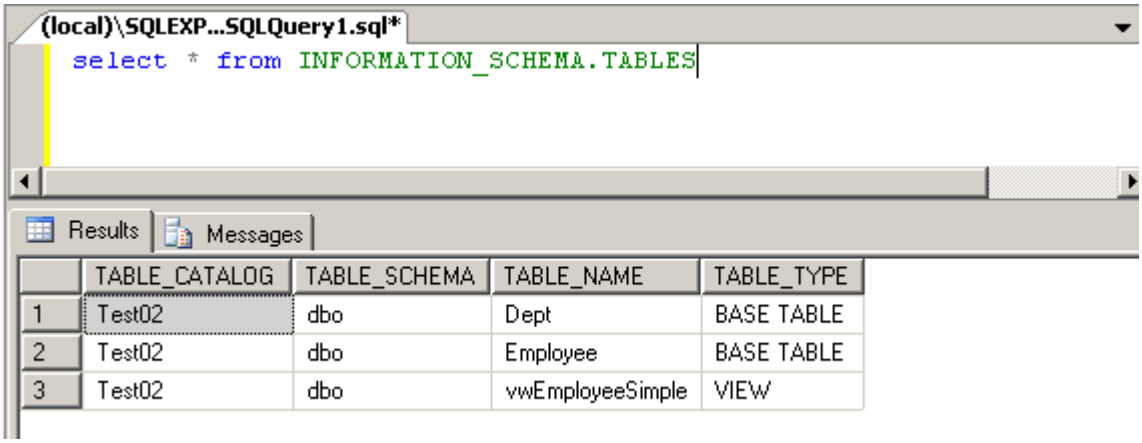
5.2. Czytanie struktury bazy danych na podstawie metadanych

Do działania opracowany program potrzebuje pozyskać strukturę bazy danych. Metadane opisują nam w jaki sposób zbudowana jest baza. Informacje te są przeważnie dostępne w postaci widoków systemowych i można je odczytywać wykonując polecenie SQL Select.

Odpytując widoki systemowych, uzyskujemy dane na temat listy tabel, ich nazw, nazw kolumn w poszczególnych tabelach, ich nazw typów, wielookości kolumn oraz wartości domyślnych. Do pobrania są również informacje na temat założonych indeksów, kluczy głównych, indeksów unikalnych. Możemy się również dowiedzieć się czegoś temat relacji pomiędzy tabelami, o kluczach obcych.

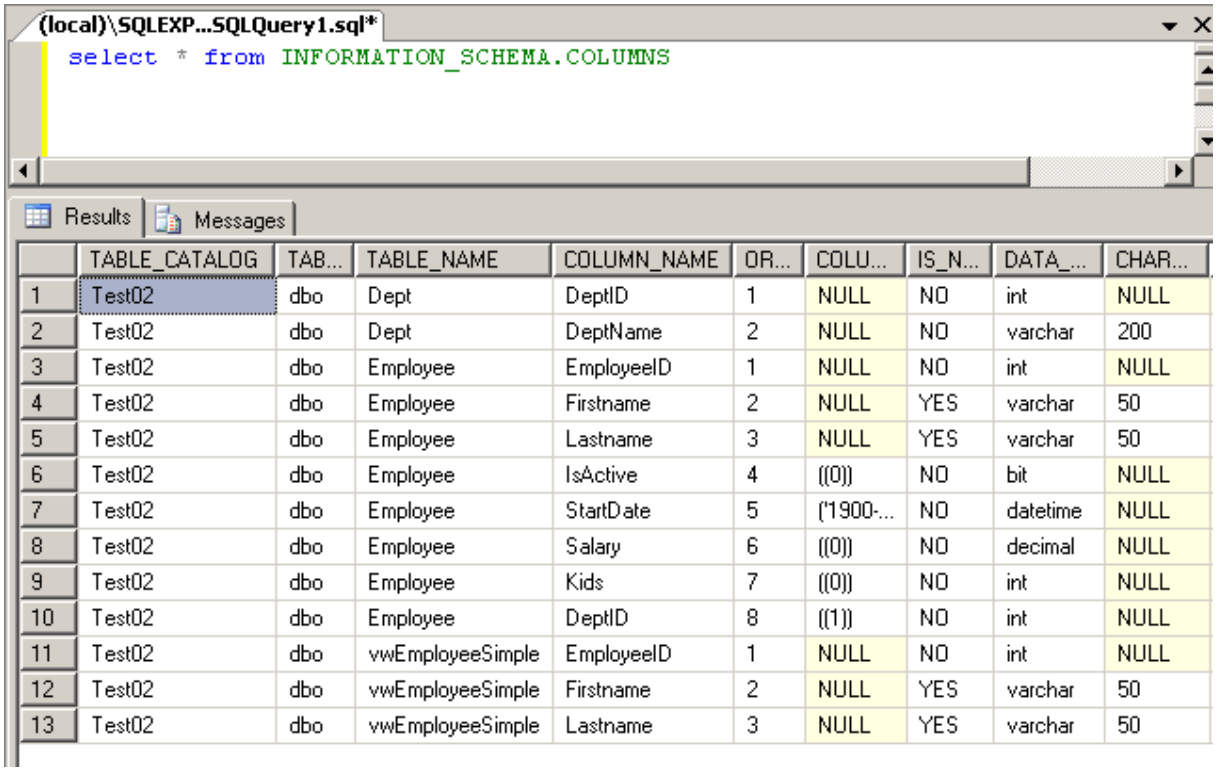
W bazie Microsoft SQL Server [6] w każdej z baz mamy wbudowany zestaw widoków systemowych znajdujących się w przestrzeni nazw INFORMATION_SCHEMA. Przedstawiono tutaj wynik dwóch podstawowych zapytań: o listę tabel (Rys. 50) i o listę kolumn (Rys. 51). Jak widać lista

kolumn zwraca kolumny dla całej bazy, dlatego, jeżeli chcemy pozyskać wyniki dla tylko jednej tabeli, musimy filtrować rekordy z użyciem polecenia WHERE.



	TABLE_CATALOG	TABLE_SCHEMA	TABLE_NAME	TABLE_TYPE
1	Test02	dbo	Dept	BASE TABLE
2	Test02	dbo	Employee	BASE TABLE
3	Test02	dbo	vwEmployeeSimple	VIEW

Rysunek 50. Zapytanie zwracajace listę tabel



	TABLE_CATALOG	TAB...	TABLE_NAME	COLUMN_NAME	OR...	COLU...	IS_N...	DATA_...	CHAR...
1	Test02	dbo	Dept	DeptID	1	NULL	NO	int	NULL
2	Test02	dbo	Dept	DeptName	2	NULL	NO	varchar	200
3	Test02	dbo	Employee	EmployeeID	1	NULL	NO	int	NULL
4	Test02	dbo	Employee	Firstname	2	NULL	YES	varchar	50
5	Test02	dbo	Employee	Lastname	3	NULL	YES	varchar	50
6	Test02	dbo	Employee	IsActive	4	((0))	NO	bit	NULL
7	Test02	dbo	Employee	StartDate	5	('1900-...	NO	datetime	NULL
8	Test02	dbo	Employee	Salary	6	((0))	NO	decimal	NULL
9	Test02	dbo	Employee	Kids	7	((0))	NO	int	NULL
10	Test02	dbo	Employee	DeptID	8	((1))	NO	int	NULL
11	Test02	dbo	vwEmployeeSimple	EmployeeID	1	NULL	NO	int	NULL
12	Test02	dbo	vwEmployeeSimple	Firstname	2	NULL	YES	varchar	50
13	Test02	dbo	vwEmployeeSimple	Lastname	3	NULL	YES	varchar	50

Rysunek 51. Zapytanie zwracające listę kolumn

W poniższym zestawieniu przedstawiono widoki dostępne w bazie Microsoft SQL 2005 [6]

- CHECK_CONSTRAINTS
- REFERENTIAL_CONSTRAINTS

- COLUMN_DOMAIN_USAGE
- ROUTINES
- COLUMN_PRIVILEGES
- ROUTINE_COLUMNS
- COLUMNS
- SCHEMATA
- CONSTRAINT_COLUMN_USAGE
- TABLE_CONSTRAINTS
- CONSTRAINT_TABLE_USAGE
- TABLE_PRIVILEGES
- DOMAIN_CONSTRAINTS
- TABLES
- DOMAINS
- VIEW_COLUMN_USAGE
- KEY_COLUMN_USAGE
- VIEW_TABLE_USAGE
- PARAMETERS
- VIEWS

Dużym utrudnieniem w pisaniu tego typu zapytań jest brak faktycznych standardów odnośnie przedstawiania struktury bazy danych. Każdy z producentów SZBD przyjął własne nazewnictwo. Powoduje to, że napisanie generycznego rozwiązanie jest bardzo trudne i pracochłonne. Szerzej napisano o tym w Rozdziale 6.

Jeżeli pobieramy metadane w języku korzystającym z .NET Framework pomocna może okazać się tutaj metoda **SqlDataReader.GetSchemaTable** Metoda ta zwraca obiekt CLR typu DataTable z kolekcją wierszy odpowiadających każdej kolumnie zapytania (Rys 52).

```
foreach (Table table1 in database1.Tables)
{
    SqlCommand comm1 = new SqlCommand("SELECT TOP 1 * FROM "
        + table1.Schema + "." + table1.TableName, conn1);
    SqlDataReader read1 = comm1.ExecuteReader();
    read1.Read();
    DataTable dt = read1.GetSchemaTable();
    read1.Close();
    foreach (DataRow dr in dt.Rows)
    {
        Column column1 = new Column();
        column1.ColumnName = "" + dr["ColumnName"];
        column1.ColumnNameLower = column1.ColumnName.ToLower();
        column1.SqlType = "" + dr["DataTypeName"];
        column1.SqlSize = Convert.ToInt32(dr["ColumnSize"]);
        column1.DataType = "" + dr["DataType"];
        column1.IsPrimary = Convert.ToBoolean(dr["IsIdentity"]);
        column1.IsUniqe = Convert.ToBoolean(dr["IsUnique"]);
        column1.AllowNull = Convert.ToBoolean(dr["AllowDBNull"]);
        table1.Columns.Add(column1);
    }
}
```

Rysunek 52. Wykorzystanie SqlDataReader.GetSchemaTable

Jest to duże ułatwienie ponieważ w przypadku korzystania z wbudowanych widoków systemowych oznaczałoby pobieranie danych z kilku widoków i na pewno nie dałoby się to zrobić w tak elegancki sposób. Należy jednak zaznaczyć, ze dane zwracane w ten sposób są informacjami podstawowymi. Mamy na przykład informacje, że kolumna jest kluczem obcym, ale nie mamy już informacji z jaką tabelą i kolumną się łączy. Te informacje trzeba pobrać w innych sposób, przy pomocy zapytania Select. Warto wspomnieć, że wymieniona tutaj metoda działa tylko dla bazy Microsoft SQL Server. Dla bazy Oracle jest to inna funkcja OracleDataReader.GetSchemaTable. To czy odpowiednik tego mechanizmu będzie dostępny dla innych baz zależy od firm/osób dostarczających sterowniki .NET do nich

5.3. Proces generowania kodu z wykorzystaniem XSLT

Jednym z głównych założeń aplikacji było stworzenie mechanizmu szablonów, który będzie się opierał na istniejących standardach, a co za tym idzie nie będzie wymagał nakładów czasowych na naukę osobnego języka.

Decyzja projektowa w wyborze silnika XSLT został podjęta ponieważ jest to język używany przez wielu programistów na co dzień. Możemy go spotkać na przykład w silnikach do tworzenia stron WWW, raportowaniu ze źródeł danych XML. Pliki XSL będące plikami tekstowymi dają się w prosty sposób modyfikować. Na rynku jest wiele narzędzi zarówno komercyjnych (StyleVision[12]

jak i darmowych (XML Fox [13]) dzięki czemu można edytować pliki w przyjaznym dla użytkownika środowisku.

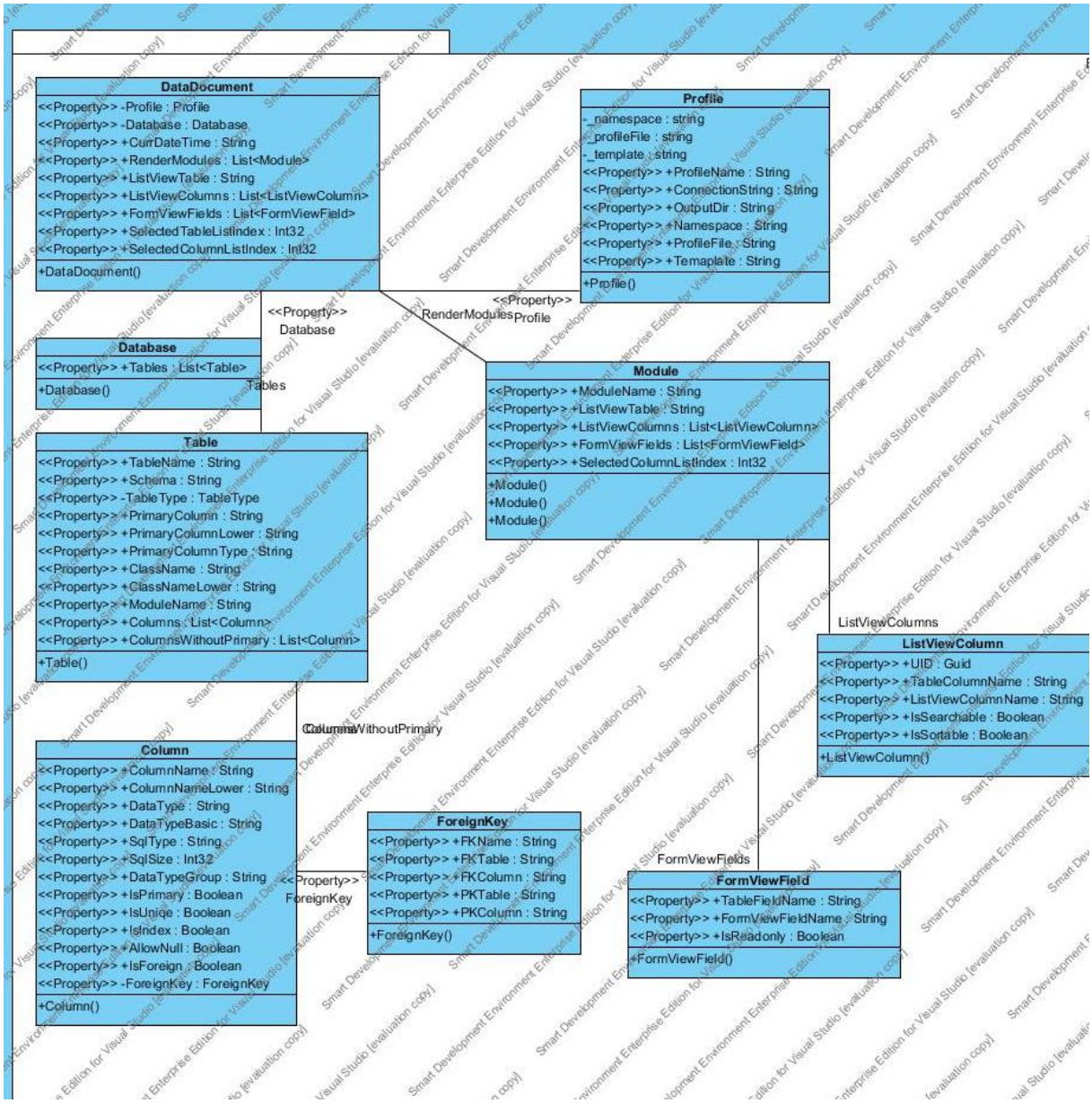
Kolejnym argumentem jest fakt, że XSLT jest standardem, co zawsze podnosi atrakcyjność rozwiązania i gwarantuje , że korzystanie z programu nie zakończy się zamarynowaniem czasu, poświęconego na naukę szablonów i wytworzenie własnych, ponieważ producent zamkniętego formatu zdecydował go daje nie rozwijać.

Warto wspomnieć również, że szukając przykładów w literaturze możemy odnieść się do książki Kathleen Dollard „Code Generation in Microsoft .NET"[5], w której autorka przedstawiła przykład opierający się na transformacji XSLT w celu automatycznego wytwarzania oprogramowania.

Jak wiadomo XSLT do działania potrzebuje plik z danymi XML oraz szablon XSL [10, 11]. Dane z tych dwóch źródeł są parsowane i w wyniku transformacji jest zwracany tekst, kod HTML lub XML.

Pomysł z wykorzystaniem XSLT sprowadza się do utworzenia pliku XML zawierającego strukturę bazy danych, na podstawie których chcemy dokonać generacji. Do pliku trzeba też dołączyć ustawienia wprowadzone przez użytkownika w samym App Generator. Chodzi tutaj o stworzony w programie wygląd modułu, czyli list i formatek.

Przed przystąpieniem do generowania kodu, trzeba utworzyć instancję klasy DataDocument i wypełnić ją potrzebnymi danymi. W tym celu dodajemy kolejne tabele wraz z kolumnami. Dane te zostały pobrane przy podłączaniu się do bazy danych przy starcie programu. Do dokumentu należy również podłączyć aktualne ustawienia (klasa Profile) oraz kolekcję definicji modułów wraz ze stworzonymi ekranami listy rekordów i formatki do edycji danych. Diagram klas został przedstawiony na Rys. 53



Rysunek 53. Diagram klasy DataDocument i połączonych

Następnym krokiem jest zamienienie tych danych do formatu XML.

W tym celu wywołujemy metodę do serializacji danych Rys. 54

```
public static string Serialize(Object obj, Type objType)
{
    StringWriter sw1 = new StringWriter();
    XmlSerializer serial = new XmlSerializer(objType);
    serial.Serialize(sw1, obj);
    return sw1.ToString();
}
```

Rysunek 54. Przykład kodu serializującego

```
<?xml version="1.0" encoding="utf-8" ?>
- <DataDocument xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/20
  - <Profile>
      <ProfileName>Praca magisterska2</ProfileName>
      <ConnectionString>Data Source=.\SQLEXPRESS;Database=Test02;Integrated Security=True;</ConnectionStr
      <OutputDir>D:\Documents and Settings\wpragacz\Desktop\App Generator Output</OutputDir>
      <Namespace>WojtekPragacz.Applikacja01</Namespace>
      <Temaplate>Blue</Temaplate>
    </Profile>
  - <Database>
    - <Tables>
      - <Table>
          <TableName>Dept</TableName>
          <Schema>dbo</Schema>
          <TableType>Table</TableType>
          <PrimaryColumn>DeptID</PrimaryColumn>
          <PrimaryColumnLower>deptid</PrimaryColumnLower>
          <PrimaryColumnType>int</PrimaryColumnType>
          <ClassName>Dept</ClassName>
          <ClassNameLower>dept</ClassNameLower>
          <ModuleName>DeptModule</ModuleName>
        - <Columns>
          - <Column>
              <ColumnName>DeptID</ColumnName>
              <ColumnNameLower>deptid</ColumnNameLower>
              <DataType>System.Int32</DataType>
              <DataTypeBasic>int</DataTypeBasic>
              <SqlType>int</SqlType>
              <SqlSize>4</SqlSize>
              <DataTypeGroup>numeric</DataTypeGroup>
              <IsPrimary>true</IsPrimary>
              <IsUniqe>false</IsUniqe>
              <IsIndex>true</IsIndex>
              <AllowNull>false</AllowNull>
              <IsForeign>false</IsForeign>
            </Column>
          - <Column>
              <ColumnName>DeptName</ColumnName>
              <ColumnNameLower>deptname</ColumnNameLower>
              <DataType>System.String</DataType>
              <DataTypeBasic>string</DataTypeBasic>
              <SqlType>varchar</SqlType>
              <SqlSize>200</SqlSize>
              <DataTypeGroup>text</DataTypeGroup>
              <IsPrimary>false</IsPrimary>
              <IsUniqe>false</IsUniqe>
              <IsIndex>false</IsIndex>
              <AllowNull>false</AllowNull>
              <IsForeign>false</IsForeign>
            </Column>
```

Rysunek 55. Wygenerowany dokument XML

Na wyjściu dostajemy dokument XML zawierający wprowadzone wcześniej dane (Rys 55).

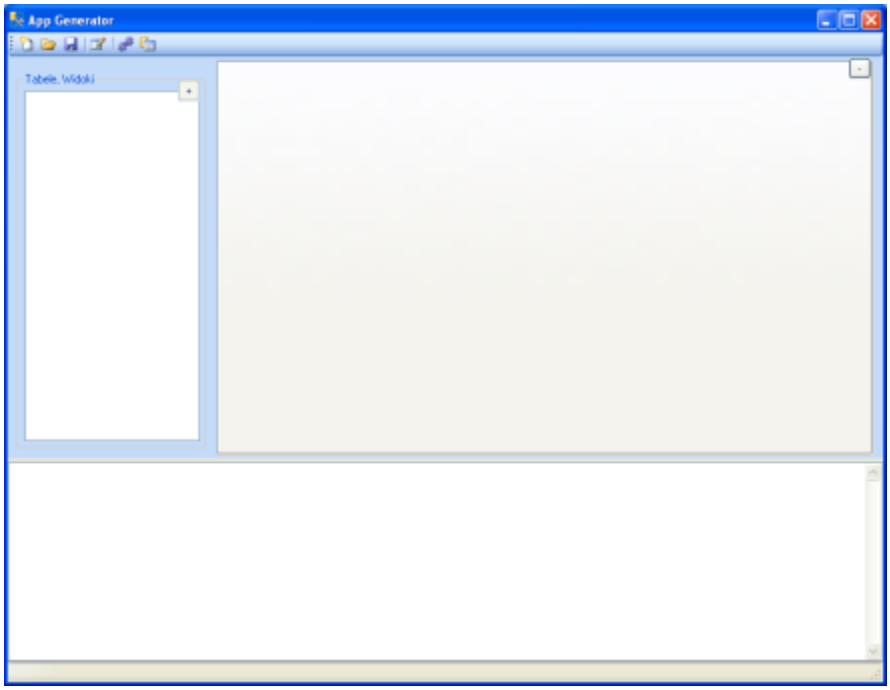
```
private static string transform(string xmlDoc, string xslPath,
                                                XsltArgumentList xslArgs)
{
    xslPath=Path.Combine(Application.StartupPath, xslPath);
    // parsing data & template
    XPathDocument xpathDoc = new XPathDocument(new StringReader(xmlDoc));
    XslCompiledTransform myXslTrans = new XslCompiledTransform();
    myXslTrans.Load(xslPath);
    StringWriter sw1 = new StringWriter();
    myXslTrans.Transform(xpathDoc, xslArgs, sw1);
    return sw1.ToString();
}
```

Rysunek 56. Przykład wywołania procesora XSLT

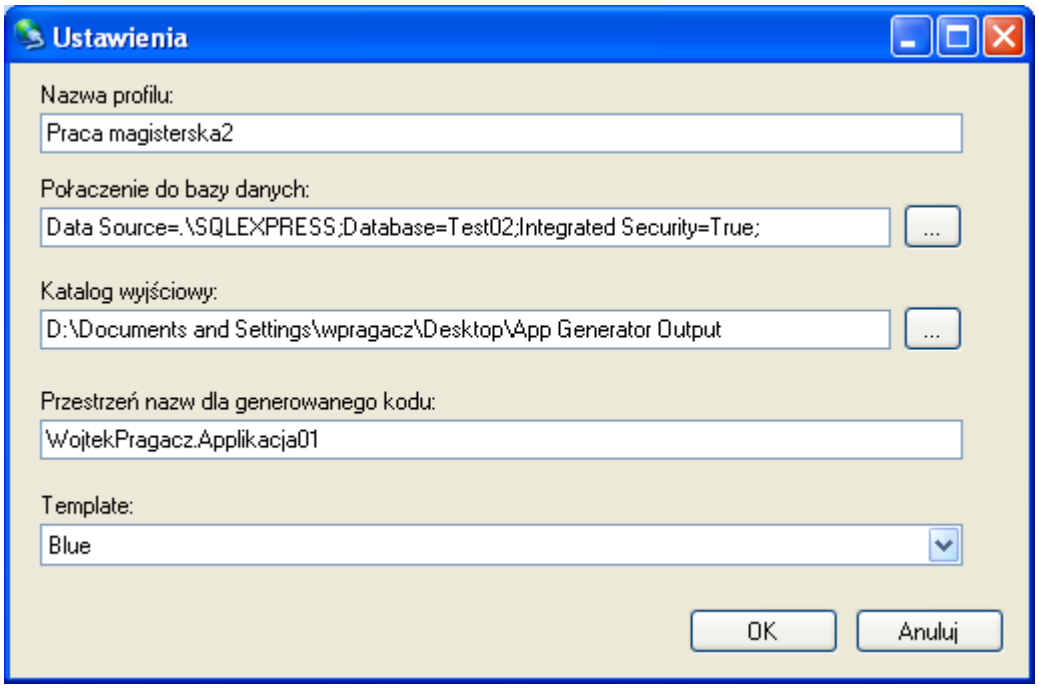
Do transformacji wykorzystano wbudowany parser - klasę XslCompiledTransform z pakietu System.Xml.Xsl. Dla kolejnych plików XSL z katalogu szablonów wywoływana jest transformacja XSLT z utworzonym plikiem XML (Rys. 56). Wynik transformacji jest zapisywany w odpowiadającym pliku. W poniższej tabelce znajduje się zestawienie szablonów:

XSL template	**Przykładowy wygenerowany plik**	**Warstwa**
edit_aspx.xsl	Edit.aspx	prezentacji
edit_aspx_back.xsl	Edit.aspx.cs	prezentacji
edit_aspx_back_design.xsl	Edit.aspx.designer.cs	prezentacji
list_aspx.xsl	List.aspx	prezentacji
list_aspx_back.xsl	List.aspx.cs	prezentacji
list_aspx_back_design.xsl	List.aspx.designer.cs	prezentacji
manager.xsl	EmployeeManager.cs	logiki biznesowej
entity.xsl	Employee.cs	dostępu do danych
hbm.xsl	Employee.hbm.xml	dostępu do danych

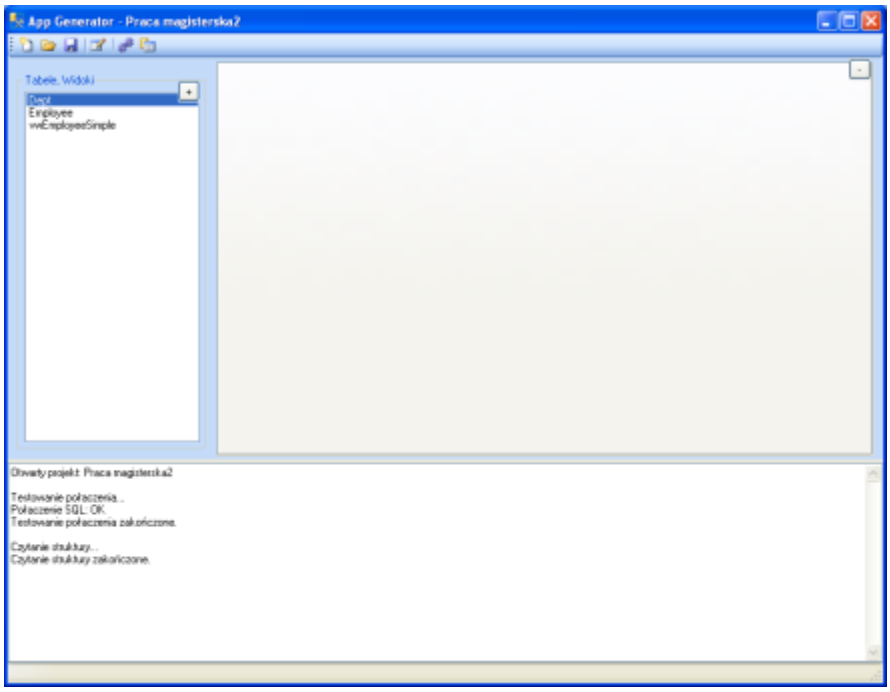
5.4. Przykład pracy z programem

Otwórz program **App Generator**

Kliknij ikonkę **Nowy**

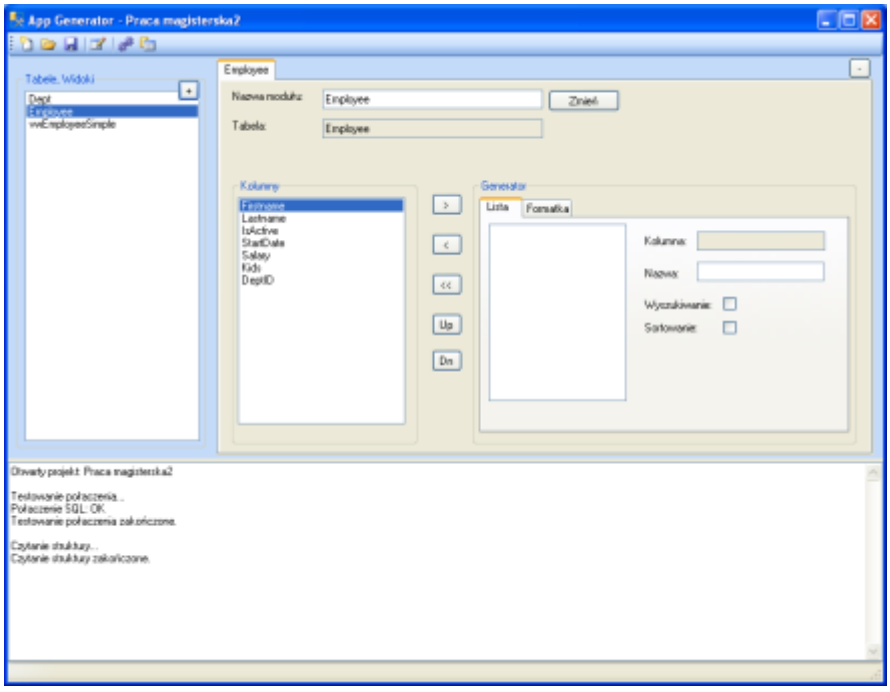
Wypełnij ustawienia projektu



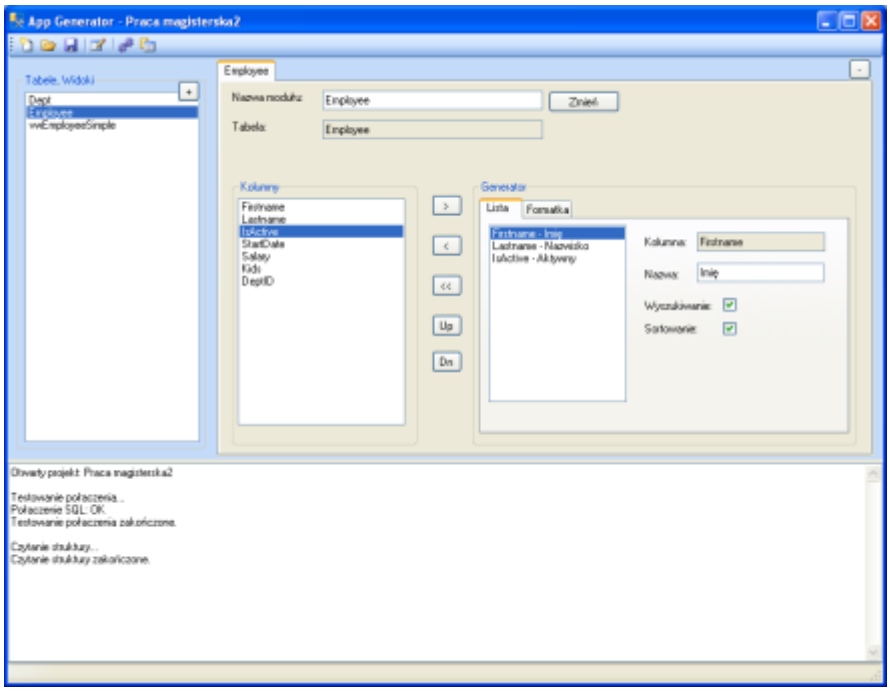
Kliknij **OK**



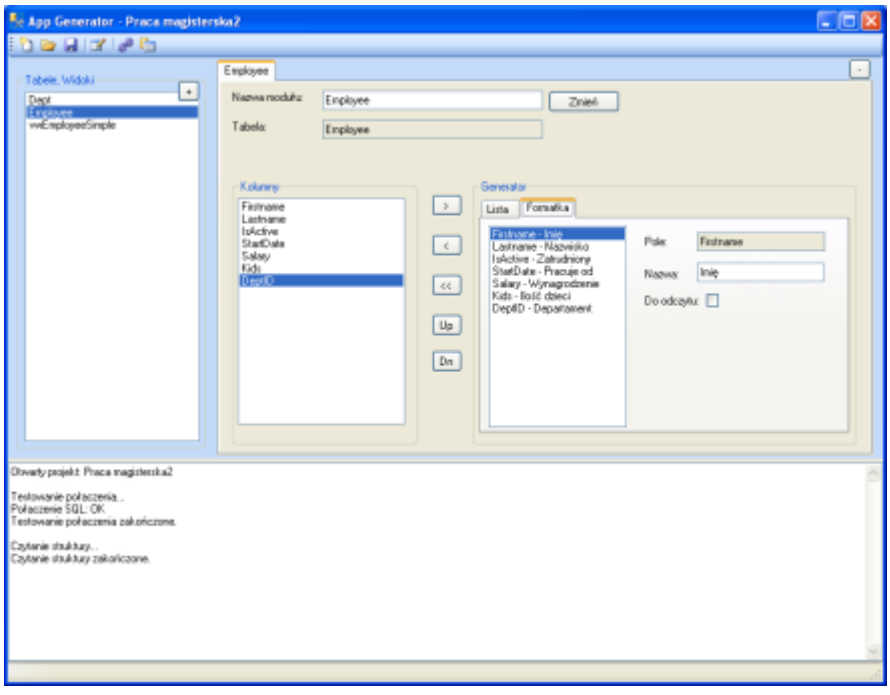
Wybierz tabelę **Employee** i kliknij przycisk **+**



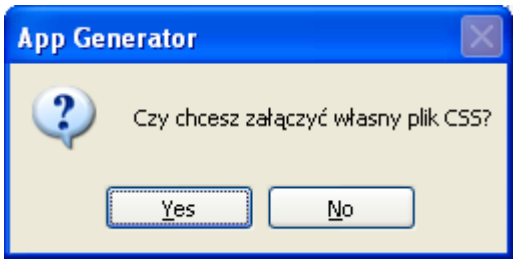
Przenieś kolumny **Firstname, Lastname oraz IsActive** do zakładki **Lista**, pozmieniaj im nazwy na polskie odpowiedniki



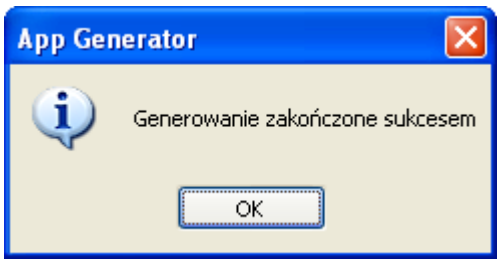
Przenieś wszystkie kolumny do zakładki **Formatka**, pozmieniaj im nazwy na polskie odpowiedniki



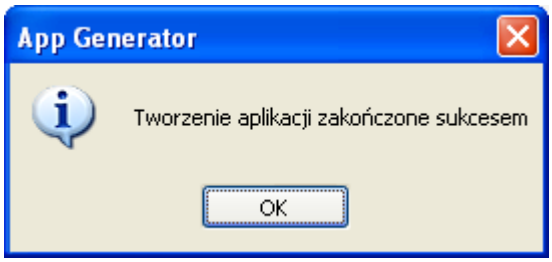
Kliknij ikonkę generowania pliakcji  (pierwsza o prawej)



Nie

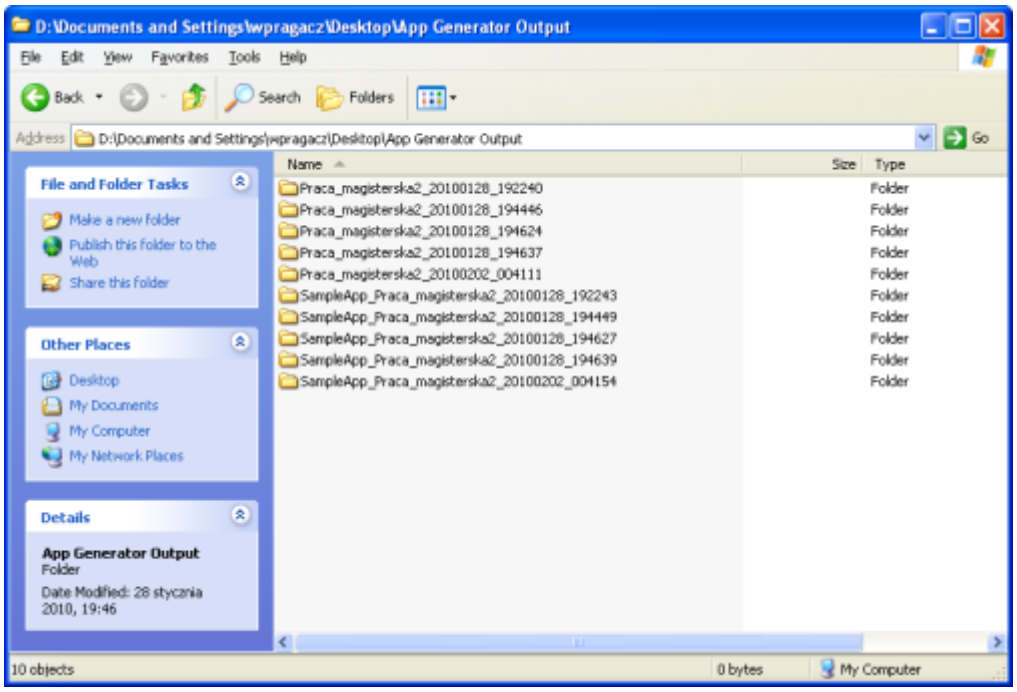


OK

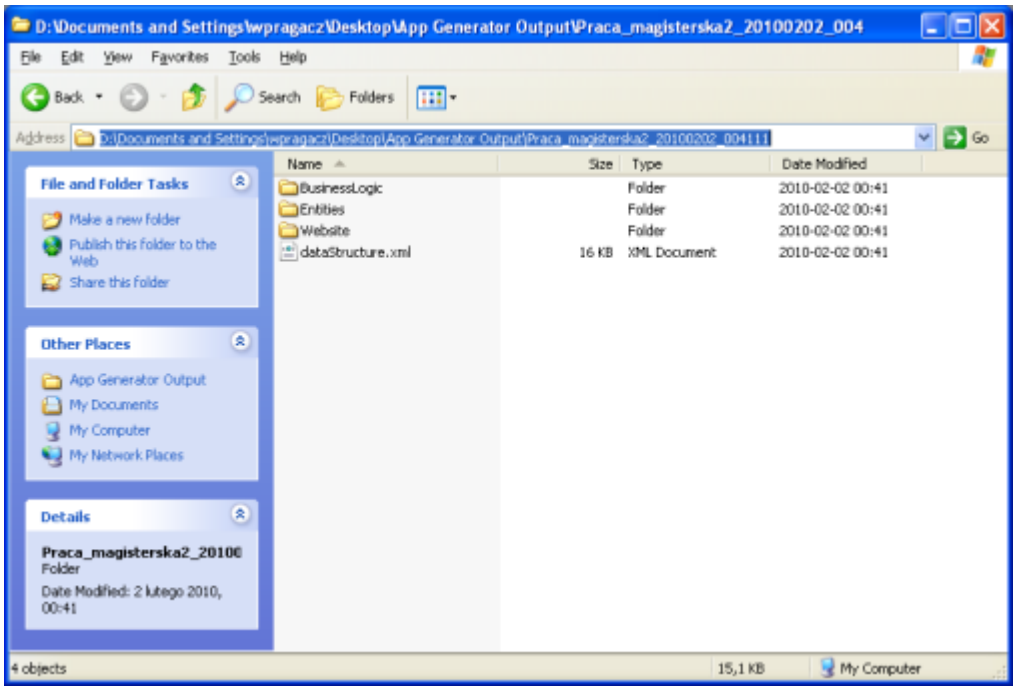


OK

Przejdź do wpisanego w ustawieniach katalogu, w tym przypadku będzie to D:\Documents and Settings\wpragacz\Desktop\App Generator Output\



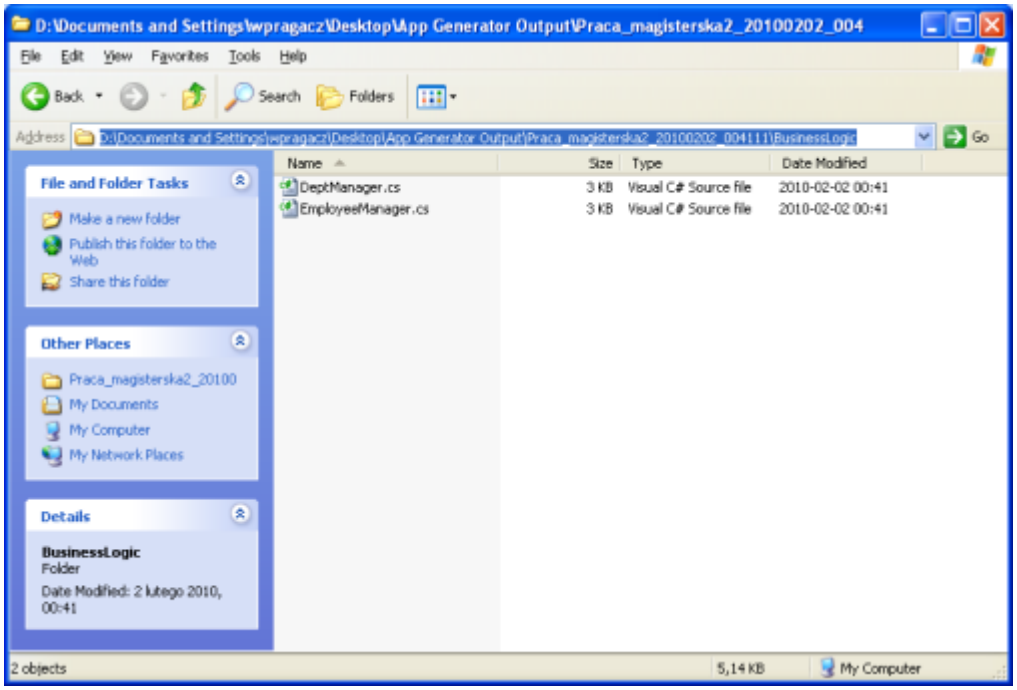
Wejdź do katalogu Praca_magisterska2_*

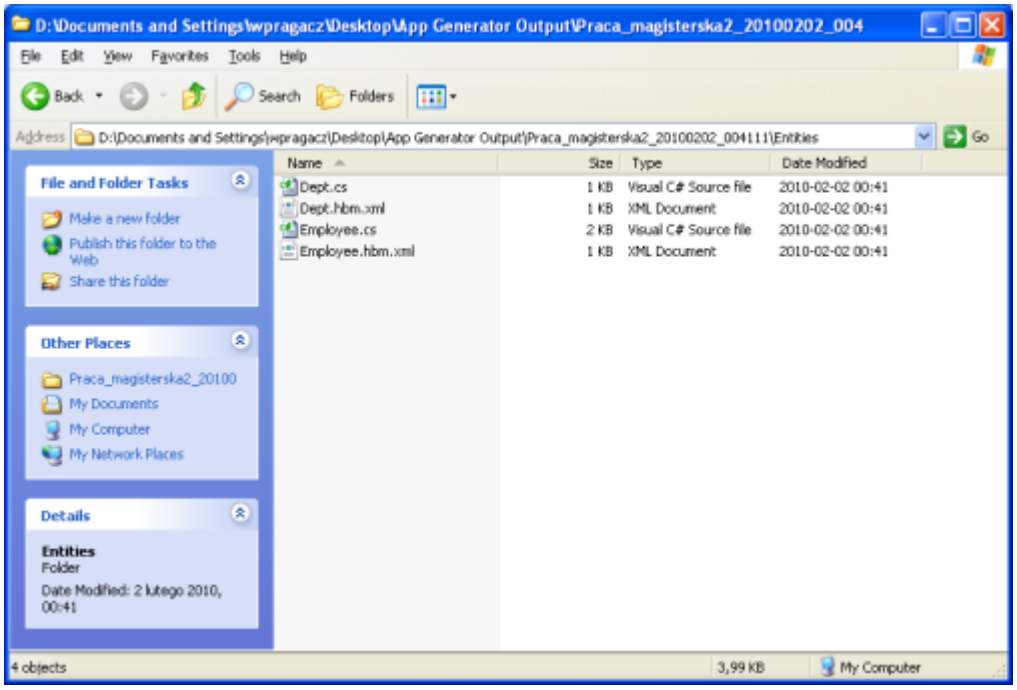


Znajduje się tutaj wygenerowany kod, który może zostać załączony do dowolnej aplikacji.

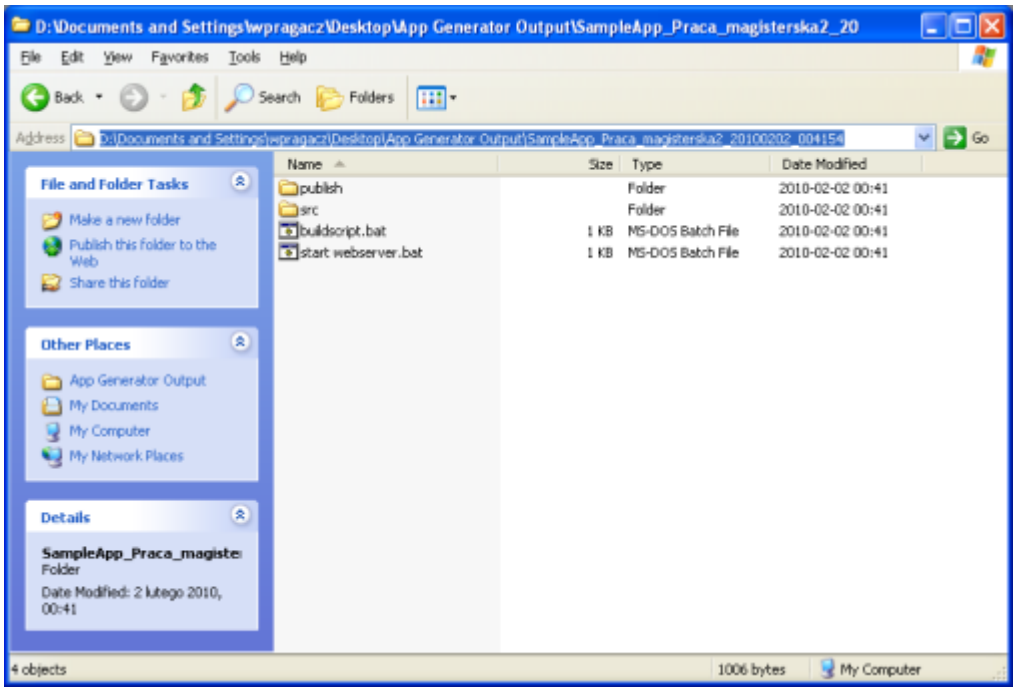
- Business logic – Warstwa logiki biznesowej (Service Layer)
- Entities – klasy encji reprezentujących byty w bazie danych oraz pliki konfiguracyjne NHibernata
- Website - Warstwa prezentacji danych (User Interface)

Ponieważ w tabeli Employee wystepuje klucz obcy DeptID z automatu tworzone są również odpowiednie klasy Business Logic i Entities dla tabeli Dept, które będą nam potrzebne do wyświetlenia list rekordów na formatce.

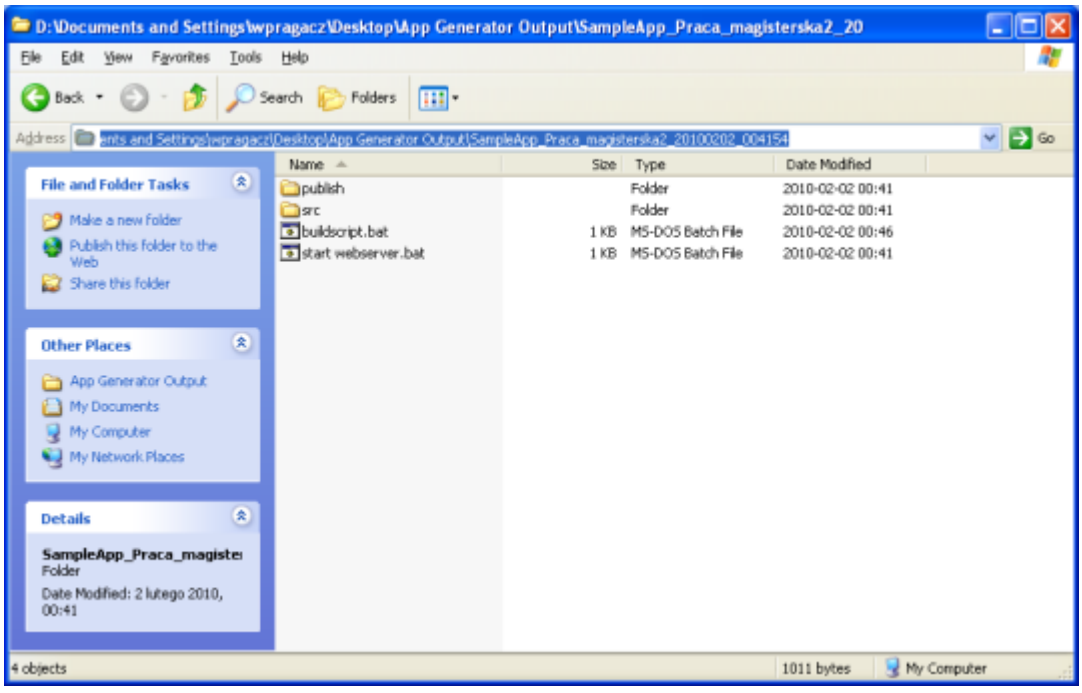




D:\Documents and Settings\wpragacz\Desktop\App Generator Output\

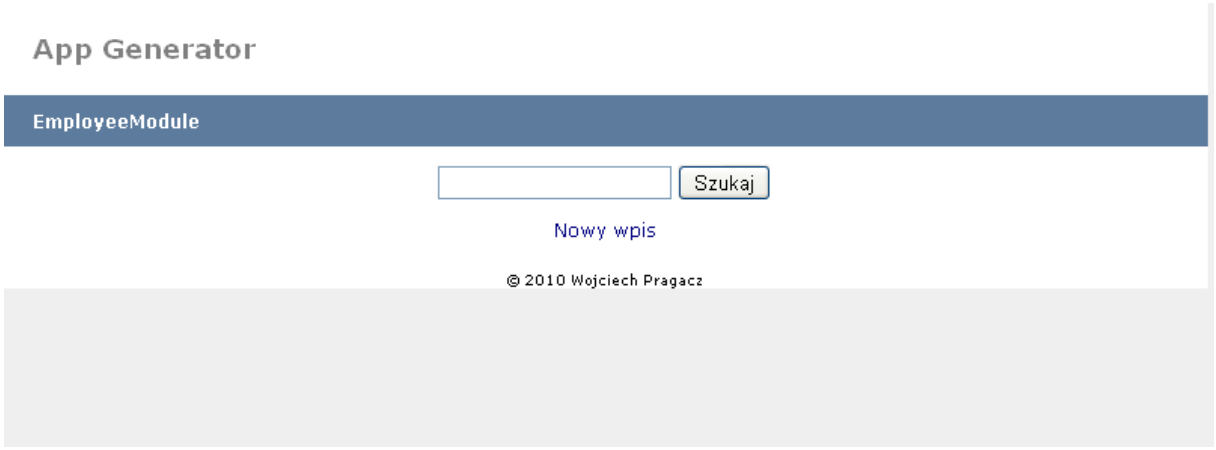


Przejdź do katalogu **SampleApp_Praca_magisterska2_****

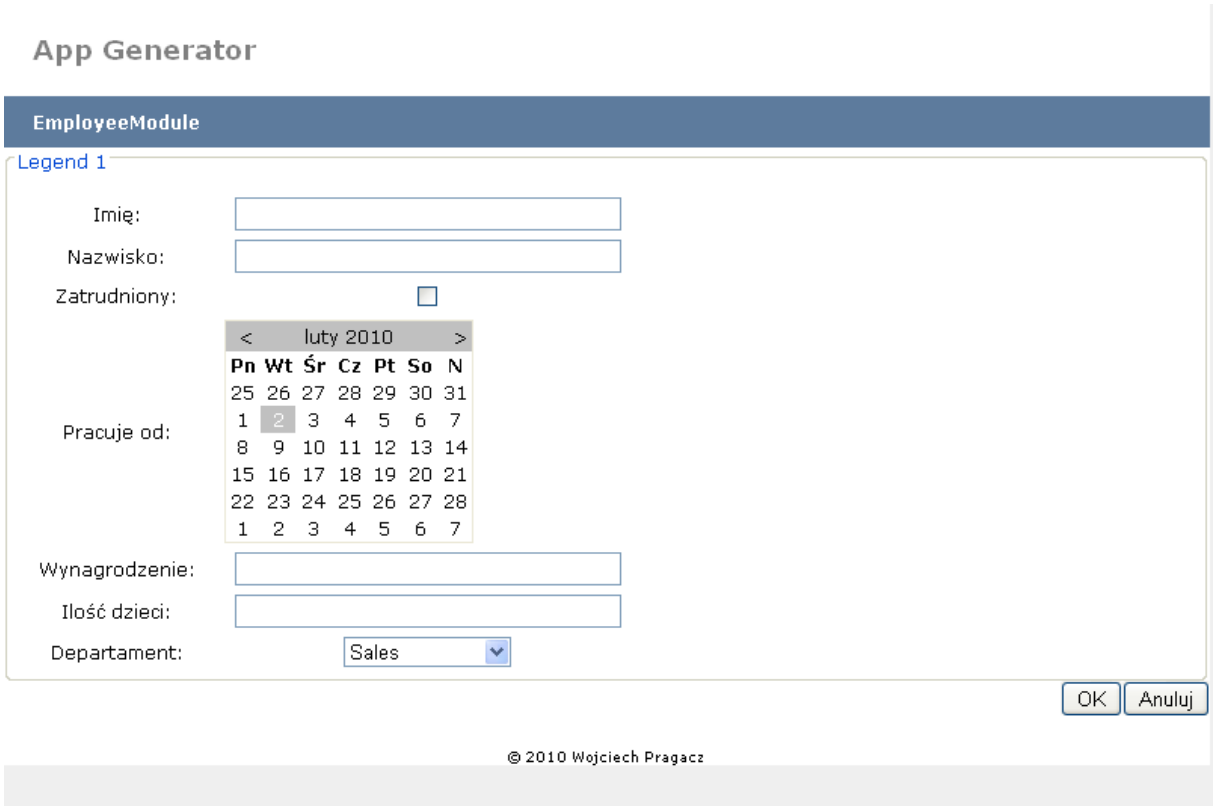


Uruchom skrypt **start webserver.bat**

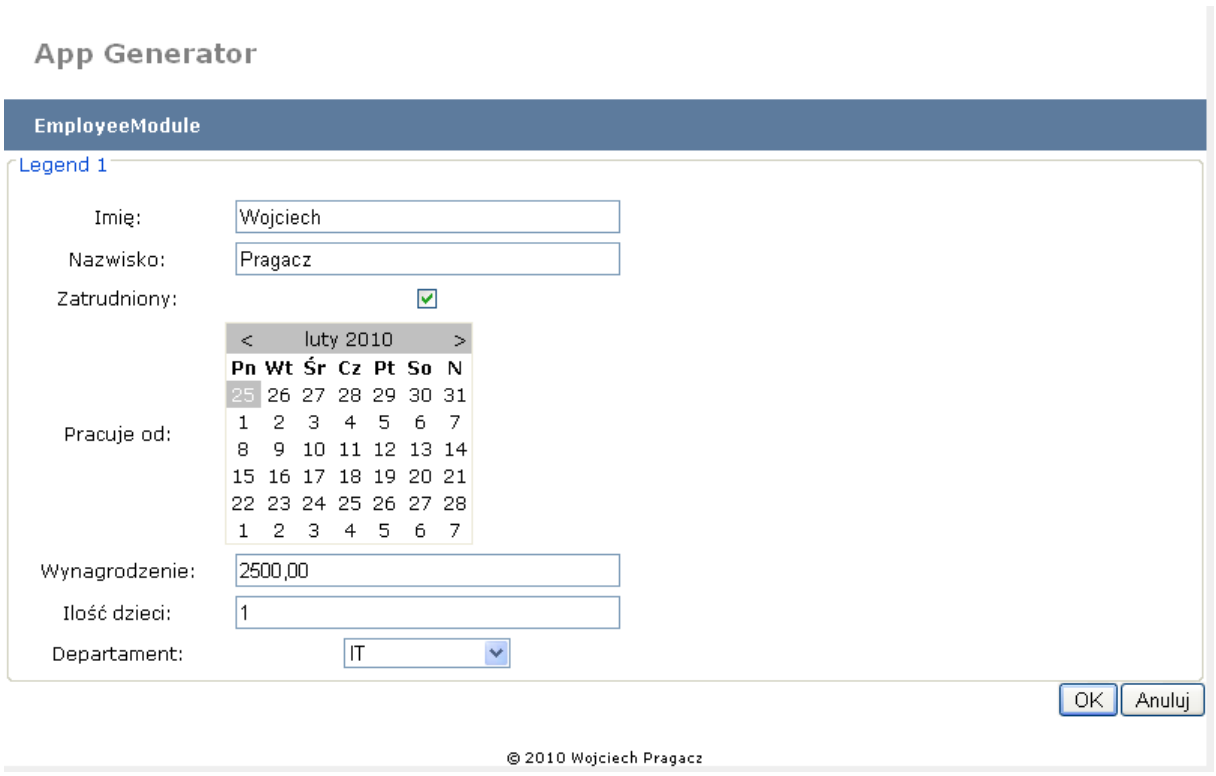
Uruchom przeglądarkę internetową i przejdź do adresu: **http://localhost:8080/**



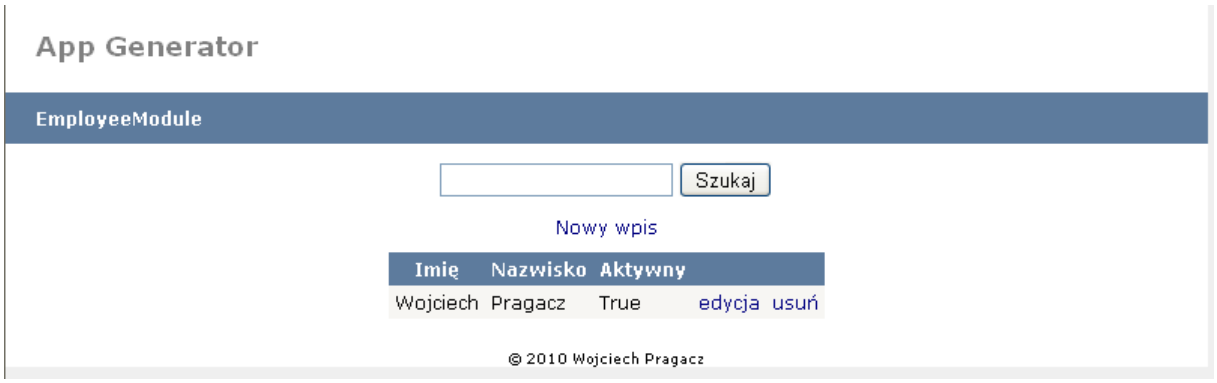
Kliknij **Nowy wpis**



Wypełnij pola danymi testowymi



Kliknij **OK**



Zakończ pracę serwera WWW klikając prawym przyciskiem myszki na **STOP**

6. Problemy związane z generowaniem kodu na podstawie struktury bazy danych i możliwe ulepszenia

W tym rozdziale zostaną przedstawione problemy, jakie pojawiły się podczas prac nad implementacją pracy dyplomowej. Autor przedstawił również znalezione rozwiązania dla niektórych z wymienionych problemów. Na koniec przedstawiono możliwe dalsze drogi rozwoju dla tej aplikacji.

6.1. Perspektywy systemowe nie są ustandaryzowane

Sporym utrudnieniem przy tworzeniu koncepcji tej aplikacji był brak jakiegokolwiek standardu dotyczącego pobierania schematu bazy danych. Jako przykład mogą posłużyć zapytania do różnych rodzajów baz zwracające listę tabel:

Microsoft SQL [6]

```
SELECT * FROM INFORMATION_SCHEMA.TABLES
```

Oracle [14]

```
SELECT * FROM ALL_TABLES ;
```

MySQL [15]

```
SHOW FULL TABLES [FROM db_name]
```

PostgreSQL [16]

```
SELECT * FROM pg_tables
```

Jak widać każdy producent przyjął w tym temacie indywidualne nazewnictwo, dlatego praktycznie dla każdego z silników baz danych trzeba pisać osobny moduł zaczytujący strukturę. Na podstawie powyższych wniosków Autor podjął decyzję o zaimplementowaniu tylko obsługi SQL Server.

6.2. Formatowanie kodu

Jednym z głównych problemów do rozwiązania było z pozoru błahe formatowanie kodu. Szablony używane w projekcie były napisane w technologii XML/XSL. Wcięcia i formatowanie były narzucane przez arkusz XSL i nie były zgodne z zaleceniami odnośnie formatowania kodu w C#. Na

przykład zastosowanie XSL -owych poleceń <xsl:for-each> lub <xsl:if> dodawało dodatkowe puste linie lub białe znaki (tabulacje, spacje) (Rys.57).

```
public string ObjectAsString
{
  get {
    string str = "";
    <xsl:for-each select="Columns/Column">
        <xsl:if test="DataTypeBasic='string'">
          str += " " + this.<xsl:value-of select="ColumnNameLower"/>;
        </xsl:if>
      </xsl:for-each>
      return str;
  }
}
```

Rysunek 57. Znaczniki powodujące puste linie na wyjściu

Był to problem w dużej mierze estetyczny, jednak zakładając, że produktem końcowym jest kod, musiał on być dostarczony w czytelnej formie bez potrzeby dalszych obróbek. Problem ten dotyczył głownie plików z kodem C# (*.cs) i nie był tak dokuczliwy w przypadku plików HTML czy XML. W tym celu zaimplementowano klasę CodeAutoFormatHelper, której funcjonalność została przedstawiona poniżej.

```
public static string FormatCode(string codeIn)
{
    string str;
    StringBuilder codeOut;
    int tabsCount;
    tabsCount = 0;
    codeOut = new StringBuilder();
    foreach (string line in codeIn.Split(Environment.NewLine.ToCharArray(),
        StringSplitOptions.RemoveEmptyEntries))
    {
        str = line.Trim();
        if (str.IndexOf("#NEWLINE#")>=0)
        {
            codeOut.Append(new string('\t', tabsCount));
            codeOut.Append("\r\n");
        }
        else if (str.Trim().Length > 0)
        {
            codeOut.Append(new string('\t',tabsCount));
            codeOut.Append(str).Append(Environment.NewLine);
            tabsCount=brackets(line, tabsCount);
        }
    }
    return codeOut.ToString();
}
```

Rysunek 58. Metoda FormatCode

Głowna metoda FormatCode (Rys.58) analizuje podawany na wejście kod C# i formatuje go poprawnie zgodnie z dobrymi praktykami Microsoftu. Usuwane są wszelkie puste linie, które powstają na przykład podczas implementacji pętli <xsl:for-each>. Może się jednak zdarzyć, że w jakimś miejscu będziemy chcieli wymusić jedną lub kilka pustych linii. W tym celu został dodany dotykowy znacznik #NEWLINE#. Zostaje on podczas formatowania zamieniony na nową linię.

Formatowanie samego kodu opiera się na zliczaniu znaków otwierających i zamykających bloki kodu { oraz }. Dla każdego otwarcia bloku ({) licznik zwiększa się, a dla każdego zamknięcia bloku kodu (}) licznik się zmniejsza (Rys. 59). Linia kodu jest formatowana w ten sposób, że z przodu są usuwane wszystkie białe znaki, a w ich miejsce są wstawiane tabulacje, których ilość odpowiada licznikowi.

```
private static int brackets(string line, int tabsCount)
{
    int openBrackets=0,closeBrackets=0;
    foreach (char ch in line.ToCharArray())
    {
        if (ch == '{') openBrackets++;
        if (ch == '}') closeBrackets++;
    }
    tabsCount = tabsCount + openBrackets - closeBrackets;
    return tabsCount;
}
```

Rysunek 59. Zliczanie znaków { i }

6.3. Praca z plikami HTML, ASP.NET czy XML

Ze względu na różnorodność generowania plików (C#, HTML, ASPX, XML) jako obowiązujący format wyjściowy transformacji XSLT przyjęto wartość „text”. Miało to swoje konsekwencje przy pracy z plikami ASPX. W celu zapewnienia poprawnego tworzenia zawartości plików, wszelkie znaki specjalne występujące w XML czyli < oraz > trzeba było zmieniać na ich odpowiedniki < i >. Dodatkowo atrybuty HTML, które zawierały wewnątrz polecenia transformacji XSLT, trzeba było otoczyć specjalnymi znakami " odpowiadającymi cudzysłowom. Stworzony w ten sposób szablon może być mało czytelny dla początkujących lub mniej wprawionych osób, dlatego jest to pole do ulepszeń na przyszłość (Rys 60).

```
<xsl:if test="$DataTypeGroup='text' or $DataTypeGroup='numeric'">
    &lt;asp:TextBox ID=&quot;txt<xsl:value-of select="TableFieldName"/>&quot; runat="server" Wid
    &lt;asp:RequiredFieldValidator ID=&quot;validRF<xsl:value-of select="TableFieldName"/>&quot;
            Display="Dynamic" ErrorMessage="Pole wymagane"&gt;&lt;/asp:RequiredFieldValidator&gt;
</xsl:if>
<xsl:if test="$DataTypeGroup='bool'">
  &lt;asp:CheckBox ID=&quot;chk<xsl:value-of select="TableFieldName"/>&quot; runat="server" &gt;
</xsl:if>
<xsl:if test="$DataTypeGroup='date'">
  &lt;asp:Calendar ID=&quot;calend<xsl:value-of select="TableFieldName"/>&quot; runat="server" &
</xsl:if>
  </xsl:otherwise>
```

Rysunek 60. Mało czytelny kod XSL

6.4. Perspektywy nie są zawsze aktualizowalne

Aplikacja pozwala na tworzenie nowych modułów nie tylko na podstawie tabel, ale także na podstawie widoków. Powstaje tutaj problem, ponieważ nie każdy widok jest aktualizowany. Wszystko zależy od tego czy w tabelach źródłowych zostały odpowiednio zdefiniowane domyślne wartości. Nie znaleziono, żadnej skutecznej metody poza próbą aktualizacji takiego widoku, aby stwierdzić, czy widok będzie aktualizowany czy nie.

6.5. Możliwe kierunki rozwoju generatora

Utworzony podczas pracy magisterskiej program obsługuje schemat metadanych bazy danych Microsoft. Ciekawym kierunkiem rozwoju byłoby dodanie obsługi innych baz danych. W tym celu należałoby rozbudować moduł systemu czytający strukturę bazy danych, aby w zależności od kontekstu i rodzaju bazy używał różnych metod do pobierania schematu obiektów z bazy. Trudności związane z tym zagadnieniem zostały opisane we wcześniejszym podrozdziale „6.1 Perspektywy systemowe nie są ustandaryzowane”

Jeżeli chodzi o rozwiązania wyjściowe sprawa jest o wiele prostsza. Praktycznie nie ma większych ograniczeń do stworzenia szablonów tworzących kod w innym języku niż C#.

Autor widzi tutaj szczególne zastosowanie dla Java EE oraz ostatnio popularnego Spring Framework. Systemy budowane na wymienionych platformach przeważnie opierają się właśnie na modelu architektury trójwarstwowej. Istnieje również możliwość tworzenia kodu aplikacji pisanych w PHP lub ROR, jednak benefit wynikający z pracy z programem jest tym większy im więcej jest plików do utworzenia.

W celu przygotowania wersji tworzącej aplikację w konwencji CRUD w technologii Spring Framework z obsługa bazy Microsoft SQL, należałoby zmodyfikować proces kompilacji w taki sposób, aby korzystał on z programu budującego Ant, a nie z kompilatora CSC. Dodatkową zmianą byłoby wybieranie w zależności od docelowej platformy pliku szablonu strony HTML. Używane obecnie pliki Masterpages nie będą współpracowały z technologiami JSF lub Spring MVC

7. Podsumowanie

Autor pracy postawił sobie za cel stworzenie narzędzia odciążającego programistów tworzących aplikacje biznesowe od codziennych powtarzających się zadań, na przykład wygenerowanie aplikacji w konwencji CRUD na podstawie wybranej tabeli z bazy danych.

W pracy został opisany sposób generowania kodu na podstawie struktury (metadanych) bazy danych z użyciem szablonów XML/XSL. W ramach pracy została zaimplementowana aplikacja dająca możliwość wygenerowania pełnego 3 warstwowego systemu ASP.NET lub jedynie samego kodu C# dla poszczególnych warstw, które następnie można dołączyć do już istniejącego projektu. W chwili obecnej rozwiązanie zostało oparte na modelu bazy Microsoft SQL Server i umożliwia generowanie kodu na podstawie tego dostawcy.

Dalsze możliwe zmiany obejmują dodanie obsługi baz Oracle oraz na przykład MySQL. Dzięki podjętym decyzjom architektonicznym, po wprowadzeniu drobnych zmian implementacyjnych, automatyczne tworzenie kodu dla innych platform i języków programistycznych (np. Java EE) jest już dzisiaj możliwe, jednak trzeba by było przygotować odpowiednie wzorce na potwierdzenie tej tezy i ułatwienie startu z ta aplikacją zainteresowanym osobom.

Autor pracy ma nadzieje, że praca z użyciem tego narzędzia przełoży się bezpośrednio na zwiększenie produktywności programistów i pozwoli im się skupić na bardziej wymagających zadaniach , a nie będących zwykłym powielaniem schematów.

Bibliografia

[1]. http://www.codesmithtools.com/features/overview.aspx

[2]. http://www.mygenerationsoftware.com/portal/Documentation/Mygeneration

[3] Code Generation in Action (Paperback), Jack Herrington

[4] Biblioteka senseGUI czyli GUI z automatu, Mariusz Trzaska, Software Developer's Journal 1/2009

[5] Code Generation in Microsoft .NET, Kathleen Dollard

[6] SQL Server Books Online, http://msdn.microsoft.com/en-us/library/ms130214.aspx

[7] http://knol.google.com/k/fabio-maulo/nhibernate-chapter-2/1nr4enxv3dpeq/6#

[8] http://en.wikipedia.org/wiki/.NET_Framework

[9] http://pl.wikipedia.org/wiki/.NET_Framework

[10] http://pl.wikipedia.org/wiki/XSL_Transformations

[11] http://www.w3.org/TR/xslt

[12] http://www.altova.com/stylevision.html

[13] http://www.xmlfox.com/

[14] http://www.oracle.com/technology/documentation/database.html

[15] http://dev.mysql.com/doc/refman/5.0/en/show-tables.html

[16] http://www.postgresql.org/docs/

Dodatki

Dodatek A: Słownik użytej terminologii i skrótów

SQL - Structured Query Language

XML - Extendible Markup Language

XSL - Extensible Stylesheet Language, funkcyjny język programowania opisujący sposób prezentacji i przekształceń dokumentów zapisanych w formacie XML

.NET Framework - platforma programistyczna firmy Microsoft

ASP.NET – technologia będąca częścią .NET Framework, służąca tworzeniu dynamicznych stron WWW od firmy Microsoft

Visual Basic .NET – język programowania wchodzący w skład .NET Framework

C# – obiektowy język programowania wchodzący w skład .NET Framework

JScript .NET – obiektowy język programowania wchodzący w skład .NET Framework, powstał na podstawie języka JavaSript

JavaScript – obiektowy język programowania, dzisiaj używany głównie do oprogramowania akcji po stronie klienta w aplikacjach i stronach www

WSS - Windows SharePoint Services, platforma portalowa firmy Microsoft

CRUD – Create, Read, Update and Delete, konwencja podejścia do przeglądania i modyfikacji danych

POCO – Plain Old CLR Objects

SZBD – System zarządzania bazą danych