Generowanie aplikacji Borland Delphi w oparciu o repozytorium CASE – Oracle Designer®

Janusz Perek, Michał Wiśniewski
ComArch –Poznań Sp. z o.o.
e–mail:Janusz.Perek@ComArch.pl, Michal.Wisniewski@ComArch.pl

Abstrakt. Artykuł poświęcony jest zaprezentowaniu narzędzia „Gandalf" do generowania aplikacji Borland Delphi w oparciu o definicję modułu zapamiętaną w repozytorium CASE. Borland Delphi 5.0 jest zaawansowanym pakietem do tworzenia klienckich aplikacji bazodanowych. Nie wspiera on jednak etapu projektowania aplikacji na podstawie analizy wymagań użytkowników. W artykule opisano sposób integracji środowiska Borland Delphi 5.0 z repozytorium Oracle Designer 2000 oraz architekturę systemu wykorzystującego wygenerowane aplikacje. Na zakończenie przedstawiono proces budowy przykładowej aplikacji z wykorzystaniem prezentowanego rozwiązania.

1. Wprowadzenie

Pomysł zbudowania narzędzia do generacji aplikacji Borland Delphi w oparciu o informacje zapisane w repozytorium pojawił się podczas wykonywania prac projektowych złożonego systemu informatycznego.

Z jednej strony mamy narzędzie - Oracle Designer posiadające możliwość generowania modułów na podstawie projektu zapamiętanego w repozytorium. Jednak docelowe środowisko wygenerowanych modułów (Oracle Forms) posiada tylko prosty interfejs użytkownika i nie pozwala na łatwe dodawanie nowych komponentów mogących go rozszerzyć[1]. Poza tym nie istnieje żadna metoda rozszerzenia funkcjonalności samego środowiska programistycznego.

Z drugiej strony mamy narzędzie Borland Delphi 5.0, będące zaawansowanym pakietem do tworzenia klienckich aplikacji bazodanowych, charakteryzujące się otwartością, ale nie posiadające możliwości generacji modułów. Co prawda istnieją generatory aplikacji dla Delphi (np.: Rational Rose 2000, Microgold WithClass), ale nie są one zintegrowane z produktami firmy Oracle. Na szczęście otwartość Delphi pozwala na łatwą implementacje generatora dla tego środowiska.

Kompilacją powyższych faktów jest „Gandalf" - narzędzie pozwalające na generowanie modułów Delphi w oparciu o repozytorium Oracle Designer.

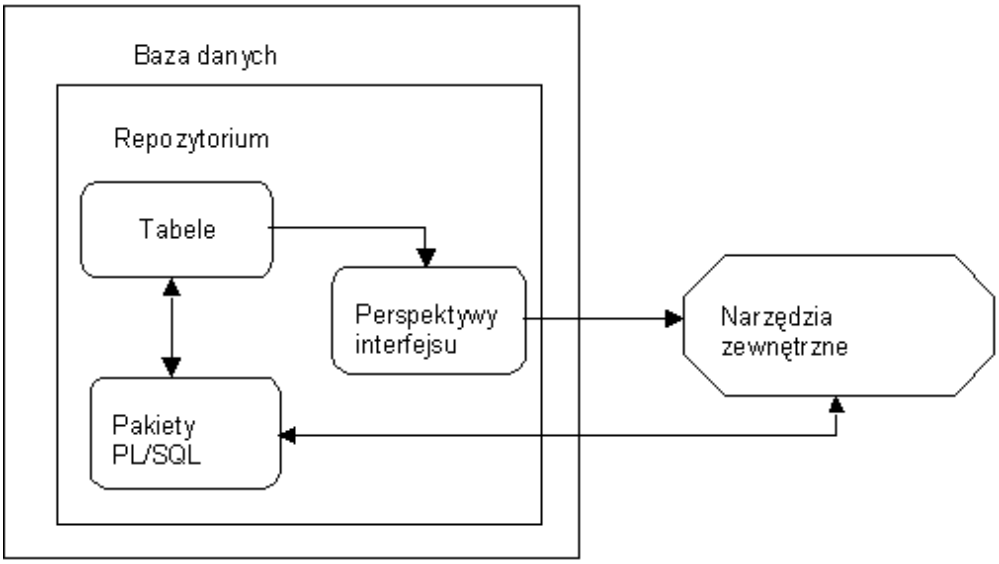
2. Przegląd interfejsów

2.1. Oracle Designer API

Oracle Designer udostępnia interfejs programowy (API) umożliwiający dostęp zewnętrznym narzędziom do repozytorium. Interfejs ten jest zbiorem perspektyw i pakietów PL/SQL'owych znajdujących się w schemacie właściciela repozytorium i zapewnia bezpieczne metody dostępu do metadanych. Metadane są przechowywane w niewielkiej liczbie tabel, pomiędzy którymi istnieje wiele złożonych powiązań, które nie są udokumentowane. (Oracle nie wspiera bezpośrednich metod dostępu do tych tabel za pomocą instrukcji DML). Informacje zawarte w tabelach są prezentowane poprzez perspektywy, które stanowią bardzo ważny element interfejsu.

[1] Oracle Forms Developer i Oracle Forms Server pozwala na tworzenie rozszerzeń przy użyciu JavaBeans i Pluggable Java Components (PJCs) oraz zastosowaniu komponentów OCX.

Kolejnym elementem, z którego zbudowany jest interfejs to pakiety PL/SQL'owe, które umożliwiają w sposób bezpieczny dokonywanie modyfikacji w zawartości tabel repozytorium przez narzędzia zewnętrzne. Jedną z najistotniejszych rzeczy jest to, iż Oracle Designer używa dokładnie tego samego interfejsu przy wstawianiu, modyfikacji, usuwaniu i pobieraniu danych z repozytorium. Rysunek 1 pokazuje w jaki sposób narzędzia zewnętrzne mogą przechowywać i pobierać informacje z repozytorium.



Rys. 1. Dostęp do repozytorium z użyciem interfejsu programowego Oracle Designer'a

Tabele znajdujące się w repozytorium mają nazwy rozpoczynające się od SDD_ lub CDI_ (np.: SDD_ELEMENTS, CDI_TEXT). Zawierają one informacje o wielu różnych typach elementów występujących w Oracle Designer'ze. Dla przykładu, definicje tabel są przechowywane w tabeli SDD_ELEMENTS przy czym atrybut EL_TYPE_OF ma wartość 'TAB' a atrybut EL_OCCUR_TYPE wartość 'TABLE'.

Nazwy perspektyw stanowiących interfejs programowy zaczynają się od CI_ i zawierają liczbę mnogą nazwy elementu. Np.: perspektywa pokazująca definicje encji ma nazwę CI_ENTITIES a perspektywa prezentująca definicje modułów aplikacyjnych CI_MODULES.

Oracle Designer zawiera system aplikacji, w którym znajdują się definicje wszystkich perspektyw wraz z powiązaniami pomiędzy nimi. System aplikacji można załadować do repozytorium z pliku ORACLE_HOME/DES2_71/MODEL/MODEL_60.DAT lub zaimportować z pliku ORACLE_HOME/DES2_71/MODEL/MODEL_60.DMP (dla wersji Oracle Designer 6.0). Informacje na temat interfejsu programowego Oracle Designer'a znajdują się w katalogu ORACLE_HOME/CDOC70/api.

Bardziej szczegółowe informacje na temat korzystania z Oracle Designer API znajdują się w [1].

2.2. Biblioteka Open Tools API (Borland Delphi)

Biblioteka Open Tools API (OTA) umożliwia dodawanie nowej funkcjonalności do zintegrowanego środowiska programistycznego (ang. IDE – Integrated Development Environment) Borland Delphi oraz Borland C++ Builder. OTA definiuje zbiór interfejsów COM (Common Object Model) udostępniających użytkownikowi informacje oraz mechanizmy ze środowiska IDE, pozwalające na wykonanie następujących czynności:

- tworzenie kreatorów form i projektów,
- tworzenie aplikacji pomocniczych,

- rozszerzanie menu, list akcji, pasków narzędziowych, edytora,
- tworzenie nowych skrótów klawiszowych oraz makr,
- manipulowanie zasobami projektu,
- manipulowanie modułami projektu – tworzenie, otwieranie, zamykanie, dodawanie nowych modułów do projektu i ich usuwanie.

Rozszerzenie środowiska programistycznego wykorzystując interfejsy OTA sprowadza się do implementacji potrzebnych interfejsów. Rozszerzenia takie mają postać bibliotek dynamicznie ładowanych. Mogą to być zarówno biblioteki typu DLL jak i BPL[2].

Szczegółowe informacje znajdują się w [2].

3. Rozwiązanie

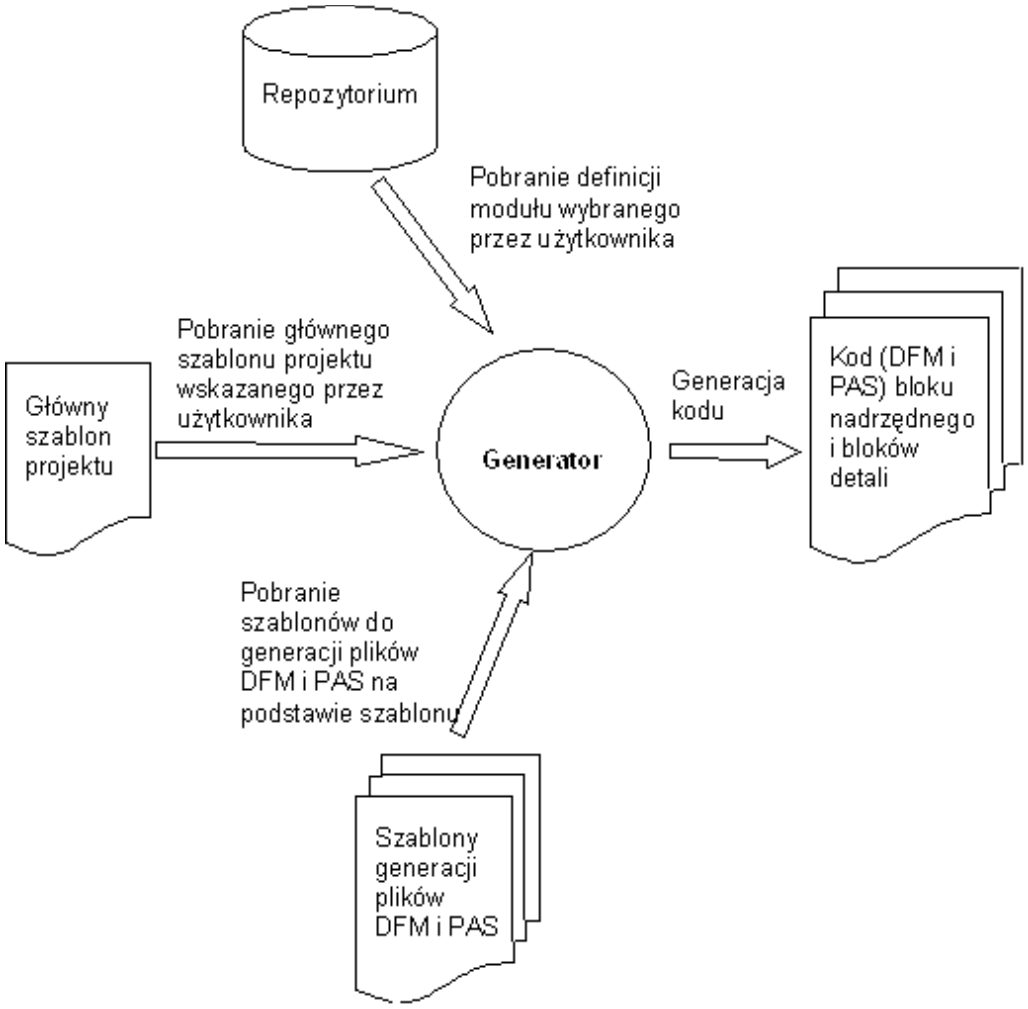
Narzędzie „Gandalf" zostało zaimplementowane jako biblioteka BPL i zintegrowane ze środowiskiem Borland Delphi wykorzystując interfejs OTA. Pobieranie informacji z repozytorium zostało zrealizowane z wykorzystaniem Oracle Designer API. W celu zapewnienia dużej elastyczności rozwiązania moduły generowane są w oparciu o szablony. Dzięki zastosowaniu komponentu firmy Dream Company – Dream Scripter do szablonów została dodana możliwość umieszczania kodu wykonywalnego w postaci skryptowej.

Proces generacji modułu podzielony jest na następujące etapy:

1. Użytkownik określa parametry procesu generacji: wybierana jest definicja modułu zapamiętana w repozytorium oraz szablon główny na podstawie, którego tworzone są wszystkie pliki projektu Delphi.
2. Generator pobiera odpowiednie informacje z repozytorium oraz wczytuje wskazany główny szablon projektu.
3. Na podstawie informacji z głównego szablonu projektu odczytywane są szablonów do generacji plików DFM i PAS.
4. Generacja kodu (pliki DFM i PAS) dla bloku nadrzędnego i bloków detali.

Po wygenerowaniu plików zawierających kody źródłowe modułu można przy użyciu środowiska Delphi wprowadzić konieczne modyfikacje np.: zmiana położenia kontrolek, kolejność nawigacji, dodanie nowej funkcjonalności do modułu a następnie utworzyć bibliotekę BPL.

[2] BPL (Borland Package Library) jest specyficznie skompilowaną biblioteką DLL.

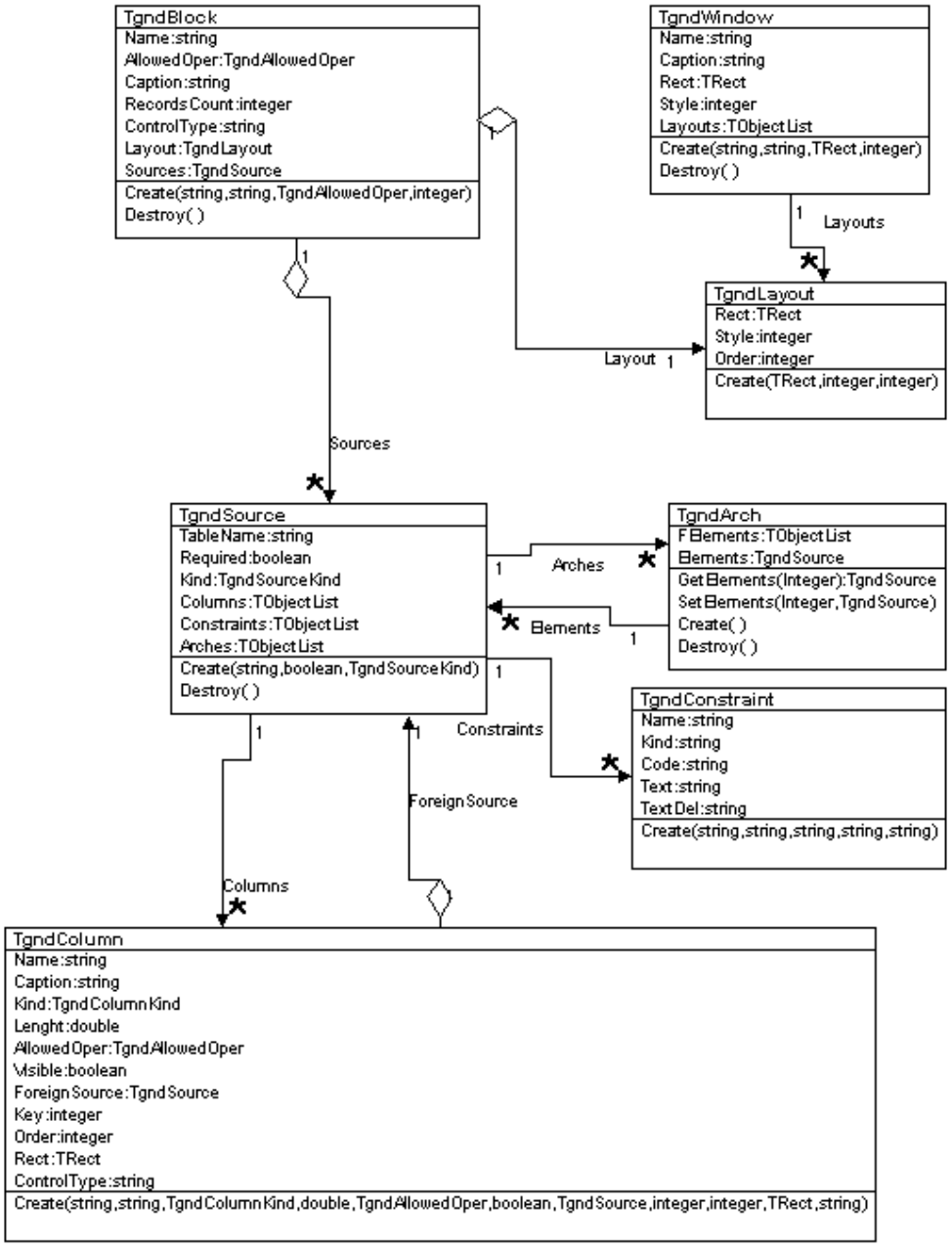


Rys. 2. Proces generacji modułu

3.1 Struktury danych

Użyte struktury danych stanowią odzwierciedlenie cech i powiązań jakie występują pomiędzy elementami definiującymi moduł w repozytorium Oracle Designer. Ze względu na fakt, że aktualnie „Gandalf" generuje pojedyncze moduły, nie została stworzona klasa odpowiadająca modułowi.

Klasa TgndWindow reprezentuje okna jakie znajdują się w generowanym module. Klasa TgndBlock stanowi reprezentacje bloku, dla którego określone są dozwolone operacje oraz liczba wyświetlanych rekordów. Klasa TgndLayout zawiera informacje na temat położenia i rozmiaru każdego bloku oraz powiązanie z oknem (TgndWindow), w którym ma się znaleźć generowany blok. Z każdym blokiem powiązane są źródła danych (TgndSource), które podzielone są na dwa typy: źródła podstawowe (ang. base) oraz źródła doczytywane (ang. lookup). Każde ze źródeł posiada kolumny (TgndColumn). Powiązanie łączące kolumnę ze źródłem danych (ForeignSource) zawiera informację o relacjach typu Master-Detail. Zależności pomiędzy źródłami doczytywanymi są zdefiniowane przy użyciu klasy TgndArch. Klasa TgndConstraint zawiera informacje na podstawie, których w fazie generacji może zostać utworzony plik definiujący komunikaty obsługi błędów w przypadku naruszenia ograniczeń integralnościowych.



Rys. 3. Diagram klas

3.2 Struktura projektu w Delphi

Każda formatka stworzona w środowisku Delphi składa się z dwóch plików:

- pliku z rozszerzeniem PAS zawierający kod wykonywalny i deklarację klasy,
- pliku z rozszerzeniem DFM zawierającym ustawienia własności wszystkich komponentów umieszczonych na formatce.

Od wersji 5 Borland Delphi zapisuje pliki DFM jako zwykłe pliki tekstowe (we wcześniejszych wersjach były to pliki binarne). Dzięki temu generacja modułu sprowadza się do stworzenia dwóch plików tekstowych o odpowiedniej strukturze i zawartości.

Dodatkowo wymagane jest posiadanie pliku projektu, który określa w jaki sposób ma zostać utworzony plik wynikowy i jakie formatki ma zawierać. Może to być plik wykonywalny, biblioteka BPL, biblioteka DLL lub obiekt COM. Również plik projektu jest plikiem tekstowym. Pełen proces generacji można więc sprowadzić do wygenerowania par plików (PAS i DFM) dla każdego[3] modułu oraz pliku projektu.

3.3 Komponent Dream Scripter

Komponent Dream Scripter jest produktem, który pozwala wbudowywać w aplikacje fragmenty skryptów używając do tego celu VBScript, JScript, Perl'a, Python'a a także Delphi Script. Skrypty wykonywane za pomocą tego komponentu mają dostęp do wszystkich zarejestrowanych klas, zdarzeń, procedur, funkcji i stałych. Przykładowo można napisać procedurę obsługi zdarzenia w JScript, w której mamy dostęp do wszystkich komponentów, z których zbudowany jest moduł napisany w Delphi. Można również wykorzystując np. VBScript dynamicznie modyfikować taki formularz – np. zmienić pewne własności pól.

Zastosowanie komponentu Dream Scripter pozwala na zdefiniowanie szablonów, na podstawie których generowane są wynikowe moduły. Szablon taki jest plikiem tekstowym zawierającym oprócz stałych informacji znajdujących się w plikach Delphi również fragmenty kodu oznaczone specjalnymi znacznikami, które są ładowane i przetwarzane przez komponent Dream Scripter. Te fragmenty kodu na podstawie informacji pobranej z repozytorium wstawiają do plików wynikowych kod w Delphi odpowiedzialny za postać generowanego modułu.

3.4 Szablony

W narzędziu „Gandalf" szablony są plikami tekstowymi. W celu wyróżnienia fragmentów, które mają zostać wykonane, zostały wprowadzone dwa znaczniki: {# - początek skryptu oraz #} – koniec skryptu. Pomiędzy tymi znacznikami można umieścić kod napisany w Delphi Script. Oprócz tego zostały wprowadzone dwie zmienne, do których można się odwoływać poza fragmentami skryptów poprzez znaczniki: %MODULE_NAME% - nazwa generowanego modułu oraz %UNIT_NAME% - nazwa aktualnego pliku wynikowego.

Proces generacji w oparciu o szablon odbywa się dwuetapowo. W pierwszym etapie podmieniane są wszystkie wystąpienia zmiennych. W drugim etapie wszystkie linie tekstu nie będące skryptem kopiowane są do pliku wynikowego. Zamiast skryptu do pliku wynikowego kopiowane są linie tekstu – argumenty wywołania procedury Print.

W narzędziu „Gandalf" zostały wyróżnione dwa typy szablonów: szablon główny (o nazwie zgodnej z nazwą katalogu, w którym jest umieszczony) – zawierający informacje o procesie generacji oraz szablony generujące – zawierające informację potrzebą do wygenerowania konkretnego fragmentu modułu.

W trakcie generacji odczytywany jest szablon główny. Za pomocą skryptu opisane jest w nim jakie pliki i na podstawie jakich szablonów powinny się wygenerować. Dzięki takiemu rozwiązaniu możliwe jest wygenerowanie oprócz definicji formatki, także pliku projektu i dowolnej liczby plików pomocniczych (np. pliku z komunikatami obsługi błędów w przypadku naruszenia ograniczeń integralnościowych). Szablon główny zawiera tylko część skryptową (nie jest tworzony dla niego żaden plik wynikowy).

Przykładowy szablon główny:

[3] W obecnej wersji „Gandalf" umożliwia generację tylko jednego modułu.

```
{#
const DETAIL_PAS_TEMPLATE  = 'DETAIL_PAS.tpl';  // szablon dla bloków detali
      DETAIL_DFM_TEMPLATE  = 'DETAIL_DFM.tpl';
      MASTER_PAS_TEMPLATE  = 'MAIN_PAS.tpl';    // szablon dla bloku głównego
      MASTER_DFM_TEMPLATE  = 'MAIN_DFM.tpl';
      PROJECT_TEMPLATE     = 'PROJECT.tpl';     // szablon projektu
      CONSTRAINTS_TEMPLATE = 'CONSTRAINTS.tpl'; // szablon dla komunikatów
var
  i:integer;
begin
// generacja bloku głównego
  Gen(0,MASTER_PAS_TEMPLATE,TgndBlock(ModuleBlocks.Items[0]).Name+'_m.pas');
  Gen(0,MASTER_DFM_TEMPLATE,TgndBlock(ModuleBlocks.Items[0]).Name+'_m.dfm');

// generacja bloków detali
  i := 1;
  while (i < blocks.count) do begin
   Gen(i,DETAIL_PAS_TEMPLATE,TgndBlock(ModuleBlocks.Items[0]).Name+'_u.pas');
   Gen(i,DETAIL_DFM_TEMPLATE,TgndBlock(ModuleBlocks.Items[0]).Name+'_u.pas');
   Inc(i);
  end;

//generacja projektu
  Gen(0,PROJECT_TEMPLATE,ModuleName+'.dpk');

// generacja komunikatów
  Gen(0,CONSTRAINTS_TEMPLATE,'Messages.sql');
end;
#}
```

Szablony generujące są szablonami, w oparciu o które tworzone są właściwe pliki opisujące formatkę i projekt. Ich zawartość jest ściśle zależna od przyjętych standardów oraz od typu projektu (plik wykonywalny, biblioteka BPL, itd.). Dzięki możliwości umieszczania skryptów w szablonach można dostosować wynik generacji do własnych potrzeb.

Fragmenty skryptu generującego (PROJECT.tpl):

```
package %MODULE_NAME%;
{$R *.RES}
{$ALIGN ON}              // reszta została usunięta dla celów przykładu
requires
  vcl50,
  ComArchp;             // reszta została usunięta dla celów przykładu
contains
{#
var i:integer;
    s:string;
begin
  for i := 0 to ModuleBlocks.count-1 do begin
    s := TgndBlock(ModuleBlocks[i]).Name;
    s := s+' in '''+s+'.pas'' {'+s+'}';
    if i = ModuleBlocks.count-1 then
      print(s+';')
    else
      print(s+',');
end;
#}
```

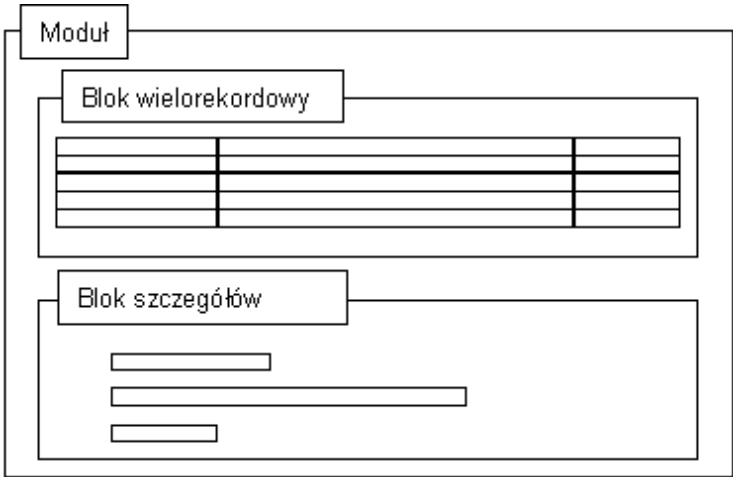
```
end.
```

4. Zastosowanie

Proces przygotowywania „Gandalfa" do generacji formatek należy rozpocząć od stworzenia szablonów generacji. W zależności od architektury realizowanej aplikacji różna będzie zawartość szablonu głównego oraz szablonu generującego projekt. Zawartość szablonów generujących formatki w większości przypadków będzie niezależna lub bardzo mało zależna od przyjętej architektury.

Architektura systemu, do której generowane są formatki z wykorzystaniem „Gandalfa" oparta jest o założenie, że każdy moduł jest osobną biblioteką BPL. Takie założenie daje dużą elastyczność w realizacji projektu – łatwość dzielenia zadań na poszczególnych wykonawców, jak również w rozszerzaniu aplikacji – wszystkie moduły ładowane są dynamicznie, a lista dostępnych modułów jest zapamiętana w bazie. Efektem takiego założenia jest odpowiedni szablon generujący plik projektu (przykład – patrz rozdział 3.4).

W każdy module przeglądanie danych odbywa się w bloku wielorekordowym, a modyfikacje i wstawianie w bloku jednorekordowym (blok szczegółów), który znajduje się poniżej bloku wielorekordowego.



Rys. 4. Struktura wizualna generowanych modułów

Często zdarza się tak, że blok szczegółów jest zagnieżdżoną drugą formatką osadzoną w formatce głównej. Jeżeli formatka posiada dużo szczegółów (nie mieszczą się na jednej formatce), albo posiada detale, to są one dodawane na kolejnych zakładkach bloku.

Proces projektowania modułu w Oracle Designer'ze wymaga zwrócenia uwagi na fakt, że standard modułu wymaga aby mieścił się on w jednym oknie.

5. Wnioski i plany na przyszłość

Zastosowanie „Gandalfa" w procesie tworzenia systemu informatycznego spowoduje skrócenie czasu potrzebnego do wykonywania nowych modułów oraz ułatwi zachowanie wewnętrznych standardów. Połączenie korzyści jakie wynikają z lepszej dokumentacji procesu produkcji oprogramowania, w oparciu o repozytorium, z możliwościami narzędzia jakim jest Borland Delphi pozwoli na stworzenie lepszego produktu.

Generacja modułów Delphi na bazie repozytorium Oracle Designer jest dopiero początkowym etapem tworzenia zaawansowanego środowiska programistycznego. W przyszłości planuje się

rozbudowanie możliwości „Gandalfa" o generowanie wielu modułów na raz, wykorzystując dodatkowe informacje z repozytorium - o powiązaniach pomiędzy modułami, wykonanie dalszej integracji narzędzi umożliwiając dostęp (przeglądanie i modyfikacje) do diagramów zapisanych w repozytorium bezpośrednio ze środowiska Borland Delphi, oraz rozszerzenie zestawu danych pobieranych z repozytorium związanych z definicją modułu.

Rozważana jest także współpraca z częścią obiektową Oracle Designer'a – tworzenia diagramów UML dla wygenerowanych modułów. Rozpatrujemy również możliwość integracji Oracle Designera z innymi narzędziami programistycznymi firmy Inprise np.: JBuilder.

Bibliografia

1. Scherer D., Koletzke P.: Oracle Designer API: Multiline Text and Other Secrets, IOUG-A Conf.,1999 Paper#705
2. Kinas R. : Open Tools API – rozbudowanie możliwości Delphi, Borland Developer's Days, Warszawa, 16 - 17 maj 2000
3. www.dreamcompany.com