Politechnika Łódzka
Instytut Informatyki

INŻYNIERSKA PRACA DYPLOMOWA

MAPOWANIE OBIEKTOWO-RELACYJNE W JĘZYKU C++

Wydział: Fizyki Technicznej, Informatyki i Matematyki Stosowanej
Promotor: dr inż. Arkadiusz Tomczyk
Dyplomant: Marcin Maciaszczyk
Nr albumu: 165466
Kierunek: Informatyka
Specjalność: Inżynieria Oprogramowania i Analiza Danych

Łódź, 10 maja 2014r.

Instytut Informatyki
90-924 Łódź, ul. Wólczańska 215, budynek B9
tel. 042 631 27 97, 042 632 97 57, fax 042 630 34 14 email: office@ics.p.lodz.pl

Spis treści

Rozdział 1

Wstęp

1.1 Uzasadnienie wyboru tematu

Wraz z rozwojem informatyki proces tworzenia oprogramowania wymaga od informatyków co raz większej wiedzy w poszczególnych dziedzinach takich jak na przykład bazy danych czy sieci komputerowe. Ciężko jest być specjalistą w każdej z nich, dlatego też podczas tworzenia oprogramowania programiści co raz częściej sięgają po różnego rodzaju narzędzia programistyczne, które mają za zadanie ułatwić im pracę.

Przykładem takich narzędzi są aplikacje szkieletowe wykorzystywane jako fundamenty dla tworzonych aplikacji czy też biblioteki programistyczne udostępniające zestawy określonych funkcji. Oczywiście nie zawsze mamy możliwość skorzystania z wspominanych narzędzi, jednak jeśli taka istnieje warto wziąć to pod uwagę podczas fazy planowania tworzenia oprogramowania, ponieważ decydując się na korzystanie z nich jesteśmy w stanie zaoszczędzić sporo czasu, a także uniknąć wielu błędów związanych z nieznajomością danej dziedziny.

Wraz z kolegą z tego samego roku studiów – Sebastianem Florkiem, zdecydowaliśmy się w ramach pisamia pracy dyplomowej na wspólne stworzenie aplikacji szkieletowej, która będzie realizowała mapowanie obiektowo-relacyjne w języku C++. Wybór C++ jako języka programowania tłumaczymy dość dobrą jego znajomością, a także pewnym doświadczeniem w programowaniu w tym języku nabytym w trakcie studiów. Mapowanie obiektowo-relacyjne jest to obecnie zagadnienie bardzo powszechne, szczególnie w językach programowania takich jak C# czy Java. Programiści C++ nie mają już tak dużego wyboru wśród dostępnych narzędzi służących do mapowania obiektowo-relacyjnego, co także braliśmy pod uwagę ustalając temat nad jakim będziemy pracować.

Podział naszej pracy zostanie opisany w dalszych rozdziałach, jednak na wstępie

warto zaznaczyć, że tematem pracy Sebastiana jest generator opisu mapowania obiektowo-relacyjnego, który jest wstępnym etapem pracy naszej aplikacji szkieletowej. Celem wspomnianego generatora jest wygenerowanie pliku projektu, a także plików nagłówkowych oraz klas na podstawie istniejącej już bazy danych. Moim zadaniem będzie samo mapowanie obiektowo-relacyjne, które będzie się opierało na wcześniej wygenerowanych plikach.

1.2 Problematyka i zakres pracy

Programowanie obiektowe jest obecnie jednym z najpopularniejszych paradygmatów programowania, a pojęcia takie jak klasa czy obiekt znane są wszystkim programistom. Podobnie jest z relacyjnym modelem organizacji baz danych i terminami takimi jak relacja czy krotka. Chcąc wykorzystać oba te podejścia w jednej aplikacji musimy zadbać o obustronną konwersję pomiędzy danymi z tabel relacyjnej bazy danych a obiektami aplikacjii. Tym właśnie zajmuje się mapowanie obiektowo-relacyjne, które wraz z tworzeniem aplikacji szkieletowych w języku C++ jest główną problematyką niniejszej pracy.

W tym momencie powstaje pytanie czy na prawdę warto korzystać z bibliotek i aplikacji szkieletowych służących do mapowania obiektowo relacyjnego? Odpowiedź nie jest jednoznaczna w wszystkich przypadkach, ale warto wymienić jego podstawowe wady i zalety, których dokładniejsza analiza znajduje się w dalszej części pracy. Zacznijmy od zalet wykorzystania narzędzi ORM[1]:

- Znacznie zredukowana ilość pracy wymagana na oprogramowanie dostępu do bazy danych.
- Nieobowiązkowa znajomość języka SQL[2]. W celu tworzenia zapytań do bazy danych korzystamy z interfejsu udostępnianego przez daną bibliotekę czy też aplikację szkieletową.
- Uniezależnienie się od rodzaju systemu zarządzania bazą danych, możemy korzystać równie dobrze z MySQL, Oracle, PostgreSQL czy też Microsoft SQL Server niekoniecznie posiadając o nich rozbudowaną wiedzę.
- Aby skorzystać z transakcji, połączenia z bazą danych a także wieu innych funkcjonalności baz danych wystarczy zazwyczaj wywołać pojedyńczą metodę.

[1] mapowanie obiektowo-relacyjne (ang. Object-Relational Mapping)
[2] strukturalny język zapytań (ang. Structured Query Language)

- Cały model danych przechowywany jest w jednym miejscu oraz nie jest ściśle związany z wykorzystywanym systemem zarządzania bazą danych dzięki czemu łatwiej jest nam wprowadzać kolejne modyfikacje w kodzie oraz nawet zmieniać system zarządzania bazą danych na inny.

Wszędzie gdzie pojawiają się zalety mamy też do czynienia z wadami, nie inaczej jest w tym przypadku:

- Konfiguracja jest najczęściej skomplikowana i wymaga sporo czasu.
- Podobnie jest z użytkowaniem, aby robić to w sposób optymalny musimy dobrze poznać daną bibliotekę czy też aplikację szkieletową.
- Proste zapytania są obsługiwane bardzo sprawnie, jednak gdy przetwarzamy duże ilości złożonych zapytań wydajność nie dorówna nigdy zapytaniom napisanym przez specjalistę znającego język SQL.
- Abstrakcja wprowadzona przez narzędzia ORM może okazać się uciążliwa, ponieważ nie zawsze zdajemy sobie sprawę z tego co dzieje się za kulisami w trakcie wykonywania poszczególnch operacji.

Naszym głównym celem jest uczynienie Qubica dobrą alternatywą dla nielicznych, ale istniejących już bibliotek oraz aplikacji szkieletowych realizujących mapowanie obiektowo-relacyjne w języku C++. Wszystkie nam obecnie znane rozwiązania są dostępne za darmo, jednak nie udostępniają one generatora opisu będącego w stanie wygenerować cały projekt aplikacji a ich interfejsy nie należą do intuicyjnych. Wprowadzenie generatora oraz intuicyjnego interfejsu użytkownika powinno uczynić Qubica istotną alternatywą dla istniejących już narzędzi.

Analizę istniejących rozwiązań przedstawia kolejny rozdział a zestawienie z rezultatami naszej pracy znajduje się natomiast w końcowej części pracy, gdzie zostaną przedstawione uzyskane przez nas wyniki oraz podsumowanie wykonanej przez nas pracy.

1.3 Cele pracy

Do najważniejszych celów niniejszej pracy dyplomowej należą:

- Analiza istniejących bibliotek oraz aplikacji szkieletowych realizujących mapowanie obiektowo-relacyjne w języku C++.

- Stworzenie własnej aplikacji szkieletowej realizującej mapowanie obiektowo-relacyjne w języku C++.
- Analiza porównawcza szybkości działania przykładowej aplikacji stworzonej w oparciu o Qubic a o inne istniejące narzędzia.
- Analiza porównawcza ilości kodu przykładowej aplikacji stworzonej w oparciu o Qubic a o inne istniejące narzędzia.

Do celów części praktycznej należą:

- Stworzenie intuicyjnego interfejsu użytkownika – aby sprawdzić w jakim stopniu udało się zrealizować założenia zostanie przeprowadzony test polegający na napisaniu tej samej aplikacji wykorzystującej Qubica oraz inne aplikacje szkieletowe i bibioteki, a następnie porównaniu ilości linii kodu stworzonych aplikacji.
- Stworzenie generatora opisu mapowania obiektowo-relacyjnego – jest to przedmiotem pracy Sebastiana, naszym wspólnym celem jest integracja obu modułów.
- Poprawne zrealizowanie założeń mapowania obiektowo-relacyjnego, a także jak najlepsza optymalizacja zapytań – aby sprawdzić w jakim stopniu udało się zrealizować założenia zostanie przeprowadzony test polegający na napisaniu tej samej aplikacji wykorzystującej Qubica oraz inne aplikacje szkieletowe i bibioteki, a następnie porównaniu ich wydajności.
- Uczynienie konfiguracji Qubica jak najprostszą.

1.4 Metoda badawcza

Studia literaturowe

Podstawowa literatura wykorzystana podczas pisania niniejszej pracy dotyczy specyfikacji języka C++ oraz zapytań w języku SQL, są to powszechne zagadnienia, więc wybór pozycji książkowych jest spory. Materiały dotyczące samego mapowania obiektowo-relacyjnego czy też aplikacji szkieletowej Qt pochodzą głównie z źródeł elektronicznych w języku angielskim, co jest związane z mniejszą popularnością tych tematyk.

Analiza istniejących rozwiązań

Analiza istniejących rozwiązań jest o tyle dobrą metodą badawczą, że stosując ją poznajemy praktyczne rozwiązania konkretnych problemów, czyli coś czego nie daje nam w żaden sposób teoria.

Stworzenie własnej aplikacji szkieletowej

Praktyka jest najczęściej najlepszą z dostępnych metod nauki i to właśnie podczas tworzenia własnej aplikacji szkieletowej realizującej mapowanie obiektowo-relacyjne można się najbardziej z danym tematem zapoznać. Wszystkie problemy, które pojawiały się poczas pisania Qubica musiały zostać w pewien sposób rozwiązane i to właśnie analiza tych problemów i ich rozwiązywanie było główną metodą badawczą wykorzystaną podczas pisania niniejszej pracy.

Analiza porównawcza oraz testy

Analizując wcześniej istniejące już rozwiązania i powównując je z własnym można dojść do najtrafniejszych wniosków. To właśnie na tym etapie często dowiadujemy się czy przyjęte przez nas założenia i zaproponowane rozwiązania były lepsze niż te z istniejących już rozwiązań.

1.5 Przegląd literatury w dziedzinie

Literatura dotycząca języka C++

Literatura dotycząca języka SQL

Literatura dotycząca mapowania obiektowo-relacyjnego

Literatura dotycząca aplikacji szkieletowej Qt

1.6 Układ pracy

Tematem niniejszej pracy jest mapowanie obiektowo-relacyjne w języku C++, zaś za jej główny cel przyjęto przeanalizowanie istniejących bibliotek oraz aplikacji szkieletowych realizujących mapowanie obiektowo-relacyjne oraz zaproponowanie własnej aplikacji szkieletowej.

W rozdziale 1 uzasadniony został wybór tematu pracy, opisane jej cele oraz problematyka i zakres. Rozdział 2 zawiera opis zagadnień teoretycznych dotyczą-

cych mapowania obiektowo-relacyjnego a także tworzenia aplikacji szkieletowych w języku C++.

Najważniejszym rezultatem pracy jest analiza oraz porównanie istniejących bibliotek programistycznych i aplikacji szkieletowych realizujących mapowanie obiektowo-relacyjne z stworzonym przez nas Qubiciem.

Rozdział 2

Zagadnienia teoretyczne dotyczące mapowania obiektowo-relacyjnego oraz aplikacji szkieletowych

2.1 Podstawowe definicje

Rozdział 3

Technologie i metody użyte w części badawczej

Rozdział 4

Analiza istniejących rozwiązań mapowania obiektowo-relacyjnego w języku C++

Rozdział 5

Aplikacja szkieletowa Qubic

5.1 Analiza wymagań

5.1.1 Wymagania funkcjonalne

5.1.2 Wymagania niefunkcjonalne

5.2 Projekt

W związku z tym, że jest to projekt aplikacji szkieletowej w rozdziale tym nie zostanie zamieszczony opis warstwy danych czy też widoku, ponieważ zaprojektowanie oraz zaimplementowanie tych warstw należy do zadań programisty korzystającego z Qubica. Znajdą się tutaj jednak opisy poszczególnych modułów które wspólnie składają się na stworzoną aplikację szkieletową.

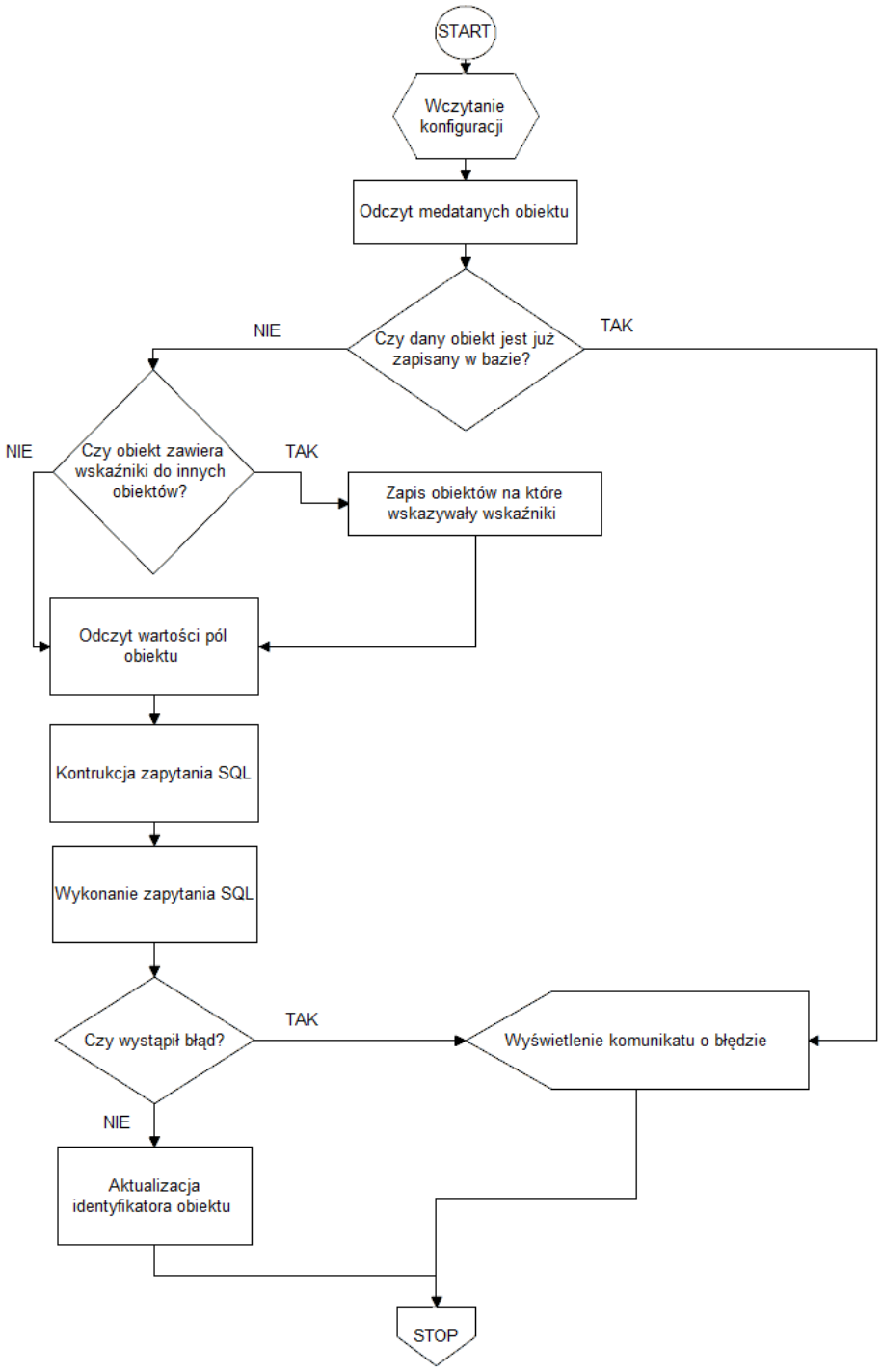
5.2.1 Interfejs CRUD

Kluczowym modułem narzędzi programistycznych realizujących mapowanie obiektowo-relacyjne jest interfejs CRUD umożliwiający manipulowanie danymi w bazie danych z poziomu kodu. Interfejs ten dostępny jest w podstaci co najmniej czterech metod, które jako argumenty przyjmują obiekty dziedziczące po jednej z klas Qubica – `QbPersistable`. Umożliwia to wykorzystanie polimorfizmu, a przecież podczas operacji na obiektach od samego początku nie znany jest dokładny typ obiektu.

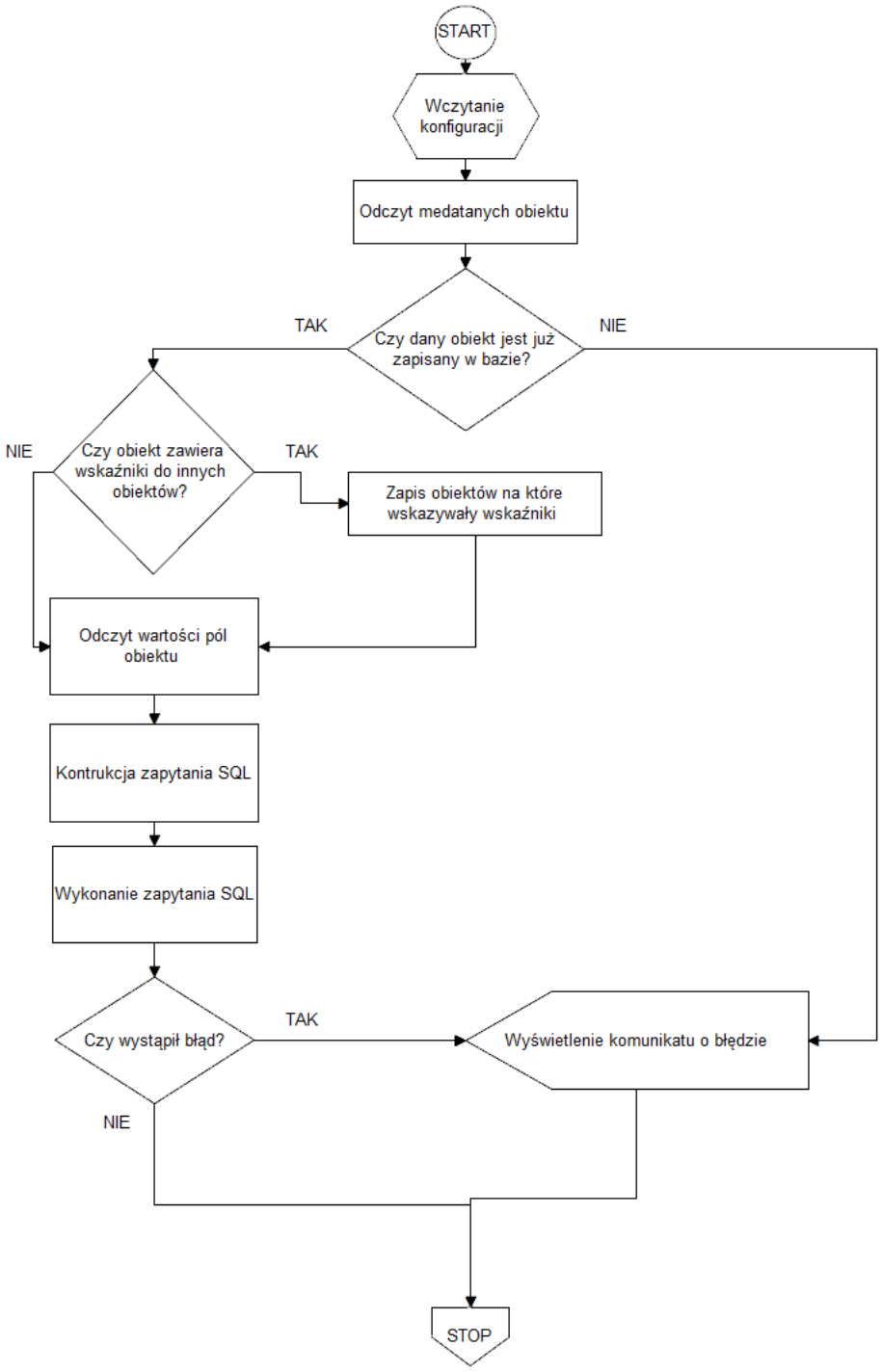
Aby mapowanie mogło skutecznie działać potrzebny jest jego jednoznaczny opis, część narzędzi je realizujących korzysta z adnotacji, które określają mapowanie pomiędzy konkretnymi polami klas a konkretnymi tabelami w bazie danych. W Qubicu problem ten został rozwiązany w trochę inny sposób, który umożliwia

stworzony generator opisu mapowania obiektowo-relacyjnego. Generowane pliki klas są tworzone według ściśle określonych zasad, co oznacza, że nazwy metod dostępowych odpowiadają w pewien sposób nazwom tabel i ich kolumn. Przykładowo dla kolumny o nazwie `NAME` w tabeli `EMPLOYEE` w pliku klasy `Employee` zostaną wygenerowane metody `getName()` oraz `setName(QString name)`. Wiedza ta została wykorzystana w trakcie korzystania z mechanizmu refleksji. Konfiguracja samego mapowania nazw została zapisana w pliku konfiguracyjnym Qubica.

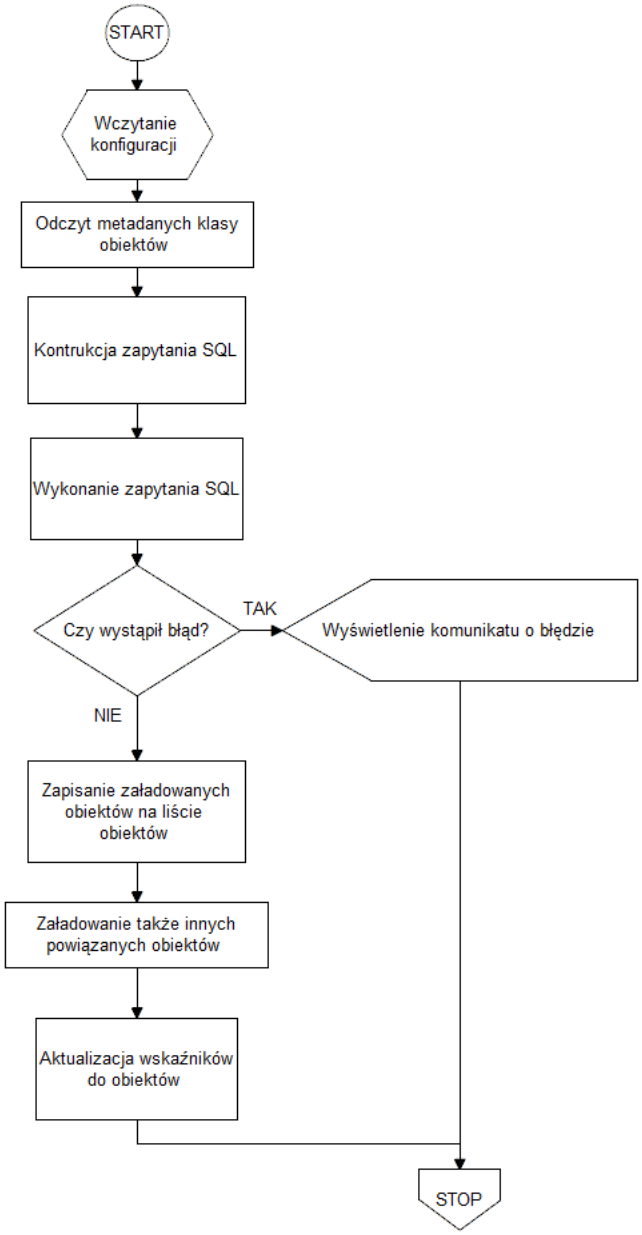
Schematy działania czterech podstawowych metod dostępu przedstawiają poniższe schematy blokowe:



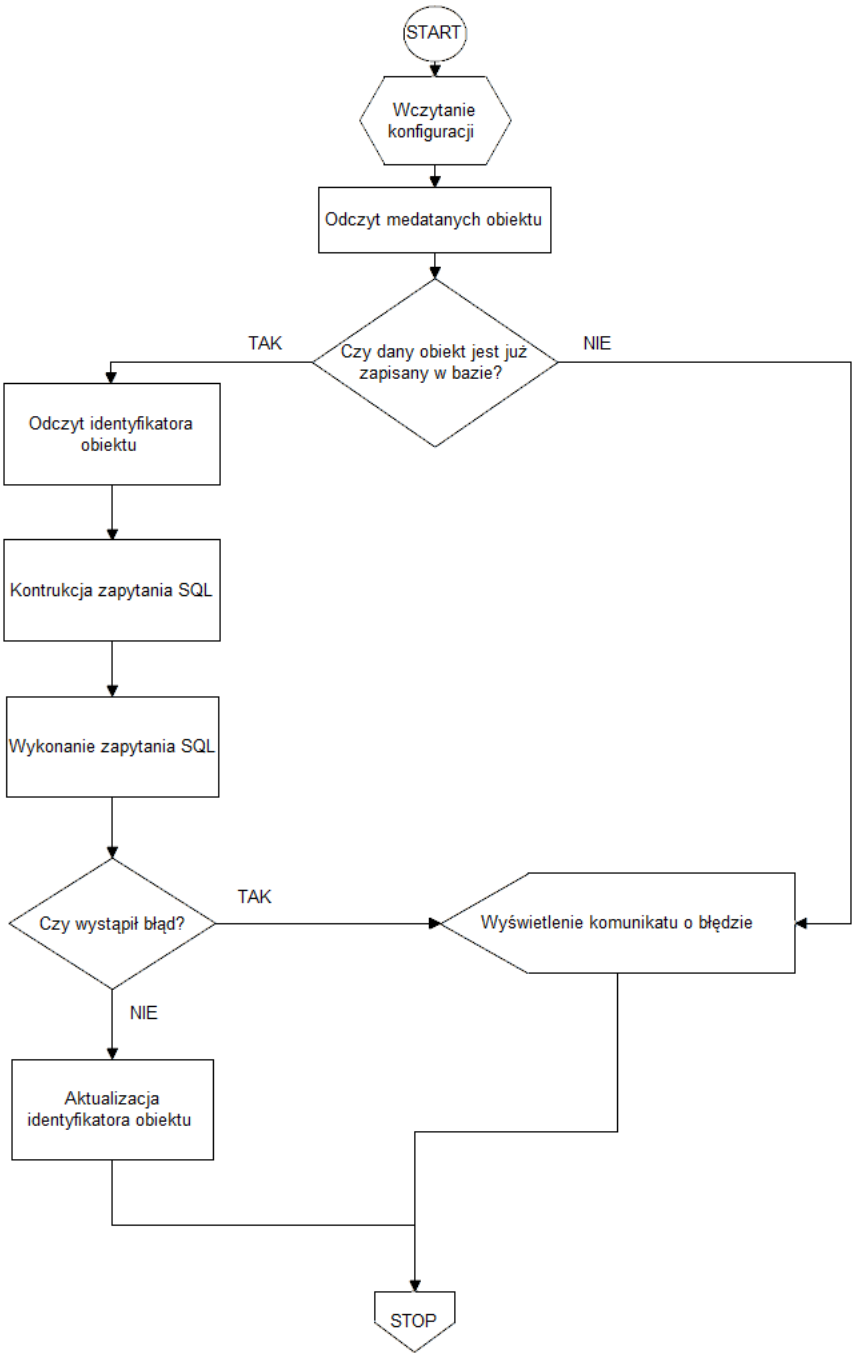
Rys. 5.1: Schemat blokowy metody zapisującej obiekty w bazie danych



Rys. 5.2: Schemat blokowy metody aktualizującej obiekty w bazie danych

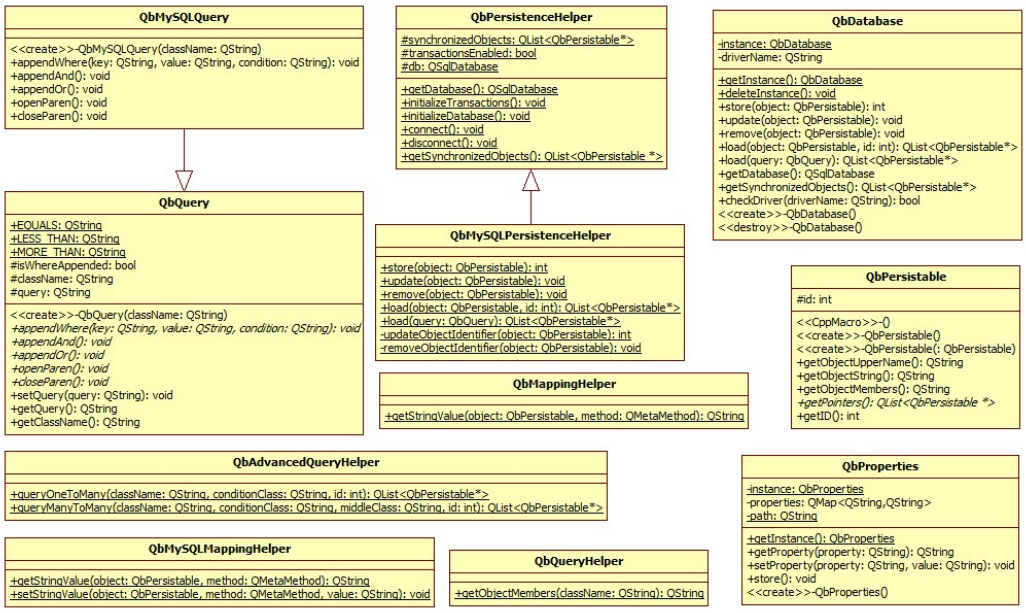


Rys. 5.3: Schemat blokowy metody ładującej obiekty z bazy danych



Rys. 5.4: Schemat blokowy metody usuwającej obiekty z bazy danych

5.2.2 Diagram klas



Rys. 5.5: Diagram klas

5.3 Implementacja

5.4 Konserwacja i inżynieria wtórna

5.5 Dokumentacja użytkownika

5.6 Przykładowa aplikacja wykorzystująca Qubica

5.7 Testy oraz ich wyniki

5.8 Perspektywy rozwoju Qubica

W celu dalszego rozwoju stworzonej aplikacji szkieletowej warto rozważyć wprowadzenie następujących usprawnień:

- System adnotacji – obecnie na opis mapowania składają się odpowienie nazwy funkcji oraz makra Qt. Istnieje jednak możliwość wprowadzenia własnych

makr, które miałyby opisywać mapowanie pomiędzy nazwami tabel z baz danych a odpowiednimi polami klas napisanych z języku C++. Dzięki temu zaistniałaby możliwość uniezależnienia nazw pól klas od nazw tabel w bazie danych.

- Interfejs zapytań – choć jest już zaimplementowany, nadal nie udostępnia on wszystkich możliwych funkcjonalności języka SQL. Implementacja obsługi takich poleceń jak JOIN czy UNION z pewnością byłaby dodatkowym atutem.

- Identyfikacja tabel także za pomocą kluczy złożonych – w tej chwili tabele identyfikowane są za pomocą kluczów głównych, co z kolei wymusza ich nadawanie w każdej z tabel.

- Pamięć podręczna – wprowadzenie pamięci podręcznej może znacznie polepszyć wydajność w przypadku ciągłych operacji na tych samych danych.

- Konfiguracja z poziomu kodu – obecnie większość konfiguracji jest zapisana w plikach konfiguracyjnych i tylko tam może być zmieniana, w celu rozwoju wprowadzenie dodatkowej możliwości jego konfiguracji wydaje się być dobrym pomysłem.

- Wsparcie dla różnych rodzajów baz danych – wprowadzenie tego usprawnienia ogranicza się do implementacji kilku interfejsów dla innych niż MySQL rodzajów baz danych. Biorąc pod uwagę możliwość wzorowania się na zaimplementowanej już logice nie powinno to stworzyć problemu gdy zaistnieje taka konieczność.

- Serializacja danych – dodanie możliwości serializacji może okazać się użyteczne w przypadku pracy z dużymi ilościami danych, w tym celu można skorzystać z wielu istniejących już bibliotek udostępniających tę możliwość.

- Internacjonalizacja – w tej chwili wszystkie logi zlokalizowane są w języku angielskim, istnieje jednak możliwość zmiany obecnego stanu poprzez wykorzystanie modułu translacji udostępnianego przez Qt.

- Wielowątkowość – wykorzystanie wielowątkowości w przypadku mapowania obiektowo relacyjnego z pewnością nie należy do najłatwiejszych zadań, jednak znacznie może to usprawnić wykonywanie bardziej wymagających operacji.

Rozdział 6

Podsumowanie

6.1 Dyskusja wyników

6.2 Perspektywy rozwoju pracy

Dzięki zrealizowaniu pracy poprawie uległa wydajność. Ponadto, o ? % skrócony został czas. Które cele pracy udało sie zrealizować? Co z tego wynika? Które cele pracy pozostały niezrealizowane i dlaczego?

Bibliografia

[1] Gręboszz, Jerzy. Symfonia C++. Standard. Wyd. 3. Kraków, 2013. ISBN 978-83-7366-134-4.

[2] C++ Language Tutorial. `http://www.cplusplus.com/doc/`.

[3] Viescas, John. SQL Queries for Mere Mortals. 2001. ISBN 83-7279-152-X.

Spis rysunków

Spis tabel

Załączniki

1.