DJANGO ORM VERSUS DOCTRINE ORM

FLOREA VICTORIA

Universitatea Tehnică a Moldovei

Abstract: *În articolul dat este efectuată o analiză comparativă succintă a două modele de date obiect-relaţionale (Object-Relational Model): Django şi Doctrine. Este descris modul de serializare utilizat de acestea, mecanismele de protecţie a integrităţii bazei de date pe care acestea le conţin şi sistemele de gestiune a bazei de date pe care le suportă. Accentul a fost pus pe paradigmele pe care acestea le implementează : Active Record şi Data Mapper. Sunt descrise diferenţele dintre aceste două modele, modul cum sunt salvate şi manipulate obiectele, precum şi avantajele şi dezavantajele fiecăruia dintre acestea.*

Cuvinte cheie: *object-relational mapper, active record, data mapper, serializare, persistenţă.*

1. Conceptul de Mapare Obiect Relaţională

Object / Relational Mapping (ORM) permite citirea şi manipularea obiectelor făcând abstracţie de sursa de date de unde provin aceste obiecte. A apărut pentru a consolida diferenţele de paradigmă dintre modelul orientat pe obiecte şi modelul relaţional (utilizat de majoritatea sistemelor de gestiune a bazelor de date). În prezent însă se observă tot mai des tendinţe de consolidare cu SGBD (sisteme de gestiune a bazelor de date) non-relaţionale.

Limbajele de programare orientate pe obiecte reprezintă datele într-un graf interconectat, în timp ce bazele de date relaţionale folosesc un mod tabelar de reprezentare. Necesitatea de a conecta atributele claselor cu câmpurile tabelelor din baza de date nu poate fi ignorată, iar scopul unui ORM este acela de a crea o relaţie naturală, transparentă, fiabilă şi de durată între cele două modele.

Procesul automat de stocare a obiectelor într-o bază de date relaţională folosind un framework ORM, constă în maparea obiectelor la tabelele corespunzătoare, asocierea dintre ele fiind descrisă folosind metadatele. Un framework ORM complet posedă următoarele funcţionalităţi:

- un API pentru operaţiile CRUD (create, read, update, delete) aferente claselor persistente;
- un limbaj pentru specificarea interogărilor adresând clasele persistente şi atributele acestora;
- un mod care să faciliteze definirea metadatelor pentru mapările dintre obiect şi tabel;
- o abordare consistentă a tranzacţiilor, a metodelor de stocare a datelor ("caching") şi a asocierilor dintre clase;
- tehnici de optimizare în funcţie de natura aplicaţiei.

2. Descrierea ORM-urilor examinate

2.1. Django ORM

Django ORM face parte din framework-ul cu acelaşi nume lansat în anul 2005 şi reprezintă cartea de vizită a limbajului Python în lumea tehnologiilor web. Acesta este bazat pe concepte precum evitarea repetabilităţii, şi web-site-uri pilotate de baze de date. Permite utilizarea unei interfeţe simple pentru funcţii de creare, citire, modificare şi ştergere a obiectelor. Django ORM conţine implicit:

- Un mecanism de serializare şi validare datelor din formulare HTML pentru păstrare în baza de date,
- Un sistem de serializare ce poate produce şi citi reprezentări XML şi JSON a instanţelor de modele,
- Un mecanism de protecţie contra atacurilor CSRF şi a injecţiilor SQL.

Astfel, în enumerările de mai sus, serializarea are semnificaţia unei acţiuni de transferare a structurilor de date într-un format ce permite păstrarea, transmiterea sau reconstruirea ulterioară a obiectelor.

CSRF – (cross site request forgery) reprezintă un atac în care un program maliţios transmite comenzi din numele unui utilizator pe care sistemul îl cunoaşte şi în care are încredere. Victime ale acestui atac pot fi atât utilizatorii cât şi sistemul.

Django ORM este compatibil cu următoarele sisteme de gestiune a bazelor de date: PostgreSQL, MySQL, SQLite, Oracle, Microsoft SQL Server. O versiune neoficială a acestuia este compatibilă şi cu MongoDB.

2.2. Doctrine ORM

Proiectul Doctrine este un set de biblioteci PHP al căror scop de bază este oferirea serviciilor de persistenţă şi a funcţionalităţilor asociate. Unul dintre punctele forte ale acestui proiect este utilizarea limbajului DQL, care este foarte similar limbajului SQL cu excepţia utilizării numelor de clase, obiecte şi câmpuri în locul utilizării tabelelor şi coloanelor, ceea ce este un nivel de abstracţie foarte comod pentru construirea interogărilor. O interogare DQL are forma :

$query = $em ->createQuery('SELECT u FROM MyProject\Model\User u WHERE u.age > 20');

Efectuarea acestor interogări se realizează prin intermediul unui Query Builder (constructor de interogări) ce transformă limbajul DQL în SQL. Pentru majoritatea operaţiuniulor nu este însă nevoie de utilizarea DQL, întrucât Doctrine are un set de funcţii ce maschează interogări parametrizate, făcând inutilă cunoaşterea minuţioasă a limbajului SQL. La baza bibliotecii Doctrine se află clasa PDO, un instrument primitiv, dar consistent pentru accesarea bazelor de date. Această relaţie se poate observa în figura 1.

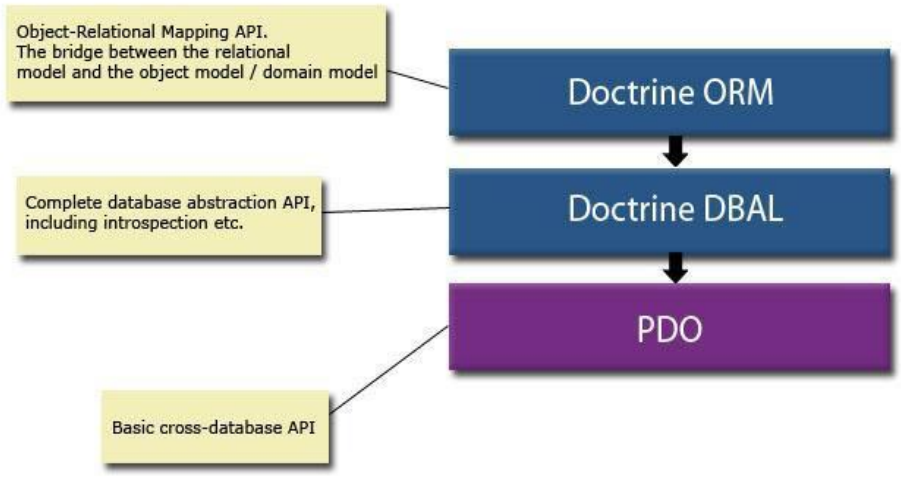

Fig. 1. Structura bibliotecii Doctrine

Scopul pentru acest ORM este activarea într-un mediu agnostic de sistemul de gestiune a bazei de date. Din această cauză, la moment el este compatibil cu multe SGBD relaţionale şi chiar şi câteva SGBD non-relaţionale : mongoDB, CouchDB, OrientDB, PHPCR.

Scopul acestei lucrări nu este compararea acestor ORM-uri din perspectiva limbajelor în care au fost scrise. Accentul este pus pe paradigma implementată, Active Record – de către Django ORM şi, respectiv, Data Mapper – de către Doctrine ORM.

3. Analiza comparativă a pattern-urilor ORM

3.1 Active Record

Acest model se deosebeşte esenţial de Data Mapper în primul rând datorită viziunii asupra obiectelor. Într-un ORM de tip Active Record un obiect se mapează perfect pe un rând dintr-un tabel din baza de date, având aceleaşi câmpuri ca şi coloanele acestuia. Din această cauză un proiect ce utilizează un ORM Active Record este foarte puţin independent de SGBD, însă permite dezvoltarea unui proiect de la zero într-un timp relativ scurt. O altă trăsătură distinctivă este incorporarea funcţiilor de interacţiune cu baza de date în structura obiectelor. De exemplu un obiect poate avea funcţiile delete() şi save(). Modul dat de lucru este destul de comod, dar nu respectă principiul specializării claselor. Acest lucru înseamnă că obiectele acestei clase nu numai păstrează informaţie despre procese din viaţa reală, dar şi interacţionează cu baza de date.

ORM-urile Active Record sunt foarte răspândite găsindu-şi implementare în multe limbaje: ActiveJBDC, JOOQ şi JactiveRecord în Java, Django în Python, Elloquent în PHP, ActiveRecord în Ruby on Rails, clasa DBIx din Perl etc.

3.2 Data Mapper

Data Mapper are o filosofie diferită despre conceptul de obiect. În primul rând, obiectul este o reflectare a business-proceselor companiei şi nu este important ca să se mapeze pe un rând dintr-un tabel. De fapt între un rând din tabel şi obiectul efectiv pot exista zeci de manipulări. La fel, legătura cu baza de date nu se realizează în constructorul clasei. Mai degrabă proprietăţile câmpurilor sunt definite în metadate, sau aşa zisele doc-block-uri. Acestea pot lua forma unor reguli scrise în comentariile clasei şi câmpurilor, într-un fişier XML sau yaml, sau chiar o clasă PHP utilizată pentru documentaţie. Un program intermediar transformă aceste descrieri în interogări SQL de tip Data Definition.

O altă diferenţă importantă este utilizarea unor clase separate pentru efectuarea interogărilor. De exemplu pentru editarea obiectelor se utilizează un Entity Manager (Manager de entităţi). Entităţile reprezintă obiecte cu identitate – fiind practic similare cu obiectele din ORM-urile Active Record. Astfel un Manager de entităţi este utilizat pentru a extrage date, şi a salva modificări asupra lor. Un exemplu al utilizării managerului de entităţi este arătat mai jos:

```
$em->persist($user);
$em->flush();
```

Acesta realizează două operaţii distincte: perisist — care nu cauzează efecte imediate asupra bazei de date, ci doar înregistrează obiectul ca fiind manage-uit şi flush — care lansează spre execuţie interogarea.

Pattern-ul Data Mapper este mai puţin răspândit şi popular decât Active Record. Aceasta se datorează faptului că bibliotecile care îl implementează ocupă mai mult spaţiu şi sunt mai greu de utilizat. Acest lucru le face mai puţin dezirabile în proiectele mici şi dinamice. Exemple notabile de astfel de ORM-uri sunt Doctrine, scrisă în PHP, Hibernate din Java, SQL Alchemy din Python, Nhibernate şi Slazure din C# şi DataMapper din Ruby on Rails.

Concluzii

Putem concluziona că nu există o soluţie universală pentru strategia de data mapping. Nu putem afirma cu siguranţă că un tip de ORM-uri este în toate aspectele mai bun decât altul. Putem, însă, observa că unele ORM-uri se descurcă mai bine cu unele sarcini decât altele. Astfel, un ORM de tip Data Mapper precum Doctrine este foarte util pentru proiectele mari, vechi, care au un set de reguli de business logică bine-definite. Acest model este bun şi atunci când programatorul a început să lucreze la o bază de date imensă, existentă de ceva timp, care este mult mai uşor de conectat cu obiectele dintr-un anumit limbaj de programare prin definirea unor reguli în metadate. Un ORM de tip Active Record ar cere construirea claselor pe baza tabelelor existente, ceea ce ar putea rezulta în clase greoi, care nu descriu corect modelul de afaceri al clientului.

Active Record pe de altă parte, este destinat proiectelor uşoare, aflate în fază incipientă sau care sunt destinate explorării unei pieţe noi. Acestea sunt mai simplu de utilizat şi oferă o imagine mai clară la proiectarea bazei de date.

Aşadar, această lucrare are scopul de a ajuta un specialist cu alegerea unui ORM pentru o situaţie specifică, precum şi pentru a descrie diferenţele dintre cele mai răspândite modele de ORM.

Bibliografie

1. Martin Fowler, *Patterns of Enterprise Application Architecture, 2003.*
2. *Data Mapper Pattern* [Resursă electronică]: Regim de acces https://en.wikipedia.org/wiki/Data_mapper_pattern
3. *Active Record Pattern*. [Resursă electronică]:Regim de acces: https://en.wikipedia.org/wiki/Active_record_pattern
4. *Diferenţa dintre Active Record şi Data Mapper* [Resursă electronică]:Regim de acces: http://culttt.com/2014/06/18/whats-difference-active-record-data-mapper/