ISTRATECH

CASE konferencija 2020

Postoji li samo jedna "ispravna" arhitektura web poslovnih aplikacija

Zlatko Sirotić, univ.spec.inf.

ISTRA TECH d.o.o., Pula

CASE konferencija 2020

Teme

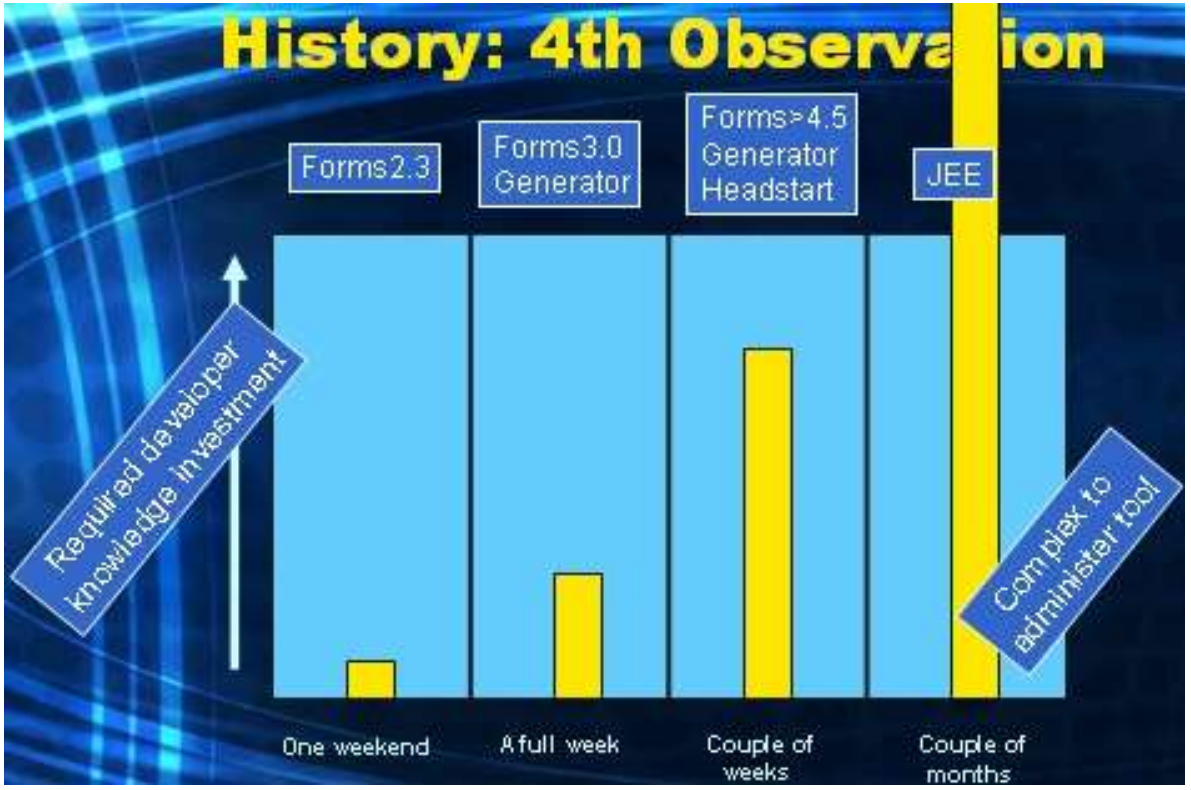
- Je li izrada web poslovnih aplikacija postala previše kompleksna?
- Treba li pisati programski kod za integritet podataka i kod za poslovnu logiku samo u bazi, samo na aplikacijskom serveru, samo na klijentu, ili nekom kombinacijom toga?
- Treba li u bazi uvijek imati samo jednog usera za pristup iz web aplikacije, ili je u redu "stari" način rada: jedna osoba = jedan user na bazi?
- Treba li konekcija između baze i aplikacijskog modula biti stalna ili privremena (na primjeru Oracle ADF-a)?
- Treba li veza između korisničke sesije u web pregledniku i aplikacijskog modula biti stalna ili privremena (ADF)?

Je li izrada web poslovnih aplikacija postala previše kompleksna?

- Mogućnosti DBMS sustava se stalno povećavaju (plavo). No, negdje poslije 2000. godine, te se mogućnosti sve manje koriste (crveno). (Prezentacija Toon Koppelaars-a)

Je li izrada web poslovnih aplikacija postala previše kompleksna?

- Prosječan developer nekada je mogao naučiti alat za izradu poslovnih aplikacija za par tjedana. Danas mu treba i par godina.



Je li izrada web poslovnih aplikacija postala previše kompleksna?

- Različiti pristupi pisanju "tankog" ili "debelog" koda u različitim arhitekturnim slojevima; 1. varijanta je najčešća
- 8. varijanta (sve tanko) ne postoji – ipak treba nešto (i) programirati ☺

Kamo staviti određeni programski kod?

- Može se reći da poslovne aplikacije imaju tri vrste programskog koda:

- **Kod za rad s korisničkim sučeljem**.
- **Kod za poslovnu logiku**. On se može podijeliti na kod za čitanje podataka iz baze i kod za ažuriranje podataka u bazi. Kod za ažuriranje podataka u bazi vrlo često koristi (i) čitanje podataka iz baze.
- **Kod za osiguravanje integriteta podataka u bazi**. Često se ovaj kod brka s kodom za poslovnu logiku, pa nije čudno da se u praksi često isprepliću kod za poslovnu logiku i kod za osiguravanje integriteta podataka.

Kamo staviti određeni programski kod?

- **Programski kod za rad s korisničkim sučeljem, kod web aplikacija može se nalaziti:**

- **Na strani aplikacijskog servera**, što je najčešće. U JEE arhitekturi riječ je o Java servletima (ili nadogradnji servleta, kao štu su JSP, JSF i dr.).
- **Na strani klijenta**. Najčešće se danas takav kod piše u JavaScriptu, a rijetko u Javi kao Java applet (više i ne može).
- **Na strani baze**, što je dosta rijetko. Takvu arhitekturu ima npr. Oracle APEX alat, kod kojeg se HTML stranice dinamički generiraju pomoću APEX-ovih PL/SQL paketa na bazi.

Kamo staviti određeni programski kod?

- **Programski kod za poslovnu logiku, kod web aplikacija može se nalaziti:**

- Na strani aplikacijskog servera, što je vrlo često. U JEE arhitekturi često je riječ o EJB-ovima.
- Na strani baze. Riječ je o tzv. pohranjenim (stored) procedurama / funkcijama i paketima na bazi. Vrlo često se na strani baze sprema programski kod za tzv. batch obradu. Činjenica je da se najčešće najbrže izvršava upravo kod (za poslovnu logiku) koji se nalazi na bazi.
- Na strani klijenta gotovo nikad, čak niti kad se na klijent strani nalaze Java appleti.

Kamo staviti određeni programski kod?

- **Programski kod za osiguravanje integriteta podataka u bazi, kod web aplikacija može se nalaziti:**

- Na strani baze. Najčešće se koriste **deklarativna integritetna ograničenja** baze, kao što su integritetna ograničenja za jedinstveni ključ (UK), vanjski ključ (FK) i check constraint (CK).
  Iako je SQL standard još od 1992. uključio CREATE ASSERTION naredbu, **praktički niti jedan SQL DBMS sustav ju ne podržava** (nedavno su se pojavili ne-SQL relacijski DBMS sustavi koji ju podržavaju).
  Zbog toga, kada se programski kod za osiguravanje integriteta podataka želi u cijelosti pisati na strani baze, mora se **pribjeći korištenju okidača baze** (proceduralni pristup).

Kamo staviti određeni programski kod?

- **Programski kod za osiguravanje integriteta podataka, osim na bazi, može se nalaziti:**

- Na strani aplikacijskog servera. Ovo je vrlo čest pristup u praksi. Želja da se izbjegne (dosta mukotrpna) realizacija integriteta podataka pomoću okidača baze često se navodi kao dovoljan razlog za ovakav izbor. No, time se omogućava da baza bude nezaštićena (u smislu integriteta, ne u smislu sigurnosti podataka općenito). Naime, **jedna aplikacija može savršeno čuvati integritet podataka u bazi, dok, nažalost, druga aplikacija može biti tako pisana da to ne osigurava**.
- Na strani klijenta. Vrlo često se tako rade samo jednostavnije provjere integriteta podataka, koje su istovremeno realizirane i na strani aplikacijskog servera ili/i baze.

Koliko usera imati na bazi?

- U klasičnim klijent-server aplikacijama, obično se radilo tako da je svaki korisnik (osoba) imao svoj vlastiti pristup u bazu, tj. svoj vlastiti korisnički račun (user, shemu) na bazi. Broj korisnika (osoba) obično je bio reda par stotina (ili manje), i svi korisnici su bili poznati (neanonimni).
- Vrlo često web aplikacije rade s puno većim brojem korisnika (osoba), koji su često i anonimni. Tada izgleda logično da se napušta nekadašnji pristup jedan korisnik (osoba) = jedan korisnički račun (user) na bazi. Najčešće se uz aplikacijsku shemu (kojih može biti i više, npr. jedna za tablice, a druga za pakete na bazi) radi samo jedan korisnički račun (user).
- **No, time se neminovno smanjuje sigurnost baze.**

Koliko usera imati na bazi?

- Nije loše koristiti srednji pristup, kod kojeg se odrede različite vrste korisnika (osoba), a onda se za svaku vrstu korisnika napravi poseban korisnički račun (korisnička shema) na bazi. Na taj način, ako netko provali u bazu kroz korisnički račun koji ima manja prava, ne može raditi ono što bi mogao da je provalio kroz korisnički račun sa većim pravima.
- No, ponekad se i kod web aplikacija može primijeniti "stari pristup", ako su svi korisnici poznati (neanonimni) i ako ih nema više od nekoliko stotina. Tada se kao prepreka pojavljuje činjenica da se kod web aplikacija često koristi pool konekcija na bazu (**connection pool**), pa se ne želi da svaki korisnik ima svoj poseban pool. No, tome se može doskočiti primjenom tzv. **proxy korisničkog računa**.

Oracle Forms alat

- Oracle Forms je Rapid Application Development (RAD) alat, koji je Oracle napravio početkom 80-ih, nedugo nakon nastanka Oracle baze verzije 2 (verzija 1 nije nikada postojala), i radio je kao znakovno orijentirana (character mode) aplikacija na serveru.
- Sredinom 90-ih napravljena je klijent-server GUI varijanta. Klijent-server varijanta pratila je baze 6, 7 i 8, a Forms verzije bile su 4, 4.5, 5, 6 i 6i. U Forms verziji 6 pojavila se paralelno i web varijanta - Web Forms.
- Nakon verzije 6i, klijent-server varijanta više ne postoji, tj. verzije od 9i do 12.2 (verzije 7 i 8 nikad nisu postojale) rade isključivo kao web Forms aplikacija (koja nije baš jeftina).

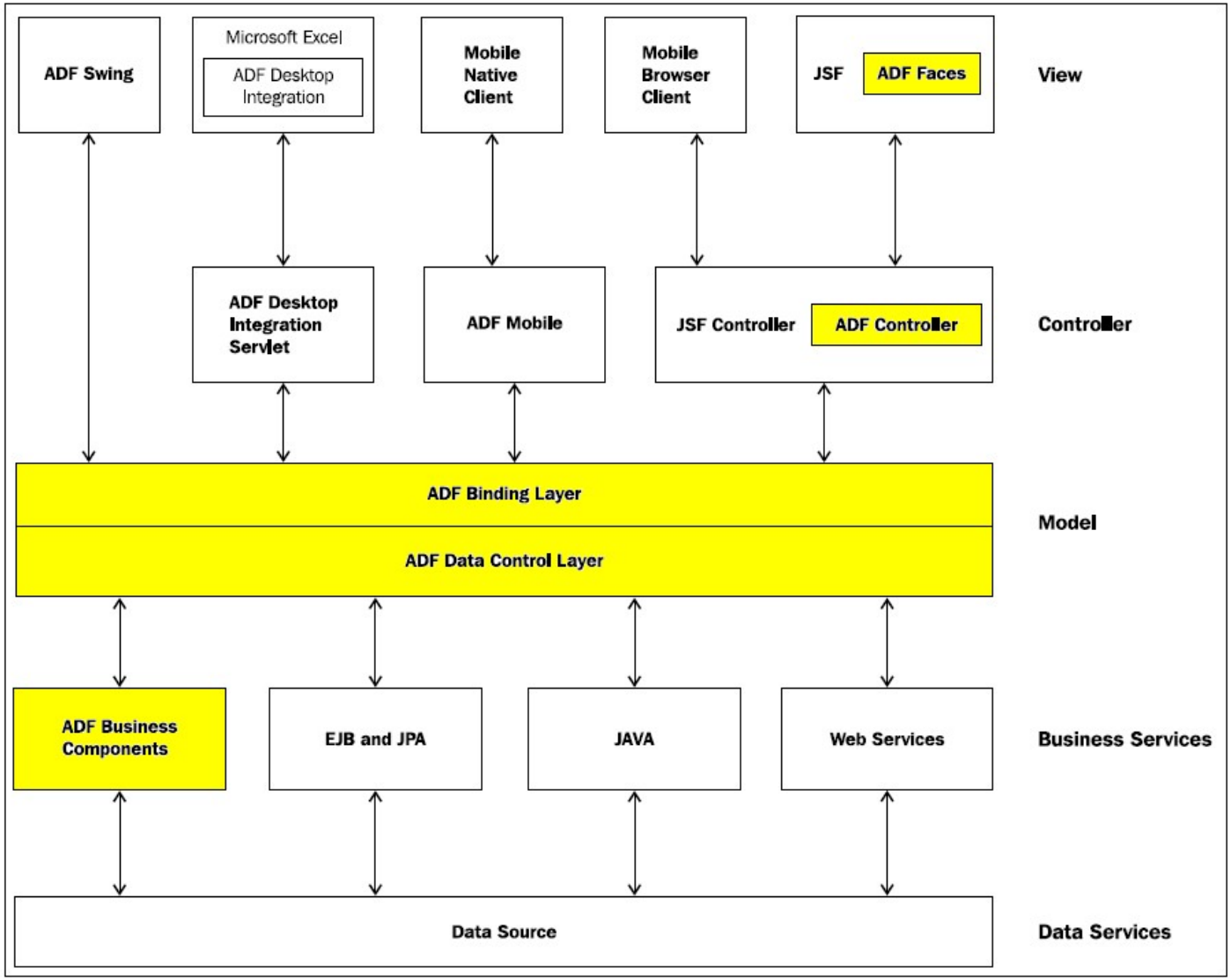
Oracle Forms
- web varijanta

- Web varijanta radi tako da Java applet (ili samostalni Java kod) na klijentu (koji se brine samo za kreiranje korisničkog sučelja) surađuje sa Forms servisom na aplikacijskom serveru.
- Forms servis manje-više čini onaj isti kod (pisan u C-u) kao i u klijent-server varijanti.
- **I u ovoj varijanti:**
  - Forms servis drži stalnu konekciju s bazom;
  - korisnik stalno drži Forms module s kojima radi, sve dok ne završi rad.
- U suštini, ovaj način rada nema nekih značajnih razlika u odnosu na klijent-server rad.

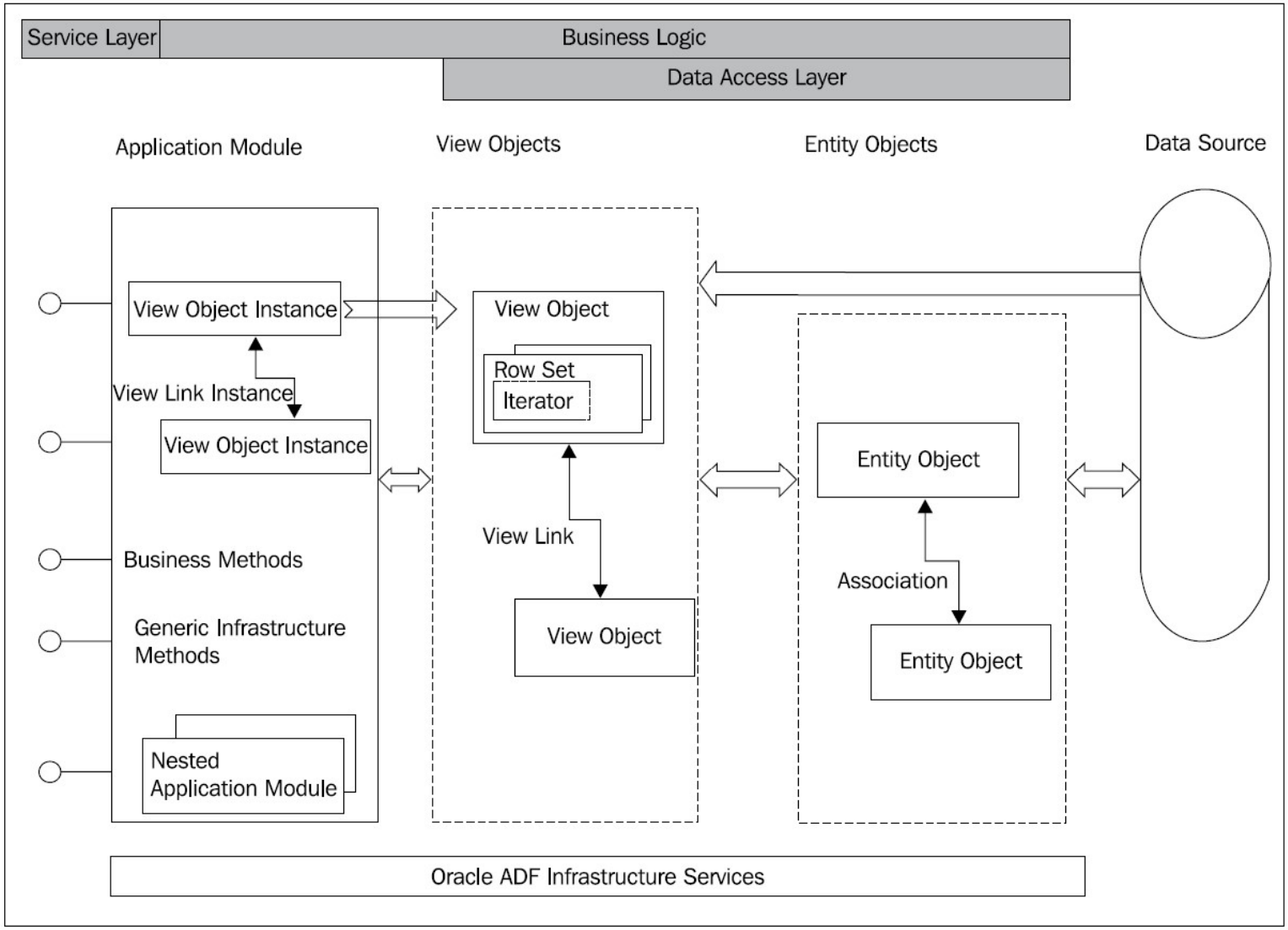
Oracle ADF (Application Development Framework)

- 6 mjeseci nakon što je Sun izdao verziju Jave 1.0, u Oracleu su odlučili raditi RAD alat temeljen na jeziku Java. Prvo izdanje frameworka, koji se tada nije zvao ADF već JBO (Java Business Objects), uslijedilo je 1999. godine. Ubrzo mu je ime promijenjeno u BC4J (Business Components for Java).
- BC4J je pokrivao onaj dio koji danas pokriva ADF BC.
  Uz BC4J, Oracle je počeo razvijati i odgovarajući IDE JDeveloper, licencirajući 1998. tadašnji Borlandov alat JBuilder. Oracle je 2001. temeljito preradio JDeveloper, pri čemu ga je u potpunosti "prepisao" u Java kod.
  Ubrzo je termin BC4J zamijenjen sa ADF.
- Od 2005. godine Oracle JDeveloper IDE je besplatan.
- **Verzija ADF Essentials je besplatna od ljeta 2012.**

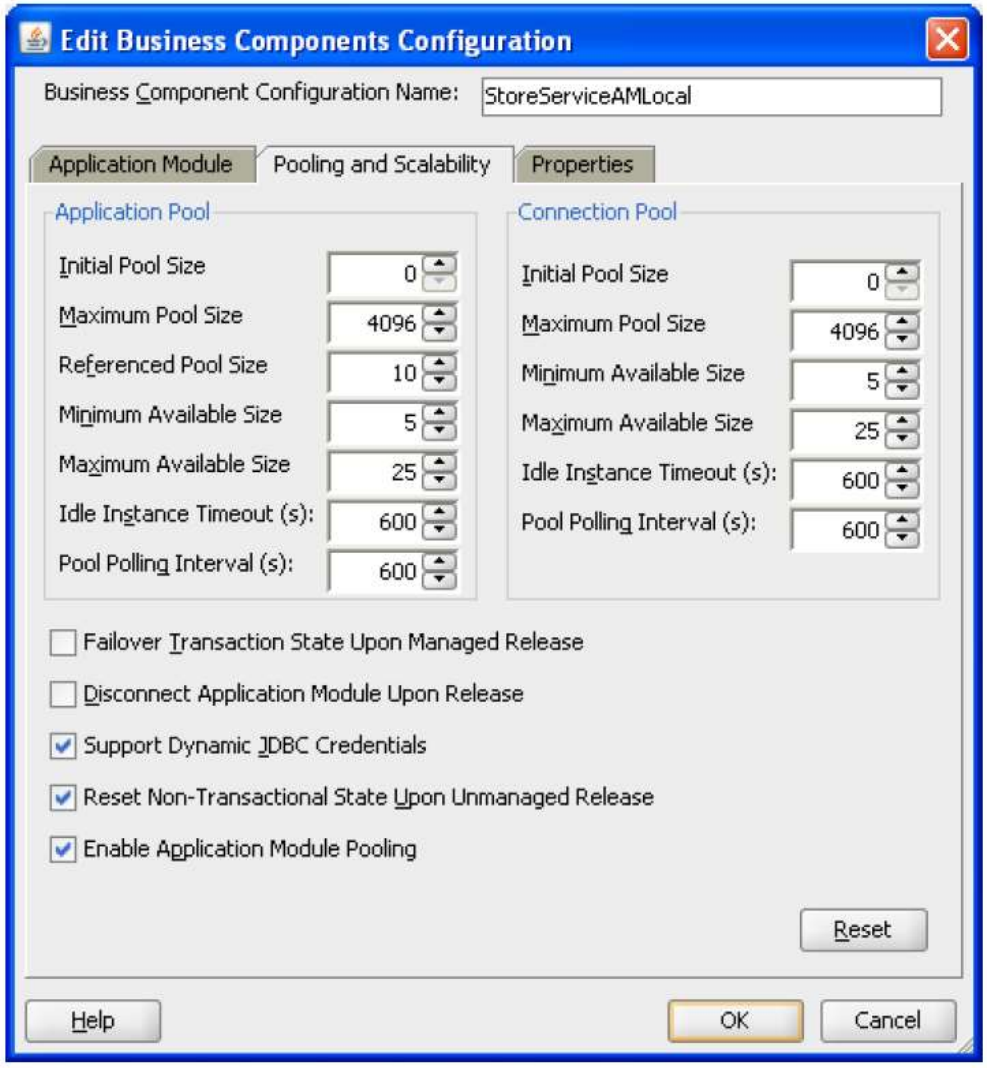
Oracle ADF - struktura



Oracle ADF Application Module (AM)



Application Module pool
i Connection pool (parametri)



Connection pool

- Postoje dvije vrste connection poolova, koje se koriste u ovisnosti o tome konfiguriraju li se konekcije kao **JDBC URL konekcije**, ili **JNDI name for a data source konekcije**. Ako se koriste JDBC URL konekcije, samo tada se koristi ADF connection pool, a inače se koristi connection pool AS-a.
- Osnovno pravilo za ADF connection pool je: po jedan connection pool (dakle, skup konekcija, a ne jedna konekcija) se kreira za svaki par <JDBCURL, Username> na svakom JVM-u, pri čemu konekciju dobiva samo root AM instanca (ugniježđene AM instance koriste tu istu konekciju).
- **Veza između AM instance i konekcije (iz connection poola) je po defaultu stalna**, ali se može postaviti da nije.

Application Module pooling
- stanja AM-ova

- AM pool je kolekcija AM instanci iste vrste. AM pool omogućava da veći broj korisnika može (kvazi) istovremeno raditi na manjem broju AM instanci.
- AM instanca u poolu može biti u jednom od tri stanja:

- **bezuvjetno slobodna** za korištenje bilo kom korisniku;
- **slobodna za korištenje, ali referencirana** na aplikacijsku sesiju koja ju je prethodno koristila i koja još nije završila; u ovom slučaju AM instanca može se ipak predati drugom korisniku, ovisno o tzv. AM State Management Release Levelu, **pri čemu će standardno doći do tzv. pasivizacije (a kasnije aktivacije) AM instance;**
- **zauzeta**, kad neki korisnik (odnosno, njegova Java dretva na AS-u) trenutačno koristi tu AM instancu.

Application Module pooling
- varijante otpuštanja AM-ova

- Kod vraćanja AM instance u AM pool, postoje tri varijante otpuštanja (release levels):

- **Managed** (default): AM pool preferira zadržati istu AM instancu za istog korisnika, ako je to moguće; **ako nije moguće, radi se pasivizacija AM instance (podaci se smještaju u bazu, rjeđe u datoteku)**, a kasnije se radi aktivacija (druge) AM instance;
- **Unmanaged**: nikakvo stanje se ne pamti;
- **Reserved**: veza 1 : 1 između AM instance i korisnika; podsjeća na Forms način rada; **uglavnom nije preporučljiva za web aplikacije, iako je najjednostavnija!**

Oracle Forms i Oracle ADF: primjeri veza klijent - aplikacijski modul - baza podataka

Alat i varijanta	Veza između klijenta i aplikac. modula	Veza između aplikac. modula i baze podataka	Broj potrebnih aplikac. modula i konekcija na bazu za 1000 klijenata
Oracle Forms	stalna	stalna	1000 modula 1000 konekcija na bazu
Oracle ADF varijanta 1	Reserved (stalna)	stalna - default	1000 modula 1000 konekcija na bazu
Oracle ADF varijanta 2	Managed - default (relativno stalna)	stalna - default	npr. 100 do 1000 modula npr. 100 do 1000 konekcija na bazu
Oracle ADF varijanta 3	Managed - default (relativno stalna)	nestalna	npr. 100 do 1000 modula npr. 10 konekcija na bazu
Oracle ADF varijanta 4	Unmanaged (nestalna)	nestalna	npr. 100 modula npr. 10 konekcija na bazu

Zaključak

- Izrada web poslovnih aplikacija postala je jako kompleksna. No ponekad je moguće nešto pojednostaviti.
- Npr. nije isto radimo li web aplikaciju za manji broj poznatih korisnika, ili ogroman broj nepoznatih korisnika.
- Barem ponekad, dobro je pisati programski kod za integritet podataka i kod za poslovnu logiku (uglavnom) u bazi, a ne (uglavnom) na aplikacijskom serveru (što je uobičajeno).
- Zbog (ne)sigurnosti, nije dobro u bazi uvijek imati samo jednog usera za pristup iz web aplikacije. Dobro je uvesti više razina usera, pa čak i "stari" način rada jedna osoba = jedan user na bazi (uz upotrebu proxy usera).
- Nažalost, često developeri gledaju na DBMS sustav kao na "crnu kutiju". Nemaju vremena za dublje upoznavanje s mogućnostima konkretnog DBMS-a, drže da su svi DBMS-ovi vrlo slični, žele pisati generički kod (neovisan o DBMS-u) itd.