STUDIA INFORMATICA 2012
Volume 33 Number 2B (106)

Artur KORNATKA
Wyższa Szkoła Biznesu – National-Louis University w Nowym Sączu, Wydział Informatyki,
Uniwersytet Marii Curie-Skłodowskiej w Lublinie, Instytut Informatyki

# PROJEKT I KONSTRUKCJA SYSTEMU GENERUJĄCEGO ELEMENTY BAZODANOWYCH APLIKACJI BIZNESOWYCH

**Streszczenie**. W pracy przedstawiono ideę autorskiego systemu generującego elementy bazodanowych aplikacji biznesowych oraz szczegółową prezentację architektury i sposobu działania prototypu generatora o nazwie BACG (Business Application Code Generator), realizującego przedstawione koncepcje.

**Słowa kluczowe**: aplikacje bazodanowe, Model View ViewModel, WPF, automatyczne generowanie składników aplikacji

# DESIGN AND ARCHITECTURE OF THE SYSTEM FOR GENERATING ELEMENTS OF DATABASE BUSINESS APPLICATION

**Summary**. In the paper we put forward the idea of a system for generating elements for database business applications and presents details of the architecture and way of working of the generator prototype called BACG (Business Application Code Generator) which implements mentioned concepts.

**Keywords**: database application, Model View ViewModel, WPF, automatic generation of application elements

## 1. Wstęp

Klasyczny model cyklu życia oprogramowania, zwany cyklem kaskadowym [2, 3], wyróżnia kolejne etapy obejmujące: planowanie, określenie wymagań, projektowanie, implementację, testowanie oraz eksploatację z konserwacją.

Implementację aplikacji biznesowych, realizowaną za pomocą najnowocześniejszych narzędzi informatycznych, można podzielić na kilka zadań. W podejściu „database first" cykl wy-

twórczy rozpoczyna tworzenie i programowanie bazy danych. Wynikiem tego jest konstrukcja i umieszenie na serwerze bazy danych stosownych tabel i powiązań między nimi oraz powstanie licznych zapytań, procedur i funkcji realizujących operacje na tworzonej bazie.
Jednym z kluczowych zadań jest programowanie logiki biznesowej aplikacji. Podczas tego etapu powstaje zazwyczaj zestaw klas z rozbudowanymi funkcjami, które pozwalają zrealizować biznesowe cele budowanego systemu. Istotne jest też tworzenie elementów wizualnych, umożliwiających realizację tych funkcji. Podczas tego etapu powstają liczne formularze pozwalające przyszłemu użytkownikowi aplikacji w sposób efektywny wykonywać te operacje.

Do tworzenia aplikacji biznesowych powszechnie stosowana jest architektura trójwarstwowa. Zakłada ona, że aplikacje składają się z warstwy prezentacyjnej, zależnej od warstwy logiki biznesowej, która w sposób bezpośredni zależy od warstwy dostępu do danych.

Najważniejszym czynnikiem decydującym o koszcie wytworzenia oprogramowania jest wysokość nakładów na pracę programistów. Zasadniczej redukcji tych kosztów sprzyja wykorzystanie wyspecjalizowanych systemów, które znacznie przyśpieszą proces powstawania oprogramowania i możliwie szeroko zastąpią programistę, czyli systemów, które pozwolą na automatyzacje wybranych etapów implementacji aplikacji biznesowych. Używanie tak zaprojektowanych i optymalnie działających innowacyjnych produktów informatycznych, przez firmy zajmujące się wytwarzaniem aplikacji biznesowych, jest kluczowym czynnikiem decydującym o przewadze tych firm na rynku.

Problem tworzenia generatorów aplikacji już od dawna poruszany jest w licznych publikacjach, przykładem takich prac mogą być [9] lub [10].

Zasadniczym celem pracy jest prezentacja idei budowy systemu generującego elementy aplikacji biznesowych oraz przedstawienie architektury i sposobu działania prototypu innowacyjnego generatora o nazwie BACG (Business Application Code Generator), realizującego przedstawione koncepcje.

Podpunkt drugi zawiera podstawy teoretyczne wzorca projektowego MVVM.

W podpunktach trzecim i czwartym przedstawiono architekturę i zasady działania prototypu BACG.

W podpunktach piątym, szóstym i siódmym szczegółowo opisano proces generowania kolejnych warstw aplikacji biznesowej.

Na podstawie przeprowadzonych badań, w podpunkcie ósmym zebrano najważniejsze cechy systemu BACG oraz przedstawiono kierunki dalszego rozwoju systemu.

## 2. Podstawy teoretyczne – wzorzec projektowy MVVM

Jak podaje [1], głównym zadaniem wzorca projektowego MVVM (Model-View-View-Model) jest oddzielenie od siebie trzech podstawowych warstw aplikacji. Wydzielonymi warstwami są tu: View, Model, ViewModel.

Warstwa View stanowi część prezentacyjną. View jest graficzną kontrolką lub zbiorem kontrolek odpowiedzialnych za przedstawianie danych modelu na ekranie. Widok może być oknem aplikacji lub kontrolką użytkownika (UserControl), którą można umieścić na dowolnym oknie aplikacji.

Warstwa Model opisuje logikę biznesową aplikacji. Składa się z wielu obiektów biznesowych realizujących konkretne cele aplikacji.

Warstwa ViewModel stanowi swoisty łącznik między modelem a widokiem. Głównym zadaniem tej warstwy jest pobranie ze źródła odpowiednich danych, a następnie przekształcenie ich do postaci charakterystycznej dla danego widoku lub przekazanie z widoku odpowiednio zmodyfikowanych informacji do źródła danych.

Więcej informacji o implementacji MVVM można znaleźć w [7].

Opisany wyżej wzorzec projektowy ma idealne zastosowanie przy tworzeniu interfejsów użytkownika za pomocą technologii WPF (Windows Presentation Foundation), czyli najnowocześniejszej technologii odpowiedzialnej za budowanie atrakcyjnych wizualnie okienek desktopowych aplikacji użytkowych.

Więcej informacji na temat WPF można znaleźć w [5].

## 3. Zasady działania oprogramowania generującego elementy bazodanowych aplikacji biznesowych

Projektowany system BACG, generujący elementy bazodanowych aplikacji biznesowych, spełnia następujące wymogi:

- zastępuje programistę przy wykonywaniu czynności, które nie wymagają ingerencji i pomysłu ludzkiego,
- wykonuje za programistę większość powtarzalnych zadań, opierając się na z góry ustalonych szablonach,
- zachowuje należytą staranność przy konstrukcji wygenerowanego kodu,
- generuje optymalnie funkcje w modelu obiektowym oraz wiąże je z procedurami składowanymi na serwerze bazy danych,
- kod aplikacji zostaje wygenerowany zgodnie ze wzorcami projektowymi obowiązującymi podczas wytwarzania zaawansowanych systemów informatycznych,

- wszystkie utworzone interfejsy działają na podstawienajnowszej technologii prezentacji danych.

W ostatnim czasie obserwuje się rozwój generatorów wybranych elementów aplikacji biznesowych. Przykładami są tu narzędzia Microsoft, dostępne w ramach pakietu ADO .NET [4]. Ogromnym osiągnięciem w tej dziedzinie jest też rozwiązanie Visual Studio LightSwitch [6], które firma Microsoft udostępniła w połowie 2011 roku.

Jednak, zdaniem autora, rozwiązania te nie spełniają najważniejszego, według programistów, warunku, czyli perfekcyjnego, optymalnego w wykonaniu i zgodnego ze wzorcami projektowymi generowania kodu. Skupiają się jedynie na wsparciu tworzenia warstwy prezentacyjnej, a pomijają istotny postulat spójności kodu wygenerowanego z kodem pisanym przez profesjonalnego, używającego wzorców projektowych, programistę.

Właśnie ten aspekt jest zasadniczym elementem odróżniającym system BACG od istniejących na rynku. Dzięki tej spójności i przejrzystości programista zachowuje pełną kontrolę nad wygenerowanymi elementami i może je bez większych problemów zmieniać na poziomie kodu, jeżeli nie będą one spełniały jego specyficznych wymagań. Ponadto, spójność i zgodność kodu ze wzorcami projektowymi, pozwoli mu szybko odnaleźć i jednoznacznie ocenić optymalność przyjętych rozwiązań, a w przypadku wykrycia błędów lub braków, dokonać szybkiej korekty.

Przy takim podejściu programista aplikacji biznesowych będzie mógł skoncentrować się głównie na tworzeniu logiki biznesowej projektu, zostawiając pozostałe warstwy w większości generatorowi.

Ponadto, projektowany system jest jedynym znanym autorowi generatorem aplikacji opartych na:

- mapowaniu ORM skojarzonym z procedurami składowanymi,
- wzorcu projektowym MVVM,
- warstwie prezentacji opartej na technologii WPF.

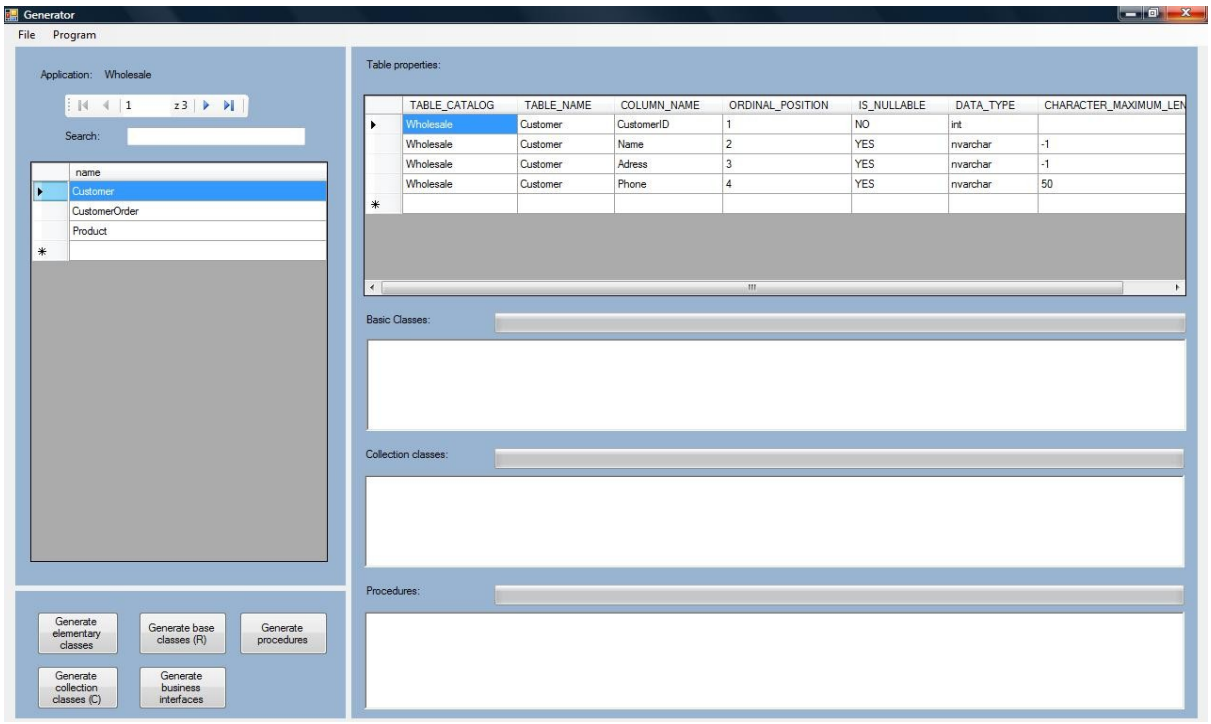
## 4. Ogólny opis generatora BACG

Prototyp systemu BACG został utworzony za pomocą środowiska programistycznego Microsoft Visual Studio 2010. Generator komunikuje się z bazą danych umieszczoną na serwerze Microsoft SQL Server 2008.

System BACG spełnia wszystkie wymogi opisane w punkcie 3. tej publikacji, a jego zasadniczą cechą jest zgodność wszystkich wygenerowanych elementów ze wzorcem projektowym MVVM.

Rozpoczęcie procesu tworzenia elementów aplikacji biznesowej, za pomocą tego generatora, wymaga utworzenia bazy danych na serwerze MS SQL Server 2008.

Główne okno systemu BACG, które wywołuje podstawowe operacje generatora, przedstawia rys. 1.




Rys. 1. Generator BACG – okno główne
Fig. 1. BACG generator – main window

W pierwszym etapie BACG dokonuje mapowania obiektowego relacyjnej bazy danych – zatem zadziała jak klasyczny generator ORM o ograniczonej funkcjonalności (więcej o koncepcji mapowania obiektowo-relacyjnego można przeczytać w [8]).

Następstwem tych działań jest utworzenie zestawu klas typu „R” (każdej tabeli odpowiada jedna klasa), o polach i właściwościach zgodnych z polami kolejnych tabel.

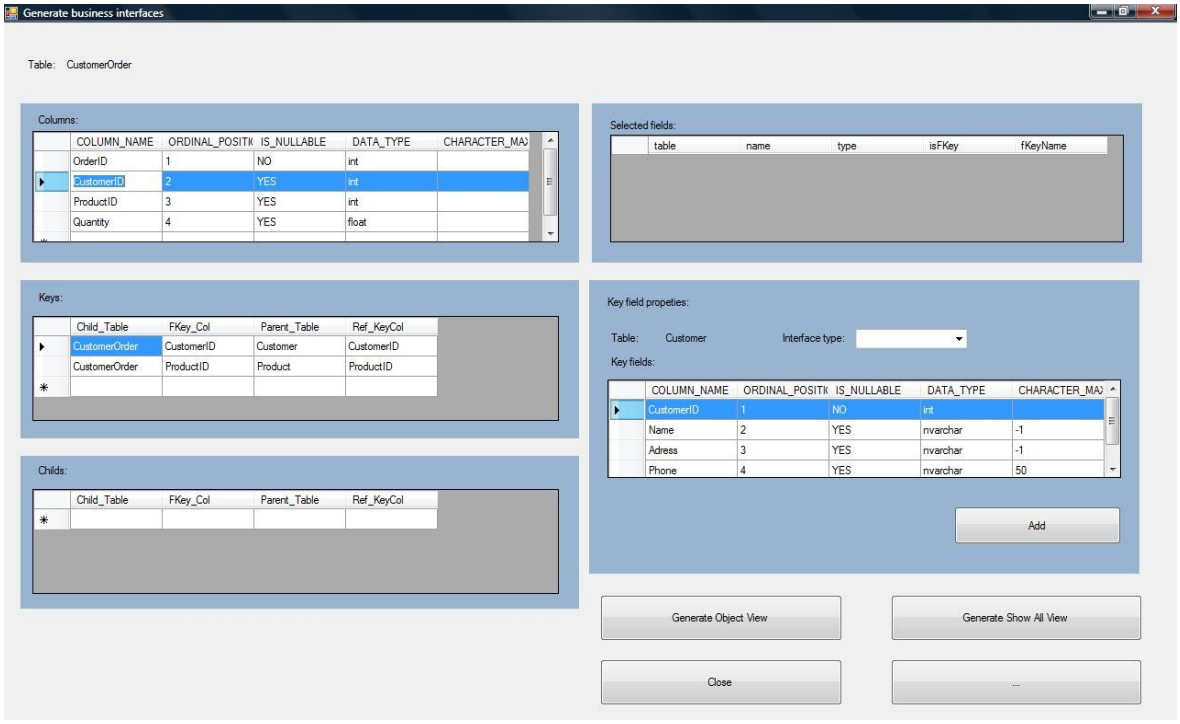
Dla pól powiązanych z innymi tabelami (kluczy obcych) powstają odpowiednie referencje do obiektów związanych.

Następnie tworzony jest zestaw klas typu „C” (każdej tabeli odpowiada również jedna klasa), z funkcjami realizującymi podstawowe operacje na danej tabeli. Równocześnie na serwerze bazy danych powstaje zestaw procedur składowanych, odpowiadający określonym funkcjom z klas typu „C”.

Kolejnym etapem jest generowanie widoków (View) oraz klas odpowiadających za operacje na widoku (ViewModel).

Okno aplikacji, służące do generowania wszystkich widoków, przedstawia rys. 2.

Widoki pozwalają prezentować, dodawać i modyfikować wszystkie obiekty z modelu danych.




Rys. 2. Generator BACG – okno widoków
Fig. 2. BACG generator – view window

Kod źródłowy generowany przez BACG powstaje za pomocą stworzonego przez autora zestawu klas, zawierającego liczne metody tworzące określone części kodu C#.

Następne punkty dostarczają szczegółowych informacji dotyczących kolejnych etapów generowania aplikacji biznesowych.

## 5. Generowanie elementów warstwy Model

Zakładamy, że w ramach implementacji aplikacji biznesowej, na serwerze bazy danych, który będzie ją obsługiwał, powstały wszystkie tabele i powiązania między nimi.

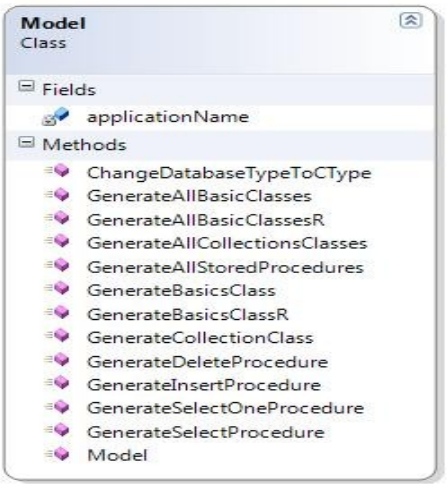
Po wykonaniu tego kroku możemy przystąpić do używania systemu BACG.

W pierwszym etapie pracy z generatorem niezbędne jest podanie parametrów połączenia z bazą danych (nazwa serwera, nazwa bazy danych itd.). Po prawidłowym połączeniu, generator wyświetli wszystkie tabele bazy danych wraz z nazwami pól tych tabel i innymi istotnymi właściwościami (np. typ danych, pozycja itp.). Po zapoznaniu się z tymi informacjami programista – użytkownik systemu BACG – może przejść do procesu generowania elementów warstwy Model swojej aplikacji biznesowej.

### 5.1. Generowanie klas typu „R”

Pierwszym etapem tego procesu jest wygenerowanie klas typu „R”. Klasy encyjne powstają na bazie tabel. Dla każdej tabeli tworzona jest klasa w języku C#, o nazwie skojarzonej z nazwą tabeli (RNazwaTabeli). Każda kolumna tabeli ma swój odpowiednik – prywatne pole klasy. Użyto tu również właściwości C#, które zapewniają elastyczny mechanizm do odczytu i zapisu prywatnych pól. Jeżeli tabela posiada klucze obce, to w odpowiadającej jej klasie powstają pola będące referencjami do obiektów powiązanych.

Ważne jest też, że mechanizm pobierania danych dla obiektu powiązanego jest ściśle połączony z utworzeniem przez generator na serwerze bazy danych stosownej procedury składowanej (pobierającej dane rekordu o wskazanym identyfikatorze). Oczywiście pobieranie obiektu wiąże się z wywołaniem przez stosowną funkcję, z modelu obiektowego, tej procedury. W systemie BACG wszystkie funkcje odpowiedzialne za generowanie klasy typu „R” znajdują się w klasie Model, którą przedstawia rys. 3.




Rys. 3. Klasa Model
Fig. 3. Class Model

Na wskazanie zasługują szczególnie dwie:

- GenerateBasicsClassR(…) – generuje pojedynczą klasę typu „R”,
- GenerateAllBasicClassesR(…) – wywołuje powyższą funkcję dla wszystkich tabel bazy danych.

Obie funkcje wykorzystują metodę public SqlConnection CreateConnection() z klasy AccessToDataBase. Funkcja ta zwraca obiekt typu SqlConnection, który pozwala na połączenie z serwerem bazy danych. Za pomocą tego połączenia do serwera wysyłane są zapytania pobierające strukturę tabel i połączeń między nimi, a na tej podstawie tworzony jest kod klas typu „R”.

Istotna jest tu też funkcja GenerateSelectOneProcedure(), która tworzy kod procedury umożliwiającej załadowanie obiektu powiązanego i umieszcza ją na serwerze bazy danych – jako procedurę składowaną.

### 5.2. Generowanie klas typu „C”

Kolejnym etapem pracy generatora jest automatyczne stworzenie klas typu „C”. Każdej tabeli z bazy danych odpowiada dokładnie jedna klasa tego typu. Standardowo zawiera ona zestaw funkcji realizujących podstawowe operacje pozwalające dodać, usunąć i pobrać obiekty klasy typu „R”. Następstwem wywoływania tych funkcji, dla określonego obiektu klasy typu „C”, jest uruchomienie na serwerze bazy danych procedury składowanej, odpowiedzialnej za dodawanie, usuwanie lub pobieranie rekordów z odpowiedniej tabeli. Wszystkie procedury tworzone są automatycznie razem z klasami typu „C”.

Z uwagi na bezpośrednie skojarzenie funkcji modelu obiektowego z procedurą składowaną, jest to najbardziej optymalny i efektywny sposób realizacji tych operacji.

Za etap automatycznego generowania klas typu „C” odpowiadają następujące funkcje klasy Model:

- GenerateCollectionClass(…) – generuje pojedynczą klasę typu „C”,
- GenerateAllCollectionsClasses(…) – dla każdej tabeli wywołuje poprzednią funkcję,
- GenerateInsertProcedure(…) – tworzy kod procedury, która pozwala dodać nowy rekord do tabeli,
- GenerateDeleteProcedure(…) – tworzy kod procedury, który pozwala usunąć rekord z tabeli,
- GenerateSelectProcedure(…) – tworzy procedurę, która pozwala pobrać wszystkie rekordy tabeli,
- GenerateAllStoredProcedures(…) – zapisuje wszystkie stworzone procedury na serwerze bazy danych, jako procedury składowane.

Oczywiście wszystkie połączenia z bazą danych, podobnie jak poprzednio, realizuje funkcja CreateConnection() z klasy AccessToDataBase.

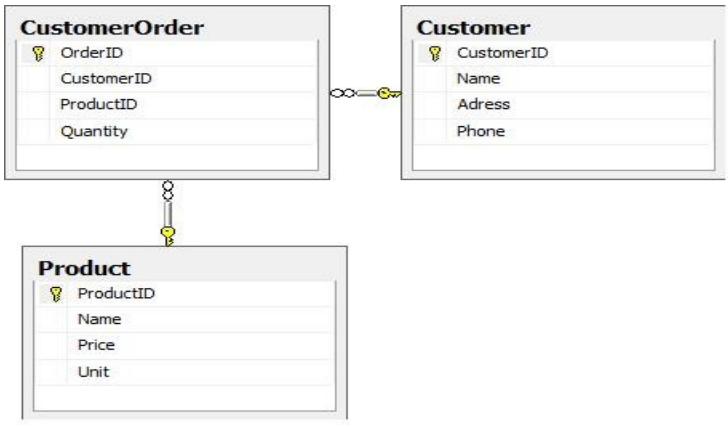
### 5.3. Przykład zastosowania

Dla lepszego zrozumienia istoty działania generatora BACG rozważmy uproszczony diagram bazy danych pozwalającej rejestrować zamówienia na towary wskazanego klienta. Diagram ten przedstawia rys. 4.

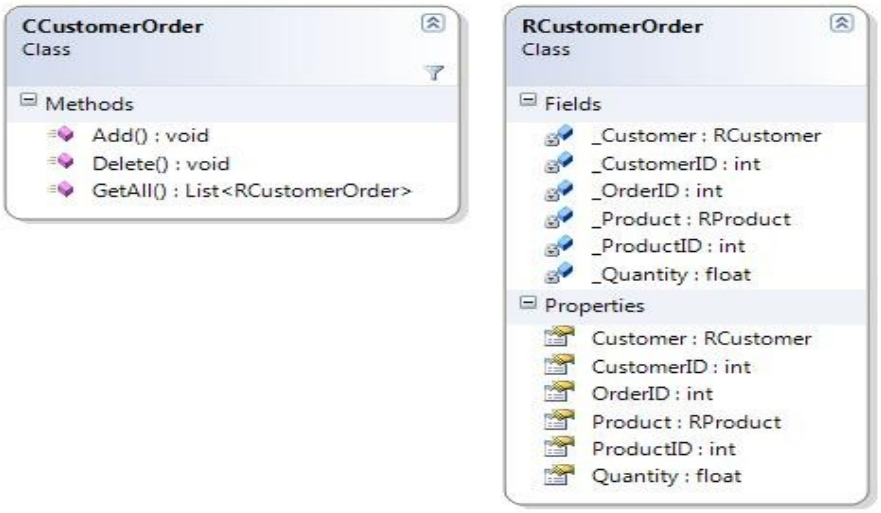
Zauważmy, że tabela CustomerOrder jest powiązana z tabelą Customer i Product – pola CustomerID i ProductID są kluczami obcymi w tej tabeli.

Dla tak skonstruowanej bazy danych generator BACG, w pierwszym etapie swojego działania, wygeneruje trzy klasy typu „R” (RCustomer, RProduct, RCustomerOrder) oraz trzy klasy typu „C” (CCustomer, CProduct, CCustomerOrder). Pola i funkcje przykładowych klas przedstawia rys. 5.

Wygenerowane klasy realizują mapowanie relacyjno-obiektowe oraz zawierają zestaw funkcji operujących na danych zawartych w bazie.




Rys. 4. Diagram bazy danych
Fig. 4. Database diagram




Rys. 5. RCustomerOrder, CCustomerOrder
Fig. 5. RCustomerOrder, CCustomerOrder

## 6. Automatyczne tworzenie elementów warstwy View

Po utworzeniu klas typu „R” i „C” możemy przystąpić do generowania interfejsów aplikacji. Prototyp generatora BACG umożliwia utworzenie widoków według dwóch szablonów. Wszystkie widoki utworzone są za pomocą technologii WPF. Każdy widok to specjalna kontrolka użytkownika (UserControl), którą można umieścić na dowolnym oknie aplikacji. Kod każdego widoku zapisany jest za pomocą xaml [5].

Szablon pierwszy:

Dla każdej tabeli z bazy danych można automatycznie stworzyć widok, który umożliwia utworzenie nowego obiektu (odpowiadającego klasie typu „R") oraz zapis utworzonego obiektu do bazy. Przy generowaniu tego widoku użytkownik wybiera tabelę, a następnie definiuje, które pola obiektu będą możliwe do uzupełnienia. Dla każdego pola system automatycznie dobiera typ wyświetlanej kontrolki (TextBox, ComboBox, CheckBox, DatePicker itd.).

Generator BACG specjalnie traktuje pola odpowiadające kluczom obcym tabeli. Dla tych pól generowane są specjalne kontrolki (ComboBox), w których, za pomocą rozwijanej listy, możemy określić wartość tego klucza na bazie wybranych danych rekordu (obiektu) z tabeli (kolekcji) powiązanej. Oczywiście użytkownik ma pełną kontrolę nad wyborem danych wyświetlanych w rozwijanej liście.

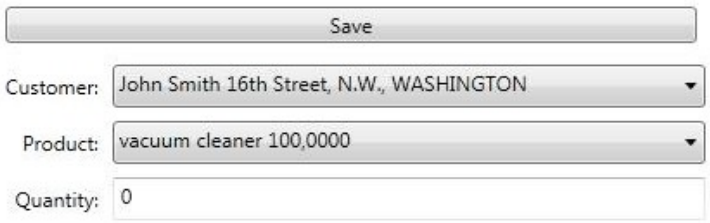
Szablon drugi:

Dla każdej tabeli bazy danych możemy wygenerować widok umożliwiający wyświetlenie wszystkich rekordów danej tabeli. Podobnie jak poprzednio, użytkownik ma pełną kontrolę nad tym, jakie pola obiektu będą widoczne. Oczywiście, zamiast wyświetlania wartości klucza obcego, możemy określić, jakie pola z obiektu powiązanego mają pojawić się podczas prezentacji.

Do każdego widoku wygenerowana zostaje klasa skojarzona – zgodnie z zasadami konstrukcji, określanymi przez technologię WPF. Oczywiście dany widok może być elementem dowolnego okna projektu.

Wszystkie wygenerowane widoki i klasy stowarzyszone zostają zapisane w postaci plików do specjalnie stworzonego katalogu View.

| OrderID | NameCustomer | AdressCustomer | NameProduct | PriceProduct | Quantity |
|---|---|---|---|---|---|
| 1 | John Smith | 16th Street, N.W., WASHINGTON | iron | 20.0000 | 45 |
| 2 | John Smith | 16th Street, N.W., WASHINGTON | vacuum cleaner | 100.0000 | 56 |
| 3 | Tony Walt | Madison Ave., NEW YORK, N.Y. 10016 | iron | 20.0000 | 23 |
| 4 | Tony Walt | Madison Ave., NEW YORK, N.Y. 10016 | vacuum cleaner | 100.0000 | 56 |
| | | | | | |




Rys. 6. Zamówienia – widoki
Fig. 6. Order – view

BACG tworzy wymienione widoki i klasy za pomocą specjalnie zaprojektowanej klasy View. Klasa ta zawiera cztery metody:

- GenerateObjectView(...) – generuje kod xaml widoku według pierwszego szablonu,

- GenerateShowAllView(...) – generuje kod xaml widoku według drugiego szablonu,
- GenerateObjectViewClass(...) – generuje kod klasy skojarzonej z widokiem utworzonym według pierwszego szablonu,
- GenerateShowAllViewClass(...) – generuje kod klasy skojarzonej z widokiem utworzonym według drugiego szablonu.

Dwie pierwsze funkcje używają obiektów klasy ViewLine w celu generowania poszczególnych linii xaml. Dwie kolejne funkcje używają obiektów klasy ClassLine w celu tworzenia linii kodu klasy skojarzonej z widokiem.

Rysunek 6 przedstawia efekt działania generatora, w kontekście generowania warstwy View, dla tabeli CustomerOrder z rys. 4.

## 7. Generowanie elementów warstwy ViewModel

Z uwagi na fakt, że warstwa ViewModel stanowi swoisty łącznik między modelem a widokiem, proces generowania elementów tej warstwy związany jest bezpośrednio z procesem generowania warstwy View. Zatem w ramach tworzenia kodu każdego widoku system BACG generuje jednocześnie odpowiadającą mu klasę ViewModel (w języku C#).

W przypadku widoku zgodnego z pierwszym scenariuszem (opisanym w poprzednim punkcie) powstaje klasa ViewModel, której zadaniem jest ustawienie wartości wybranych pól obiektu wskazanej klasy typu „R”. Posiada ona dwa kluczowe pola. Pierwszym jest obiekt klasy typu „R”, a kolejnym jest obiekt klasy typu „C”. Pierwszy obiekt jest tym, który zostanie uzupełniony przez skojarzony widok, natomiast drugi obiekt dostarcza funkcji działających na kolekcji obiektów klasy typu „C”. Ustawianie wartości pól obiektu klasy „R” zostało zrealizowane za pomocą mechanizmu właściwości dostępnego w C#.

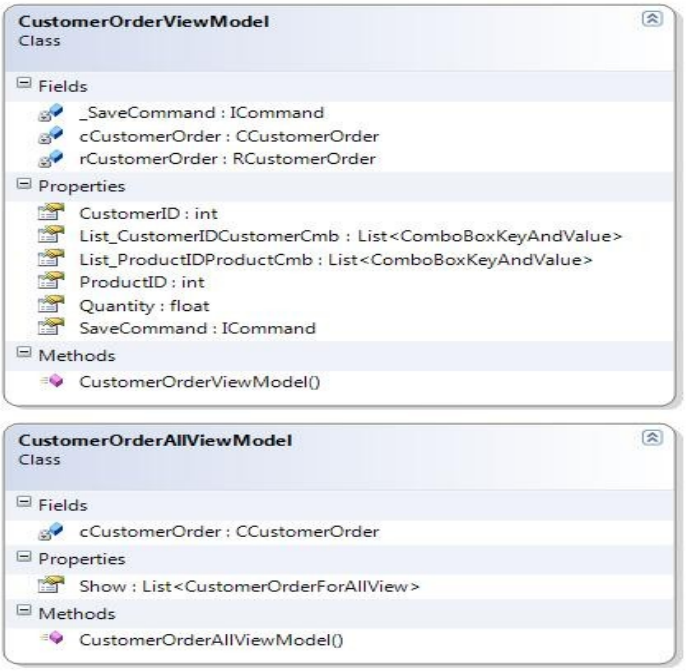
Wygenerowana klasa zawiera też publiczną właściwość SaveCommand (typu ICommand), która umożliwia zapis nowego obiektu do bazy danych przez wywołanie metody Add(…) na rzecz obiektu określonej klasy typu „C”. Klasa ta zostaje automatycznie skojarzona z widokiem, dla którego została stworzona, a kolejne kontrolki widoku są automatycznie skojarzone z jej właściwościami.

Z uwagi na pola odpowiadające kluczom obcym w tabeli, w trakcie generowania klas ViewModel dochodzi do uzupełnienia klas typu „C” o dodatkowe metody.

Zgodnie z założeniem, metody te mają zwracać kolekcję obiektów, która następnie wyświetlana jest w postaci listy wyboru w odpowiadającym tym polom kontrolkom typu ComboBox – umieszczonym na widoku. Należy tu zaznaczyć, że o tym, jakie pola pojawiają się na liście wyboru, decyduje wyłącznie użytkownik systemu, zatem jego decyzja wymusza

stworzenie nowej klasy do przechowywania obiektów o określonych polach. Klasę tę nazwano ComboBoxKeyAndValue. Natomiast metoda o nazwie GetAllOnlySelectedFieldsFieldsName, którą generator dodaje do klasy typu „C", zwraca kolekcję obiektów klasy ComboBoxKeyAndValue. Następstwem tej operacji jest umieszczenie na serwerze bazy danych dodatkowej procedury składowanej, pobierającej tylko określone dane, potrzebne do nadania wartości wszystkim polom obiektów tej kolekcji.

W przypadku widoku zgodnego z drugim scenariuszem (opisanym w poprzednim punkcie) powstaje klasa ViewModel, której zadaniem jest wywołanie metody dostarczającej wybrane przez użytkownika pola wszystkich obiektów. Posiada ona pole klasy typu „C". Na rzecz obiektu tej klasy wywoływana jest funkcja getForAllView(). Oczywiście podczas tworzenia widoku generator uzupełnia odpowiednią klasę typu „C" o tę funkcję. Z uwagi na typ zwracanej wartości przez tę funkcję, konieczne jest też uzupełnianie modelu o dodatkową klasę zawierającą wszystkie wskazane przez użytkownika pola. Następstwem tego jest również dodanie stosownej procedury składowanej, która ma dostarczyć dane dla tej funkcji. W końcu powstała klasa ViewModel zawiera publiczną właściwość List<ClassForAllView> Show, której get wywołuje metodę getForAllView(). Z tą właściwością kojarzona jest kontrolka wyświetlająca kolekcję obiektów na widoku.




Rys. 7. CostomerOrderViewModel
Fig. 7. CostomerOrderViewModel

Za operację generowania elementów ViewModel odpowiada klasa ViewModel generatora. Zawiera ona 5 funkcji:

- GenerateObjectViewModel(…) – generuje kod klasy ViewModel dla widoku zgodnego z pierwszym scenariuszem,

- GenerateAllViewModel(…) – generuje kod klasy ViewModel dla widoku zgodnego z drugim scenariuszem,
- CompleteCClassForObjectViewModel(…) – uzupełnia odpowiednie klasy typu „C" o funkcje niezbędne do prawidłowego działania widoku według pierwszego scenariusza,
- GenerateClassForAllView(..) – generuje klasę niezbędną do prawidłowego zwracania danych przez funkcje getForAllView(),
- CompleteCClassGetForAllView(…) – uzupełnia odpowiednie klasy typu „C" o funkcję niezbędną do prawidłowego działania widoku według drugiego scenariusza.

Wszystkie wygenerowane klasy warstwy ViewModel zostają zapisane przez BACG w postaci plików do specjalnie stworzonego katalogu ViewModel.

Rysunek 7 przedstawia efekt działania generatora w kontekście generowania warstwy ViewModel, dla tabeli CustomerOrder z rys. 4.

## 8. Podsumowanie

Prezentowany generator BACG jest systemem do generowania bazodanowych aplikacji biznesowych. O jego unikalności decydują następujące cechy:

- system jest optymalnym generatorem kodu powiązanego z biznesową warstwą prezentacyjną (w przeciwieństwie do innych tego typu systemów, które koncentrują się wyłącznie na generowaniu formularzy bez dbałości o jakość kodu),
- wszystkie generowane elementy są zgodne z nowatorskim wzorcem projektowym MVVM, co pozwala programiście na pełną kontrolę nad tworzonym kodem,
- BACG ściśle wiąże generowane elementy obiektowe warstwy Model (klas typu „R" i „C") z generowanymi procedurami składowanymi na serwerze bazy danych – zapewniając wysoką wydajność kodu,
- elementy warstwy View generowane są za pomocą najnowszej technologii tworzenia widoków dla Windows, czyli Windows Presentation Foundation,
- generator tworzy elementy warstwy View, dbając o ich biznesową funkcjonalność,
- architektura BACG pozwala na stosunkowo łatwą rozbudowę systemu.

W kolejnych etapach rozbudowy prototypu systemu BACG autor planuje:

- uzupełnić warstwę Model o mechanizm obsługi błędów przez generowanie specjalnych klas wyjątków, następstwem tego będzie dodanie do procesu generowania widoków opcji sprawdzania poprawności wprowadzanych danych wraz z prezentacją komunikatów o błędach,
- rozbudować generator o mechanizm automatycznej refaktoryzacji kodu,

- dodać do etapu generowania widoków kilka dodatkowych szablonów, które umożliwią realizację zaawansowanych funkcji biznesowych (np. szablon pozwalający wykonywać operacje na obiektach biznesowych typu „jeden-wszystkie" – jedna faktura \ oferta \ zamówienie ze wszystkimi pozycjami tego obiektu itp.),
- uzupełnić generowany kod o mechanizm umożliwiający obsługę obiektowego języka zapytań, operującego na obiektach warstwy Model, który zostanie ściśle powiązany z generatorem odpowiednich procedur składowanych na serwerze bazy danych (zapytania obiektowe będą wywoływały stosowne procedury składowane).

## BIBLIOGRAFIA

1. Garofalo R.: Building Enterprise Application with Windows Presentation Foundation and the Model View ViewModel Pattern. O'Reilly, 2011.
2. Pressman R. S.: Praktyczne podejście do inżynierii oprogramowania. WNT, 2008.
3. Górski J.: Inżynieria oprogramowania. Mikom, 1999.
4. Patrick T.: Microsoft ADO.NET 4. Microsoft Press, 2010.
5. Nathan A.: Windows Presentation Foundation. Sams Publishing, 2007.
6. Visual Studio LightSwitch, http://msdn.microsoft.com/pl-pl/library/ff851953.aspx.
7. Sorensen E., Mikailesc M.: Model-View-ViewModel (MVVM) Design Pattern using Windows Presentation Foundation (WPF) Technology, http://megabyte.utm.ro/articole-/2010/info/sem1/InfoStraini_Pdf/1.pdf.
8. Ambler S. W.: Designing a Robust Persistence Layer. Softw. Dev., Vol. 6(2), 1998, s. 73÷75.
9. Cleaveland J. G.: Building application generators. IEEE Software, 1988.
10. Jones N. D., Gomard C., Sestoft P.: Partial Evaluation and Automatic Program Generation. International Series in Computer Science. Prentice-Hall, Engle Wood Cliffs, NJ 1993.

Wpłynęło do Redakcji 13 stycznia 2012 r.

## Abstract

The object of the paper is to present an idea of the structure of a system generating elements of business application together with its architecture and way of operating of the generator prototype called BACG (Business Application Code generator).

The paper begins with the description of a design template MVVM which is particularly suitable for creating user interfaces with the help of the WPF technology (Windows Presentation Foundation).

Subsequent sections describe principles of software operation which generate database components of business applications and present a general characteristics of the BACG generator.

The fundamental ingredient of the presented work are sections that show the process of building, with the help of the BACG generator, subsequent layers of business applications in accordance with the design template MVVM.

The paper describes in detail a process of generating of the Model, View, and ViewModel layers. That part contains also the key classes of the BACG generator and main elements of the source code created by this generator. These descriptions are supplemented by specific examples of usage of the presented generator.

The paper is finalized by the section which gathers innovative features of the BACG system and presents its possible future development.

**Adres**

Artur KORNATKA: Wyższa Szkoła Biznesu – National-Louis University, Wydział Informatyki, ul. Zielona 27, 33-300 Nowy Sącz, Polska, akornatka@wsb-nlu.edu.pl.