ROZWIĄZANIE PROBLEMU TRWAŁOŚCI DANYCH W KOMPUTEROWYM SYSTEMIE OBLICZEŃ I DORADZTWA DOTYCZĄCYM SUBSTYTUCJI BIOMASĄ KONWENCJONALNYCH ŹRÓDEŁ ENERGII

Michał Wacięga, Krzysztof Molenda
Instytut Inżynierii Rolniczej i Informatyki, Uniwersytet Rolniczy w Krakowie

Streszczenie. W pracy przedstawiono rozwiązanie problemu trwałości danych w implementowanym programie komputerowym BiOBKalkulator wspomagającym obliczenia i doradztwo w zakresie substytucji biomasą konwencjonalnych źródeł energii. Zaprojektowano i zaimplementowano mechanizmy obiektowej reprezentacji danych będących wartościami atomowymi, przedziałami wartości lub listami przedziałów lub wartości atomowych. Zamodelowano hierarchię klas reprezentujących konkretne urządzenia do przetwarzania i spalania biomasy z wykorzystaniem tych mechanizmów i z zapewnieniem trwałości obiektów przy pomocy technologii *Object-Relational Mapping*. Proponowane rozwiązanie pozwala na ujednolicony dostęp do danych o potencjalnie złożonej strukturze oraz na uniezależnienie się modelu obliczeniowego aplikacji od warstwy bazodanowej.

Słowa kluczowe: system doradczy, baza danych, Object-Relational Mapping, JPA

Wstęp

Będąca w fazie implementacji aplikacja BIOBKalkulator jest komputerowym systemem obliczeń i doradztwa, wspomagającym podejmowanie decyzji w zakresie substytucji biomasą niekonwencjonalnych źródeł energii. Udostępnianą przez nią funkcjonalność logicznie pogrupowano w ramach czterech modułów:

– Moduł 1: Określenie zapotrzebowania na energię do ogrzewania danego obiektu (budynku).
– Moduł 2: Określanie nakładów pracy i kosztów produkcji różnych rodzajów biomasy.
– Moduł 3: Określanie nakładów pracy i kosztów przetwarzania różnych rodzajów biomasy w produkt końcowy.
– Moduł 4: Bazy danych urządzeń do przetwarzania oraz spalania biomasy.

Aplikacja budowana jest w technologiach Java EE, z uwzględnieniem klasycznej architektury wielowarstwowej: warstwa prezentacji zrealizowana webowo (*JavaServer Faces* w implementacji *RichFaces*, z uwzględnieniem technik AJAX), warstwa biznesowa z wyróżnieniem warstwy logiki, opartej o POJO (*Plain Old Java Objects*) i warstwy trwałości danych (*persistence layer*) oraz warstwy bazy danych.

Uwarunkowania projektu

Uwarunkowania biznesowo-organizacyjne projektu są stosunkowo skomplikowane. Aplikacja bazuje na pozyskiwanej od ekspertów wysoce specjalistycznej wiedzy i budowanych równolegle modelach obliczeniowych. Zachodzi więc konieczność zastosowania takich technologii i narzędzi, które pozwolą na łatwą – możliwie zautomatyzowaną – propagację zmian w warstwie biznesowej do pozostałych warstw.

Każdy z modułów aplikacji wykorzystuje swoje bazy danych. Pozyskiwane dane (zwłaszcza na potrzeby modułów 2 i 3) nierzadko są w postaci trudnej do automatycznej obróbki (ręczna, ze względu na skalę, nie wchodzi w grę). W szczególności:

Producenci udostępniają różne zestawy danych, charakteryzujących dany produkt – nawet w ramach tej samej klasy urządzeń.

Pozyskane na potrzeby aplikacji dane są w postaci niekoniecznie nadającej się do łatwego odwzorowania w kolumny tabel relacyjnej bazy danych (zakładając użycie innego niż tekstowy typu danych). Instancja pewnej cechy może być opisywana nieatomową (złożoną) porcją danych, np. w postaci:

– wartości złożonej, choć łatwo transformowalnej do pojedynczych kolumn tabeli, np. *Wymiary otworu załadowczego [mm]: 300 x 300*, gdzie możliwe jest zaprojektowanie dwóch kolumn, przechowujących dane typu liczbowego: *długość*, *szerokość*,
– listy wartości (niekoniecznie atomowych), np. *Paliwo: granulat z trocin, groszek węglowy, zboże, drewno,*
– przedziału wartości, np. *Zakres regulacji mocy [kW]: 5-15.*

Sformułowanie zadań

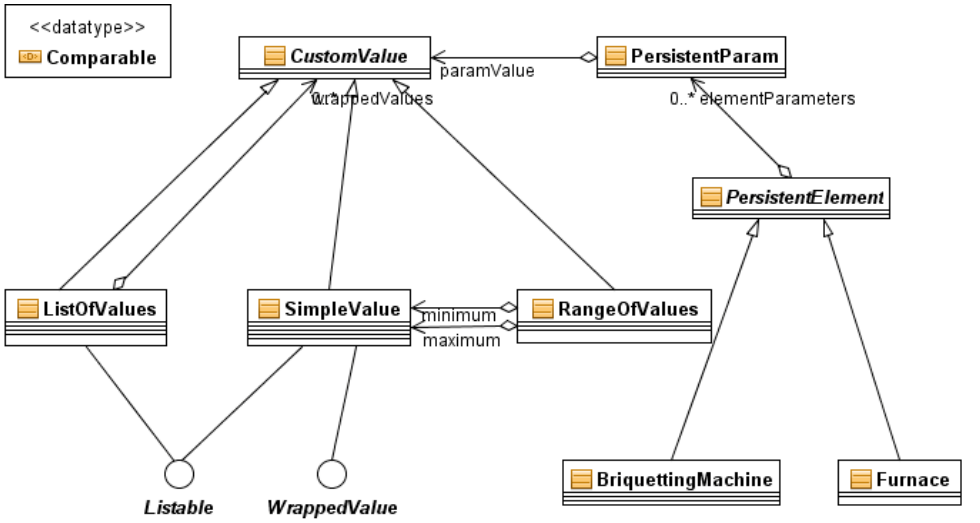
Biorąc pod uwagę zarówno uwarunkowania biznesowe i technologiczne projektu, jak i specyfikę danych, na których operuje sama aplikacja, sformułowano następujące zadania:

– Zaimplementować mechanizmy, pozwalające na wygodne posługiwanie się przez aplikację przechowywanymi w relacyjnej bazie danymi: odwzorować dane ujęte w strukturę relacyjną w obiekty. Należy mieć tu na uwadze, iż każdy byt, opisywany przez takie dane, może mieć nieco inny zestaw parametrów – nawet w ramach podobnych elementów. Dane mogą być wartościami atomowymi (złożonymi), przedziałami wartości lub listami (przedziałów lub wartości atomowych).
– Zaproponować schemat bazy danych dla wspomnianych danych, czy szerzej, zbudować możliwie łatwą w modyfikacji usługę zapewnienia trwałości obiektów.

Realizacja

Wspomniane zadanie zrealizowano w trzech etapach:

1. Zaprojektowano mechanizm reprezentacji złożonych wartości (co wynika ze specyfiki danych).
2. Zaprojektowano klasy, z których obiekty symbolizować będą konkretne urządzenia do przetwarzania czy spalania biomasy.
3. Dodano do klas z pkt. 2 mechanizmy zapewnienia trwałości.



Źródło: opracowanie własne

Rys. 1. Szkic klas reprezentacji trwałych danych urządzeń i ich parametrów, w ramach warstwy biznesowej

Fig. 1. Draft of representation classes for stable data of equipment and its parameters, in the scope of business layer

Istotę rozwiązania (diagram klas) prezentuje rysunek 1. Mechanizm reprezentacji złożonych wartości realizowany jest przez klasy:

Abstrakcyjna klasa `CustomValue`: reprezentuje złożoną wartość: listę, przedział, lub po prostu „opakowaną" dla ujednolicenia pojedynczą wartość. Definiuje pewne wspólne dla wszystkich złożonych wartości operacje: przede wszystkim możliwość poznania największej i najmniejszej wartości, wchodzącej w jej skład (abstrakcyjnie). Na poziomie tej klasy realizowane jest porównywanie w obrębie różnych wartości złożonych – do tego potrzebne są wspomniane metody.

Klasa `SimpleValue`: podstawowa klasa otaczająca pojedynczą wartość (*wrapper* dla standardowego typu danych). Zawiera statyczną metodę fabrykującą, pozwalającą na zwrócenie obiektu `SimpleValue`.

Klasa `RangeOfValues`: reprezentuje przedział, określany tu przez dwie wartości `SimpleValue` (`minimum` i `maximum`).

Klasa `ListOfValues`: dziedziczy po `CustomValue`, będąc jednocześnie w agregacji z nią, dzięki czemu lista zarówno jest złożoną wartością, jak i składa się z innych złożonych wartości.

`Listable` oraz `WrappedValue`: pomocnicze interfejsy.

Interfejs `Comparable`: jeden ze sposobów na zapewnienie porównywania obiektów w języku Java

Klasy, z których obiekty symbolizować będą konkretne urządzenia do przetwarzania czy spalania biomasy, wraz z elementami zapewniającymi usługę trwałości:

Abstrakcyjna klasa `PersistentElement`: stanowi nadklasę klas reprezentujących urządzenia. Dane urządzeń mogą być dwojakiego rodzaju: definiowane jako standardowe pola danych, oraz jako element kolekcji obiektów `PersistentParam`. Klasa zawiera proste mechanizmy, pozwalające na traktowanie pola jako elementu `PersistentParam` nawet, jeżeli jest bardziej tradycyjnego typu.

Klasa `PersistentParam`: obiekt z tej klasy symbolizuje parametr (element charakterystyki) bytu, istotnego z punktu widzenia Modułu 4. Parametr charakteryzowany jest przez nazwę, jednostkę oraz wartość (będącą obiektem `CustomValue`).

Klasy `BriquettingMachine`, `Furnace` itp.: reprezentują konkretne urządzenia. Mogą definiować własne pola, służące jako parametry.

Utrwalanie danych zrealizowano w warstwie trwałości danych, przy pomocy technologii JPA (*Java Persistence API*), będącą realizacją idei ORM (*Object-Relational Mapping*) [Mike 2006]. W uproszczeniu, realizuje ona usługę automatycznego utrwalania obiektów do relacyjnej bazy danych [Barcia i in. 2008], z uwzględnieniem związków pomiędzy ich klasami. Zmiany, których należało dokonać w klasach i samej strukturze projektu, to:

– Dodanie adnotacji `@Entity` do nagłówków klas, których obiekty mają być utrwalane (wszystkie klasy z rysunku 1).
– Dodanie odpowiednich adnotacji (`@OneToOne`, `@OneToMany` itp.) do pól, realizujących związki asocjacji i agregacji pomiędzy klasami-encjami. Związek dziedziczenia realizowany jest poprzez adnotację `@Inheritance`, wraz z wyborem strategii odwzorowania dziedziczenia (tu: *Joined Subclass*).
– Dodanie w każdej wspomnianej klasie pola-klucza (`@Id`), metod `equals()` i `hashCode()` oraz bezparametrowego konstruktora. Klasa encyjna musi też być serializowalna.
– Konfiguracja tzw. jednostki trwałości (*persistence unit*): dodano plik *persistence.xml*, w którym określono podstawowe parametry połączenia ze źródłem danych, konfigurację dostawcy usługi trwałości (*persistence provider*), listę klas zarządzanych przez tę jednostkę trwałości itd.

W obecnej fazie rozwoju baza danych ma schemat wygenerowany automatycznie przez dostawcę JPA (*Toplink Essentials*) na podstawie diagramu klas-encji. Dzięki takiemu podejściu zmiany w warstwie logiki biznesowej wprowadzane jako rezultat prac ekspertów, są łatwo propagowane do warstwy bazy danych.

Dyskusja proponowanego rozwiązania

1. Jest bardzo prawdopodobne, że generowany automatycznie schemat bazy danych jest nieco mniej efektywny od dobrze zaprojektowanego własnego rozwiązania - postawiono tu jednak na szybkie wytworzenie (potencjalnie) mniej efektywnego produktu. Należy podkreślić, że sama aplikacja nie nakłada na bazę danych szczególnych wymagań wydajnościowych: baza danych dla aplikacji nie jest duża (kilkaset – kilka tysięcy rekordów), nie ma potrzeby dostępu transakcyjnego, zaś większość danych jest tylko do odczytu.
2. Prezentowany na rysunku 1 diagram kilku klas uzupełniono oczywiście o klasy zarządzające trwałymi obiektami. W szczególności dotyczy to metod, pozwalających na

przeszukiwanie bazy danych, realizujących specyficzne zapytania JPQL. Zapytania takie są polimorficzne, co umożliwia ujednolicony sposób posługiwania się przechowywanymi obiektami.

3. Kłopotliwą kwestią jest zapewnienie spójności traktowania danych urządzenia, w ramach jego klasy, w postaci związanych obiektów `PersistentParam`, a zwykłych pól danych – część danych konkretnego urządzenia modelowana będzie w sposób standardowy, jako pola danych odpowiednich klas. Chcąc zaznaczyć, że zwykłe pole danych (np. typu `Double`, `String`) powinno być traktowane jako element `PersistentParam`, należy je opatrzyć specjalnie w tym celu skonstruowaną adnotacją `ParamField`. Adnotacja ta przetwarzana jest przez metodę zwracającą listę obiektów `PersistentParam` na zasadzie refleksji (mechanizm *Java Reflection*). Dzięki temu istnieje możliwość dostarczenia „widoku" pola danych w formie obiektu `PersistenParam`. Filozofię przedstawia poniższy fragment kodu:

```
    public class BriquettingMachine
    {

@ParamField(name="Wydajność",unitOfMeasure="kg/h")
             private Double capacity;
            private List<PersistentParam> otherParameters;
            //…
    }
```

Podsumowanie

1. W pracy zaproponowano realizację mechanizmu przechowywania przez obiekt, symbolizujący pewne dane z bazy, danych o potencjalnie złożonej strukturze (pojedyncze wartości, listy, przedziały wartości), wraz z możliwie ujednoliconym sposobem korzystania z tych wartości (zwłaszcza porównań).
2. Dane na których pracuje aplikacja, są reprezentowane jako obiekty trwałe (POJO, ale z funkcjonalnością zapisu i odczytu do relacyjnej bazy danych). Zastosowano tu standardowy ORM technologii Java EE, czyli *Java Persistence API*.
3. Główną motywacją było ułatwienie w dopasowaniu modelu bazy danych do modelu aplikacji, zrealizowane tu poprzez autogenerowanie schematu bazy danych na podstawie utrwalanych obiektów poprzez dostawcę usługi trwałości oraz zapewnienie podstawowej funkcjonalności w dostępie do takiej bazy danych – właśnie za pośrednictwem obiektów klas dziedziny modelu i usługi trwałości JPA.
4. Rozwiązanie to sprawdzi się zarówno w rozproszonej wersji aplikacji BiOBKalkulator (web), jak i w wersji standardowej (offline).

Praca zrealizowana w ramach Projektu Badawczego Zamawianego PBZ-MNiSW-1/3/2006 „Nowoczesne technologie energetycznego wykorzystania biomasy i odpadów biodegradowalnych /BiOB/ – konwersja BiOB do energetycznych paliw gazowych", zadania nr FZP/CPC/36/08 pt. „Komputerowy system obliczeń i doradztwa dotyczący substytucji biomasą konwencjonalnych nośników energii".

Bibliografia

Barcia R., Hambrick G., Brown K., Peterson R., Bhogal K.S. 2008. Persistence in the Enterprise: A Guide to Persistence Technologies. IBM Press. ISBN: 0-13-158756-0.
Mike K., Schincariol, M. 2006. Pro EJB 3: Java Persistence API, APress. ISBN: 1-59059-645-5.

SOLVING THE PROBLEM OF DATA STABILITY IN A COMPUTER SYSTEM FOR COMPUTATIONS AND CONSULTANCY, CONCERNING SUBSTITUTION OF CONVENTIONAL ENERGY SOURCES FOR BIOMASS

Abstract. The paper presents solution for the problem of data stability in an implemented computer application - BiOBKalkulator, which supports computations and consultancy in the field of substituting conventional energy sources for biomass. The research involved design and implementation of mechanisms for object representation of data constituting atomic values, value ranges, or lists of ranges or atomic values. Modelling covered hierarchy of classes representing specific equipment for biomass processing and combustion, and was carried out using these mechanisms and ensuring stability of objects with help of the Object-Relational Mapping technology. The proposed solution allows to ensure standardised access to data possessing potentially complex structure, and to make computational model of the application independent from database layer.

Key words: consulting system, database, Object-Relational Mapping, JPA

Adres do korespondencji:
Michał Wacięga; e-mail; michal.waciega@ur.krakow.pl
Instytut Inżynierii Rolniczej i Informatyki
Uniwersytet Rolniczy w Krakowie
ul. Balicka 116B
30-149 Kraków