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 wykonując jej część za nich.

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 co raz bardziej 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.

Tutaj 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.
- Aby efektownie korzystać z narzędzi do mapowania obiektowo-relacyjnego wymagana jest od nas ich dobra znajomość.
- 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 zakładając, że pozostała funkcjonalność mapowania obiektowo-relacyjnego zostanie zrealizowana poprawnie.

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++ – przeanalizowanie istniejących już narzędzi umożliwi dokładniejsze zapoznanie się z tematyką mapowania obiektowo-relacyjnego a także ze sposobem działania istniejących już rozwiązań, co umożliwi zaczerpnięcie najciekawszych pomysłów dla Qubica a także wskaże elementy, które mogą ulec w nim poprawie.

- Stworzenie własnej aplikacji szkieletowej realizującej mapowanie obiektowo-relacyjne w języku C++ – ułatwi dokładniejsze poznanie mechanizmów działających podczas mapowania obiektowo-relacyjnego oraz sposobów rozwiązania pojawiających się problemów.

- Porównanie szybkości działania oraz ilości kodu wymaganego do stworzenia przykładowej aplikacji stworzonej w oparciu o Qubic a o inne istniejące narzędzia – przeprowadzenie testów pozwoli stwierdzić czy przyjęte założenia i zastosowane rozwiązania wpłynęły na stworzenie wartej uwagi aplikacji szkieletowej realizującej mapowanie obiektowo-relacyjne.

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

1.4.1 Studia literaturowe

Literatura wykorzystana podczas pisania niniejszej pracy dotyczy głównie czterech zagadnień, czyli tworzenia aplikacji szkieletowych w języku C++, tworzenia zapytań w języku SQL, aplikacji szkieletowej Qt oraz mapowania obiektowo-relacyjnego. Pierwsze dwa należą są dość popularne, dlatego też wybór pozycji książkowych jest dość spory. Na temat mapowania obiektowo-relacyjnego trudniej jednak znaleźć podobną ilość książek, jednak istnieje spora ilość artykułów w wersji elektronicznej. Najczęściej są one napisane w języku angielskim. Tytuły wykorzystanych pozycji bibliograficznych wraz z ich krótkimi opisami znajdują się w kolejnym rozdziale.

1.4.2 Analiza istniejących rozwiązań

Poza podstawowym źródłem informacji jakim są studia literaturowe podczas pisania tej pracy przeprowadzona została analiza istniejących już narzędzi realizujących mapowanie obiektowo-relacyjne. Analiza taka umożliwia poznanie praktycznych rozwiązań problemów pojawiających się podczas prac badawczych nad daną tematyką.

1.4.3 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.

1.4.4 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

1.5.1 Literatura dotycząca języka C++ oraz Qt

W celu zasięgnięcia informacji na temat języka C++ i zagadnień z nim związanych najczęściej wykorzystywaną pozycją książkową była „Symfonia" Jerzego Grębosza [1]. Najlepszym jej określeniem jest „kurs programowania w języku C++", opisane zostały w niej jednak zagadnienia dotyczące nie tylko języka C++, a także co istotne dla autora niniejszej pracy zagadnienia dotyczące obiektowości.

Poza tym wartościowym źródłem wiedzy podczas tworzenia Qubica była specyfikacja języka C++ [2] oraz dokumentacja aplikacji szkieletowej Qt [3].

1.5.2 Literatura dotycząca języka SQL

Kluczowym zadaniem Qubica jest tworzenie jak najefektywniejszych zapytań w języku SQL, wiedza autora na ten temat pochodzi w głównej mierze z książki Johna Viescasa o tytule „SQL Queries for Mere Mortals" [4]. Wyjaśnione zostały w niej zagadnienia dotyczące tworzenia zapytań w języku SQL oraz podstawy związane z bazami danych.

W tym przypadku także wartościowe okazały się specyfikacje na przykład języka MySQL [5].

1.5.3 Literatura dotycząca mapowania obiektowo-relacyjnego

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 stworzenie własnej aplikacji szkieletowej.

Najważniejszym celem pracy jest przeanalizowanie istniejących narzędzi służących do mapowania obiektowo-relacyjnego w języku C++ oraz stworzenie na podstawie tej analizy własnej aplikacji szkieletowej realizującej to samo zagadnienie.

Rodział 1 otwiera uzasadnienie wyboru tematu pracy, opisane zostały w nim także cele pracy, jej problematyka, zakres, wykorzystane metody badawcze, przegląd literatury oraz jej układ.

Rozdział 2 zawiera objaśnienia podstawowych zagadnień teotetycznych związanych z mapowaniem obiektowo-relacyjnym oraz tworzeniem aplikacji szkieletowych w języku C++.

Rozdział 3 przedstawia analizę istniejących już narzędzi służących do mapowania obiektowo-relacyjnego napisanych w języku C++.

Rozdział 4 zawiera opis technologii oraz metod wykorzystanych w części badawczej niniejszej pracy.

Rozdział 5 przedstawia projekt aplikacji szkieletowej Qubic, postawione wymagania, opis jego implementacji, wyniki testów oraz końcowe porównanie z wcześniej analizowanymi narzędziami w rozdziale 3.

Rozdział 6 zawiera podsumowanie niniejszej pracy inżynierskiej, biorąc pod szczególną uwagę dyskusję wyników oraz dalsze prespektywy rozwoju pracy.

Rozdział 2

Zagadnienia teoretyczne dotyczące mapowania obiektowo-relacyjnego oraz aplikacji szkieletowych

2.1 Programowanie obiektowe

2.2 Relacyjne bazy danych

2.3 Mapowanie obiektowo-relacyjne

Rozdział 3

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

Rozdział 4

Technologie i metody użyte w części badawczej

Rozdział 5

Aplikacja szkieletowa Qubic

5.1 Analiza wymagań

5.1.1 Wymagania funkcjonalne

5.1.2 Wymagania niefunkcjonalne

+ aplikacje szkieletowe i biblioteki? czy tylko jedno z nich? + poprawić książkę do sql + co tak na prawdę będzie testowane/analizowane? + literatura dot. orm + 3x teoria + poprawki od początku + definicje polimorfizmu, refleksji, operacji CRUD, diagram UML/klas + dokładny opis terminologii pojęć zasadniczych dla tematu pracy, którymi autor będzie się posługiwał przy realizacji głównych celów pracy + analiza istniejacych rozwiazan + technologie (sprzęt, oprogramowanie, serwer bazy danych, technologie i metodologie, język, biblioteki, wzorce) + 2x wymagania

5.2 Projekt

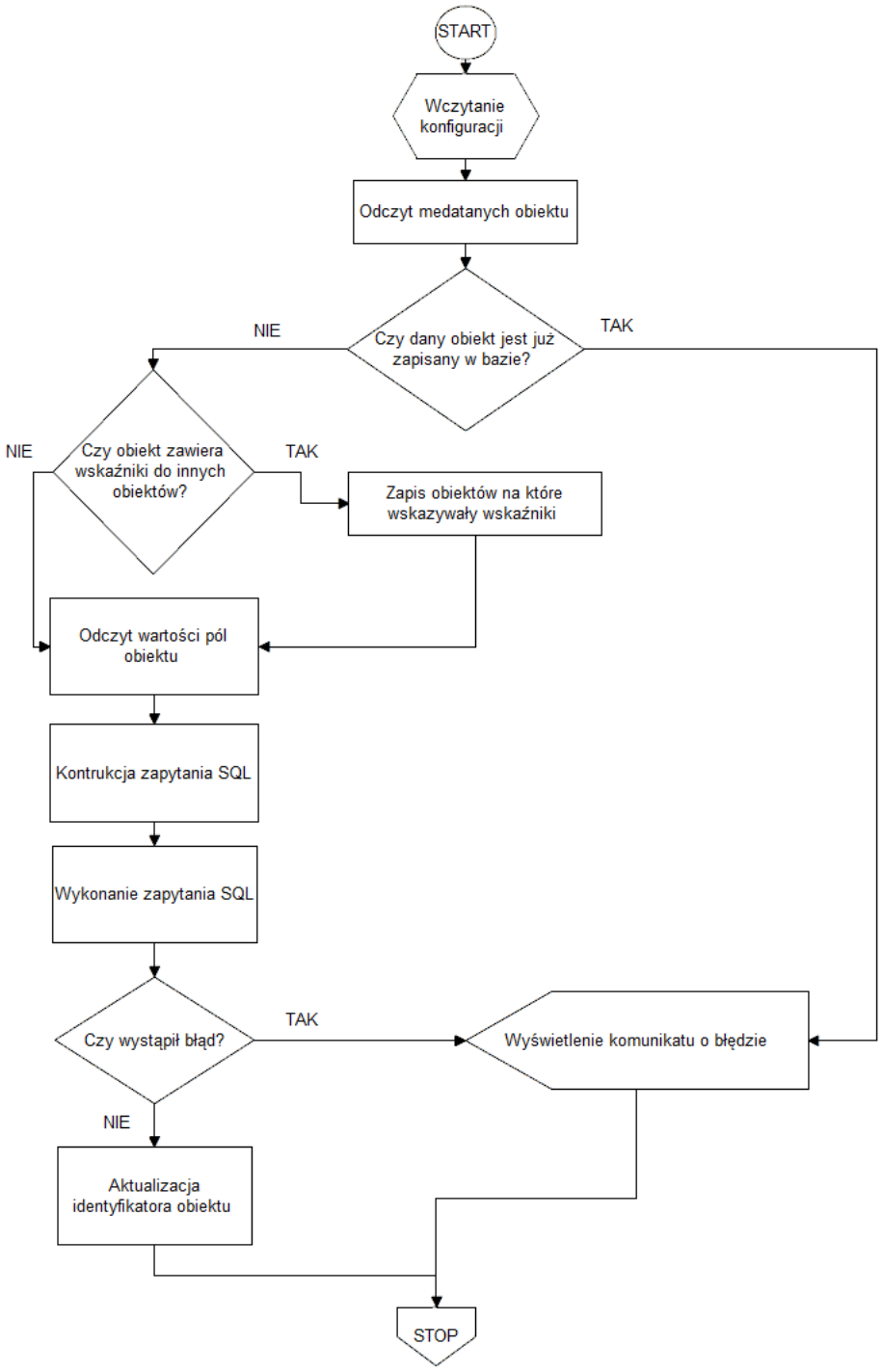
W związku z tym, że jest to projekt aplikacji szkieletowej w rozdziale tym nie zostanie zamieszczony opis warstwy danych czy też widoku, ponieważ zaprojektowanie oraz zaimplementowanie tych warstw należy do zadań programisty korzystającego z Qubica. Znajdą się tutaj jednak opisy poszczególnych modułów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

5.2.1 Interfejs CRUD

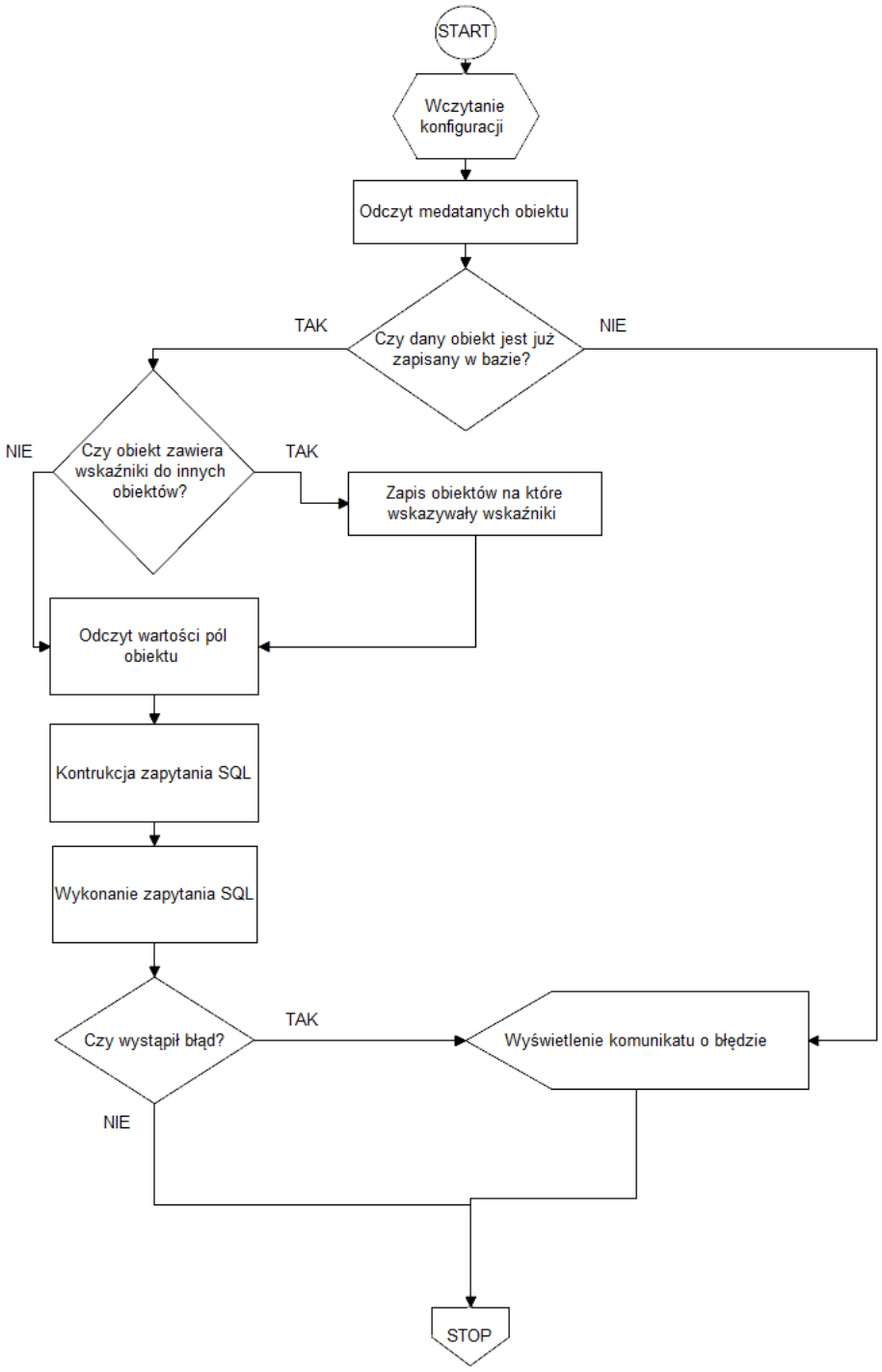
Kluczowym modułem narzędzi programistycznych realizujących mapowanie obiektowo-relacyjne jest interfejs CRUD umożliwiający manipulowanie danymi w bazie danych z poziomu kodu. Interfejs ten dostępny jest w podstaci co najmniej czterech metod, które jako argumenty przyjmują obiekty dziedziczące po jednej 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.

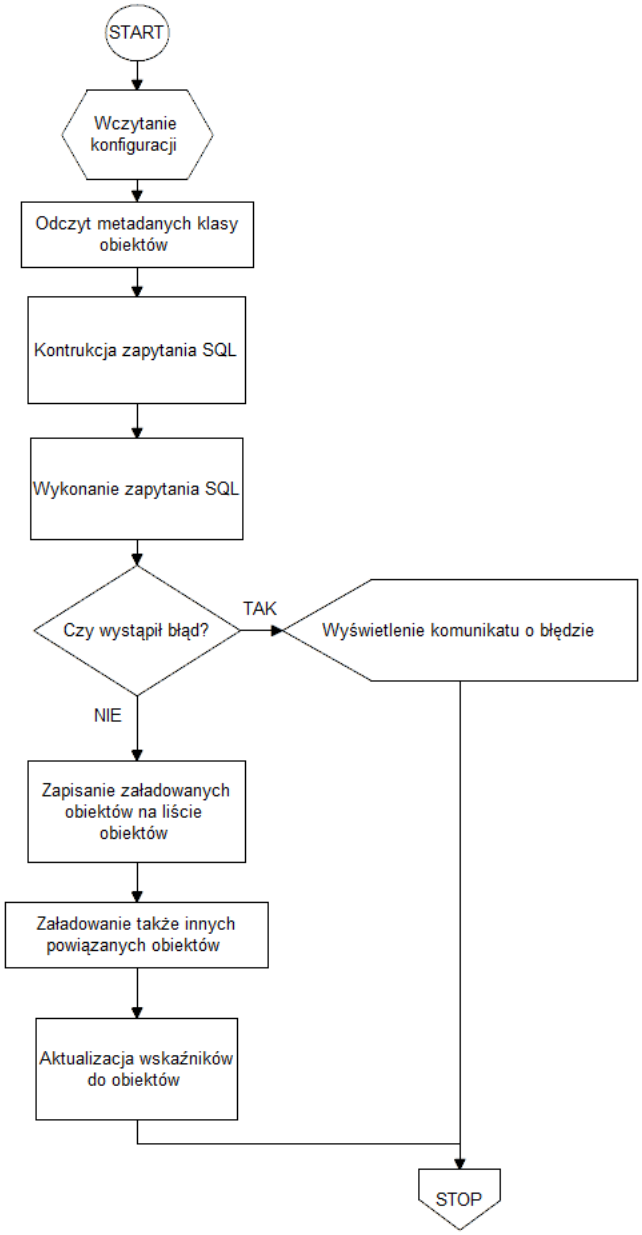
Schematy działania czterech podstawowych metod dostępu przedstawiają poniższe schematy blokowe:



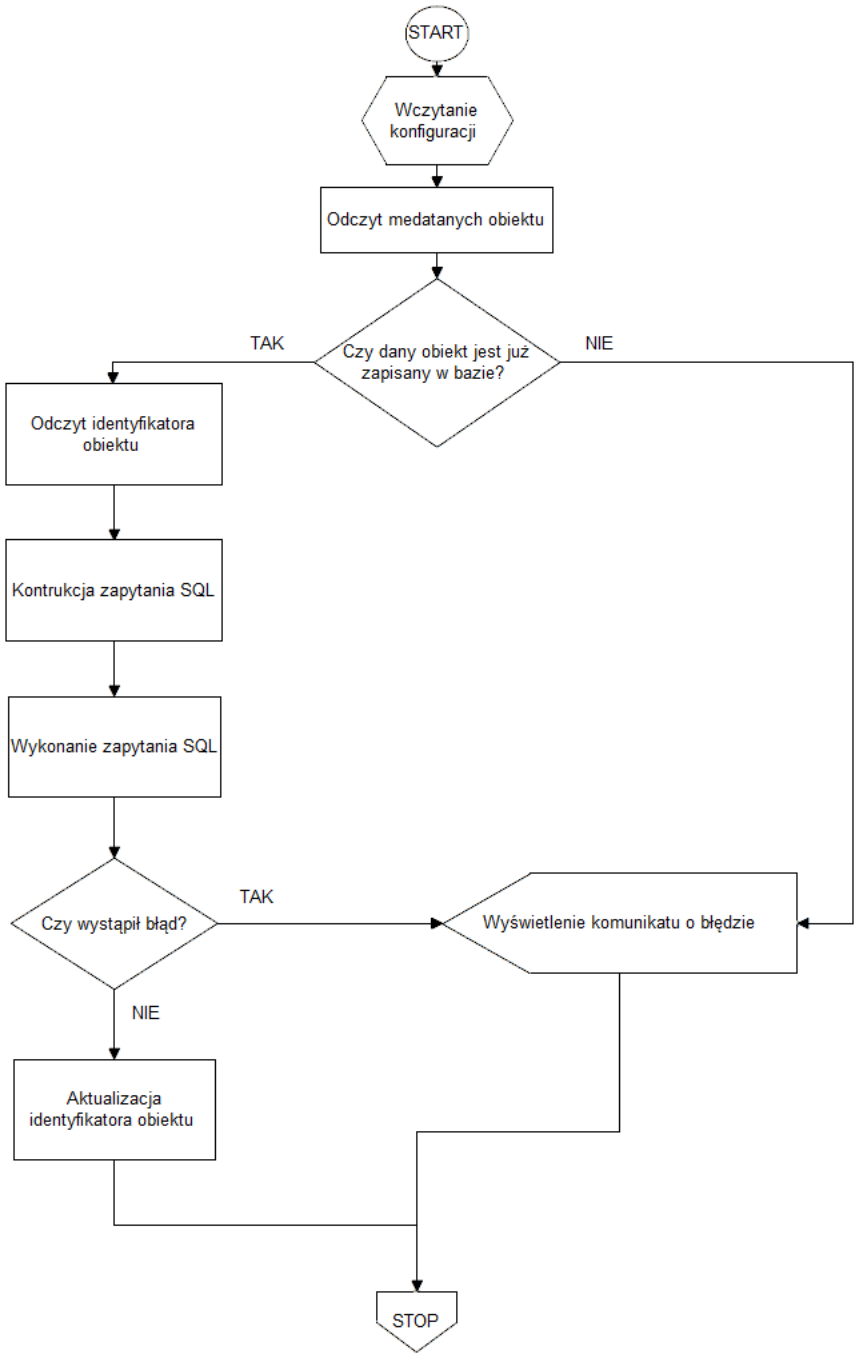
Rys. 5.1: Schemat blokowy metody zapisującej obiekty w bazie danych



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

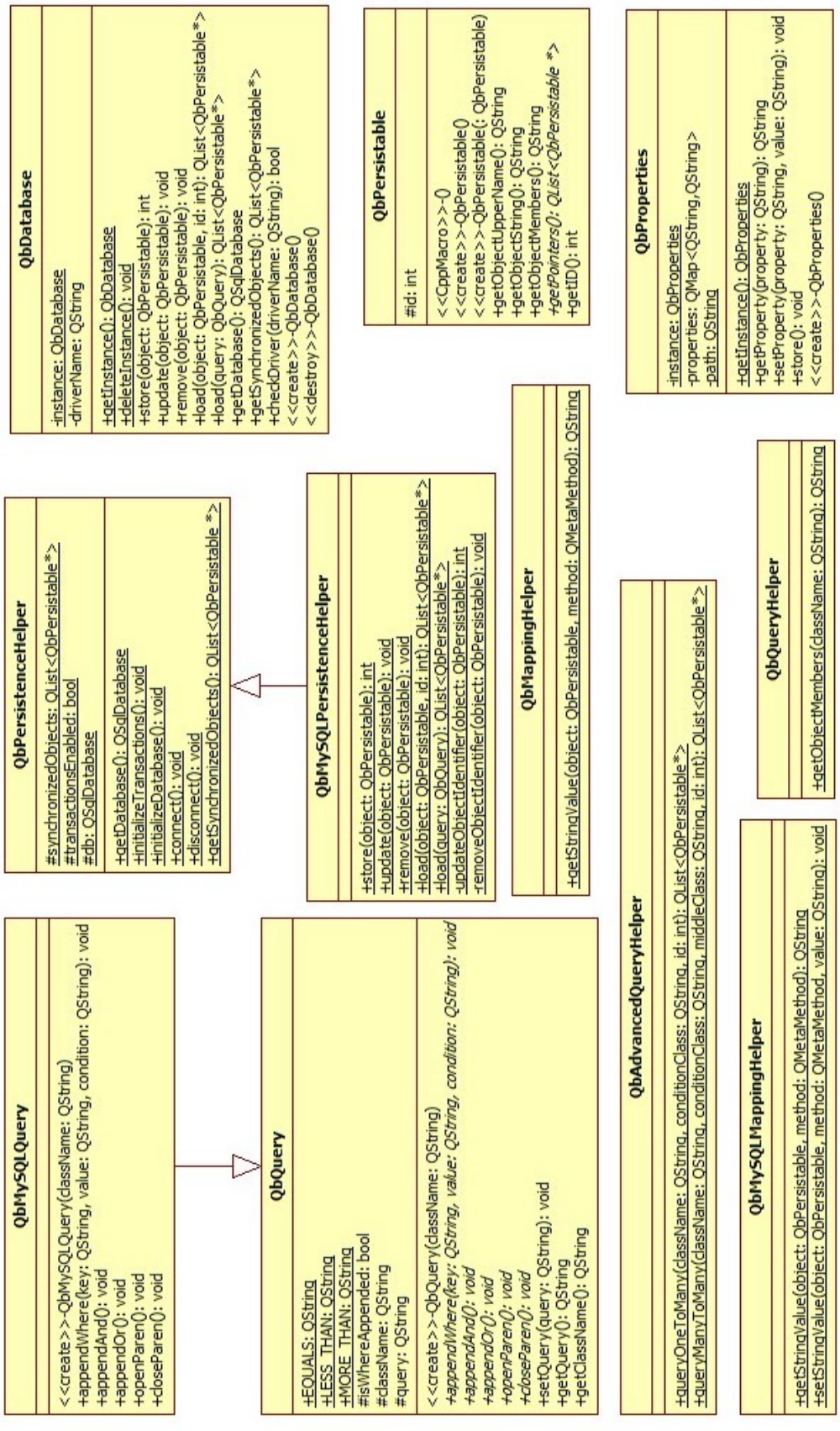


Rys. 5.3: Schemat blokowy metody ładującej obiekty z bazy danych



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

5.2.2 Diagram klas


QbDatabase
-instance: QbDatabase
-driverName: QString
+getInstance(): QbDatabase
+deleteInstance(): void
+store(object: QbPersistable): int
+update(object: QbPersistable): void
+remove(object: QbPersistable): void
+load(object: QbPersistable, id: int): QList<QbPersistable*>
+load(query: QbQuery): QList<QbPersistable*>
+getDatabase(): QSqlDatabase
+getSynchronizedObjects(): QList<QbPersistable*>
+checkDriver(driverName: QString): bool
<<create>>-QbDatabase()
<<destroy>>-QbDatabase()
QbPersistable
#id: int
<<CppMacro>>-()
<<create>>-QbPersistable()
<<create>>-QbPersistable(: QbPersistable)
+getObjectUpperName(): QString
+getObjectString(): QString
+getObjectMembers(): QString
+getPointers(): QList<QbPersistable *>
+getID(): int
QbProperties
-instance: QbProperties
-properties: QMap<QString,QString>
-path: QString
+getInstance(): QbProperties
+getProperty(property: QString): QString
+setProperty(property: QString, value: QString): void
+store(): void
<<create>>-QbProperties()
QbPersistenceHelper
#synchronizedObjects: QList<QbPersistable*>
#transactionsEnabled: bool
#db: QSqlDatabase
+getDatabase(): QSqlDatabase
+initializeTransactions(): void
+initializeDatabase(): void
+connect(): void
+disconnect(): void
+getSynchronizedObjects(): QList<QbPersistable *>
QbMySQLPersistenceHelper
+store(object: QbPersistable): int
+update(object: QbPersistable): void
+remove(object: QbPersistable): void
+load(object: QbPersistable, id: int): QList<QbPersistable*>
+load(query: QbQuery): QList<QbPersistable*>
-updateObjectIdentifier(object: QbPersistable): int
-removeObjectIdentifier(object: QbPersistable): void
QbMappingHelper
+getStringValue(object: QbPersistable, method: QMetaMethod): QString
QbMySQLQuery
<<create>>-QbMySQLQuery(className: QString)
+appendWhere(key: QString, value: QString, condition: QString): void
+appendAnd(): void
+appendOr(): void
+openParen(): void
+closeParen(): void
QbQuery
+EQUALS: QString
+LESS_THAN: QString
+MORE_THAN: QString
#isWhereAppended: bool
#className: QString
#query: QString
<<create>>-QbQuery(className: QString)
+appendWhere(key: QString, value: QString, condition: QString): void
+appendAnd(): void
+appendOr(): void
+openParen(): void
+closeParen(): void
+setQuery(query: QString): void
+getQuery(): QString
+getClassName(): QString
QbAdvancedQueryHelper
+queryOneToMany(className: QString, conditionClass: QString, id: int): QList<QbPersistable*>
+queryManyToMany(className: QString, conditionClass: QString, middleClass: QString, id: int): QList<QbPersistable*>
QbMySQLMappingHelper
+getStringValue(object: QbPersistable, method: QMetaMethod): QString
+setStringValue(object: QbPersistable, method: QMetaMethod, value: QString): void
QbQueryHelper
+getObjectMembers(className: QString): QString

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

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

[3] Qt Project Documentation. http://qt-project.org/doc/.

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

[5] MySQL Documentation. http://dev.mysql.com/doc/

Spis rysunków

Spis tabel

Załączniki

1.