UNIWERSYTET MIKOŁAJA KOPERNIKA W TORUNIU

W Y D Z I A Ł M A T E M A T Y K I I I N F O R M A T Y K I

Paweł Piątkowski

GRAFICZNY EDYTOR ZAPYTAŃ DO BAZY DANYCH DB4O

Toruń 2010

Praca magisterska
napisana pod kierunkiem
dr hab. Krzysztofa Stencla prof. UMK

Streszczenie

Celem niniejszej pracy magisterskiej było stworzenie wygodnego interfejsu pozwalającego tworzyć zapytania do obiektowej bazy danych db4o. W wyniku prac powstał graficzny edytor zapytań napisany jako wtyczka do Eclipse IDE. Edytor pozwala wizualnie tworzyć graf zapytania. Graf ten określa warunki jakie muszą spełniać obiekty aby znaleźć się w zbiorze rezultatów. Na podstawie grafu generowany jest kod języka Java realizujący zapytanie używając SODA API - jednego z dostępnych w db4o mechanizmów tworzenia zapytań. Aplikacja napisana jest z użyciem biblioteki GEF (Graphical Editing Framework).

Spis treści

Wstęp

TODO

Rozdział I
Wprowadzenie

1. Omówienie problemu

1.1. Obiektowe Bazy Danych

Jednym z głównych powodów rozwoju obiektowych baz danych było rozwiązanie tzw. problemu niezgodności impedancji (ang. impedence mismatch). Problem ten powstaje ze względu na fundamentalne różnice pomiędzy paradygmatem relacyjnym a obiektowym. Na paradygmacie relacyjnym oparte są Relacyjne Systemy Zarządzania Bazą Danych (RSZBD). Natomiast obiektowe języki programowania wyrastają z paradygmatu obiektowego. Gdy łączymy ze sobą te dwa światy nieuchronnie napotykamy trudności wynikające z różnic pomiędzy nimi. Programista piszący program w duchu OOP (Object Oriented Programming) zmuszony jest wyjść ze świata obiektowego chcąc uzyskać dostęp do danych zapisanych w relacyjnej bazie danych.

Załóżmy, że w programie napisanym obiektowo istnieje potrzeba utrwalenia (ang. persist) pewnych obiektów. Typowym rozwiązaniem takiego problemu za pomocą RSZBD byłoby stworzenie w bazie danych tabeli odwzorowującej klasę obiektów, które chcemy utrwalać. Następnie należy napisać kod odpowiedzialny za zapisywanie informacji zawartych w obiektach do utworzonej tabeli. Odbywa się to poprzez wykonanie odpowiedniego zapytania SQL. Analogicznie postępujemy chcąc odczytać utrwalone dane. Wykonujemy zapytanie, powołujemy do życia nowe obiekty i wypełniamy je danymi otrzymanymi z bazy danych. Takie podejście zdaniem autora jest kłopotliwe. Wymaga ono od programisty ciągłego manewrowania na granicy świata relacyjnego i obiektowego. Ponadto, konieczne jest używanie w kodzie zapytań SQL co jest nieczytelne, czyni kod niepodatnym na automatyczny refactoring i jest źródłem zagrożenia bezpieczeństwa danych (SQL injection).

Przedstawiony scenariusz jest oczywiście uproszczony w celu uwypuklenia problemu niekompatybilności paradygmatów relacyjnego i obiektowego. W codziennej praktyce zwykle stosuje się gotowe narzędzia ORM (Object-Relational Mapping) takie jak Hibernate czy Toplink. Ułatwiają one tworzenie odwzorowania obiekt – relacja uwalniając programistę od żmudnego procesu ręcznego odwzorowywania obiektów na tabele. Są one z sukcesem stosowane, jednak mają wiele wad. Z tego powodu wielu ekspertów uważa je za problematyczne - obniżają wydajność, komplikują projekt, zwiększają koszty utrzymania oprogramowania. Ten temat szeroko omawia (Neward, 2006).

Obiektowe Systemy Zarządzania Bazą Danych (OSZBD) wyrastają bezpośrednio z paradygmatu programowania obiektowego. Zasobami przechowywanymi w obiektowej bazie danych są obiekty. Dzięki temu komunikacja programu napisanego w stylu OOP z obiektową bazą danych jest prosta i elegancka. Gdy chcemy utrwalić obiekt czynimy to w jednej linijce kodu. Wczytując dane otrzymujemy obiekty, z którymi możemy pracować. Nie ma tu potrzeby odwzorowywania tabel na obiekty, które to zwykle jest konieczne w przypadku baz relacyjnych.

Tak fundamentalnie różne podejście do przechowywania danych w obu systemach zarządzania bazą danych skutkuje odmiennymi metodami dostępu do nich. Istnieje wiele metod dostępu do danych w zapisanych w OSZBD. W niniejszej pracy skupię się na jednym z trzech mechanizmów dostępnych w obiektowej bazie danych db4o – zapytaniach typu SODA.

1.2. SODA - Simple Object Database Access

SODA to niskopoziomowe API umożliwiające tworzenie zapytań do bazy danych db4o. Powstało w 2002 roku jako projekt Open Source rozwijany przez Carla Rosenbergera na potrzeby tworzonej wówczas przez niego obiektowej bazy danych – db4o. W tym czasie SODA była jedynym mechanizmem dostępu do danych oferowanym przez db4o. W chwili obecnej db4o oferuje cztery takie mechanizmy:

- Native Queries
- Query By Example
- LINQ
- SODA

Należy w tym miejscu zaznaczyć, iż db4o rozwijana jest dla dwóch platform – .NET i Java. LINQ dostępny jest tylko w wersji db4o przeznaczonej dla platformy .NET. Natomiast pozostałe trzy metody dostępu do danych znajdziemy w obu wersjach db4o. Warto wiedzieć, że SODA jest podstawą wszystkich innych mechanizmów dostępu do danych – zapytania napisane za pomocą LINQ, Query By Example czy Native Queries są tłumaczone na zapytania SODA przez silnik db4o i dopiero w takiej postaci są faktycznie wykonywane.

SODA nie jest pozbawiona wad. Tworzenie zapytań wymaga używania łańcuchów znakowych, co sprawia iż SODA w przeciwieństwie do Native Queries, nie poddaje się refactoringowi. Co więcej, błąd w nazwie pola nie będzie wykryty podczas kompilacji ani podczas wykonywania programu. Odwołanie poprzez łańcuch znakowy do nieistniejącego pola klasy zaowocuje pustym zbiorem wynikowym co może być źródłem trudnych do znalezienia błędów. SODA jest również mało intuicyjna. Z tych powodów Native Queries są zalecaną metodą tworzenia zapytań dla typowych zastosowań. Z drugiej strony wielką zaletą zapytań SODA jest ich szybkość – chociażby z tego powodu każdy użytkownik db4o powinien potrafić używać tego API.

W zapytaniach typu SODA programista tworzy graf zapytania za pomocą wywołań SODA API. Graf ten określa jakie warunki muszą spełniać obiekty aby znaleźć się w zbiorze wynikowym zapytania. Tworzenie zapytań typu SODA opiera się na wywołaniach metody `descend()`, która przechodzi w głąb klasy, będącej punktem wejścia, pozwalając programiście dostać się do jej zagnieżdżonych obiektów. Za pomocą metody `constrain()` określa się ograniczenia nakładane na obiekty kandydujące do znalezienia się w zbiorze wynikowym.

Dołączany do dystrybucji db4o dokument „db4o Reference Documentation” opisuje proces tworzenia zapytania SODA jako kombinację poniższych pięciu podstawowych idiomów:

- utwórz główny węzeł grafu zapytania

```
Query root = db.query();
```

- dodaj ograniczenia do dowolnego węzła grafu

```
root.constrain(Foo.class);
```

- przejdź z dowolnego węzła do dowolnego z jego podwęzłów

```
Query foo = root.descend(„foo”);
Query bar = root.descend(„bar”);
```

- dodaj kolejne ograniczenia do dowolnego węzła

```
Constraint fooConstraint = foo.constrain(5);
Constraint barConstraint = bar.constrain(„F”);
```

- ustaw tryb ewaluacji ograniczenia

```
fooConstraint.greater().not();
barConstraint.startsWith(true);
barConstraint.like();
```

Zdaniem autora warto do tej listy dodać punkt szósty:

- ustanów logiczne relacje pomiędzy ograniczeniami

```
fooConstraint.or(barConstraint);
```

Powyższe idiomy można dowolnie, logicznie łączyć w celu uzyskania złożonych zapytań z wieloma warunkami. Prześledźmy proces tworzenia zapytania na przykładach.

1.2.1. Przykłady

Zdefiniujmy klasę `Pracownik`, która posłuży nam jako przykład:[1]

```
class Pracownik {
      Pracownik szef;
      String imie;
      int placa;
};
```

W poniższych przykładach zakładamy, że mamy obiekt `db` typu `ObjectContainer`, który jest naszym punktem dostępu do danych zapisanych w bazie.

Przykład 1 - „pracownicy, których szef ma na imię Jan”

Szukamy wszystkich pracowników, których szef ma na imię Jan. Za pomocą SODA takie zapytanie możemy wyrazić w następujący sposób:

```
1  Query query = db.query();
```

1 Wszystkie fragmenty kodu zawarte w pracy są napisane w języku Java - o ile nie zaznaczono inaczej.

```
2  query.constrain(Pracownik.class);
3  query.descend("szef").descend("imie").constrain("Jan");
4  ObjectSet result = query.execute();
```

W pierwszej linii uzyskujemy dostęp do obiektu `Query` wywołując metodę `query()` na obiekcie `ObjectContainer`. Następnie określamy, że chcemy uzyskać obiekty klasy `Pracownik`. Trzecia linia jest zdecydowanie najciekawsza – za pomocą metody `descend()` przechodzimy w głąb klasy `Pracownik`. Najpierw przechodzimy do pola `szef`, a z niego do pola `imie`. W tym momencie mamy dostęp do interesującego nas pola zatem nakładamy na nie ograniczenie. W linii czwartej wykonujemy zapytanie otrzymując kolekcję obiektów klasy `Pracownik`.

Kluczowym elementem SODA API jest metoda `descend()`. Pozwala ona dostać się do dowolnego pola klasy co jest niezbędne, w przypadku gdy chcemy nałożyć na nie ograniczenie. Tę podróż w głąb klasy można intuicyjnie przedstawić w postaci grafu (Diagram 1).

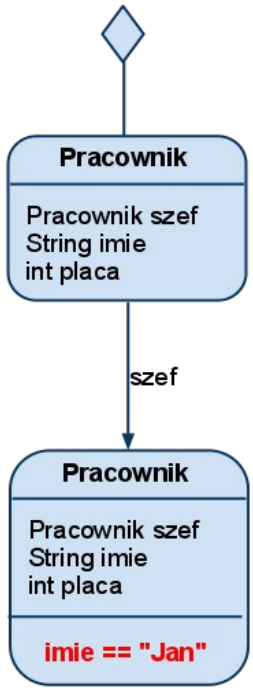

Diagram 1

Romb reprezentuje punkt wejścia. Węzeł połączony z rombem określa klasę obiektów zwracanych przez zapytanie – w tym przypadku będą to obiekty klasy `Pracownik`. Niżej widzimy wychodzącą z węzła krawędź oznaczoną etykietą „szef”. Krawędź ta oznacza, że sięgamy do pola `szef` klasy `Pracownik`. W rozwiniętym polu `szef` sięgamy do pola `imie` i stawiamy warunek `imie == „Jan”`.

Zdaniem autora powyższy graf w sposób czytelny i intuicyjny reprezentuje zapytanie „pracownicy, których szef ma na imię Jan”. Za pomocą podobnych grafów można wizualizować

zapytania typu SODA, co ułatwia i przyspiesza ich analizę. Można argumentować, iż kod zapytania jest na tyle prosty, że nie ma potrzeby odwoływać się do pomocy wizualnych. O ile w przypadku powyższego zapytania można się z taką argumentacją zgodzić, to w przypadku bardziej skomplikowanych zapytań diagram jest nieocenioną pomocą. Postaram się pokazać to za pomocą następnego przykładu.

Przykład 2 - „pracownicy, których szef ma na imię Jan zarabiający więcej niż 1000”

W przykładzie drugim pokażemy, w jaki sposób można łączyć ze sobą pojedyncze ograniczenia tworząc złożone zapytania. Weźmy zapytanie z pierwszego przykładu jako punkt wyjścia i dodajmy do niego dodatkowe ograniczenie.

```
1  Query query = db.query();
2  query.constrain(Pracownik.class);
3  Query imieSzefa = query.descend("szef").descend("imie");
4  Constraint imieJan = imieSzefa.constrain("Jan");
5  Query placa = query.descend("placa");
6  Constraint placa1000 = placa.constrain(1000).greater();
7  placa1000.and(imieJan);
8  ObjectSet result = query.execute();
```

Pierwsze cztery linijki odpowiadają zapytaniu z Przykładu 1. W linii piątej przechodzimy do pola `placa`, żeby w linii szóstej nałożyć na nie ograniczenie. Interesują nas pracownicy z pensją większą niż 1000. W tym celu stosujemy metodę `greater()`. W linii siódmej łączymy oba utworzone ograniczenia za pomocą metody `and()`.

Diagram 2 przedstawia graf powyższego zapytania.

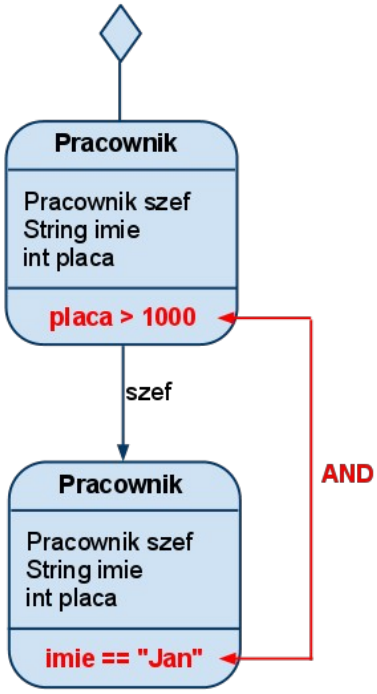

Diagram 2

Oczywiście przy odrobinie wprawy analizując kod zapytania z przykładu drugiego można szybko odczytać jakie zapytanie on realizuje. Jednak graficzna reprezentacja zapytania wydaje się być o wiele bardziej czytelna. Jest to istotne szczególnie dla osób, które dopiero zapoznają się z zapytaniami SODA. Zwłaszcza jeżeli jest to ich pierwszy kontakt z obiektowymi bazami danych. Stąd motywacja do stworzenia Graficznego Edytora Zapytań SODA - przedmiotu niniejszej pracy.

2. Cel pracy

Założonym celem pracy było stworzenie narzędzia pozwalającego na wygodny i intuicyjny dostęp do bazy danych db4o. Planowane funkcjonalności:

- odczytywanie bazy danych db4o i wyświetlanie klas przechowywanych w niej obiektów
- tworzenie i edytowanie grafów zapytań
- generowanie kodu zapytania na podstawie grafu
- analizowanie gotowych zapytań poprzez wizualizację zapytania w postaci grafu (generowanie grafu na podstawie kodu zapytania)
- zapisywanie grafów w postaci plików graficznych

3. Przyjęte rozwiązania

Edytor zapytań będący efektem niniejszej pracy powstał jako wtyczka do Zintegrowanego Środowiska Programistycznego Eclipse[2]. Decyzja ta była motywowana dużą popularnością platformy Eclipse, co ułatwia dostęp do Edytora potencjalnym zainteresowanym. Rozszerzanie istniejącej, stabilnej platformy daje również komfort korzystania ze sprawdzonej bazy kodu, który w innym przypadku autor zmuszony byłby napisać, a który nie jest istotny dla tematu niniejszej pracy.

Platforma Eclipse oferuje framework przeznaczony do tworzenia graficznych edytorów o nazwie Graphical Editing Framework (GEF)[3]. W niniejszej pracy jest on intensywnie wykorzystywany. Użycie biblioteki GEF pozwoliło autorowi skupić się na centralnym problemie niniejszej pracy – wizualnym tworzeniu zapytań typu SODA.

TODO dlaczego SODA

4. Organizacja pracy

Mając za sobą wprowadzenie w kolejnych rozdziałach skupimy się na Graficznym Edytorze Zapytań SODA stworzonym w ramach niniejszej pracy.

Rozdział drugi to opis stworzonego oprogramowania. Zawiera on omówienie projektu i przedstawienia architektury oprogramowania. Następnie następuje szczegółowy opis funkcjonalności wraz z typowymi przypadkami użycia. Rozdział kończy podsumowanie użytych bibliotek i technologii oraz uzasadnienie ich użycia.

Rozdział trzeci przedstawia rezultaty pracy - co udało się osiągnąć, jakie problemy pozostały nierozwiązane. Następnie omówione zostały wady i zalety stworzonego oprogramowania, propozycje jego rozwinięcia oraz krótkie porównanie z innymi dostępnymi na rynku produktami o podobnych funkcjonalnościach. Następnie zaprezentowane są ciekawe problemy implementacyjne napotkane podczas prac nad Edytorem.

Do pracy dołączone są następujące załączniki:

- Załącznik A – Podręcznik Użytkownika
- Załącznik B – Kod źródłowy

2 Eclipse IDE - http://eclipse.org/
3 Graphical Editing Framework - http://www.eclipse.org/gef/

Rozdział II
Opis stworzonego oprogramowania

1. Graf zapytania

2. Projekt i architektura oprogramowania

1.1. Wymagania

Graficzny Edytor Zapytań powinien pozwalać użytkownikowi stworzyć graf zapytania. Następnie na podstawie tego grafu Edytor powinien wygenerować poprawny syntaktycznie i semantycznie kod zapytania używający SODA API. Jest to ogólnie sformułowany podstawowy cel, który został postawiony przed oprogramowaniem stworzonym w ramach niniejszej pracy. Podczas analizy wymagań zidentyfikowano konkretnie funkcjonalności, które powinno posiadać tworzone oprogramowanie:

- Edytor powinien otwierać dowolny plik bazy danych db4o i wyświetlać obiekty zawarte w tej bazie
- Edytor powinien pozwalać na stworzenie grafu zapytania poprzez dwukrotne kliknięcie na dowolny obiekt z wczytanej bazy danych
- Edytor powinien pozwalać na rozwinięcie dowolnego węzła o ile jest to logicznie możliwe
- Edytor powinien pozwalać na dodawanie ograniczeń do dowolnego węzła grafu
- Edytor powinien pozwalać na ustanawianie logicznych relacji pomiędzy istniejącymi ograniczeniami
- Edytor powinien generować kod zapytania na podstawie grafu
- Edytor powinien zapisywać graf zapytania w postaci pliku graficznego

Według zamierzeń autora głównym polem zastosowań stworzonego narzędzia jest dydaktyka. Oprogramowanie to ma wspomagać proces zapoznawania się z SODA API. Dlatego ważne jest aby było proste w obsłudze a interfejs użytkownika był intuicyjny i przejrzysty.

1.2. Projekt

3. Możliwości i funkcjonalności

Opis możliwości Edytora wraz z typowymi przypadkami użycia.

1.1. Przypadki użycia

4. Użyte narzędzia i technologie

Przedstawienie użytych bibliotek i technologii z uzasadnieniem ich użycia.

1.2. db4o

1.2.1. SODA

1.3. Eclipse Platform

1.3.1. Graphical Editing Framework

Rozdział III
Rezultaty pracy

1. Stan oprogramowania

Przedstawienie stanu Edytora - osiągniętych założonych celów, nierozwiązanych problemów, zalet, wad, ...

1.1. Osiągnięte cele

1.2. Nierozwiązane problemy

1.3. Zalety i wady oprogramowania

1.4. Propozycje rozwinięcia oprogramowania

2. Porównanie z podobnymi narzędziami

3. Ciekawe problemy implementacyjne

Załącznik A
Podręcznik użytkownika

BIBLIOGRAFIA

1. Edlich S., Paterson J. i inni: *The definitive guide to db4o* (2006)
2. Moore B., Dean D. i inni: *Eclipse Development using the Graphical Editing Framework and the Eclipse Modeling Framework.* IBM Redbooks (2004)
3. Neward T.: *The Vietnam of Computer Science* (2006) (http://blogs.tedneward.com/2006/06/26/The+Vietnam+Of+Computer+Science.aspx)