Komponentové aplikace a HyperQbs

René Michálek

Qbizm technologies, a.s.
E-mail: rene.michalek@qbizm.com

Abstrakt: Komponentové technologie představují v současné době perspektivní způsob tvorby aplikací. Gartner Group předpokládá, že v roce 2001 bude vývoj nejméně 60 % všech aplikací založen na bázi komponent namísto psaní zdrojového kódu „from scratch“. Přínos komponentových technologií se očekává ve všech fázích softwarového vývoje, počínaje analýzou a designem aplikace konče testováním. Obsahem přednášky bude:

- důvody pro použití komponentových technologií,
- krátké seznámení s komponentovými technologiemi na bázi Javy pro aplikační logiku (EJB) a prezentační logiku internetových aplikací (HyperQbs).

1 Úvod

Na začátku devadesátých let se hodně diskutovalo o výhodách a nevýhodách objektového přístupu při vývoji aplikací. Mnoho odpůrců poukazovalo na větší režii objektových aplikací, horší výsledky při psaní aplikace „z hlavy“ bez důkladné analýzy, atd. Nakonec se ale prokázalo, že objektový princip umožňuje přirozenějším způsobem popisovat chování a vztahy v rámci vyvíjených systémů a nevýhody, které se objektovým technologiím přisuzovaly se postupem času stávaly stále méně aktuální. Vãsoučasné době je k dispozici nepřeberné množství objektově orientovaných technologií, metodik i vývojových nástrojů a snad ani neexistuje programátor, který by neznal alespoň jeden objektově orientovaný programovací jazyk.

Nyní po 10 letech se zdá, že se situace opakuje. Relativně nový princip který je v současnosti diskutován jsou komponentové technologie. Mnoho příznivců objektového přístupu argumentuje, že komponentové technologie postrádají dědičnost (jeden z velmi silných rysů objektů), jsou náročné na výkon, atd. Přesto se předpokládá, že v nejbližší budoucnosti budou komponentové technologie stále více nasazovány.

Již velmi dlouhou dobu je patrná snaha o budování aplikací z připravených „součástek“. Podobně jako když se staví dům z panelů, cihel, oken nebo třeba elektronické zařízení z odporů, kondenzátorů, tranzistorů a integrovaných obvodů. Drtivá většina „komponent“ byla ve zmíněných oblastech zakoupena od dodavatelů (kteří se specializují na jejich vývoj a výrobu) a poté výrobcem efektivně sestavena ve funkční celek. Podobný styl práce může být při použití komponentových technologií aplikován i na software.

Jan Kasprzak, Petr Sojka (editoři): SLT 2001 – sborník semináře o Linuxu a TEXu, str. 123–127, 2001.

Studie zaměřené na tvorbu aplikací zjistily, že 30 % procent času věnovaného na vývoj je stráveno psaním zdrojového kódu. Zbytek času je vynaložen na analýzu, návrh řešení a testování. Jestliže komponenty umožní uspořit nejenom čas nezbytný na psaní zdrojového kódu ale i čas potřebný pro návrh implementace i testování napsaného kódu – potom komponentové technologie umožňují redukovat celkový vývojový proces. Kromě zmíněného znovupoužitelnosti kódu je nutné se ještě zmínit o dalších výhodách spočívajících v možnost škálovatelnosti aplikace a podpoře heterogenní výpočetní infrastruktury.

Komponenty jsou navrženy s ohledem na práci v jistém frameworku. Tzn. na jednu stranu je nutné aby vývojář komponenty zabezpečit jistou nezbytnou funkcionalitu vyžadovanou hostujícím systémem, na straně druhé, není nutné se starat o všechny aspekty, které s sebou psaní velké distribuované aplikace přináší. Je možné se zaměřit pouze na specifický úkol a další služby (transakce, síťový přístup, . . .) zabezpečí příslušný framework.

Softwarová komponenta má svá specifika, které popisuje následující definice.

- Komponenta je navržena tak aby ji bylo možné znovupoužít bez zásahu do zdrojového kódu komponenty nebo nutnosti vytváření potomků tříd, které obsahuje.
- Komponenta může být strukturována různým způsobem. Může být tvořena jedinou třídou, více třídami nebo celou aplikací.
- Komponenty většinou není možné 'spouštět' samostatně. Ke své činnosti potřebují tzv. kontejner. Ten poskytuje prostředí a služby (transakce, pooling, aktivace/deaktivace, přidělování výpočetních zdrojů, . . .) pro jejich běh.

Vývojář, který komponentu používá v aplikaci se nestará o to jak je uvnitř vytvořena ale využívá pouze její funkcionalitu a to nejen v rámci jednoho počítače ale může přistupovat ke službám této komponenty přes síť. Tento koncept samozřejmě vyžaduje větší režii při provozování aplikace, na druhé straně lze velmi efektivně zajistit rozšiřitelnost a škálovatelnost takového systému.

Komponentové technologie jsou většinou součástí tzv. aplikačních serverů, což jsou systémy poskytující vývojářům služby pro budování vícevrstvých aplikací. Mezi služby aplikačních serverů může patřit třeba transaction management, clustering, load balancing, atd.

Výrobci aplikačních serverů založených na Javě se v současné době snaží zajistit aby jejich produkt splňoval nároky specifikace J2EE (Java 2 Enterprise Edition). Před existencí této specifikace každý výrobce Javovských aplikačních serverů implementoval vlastní více či méně komponentový systém, což znamenalo minimální možnost přenositelnosti a znovupoužitelnosti kódu.

Komponentové technologie jsou ale stále v ranném období. Důkazem toho je bouřlivý vývoj samotných technologií i neexistence příslušných metodik komponentového vývoje. Například UML zatím neobsahuje téměř žádné výrazové prostředky určené pro komponentové technologie.

Následující kapitoly se budou věnovat dvěma komponentovým technologiím které jsou založeny na jazyce Java.

2 EJB – komponentová technologie pro tvorbu aplikační logiky

Technologie EJB neboli Enterprise Java Beans je součástí širší specifikace J2EE a poskytuje framework pro procesování komponent aplikační logiky.

Princip funkce EJB komponent je následující. K samotné komponentě, jejíž kód je instanciován a spravován v rámci kontejneru se přistupuje přes dvě rozhraní, které se nazývají „Home interface“ a „Remote interface“. Tato rozhraní jsou v rámci aplikace, která využívá EJB komponenty reprezentovány objekty poskytujícími metody v souladu s metodami, které poskytuje samotná komponenta.

Objekt reprezentující „Home interface“ (factory object) slouží k tomu aby bylo možné manipulovat s instancemi komponenty. Na základě metod, které tento interface poskytuje je možné „požádat“ o vytvoření instance komponenty pro potřeby aplikace, nebo zrušit komponentu, kterou aplikace již nepotřebuje. Samotný akt vytvoření (nebo zrušení) komponenty je pouze zdánlivý, protože o to jestli je potřeba instanciovat další objekt (nebo zrušit) pro danou komponentu nebo stačí použít již existující instanci komponenty se stará samotný kontejner. Pokud se „Home interface“ použije pro vytvoření komponenty, vrátí jako výsledek své činnosti instanci objektu reprezentujícího „Remote interface“.

Objekt „Remote interface“ (proxy object) slouží jako prostředník mezi aplikací a kódem komponenty. Použití určité metody objektu „Remote interface“ způsobí zavolání příslušné metody komponenty v kontejneru. Pokud metoda komponenty vrací nějakou hodnotu jako výsledek své aktivity, je tato hodnota prostřednictvím „Remote interface“ samozřejmě také dopravena do aplikace.

EJB komponenty jsou rozděleny na dvě skupiny: *Session Beans* a *Entity Beans*.

Komponenty patřící do první skupiny (Session Beans) jsou vlastně knihovny funkcí. Tzn. poskytují metody, které zajišťují specifickou funkcionalitu. Session Beans jsou dále rozděleny do kategorií „Stateless“ a „Stateful“. Komponenty spadající do kategorie Stateful Session Beans, jak vyplývá z názvu, si pamatují stav mezi voláním jednotlivých metod. Komponenty typu Stateless Session Beans tuto vlastnost nemají.

Entity Beans představují reprezentaci dat získaných z nějakého datového zdroje (většinou databáze). Význam těchto komponent spočívá v tom, že pokud klientská aplikace zavolá metodu, která mění atributy komponenty, promítne se tato změna zároveň i do příslušných polí databáze. Entity Beans se také dělí na dvě další podkategorie. Komponenta z kategorie nazvané Container Managed Entity Bean se nestará o to jakým způsobem data uložit do databáze a jak je z ní opět získat. To za ní provádí kontejner. Během deploymentu komponenty do kontejneru se pouze specifikuje mapování mezi atributy komponenty a příslušnými poli v databázi. Komponenty typu Bean Managed Entity Bean, na rozdíl od předchozích komponent, mají implementován mechanismus zápisu do databáze a aktualizace atributů z databáze. Je zřejmé, že Container Managed Entity Bean jsou lépe přenositelné a vpodstatě nezávislé na typu databáze. Na druhé straně jejich provoz je asi méně efektivní než je tomu u Bean Managed Entity Beans.

3 HyperQbs – komponentová technologie pro tvorbu prezentační logiky

Prezentační logika pro Internetové aplikace představuje v současné době poměrně horké téma. Přes značné úsilí věnované vývoji Internetových technologií se zdá že vývojáři těchto aplikací se potýkají se stejnými problémy jako vývojáři aplikací pro stolní počítače ve druhé polovině osmdesátých let. Ještě dříve v době terminálových aplikací bylo nutné věnovat značné množství času tvorbě uživatelského rozhraní – zpracování událostí generovaných uživatelem a následně zobrazování příslušných obrazovek. Tento stav je možný porovnat s obdobím kdy se začaly objevovat první Internetové aplikace, které byly postaveny na CGI. Toto rozhraní poskytovalo velmi omezené množství služeb, které mohl vývojář ve svých aplikacích využít a značnou část aplikace musel psát vždy znovu.

Jistým pokrokem v oblasti desktopových aplikací byly objektové knihovny jako bylo například TurboVision, které poskytovalo vývojářům poměrně luxusní služby. Vývojář se potom mohl více soustředit na řešení aplikační logiky, která reprezentovala skutečnou funkcionalitu aplikace. Tento přístup by se dal přirovnat k současnému stavu technologií pro prezentační logiku Internetových aplikací. Technologie jako jsou PHP, JSP, servlety sice poskytují poměrně luxusní služby, vývojář ale stále musí spoustu kódu psát „from scratch“ a znovupoužitelnost je minimální.

Další etapou v psaní desktopových aplikací byly operační systémy s grafickou nadstavbou (Windows, X-Window, NextStep), která poskytovala bohatý soubor služeb pro tvorbu prezentační logiky. V oblasti technologií pro tvorbu prezentační logiky Internetových aplikací je v současné době velmi rušno. Je jasné, že servlety a JSP (součást specifikace J2EE) nevyhovují potřebám na tvorbu a údržbu současných velkých aplikací. Z tohoto důvodu se objevilo mnoho nových technologií, které se pokoušejí situaci změnit.

HyperQbs [4] je technologií, která by mohla představovat další generační skok při vývoji Internetových aplikací.

Její hlavní rysy jsou:

- znovupoužitelnost komponent
- událostně řízený systém
- oddělení práce webdesignera a programátora
- nezávislost na klientském zařízení (WWW, WAP, e-mail, . . .)

Aplikace napsaná pomocí HyperQbs je sestavena z komponent, které mohou prezentovat určité vizuální části WWW (WAP, . . .) stránky. Těmto komponentám se říká Qbs (fonetický přepis anglického cubes). Každá komponenta je zodpovědná za zpracování události, které jsou generovány aktivními prvky na straně prohlížeče, které leží v oblasti zobrazované komponentou. Na základě výsledku zpracování události se provede přesměrování na další Qb, která zajistí zobrazení další stránky. Toto přesměrování je konfigurovatelné v rámci aplikaci a umožňuje měnit vývojáři aplikace sled stránek bez nutnosti zásahu do jakéhokoliv

zdrojového kódu komponenty. Vzájemné provázání stránek není řešeno pomocí externích odkazů na stránkách.

Komponenty, které vývojář aplikace používá mohou být různého typu. Může to být komponenta zabezpečující zobrazení formuláře a nakonfigurovaná tak, že kontroluje zda data vložená uživatelem v daném kontextu jsou správná (například, že do položky „věk“ napsal kladné celé číslo, atd.) nebo třeba komponenta zobrazující tabulku.

Problematickým místem vývoje Internetových aplikací je spolupráce webdesignera (produkujícího HTML kód stránek) a programátora, který výsledky práce webdesignera začleňuje do zdrojového kódu aplikace. Jde o proces velmi pracný a náchylný k chybám. HyperQbs tuto situaci řeší tak, že komponenty neobsahují výsledný kód stránek. Ten je uložen v separátní šabloně, kterou může webdesigner zpracovat běžnými nástroji, které ke své práci používá (Homesite, . . .). Při zobrazování stránky, komponenta nejdříve připraví data, která se budou používat na výsledné stránce a HyperQbs engine zajistí sloučení těchto dat s příslušnou šablonou.

Každá komponenta nemusí být prezentována pouze jedním způsobem ale HyperQbs engine je schopen na základě typu klientského zařízení nebo volby jazyka zajistit aby byla použita nejvhodnější šablona pro danou situaci.

Jak již bylo řečeno HyperQbs komponenty jsou díky své koncepci určeny pro tvorbu prezentační logiky Internetových aplikací. Samotný systém HyperQbs nemá žádné prostředky zabezpečující transakce nebo třeba přístup do databáze. K uvedeným účelům slouží aplikační servery – nejlépe splňující specifikaci J2EE. HyperQbs potom slouží jako systém zabezpečující pro daný aplikační server služby prezentační logiky.

Reference

1. *Technology Forecast 2000*, PrivacywaterhouseCoopers Technology Centre, April 2000.
2. *HyperQbs 2.0 Getting Started*, Qbizm technologies, a.s., January 2001.
3. Paul J. Perrone, *Building Java Enterprise Systems with J2EE*, Sams Publishing, June 2000.
4. `http://www.hyperqbs.org/`