Primerjava tehnologij ADO.NET in Hibernate

Marko Štrukelj
Ixtlan Team, d. o. o., Ravbarjeva ulica 13, 1000 Ljubljana
http://www.ixtlan-team.si/
marko.strukelj@ixtlan-team.si (marko.strukelj@nevtron.si)
Marko Bajec
Univerza v Ljubljani, Fakulteta za računalništvo in informatiko, Tržaška 25, 1000 Ljubljana
marko.bajec@fri.uni-lj.si

Povzetek

Hibernate in ADO.NET sodita med najbolj popularne programske rešitve za dostop do relacijskih podatkovnih baz. Hibernate deluje v okolju java. Omogoča objektno-relacijsko preslikavo, kar pomeni, da podatke iz relacijske podatkovne baze preslika v objekte. To mu omogoča tudi poizvedovanje z objektnim poizvedovalnim jezikom HQL ter predpomnenje. Hibernate je neodvisen od podatkovne baze. Tehnologija ADO.NET pa je del ogrodja .NET. V nasprotju z rešitvijo Hibernate ne ponuja objektno relacijske preslikave. Podatkovni model je še vedno relacijski. Kljub temu gre za sodobno rešitev, saj prinaša številne prednosti v primerjavi s starejšimi tehnologijami. V prispevku predstavljamo izbrane rezultate raziskave, v kateri smo primerjali tehnologiji ADO.NET in Hibernate.

Abstract

Comparison of Technologies ADO.NET and Hibernate

Hibernate and ADO.NET are the most popular technologies for accessing relational databases. Hibernate works in java environment. It supports object/relational mapping, which transforms data from relational databases to objects. Hibernate includes interesting features like querying with object-oriented query language HQL, caching and database independence. ADO.NET, on the other hand, is a part of .NET framework and does not support object/relational mapping. The data model is still relational. Nevertheless, it brings contemporary solutions with many new futures comparing to older technologies. In this paper we present some of the results from the research in which we have compared ADO.NET and Hibernate technology.

1 Uvod

Pri razvoju informacijskih sistemov se danes večinoma uporablja objektne programske jezike v kombinaciji z relacijskimi podatkovnimi bazami. Veliko bolj naravna povezava bi seveda bila objektni jezik z objektno podatkovno bazo, vendar se slednje v praksi žal niso uveljavile. Pri uporabi objektnega programskega jezika programer navadno ne uporablja podatkov neposredno iz podatkovne baze, temveč dela z objekti, ki se predhodno kreirajo in napolnijo s podatki iz podatkovne baze. Podobno se objekti preslikajo v relacije oziroma zapise v tabelah relacijske podatkovne baze, ko jih želimo shraniti. Da to lahko storimo, potrebujemo za vsak objekt štiri poizvedbe: za branje, spreminjanje, dodajanje in brisanje. Pisanje takšnih enostavnih poizvedb in pretvorba podatkov v objekte ter nazaj je sicer rutinsko delo, vendar zelo potratno in zato zajema sorazmerno velik del časa razvoja informacijskega sistema. Kljub omenjenim razlikam med predstavitvijo podatkov v objektih in relacijskih tabelah je možna avtomatska pretvorba iz ene oblike v drugo. Takšno pretvorbo imenujemo objektno-relacijska preslikava. Ta avtomatsko generira vse potrebne poizvedbe in objekte v času izvajanja. Programerju se (načeloma) ni več treba ukvarjati s tem, na kakšen način se podatki berejo in shranjujejo.

Objektno-relacijska preslikava je postala popularna, ko je Sun skupaj z J2EE predstavil tehnologijo poslovnih javanskih zrn (Enterprise JavaBeans - EJB). Tehnologija je bila revolucionarna novost, zato so ji napovedovali svetlo prihodnost. Žal pa je bila za praktično uporabo veliko prekompleksna in je, namesto da bi pospeševala razvoj, le-tega zavirala (3). Zaradi kompleksnosti tehnologije entitetnih zrn in njenih slabosti se je pojavilo veliko novih tehnologij, ki pa so obdržale osnovno zamisel - pretvorbo relacijske podatkovne sheme v objektno. Tako je Sun predstavil JDO, Oracle Top Link itd. Pojavilo se je tudi veliko število odprtokodnih rešitev. Med njimi glavno vlogo igra Hibernate. Mesečno ga s spleta prenese kar 15.000 uporabnikov (5). Tako je postal najbolj popularna rešitev za dostop do podatkov v okolju java.

V okolju .NET-a se objektno-relacijska preslikava ni prijela tako dobro kot v javi. Najbolj razširjen način

dostopa do podatkov je tehnologija ADO.NET, ki je bila predstavljena v okviru .NET-a. ADO.NET ne uporablja objektno-relacijske preslikave, zato je podatkovni model še vedno relacijski.

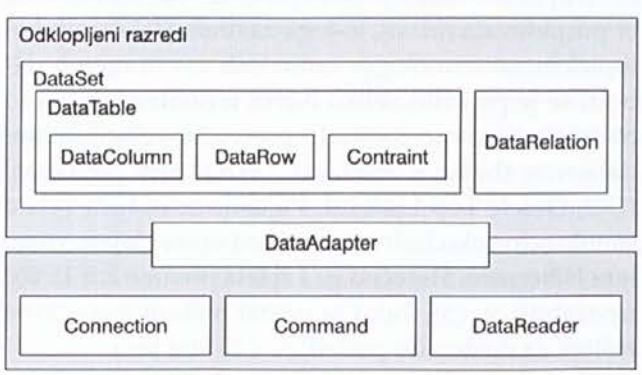
V članku opisujemo rezultate primerjave tehnologij ADO.NET in Hibernate. Najprej sta okvirno predstavljeni obe tehnologiji. Sledi teoretična primerjava po posameznih funkcionalnostih, za tem pa še analiza zmogljivosti. Na koncu podajamo nekaj smernic, ki povedo, kdaj uporabljati eno in kdaj drugo tehnologijo. Prispevek predstavlja povzetek širše raziskovalne naloge.

2 Predstavitev tehnologij

2.1 ADO.NET

ADO.NET je del ogrodja .NET. Ne omogoča preslikave v objekte. Podatkovni model je še vedno relacijski. Njegova pomembna lastnost je tesna integracija z XML zapisom podatkov.

Osnovna arhitektura ADO.NET-a je predstavljena na sliki 1. Razrede ADO.NET lahko v grobem razdelimo v dve skupini. Prva so tako imenovani povezani razredi *(Connected)*, ki skrbijo za dostop do podatkovne baze. Drugo skupino sestavljajo odklopljeni razredi *(Disconnected)*, ki jih uporabljamo za delo s podatki. Vmesnik med njima predstavlja DataAdapter, ki napolni objekte odklopljenih razredov s pomočjo objektov povezanih razredov. DataAdapter v ta namen vsebuje poizvedbe za branje, spominjanje, vstavljanje in brisanje. Logika branja in shranjevanja je tako ločena od podatkov. Poleg tega takšna arhitektura ne zahteva stalno aktivne povezave s podatkovno bazo. Omogočen je odklopljen način dela – povezava z bazo se vzpostavi le v času branja in shranjevanja.

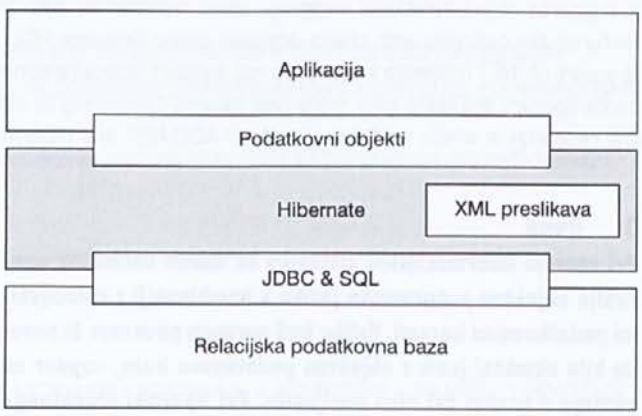


Slika 1: **Arhitektura ADO.NET-a**

ADO.NET v primerjavi s svojim prednikom ADO in ostalimi tehnologijami prejšnjega rodu (JDBC, BDE) omogoča povezave med tabelami. S tem lahko povezani tabeli obravnavamo kot celoto. Če je tabela A povezana s tabelo B, lahko iz zapisa tabele A dostopamo do povezanih zapisov v tabeli B in obratno.

2.2 Hibernate

Hibernate je objektno-relacijski preslikovalni mehanizem za javo, kar pomeni, da podatke, zapisane v relacijski podatkovni bazi pretvori v javanske objekte. To so popolnoma običajni javanski objekti *(POJO - Plain Old Java Objects oz. Plain Ordinary Java Objects)* in zato ne vsebujejo mehanizma, ki bi skrbel za shranjevanje. Ne zavedajo se, na kakšen način se preberejo iz baze ali se vanjo shranijo, podobno kot objekti odklopljenih razredov pri ADO.NET-u. Za to skrbi Hibernate, ki v času izvajanja generira vse potrebne poizvedbe. Cilj Hibernata je, da tako rešimo 95 % dela s podatkovno bazo. Za ostalo še vedno napišemo klasične poizvedbe SQL.



Slika 2: **Arhitektura Hibernata**

Osnovna arhitektura Hibernata je predstavljena na sliki 2. Objektno-relacijska preslikava ustvarja poseben sloj, ki se programerju predstavlja kot objektna podatkovna baza. Ne dela več s tabelami, stolpci, vrsticami, povezavami – elementi, ki sestavljajo relacijsko bazo. Podatki so predstavljeni kot objekti s svojimi metodami, atributi in povezavami z drugimi objekti. Do atributov objekta lahko dostopamo s pomočjo "set" in "get" metod.

Poseben sloj Hibernatu omogoča poizvedovanje s posebnim poizvedovalnim jezikom HQL, ki spominja na SQL, vendar je popolnoma objekten. Da Hibernate

lahko prebere podatke, generira klasične stavke SQL. Do podatkovne baze dostopa s pomočjo javanske tehnologije JDBC. Hibernate lahko prebrane objekte shranjuje v predpomnilnik, kar zelo pospeši delovanje pri pogostem dostopu do istih podatkov.

Poseben nivo objektov in jezik HQL Hibernatu omogočata, da je neodvisen od podatkovne baze. Določimo mu različico SQL jezika, ki naj jo uporablja, ter poskrbimo za ustrezen gonilnik JDBC. Ob menjavi podatkovne baze samo zamenjamo nastavitve za vrsto podatkovne baze in gonilnik JDBC. Takšen način pa deloma omejuje zmogljivost Hibernata, saj lahko podpira samo skupno podmnožico zmogljivosti vseh podatkovnih baz. Do baze pa lahko dostopamo tudi s klasičnimi SQL stavki, kar omogoča izkoriščanje večine zmogljivosti podatkovne baze in optimizacijo poizvedb.

Programer mora za objektno-relacijsko preslikovanje izdelati XML datoteko, ki vsebuje podatke za preslikavo. Iz nje se lahko avtomatično ali ročno izdela javanske razrede. Možna je tudi izdelava XML datoteke iz javanskih razredov, če vanje dopišemo potrebne parametre v zapisu JavaDoc.

Primer zapisa objektno-relacijske preslikave v XML zapisu:

```
<class name="Racun" table="RACUN">
 <id name="id" column="ID">
  <generator class="native"/>
 </id>

 <property name="stRacuna" column="st_racuna"
  type="string" length="30"/>
 <property name="datum" type="date" />
 <many-to-one name="partner" class="Partner"
  column="partnerID" not-null="true"/>
 <set name="postavke" cascade="all" >
  <key column="prej_racID"/>
  <one-to-many class="Postavke"/>
 </set>
</class>
```

Hibernate zna iz preslikave sam tvoriti tabele, stolpce in omejitve v podatkovni bazi. To lahko izvede celo sam ob zagonu programa.

Poizvedbe, napisane v jeziku HQL, Hibernate prevede v SQL poizvedbe. Prevajanje ni vedno optimalno. Po trditvah iz uradne dokumentacije (7) naj bi to povzročilo manj kot 10 % dodatnih JDBC klicev. Zaradi predpomnjenja je število klicev lahko manjše, kot bi jih bilo sicer, pri nepravilnem delu pa tudi bistveno večje.

Hibernate je, tako kot tudi ostale tehnologije za objektno-relacijsko preslikavo, nastal po vzoru poslovnih javanskih zrn. Poslovna javanska zrna sestavljajo trije tipi poslovnih objektov (zrn):

- sejni *(session beans)*, kjer se nahaja poslovna logika,
- entitetni *(entity beans)*, ki predstavljajo podatke,
- sporočilni *(message driven beans)*, za procesiranje *Messaging Service (JMS)* sporočil.

Hibernate nadomešča le entitetna zrna (*entity beans*). Njegova pomembna prednost v primerjavi z entitetnimi zrni je, da ga lahko uporabljamo tudi zunaj programskega strežnika. Ko ga uporabljamo v programskem strežniku, sejna in sporočilna zrna še vedno uporabljajo svojo funkcijo.

3 Primerjava

V nadaljevanju sledi podrobnejša primerjava tehnologij Hibernate in ADO.NET. V primerjavo smo vključili vse kriterije, ki so se nam zdeli pomembni za tovrstno tehnologijo. Ti so:

- preslikava v objekte in tipizacija atributov,
- predpomnjenje,
- odklopljen način dela s podatki,
- poizvedovanje po podatkih,
- generiranje enostavnih poizvedb,
- branje podatkov po potrebi,
- prenos prek omrežja,
- poizvedovanje po podatkih v pomnilniku,
- vezava na komponente uporabniškega vmesnika,
- sočasno spreminjanje podatkov,
- podpora podatkovnih baz,
- čas učenja,
- podpora in dokumentacija,
- licenčne pravice,
- razvoj v prihodnosti.

3.1 Preslikava v objekte in tipizacija atributov

Atributi objekta (oz. v relacijskem modelu stolpci tabele) so lahko glede na način, kako se do njih dostopa, tipizirani ali netipizirani. Do tipiziranih atributov dostopamo prek "set" in "get" metod, kar imenujemo močna tipizacija atributov. Primer (v javi):

```
String nazivPartnerja = partner.getNaziv();
partner.setNaziv(nazivPartnerja);
```

Takšen način znatno olajša programiranje, saj sodobna razvojna orodja s pomočjo dokončevanja kode pomagajo programerju. S tovrstno tipizacijo razvojnemu orodju omogočimo opozarjanje na napake v kodi. Ne more se npr. zgoditi, da bi uporabili atribut/stolpec, ki ne obstaja. Na koncu pa se takšen program tudi ne prevede, kar je bistveno boljše kot pri netipiziranih atributih, kjer napake odkrijemo šele pri izvajanju programa.

Nasprotno do neptipiziranih atributov ne dostopamo s pomočjo metod, temveč prek njihovega imena ali številke. Primer (v C#):

```
string nazivPartnerja = row["Naziv"];
row["Naziv"] = nazivPartnerja;
```

Hibernate podatke preslika v objekte, tako da so atributi tipizirani, kar je njegova prednost. Včasih pa želimo do atributov dostopati s pomočjo niza, ki predstavlja njihovo ime. To je pri Hibernatu oteženo. Ena od rešitev je uporaba "reflekcije". "Reflekcija" je zmožnost programskega jezika, da bere podatke o svoji strukturi, kot so objekti, metode, atributi.

V ADO.NET atributi/stolpci niso tipizirani. Kljub temu razvojno orodje Visual Studio omogoča generiranje razredov, ki imajo močno tipizirane atribute. To mu omogoča podobno funkcionalnost, kot jo ima Hibernate. Takšna rešitev je sicer dobra, a slabša kot pri Hibernatu, ki že v osnovi ponuja takšne razrede brez generiranja kode. Poleg tega pri ADO.NET-u ne moremo izkoriščati dedovanja, polimorfizma ipd. V same objekte pa tudi težko vgradimo dodatno logiko.

3.2 Predpomnjenje

Možnost predpomnjenja podatkov pride do izraza pri pogostem dostopu do istih podatkov. Tak primer so sodobne spletne aplikacije, kjer lahko večje število uporabnikov dostopa do istih podatkov. Predstavljajmo si npr. spletno stran z novicami. Vsi uporabniki dostopajo do novic. Če te niso dostopne v pomnilniku, mora aplikacija za vsakega uporabnika prebrati novice iz podatkovne baze.

Hibernate uporablja vmesni nivo preslikave v objekte, kar mu omogoča predpomnjenje. Sledenje je rešeno popolnoma transparentno s stališča programerja. Programerju ni treba vedeti, od kod se bodo brali podatki. Hibernate uporablja dva tipa predpomnilnika: sejni in drugonivojski. Sejni deluje samo v času Hibernatove seje, kar v praksi običajno predstavlja eno uporabnikovo akcijo ali eno transakcijo podatkovne baze. Ni ga mogoče izključiti. Drugi, uporabnejši, pa je drugonivojski pomnilnik, ki je skupen za celoten Hibernate. Če uporabljamo Hibernatov predpomnilnik, moramo zagotoviti, da le ena instanca Hibernata dostopa do podatkov. V nasprotnem primeru v predpomnilniku nimamo zadnje verzije podatkov.

ADO.NET ne omogoča predpomnjenja. Hibernatovo predpomnjenje predstavlja eno večjih prednosti v primerjavi z ADO.NET-om.

3.3 Odklopljen način dela s podatki

Pri velikem številu dostopov do podatkovne baze je pomembno, da je vsak dostop čim krajši. Dostop do podatkovne baze je potreben le pri branju in shranjevanju. Stalna povezava ni potrebna, kar je bil problem pri starejših tehnologijah. S tem se bistveno povečuje zmogljivost in razširljivost, saj uporablja veliko manj strežniških virov v primerjavi z rešitvijo, ki uporablja trajno aktivno povezavo.

Tako ADO.NET kot Hibernate omogočata polno funkcionalnost brez stalne povezave z bazo. Pravzaprav se pri obeh podatkovni objekti sploh ne zavedajo, na kakšen način se podatki preberejo ali shranijo. Pri ADO.NET-u se za shranjevanje uporablja objekt DataAdapter. Ta vsebuje ukaze SQL, ki skrbijo za branje, shranjevanje, vstavljanje, brisanje. Način zapisa podatkov v bazo pri Hibernatu pa je določen v objektno-relacijski preslikavi v XML zapisu, kjer je navedeno, kateri atributi objektov se preslikajo v katere atribute v relacijski bazi. Pri obeh tehnologijah se povezava odpre le v času branja in pisanja. Za optimalno uporabo povezave skrbi mehanizem izmenjave povezave *(connection pooling)*.

3.4 Poizvedovanje po podatkih

Podatke iz podatkovnega strežnika relacijske podatkovne baze pridobivamo s pomočjo poizvedovalnega jezika SQL. Dobljeni rezultat je tabela podatkov. S pomočjo SQL poizvedb dostopajo do podatkov tudi tehnologije, ki uporabljajo objektno-relacijsko preslikavo, s to razliko, da se poizvedbe generirajo avtomatsko. Objektno-relacijska preslikava pretvori rezultate v objekte. Poseben nivo preslikanih objektov, ki jih ponuja objektno-relacijska preslikava, pa omogoča poizvedovanje na objekten način, torej z objektnim poizvedovalnim jezikom, ki namesto tabele podatkov vrača objekte.

Pri Hibernatu po podatkih poizvedujemo s pomočjo posebnega poizvedovalnega jezika HQL (Hibernate

Query Language), ki je popolnoma objekten in zato omogoča funkcionalnosti, ki jih pri klasičnem jeziku nismo vajeni. Poizvedbe vračajo objekte in ne tabele s podatki, kot to stori klasični SQL. Pisanje poizvedb je bistveno lažje in hitrejše. Pri združevanju (*join*) ni več potrebno navajati atributov oz. ključev, po katerih združujemo, saj Hibernate ve za ključe posameznih objektov in povezave med njimi. Napisane poizvedbe so bistveno krajše in preglednejše. Hibernate v času izvajanja pretvori HQL poizvedbo v eno ali več poizvedb SQL. Poizvedbe HQL so tako neodvisne od podatkovne baze.

Preprosta SQL poizvedba

Branje vseh partnerjev:

```
SELECT * FROM partner
```

Enakovredna HQL poizvedba:

```
from Partner
```

Zahtevnejša poizvedba SQL:

Seznam dobaviteljev, ki nam dobavljajo material granulat:

```
SELECT partner.naziv AS partner_naziv
 FROM prejeti_racun INNER JOIN
    postavka_prej_rac ON prejeti_racun.ID =
      postavka_prej_rac.prej_racID
    INNER JOIN material material
      ON postavka_prej_rac.materialID = material.ID
    INNER JOIN partner ON prejeti_racun.partnerID =
      partner.ID
 WHERE (material.naziv = 'granulat')
```

Enakovredna HQL poizvedba:

```
select dobavnica.partner.naziv
from PrejetiRacun dobavnica
  inner join dobavnica.postavke postavka
where postavka.material.naziv = 'granulat'
```

Poizvedbe lahko gradimo tudi dinamično s pomočjo kriterijskega poizvedovanja:

```
Criteria crit = session.createCriteria(Partner.class);
crit.add(Expression.eq("naziv", "Podjetje d.o.o."));
List partnerji = crit.list();
```

Huda pomanjkljivost HQL-a je, da podpira operacije le v *where* delu poizvedbe. To je velika ovira, saj operacije velikokrat potrebujemo tudi v *select* delu poizvedbe (npr. želimo napisati poizvedbo, ki poleg stolpca a in b vrne tudi njun zmnožek).

Možna je tudi uporaba vgnezdenih poizvedb, vendar le v *where* delu poizvedbe. Če podatkovni strežnik podpira vgnezdene poizvedbe, zna Hibernate to izkoristiti. V nasprotnem primeru pa generira več klasičnih poizvedb. Pogrešamo pa uporabo vgnezdenih poizvedb v *from* delu poizvedbe, saj so te najbolj uporabne in performančno ugodnejše.

HQL jezik je zmogljivejši od večine konkurenčnih tehnologij za objektno-relacijsko preslikavo. Kljub temu moramo zaradi nekaterih pomembnejših omejitev jezika HQL veliko zahtevnejših poizvedb še vedno napisati v SQL-u. V poizvedbi SQL navedemo, v katere atribute na objektih naj se rezultati preslikajo, tako da je delo s podatki še vedno objektno.

Pri ADO.NET-u do baze dostopamo neposredno s pomočjo stavkov SQL, ki so specifični za uporabljeno podatkovno bazo.

3.5 Generiranje enostavnih poizvedb

Pri izdelavi informacijskega sistema je za dostop do podatkovne baze treba napisati veliko poizvedb. Velik del teh je enostavnih, saj gre navadno za branje ali pisanje v eno tabelo. Dobre tehnologije za dostop do podatkov to rutinsko delo olajšajo z generiranjem enostavnejših poizvedb.

Kot smo že omenili, so podatki v Hibernatu predstavljeni kot objekti. Hibernate sam skrbi za polnjenje objektov s podatki iz podatkovne baze in tudi za zapisovanje spremenjenih podatkov nazaj vanjo. To pomeni, da sam generira poizvedbe za branje in shranjevanje. Pri tem zna biti kar se da optimalen. Bere le podatke, ki jih nima v predpomnilniku. Shranjuje le podatke, ki so bili spremenjeni. Omogoča branje podatkov po potrebi, torej šele ko in če jih potrebujemo. Generirati zna tudi poizvedbe za branje podatkov povezanih objektov.

Napreden mehanizem generiranja poizvedb pa ne pomeni, da so generirane poizvedbe vedno optimalne. Programer mora dobro poznati delovanje Hibernata, da vedno uporabi najustreznejši mehanizem. V nasprotnem primeru lahko pride do zelo neoptimalnih poizvedb, kot npr. generiranje poizvedb za branje vsakega zapisa tabele posebej, namesto branja vseh zapisov z eno poizvedbo.

ADO.NET ne pozna tako naprednega mehanizma generiranja poizvedb, kot ga omogoča Hibernate. Poizvedbo moramo napisati sami. Kadar za branje uporabljamo zelo enostavno poizvedbo, lahko s pomočjo objekta razreda CommandBuilder generiramo poizvedbe za

vstavljanje, spreminjanje in brisanje. Generirane poizvedbe so neoptimizirane, zato se njihova uporaba ne priporoča. Več od generiranja poizvedb za osnovno branje in pisanje pa CommandBuilder ne omogoča.

ADO.NET za branje in shranjevanje podatkov vedno uporablja poizvedbe iz objekta DataAdapter. Torej poizvedbe ne optimizira za vsako branje ali shranjevanje.

3.6 Branje podatkov po potrebi

Pri programiranju velikokrat preberemo več podatkov, kot je potrebno, ker ne vemo, katere bomo potrebovali. Rešitev je mehanizem, imenovan branje po potrebi *(lazy loading)*, ki avtomatsko poskrbi za branje podatkov, ko jih potrebujemo.

Takšen način omogoča Hibernate. Če uporabimo ta mehanizem, nam kot rezultat poizvedbe vrne posredniške objekte *(proxy)*. Ti delujejo kot običajni objekti, a ne vsebujejo podatkov. Sami poskrbijo za njihovo branje, ko do njih prvič dostopamo.

Še bolj uporabna pa je možnost branja po potrebi pri branju povezanih objektov. Ko že imamo na voljo objekt, želimo dostopati do objektov, s katerimi je ta povezan. Ti se lahko nalagajo takoj ob branju izhodiščnega objekta ali pa se naložijo po potrebi, ko do objektov dostopamo. Npr. če iz baze preberemo artikel in želimo prek njega dostopati do vseh dobaviteljev, se le-ti, ko jih potrebujemo, avtomatsko preberejo iz baze. To se zgodi brez posebne zahteve programerja. Slednji dela s podatki, kot da bi imel vse na voljo v pomnilniku.

ADO.NET branja po potrebi ne omogoča.

3.7 Prenos prek omrežja

Sodobni informacijski sistemi so z vidika arhitekture sestavljeni večnivojsko. Med posameznimi nivoji je pomembna optimizacija prenosa podatkov. Pomembni so naslednji vidiki:

- **združevanje zahtevanih podatkov:** Če npr. dostopamo do podatkov partnerjev, mora strežnik vrniti vse podatke vseh partnerjev, ki jih odjemalec potrebuje. V nasprotnem primeru bi moral odjemalec zahtevati vsak podatek posebej; npr. naziv, naslov, telefonsko posebej in tako naprej za vsakega partnerja posebej.
- **pošiljanje povezanih podatkov:** Npr. objekt partnerja je lahko povezan s kontaktnimi osebami. Če odjemalec potrebuje kontaktne osebe, mu jih mora strežnik posredovati skupaj s podatki partnerja.
- **vračanje sprememb:** Če popravimo naziv enega od partnerjev, moramo nazaj poslati samo nov naziv, ne pa vseh podatkov tega partnerja, še manj pa podatke o vseh partnerjih. Obstajati mora mehanizem, ki skrbi za sledenje spremembam, ki so bile narejene na podatkovnih objektih.

Opisani problemi se pojavljajo pri uporabi tehnologije entitetnih zrn *(Entity Beans)*. Najustreznejša rešitev je, da podatke zapakiramo v enoten objekt. Načrtovalski vzorec, ki rešuje vse naštete probleme, imenujemo prenosni objekt *(Transfer Object)* (4). Eden od glavnih problemov J2EE je, da ni ustreznega standarda za implementacijo prenosnih objektov. Najbolj resno je zastavljena arhitektura SDO (Service Data Objects), katere predlog sta uskladila IBM in BEA, ki za zdaj v glavnem ostaja le pri dokumentaciji.

Pri uporabi prenosnega objekta odjemalec sproži le eno zahtevo. Strežnik v ta namen kreira prenosni objekt, skopira podatke vanj in ga vrne odjemalcu. Na prejetem objektu odjemalec lokalno dostopa do posameznih atributov. Pri spreminjanju atributov je avtomatično označeno, kateri atributi so bili popravljeni. Ko odjemalec vrne spremenjeni prenosni objekt, se pošljejo samo spremembe. Pri tem se pojavlja problem, da se prenosni objekt ne zaveda sprememb, ki so jih medtem naredili drugi odjemalci. Zato je potrebno slediti verzijam podatkov, kjer je to pomembno.

Pri uporabi Hibernata se lahko izognemo uporabi prenosnih objektov. Hibernate namreč omogoča, da lahko objekte, ki jih preberemo v eni seji/transakciji, uporabimo v drugi. Objekte, ki niso več povezani s sejo, pri Hibernatu imenujejo ločeni objekti *(detached objects)*. Lahko jih posredujemo na druge nivoje in vračamo spremenjene nazaj. Vrnjene objekte nato normalno uporabljamo naprej. Problem združevanja podatkov je tako rešen. Rešen je tudi problem povezanih objektov. Povezani podatki, ki so prebrani iz baze, se posredujejo skupaj z osnovnim objektom. Globino grafa objektov, ki ga posredujemo na drugi nivo, določimo, ko podatke beremo iz podatkovne baze. Edini problem, ki pri tem ostaja nerešen, je, da se vračajo vsi podatki in ne samo spremenjeni.

Precej enostavnejša je rešitev pri ADO.NET-u, saj je podatkovni model relacijski. Podatki se nahajajo v tabelah. Nimamo opraviti s hierarhijo objektov, kar problem močno poenostavi. ADO.NET poleg spremenjenih podatkov hrani tudi originalne. Pri posredovanju čez omrežje se tako lahko enostavno pošljejo samo razlike.

3.8 Poizvedovanje po podatkih v pomnilniku

Pomembna lastnost podatkovne baze je, da lahko po podatkih poizvedujemo. Če nas zanimajo seštevki, želimo podatke urediti, filtrirati ipd., napišemo poizvedbo SQL. To pa ni tako enostavno pri podatkih, ki jih je uporabnik pravkar vnesel in jih še nismo shranili v podatkovno bazo, jih mogoče tudi ne želimo ali pa zanje v bazi ni ustrezne strukture. Želimo si, da bi lahko poizvedovali tudi po teh podatkih.

ADO.NET omogoča enostavno poizvedovanje po podatkih v pomnilniku. Podpira seštevanje in prešteva nje stolpcev tabele, filtriranje in sortiranje. Hibernate tega ne omogoča.

3.9 Vezava na komponente uporabniškega vmesnika

S pomočjo komponent uporabniškega vmesnika, ki omogočajo vezavo na podatke, lahko podatke prikažemo in spreminjamo, ne da bi bilo treba napisati veliko kode.

Skupaj z ogrodjem .NET dobimo komponente uporabniškega vmesnika, ki omogočajo vezavo na podatke ADO.NET-a. Oblikoval se je tudi velik trg tovrstnih komponent. Sestavljanje že narejenih komponent precej skrajša čas razvoja in hkrati omogoča dodajanje funkcionalnosti, ki si jih sicer verjetno ne bi privoščili. Najučinkovitejša je vezava na komponente pri klasičnih aplikacijah. Pri spletnih aplikacijah so zaradi drugačne arhitekture takšne komponente manj zmogljive.

Pri Hibernatu ne obstajajo komponente uporabniškega vmesnika, ki bi omogočale vezavo.

3.10 Sočasno spreminjanje podatkov

Večina informacijskih sistemov je večuporabniških. Problem nastane, ko več uporabnikov spreminja iste podatke. Na takšne dogodke reagiramo z zaklepanjem podatkov, ki je lahko pesimistično ali optimistično.

Pri pesimističnem zaklepanju se podatki, ki jih uporabnik spreminja, zaklenejo in tako postanejo nedostopni drugim uporabnikom. Ta način je za sodobne aplikacije z velikim številom uporabnikov večinoma nesprejemljiv. Pri optimističnem zaklepanju pa domnevamo, da samo en uporabnik spreminja podatke in jih zato ne zaklepamo. V primeru, da drug uporabnik spremeni podatke, imamo dve možnosti. Lahko spremembe drugega uporabnika enostavno shranimo in s tem izbrišemo spremembe prvega. Druga možnost je, da pred shranjevanjem preverimo, ali so se podatki od zadnjega branja spremenili. Če so se, uporabnika obvestimo, da shranjevanje ni uspelo. Če naš problem zahteva tovrstno preverjanje, je treba za to zagotoviti ustrezen mehanizem. Obstajata dve alternativi:

- preverjanje verzije. Vsak zapis v bazi razširimo s stolpcem, ki hrani verzijo zapisa. Verzija je lahko v obliki številke verzije ali časa zadnjega shranjevanja. Pred shranjevanjem preverimo, če se je od zadnjega branja verzija oz. čas spremenil.
- preverjanje sprememb podatkov. Pred shranjevanjem preverimo, če se je spremenil katerikoli podatek v zapisu.

Hibernatov prevzeti mehanizem je optimistično zaklepanje brez preverjanja sprememb. V kolikor potrebujemo preverjanje, nam Hibernate omogoča oba našteta načina. Priporočeno je preverjanje verzije podatkov. Preverjanje sprememb podatkov je možno samo v okviru iste Hibernatove seje, saj slednja hrani stare podatke, ki so potrebni za primerjavo. Oba mehanizma sta s stališča programerja transparentna. Določiti mora samo, kateri tip preverjanja naj se uporablja.

Pri ADO.NET-u praviloma sami napišemo poizvedbe za shranjevanje, zato moramo tudi sami poskrbeti za način zaklepanja podatkov oz. morebitno preverjanje sprememb podatkov. V kolikor za generiranje poizvedb uporabljamo v poglavju 3.5 omenjen CommandBuilder, nam ta generira poizvedbe, ki pred shranjevanjem preverijo spremembe podatkov.

3.11 Podpora podatkovnih baz

Hibernate podpira večino podatkovnih baz, ki so danes v uporabi. Do baze dostopa prek vmesnika JDBC, za kar potrebujemo ustrezen gonilnik. Poleg tega mora Hibernate podpirati različico jezika SQL, ki je specifična za podatkovno bazo. V kolikor različica SQL jezika, ki ga podatkovna baza uporablja, v okviru Hibernata ni podprta, lahko brez večjih težav dodamo podporo zanjo.

ADO.NET je na trgu že veliko časa, zato so tudi zanj na voljo gonilniki za vse pomembnejše podatkovne baze. Poleg tega ADO.NET že v osnovi podpira dostop do baze prek vmesnikov ODBC in OLE DB.

3.12 Čas učenja

Čas, ki ga potrebujemo za usvojitev osnovnega znanja za delo s Hibernatom, bi lahko ocenili kot trikrat daljši od učenja ADO.NET-a. Potrebno je namreč natančno poznavanje njegovega kompleksnega načina delovanja. V

nasprotnem primeru lahko pridemo do neoptimalnih rešitev (poglavje 3.5).

3.13 Podpora in dokumentacija

Uspešnost uvajanja nove tehnologije je precej pogojena z ustreznostjo dokumentacije in kvaliteto podpore.

Za odprtokodne projekte je značilna slaba dokumentacija. Kljub temu lahko rečemo, da je Hibernate dobro dokumentiran. Poleg brezplačne literature je na voljo nekaj komercialnih knjig. Nekomercialna podpora za Hibernate je na voljo prek njihovega foruma. Komercialna podpora pa je možna prek podjetja JBoss, Inc.

Ker je trenutno .NET in z njim ADO.NET ena najbolj popularnih tehnologij, zanj obstaja na kupe zelo dobrih knjig. Uradno komercialno in nekomercialno podporo seveda nudi Microsoft.

3.14 Licenčne pravice

Hibernate je odprtokodni projekt, izdan pod licenco LGPL *(Lesser General Public License)*, ki omogoča brezplačno uporabo tudi v komercialne namene.

ADO.NET je razvil Microsoft, kar pomeni, da njemu pripadajo vse licenčne pravice. To predstavlja delno tveganje, saj se lahko Microsoft kadarkoli odloči, da tehnologije ne bo več razvijal ali pa jo spremeni v tolikšni meri, da naša že napisana koda ne bo več združljiva. Tako se je npr. že zgodilo pri Visual Basicu. Novi Visual Basic.NET je namreč skoraj popolnoma nezdružljiv s starim. Enako velja tudi za ADO in ADO.NET.

3.15 Razvoj v prihodnosti

Odločitev za uporabo določene tehnologije temelji tudi na pričakovanjih o njenem prihodnjem razvoju in spreminjanju. Pred spremembami in vgrajevanjem nove funkcionalnosti ima praviloma prednost združljivost s starimi različicami, saj želimo, da naše rešitve tudi po prehodu na novejšo verzijo nemoteno delujejo.

V času pisanja prispevka je bila na Hibernatovi strani dostopna alfa različica Hibernata 3. Prinaša številne novosti, ki bodo prišle prav predvsem v primerih, kjer smo s sedanjo verzijo naleteli na omejitve. Naj naštejemo nekaj novosti:

- Vse generirane poizvedbe bo moč nadomestiti z ročno napisanimi. Tako bomo imeli možnost, da dostop do baze sami poljubno optimiziramo, kjer je to potrebno. Realizirati bo mogoče funkcionalnosti, ki smo jih do sedaj zelo težko.
- Možno bo uporabljati shranjene procedure.
- Več možnosti bo pri izdelavi preslikave, med drugim preslikava enega razreda v več tabel.
- Možno bo filtriranje podatkov na nivoju seje.
- Na voljo bo zmogljivejši poizvedovalni jezik HQL.
- Integrirane bodo funkcionalnosti, ki jih prinaša java 5.
- Branje podatkov po potrebi bo možno tudi na nivoju stolpcev.

Številne novosti bo prinesel tudi novi ADO.NET 2.0 (1), ki bo izšel leta 2005 z novim .NET ogrodjem, Visual Studijem 2005 in SQL Serverjem 2005. Veliko večje novosti bo prinesla nova verzija operacijskega sistema Winodws, imenovana Longhorn. Ta bo vsebovala tehnologijo za objektno-relacijsko preslikavo, imenovano ObjectSpaces.

4 Analiza zmogljivosti

V tem razdelku podajamo rezultate analize zmogljivosti tehnologij ADO.NET in Hibernate. Namen analize je prikazati, kako konceptualne razlike vplivajo na hitrost. Pri tem nismo želeli ovrednotiti, katera od rešitev je hitrejša. Za to bi bilo treba natančno določiti problemsko domeno – namen uporabe aplikacije, število uporabnikov, strojno podlago itd.

Glavne razlike v hitrosti med ADO.NET in Hibernate izhajajo iz dejstva, da Hibernate omogoča predpomnjenje podatkov. To pomeni, da je prvo branje nekoliko počasnejše, saj je potreben čas za inicializacijo predpomnilnika, drugo branje pa je občutno hitrejše. Pri analizi smo se zato osredotočili samo na branje podatkov. Primerjave hitrosti shranjevanja nismo vključili, ker med tehnologijama ni večjih konceptualnih razlik.

Ne java ne .NET ne omogočata natančnega merjenja časa. Zato smo morali v obeh primerih klicati sistemske klice operacijskega sistema.

Podatki o merjenju

Konfiguracija računalnika, na katerem smo izvajali teste:

Procesor:	AMD Athlon 2,6 GHz
Pomnilnik:	516 MB
Krmilnik diska:	SATA
Disk:	WD 120 GB, 7200 obratov/s.
Operacijski sistem:	Windows XP

Konfiguracija računalnika, na katerem je delovala podatkovna baza:

Procesor:	Intel Pentium II 350 MHz
Pomnilnik:	384 MB
Krmilnik diska:	ATA 66
Disk:	IBM 40 GB, 7200 obratov/s.
Operacijski sistem:	Windows 2000
Podatkovna baza:	Microsoft SQL Server 2000
Nastavitve Hibernata:	
JDBC gonilnik:	JTurbo

Uporabljen sekundarni predpomnilnik: EH
CacheMehanizem za izmenjavo povezave: C3P0
Število maksimalnih sočasnih izmenljivih povezav: 5
Velikost "prepared statement" predpomnilnika: 100

4.1 Branje v odvisnosti od količine podatkov

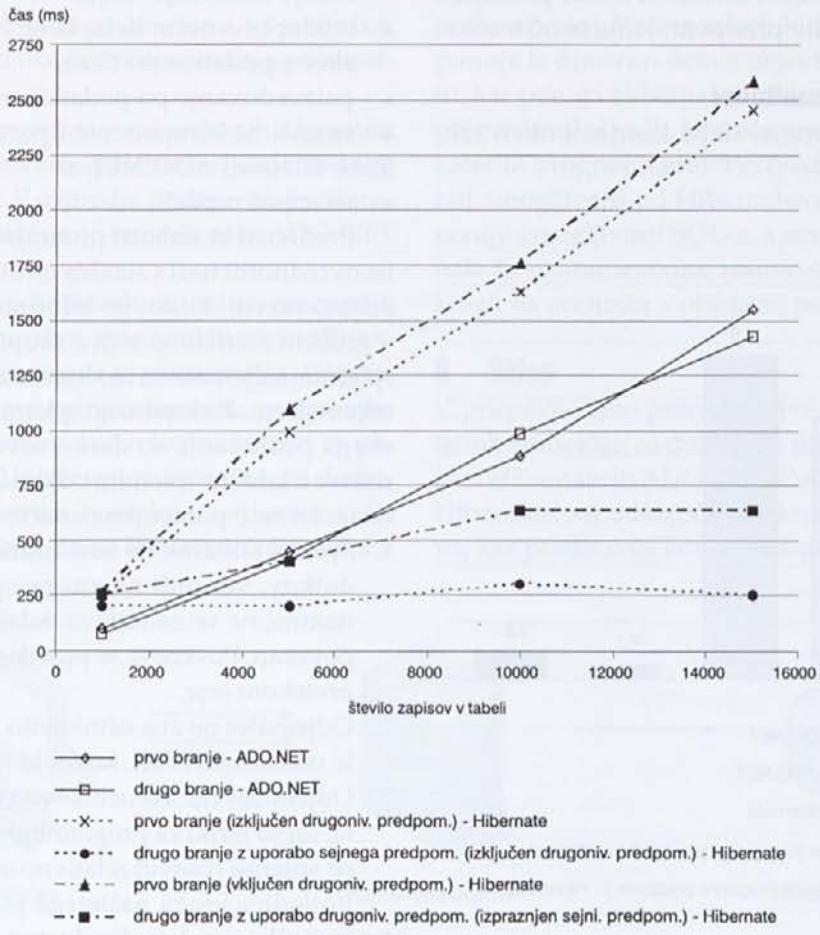
Za merjenje hitrosti branja smo kreirali tabelo, jo postopoma polnili s podatki in jih nato prebrali. Pri tem smo merili čas različnih načinov branja.

Glavna razlika med tehnologijama izhaja iz možnosti predpomnenja, ki ga ponuja Hibernate. Kadar so podatki v predpomnilniku, je branje občutno hitrejše. Kljub temu pa branje v našem primeru ni možno brez dostopa do baze. Hibernate najprej prebere vse ključe tabele, nato pa pogleda, če se posamezen zapis že nahaja v predpomnilniku. Če se, mu celotnega zapisa ni treba prebrati.

Pri načinu drugega branju pri Hibernatu je treba omeniti, da smo klicali drugo metodo kot pri prvem branju. Ta omogoča izkoriščanje podatkov v predpomnilniku, vendar je neoptimalna, če v predpomnilniku ni dovolj velikega deleža podatkov.

Rezultati

Rezultati meritev so prikazani na grafu 1. Čas prvega branja pri Hibernatu je skoraj še enkrat daljši od branja pri ADO.NET-u. Razlog je deloma hkratno shranjevanje podatkov v sejni in drugonivojski predpomnilnik. Drugonivojski predpomnilnik lahko izklopimo in s tem prvo branje nekoliko pospešimo, kar je razvidno iz grafa.



Graf 1: **Čas branja v odvisnosti od količine podatkov**

Drugo branje pri Hibernatu je zaradi uporabe predpomnilnika bistveno hitrejše od branja pri ADO.NET-u. Tu se pokaže prava moč Hibernata. Če bi za testiranje uporabili zahtevnejšo poizvedbo ali če bi bazo bolj obremenili in s tem dosegli daljši odzivni čas, bi bila razlika še veliko večja, pravzaprav poljubno večja.

4.2 Branje zapisa tabele z znanim ključem

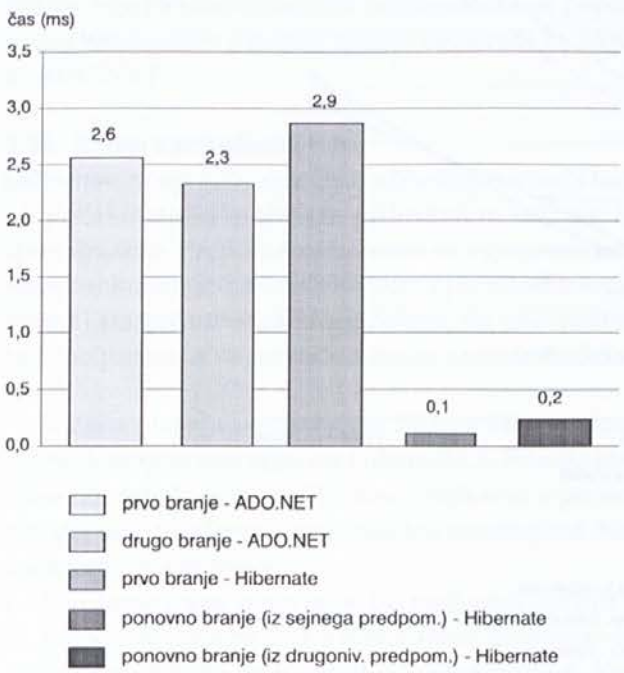
Ponovno smo brali podatke iz tabele, v tem primeru le en zapis. Do njega smo dostopali s pomočjo njegovega ključa. To je najhitrejši način dostopa pri Hibernatu v primeru, da so podatki dostopni v predpomnilniku. Zato v nasprotju s prejšnjo primerjavo pri drugem branju ne potrebujemo dostopa do baze.

Rezultati

Rezultati meritev so prikazani na grafu 2. Čas prvega in drugega branja z ADO.NET-om je po pričakovanju skoraj identičen. Pri Hibernatu je prvo branje daljše in drugo zaradi predpomnilnika občutno krajše. Če bi bazo pri tem močno obremenili in s tem podaljšali odzivni čas, bi bil vpliv predpomnjenja še očitnejši.

5 Interpretacija rezultatov

Strnimo glavne dobre in slabe lastnosti obeh tehnologij.



Graf 2: **Branje zapisa tabele z znanim ključem**

Dobre lastnosti Hibernata:

- pravi objektni model, kar bistveno poenostavlja programiranje in kar je bil tudi vzrok nastanka Hibernata,
- ločenost podatkovnih objektov od mehanizma za branje in shranjevanje,
- odklopljen način dela, ki ne zahteva stalne povezave s podatkovno bazo,
- predpomnjenje,
- avtomatsko generiranje poizvedb za branje in shranjevanje,
- branje po potrebi,
- prenosljivost med podatkovnimi bazami.

Slabe lastnosti Hibernata:

- premalo zmogljiv poizvedovalni jezik HQL,
- onemogočeno ali oteženo izkoriščanje vseh funkcionalnosti podatkovne baze,
- zahteva veliko znanja.

Dobre lastnosti ADO.NET-a:

- ločenost podatkovnih objektov od mehanizma za branje in shranjevanje,
- odklopljen način dela, ki ne zahteva stalne povezave s podatkovno bazo,
- poizvedovanje po podatkih v pomnilniku,
- vezava na komponente uporabniškega vmesnika.

Slabe lastnosti ADO.NET-a:

- relacijski model.

Prednosti in slabosti primerjanih tehnologij je treba ovrednotiti tudi s stališča arhitekture sistema, ki ga želimo razviti. Poslovne informacijske sisteme lahko v grobem razdelimo v dve skupini: sisteme z lahkim spletnim odjemalcem in sisteme z bogatim ali debelim odjemalcem. Z ekspanzijo spletnih rešitev in elektronskega poslovanja so danes seveda zelo popularne rešitve z lahkim spletnim odjemalcem. Takšne rešitve imajo tri zelo pomembne značilnosti (slika 3):

1. Spletni strežnik ne sme hraniti večje količine podatkov, vezanih na enega uporabnika. Strežnik namreč ne ve zanesljivo, kdaj je odjemalec z njim povezan. Povezava se praviloma konča s časovnim pretekom seje.
2. Odjemalec ne zna učinkovito hraniti podatkov, saj le prikazuje HTML kodo, ki jo generira strežnik.
3. Odjemalec ne zna učinkovito manipulirati s podatki, saj so jeziki za programiranje odjemalca (v okviru spletne rešitve) relativno omejeni.

Posledica vsega naštetega je, da skoraj vsak klik uporabnika predstavlja dostop do spletnega strežnika. Ker pa uporabnikovih podatkov praviloma ne

Slika 3: **Generiranje zahtev pri spletnih odjemalcih**

smemo hraniti ne na odjemalcu in ne na spletnem strežniku, moramo dostopati do podatkovne baze. Prihaja torej do pogostega dostopa do podatkovne baze. Sistem predpomnenja je pri tem skoraj nujen. Za takšne rešitve je zelo primeren Hibernate. Na drugi strani pa prednosti, ki smo jih omenili pri tehnologiji ADO.NET, pri spletnih aplikacijah skoraj ne pridejo do izraza. Poizvedovanje po podatkih v pomnilniku je manj pomembno, saj podatke v glavnem beremo iz podatkovne baze ob vsakem dostopu uporabnika. Omejena je tudi vezava na komponente uporabniškega vmesnika.

Pri sistemih drugega tipa, torej sistemih z bogatim ali debelim odjemalcem, predpomnjenje ni tako pomembno (slika 4). V ospredje pridejo vse prednosti ADO.NET-a. Za takšne sisteme je zato ADO.NET primernejši.

Pri tehnoloških odločitvah navadno nastopajo številni kriteriji in omejitve, zato je nehvaležno govoriti o tem, katera tehnologija je primernejša. Objektno-relacijska preslikava prinaša korenite spremembe pri dostopu do podatkov, kar zelo poenostavi in pohitri programiranje. Upoštevati pa moramo, da dodatni nivo preslikave poveča kompleksnost sistema. Če upoštevamo še dejstvo, da se Hibernate uporablja v okolju java, ki velja za kompleksno okolje, lahko ocenimo, da pridejo Hibernatove prednosti do izraza predvsem pri večjih projektih. ADO.NET je bistveno bolj primeren za manjše projekte, kjer bi vpeljava posebnega nivoja objektno-relacijske preslikave ter učenje Hibernata vzela preveč časa.

Hibernate ponuja funkcionalnosti, ki naj bi jih ponudile objektne podatkovne baze. Te se napovedujejo že vrsto let, vendar nič ne kaže, da bi se kmalu uveljavile v praksi. Večina relacijskih podatkovnih baz pa ponuja le osnovno delo z objekti. Ponuditi bi morale to, kar ponuja Hibernate – delo z objekti ter objektni poizvedovalni jezik, ki vrača prave objekte v jeziku, s katerim programiramo. Poizvedovalni jezik bi moral biti zmogljivejši od Hibernatovega HQL-la, torej na nivoju zmogljivosti SQL-a, a seveda objekten. Hibernate in njemu sorodne rešitve mogoče predstavljajo korak na prehodu v objektne podatkovne baze.

6 Sklep

V prispevku smo primerjali dve izmed najbolj popularnih tehnologij za dostop do relacijskih podatkovnih baz: Hibernate in ADO.NET. Posebno mesto zavzema Hibernate, saj omogoča objektno-relacijsko preslikavo, kar predstavlja konceptualno drugačen pristop v

Slika 4: **Generiranje zahtev pri bogatih odjemalcih**

primerjavi s starejšimi tehnologijami. Objektno-relacijska preslikava pretvori podatke iz relacijske podatkovne baze v objekte. Ti se programerju predstavljajo kot objektna podatkovna baza. To znatno olajša programiranje, hkrati pa Hibernatu omogoča dodatne funkcionalnosti, kot so poizvedovanje z objektnim poizvedovalnim jezikom, predpomnenje ter neodvisnost od podatkovne baze. Treba pa je poudariti, da gre za kompleksno tehnologijo, ki od programerja zahteva polno poznavanje njenega delovanja.

ADO.NET je bil narejen skupaj z ogrodjem .NET in razvojnim orodjem Visual Studio.NET. Zato je tesno integriran tako v ogrodje .NET kot v razvojno okolje. Ker je podatkovni model še vedno relacijski, ne moremo izkoriščati vseh prednosti, ki jih ponujajo objektni jeziki. Njegova slabost je tudi nezmožnost predpomnjenja. Kljub temu pa z odklopljenim načinom dela, povezavami med tabelami, avtomatskim generiranjem osnovnih poizvedb in možnostjo generiranje kode z močno tipiziranimi atributi omogoča podobno funkcionalnost kot objektni model. Ponuja tudi številne funkcionalnosti, ki poenostavljajo programiranje.

7 Literatura

1. B. Beauchemin, ADO.NET 2.0 Feature Matrix, julij 2004 http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnvs05/html/ado2featurematrix.asp
2. C. Bauer, G. King, Hibernate in Action, Manning Publications, 2004, ISBN 1-932394-15-X
3. C. Mathews, A Case Against EJB, 10. 10. 2004 http://www.javaranch.com/newsletter/200401/ACaseAgainstEJB.html
4. Core J2EE Patterns - Transfer Object, Sun Microsystems, Inc., 10. 10. 2004 http://java.sun.com/blueprints/corej2eepatterns/Patterns/TransferObject.html
5. G. King, C. Bauer, Object/Relational Persistence for idiomatic Java, 10. 10. 2004 http://www.hibernate.org/hib_docs/online/atljug04_presentation/hibernate_atljug2004.pdf
6. G. King, The Scope of Hibernate Three, 2. 4. 2004 http://blog.hibernate.org/cgi-bin/blosxom.cgi/2004/04/02/
7. Hibernate2 Reference Documentation, Version: 2.1.4 http://www.hibernate.org/hib_docs/reference/en/pdf/hibernate_reference.pdf
8. M. MacDonald, B. Hamilton, ADO.NET in a Nutshell, O'Reilly, 2003, ISBN 0-596-00361-7

Marko Štrukelj je leta 2004 diplomiral na Fakulteti za računalništvo in informatiko. Zaposlen je v podjetju Ixtlan Team, d. o. o. kot razvojni inženir.

Marko Bajec je docent na Fakulteti za računalništvo in informatiko, Univerza v Ljubljani, kjer predava predmete s področja informacijskih sistemov. Raziskovalno in v praksi se ukvarja predvsem s področji, kot so načrtovanje in uvajanje metodologij razvoja informacijskih sistemov (poudarek na uporabi sodobnih pristopov ter najboljših praks), strateško planiranje informatike, poslovno modeliranje, elektronsko poslovanje ipd. Je član več strokovnih in znanstvenih združenj. Svoje prispevke objavlja v domačem in mednarodnem prostoru.