Analiza porównawcza technologii odwzorowania obiektowo-relacyjnego dla aplikacji Java

Piotr Błoch, Marek Wojciechowski
Politechnika Poznańska
e–mail: Marek.Wojciechowski@cs.put.poznan.pl

Abstrakt. Zaawansowane technologie do obsługi komunikacji z bazą danych w aplikacjach Java są oparte o koncepcję odwzorowania obiektowo-relacyjnego. Idea ta stanowi również podstawę nowego standardu Java Persistence API, opracowanego wraz z EJB 3.0. Celem artykułu jest porównanie możliwości i wydajności najpopularniejszych technologii odwzorowania obiektowo-relacyjnego: Hibernate i Oracle Toplink oraz standardów JDO i Java Persistence API.

1. Wprowadzenie

Tradycyjne podejście do tworzenia aplikacji operujących na danych składowanych w relacyjnej bazie danych zakłada, że w pierwszej kolejności utworzony będzie relacyjny schemat bazy danych, a następnie implementowana będzie aplikacja z wykorzystaniem programistycznego interfejsu dostępu do bazy danych z poziomu wybranego języka programowania. Dla języka Java takim interfejsem programistycznym jest JDBC – wszystkie aplikacje Java SE i Java EE bezpośrednio lub pośrednio wykorzystują ten interfejs.

Implementacja aplikacji Java pracujących na relacyjnej bazie danych na poziomie interfejsu JDBC jest czasochłonna i uciążliwa. JDBC jest interfejsem stosunkowo niskopoziomowym, wymaga od programistów pamiętania o zwalnianiu zasobów takich jak połączenia z bazą danych, polecenia i zbiory wynikowe, a także przechwytywania i obsługi wyjątków, których przyczyny nie można dociec bez analizy kodów błędów zwróconych przez serwer bazy danych. Odpowiedzią na te mankamenty JDBC było opracowywanie przez firmy na własne potrzeby bibliotek „opakowujących" wywołania JDBC, a także powstawanie gotowych, uniwersalnych rozwiązań tego typu takich jak framework Spring JDBC [Spring].

Podstawowym problemem w tworzeniu w języku Java aplikacji dla relacyjnych baz danych nie są jednak mankamenty JDBC, ale niezgodność modeli danych na poziomie aplikacji i bazy danych. Język Java oparty jest o model obiektowy, pozwalający na modelowanie bardziej złożonych struktur danych i powiązań niż model relacyjny. Reprezentacja danych relacyjnych przez obiekty nie stanowi oczywiście wyzwania, gdyż możliwości modelu obiektowego są nadzbiorem modelu relacyjnego. Problemem jaki napotykają twórcy złożonych aplikacji obiektowych jest znalezienie sposobu zapewnienia trwałości obiektów aplikacji w relacyjnej bazie danych. Do wersji 1.4 włącznie platforma Java Enterprise Edition nie oferowała satysfakcjonującego rozwiązania tego problemu. Funkcję tę miały z założenia pełnić komponenty encyjne w ramach technologii Enterprise JavaBeans (EJB), ale były one nienaturalne, nie wykorzystywały pełni możliwości modelu obiektowego, a do tego w praktyce okazały się nieefektywne.

Jako alternatywę dla encyjnych EJB, różne środowiska zaproponowały technologie automatyzujące odwzorowanie obiektów na poziomie programu Java w struktury relacyjne. Technologie te są określane jako technologie odwzorowania obiektowo-relacyjnego (Object-Relational Mapping – w skrócie ORM). Najpopularniejsze implementacje koncepcji odwzorowania obiektowo-relacyjnego dla aplikacji Java to Hibernate [Hibernate] (rozwiązanie Open Source firmy JBoss) i Oracle Toplink [Toplink] (rozwiązanie firmowe firmy Oracle). Mniejszą popularność zyskał standard JDO (Java Data Objects) [JDO]. Z technologii ORM można korzystać również w celu uzyskania obiektowej reprezentacji danych dla istniejącego schematu relacyjnej bazy danych,

choć w przypadku gdy punktem wyjścia jest model relacyjny atrakcyjną alternatywą są „lżejsze" technologie takie jak ADF Business Components [ADFBC].

Sukces technologii ORM spowodował ewolucję technologii EJB w stronę odwzorowania obiektowo-relacyjnego. W oparciu o najlepsze idee Toplink, Hibernate i JDO, wraz z EJB 3.0, opracowany został nowy standard zapewniania trwałości obiektów w aplikacjach Java, o nazwie Java Persistence API (JPA) [JPA]. Mimo że technologia JPA może być wykorzystana zarówno w aplikacjach Java EE jak i Java SE, ze względów historycznych stanowi ona część specyfikacji Java EE (od wersji 5.0). JPA jedynie standaryzuje interfejs programistyczny (API) do zarządzania trwałością obiektów w aplikacjach języka Java, opierając się głównie o sprawdzone i popularne technologie Hibernate i Oracle Toplink, co daje JPA duże szanse na sukces.

Celem niniejszego artykułu jest porównanie cech, możliwości i wydajności najpopularniejszych technologii odwzorowania obiektowo-relacyjnego: Hibernate i Oracle Toplink oraz standardów JDO i Java Persistence API. W testach efektywnościowych standardy JDO i Java Persistence API będą reprezentowane przez ich najpopularniejsze implementacje.

2. Zestawienie cech i możliwości porównywanych technologii

2.1. Konfiguracja, zastosowanie

Poniższa tabela zawiera porównanie cech systemów ORM związanych z konfiguracją odwzorowania i możliwościami ich zastosowania.

Tabela 1. Zestawienie cech i wymagań związanych z konfiguracją

	Oracle Toplink	**Hibernate**	**JPA**	**JDO**
sposób konfiguracji	XML, Java	XML, adnotacje	XML, adnotacje	XML, adnotacje
generacja schematu relacji na podstawie klas Java/konfiguracji	tak	tak	tak	tak
generacja konfiguracji na podstawie schematu bazy	tak	tak	tak	nie
graficzne środowisko projektowe	tak	tak	tak	nie
dowolny wybór RDBS	tak	tak	tak	tak
inne formaty składowania danych	XML	-	-	XML, OBD, pliki tekstowe
możliwość wykorzystania w aplikacji Java SE	tak	tak	tak	tak
możliwość wykorzystania w aplikacji Java EE	tak	tak	tak	tak

Najbardziej od pozostałych technologii pod względem konfiguracji odbiega JDO: nie generuje schematu relacji na podstawie klas, nie posiada graficznego środowiska projektowego, ale za to w przeciwieństwie do pozostałych technologii nie jest rozwiązaniem zorientowanym wyłącznie na składowanie obiektów w relacyjnej bazie danych. JDO może składować obiekty również w obiektowych bazach danych, a także plikach XML i tekstowych.

Cechą wspólną wszystkich porównywanych technologii jest możliwość ich wykorzystania zarówno w aplikacjach Java Enterprise Edition (Java EE) jak i Java Standard Edition (Java SE).

2.2. Zapytania, operacje SQL

Poniższa tabela zawiera porównanie funkcjonalności technologii ORM w zakresie różnych sposobów wydawania zapytań do bazy danych i możliwości realizacji innych operacji języka SQL.

Tabela 2. Zestawienie sposobów wydawania zapytań i wsparcia dla mechanizmów SQL

	Oracle Toplink	**Hibernate**	**JPA**	**JDO**
zapytania dynamiczne	tak	tak	tak	tak
natywny SQL bazy danych	tak	tak	tak	tak
dedykowany mechanizm zapytań	Expression, obiekty zapytania	HQL, Criteria	JPQL	JDOQL
możliwość wykonywania natywnych operacji DML	tak	tak	tak	nie
ładowanie dynamiczne	tak	tak	tak	tak
odczyt połączeniowy	tak	tak	tak	nie
odczyt wsadowy	tak	tak	nie	nie
bulk update/delete	tak	tak	tak	nie
wsparcie dla procedur i funkcji składowanych	tak	tak	nie	nie

Wszystkie porównywane technologie pozwalają na wykonywanie zapytań dynamicznych i użycie natywnego dialektu SQL wykorzystywanego systemu zarządzania bazą danych, a do tego oferują własny język lub innego rodzaju mechanizm zapytań. Spośród technik optymalizacji dostępu do bazy danych wszystkie technologie wspierają ładowanie dynamiczne, polegające na odczytywaniu powiązanych obiektów dopiero gdy następuje do nich odwołanie. Największy zestaw mechanizmów poprawiających efektywność i najlepszy dostęp do funkcjonalności serwera bazy danych oferują Oracle Toplink i Hibernate. Standard JPA, mimo że jest zorientowany na wykorzystanie relacyjnych baz danych, nie wspiera odczytów wsadowych i procedur składowanych. Jeszcze mniejszą funkcjonalność posiada JDO, niewspierające odczytów połączeniowych (pozwalających zredukować liczbę zapytań przy odczycie powiązanych obiektów) i operacji bulk update/delete (efektywnych operacji UPDATE i DELETE na kolekcjach obiektów, bez konieczności ładowania ich do pamięci).

2.3. Standaryzacja, dostępność

Oracle Toplink i Hibernate to rozwiązania firmowe, niemające statusu standardu. Oracle Toplink jest rozpowszechniany na zasadach komercyjnych, wymaga płatnej licencji. Należy jednak zwrócić uwagę, że licencja Oracle Application Server obejmuje licencję na Oracle Toplink, więc dla użytkowników korzystających z innych rozwiązań i produktów firmy Oracle fakt, że Toplink nie jest darmowy może nie mieć znaczenia. Z kolei Hibernate ma status open source. Przez swoich twórców jest on określany mianem „professional open-source", gdyż w przeciwieństwie do wielu produktów dystrybuowanych na zasadach open-source, za Hibernate stoi firma - JBoss Inc, znana jako producent serwera aplikacji Java EE – JBoss.

JPA i JDO są standardami w formie Java Specification Request (JSR), opracowanymi w efekcie współpracy wielu firm (w tym oczywiście Suna) w ramach Java Community Process (JCP). JPA

jest standardem nowszym niż JDO i stanowiącym część Java Enterprise Edition od wersji 5. Wydaje się, że z chwilą gdy okazało się iż JPA będzie standardem zarówno dla aplikacji Java EE jak i Java SE, JDO straciło na znaczeniu. Świadczy o tym fakt przekazania JDO Apache Software Foundation, gdzie obecnie JDO jest rozwijane w ramach projektu Apache JDO.

Dwie najpopularniejsze implementacje standardu JPA to Hibernate i Toplink Essentials [TopJPA]. Hibernate implementuje JPA poprzez rozszerzenie podstawowego produktu, określanego obecnie jako Hibernate Core, o Hibernate Annotations i Hibernate EntityManager. Toplink Essentials to okrojona wersja Oracle Toplink, ograniczona do podstawowych mechanizmów odwzorowania obiektowo-relacyjnego, specyfikowanych przez JPA. Toplink Essentials ma status referencyjnej implementacji standardu JPA i jest dystrybuowany na zasadach open-source oraz darmowy (podobnie jak implementacja Hibernate). Oracle Toplink pozostaje odrębnym produktem, oferującym poza funkcjonalnością JPA m.in. wsparcie dla specyficznych możliwości serwera bazy danych Oracle.

Standard JDO doczekał się wielu implementacji open-source. Referencyjną implementacją JDO 2 (drugiej edycji standardu) jest JPOX [JPOX].

3. Porównanie efektywności systemów ORM

3.1. Środowisko testowe

Systemem zarządzania relacyjną bazą danych wykorzystanym w testach był Oracle 10g dostępny lokalnie na komputerze, na którym uruchamiane były testowe aplikacje. Dzięki temu rozwiązano problem opóźnień komunikacyjnych na łączach sieciowych. Wykorzystanym językiem programowania była Java w wersji 1.5. Całość testów przeprowadzono na komputerze PC z procesorem INTEL PENTIUM IV 1,7GHZ i 512 MB pamięci operacyjnej. Wykorzystywanym środowiskiem programistycznym był Oracle JDeveloper 10g.

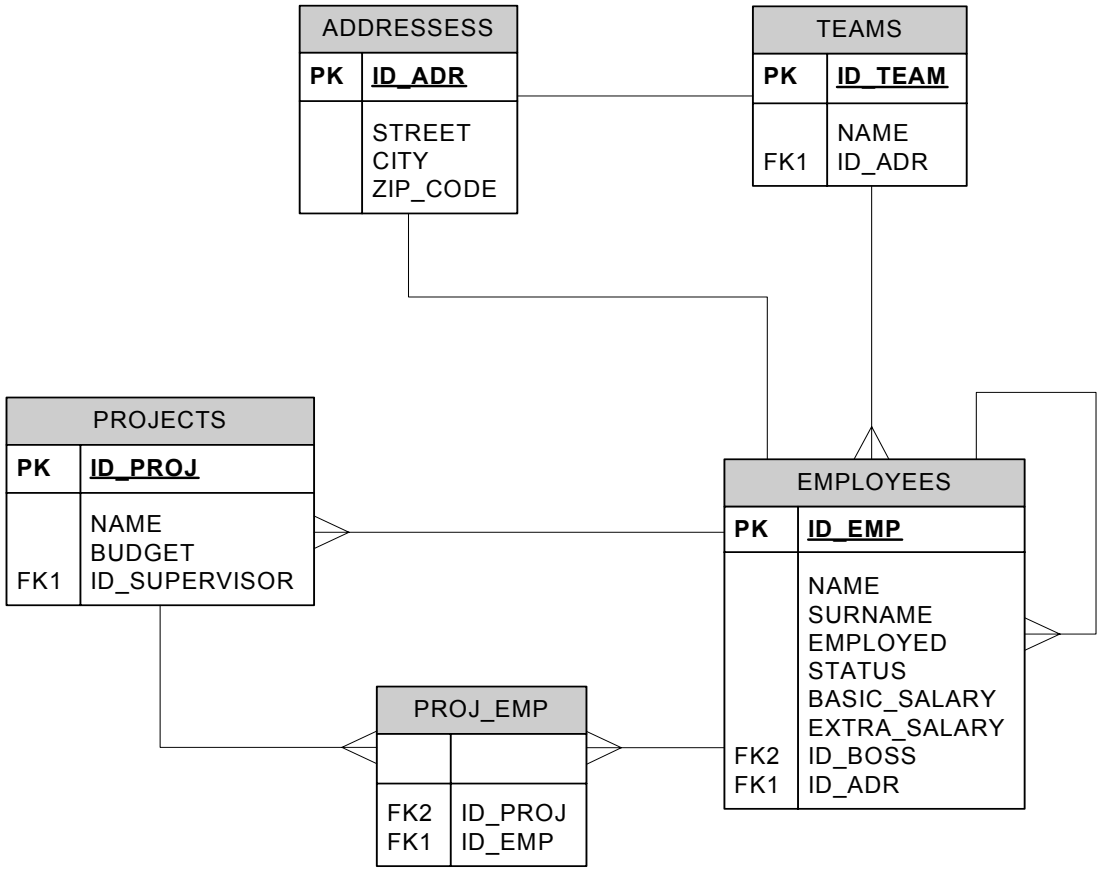
Aby zapewnić porównywalność otrzymywanych wyników (w obrębie różnych systemów) stworzono specjalną platformę testową. Jej zadaniem było uruchamianie zestawów zapytań dla każdego z systemów, mierzenie czasu ich wykonania, oraz odtwarzania stanu bazy danych po każdej serii badań. Każdy pomiar powtarzany był 4 razy, a wynik uśredniany. W przypadku, gdy kolejne wyniki różniły się między sobą o więcej niż 10% badanie było powtarzane (cały cykl).

Po każdej serii testów (dla każdego z systemów) baza danych była przywracana do pierwotnej postaci. Aby wyniki były porównywalne, dla każdego systemu wykonywano w bazie danych te same operacje (tzn. pobierano lub modyfikowano dokładnie te same obiekty).

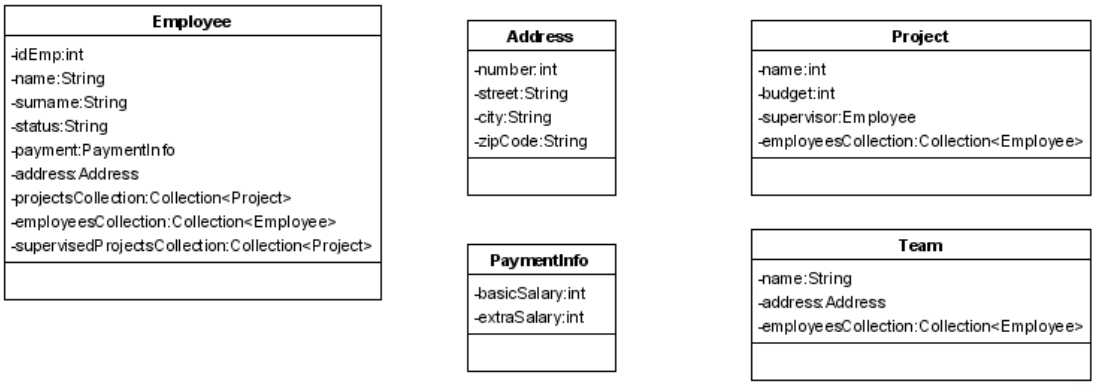
W testach wydajnościowych standard JPA był reprezentowany przez implementacje Hibernate i Toplink Essentials, a standard JDO przez implementację JPOX.

3.2. Testowy model danych

Model danych dla aplikacji testowej został tak zaprojektowany, aby pozwalał na przetestowanie obsługi możliwie różnych związków między obiektami (1-1, 1-N, 1-M, dwu- i jednokierunkowe, związek rekurencyjny). Odpowiadające sobie schemat relacji i model obiektowy zostały przedstawione na poniższych rysunkach:



Rys. 1. Schemat relacyjnej bazy danych



Rys. 2. Obiektowy model danych aplikacji

Przyjęty przykładowy model danych reprezentuje informacje o pracownikach firmy. Z każdym pracownikiem związany jest (oprócz wartości prostych) adres (związkiem 1-1), obiekt opisujący zarobki (związkiem 1-1), lista podwładnych (związkiem 1-N, dwukierunkowym), lista projektów w których bierze udział (związkiem M-N, dwukierunkowym), lista projektów którymi zarządza (związkiem 1-N, dwukierunkowym). Obiekty reprezentujące zarobki i adres są prywatnie zarządzane i wczytywane zawsze, gdy wczytywany jest obiekt nadrzędny (czyli pracownik).

Na potrzeby eksperymentów relacyjna baza danych została wypełniona danymi testowymi. Poniższa tabela pokazuje liczbę wierszy w poszczególnych relacjach schematu.

Tabela 3. Statystyka wierszy w bazie danych

obiekt	liczność
Employee	10020
Project	20
Team	10
Address	10040
PaymentInfo	10020

3.3. Wydajność generowania zapytań i odwzorowania

Celem pierwszej serii testów wydajnościowych było porównanie wszystkich systemów pod względem szybkości generowania zapytań i odwzorowywania. Scenariusz tego etapu wyglądał następująco: dla każdego systemu sekwencyjnie uruchamiano po jednym zapytaniu i mierzono całkowity czas jego wykonania. Poniżej przedstawiono szczegółową listę wykonywanych operacji według kolejności ich wykonywania.

Tabela 4. Wykaz operacji testowych
(kolumna typ: o – odczyt, m – modyfikacja, w – wstawienie, u – usunięcie)

nr	typ	nazwa	opis	cel
1	o	readChilds	pobranie pracowników na etacie kierownika wraz z adresami	zbadanie sposobu wczytywania obiektów będących w związku 1-1
2	o	readAll	pobranie wszystkich pracowników wraz z adresami	odczyt dużego wolumenu danych
3	o	readJoin	pobranie projektów w których uczestniczą pracownicy zespołu drugiego	wykorzystanie połączeń do pobrania obiektów podrzędnych
4	o	readSQL	pobranie pracowników pracujących w więcej niż czterech projektach	sprawdzenie możliwości wykonywania zapytań SQL podanych przez użytkownika
5	o	readCollection	pobranie pracowników zarabiających więcej niż 4000 wraz z kolekcją podwładnych	sprawdzenie wydajności wczytywania kolekcji obiektów podrzędnych
6	m	modifySQL	modyfikacja pensji pracowników z miast na literę P	sprawdzenie możliwości wykonywania instrukcji DML podanych przez użytkownika
7	m	updateNative	modyfikacja pensji pracowników na etacie kierownika	sprawdzenie możliwości modyfikowania obiektów bez potrzeby wczytywania ich do pamięci
8	m	updateJava	modyfikacja pensji podwładnym określonego pracownika	sprawdzenie szybkości generowania instrukcji DML
9	w	insert	dodanie 100 pracowników wraz z adresami	sprawdzenie szybkości dodawania nowych obiektów
10	u	delete	usunięcie pracowników mających więcej niż czterech podwładnych	sprawdzenie szybkości usuwania obiektów

Jako, że wykonywano tylko jedną operację podczas pojedynczego uruchomienia testów nie korzystano z pamięci podręcznej systemu ORM (tzn. dane zebrane przez jedno zapytanie nie były wykorzystywane przez żadne inne). Mierzono więc *de facto* tylko czas generacji zapytania, jego wykonania w bazie danych i odwzorowania z/na postać obiektową. Uzyskano następujące wyniki (czasy w milisekundach):

Tabela 5. Czasy wykonania poszczególnych operacji (symbol „–" oznacza, że danej operacji nie wykonano)

operacja	Toplink	Hibernate	JDO	Toplink JPA	Hibernate JPA
1	766	1656	2766	2453	1375
2	6875	8610	37391	9812	10204
3	578	1547	2343	3188	1406
4	4843	5688	-	7297	6156
5	9266	9797	14890	17907	7031
6	562	671	-	703	641
7	266	922	-	734	703
8	531	1281	782	750	734
9	3610	2453	18953	4937	2953
10	37891	9875	14782	11719	11343

3.4. Wpływ pamięci podręcznej na wydajność

Celem drugiej serii testów wydajnościowych była ocena wydajności mechanizmów pamięci podręcznej oferowanych przez poszczególne systemy ORM. Aby tego dokonać zmodyfikowano nieco przebieg testów. Zamiast pojedynczego uruchamiania zapytań, uruchomiono ich zbiór (pięć pierwszych zapytań) i zmierzono czasy po wykonaniu każdej operacji. Scenariusz ten symuluje działanie zwykłej aplikacji (gdzie uruchamianych jest więcej niż tylko jedno zapytanie). Tabela 6 przedstawia zmierzone czasy wykonania operacji, a Tabela 7 procentowy zysk dzięki wykorzystaniu pamięci podręcznej w stosunku do scenariusza bez pamięci podręcznej.

Tabela 6. Czasy wykonania poszczególnych operacji (w serii)

operacja	Toplink	Hibernate	JDO	Toplink JPA	Hibernate JPA
1	750	1719	2875	2235	1468
2	5984	7906	35657	7890	10610
3	109	250	1344	2485	547
4	344	3563	-	281	343
5	7391	453	10313	10594	610

Tabela 7. Wpływ pamięci podręcznej (sumaryczne czasy dla testowanych operacji)

	Toplink	Hibernate	JDO	Toplink JPA	Hibernate JPA
bez cache	22328	27298	57390	40657	26172
z cache	14578	13891	50189	23485	13578
zysk w %	34,71%	49,11%	12,55%	42,24%	48,12%

3.5. Wnioski

Uzyskane wyniki eksperymentów pozwalają sklasyfikować badane systemy pod względem wydajności działania. Jeśli chodzi o operacje odczytu to bezwzględnie najlepszy okazał się Oracle Toplink. Przedstawione powyżej testy wyraźnie dowodzą, że ma on najszybszy system odwzorowywania danych pochodzących z zapytania na obiekty w aplikacji. Niektóre z operacji (1 i 3) wykonywał 2-3 razy szybciej od pozostałych systemów. Wielką zaletą Toplinka jest możliwość łatwego wpływania na sposób wykonania zapytań, tak by były one najszybsze (np. przez zastosowanie połączeń w miejsce kosztownych podzapytań). Nieco gorsze czasy odczytów uzyskał Hibernate (zarówno samodzielnie i jako implementacja JPA). Również operacje modyfikacji obiektów bez wczytywania do pamięci wykonał nieco wolniej od Toplinka. Zdecydowanie jednak lepiej wypadł w testach sprawdzających szybkość generacji instrukcji DML i SELECT do bazy danych. Szczególnie widoczne było w teście piątym (generowanie wielu podzapytań o adresy pracowników) oraz testach dziewiątym (generowanie zbioru instrukcji INSERT) oraz dziesiątym (kaskadowe usuwanie i modyfikacja). Wyniki tych operacji dowodzą wyższości mechanizmu generowania instrukcji SELECT i DML Hibernate nad innymi systemami.

Ciekawe wnioski płyną także z porównania obu implementacji JPA (Hibernate i Toplink Essentials). Z przeprowadzonych badań wynika, że nieco wydajniejsza jest implementacja Hibernate. Szczególnie dobrze widać to po czasach operacji odczytu. W przypadku modyfikacji, wydajność obu systemów jest porównywalna.

Innym ciekawym spostrzeżeniem wypływającym z testów jest różnica w czasach działania implementacji komercyjnej Toplink i darmowej Toplink Essentials. Po uzyskanych czasach widać, że w obu produktach firmy Oracle zastosowano różne pod względem wydajnościowym rozwiązania.

Najgorsze rezultaty testów uzyskała implementacja JPOX technologii JDO. W każdej z operacji odczytu wykorzystywała ona podzapytania, co miało spory wpływ na obniżenie wydajności. Na uwagę zasługuje także bardzo słabo rozwinięty mechanizm zapytań i brak możliwości wykonywania instrukcji DML podanych przez użytkownika.

Z wyników przeprowadzonego badania wpływu pamięci podręcznej wynika, że dzięki wykorzystaniu mechanizmu pamięci podręcznej wydajność systemów ORM wyraźnie wzrasta. Dzięki niemu nie jest konieczne wykonywanie odczytów obiektów, do których wcześniej dokonywano dostępu. W skrajnym przypadku (zapytanie 3), jeśli na początku wykonano operację wczytania większości/wszystkich obiektów, później można działać na nich już tylko w pamięci (do momentu zapisu). Przeprowadzone badania wskazują, że zdecydowanie najlepszy mechanizm cache posiada Hibernate (także jako implementacja JPA). Zysk z jego użycia to prawie 50%. Za nim uplasował się Toplink. Zdecydowanie najgorzej wypadł w tej konkurencji JPOX (JDO), uzyskując wynik poniżej 15%.

4. Podsumowanie

W artykule porównano najpopularniejsze technologie odwzorowania obiektowo-relacyjnego (ORM) dla aplikacji tworzonych w języku Java. Uwzględniono dwa firmowe rozwiązania: Hibernate oraz Oracle Toplink, a także dwa standardy: starszy JDO (Java Data Objects) i nowszy JPA (Java Persistence API).

Porównanie funkcjonalności technologii ORM obejmowało takie cechy jak: sposób konfiguracji odwzorowania, możliwość wykorzystania w różnych typach aplikacji, sposoby wydawania zapytań, dostęp do funkcjonalności serwera bazy danych poprzez SQL, dostępność i status standardu. Wydajność porównywanych systemów została przetestowana na przykładowym modelu danych dla różnego rodzaju operacji na danych. Testy wydajnościowe zostały przeprowadzone w dwóch seriach. W pierwszej serii operacje były wykonywane w taki sposób aby nie była wykorzystywana

pamięć podręczna ORM. Celem drugiej serii eksperymentów była ocena wpływu mechanizmu pamięci podręcznej dla poszczególnych technologii i implementacji standardów.

Podstawowym wnioskiem dotyczącym funkcjonalności porównywanych rozwiązań jest to, że standardy cechują się ograniczonymi możliwościami współpracy z relacyjnymi bazami danych w porównaniu z rozwiązaniami firmowymi. Szczególnie widoczne jest to w przypadku JDO, ale powodem słabości tego standardu jako technologii ORM są jego założenia, zgodnie z którymi standard ten wykracza poza odwzorowanie obiektowo-relacyjne i w konsekwencji w mniejszym stopniu się na nim koncentruje.

Pod względem wydajności najlepsze rozwiązania to Oracle Toplink i Hibernate, a zdecydowanie najsłabsze to JDO. Porównanie efektywności Toplink i Hibernate jako implementacji JPA wykazało, że o ile w przypadku Hibernate mamy do czynienia z tym samym produktem, to darmowa implementacja Toplink Essentials nie dorównuje wydajnością płatnemu Toplinkowi.

Bibliografia

[ADFBC] Oracle Application Development Framework - Oracle ADF, http://www.oracle.com/technology/products/adf/index.html

[Hibernate] Hibernate, http://www.hibernate.org/

[JDO] Java Data Objects (JDO), http:// java.sun.com/products/jdo/

[JPA] Java Persistence API, http://java.sun.com/javaee/technologies/persistence.jsp

[JPOX] JPOX, http://www.jpox.org/

[Spring] Spring Framework, http://www.springframework.org/

[TopJPA] Toplink JPA, http://www.oracle.com/technology/products/ias/toplink/jpa/index.html

[Toplink] Oracle Toplink, http://www.oracle.com/technology/products/ias/toplink/