UNIVERZITA KOMENSKÉHO V BRATISLAVE

FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

DYNAMICKÁ VIZUALIZÁCIA INFORMÁCIÍ V SYSTÉMOCH NA PODPORU ROZHODOVANIA NA PLATFORME LOTUS NOTES / DOMINO

Bc. PAVOL SZÓRÁD **2009**

Dynamická vizualizácia informácií v systémoch na podporu rozhodovania na platforme Lotus Notes/Domino

DIPLOMOVÁ PRÁCA

Bc. Pavol Szórád

UNIVERZITA KOMENSKÉHO V BRATISLAVE

FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

KATEDRA INFORMATIKY

Študijný odbor: 9.2.1 Informatika

Vedúci diplomovej práce:

Ing. Miroslav Uhlár

BRATISLAVA 2009

Abstrakt

SZÓRÁD, Pavol: Dynamická vizualizácia informácií v systémoch na podporu rozhodovania na platforme Lotus Notes/Domino. [Diplomová práca] / Bc. Pavol Szórád. - Univerzita Komenského v Bratislave. Fakulta matematiky, fyziky a informatiky; Katedra informatiky. - Školiteľ: Ing. Miroslav Uhlár - Bratislava: FMFI, 2009. -86 s.

Práca skúma problematiku tvorby vizualizačných komponentov pre systémy na podporu rozhodovania na platforme Lotus Notes / Domino. Prináša prehľad vizualizačných metód v kontexte podpory rozhodovania a rozoberá tému kompozitných aplikácií v Lotus Notes. Najdôležitejšou časťou práce je podrobný návod na tvorbu všeobecných komponentov, ktorý umožňuje vytvárať kvalitné znovupoužiteľné komponenty. Práca obsahuje aj prehľad projektov vizualizujúcich dáta na spomenutej platforme a odporúčania pri ich transformácii na vizualizačné komponenty.

Kľúčové slová:

Komponent. Kompozitná aplikácia. Lotus Notes. Podpora rozhodovania. Vizualizácia.

Ďakujem môjmu školiteľovi Ing. Miroslavovi Uhlárovi za odborné vedenie a cenné pripomienky pri tvorbe diplomovej práce. Chcem sa tiež poďakovať mojim rodičom, súrodencom a mojej priateľke Janke za podporu počas celého môjho štúdia.

Čestne vyhlasujem, že túto diplomovú prácu som vypracoval samostatne s použitím citovaných zdrojov.

V Bratislave, 7. mája 2009

.....

Bc. Pavol Szórád

Obsah

Zoznam obrázkov

Predhovor

V rámci svojej profesijnej kariéry som sa ako programátor stretol s pomerne veľkým množstvom programovacích jazykov a vývojových prostredí. Každý programovací jazyk bol určitým spôsobom špecifický, no všetky mali jednu spoločnú črtu. S príchodom novej verzie sa nielen snažili vylepšiť poskytovanú funkcionalitu, ale aj priniesť niečo nové z aktuálnych trendov.

Už dlhší čas sa zaoberám vývojom softvéru na platforme *IBM Lotus Notes/Domino*, ktorej špecifickosť spočíva v „legendárnej" spätnej kompatibilite. IBM garantuje, že existujúce aplikácie budú na novej verzii fungovať rovnako ako predtým. Pre klienta je kľúčové, aby s prechodom na novú verziu nestratil svoje investície. Tento prístup ale sťažuje presadzovanie nových revolučných konceptov pri tvorbe softvéru.

Nová verzia Lotus Notes/Domino 8 prináša zmenu tohto trendu. Jadro systému je postavené na otvorených štandardoch a ponúka natívnu rozšíriteľnosť. Tá spočíva v možnosti vytvárať samostatné programové moduly, takzvané komponenty. Tento krok znamená výrazný posun k architektúre *SOA (Service Oriented Architecture)*. V súvislosti so spomínanými zmenami znova ožíva záujem o open-source iniciatívu zastrešovanú internetovým portálom *OpenNTF.org*. Aktívna Lotus Notes komunita spôsobuje rapídny nárast nových materiálov k verzii 8 – diskusie, overené praktiky, šablóny a predovšetkým knižnice komponentov.

V bakalárskej práci „*Vizualizácia základných algoritmov*" (FMFI UK, 2007) som spracoval možnosti vizualizácie pri výučbe vybraných kapitol z informatiky. K tejto téme mám naďalej blízko a aj preto som si všimol nedostatok komponentov zabezpečujúcich vizualizáciu dát na platforme Lotus Notes. Mojou snahou bolo určitým spôsobom zaplniť túto medzeru. Od pedagogického prostredia som sa ale presunul k „biznis prostrediu" – k systémom na podporu rozhodovania. Pôvodne v Lotus Notes chýbajú možnosti natívnej vizualizácie dát a práve v tomto druhu aplikácií je výrazný rozdiel medzi výpovednou hodnotou nespracovaných dát a ich spracovanou verziou.

Úvod

Vizualizácia dát je grafická reprezentácia konkrétnej informácie. V posledných rokoch sa jej výhody začali vo veľkej miere uplatňovať v podnikovom sektore. Vizualizácia totiž ponúka účinný spôsob ako dáta rýchlo pochopiť a zároveň prezentovať druhým. Efektívna analýza sady dát je kľúčovým elementom systémov na podporu rozhodovania. Systém analyzuje biznis dáta a tie predkladá používateľovi, ktorý môže následne ľahšie robiť informované rozhodnutia. Používateľské rozhranie takého systému sa najčastejšie skladá z niekoľkých rámcov, z ktorých aspoň jeden ponúka práve kontextovú a dynamickú vizualizáciu dát. Softvérová platforma Lotus Notes je líder na trhu s informačnými systémami zameranými na tímovú spoluprácu, takzvaných *groupware*. Jej nová verzia podporuje aplikácie vytvorené zo samostatných rámcov – komponentov. Komponenty sú schopné vizualizácie dát, čo predtým nebolo možné. Ak sa spojí aplikácia s dátami kritickými pre biznis a nová možnosť vizualizácie dát, vzniká ad-hoc systém na podporu rozhodovania s minimálnymi nákladmi vývoj.

Cieľom práce je preskúmať tematiku vizualizačných metód v kontexte systémov na podporu rozhodovania a uviesť čitateľa do problematiky kompozitných aplikácií na platforme Lotus Notes. Najdôležitejšou časťou práce je podrobný návod na tvorbu všeobecných komponentov. Po jeho osvojení by malo byť jednoduché tvoriť kvalitné komponenty pre platformu Lotus Notes. Posledným cieľom je vytvoriť prehľad projektov vizualizujúcich dáta na spomenutej platforme a každému projektu pripojiť odporúčanie, či je vhodné transformovať ho na vizualizačný komponent.

V prvej kapitole popisujeme základné prvky, ktoré viedli k definovaniu cieľov diplomovej práce. Objasňujeme pojem *podpora rozhodovania* a jeho vzťah k informáciám, s ktorými pracuje koncový používateľ. Popisujeme ako sa dajú tieto informácie transformovať do čitateľnejšej podoby – *vizualizovať*. Preberáme postupne vizualizačné metódy. Zo všeobecného hľadiska popisujeme aplikačnú platformu *IBM Lotus Notes*, ktorú sme zvolili na reprezentáciu informácií a vykonávanie vizualizácie.

Udávame dôvody pre tento výber a ukazujeme, ako sa všetky tri hlavné pojmy tejto kapitoly – rozhodovanie, informácia a vizualizácia stretávajú práve na zmieňovanej platforme.

V druhej kapitole objasňujeme architektonické pozadie práce. Pozeráme sa na platformu *IBM Lotus Notes/Domino* z hľadiska technológie a rozoberáme zmeny, ktoré nastali v novej verzii 8. Ide predovšetkým o transformáciu pôvodného prísne proprietárneho a uzavretého systému na otvorený systém založený na platforme *Eclipse*. Týmto krokom je v konečnom dôsledku umožnené plnohodnotné nasadenie aplikácií podľa požiadaviek iniciatívy *Service-Oriented Architecture (SOA).*

Koncept kompozitných aplikácií je centrálnym bodom tretej kapitoly. Zavádzame pojem *kompozitná aplikácia,* vymenovávame klady a zápory kompozitných aplikácií a vo vzťahu k nim nové role členov tímu. Rozoberáme *Component Application Editor (CAE)* – systém na tvorbu kompozitných aplikácií platformy Lotus Notes. Popisujeme jednotlivé typy komponentov a ich interakciu. Ukazujeme, ktoré entity hrajú rolu pri výmene informácií medzi komponentmi.

V kontexte Slovenska je povedomie o možnostiach novej verzie stále minimálne. Literatúra o tvorbe kvalitných komponentov v slovenčine neexistuje. Ťažiskom práce je preto štvrtá kapitola, v ktorej uvádzame súbor najlepších praktík použiteľných pri tvorbe komponentov, všetko v rámci konvencií načrtnutých architektúrou SOA. Tvorba komponentov a ich skladanie do kompozitných aplikácií je viacfázový proces. Býva distribuovaný medzi celú škálu členov tímu – od dizajnérov, cez vývojárov, administrátorov až po používateľov. Fázy vývoja zahŕňajú plánovanie, dizajn, vývoj NSF a Eclipse komponentov, ich skladanie do aplikácií, testovanie a nasadzovanie kompozitných aplikácií. V každej časti bližšie rozoberáme detaily jednotlivých činností.

Obsahom poslednej kapitoly je prehľad projektov vizualizujúcich dáta na platforme Lotus Notes. Analytickou metódou sme skúmali každý projekt a výsledkom je odporúčanie, ako je možné ho transformovať na vizualizačný komponent.

1 Informácie a rozhodovanie

1.1 Rozhodovanie

Podpora rozhodovania

Podpora rozhodovania je v princípe všetko, čo pomáha pri tvorbe správnych rozhodnutí. Je to veľmi široký pojem a môže označovať čokoľvek od jednoduchých tabuliek až po komplexné neurónové siete. Cieľ týchto nástrojov je však ten istý – napomáhať v procese rozhodovania. V dnešnom svete sú spoločnosti nútené rýchlo robiť správne rozhodnutia aby boli schopné prežiť v konkurenčnom trhovom prostredí. Preto sa včasné a presné informácie stali kritickým prvkom, ktorý rozhoduje o úspechu alebo neúspechu.

Jednou z najdôležitejších činností súčasných IT oddelení sa stala úloha získavania informácií z nespracovaných dát, ktoré produkujú firemné informačné systémy. Výstupy sú pripravené do podoby, ktorá umožňuje ich použite pre rozhodovanie. Spracované informácie sú následne použité na hlbšie pochopenie súvislostí, ako sú rozpoznanie a zhodnotenie nastupujúcich trendov, odhalenie vnútrofiremných problémov a aj identifikácia potrieb zákazníka. Najmä pri rýchlo sa meniacich pomeroch v obchodnom prostredí predstavuje skoré odhalenie špecifických nastávajúcich zmien rozhodujúci náskok oproti konkurencii.

Systémy na podporu rozhodovania

Systémy na podporu rozhodovania sú zjednodušene počítačové programy, ktoré napomáhajú rozhodovaniu. Anglický výraz pre je tento termín znie *decision support systems (DSS)*. DSS aplikácia analyzuje biznis dáta a tie predkladá používateľovi, ktorý môže následne ľahšie robiť informované rozhodnutia. V štandardnej aplikácii pribúdajú dáta v priebehu práce koncového používateľa. DSS naopak cielene zbiera dáta, spracuje ich a prezentuje podľa zvoleného cieľa (Gucer, a iní, 1999). Pre naše účely budeme používať termín *systém na podporu rozhodovania* vo

veľmi voľnom zmysle. V podstate pôjde o ľubovoľný informačný systém postavený na platforme *Lotus Notes*, ktorý je vhodné vylepšiť o vizualizáciu. Vhodné sú také aplikácie, v ktorých sú kvantitatívne (absolútne hodnoty) a kvalitatívne (trendy, súvislosti) parametre údajov prehľadnejšie po vizualizácii.

Oblasti použitia

Klienti z oblasti biznisu sú prvým a najčastejším používateľom systémov na podporu rozhodovania. V rámci tejto práce sa zameriavame práve na oblasť podnikových systémov. Podpora rozhodovania má však svoje uplatnenie aj v iných oblastiach, príklady sú klinický rozhodovací systém, DSS prepojený s geografickým informačným systémom, DSS pre identifikáciu digitálnych súborov alebo aj sociálny DSS, ktorý analyzuje skutočný život používateľa. Všetci chceme robiť správne rozhodnutia, ktoré povedú k požadovaným cieľom a tieto systémy k tomu napomáhajú.

Kategórie

Haettenschwiler (1999) podľa zdroja zo stránky en.wikipedia.org rozdeľuje systémy na podporu rozhodovania podľa vzťahu k používateľovi na *pasívne, aktívne* a *kooperatívne*. Pasívny systém na podporu rozhodovania napomáha pri procese, ale neprichádza s konkrétnym návrhom alebo riešením. Aktívny systém navrhuje riešenia. Pri kooperatívnom systéme používateľ dostane návrhy, doplní ich a odošle naspäť systému na validáciu. Systém tiež doplní návrhy a pošle používateľovi. Tento cyklus sa opakuje, kým sa nedosiahne konečné stanovisko. Grafické znázornenie dát je podľa tohto rozdelenia vhodné v každom type DSS. Pri pasívnych najviac, pretože vizualizácia priamo pomáha pochopiť význam dát. Pri aktívnom a kooperatívnom type sa zobrazí grafika vysvetľujúca, čo viedlo k výberu konkrétneho riešenia.

Power (2002) navrhuje taxonómiu na základe typu podpory nasledovne:

- *DSS na základe modelu* – tieto systémy pracujú na základe štatistického, finančného, optimalizačného alebo simulačného modelu. Vstupom pre model

je niekoľko parametrov, ktoré systém potrebuje na zhodnotenie situácie. Dát v princípe nemusí byť veľké množstvo.

- *DSS orientovaný na komunikáciu* podporuje prácu tímu ľudí na jednom spoločnom projekte.
- *DSS orientovaný na dáta* – dôraz je na prácu s časovo závislými internými a externými dátami spoločnosti.
- *DSS orientovaný na dokumenty* získava, spravuje a manipuluje s neštruktúrovanými informáciami v elektronických dokumentoch.
- *DSS orientovaný na znalosti* využíva na riešenie problémov znalosti z niektorej konkrétnej expertnej oblasti uložené ako fakty, pravidlá a procedúry.

Vizualizácia informácie má uplatnenie predovšetkým na DSS na základe modelu a na DSS, ktoré sú orientované na dáta.

História a techniky

Prvé DSS existovali už v roku 1980. Ich účelom bolo pomáhať pri rozhodovaní v obchodných záležitostiach. Vo väčšine prípadov išlo o tabuľky, ktoré boli vylepšené o analýzy typu *„čo ak“*. Niektoré produkty dokonca používali lineárnu analýzu na generovanie informácií z limitovaných pracovných dát (Gucer a iní, 1999).

Najstaršou a najjednoduchšou metódou je prístup *„ukáž mi čo sa stalo“*. Vo všeobecnosti do tejto kategórie spadajú tabuľkové procesory a základné ad-hoc informačné nástroje. Výsledné správy sú vytvárané pomocou preddefinovaných dotazov. Následne je na tieto dáta použitý nejaký druh filtrovania. Pre niektoré situácie je tento prístup vyhovujúci a ponúka dostatok relevantných informácií.

Ďalšou technikou je takzvané *data mining* – dolovanie dát. Jednoducho povedané, data mining ponúka odpovede na otázky typu *„ukáž mi moje trendy“*, alebo *„ukáž mi zaujímavú vec o mojom biznise“*. Data mining je metóda získavania skrytých informácií a prognóz z pracovných databáz. Pomáha používateľom vyťažiť zo svojich dát maximum a predpovedať budúce trendy a správanie sa skúmaných veličín. Dáta sa

vždy spracovávajú podľa určitého modelu a preto patrí technika data mining do kategórie *DSS na základe modelu*. Model môže byť rozličnej zložitosti, ale vo všeobecnosti automatizuje proces nachádzania skrytých vzorov, ktoré je pri manuálnom hľadaní ľahké prehliadnuť. Vizualizácia má pri data miningu široké uplatnenie, zobrazuje sa nielen model, ale aj konkrétne výsledky.

Rozšírená je aj technika *OLAP (Online analytical processing).* Je možné ju zaradiť do kategórie *DSS orientované na dáta,* pretože jej hybnou silou sú informácie, ktoré analyzuje z viacerých rozličných perspektív. Analýzy OLAP sú vždy späté s nejakou konkrétnou otázkou – na rozdiel od data minig, kde otázka nie je jasná.

Aktuálne trendy

Moderné systémy požívajú rôzne multidisciplinárne prístupy – spájajú v sebe oblasti ako sú databázy, umelá inteligencia, interakcia človeka s počítačom, simulačné metódy, softvérové inžinierstvo a vizualizácia dát (Wikipedia, 2009). Tieto lepšie systémy na podporu rozhodovania tak preklenuli určitú kritickú hranicu. Sú kľúčovým prvkom systémov pre manažérov, už nie ako novátorstvo a experiment, ale ako vyžadovaná súčasť aplikácií zameraných na biznis. Stále vo väčšej miere sa vkladajú do systémov, kde to predtým nebolo zvykom. Jedným z takých biznis systémov je aj IBM Lotus Notes. DSS aplikácie na tejto platforme sa podľa zaraďujú medzi pasívne DSS. Orientované môžu byť na dáta, komunikáciu a dokumenty. Data mining ani OLAP natívne neposkytujú. V ďalších kapitolách popisujeme, ako je možné tento stav zmeniť.

1.2 Vizualizácia informácií

Spoločnosť je konfrontovaná explóziou dát. Techniky vizualizácie umožňujú výskumníkom, analytikom, manažérom a bežným používateľom rýchlo získať prehľad o týchto dátach. Je to vďaka unikátnym schopnostiam ľudského vizuálneho vnímania, ktorý umožňuje rozpoznať zaujímavé skutočnosti a vzory v krátkom čase. Pojem vizualizačná metóda sa vo všeobecnosti dá definovať ako grafická reprezentácia zobrazujúca informáciu. Metóda musí byť externá, systematická a založená na pravidlách. Spôsob reprezentácie musí zlepšovať prehľad, napomáhať detailnému pochopeniu alebo sprostredkovávať skúsenosti (Lengler, a iní, 2006).

Kategorizácia

Aj vďaka výraznému posunu v informačnej sile počítačov bolo v posledných rokoch vytvorených mnoho vizualizačných techník. Ralph Lengler a Martin Eppler z Institute of Corporate Communication vo Švajčiarsku vypracovali v roku 2006 sústavu 100 vizualizačných metód a nazvali ju *Periodická tabuľka vizualizačných metód pre manažment* (uvádzame ju v prílohe). Prináša popisný prehľad metód v celej sfére záujmu a slúži ako štruktúrovaný inventár metód. Vďaka prehľadnosti a kompletnosti tejto tabuľky je jednoduché priradiť konkrétnemu vizualizačnému problému vhodné riešenie pre individuálny aplikačný kontext. Pre ďalšie detaily o jednotlivých metódach odkazujeme na tento projekt, pre naše účely prinášame krátky prehľad.

Typy vizualizácie

Vizualizácia informácií je definovaná ako používanie interaktívnej vizuálnej reprezentácie dát na umocnenie kognitívneho rozmýšľania o nich. Sú to napríklad sémantické siete alebo zobrazenie informácií v stromoch (Lengler, 2006).

Vizualizácia dát zahŕňa štandardné kvantitatívne formáty ako sú koláčové, plošné alebo čiarové grafy. Je to vizuálna reprezentácia dát v schematickej forme so súradnicovou osou (alebo bez nej) (Friendly, a iní, 2006). Používajú sa v mnohých situáciách, často na zobrazenie prehľadu dát, ktoré sú časovo závislé. Odpovedajú na

základnú otázku *„koľko“* a tým často aj na otázky *„prečo“* a *„ako“.* Práve tento druh odpovedí priamo súvisí so správnym rozhodovaním.

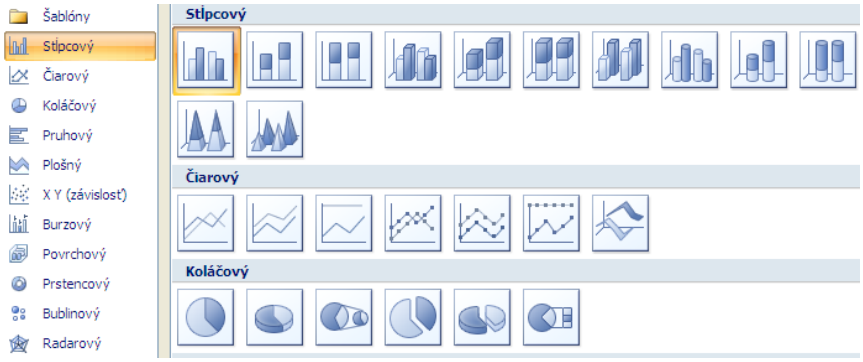
Vizualizácia konceptov, napríklad v Ganntovom diagrame, slúži na rozpracovanie kvalitatívnych konceptov, ideí a plánov. Jednotlivé myšlienky sú zvyčajne pospájané, ak medzi nimi existuje súvislosť. V poslednej kapitole práce sa zaoberáme aj projektmi na tvorbu myšlienkových máp. Tieto patria práve do spomenutej kategórie.

Vizualizácia pomocou metafory sa dá predstaviť najjednoduchšie na príklade šablóny komixového príbehu. Jednotlivé obrázky sú usporiadané do štruktúry a každý rámec zobrazuje kľúčové charakteristiky použitej metafory (Eppler, a iní, 2005).

Vizualizácia stratégie zlepšuje analýzu, vývoj, formuláciu, komunikáciu a implementáciu stratégií v organizáciách. Najviac sú tieto metódy relevantné v manažmente. Príkladom je Strategy Canvas alebo technology roadmap.

Kombinácia vizualizácií spája do jedného kontextu viac druhov vizualizácií, ktoré sú schopné existovať samostatne, ale spolu ponúkajú pohľad na koreláciu dát.

Napriek spomenutým možnostiam sa tieto vizualizačné metódy v reálnej praxi používajú málo. Predstava používateľov je zaťažená skúsenosťami s tabuľkovým editorom Microsoft Excel. Na otázku, čo požadujú od systému na vizualizáciu dát často zaznieva odpoveď – *„to čo je v Exceli”*. Znamená to zobraziť hodnoty vo forme grafu.



Obrázok 1 Ukážka typov grafov v editore MS Excel *(Zdroj: Microsoft, 2009)*

1.3 Aplikačná platforma Lotus Notes

Kľúčové prínosy

Ako aplikačnú platformu na ukladanie dát, ich transformáciu a vizualizáciu sme zvolili produkt IBM Lotus Notes / Domino. Softvér na základoch Lotus Notes je hlavne o flexibilite v biznise. Napomáha individuálnej a organizačnej efektivite tým, že poskytuje nástroje na skvalitnenie tímovej spolupráce v celej šírke firemných procesov. Potenciálne znižuje výdavky spojené s IT službami, pretože dovoľuje pracovať s rozdielnymi ľuďmi, informáciami a aplikáciami z jediného používateľského prostredia. Ľahko sa kombinujú a spracovávajú informácie z jedného alebo viacerých zdrojov, a to poskytuje významnú pridanú hodnotu. Práca s dátami prebieha v aplikáciách, ktoré:

- sú zamerané na ľudí (*people-centric applications*),
- podporujú spoluprácu v rámci tímu (*collaboration software),*
- umožňujú prácu, ktorej stredobodom je práve vykonávanú činnosť (*activity-centric computing).*

Táto robustná platforma sa neustále vyvíja a ponúka prispôsobiteľné aplikácie. Na vývoj kvalitných riešení predurčujú platformu Lotus Notes integrované kľúčové schopnosti, medzi ktoré patria napríklad:

- multiplatformovosť a spätná kompatibilita,
- robustné bezpečnostné prvky,
- integrované úložisko dát,
- fulltextové vyhľadávanie,
- centrálna správa aplikácií,
- podpora práce v režime offline.

K týmto treba pripočítať aj platformu pre rýchly vývoj aplikácií (*rapid application development framework*) a najnovšie prídavky ako sú rozšíriteľná klientská platforma *Eclipse* alebo natívna podpora technológie *Web services.* Všetko je prístupné

cez štandardný model *kompozitných aplikácií* (Lotus Software, 2007). Práve v modeli kompozitných aplikácií sa spájajú pojmy informácia, vizualizácia a rozhodovanie.

Reprezentácia informácií v Lotus Notes

Jednou zo zvláštností terminológie Lotus Notes je ťažko rozoznateľný rozdiel medzi pojmami databáza a aplikácia. Príčinou tejto nejasnosti je integrované úložisko dát. Práca s ním je navrhnutá veľmi intuitívne a vývojár sa nemusí starať o detaily zapisovania a čítania z databázy, ktorá leží pod používateľským rozhraním. Toto stavané úložisko dát robí spolu s bezpečnosťou a fulltextovým vyhľadávaním z Lotus Notes ideálnu platformu pre agregáciu firemných dokumentov a dát.

Vizualizácia v Lotus Notes

Vizualizácia vo všeobecnosti nebola silnou stránkou Lotus Notes. Nová verzia Lotus Notes 8 ale umožňuje kvázi natívnu vizualizáciu dát. Dáta síce neopustia bezpečné prostredie klienta Lotus Notes, no vizualizáciu vykonáva špeciálna importovaná súčasť používateľského prostredia (komponent) naprogramovaná na platforme Eclipse. Ako je toto dosiahnuté vysvetľujeme v nasledujúcej kapitole.

Rozhodovanie

Softvér Lotus Notes je nasadzovaný predovšetkým v biznis prostredí. Tu platí zaujímavá rovnica, ktorú si zatiaľ ešte mnoho organizácií neuvedomilo: biznis prostredie spolu so šikovne uloženými a pozbieranými dátami v pôvodných aplikáciách plus nové možnosti vizualizácie vo výsledku dávajú ad-hoc systém na podporu rozhodovania. Ten je vzhľadom na štandardné systémy na podporu rozhodovania neobvyklý, pretože bežnú požívateľskú aplikáciu zmeníme na DSS. Vizualizáciu je možné pridať do akéhokoľvek informačného systému na platforme *Lotus Notes 8*, čím sa zvýši hodnota aplikácie a produktivita používateľa.

2 Architektúra

2.1 IBM Lotus Notes

Lotus Notes/Domino je to predovšetkým platforma na vývoj *groupware* aplikácií, teda softvéru, ktorý podporuje tímovú spoluprácu. Príkladom je aplikácia pre mobilné zariadenia, ktorá umožňuje *real-time* spoluprácu tímov rozmiestnených po celom svete a poskytuje zdieľané úložné priestory jednotlivé pracovné skupiny.

Renomované prieskumy odhadujú celosvetový počet používateľov aplikácií na platforme Lotus Notes na vyše 130 miliónov (Lotus Software, 2007). V budúcnosti sa predpokladá nárast počtu používateľov. IBM spúšťa iniciatívu, aby sa nielen rozširoval záber aplikácií, ktoré bežia na tejto platforme, ale aby vývojári mali možnosť zdieľať medzi sebou nové, svieže myšlienky. Ku všetkému prispieva aj začlenenie sa do svetovej komunity okolo platformy Eclipse, na ktorej je postavené od verzie 8 jadro systému. Má preto zmysel vyvíjať úsilie, aby táto komunita mala k dispozícii stále lepšie prostriedky na prácu a aby bol naplno využitý jej potenciál.

Lotus Domino, Notes, Designer a Administrator

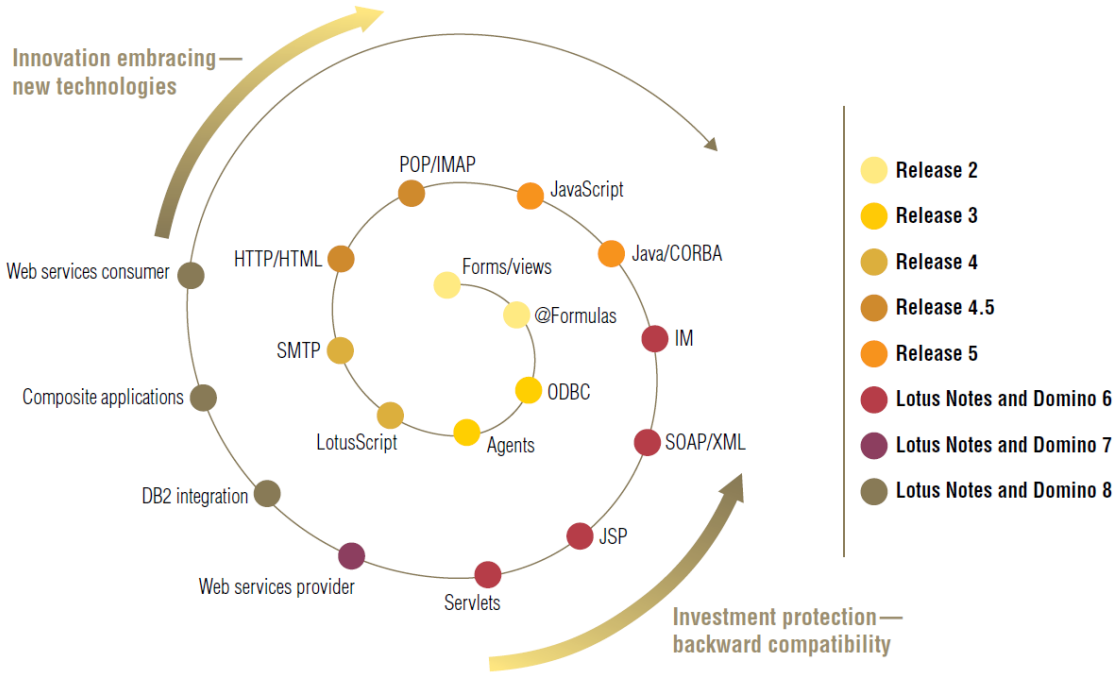
Architektúra platformy je typu klient – server. Server nesie označenie *Domino* a klient sa nazýva *Notes*. *Lotus Domino* je robustný databázový server. Sú na ňom umiestnené aplikácie prístupné cez klienta Lotus Notes alebo aj cez internetový prehliadač. Aplikácia môže byť uložená na viacerých serveroch a preto je prostredie Domino distribuovaným systémom. Synchronizácia jednotlivých databáz sa zabezpečuje procesom *replikácie.* Domino podporuje otvorené štandardy ako XHTML, XML a Java a celý súbor služieb, napríklad POP3, IMAP, LDAP a HTTP. Dôležitým znakom Lotus Notes je schopnosť centrálne organizovať nasadzovanie aplikácií a *pluginov.* Tento znak sa nazýva *server-managed provisioning* (Mindzora, a iní, 2007). Klient *Lotus Notes* ponúka nástroje podporujúce tímovú spoluprácu. Tieto je možné nasadiť ako základnú infraštruktúru elektronickej pošty a plánovania alebo ako aplikačnú platformu pre biznis, pripadne je možné nasadiť oboje.

Ku skupine patria ešte vývojové prostredie Designer a nástroj na spravovanie servera a klientov Administrator. *Lotus Domino Designer* je integrované vývojové prostredie (*Integrated Development Enviroment - IDE*) na vývoj aplikácií pre Lotus Domino. Od verzie 8.5 je založené na *Eclipse RCP.* Plne podporuje model kompozitných aplikácií a zabezpečuje podporu vývoja znovupoužiteľných komponentov. Na ich vývoj slúži integrovaný nástroj *Composite Application Editor*. *Lotus Domino Administrator* je klient na správu servera Domino, ktorý umožňuje manažovať používateľov, ich práva, zaradenie do skupín a pomáha administrovať servery a umiestnenie databáz. Administrácia sa vykonáva v grafickom prostredí alebo cez konzolu a často pomocou takzvaného administračného procesu bežiaceho v pravidelných intervaloch.

Lotus Notes verzia 8

V auguste 2007 vyšla verzia produktového radu Lotus Notes/Domino s majoritným číslom 8. Priniesla mnoho zmien, najdôležitejšou bolo prepracované jadro systému, ktoré je teraz založené na platforme Eclipse. Tým, že sú základy technológie Lotus Notes 8 postavené na otvorenej platforme sa Lotus Notes stáva automatickým prijímateľom všetkých vylepšení od *open-source* komunity - prídavky a opravy chýb v Eclipse sú zároveň vkladané aj do Lotus Notes (Auriemma, 2008).

Lotus Notes and Domino 8 zohľadňuje myšlienky *Service-oriented Architecture (SOA)* vo významnejšej miere ako predchádzajúce verzie. Okrem natívnej podpory technológie *web services,* majú vývojári vo verzii 8 k dispozícii vývojový model na tvorbu *kompozitných aplikácií,* čo je jeden zo spôsobov ako zaviesť SOA architektúru. Základom kompozitnej aplikácie je voľné pospájanie nezávislých, znovupoužiteľných častí používateľského prostredia (takzvaných *komponentov*) do spoločného kontextu relevantného pre koncového používateľa. Zlúčenie technológie Eclipse so softvérovou platformou Lotus Notes umožňuje používať aj komponenty z heterogénnych technológií. Tým sa IT stratégia môže zmeniť smerom k SOA a zároveň sa využijú prostriedky investované do aktuálnych systémov (Darmawan, a iní, 2008).



Obrázok 2 Verzie Notes, nové technológie a spätná kompatibilita *(Zdroj: IBM, 2008)*

Ochrana investícií

Každá verzia Lotus Notes Domino priniesla nové a vylepšené vývojové nástroje na tvorbu softvéru. Jednu črtu však zachovávajú všetky verzie – „legendárnu“ spätnú kompatibilitu. Aplikácie vyvinuté na predchádzajúcich verziách sa dajú vo verzii 8 spustiť a používať korektne bez akejkoľvek zmeny. Môžu však byť rozšírené a využívať nové možnosti platformy. Je teda na výber pôvodná politika tesnej integrácie aplikácií s prostredím, alebo sa zmena aplikácií na kompozitné s voľnými komponentmi. Do určitej miery je zabezpečená aj dopredná *forward* kompatibilita, ako popisuje Benz (2005). Investície do aplikácií tak zostávajú zachované.

Multiplatformovosť

Podpora viacerých operačných systémov dovoľuje nasadiť aplikácie na báze Lotus Notes do produkčného prostredia na všetkých hlavných platformách (Windows, Linux, OS X a iné). Pre konkrétny operačný systém je možné využívať jeho špecifické prednosti aby Lotus Notes dosahoval vysoký výkon a robustnosť. Táto škálovateľnosť znamená jednoduchú migráciu aplikácií z jednej podporovanej platformy na druhú.

Bezpečnosť

Lotus Notes obsahuje integrovaný zložitý autentifikačný model a detailnú granulárnu bezpečnosť. Je založená na bezpečnostných prvkoch uložených v *ID súbore* jedinečnom pre každého používateľa. Asymetrické šifrovanie *RSA* zabezpečuje štyri úrovne bezpečnosti: overenie totožnosti, povolenie prístupu, šifrovanie správ a digitálne podpisy (Wikipedia, 2009). Oprávnenie používateľa na prístup ku konkrétnej informácii sa riadi v šiestich stupňoch:

- Prvý stupeň je prístup k Domino serveru, overuje sa cez certifikáty patriace organizácii, do ktorej používateľ patrí.
- Prístup ku konkrétnej databáze je riadený cez jej *Access Control List (ACL)*.
- Každá identita má ešte presnejšie špecifikované práva – môže byť manažér aplikácie, vývojár, editor, autor, čitateľ, zapisovateľ alebo môže mať explicitne zakázaný prístup. Navyše sa dajú zakázať alebo povoliť ďalšie práva ako mazanie dokumentov alebo spúšťanie akcií. Identita môže mať pridelenú svoju rolu, čo slúži na medziúrovňové rozdelenie identít do skupín.
- Každý záznam má riadený prístup – svojich autorov a čitateľov, iné identity k nemu nemôžu pristupovať, hoci sa dostanú do databázy.
- Dokumenty môžu mať navyše chránené sekcie a
- chránené polia.

Bezpečnostný model umožňuje mať na jedinom Domino serveri niekoľko kritických systémov, bez vzájomnej interferencie. Nastavenia bezpečnosti zahŕňajú okrem iných aj šifrovanie lokálnych databáz, politiky bezpečnosti klientov a *Execution Control List (ECL)*, ktorý zabezpečuje čo všetko môžu jednotlivé entity systému na danej klientskej stanici vykonávať. Tieto možnosti zabezpečenia pomáhajú znižovať riziko kompromitácie kritických firemných dát.

Replikácia

Jednou z podstatných schopností aplikácií na platforme Lotus Notes and Domino 8 je podpora fungovania v režime *offline*. Zmeny, ktoré sa vykonajú na dátach sú potom zasielané pri najbližšom pripojení na server. Tento proces sa volá *replikácia.* Jedna inštancia aplikácie sa nazýva *replika* a má rovnaké identifikačné číslo *(replicaID)* ako ostatné repliky konkrétnej databázy. Všetky repliky si medzi sebou navzájom môžu vyrovnávať dáta, najčastejší prípad je ten, že jedna replika je uložená na serveri a k nej sa pripájajú ostatné. Replikácia prebieha manuálnym spustením alebo automaticky, obojstranne alebo jednostranne a je možné vybrať aj selektívnu replikáciu. Replikácia nekompromituje dáta uložené v databázach, používateľ má k nim prístup podľa pôvodného bezpečnostného modelu. Schopnosť replikácie dát umožňuje efektívnu spoluprácu tímu ľudí, ktorý sú na geograficky rôznych miestach alebo často offline.

Rýchly vývoj aplikácií a znovupoužite kódu

Lotus Notes podporuje takzvaný rýchly vývoj aplikácií (z anglického *Rapid application development*). Lotus Notes podporuje rýchly vývoj aplikácií možnosťami ako sú efektívne nasadzovanie aplikácií pomocou centrálnej politiky, predštruktúrovaný úložný priestor pre dáta, vstavaná tvorba používateľského rozhrania, široká podpora programovacích jazykov, objektový model a prístupné API (Application programming interface). Čas, ktorý ubehne od zberu požiadaviek k nasadzovaniu aplikácie býva veľmi krátky. Bez nutnosti značných investícií do vývoja je možné jednoducho integrovať všetky elementy potrebné k práci koncového používateľa. Fakt, že Lotus Notes 8 je postavený na platforme Eclipse otvára nové možnosti. Nová platforma nijako nenarúša integritu pôvodných aplikácií, stále sú rovnako použiteľné ako predtým. Všetky prvky pôvodnej *Lotus Notes storage facility (NSF)* aplikácie je možné separátne vybrať a znovu použiť ako komponenty v novej kompozitnej aplikácii, spolu s komponentmi na báze iných technológií. Lotus Notes podporuje tvorbu znovupoužiteľných komponentov, a tak má firma nástroje na rýchlu a jednoduchú tvorbu prispôsobiteľných aplikácií podľa aktuálnych požiadaviek.

Programovacie nástroje a technológie

Lotus Notes podporuje tieto druhy programovacích a skriptovacích jazykov:

- *Simple actions* – v preklade *jednoduché akcie* sa používajú na tvorbu triviálnej programovej logiky. Najčastejšie ide o nenáročné nastavovanie logických väzieb v časovaných procesoch v aplikácii (tieto programové jednotky sa nazývajú *agenti*). Simple actions sa vytvárajú pomocou základného editora, kde sa akcie vyberajú z roletového menu a spájajú pomocou logických spojok .
- *Jazyk Formula* – je to nekomplikovaný skriptovací jazyk. Niekedy je nazývaný podľa svojich dvoch častí *@-functions* a *@-commands,* keďže ide v podstate o súbor okamžite použiteľných funkcií a príkazov viazaných na platformu Lotus Notes. Šikovný vývojár je schopný pomocou programových konštruktov v jazyku formula vytvoriť takmer tú istú funkcionalitu ako vo vyšších jazykoch.
- *Lotus Script* – je to objektovo orientovaný jazyk na základe jazyka *Basic.* Lotus Domino poskytuje objektový model a tento procedurálny jazyk je používaný tam, kde je potrebná komplexná programová logika. Ako samozrejmosť podporuje znovupoužiteľosť vlastných objektových tried a rozšírení.
- *Java* – Java rozhrania smerom k objektovému modelu Lotus Domino umožňujú používať javovský kód v aplikáciách na báze Lotus Notes a poskytujú lokálne a vzdialené možnosti volania pre iné systémy.
- *C, C++* – pre tieto jazyky existuje *API* na sprístupnenie aplikácií na Domine

Vývojové šablóny

Model šablón v rámci Lotus Notes umožňuje okamžite vytvoriť novú kompletnú aplikáciu na základe inej a následne manažovať zmeny dizajnu práve cez zmeny v šablóne. Je možnosť nastaviť, aby aplikácia automaticky dedila zmeny dizajnu zo svojej šablóny. Týmto sa dá efektívne oddeliť vývojová aplikácia od nasadenej, v ktorej môžu byť citlivé dáta. Je možné použiť tie šablóny, ktoré sú predprogramované inými vývojármi a spoločnosťami, alebo vytvoriť šablónu z ľubovoľnej existujúcej aplikácie. V každom prípade je vývoj pomocou modelu šablón komfortný a prináša veľký nárast pracovnej efektivity vývojára.

2.2 Eclipse Rich Client Platform

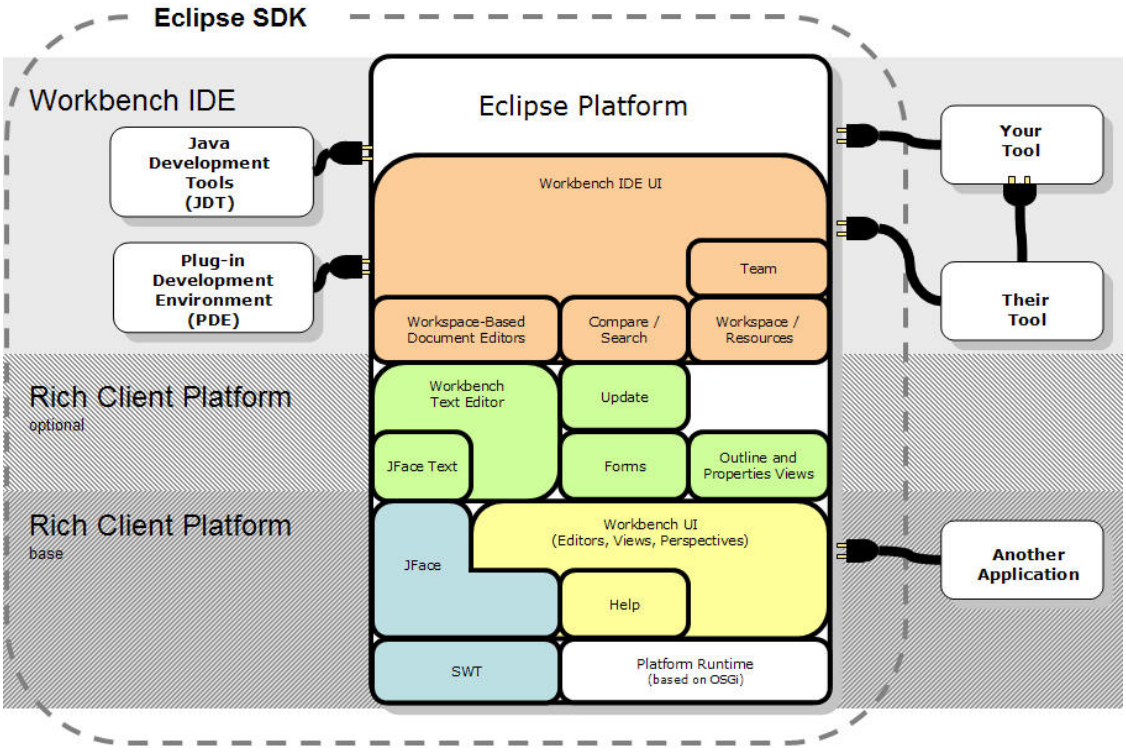
Vývojová platforma Eclipse

Programovací jazyk Java má potenciál splniť lákavú predstavu *„napíš raz, spusti kdekoľvek“* (Carlson, a iní, 2008). S touto predstavou podporila firma IBM v roku 2001 open-source projekt *Eclipse* ako odpoveď na chýbajúce IDE pre Javu. Epicentrom projektu je tvorba otvorenej vývojovej platformy s rozšíriteľnými frameworkmi, nástrojmi a knižnicami. Platforma sa zameriava na manažovanie životného cyklu softvérového projektu od tvorby, cez nasadzovanie až po udržiavanie. IBM v roku 2004 upustila od roly lídra a projekt Eclipse začala spravovať *Eclipse Foundation.* S podporou komunity sa rozhodlo, že Eclipse nebude len IDE pre vývoj v Jave. Za účelom získania širšej podpory bola do Eclipse IDE pridaná vrstva striktne oddeľujúca bežné aspekty vývoja (editovanie, verzionovanie a manažment súborov) od konkrétnych programovacích jazykov, syntaxe, kompilovania a administrácie balíčkov. Vďaka tomuto prístupu pracujú všetci prispievatelia projektu na spoločnom základe a pritom separátne v rôznych programovacích jazykoch. Každý prispievateľ získava viac zo spoločnej práce, ako by bolo možné na úplne oddelených projektoch. Túto spoluprácu zabezpečuje architektonický model Eclipse IDE tým, že udržiava informácie o vzťahoch medzi oblasťami kódu. Napríklad je mnohonásobne použitý prístup *one-to-many,* teda jeden prvok vo vzťahu k viacerým – jedna služba (ovládací prvok) je použitá vo viacerých systémoch.

Eclipse efektívne manažuje inštaláciu, aktualizáciu a odstraňovanie softvérových komponentov. Preto je jednoduché predstaviť si nasledujúci scenár: ak sa na spoločný základ Eclipse pridajú všetky rozšírenia pre vývoj v Jave, vznikne Java IDE, ak sa pridajú len rozšírenia pre vývoj v C++, vznikne IDE pre C++ a ak sa pridajú obe, výsledok je IDE pre Javu aj C++. Popisovaný model sa ukázal byť natoľko úspešný, že sa Eclipse rýchlo stal významnou vývojovou platformou a upútal pozornosť nielen vývojárov nových IDE, ale aj vývojárov aplikácií pre prácu koncového používateľa v produkčnom prostredí.

Rich Client Platform

Vývojové prostredia sú väčšinou dodávané s knižnicami kódu, ktoré umožňujú rýchly vývoj bežnej funkcionality. Programátori pracujúci s Eclipse v tomto prostredí rozpoznali skrytý potenciál stať sa flexibilnou a rozšíriteľnou platformou na tvorbu používateľských aplikácií. S týmto cieľom začala Eclipse Foundation refaktorovanie Eclipse IDE na *Eclipse Rich Client Platform* (Grant, 2008). Všetky elementy frameworku Eclipse nešpecifické pre IDE boli vyčlenené a utvorili množinu základných knižníc z ktorých sa v konečnom dôsledku skladá *Eclipse RCP.* Vlastné aplikácie vznikajú rozšírením základnej štruktúry RCP – tá poskytuje niekoľko východiskových stavebných blokov na tvorbu bežných súčastí aplikácie, ako sú okno, menu alebo nástrojová lišta. Týmto je zaručený rýchly vývoj bohatých grafických používateľských prostredí – *GUI.*



Obrázok 3 Platforma Eclipse a jej súčasti *(Zdroj: IBM, 2006)*

Vytvoriť rozšíriteľnú, adaptovateľnú a spoľahlivú platformu je zložitý problém. Eclipse RCP to dosiahol aj tým, že je open-source. Vývojári majú tak vždy úplný prístup k informáciám ako platforma funguje. Otvorenosť kódu a politika Eclipse Foundation umožňuje zasielať vylepšenia kódu prospešné pre celú platformu s cieľom ponúknuť nové nástroje a funkcie s každou novou verziou.

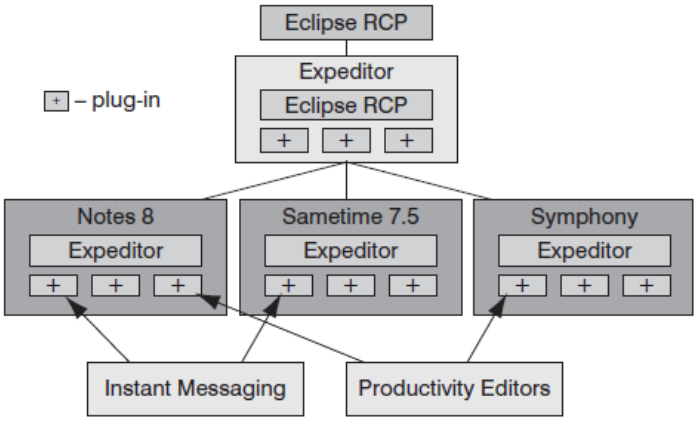
Lotus Expeditor

Ďalší prídavok do produktového radu spoločnosti IBM, ktorý predznačil cestu k Lotus Notes 8, vznikol na podnet meniacich sa požiadaviek trhu. Používateľom nestačili funkcie importu a exportu medzi súborovými formátmi, trendom sa stala tesná integrácia všetkých licencovaných produktov. Táto architektúra orientovaná na služby *(SOA)* sa stala hybnou silou na vytvorenie produktu s názvom *Lotus Expeditor.*

Hoci open-source licencia Eclipse nedovoľuje komerčne predávať samotné RCP, umožňuje to pre prídavky na tomto základe. Podľa pravidiel uvedenej licenčnej politiky pridala IBM množstvo funkcií a programových knižníc nad základné RCP a tak vytvorila Lotus Expeditor. Spoločnosti, ktoré chceli premeniť svoje produkty podľa princípov SOA mohli použiť práve tento základ. Lotus Expeditor ponúka navyše služby ako sú napríklad lokálna relačná databáza, internetový prehliadač, správa bezpečnostných prvkov a ďalej nástroje medzi ktoré patrí server, *toolkit* a univerzálny klient pre PC, tablety a mobilné zariadenia (Carlson, a iní, 2008).

S prostredím na báze Eclipse je jednoduché vyčleniť službu vyvinutú pre jednu konkrétnu aplikáciu a postupne ju použiť v celom rade produktov na základe RCP – zakaždým využiť možnosti navyše, ktoré ponúka aktuálna konfigurácia. Ak tieto služby využívajú niektorú z pridaných Lotus Expeditor funkcií, tak je samozrejme možné nasadiť ich len na produkty so základom Lotus Expeditor. Napríklad službu, ktorá agreguje dáta z *back-end* dátového úložišťa do lokálnej relačnej databázy, je možné bez ďalšej vývojárskej práce použiť v akejkoľvek Expeditor aplikácii. Vývojár sa nemusí starať o rozširovanie infraštruktúry a môže koncentrovať úsilie do oblasti svojej expertízy, čím sa skracuje čas potrebný na samotný vývoj.

Práve na základe Lotus Expeditor sa rozhodla firma IBM prepracovať produktovú sadu Lotus Notes/Domino. Nová verzia 8 je postavená na báze otvorenej vývojovej platformy Eclipse. Získava tak všetky jej vlastnosti a pozitíva. V prostredí Lotus Notes 8 je k dispozícii všetka funkcionalita Eclipse, takmer celý Lotus Expeditor a nové funkcie. Veľa z nich je priamym dôsledkom novozískanej rozšíriteľnosti platformy. Predovšetkým spomenieme *kompozitné aplikácie*, ktoré podrobne rozoberáme v ďalších kapitolách. Cenou za prechod na Expeditor je oproti predchádzajúcim verziám vyššia záťaž procesora a väčšie nároky na pamäť.



Obrázok 4 Inklúzie vývojových platforiem na základe RCP *(Zdroj: Carlson, 2008)*

Balíčkovanie elementov a pluginy

Aby bolo RCP schopné inkrementálne agregovať funkcionalitu, je potrebné presne definovať, ako sa jednotlivé elementy spájajú. V produktoch založených na Eclipse je jednotkou balíčkovania softvéru *plugin.* Je to samostatný súbor funkcionality podobný dynamicky linkovanej knižnici DLL na platforme Windows. Všetky väčšie softvérové elementy sú poskladané z pluginov. Na rozdiel od DLL však Eclipse uvádza pre pluginy prísne štandardy. Každý plugin má svoje číslo verzie – reťazec skladajúci sa z hlavného, vedľajšieho a inkrementálneho čísla, (napríklad *8.0.2*). Navyše musí mať každý plugin svoju identitu, ktorá je najčastejšie nejaká forma hierarchickej adresy ako je *Java package name* (napríklad *com.ibm.notes.java.api*). Eclipse architektúra vypočítava z identity a čísla verzie presné závislosti, ktoré má konkrétny plugin k ostatným.

Vývojár môže obmedziť používanie pluginu A vo vzťahu k pluginu B na

- špecifickú verziu pluginu B
- ľubovoľnú verziu vyššiu ako určená hodnota
- verziu, ktorej hlavné a vedľajšie číslo je rovnaké ako pri plugine A
- inú kombináciu odvodenú od identity a verzie pluginu A

Eclipse runtime po spustení skontroluje všetky pluginy a ich závislosti. Ak pre niektorý nie sú prítomné všetky požadované pluginy, ten konkrétny plugin sa nenačíta. Eclipse tak nenačítava funkcionalitu, ktorú nemôže spustiť. Chyby v kľúčových pluginoch zabránia spusteniu celej aplikácie, ale menej podstatné súčasti sa iba v aplikácii jednoducho neobjavia. Popisovaný prístup je v ostrom kontraste s fungovaním závislostí pri DLL knižniciach. DLL knižnica nič nevie o dostupnosti druhej, kým sa ju nepokúsi načítať. To predstavuje príčinu mnohých neočakávaných pádov aplikácie. Podobne je to aj s identitou. DLL majú čísla verzie, avšak chýba štandard a ich formát závisí na rozhodnutí vývojára. Identita je zároveň priamo naviazaná na meno súboru a premenovanie spôsobí nedostupnosť knižnice.

Pluginy sa spájajú do väčších celkov, tieto entity balíčkovania softvéru sú:

- *feature* – je to agregácia pluginov
- *update site* – je to agregácia viacerých feature
- *aplikácia* – je to súbor pluginov nad spoločným základom (RCP).

Eclipse update site je najčastejšie web stránka prípadne lokálny adresár v špeciálnom formáte, ktorý používajú nástroje na báze Eclipse na balíčkovanie pluginov (cez medzistupeň *feature)*. Kvôli centrálnemu prístupu je vhodné uložiť update site na *HTTP server*, prípadne využiť pokročilú šablónu v Lotus Notes a nahrať update site do *NSF* súboru. Tým sa nekompromituje doterajší spôsob distribúcie zmien ale zachovávajú sa všetky výhody notesovských databáz. Pre všetky tieto dôvody je Eclipsovský model balíčkovania elementov veľkým prínosom pri tvorbe spoľahlivého a udržiavateľného softvéru.

Rozširovanie aplikácií

Balíčkovací model pluginu môže ako taký obsahovať rôzne položky – súbory s vlastnosťami, manifesty, kód, ďalej je možné zahrnúť DLL knižnice so skompilovaným kódom, súbory *JAR (Java Archive)* s javovským kódom alebo aj súbory javovských tried. Plugin je v podstate iba *JAR* súbor plný rôznych položiek. Jednou z týchto entít môže byť takzvaný *bod rozšírenia (extension point)*. Definícia bodu rozšírenia hovorí, ako vytvoriť iný plugin s *rozšírením*, ktoré vyhovuje danému bodu rozšírenia. Bod rozšírenia a rozšírenie sú v istom zmysle pripodobniteľné k elektrickej zásuvke a zástrčke. Zásuvka má určitý špecifický tvar a ponúka možnosť pripojenia. Ak výrobca zariadenia splní podmienky pre tvar zástrčky, môže do zásuvky zapojiť zariadenie s takouto koncovkou. Definícia každého bodu rozšírenia obsahuje schému *XML (Extensible Markup Language),* ktorá ukazuje ako deklarovať druhý fragment XML – rozšírenie tak, aby spĺňal požiadavky na interkomunikáciu medzi týmito dvoma entitami. Bod rozšírenia a rozšírenie sú uložené v súbore *plugin.xml,* čo je súbor pre plugin manifest.

Eclipse RCP je súbor takzvaných kľúčových *(core)* pluginov. Mnoho z nich obsahuje definície bodov rozšírenia napríklad pre pridávanie položiek do menu. Vývojári môžu pridávať do RCP vlastné pluginy, ktoré obsahujú rozšírenia podľa uvedených definícií. Napríklad kľúčový plugin *org.eclipse.ui* slúži ako definícia bodu rozšírenia *org.eclipse.ui.actionSets*, ktorý pridáva elementy do panela nástrojov. Nasledujúci príklad schémy hovorí, kde možné získať obrázok pre ikonu, lokalizovaný *tooltip* text a ako vytvoriť inštanciu triedy, ktorá implementuje určené rozhranie.

Príklad 1: XML fragment rozšírenia na pridávanie položky do panela nástrojov *(2008)*

```
<extension point="org.eclipse.ui.actionSets">
  <actionSet
     id="main.actionSet"
     label="Základné príkazy"
     visible="true">
    <action
     id="jo.ttg.adv2.ui.actions.ActionMoney"
     label="Účty"
     style="push"
     icon="obrazky/ikony/tlacidlo1.gif"
     tooltip="Spravovať financie"
     class="jo.ttg.adv2.ui.actions.ActionMoney"/>
  </actionSet>
</extension>
```

2.3 Service Oriented Architecture

Čo je Service oriented architecture (SOA)?

SOA je skratka pojmu *service oriented architecture* – architektúra orientovaná na služby. Niekedy je tento pojem obohatený o slovo *biznis*, pretože podstatná hodnota SOA pramení zo zamerania sa na podnikovú sféru. V IT terminológii vyjadruje pojem SOA architektonický koncept pri tvorbe softvéru, kde sú na podporu používateľských požiadaviek používané služby. Architektúra je súbor najlepších praktík a návrhových vzorov a dalo by sa povedať, že SOA je vlastne disciplína (Abdollah, 2007). V SOA prostredí ponúkajú jednotlivé uzly svoje prostriedky ostatným uzlom ako nezávislé služby so štandardizovaným prístupom. Definície SOA najčastejšie predpokladajú implementáciu cez *Web Services* (napríklad *SOAP* a predovšetkým *WSDL*, ktoré je popisované v ďalších kapitolách), avšak SOA je možné implementovať akoukoľvek technológiou založenou na službách.

Názory na SOA vo všeobecnosti spadajú do dvoch kategórií: SOA je jedna z najlepších vecí posledných rokov alebo SOA je iba veľa kriku pre nič (Abdollah, 2007). Toto sú extrémne názory a SOA leží niekde uprostred. Treba si uvedomiť, že SOA je skôr evolúcia ako revolúcia. Koncept SOA sa objavil začiatkom deväťdesiatych rokov a prešiel dôležitým vývinom. Dnešné štandardy *web services* sú medzi biznis klientmi a používateľmi omnoho viac akceptované v porovnaní s inými technológiami, čo dopomohlo SOA k rýchlejšiemu prijatiu.

Použitie

Výrazne užitočná je SOA v heterogénnych prostrediach, kde sa často menia biznis požiadavky. Nasadzovať architektúru orientovanú na služby nemá význam, ak je IT prostredie klienta homogénne, alebo ak je prvoradým cieľom výsledný výkon konkrétnej technológie. A keď sa spoločnosť rozhodne pre nasadenie SOA, nikdy to nebude jediná architektúra. Po prvé, SOA veľmi dobre spolupracuje s ostatnými architektúrami a po druhé, SOA ako taká nerieši výzvy spojené s integráciou informácií a sémantiky. Na tieto účely slúžia prídavné technologické riešenia.

Architektúru SOA je možné použiť pri:

- vývoji distribuovaných systémov, ktoré ako služby ponúkajú funkcionalitu koncovým aplikáciám alebo iným službám,
- dizajne a implementácii distribuovaných systémov, ktoré ponúkajú tesnú väzbu medzi biznis modelom a jeho IT realizáciou,
- manažovaní služieb pochádzajúcich z rôznych softvérových balíčkov – ich ďalšie použitie a rekonfiguráciu (Speed, a iní, 2007).

Úspešná implementácia SOA navyše vyžaduje transformáciu procesov v organizácii (vľavo tradičný prístup, vpravo SOA):

- Orientované na funkcie – orientované na procesy.
- Cieľom je vytrvať dlhú dobu – cieľom je možnosť častej zmeny.
- Dlhotrvajúci vývojový cyklus – inkrementálny vývoj a nasadzovanie.
- Aplikačné *silá*, tesná integrácia – voľne zviazané služby.

Najlepší prístup k budovaniu SOA je kombinovať prístup zdola nahor pre tvorbu služieb, ktoré riešia integračné problémy, spolu s prístupom zhora nadol pre návrh architektúry. Ten zabezpečí svieže služby orientované na firemné procesy a modelovacie techniky zase zameranie služieb na správne dlhodobé obchodné ciele.

Charakteristiky SOA

Medzi všeobecne akceptované charakteristiky SOA patria:

- Služby sú znovu použiteľné a volané mnohými aplikáciami.
- Namiesto priamych volaní je prístup k službám cez komunikačné protokoly.
- Služby sú autonómne a voľne zviazané.
- Rozhrania sú definované spôsobom nezávislým na platforme.
- Služby sú implementované podľa pravidiel abstrakcie. Sú zapuzdrené a ich rozhranie neodhaľuje ako je konkrétna služba implementovaná.
- Služby zdieľajú formálny kontakt.

- Služby je možné poskladať do kompozitných aplikácií.
- Služby neudržiavajú vnútorný stav (sú bezstavové - *stateless*).
- Služby je možné zistiť *(discoverable).*

Predpovede nástupu SOA

Na základe priemyselného výskumu Saugatuck Technology budú veľké spoločnosti nasadzovať (alebo už nasadili) SOA ako hlavnú IT stratégiu nezávisle od používanej technológie (Auriemma, 2008). Táto štúdia obsahuje 40 hĺbkových rozhovorov s manažérmi IT oddelení veľkých firiem a odhaľuje, že 37 percent z nich čiastočne alebo úplne nasadili SOA (2007). Hoci Gartner predpovedal až 80 percentnú penetráciu trhu SOA architektúrou (2007) a Saugatuck Technology penetráciu na úrovni 67 percent, tieto očakávania sa kvôli svetovej hospodárskej kríze nenaplnili. Aktuálne predpovede chýbajú, doterajší trend však indikuje pomalý, ale stabilný rast počtu firiem zavádzajúcich SOA. Jeden z kľúčových faktorov vzniku a úspechu SOA je evolúcia štandardov. V minulosti sa mnoho softvérových spoločností snažilo vyvinúť štandard, ktorý by podporoval nejakú formu SOA. Istý čas existovali pokusy ako *CORBA* a *DCOM,* ale nerozšírili sa dostatočne (Speed, a iní, 2007).

Životný cyklus SOA

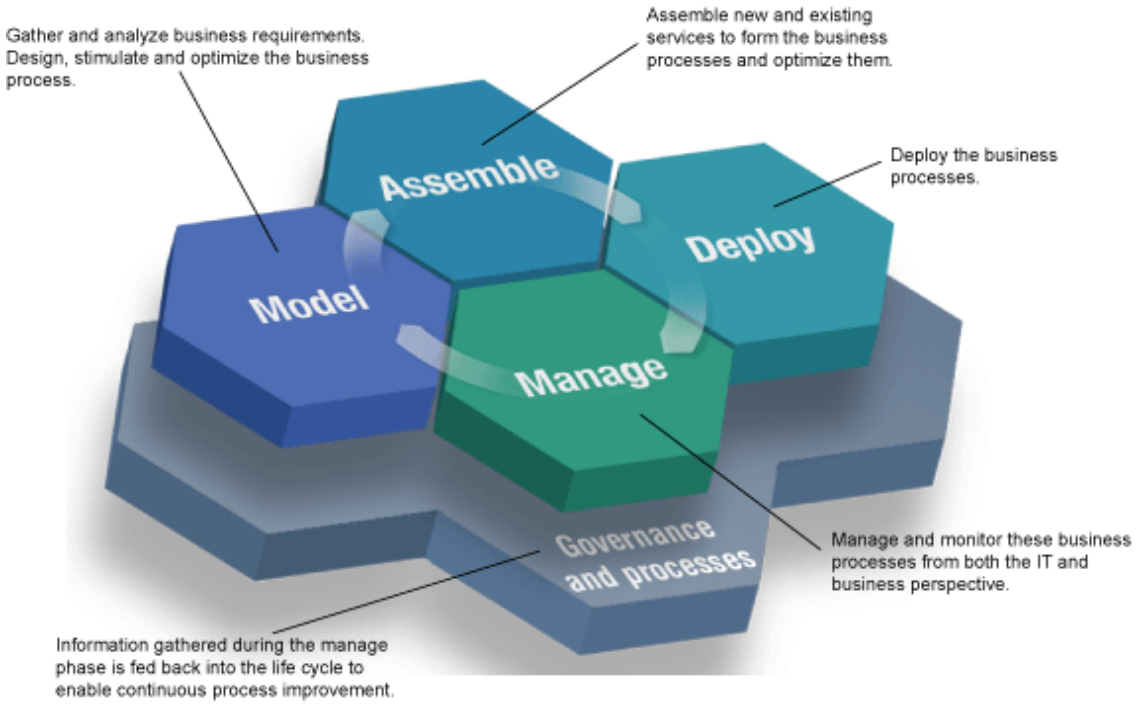
Životný cyklus popisovaný v tejto kapitole je potrebné vnímať ako akýsi framework, v rámci ktorého je možné budovať architektúru orientovanú na služby. Začať s implementáciou SOA je však možné v ktorejkoľvek fáze životného cyklu. Jedna z kľúčových vlastností SOA je možnosť rýchlo získať výhody z nasadenia služieb bez nutnosti čakať na jej plnohodnotnú definíciu (Abdollah, 2007).

Fáza modelovania začína hľadaním častí doterajších aplikácií, ktoré by boli dobrými kandidátmi na znovupoužite. Identifikujú sa služby a definuje sa, ako a kde presne zapadajú do procesov aktuálneho biznis modelu a podľa zistení sa procesy vhodne prispôsobia.

Vo *fáze skladania* sú programy zaobalené ako služby a sú následne použité ako základné kamene pri tvorbe kompozitných aplikácií. Takto sa spoja do spoločného kontextu kľúčové časti biznis procesu, ktoré niekedy pochádzajú zo širokej škály platforiem. Lotus Notes aktuálne podporuje predovšetkým túto fázu. Otvorenosť platformy ale indikuje postupne väčšiu rolu Lotus Notes pri nasadzovaní SOA.

Procesy v *fáze nasadzovania* slúžia na nastavovanie a škálovanie prostredia pre beh kompozitných aplikácií. Cieľom je získať prostredie robustné a bezpečné.

Fáza manažovania zahŕňa vedenie a udržiavanie kontroly nad verziami služieb, sledovanie ich dostupnosti a času odozvy. Monitorovanie faktorov vplývajúcich na výkon je nevyhnutné pre prevenciu, izolovanie, diagnózu a opravu vzniknutých problémov. Sledovať systém je nutné ako z používateľskej, tak aj z IT perspektívy. Zároveň sa z fázy manažovania samostatne vyhodnocujú dáta, ktoré indikujú cesty k zlepšeniu a vďaka ktorým sa životný cyklus stáva kontinuálnym procesom.



Obrázok 5 Životný cyklus SOA *(Zdroj: IBM, 2008)*

3 Kompozitné aplikácie

3.1 Čo sú kompozitné aplikácie

Kompozitná aplikácia je súbor rozhraní k iným individuálnym aplikáciám, ktoré sú prístupné z jediného používateľského prostredia. Je zložená z množstva nezávisle vytvorených, navzájom komunikujúcich komponentov. Tie sú poskladané do spoločného kontextu podľa aktuálnej potreby určitej skupiny používateľov a biznis zámeru organizácie. Kompozitná aplikácia môže obsahovať niekoľko stránok a každá stránka niekoľko komponentov. Vďaka spoločnému kontextu už nie je nutné prepínať medzi oknami viacerých aplikácií, pretože každé potrebné rozhranie k informáciám sa agreguje do jediného miesta (IBM, 2009). Kompozitná aplikácia môže na jednej obrazovke obsahovať interkomunikujúce komponenty ako napríklad vyhľadávač osôb v organizácii, knižnicu dokumentov, virtuálnu komunikačnú miestnosť pre projektový tím alebo *personal management information (PIM)* komponenty – poštového klienta, kalendár a knihu adries.

Prínosy kompozitných aplikácií

Interakcia – Kompozitné aplikácie podporujú komunikáciu medzi komponentmi, čím efektívne znižujú počet krokov potrebných k dokončeniu pracovného procesu. Znižuje sa počet potenciálnych chýb, ktoré môžu nastať pri práci na viacerých systémoch spôsobených viacnásobným vkladaním rovnakej informácie. Spájanie informácií do jedného kontextu napomáha plynulejšej práci a zefektívňuje využívanie pracovného času.

Flexibilný a rýchly vývoj – Existujúce aplikácie na platforme Lotus Notes je možné znova použiť a agregovať do kompozitnej aplikácie. Je možné použiť nové technológie ako sú Java a Eclipse a spojiť niekoľko aplikácií do jednej. Komponenty sú nezávislé – nie sú napevno zabudované do infraštruktúry. IT oddelenie môže poskytovať komponenty vytvorené z pôvodných systémov. Skladanie komponentov

systémom *drag and drop* a voľné spájanie komponentov (*loose coupling*) podporuje ich znovupoužiteľnosť a vo všeobecnosti aj rýchly vývoj aplikácií.

Podpora práce – Spájanie vhodných komponentov do kompozitných aplikácií podporuje prácu zamestnancov – pracujú rýchlejšie a pohotovejšie. Jeden zákazník IBM, po prechode na systémy založené na kompozitných aplikáciách, uvádza pri internom procese časovú úsporu až 85%. Jedná sa o proces, do ktorého je zapojených mnoho systémov. Výsledná kompozitná aplikácia agreguje informácie z každého z nich do jedného okna a tým skracuje čas potrebný na vykonanie procesu z pôvodných 6 minút na konečných 40 sekúnd (Harwood, a iní, 2009).

Integrácia – Kompozitné aplikácie poskytujú jednotný pohľad na početné back-end systémy a aplikácie. Kompozitné aplikácie podporujú široký záber technológií – ako sú napríklad Lotus Notes NSF, .NET alebo Java. Vývojári môžu existujúce aktíva transformovať podľa meniacich sa biznis požiadavok. Nové a aktualizované aplikácie je možné efektívne distribuovať k používateľovi spolu s lokálnym dátovým skladom, biznis logikou a bohatým používateľským rozhraním (Carrier, a iní, 2008).

Kompozitné aplikácie sú tiež ústredným bodom a *front-end* reprezentáciou SOA. Umožňujú takzvanú *kontextovú tímovú spoluprácu*. Tímová spolupráca pochádza z integrácie s novými notesovskými funkciami. Kontext je zabezpečený tak, že akcia v jednom komponente vyvolá reakciu v druhom komponente. Napríklad spustí predprogramované operácie alebo veľmi často aktualizuje zobrazované dáta. Toto umožňuje vykonávať úlohy rýchlejšie – používateľ nemusí na zistenie kontextu prepínať medzi aplikáciami. Zároveň je to riešenie pre vizualizáciu dát, pretože stačí vytvoriť jeden vizualizačný komponent a ten nasadzovať podľa potreby do kompozitných aplikácií. Komponent by bol tiež kontextový – graficky by zobrazoval iba informácie, ktoré sú zobrazené alebo vyznačené v zdrojovom komponente.

Typy kompozitných aplikácií

Datacentric – tento typ kompozitnej aplikácie sa sústreďuje na prácu s dátami v oblasti expertízy danej spoločnosti. Dizajn aplikácie umožňuje efektívne manažovanie, prehľadávanie, editáciu a združovanie informácií. Komponenty zobrazujú kompletné dátové entity, ich prehľady ako aj ich agregáty.

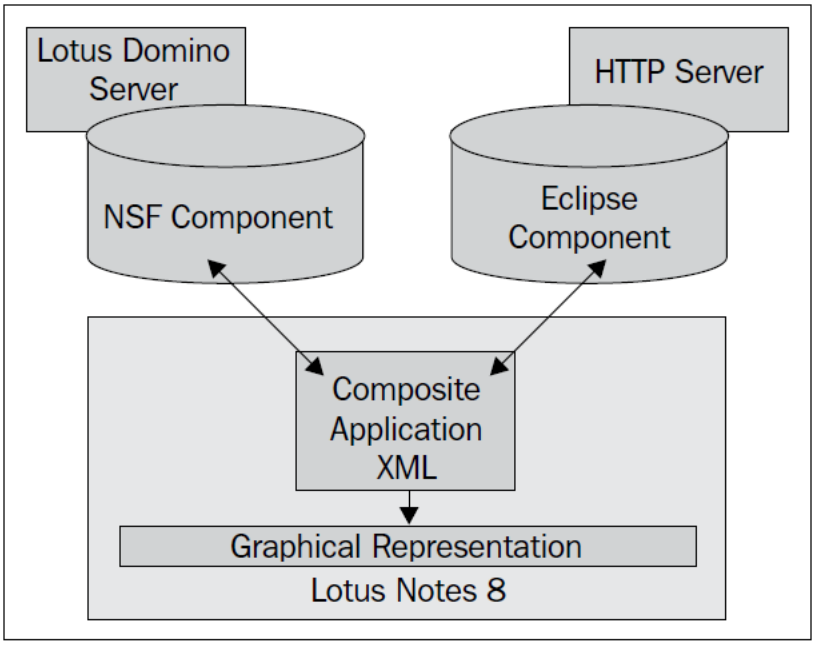
Process-centric – stredobodom tohto typu aplikácie je vykonanie určitých úloh v pracovnom procese koncového používateľa. Napríklad sekretárka vo firme má za úlohu zorganizovať poradu. Aplikácia ju prevedie celým procesom ako cez sériu stránok, na ktorých má vždy potrebné nástroje a informácie k dokončeniu aktuálnej fázy – od vyhľadania kontaktných informácií, cez generovanie pozvánok, objednávku občerstvenia až po tlač dokumentov. Rozhranie takejto aplikácie sa nazýva *dashboard.*

Aggregation model – ide o aplikáciu poskladanú z množstva komponentov usporiadaných na jednej obrazovke. Všetky nástroje pre prácu používateľa sú k dispozícii, chýba ale orientácia na konkrétny proces. Tento model je typický pre prvú fázy nasadzovania kompozitných aplikácií v organizácii (Wolpert, a iní, 2008).

Kompozitné aplikácie v Lotus Notes

Schopnosť zlúčiť do jednej aplikácie heterogénne komponenty je extrémne pôsobivá. Programovací model kompozitných aplikácií v Lotus Notes 8 umožňuje kombinovať komponenty rôznych typov a z rôznych aplikácií. Notes aplikácie teraz obsahujú entity ako sú Eclipse *pluginy*, *portlety*, *gadgety* a *widgety* a priamo umožňujú spoločné fungovanie týchto eclipsovských komponentov s notesovskými komponentmi (dizajnové prvky Lotus Notes ako sú *databázy, framesety, pohľady* a *formuláre*).

Kompozitné aplikácie v Lotus Notes majú niektoré špecifická. Nasadzovanie a údržba prebieha z úrovne servera a je možné ich používať online aj offline. Staré aplikácie je možné rozobrať na komponenty a zložiť do nových tak, že používatelia pracujú v známom prostredí a nie je potrebné preškolenie.



Obrázok 6 Kompozitné aplikácie v Lotus Notes *(Zdroj: Carter, 2008)*

Role v kompozitných aplikáciách

Zodpovednosť tvorby a údržby kompozitných aplikácií je najčastejšie distribuovaná medzi viacerých členov tímu. Každý môže mať pridelenú niektorú z nasledujúcich rolí:

- *Vývojár komponentov* – dizajnuje a vytvára komponenty na báze technológií NSF a Eclipse. Používa svoje preferované vývojárske nástroje, pretože platforma ponúka široké možnosti kompatibility.
- *Application assembler* – Úlohou application assemblera je pomocou *CAE* skladať z komponentov kompozitné aplikácie. Zamestnanec poverený touto rolou nemusí mať žiadne znalosti kódovania v programovacích jazykoch a potenciálne nepotrebuje žiadnu pomoc od IT oddelenia.
- *Administrátor aplikácie* – spravuje NSF komponenty na Domino serveri, vytvára lokálne Eclipse update site.
- *Používateľ* – žiada o nové komponenty a kompozitné aplikácie podľa svojich meniacich sa pracovných procesov

3.2 Composite Application Editor

Vývojári na RCP môžu zložiť viac Eclipse pohľadov do jednej perspektívy. Skompilovaný kód pobeží aj na klientovi Notes 8, pretože perspektívy majú závislosti iba s RCP. Lotus Notes 8 však prichádza s novým nástrojom na zoskupovanie viacerých perspektív do hierarchie stránok. Tento nástroj sa volá *Composite Application Editor (CAE)* a je pri inštalácii klienta Lotus Notes voliteľnou položkou. Tvorba perspektív prebieha bez zmeny kódu – iba vizuálne pomocou *drag and drop.* Je to dvojfázový proces - zvolí sa komponent z palety komponentov a následne sa potiahne a pustí na svoje miesto na stránke v strednej časti obrazovky.

Obrázok 7 Inštalácia editora Composite Application Editor *(2008)*

CAE znižuje odborné požiadavky na rolu tvorcu aplikácií. Pokročilí používatelia Notes so základnými znalosťami administrácie a vývoja môžu skladať kompozitné aplikácie bez nutnosti použiť Domino Designer. Keď sa ukončí zoskupovanie stránok, CAE uloží definíciu kolekcie ako dizajnový element v štandardnej databáze Lotus Notes. Domino Designer potom tento dizajnový prvok zobrazí v novej sekcii s názvom *Composite Applications.* Je možný import a export dizajnu, ale najčastejšie bude použitá editácia cez CAE.

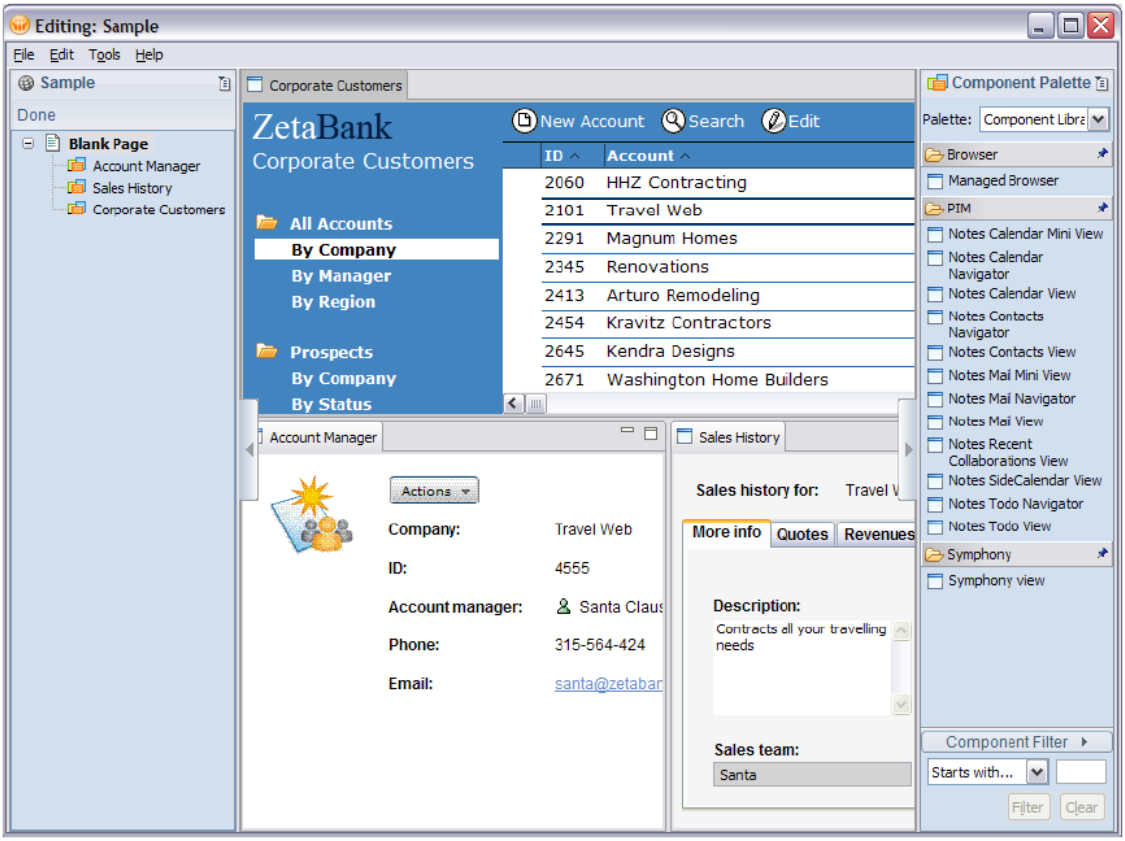
Keď sa spustí kompozitná aplikácia z klienta Lotus Notes, zobrazí sa hierarchia perspektív dizajnovaná predtým. Pri predchádzajúcich verziách sa spustil prednastavený dizajnový prvok - *frameset* alebo stránka (kvôli spätnej kompatibilite je stále možné nastaviť aj takéto správanie). Zmena aktuálnej perspektívy je niekedy riadená programovou logikou, ale najčastejšie môže používateľ sám prepínať medzi perspektívami. CAE podporuje zobrazovanie na kartách (*tabs*), aké sú bežné pri internetových prehliadačoch a táto funkcionalita sa dá detailne nastaviť.

Negatívom pri tvorbe kompozitných aplikácií cez CAE je, že pri editácii nie je kompozitná NSF aplikácia v uzamknutom stave. Môže vzniknúť konflikt ak tú istú aplikáciu editujú v rovnakom istom čase dvaja ľudia. Ktorý z nich uloží aplikáciu ako posledný, prepíše všetky zmeny vytvorené druhým používateľom.

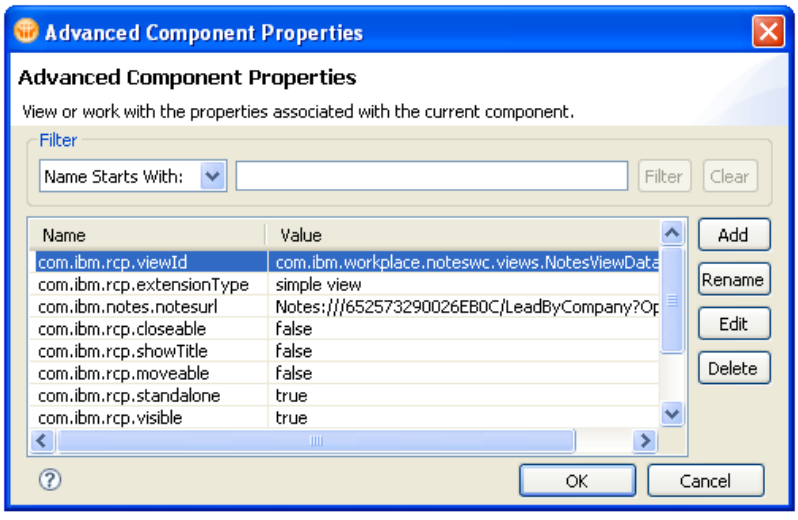
Rozhranie CAE

V kompozitnej aplikácii spustíme CAE cez príkaz v menu *Actions -> Edit Application*. Zobrazí sa integrované vývojové prostredie CAE. V strede obrazovky je umiestnená kompozitná aplikácia a jej komponenty. V pravom paneli je paleta komponentov. Je možné používať viac paliet, medzi nimi prepínať a do každej pridávať komponenty. Ako bolo spomenuté, pridávanie komponentu z palety do kompozitnej aplikácie sa deje systémom potiahni a pusti (drag and drop). V ľavom paneli je v stromovej štruktúre zoznam stránok aplikácie a ku každej prislúchajúce komponenty. Pri každej stránke a pri každom komponente je možné zobraziť pokročilé vlastnosti elementu stlačením pravého tlačidla a výberom možnosti *Advanced properties*.

Ak chceme ukončiť editáciu aplikácie, klikneme *File -> Finished Editing.* Uloženie zmien potvrdíme v dialógovom okne stlačením *OK*. Toto je spomínaný moment, pri ktorom sa natrvalo uložia do aplikácie zmeny bez možnosti ich vrátenia. Považujeme to za veľkú nevýhodu prostredia, pretože prináša nutnosť externe manažovať prístup rôznych vývojárov k zdieľaným zdrojom.



Obrázok 8 Editácia kompozitnej aplikácie v CAE *(Zdroj: IBM, 2008)*



Obrázok 9 Pokročilé vlastnosti komponentu *(2009)*

3.3 Komponenty

Komponent je jeden nezávislý fragment kompozitnej aplikácie. V princípe je jeho viditeľná časť iba obdĺžnik používateľského rozhrania. Je to najčastejšie špecifický pohľad na dáta biznis systému, ovládací prvok alebo niekedy okno aplikácie na inej platforme. V kompozitnej aplikácii komunikuje komponent s ostatnými komponentmi tak, že zverejňuje niektoré svoje dáta a iné prijíma. Ponúkané dáta, ktoré sú vždy presne určeného dátového typu sa volajú vlastnosti *(properties)*. Logika, ktorá v komponente dáta prijme sa volá akcia *(action)*. Spojenia *(wires)* spájajú tieto dve entity a tak umožňujú vzájomnú komunikáciu komponentov.

Typy komponentov v CAE

CAE dovoľuje použiť tri základné typy komponentov:

Portlety – je ich možné zabaliť do triedy *portlet* v prostredí Eclipse a používať ako komponent pomocou CAE.

Eclipsovský pohľad (Eclipse view) – akýkoľvek eclipsovský pohľad definovaný v plugine nasadenom v klientovi Lotus Notes môže byť použitý ako komponent. Vizuálne sa zobrazí do vývojárom definovanej obdĺžnikovej oblasti v používateľskom rozhraní. Aby bola možná funkcionalita vlastností a akcií, je nutné vyplniť niekoľko bodov rozšírenia a pridať kód na podporu zverejňovania a spracovávania dát. Vlastnostiam v plugine sa potom rozumne definujú názvy a dátové typy, používa sa formát WSDL.

Dizajnové prvky Lotus Notes – z určitých notesovských dizajnových elementov je možné vytvoriť komponenty. Sú to: *pohľad (view), frameset, databáza a formulár*. Pomocou CAE sa bez ďalšej práce dajú vytvoriť spojenia s ostatnými komponentmi. Rovnako ako pri eclipsovských pohľadoch sa tieto komponenty vizuálne zobrazia do definovanej obdĺžnikovej oblasti. Aby bola možná komunikácia, musí vývojár nadefinovať akcie do WSDL súboru a prepojiť ich s notesovskými akciami. Použiť je možné aj novú triedu *NotesPropertyBroker* v jazyku LotusScript, ktorá

obsahuje API na prijatie dát z iných komponentov, ako aj zverejnenie dát cez definované spojenia. To znamená, že je možné vykonať logiku na zverejnenie vlastností odkiaľkoľvek, kde sa dá spustiť LotusScript. Domino Designer umožňuje tiež cez vlastnosť zverejniť informácie z niektorého stĺpca pohľadu *(column in a view)*. Hodnoty sa pošlú, keď používateľ vyberie niektorú položku v zozname.

Každý typ komponentu má svoje výhody v rôznych situáciách. Pokročilý programátor dokáže v Eclipse zabaliť do eclipsovských pohľadov iné technológie ako sú ActiveX, C++ alebo .NET a následne ich použiť ako komponenty. Vývojár v Lotus Notes zase ľahko pridá do kompozitných aplikácií existujúce dizajnové prvky z rôznych aplikácií. Vo väčšine prípadov sú tieto dve technológie rovnako funkčné (Grant, 2008). Toto tvrdenie neplatí pre vizualizačné komponenty, ich funkcionalita sa objektívne nedá vytvoriť pomocou štandardných dizajnových prvkov Lotus Notes, ak nerátame možnosť inklúzie Java kódu.

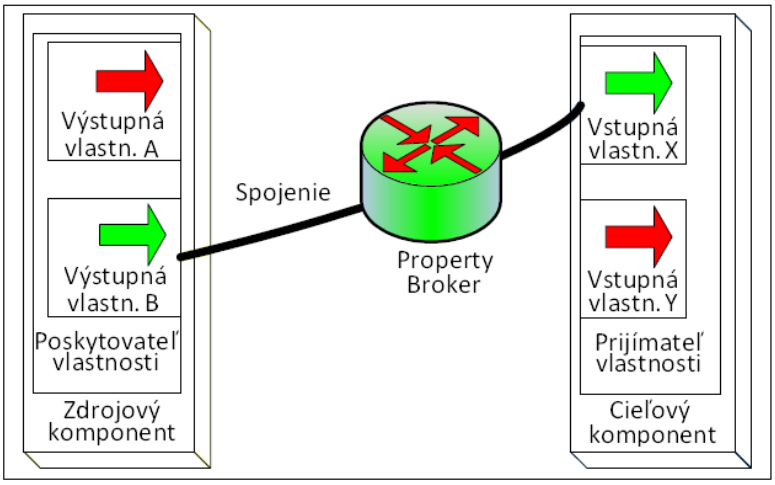
Interakcia komponentov cez Property broker

Funkcionalita v štandardnom klientovi Lotus Notes 8 nazývaná *Property broker* je *runtime* prostredie na spájanie komponentov v NSF kompozitnej aplikácii a manažovanie komunikácie medzi nimi. Komponenty je vďaka Property brokeru možné dynamicky integrovať počas tvorby aplikácie. Nie je nutná predchádzajúca koordinácia počas vývojovej fázy jednotlivých komponentov. Ak je z aplikácie niektorý komponent odobratý, ostatné komponenty pracujú naďalej korektne a koordinovane.

Property broker umožňuje komunikáciu medzi komponentmi tak, že komponenty v ňom registrujú svoje vlastnosti a akcie. *Vlastnosti* ponúkajú ako výstup dáta a *akcie* sú prijímateľmi dát. Napríklad komponent s poštovou schránkou ponúka na výstupe ako vlastnosť *Notes URL* pre práve zvolený Notes dokument zakaždým, keď používateľ zmení výber. Komponent rýchleho pohľadu na aktuálny dokument je prijímateľ daného dátového typu a podľa Notes URL identifikátora vie, ktorý dokument má zobraziť. Aby bolo možné spojiť *(wire)* vlastnosť jedného komponentu a akciu druhého komponentu, je nutné vytvoriť medzi nimi virtuálne spojenie pomocou

nástroja *Wiring* v CAE. Tieto spojenia sa potom stanú súčasťou kompozitnej aplikácie. Keď sa za behu programu zmení hodnota na jednej strane spojenia, táto hodnota je ponúknutá komponentu na druhej strane. Ak je prijímajúci komponent uložený na inej perspektíve, aktuálny pohľad sa prepne do nej. Spojenia môžu byť vo vzájomnom vzťahu jedna k jednej *(1-to-1)* alebo jedna k mnohým *(1-to-many).*

Property broker navyše podporuje jednoduchú tvorbu a manažovanie *WSDL* súborov, ktoré definujú jednotlivé spojenia medzi komponentmi. Na vytvorenie funkčných spojení nie je nutné ovládať syntax WSDL. Property broker a CAE vykonajú túto činnosť namiesto vývojára, ktorý všetko nastaví cez grafické rozhranie.



Obrázok 10 Interakcia komponentov a Property broker *(Zdroj: Rodriguez, 2007)*

Vlastnosť (property)

Vlastnosť (property) je dátová položka definovaného typu, ktorú produkujú komponenty. Všetky komponenty v kompozitnej aplikácii používajú jediný Property broker. Ten šíri správy medzi komponentmi. Komponenty vlastnosti zverejňujú *(publish)* a prijímajú *(consume)*. Zverejnenie vlastnosti prebieha tak, že sa dáta odošlú na preposielanie pre Property broker. Prijímanie vlastnosti znamená, že Property broker prepošle dáta danému komponentu a ten ich spracuje.

Zverejňovanie môže byť automatické alebo riadené programovou logikou. V aktuálnej verzii dovoľuje Property broker zverejňovať iba položky, ktoré sa dajú reprezentovať ako reťazec (Wolpert, a iní, 2008). Neexistuje priamy spôsob ako odoslať komplexnú dátovú entitu tak ako je. Toto obmedzenie sa dá obísť použitím sériového posielania dát. Entita sa rozloží, popíše do XML súboru a ten je následne sériovo poskladaný do jedného reťazca. Táto technika má svoje zjavné nedostatky. Ak sa napríklad zmení jediná položka, je nutné znova poslať celý reťazec. Zároveň s narastajúcou zložitosťou formátu posielaných dát sa znižuje šanca, že bude komponent schopný ľahko komunikovať s inými komponentmi. Pre komponent vizualizujúci dáta je toto obmedzenie obrovské. Vizualizácia má zmysel pre veľké množstvo informácií a nie pre jediný reťazec. Jedinou možnosťou je nateraz dáta odosielať sériovo, pomôcť by mohol špecializovaný komponent na tento účel.

Jednou z charakteristík SOA architektúry je, že služby sú bezstavové. Napriek tomuto faktu sa zverejňovanie vlastnosti sa používa na notifikáciu udalostí. Komplexný komponent môže po skončení výpočtov zverejniť svoj stav, na ktorý čakajú ostatné komponenty a tie môžu následne prispôsobiť zase svoj stav. Takéto zverejňovanie by v záujme znovupoužiteľnosti nemalo ovládať *workflow* v aplikácii. Namiesto toho by mal existovať spoločný komponent na koordinovanie stavov. Detaily zmien stavov sú tak definované až vo fáze skladania aplikácie.

Akcia (action)

Akcia (action) je logika na prijímanie vlastností, teda dát ktoré pošle Property broker. V Lotus Notes je možné spracovávať tieto prichádzajúce dáta iba pomocou programového kódu napísaného v jazyku LotusScript. Termín *akcia* (kvôli prehľadnosti niekedy označovaná ako *WSDL akcia*) v kontexte kompozitnej aplikácie nie je totožný s akciami v Lotus Notes, štandardne používanými ako tlačidlá vo formulároch a pohľadoch *(action buttons)*. Všetky komponenty môžu mať WSDL akcie, bez ohľadu či sú to komponenty notesovské alebo nie. Medzi WSDL akciami a notesovskými akciami je možné vytvoriť pomocou Domino Designera určitý vzťah. Notesovská akcia môže slúžiť ako mechanizmus spracovávania vlastnosti asociovanej

s WSDL akciou. Lotus Notes potom vie identifikovať notesovskú akciu, ktorej má posunúť dáta prichádzajúce cez spojenie definované danou WSDL akciou.

Tak ako pri vlastnostiach, akcie prijímajú len dátové typy, ktoré sa dajú reprezentovať reťazcom. Riešenia na tento problém existujú dve. Prvý spôsob je rovnaký ako predtým – poukladať požadované dáta do série a poslať jediný reťazec. Takto fungujú mailové komponenty – očakávajú dátový typ mailto:string. Ten obsahuje adresy, predmet a obsah správy. Druhé riešenie je založené na nasledovnom princípe: pre každú očakávanú jednotku dát jednoduchého typu (reťazec) vytvoríme zodpovedajúcu akciu. V aplikácii následne umiestnime ovládací prvok (napríklad tlačidlo), ktorý po aktivovaní zozbiera všetky hodnoty z akcií a vykoná tú finálnu. Takto môžu zostať všetky dátové typy jednoduché.

Spojenie (wire)

Spojenie (wire) je externá linka, ktorá spája vlastnosť v jednom komponente s akciami v druhom komponente. Zverejnenie informácie cez vlastnosť tak spustí súvisiacu akciu. Spojenia sa definujú na úrovni kompozitnej aplikácie, zabezpečujú voľné spájanie komponentov a napomáhajú ich znovupoužitiu naprieč mnohými aplikáciami.

WSDL

Komponenty majú k sebe pripojený súbor *Web Services Definition Language (WSDL).* Obsahuje výpočet všetkých použiteľných vlastností a akcií, ktoré je možné spojiť s inými komponentmi, ďalej menný priestor komponentu a iné atribúty. WSDL sa ako súborový formát používa pri definovaní rozhraní web services a je to štandardný XML formát špecifikovaný *World Wide Web Consortium (www.w3.org*). V kontexte kompozitných aplikácií je to iba vhodný formát na ukladanie definícií pre rozhrania komponentov. V Domino Designer 8 sa WSDL súbor vytvára v časti *Wiring properties*. Pre Eclipse komponent sa zahrnie do pluginu a referencuje cez bod rozšírenia v súbore plugin.xml. V obidvoch prípadoch sa používa na tvorbu WSDL súboru Property Broker Editor. Súbory je možné samozrejme napísať aj manuálne.

Menný priestor

V XML poskytuje *menný priestor (namespace)* spoločný kontext pre identifikátory (názvy alebo výrazy), ktoré obsahuje. Presný význam toho istého identifikátora môže byť rôzny v rôznych menných priestoroch. Napríklad to isté šesťciferné telefónne číslo môže označovať celú množinu ľudí v závislosti od použitého menného priestoru – v tomto prípade od predvoľby, ktorá poskytne ten správny kontext. Tento koncept sa používa na rozlíšenie entít, ktoré majú rovnaké meno, ale neobsahujú informácie rovnakého dátového typu.

V kompozitnej aplikácii je možné *spojením* prepojiť *vlastnosť* jedného komponentu s *akciou* druhého komponentu iba ak majú rovnaký menný priestor a dátový typ. V CAE sa menné priestory definujú v časti Property Broker Editor. Ak sa nedefinuje žiadny, CAE použije predvolený *(default)* menný priestor a zároveň bude počas spájania komponentov hľadať zodpovedajúce menné priestory. Vývojárom sa vo všeobecnosti odporúča použiť predvolený menný priestor.

Dátový typ

Dátový typ spája dokopy entity s rovnakou definíciou dát. Definícia nemusí byť súčasťou mena dátového typu, ale vývojári musia sami ustrážiť prideľovanie rovnakých dátových typov entitám s rovnakou definíciou. Ako bolo spomenuté, preferovaným dátovým typom sú jednoduché dátové typy *(simple data types)*. Sú to také, ktoré sa dajú priamo reprezentovať ako reťazec. Je to preto, lebo Property broker dovoľuje výmenu informácií medzi komponentmi práve v tomto formáte.

4 Tvorba komponentov

4.1 Prehľad

Ako sme uviedli, komponenty môžu byť v princípe na báze Notes alebo Eclipse. Nástroje na ich vývoj zahŕňajú napríklad Domino Designer 8 pre NSF komponenty alebo Rational Application Developer pre Eclipse komponenty. Composite Application Editor sa používa na vizuálne skladanie komponentov do kompozitných aplikácií a ich logické prepojenie. V pozadí sa o procesy stará Property Broker. Tento nástroj slúži aj na definíciu elementov, ktoré umožňujú komponentom vymieňať medzi sebou dáta. Ako súborový formát pre definície slúži WSDL, čo je špeciálny druh XML. Composite Application Editor používa tiež formát XML – na uloženie všetkých spomenutých informácií do takzvanej definície kompozitnej aplikácie. Súbor s definíciou sa má označenie ca.xml a je uložený v konkrétnom NSF, ktoré sa otvára cez Lotus Notes 8 klienta.

4.2 Plánovanie

Spoločné fórum

Skutočná integrácia sa nedosahuje iba vytvorením komponentov. Keď chce organizácia využiť plný potenciál kompozitných aplikácií, musí zabezpečiť pre vývojárov spoločné fórum. Na tejto pôde prebiehajú diskusie k aktuálnym otázkam ohľadom vývoja komponentov. Nemalo by však ísť iba o manažovanie komponentov v rámci vývojárskeho tímu. Členovia rôznych pracovných skupín môžu pridávať svoje návrhy, sledovať zmeny vo vývoji a vyhľadávať komponenty užitočné pre ich prácu. Touto synergiou sa niekoľkonásobne zvyšuje hodnota investície do vývoja kompozitných aplikácií. Často sa stáva, že komponent sa nakoniec použije v prostredí, kde by to vývojár vôbec neočakával. Proces funguje aj naopak, používateľ môže natrafiť na komponent, o ktorom nevedel, že je k dispozícii. Kompozitná aplikácia vždy odzrkadľuje aktuálne požiadavky skupiny ľudí na funkcionalitu.

Fáza plánovania

Požiadavky zo spoločného fóra sa časom sformujú do konkrétnej predstavy o funkciách nového komponentu. Tu prichádza na rad fáza plánovania. Vývojár musí pred tvorbou dizajnu zodpovedať niekoľko kľúčových otázok, ktoré ovplyvňujú celý životný cyklus komponentu. Aký je hlavný cieľ komponentu? Pre koho je predovšetkým určený? Aké bude používateľské prostredie? Aké vlastnosti a akcie bude komponent podporovať? Koľko bude stránok? Aká bude interakcia medzi používateľom, čo sa má následne diať v komponente? Tieto otázky úzko súvisia požiadavkou, že komponenty sa musia správať konzistentne a predvídateľne. Musia mať dobrú dokumentáciu, ktorá popisuje prípustné vstupné hodnoty a očakávané výstupné hodnoty.

4.3 Dizajn komponentov

Dizajn a zmeny

Kľúč k maximalizovaniu znovupoužiteľnosti spočíva v predvídaní nových potrieb a zmien v aktuálnych požiadavkách a vzhľadom na tieto faktory dizajnovať systém, ktorý sa dokáže vhodne vyvíjať. Aby si systém zachoval svoju robustnosť napriek všetkým zmenám, je nutné zvážiť, ako sa bude meniť počas svojho životného cyklu. Ak sa faktor zmeny neberie do úvahy, riskujú sa v budúcnosti veľké prestavby. Môžu zahŕňať redefiníciu tried, novú implementáciu, zmeny na klientovi, opakované testovanie a podobne. *Redizajn* sa stáva príliš drahým. Dizajnéri sa snažia zabrániť tomto stavu použitím návrhových vzorov, ktoré predpokladajú niektoré zmeny na systéme. Každý zo vzorov umožňuje pozmeňovať časť systému nezávisle na iných aspektoch a tým vyvinúť robustnejšiu aplikáciu vo vzťahu ku konkrétnemu typu zmeny.

Ako negatívny príklad v kontexte SOA slúžia *tesne zviazané (tight coupled) triedy.* Je ich ťažko použiť izolovane, keďže závisia jedna na druhej. Vedie to k monolitickým systémom, kde nie je možné zmeniť alebo odobrať triedu bez zmeny mnohých ďalších tried. Systém sa stane masou kódu, ktorá sa udržiava veľmi ťažko. Opakom je *voľné spájanie (loose coupling)*, kde je vyššia pravdepodobnosť, že triedy

budú znovu použité aj inde, a že systém je možné ľahko pochopiť a následne modifikovať alebo rozšíriť.

Návrhové vzory

Návrhové vzory *(design patterns)* umožňujú znovupoužiteľnosť úspešných a overených dizajnov a architektúr. Tieto techniky slúžia dizajnérom, na nájdenie „tej správnej cesty“ pri vývoji svojich systémov už od začiatku. Použitie správneho návrhového vzoru zlepšuje nielen výsledný programový kód a dokumentáciu, ale dokáže uľahčiť údržbu – vďaka explicitnej špecifikácii všetkých elementov kódu. Najviac návrhových vzorov je určených pre objektovo orientované jazyky a vychádza zo skúseností s nimi. Niektoré sa ale dajú vhodne transformovať a môžu pomôcť aj pri kvalitnom návrhu komponentov.

Ako uvádzajú Gamma a iní (1994), vzor má štyri najdôležitejšie prvky:

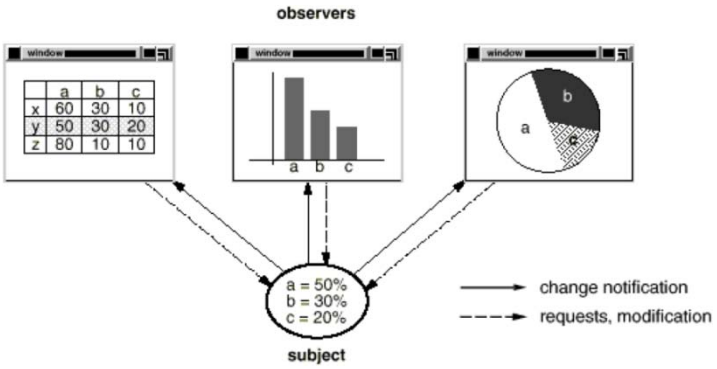
- *Meno* – správne pomenovať návrhový vzor znamená vložiť do jedného, dvoch slov popis problému, jeho riešenia a následky.
- *Problém* je popis, kedy je vhodné aplikovať vzor a za akých podmienok. Objasňuje problém a jeho kontext. Môže obsahovať aj ukážku ako by vyzeral opak dobrého dizajnu – *antipattern*.
- *Riešenie* je súbor detailných popisov elementov dizajnu – ich vzťahov, zodpovedností a spolupráce. Riešenie neobsahuje konkrétnu implementáciu alebo presný dizajn, skôr popis určitej šablóny, ktorá je použiteľná v množstve konkrétnych situácií. Vzor ponúka abstraktný popis problému ako ho rieši všeobecné usporiadanie štandardných elementov ako sú triedy a objekty.
- *Následky* – sú to výsledky nasadenia konkrétneho vzoru a kompromisy s tým spojené. Následky sú kritické pre správne zhodnotenie alternatívnych dizajnov, výhod a nevýhod nasadenia vzoru. Softvérové následky sa týkajú zložitosti času a priestoru, môžu obmedzovať použiteľný programovací jazyk a určovať detaily implementácie. Následky použitia vzoru na znovupoužiteľnosť kódu zahŕňajú kompromisy v flexibilite, rozšíriteľnosti a multiplatformovosti systému.

Model – View – Controller

Model – view – controller (MVC) je architektonický návrhový vzor, ktorého použitie izoluje programovú logiku od používateľského rozhrania. V aplikácii je následne jednoduchšie modifikovať buď jej vizuálnu stránku alebo logiku bez vzájomného ovplyvňovania. V MVC reprezentuje *model* informácie v aplikácii, *view* symbolizuje elementy používateľského rozhrania a *controller* spravuje komunikáciu medzi nimi (Wikipedia, 2009). View musí zaručiť, že zobrazuje aktuálny stav modelu. Vykonáva sa to cez oznamovací protokol nasledovným spôsobom: keď sa model zmení, oznámi to všetkým view a tie sa môžu následne aktualizovať. Pridávanie a odoberanie view sa uskutočňuje nezávisle na modeli (Gamma, a iní, 1994).

Observer

Návrhový vzor MVC je aplikovateľný na omnoho všeobecnejší problém. Je to spôsob, ako oddeliť objekty tak, aby jeden dokázal ovplyvniť niekoľko ďalších bez toho, aby poznal ich detaily. Observer je návrhový vzor, ktorý vychádza z MVC. Používa sa na udržiavanie konzistencie objektov v systéme spolupracujúcich tried (alebo komponentov). Jedna trieda môže notifikovať o zmene svojho stavu ostatné triedy bez nutnosti ich tesného spájania, ktoré by znížilo celkovú znovupoužiteľnosť. Kľúčovými elementmi vo vzore sú takzvaní *subject* a *observer.* Subject môže mať pripojených viac observerov, ktorých upozorňuje na svoje zmeny stavu (Gamma, a iní, 1994). Tento návrhový vzor sa používa pri komunikácii komponentov v kompozitnej aplikácii, kde všetky spojenia spravuje Property broker.



Obrázok 11 Príklad použitia návrhového vzoru Observer *(Zdroj: Gamma, 1994)*

Typy komponentov podľa vzťahu k doméne

Takzvané *doménovo-centrické* komponenty sú základom kompozitných aplikácií. Pokrývajú funkcionalitu z konkrétnej oblasti, v ktorej používateľ pracuje a majú zmysel len v tomto kontexte. Ich cieľom nie je dosiahnuť maximálnu znovupoužiteľnosť. Druhý typ komponentov sú *doménovo-kontextové*. Pre prácu používateľa nie sú nevyhnutné, ale ponúkajú relevantný prídavný obsah pre hlavné komponenty a sú použiteľné vo viacerých nezávislých projektoch. *Všeobecné* komponenty sú posledným typom a ich charakteristikou je, že nemajú žiadnu väzbu k aplikačnej doméne organizácie (Wolpert, a iní, 2008).

Typy komponentov podľa funkcie

Informačné komponenty zobrazujú detaily jedného dátového záznamu. Zobrazovať sa môžu všetky položky, alebo iba výťah najdôležitejších informácií. Kvôli úspore pracovného priestoru sa používa na princíp vyskakujúceho okna.

Komponent typu *Launcher* prijíma dáta z iných komponentov a namiesto informácií pridáva funkcionalitu. Napríklad pozbiera z rôznych elementov potrebné prvky a stlačením jedného tlačidla (alebo splnením konkrétnych požiadaviek) vytvorí e-mail s predvyplnenými poliami. Je to doménovo-kontextový druh komponentu.

Komponent navrhnutý ako *Selection* umožňuje, aby pohyb informácií presahoval jedinú doménu. Ak má viditeľnú reprezentáciu, ide o doménovo-kontextový komponent, ktorý umožňuje vyhľadať konkrétne informácie. Napríklad k osobe to môže byť email, telefónne číslo alebo iné informácie uložené v databáze.

Účelom komponentov typu *Calculation* je prijať vstup, spracovať ho a nové hodnoty ponúknuť na výstupe. Výpočet môže byť napríklad zložité prehľadávanie doménových informácií. Calculation komponenty často nemajú vizuálnu reprezentáciu.

Komponent typu *Aggregation* udržiava mnoho hodnôt a k nim symetrické vlastnosti a akcie. Slúži na prenášanie hodnôt zo stránky na stránku spolu s ich kontextom. Ak akcia nastaví niektorú hodnotu, zverejní sa korešpondujúca vlastnosť.

4.4 Tvorba komponentov založených na NSF

Komponent môže byť založený na technológii *Notes Storage Facility (NSF)*. Dizajnové prvky, z ktorých sa dajú vytvoriť NSF komponenty sú *databáza, stránka (page), formulár (form), frameset, pohľad (view), zložka (folder).* Domino Designer 8 ponúka navyše tvorbu viacstránkových používateľských rozhraní, kontrolu atribútov jednotlivých častí (farba, veľkosť, umiestnenie, rolovanie) a vytváranie programovateľných odkazov, ktoré sa aktualizujú automaticky.

Procesy tvorby NSF komponentov

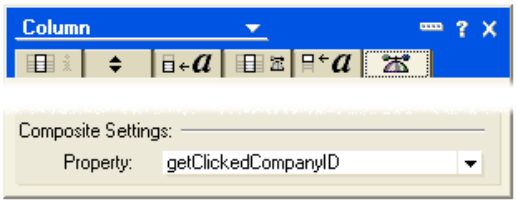
NSF komponent je odkaz na dizajnový prvok v pôvodnej NSF databáze. Bez zmeny je možné ho použiť v kompozitnej aplikácii. Editácia dát v NSF komponente sa vykonáva na dátach pôvodnej databázy. Takýto komponent je samostatným oknom v používateľskom prostredí. Ak však chceme implementovať komunikáciu medzi komponentmi, musíme v Domino Designeri pozmeniť tieto elementy. Proces býva najčastejšie iteratívny a pozostáva z niekoľkých krokov:

1. Určíme vlastnosti, ktoré bude komponent zverejňovať.
2. Definujeme akcie, ktoré sa vykonajú pri napojení nášho komponentu na vlastnosť iného komponentu.
3. Vytvoríme súbor vo formáte Web Services Description Language (WSDL), ktorý bude obsahovať akcie a vlastnosti zverejnené našim komponentom.
4. Importujeme WSDL súbor do NSF zdroja pre komponenty.
5. Modifikujeme dizajnové elementy a prepojíme ich s definovanými vlastnosťami akciami. Ak je vytvorený WSDL súbor, je tento krok možné robiť aj priebežne.

Zverejnenie vlastnosti

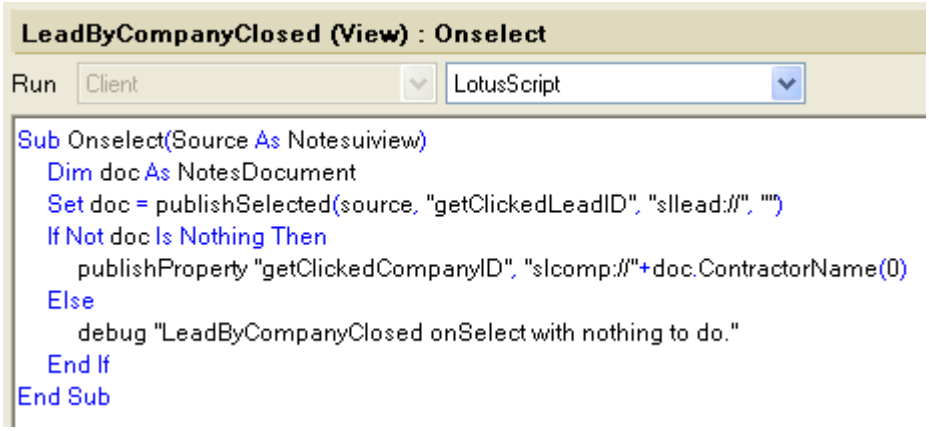
Všetky komponenty v jednej databáze zdieľajú rovnakú množinu dátových typov, vlastností a akcií. Komponent nemusí zverejňovať všetky výstupné vlastnosti, alebo vykonávať všetky akcie, ktoré sú definované vo WSDL súbore. Sú dva spôsoby ako môže komponent zverejniť vlastnosť. Ak je komponent pohľad alebo zložka, je možné asociovať niektorý stĺpec s výstupnou vlastnosťou definovanou vo WSDL.

Nastavenia sa nachádzajú vo infoboxe stĺpca pod kartou *Advanced.* Keďže stĺpec obsahuje len jednu hodnotu, môžu byť k nemu asociované iba vlastnosti typu *simple.*



Obrázok 12 Výrez z infoboxu stĺpca a prepojenie s WSDL vlastnosťou *(2009)*

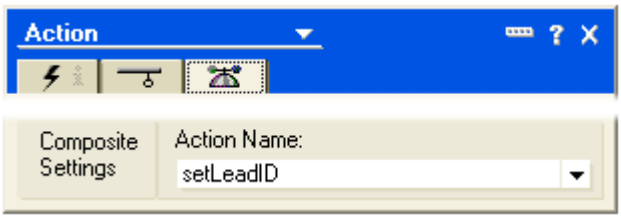
Druhá možnosť je použiť LotusScript API a tak vykonávať zverejnenie kdekoľvek, kde sa dá LotusScript spustiť – vrátane *agentov*. Využívajú sa na to nové triedy *NotesSession.GetPropertyBroker*, *NotesPropertyBroker.SetPropertyValue* a *NotesPropertyBroker.Publish*.



Obrázok 13 LotusScript kód, ktorý zverejňuje vlastnosť cez metódu publish *(2009)*

Prijatie vlastnosti cez akciu

Prijatie vlastností od iného dokumentu je možné len jedným spôsobom. Komponent musí obsahovať notesovskú akciu, ktorá zodpovedá WSDL akcii spojenej s druhým komponentom. Táto asociácia sa nastavuje v infoboxe notesovskej akcie. Aktivácia notesovskej akcie ako reakcia na WSDL akciu pri zmene na ňu napojenej vlastnosti v inom komponente nastane iba ak je dizajnový prvok obsahujúci túto notesovskú akciu na tej istej stránke kompozitnej aplikácie.

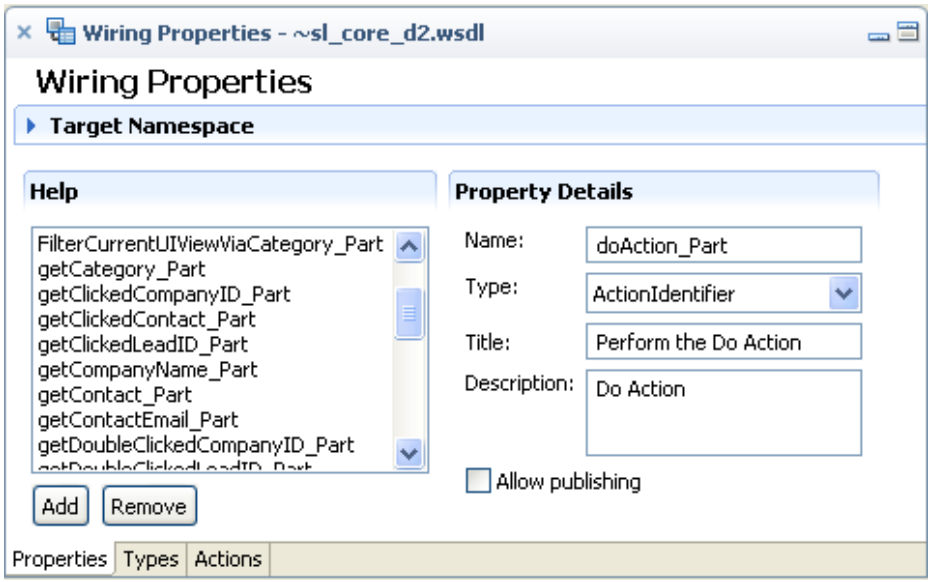


Obrázok 14 Infobox notesovskej akcie a jej prepojenie na WSDL akciu *(2009)*

Je možné spojiť s WSDL akciou aj zdieľanú *(shared)* notesovskú akciu, ale vykonávať sa bude iba ak je umiestnená na formulári v komponente, kde sa má spustiť. Atribút *Hide* na skrývanie notesovskej akcie neovplyvňuje jej vykonávanie vo vzťahu k WSDL akcii. Najlepšou praktikou je používať tento atribút vždy (s hodnotou *true*) pre notesovské akcie volané pri zmenách vlastností. Tieto akcie používajú na načítanie vstupu Property broker a ak by boli viditeľné a aktivované priamo používateľom, tak Property broker pre nich nemá pripravené žiadne hodnoty a vznikne chyba.

Tvorba WSDL súboru

Ak sme identifikovali vlastnosti a akcie pre komponent, vygenerujeme WSDL súbor. Následne ho importujeme do Lotus Notes aplikácie, ktorá obsahuje dizajnové prvky z ktorých chceme vytvoriť NSF komponenty.



Obrázok 15 Property broker, časť Wiring properties *(2009)*

O tieto dva kroky sa stará Domino Designer, import prebieha automaticky. Rozbalíme cestu k dizajnovým prvkom *Composite applications->Wiring properties.* Následne máme na výber *New wiring properties* – ak sme WSDL súbor ešte nevytvorili, alebo ak ideme len editovať, tak možnosť *Open file.*Výberom jednej z týchto dvoch možností sa nám otvorí Property broker, kde definujeme akcie, vlastnosti, ich dátové typy a menný priestor.

Znovupoužitie NSF komponentov

Staršie notesovské aplikácie majú niekoľko zabudovaných vlastností a akcií, ktoré sa v CAE dajú povoliť bez aktualizácie NSF súborov, z ktorých pochádzajú. Toto je výhodné, ak chceme použiť komponenty v kompozitných aplikáciách, ale zároveň chceme zachovať integritu pôvodnej aplikácie. Po sprístupnení komponentu v CAE sa tieto automaticky zaregistrujú do Property Brokera. V dialógovom okne pokročilých vlastností je nutné nastaviť tieto vlastnosti:

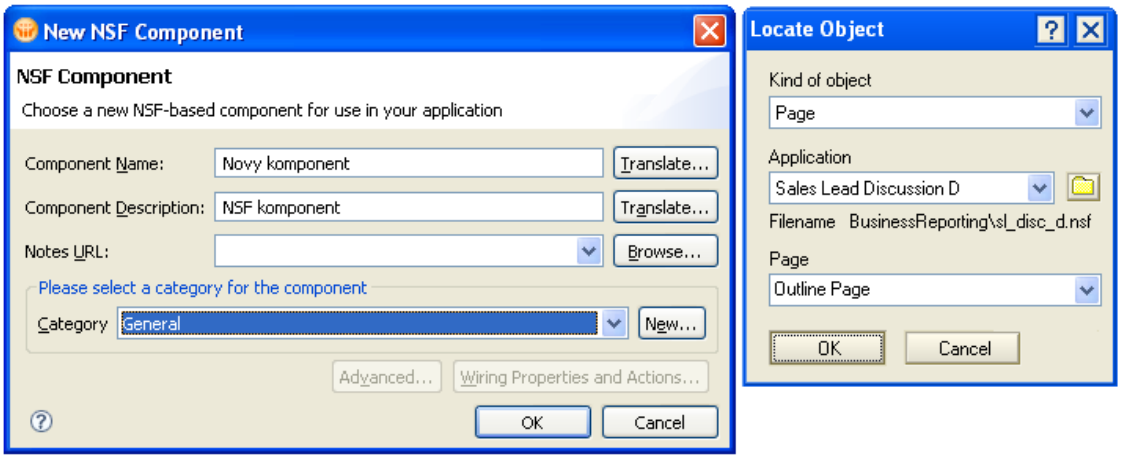
- *com.ibm.notes.enable.preferences* na hodnotu „true“ na stránke obsahujúcej použité komponenty,
- *com.ibm.notes.enableBuiltInPB* na „true“ na každom Notes pohľade,
- *com.ibm.notes.publishBuiltInPropsFromView* na „[view1],[folder1]“ ak chceme zverejniť vlastnosť zo špecifického pohľadu/zložky.

Následne môžeme použiť:

- vlastnosť *SelectedNotesDocumentURLChanged,* ktorá zverejňuje Notes URL aktuálne zvolenej položky v pohľade,
- akciu *FilterCurrentUIViewViaCategory,* ktorá filtruje obsah kategorizovaného pohľadu, rozbalí danú kategóriu a ukáže dokumenty na nižšej úrovni (ak kľúč v pohľade neexistuje, zobrazí sa prázdny pohľad),
- akciu *SearchCurrentUIView,* ktorá prehľadá aktuálny pohľad, výsledky a používateľské prostredie je rovnaké ako pri manuálnom prehľadávaní.

NSF komponent v CAE

Ak chceme komponent používať v CAE, musíme ho najprv vložiť do palety komponentov. Klikneme pravým tlačidlom na paletu a vyberieme z roletového menu možnosť *Add Components -> Add NSF Components.* Zobrazí sa nám dialógové okno ako na obrázku 15. Komponent nazveme, pridáme popis a vyberieme konkrétny odkaz na dizajnový objekt (reprezentovaný cez Notes URL) cez tlačidlo *Browse.* Objaví sa druhé dialógové okno, kde vyberieme typ objektu, cestu k aplikácii, v ktorej sa objekt nachádza a následne dostaneme na výber objekty spĺňajúce uvedené kritériá. Dvakrát potvrdíme výber a zvolený komponent sa zobrazí v ľavej lište na palete komponentov, odkiaľ je možné ho priamo potiahnuť a pustiť na zvolené miesto v kompozitnej aplikácii.



Obrázok 16 Dialógové okná pridávania NSF komponentu do palety *(2009)*

4.5 Tvorba komponentov založených na Eclipse

Proces tvorby Eclipse komponentov

Vývoj všeobecného pluginu v Eclipse pozostáva z niekoľkých krokov: založenie nového projektu pre plugin, vytvorenie tried *(classes)* a vytvorenie grafického rozhrania pluginu. Podrobný popis spomenutých procesov je mimo rozsahu tejto práce. V princípe je možné všetko robiť štandardnými vývojárskymi postupmi. Začiatočníkom odporúčame preštudovať si príslušnú literatúru, prípadne začať programovať s pomocou niektorého predpripraveného balíka ako je napríklad IBM Lotus Expeditor Toolkit (na stiahnutie na adrese *http://www14.software.ibm.com/webapp/download/nochargesearch.jsp?q=Lotus+Expeditor+Toolkit*) alebo ako sú pomocné triedy na prácu s Eclipse komponentmi projektu Composite Applications Component Library (*http://www.openntf.org/Projects/pmt.nsf/ProjectLookup/Composite%20Application%20Component%20Library*).

Komunikácia

Detaily pluginu závisia od konkrétneho projektu, jeho cieľov a implementácie. Spoločný je ale spôsob, ako plugin v roli komponentu komunikuje s ostatnými komponentmi. Pri programovaní vykonávame nasledovné procesy:

1. Určíme vlastnosti, ktoré komponent zverejňuje a akcie, ktoré komponent vykoná po prepojení s iným komponentom.
2. Vytvoríme WSDL súbor so zoznamom akcií a vlastností komponentu.
3. Definujeme bod rozšírenia v plugine (v *plugin.xml*) odkazujúci na WSDL súbor.
4. Naprogramujeme zverejňovanie vlastností, môžeme využiť nové Java API.
5. Naprogramujeme akcie implementujúce rozhrania definované v súbore plugin.xml. Trieda, ktorá implementuje akciu musí už byť definovaná v bode rozšírenia v súbore plugin.xml.
6. Nasadíme komponent vytvorením alebo aktualizovaním projektu Update site (popisované v kapitole 4.8).

Vlastnosti a akcie

Tak ako pri NSF komponente, na začiatku vývoja komponentu naplánujeme vlastnosti, ktoré bude komponent zverejňovať a akcie, ktoré bude používať na prijímanie vlastností. Tieto entity sa potom registrujú v Property brokeri a sú viditeľné v nástroji Wiring na vytváranie spojení v CAE.

Tvorba WSDL súboru

WSDL súbor môžeme vytvárať v Domino Designeri rovnako ako pri NSF komponentoch. Niekedy však vývojár v Eclipse nemá prístup k tomuto nástroju. Ako alternatívu môžeme použiť neintuitívne rozhranie na tvorbu WSDL súborov integrované v Eclipse IDE. Ak sa rozhodneme vytvárať WSDL súbor ručne, pre každý prvok v použitom dátovom modeli je nutné ho deklarovať v štyroch elementoch:

Element *<types>* určuje typ zverejňovanej vlastnosti. V základnej implementácii sú všetky dátové typy reťazce, avšak môžeme pridať do WSDL aj sémantický dátový typ. V rámci viacerých komponentov ale musia byť konzistentné, pretože iba tak ich bude možné prepojiť spojením.

Príklad 2 Oblasť <types> WSDL súboru *(Grant, 2008)*

```
<types>
   <xsd:schema targetNamespace="http://www.ibm.com/wps/c2a">
      <xsd:simpleType name="NasDatovyTyp">
         <xsd:restriction base="xsd:string"/>
      </xsd:simpleType>
   </xsd:schema>
</types>
```

Element *<message>* určuje vstupné a výstupné parametre volania funkcie. Pre každý *getter* a *setter* musí existovať jeden element, ktorý ich popisuje.

Príklad 3 Oblasť <message> WSDL súboru *(Grant, 2008)*

```
<message name="getPrimaryDataResponse">
   <part name="getPrimaryDataValue" type="tns: NasDatovyTyp "/>
</message>
<message name="setPrimaryDataRequest">
   <part name="PrimaryDataValue" type="tns: NasDatovyTyp "/>
</message>
```

Element <portType> spája operácie s elementmi <message>, čím sa určujú vstupné a výstupné parametre.

Príklad 4 Oblasť <portType> WSDL súboru *(Grant, 2008)*

```
<portType name="NasPluginView_Service">
   <operation name="getPrimaryData">
      <output message="tns:getPrimaryDataResponse"/>
   </operation>
   <operation name="setPrimaryData">
      <input message="tns:setPrimaryDataRequest"/>
   </operation>
</portType>
```

Ako posledné definujeme operácie v elemente <binding>, ktoré definujú celý profil a odvolávajú sa na predchádzajúce definície. Určíme aj nadpisy a prídavné popisy pre vlastnosti, ktoré sa zobrazia v CAE.

Príklad 5 Použitie elementu <binding> *(Grant,2008)*

```
<binding name="TagCloudViewBinding" type="tns: NasPluginView_Service ">
   <portlet:binding/>
   <operation name="getPrimaryData">
      <portlet:action name="getPrimaryData" caption="PrimaryData"
      description="primarne data pluginu"/>
      <output>
         <portlet:param name="getPrimaryData" partname="getPrimaryDataValue"
         caption="PrimaryData"
         description=" primarne data pluginu "/>
      </output>
   </operation>
   <operation name="setPrimaryData">
      <portlet:action name="setPrimaryData" caption="PrimaryData"
      description=" primarne data pluginu "/>
      <input>
         <portlet:param name="setPrimaryData" partname="PrimaryDataValue"
         caption="PrimaryData"
         description=" primarne data pluginu "/>
      </input>
   </operation>
</binding>
```

Aby bol WSDL súbor prístupný, po vytvorení ho vložíme do pluginu do novovytvoreného adresára *WSDL*.

Bod rozšírenia

V súbore plugin.xml definujeme rozšírenie, ktoré bude obsahovať názov triedy, ktorá implementuje akciu a odkaz na vytvorený WSDL súbor. V príklade je kurzívou vyznačené miesto, kde sa definuje trieda (class) a odkaz (file).

Príklad 6 Definícia bodu rozšírenia ukazujúceho na WSDL súbor *(IBM, 2009)*

```
<plugin>
  <extension
    id="akciaNacitajURL"
    name="Nacita URL do nasho okna"
    point="com.ibm.rcp.propertybroker.PropertyBrokerAction">
   <action
     active="true"
     class="com.ibm.pvc.samples.propertybroker.browser.LoadURLAction"
     file="loadurlaction.wsdl"/>
  </extension>
  ...
</plugin>
```

Implementácia vlastnosti

Vlastnosť môžeme zverejniť niekoľkými spôsobmi, prvý z nich je vytvoriť objekt typu *PropertyController* a vyplniť všetky polia manuálne. Toto riešenie však predpokladá prítomnosť balíka Lotus Expeditor v inštalácii Eclipse. Druhou možnosťou je použiť nové API pre prístup k Property brokeru.

Príklad 7 Implementácia zverejňovania vlastností *(IBM, 2009)*

```
public Property getProperty(Object owner, String namespace, String propertyName) throws
PropertyBrokerException;

// Následne by volanie vyzeralo takto:

Property prop = broker.getProperty(getViewId(), "http://www.w3.org/2001/XMLSchema",
"ExampleProperty");
```

Implementácia akcie

Je jedno, či používame na tvorbu akcií Expeditor Toolkit alebo bod rozšírenia *PropertyBrokerDefinitions*, v konečnom dôsledku je trieda pre akciu odvodená od *org.eclipse.core.commands.IHandler*. Akcie sú volané pomocou metódy execute vo vnútri triedy, dokumentuje to nasledovný príklad:

Príklad 8 Implementácia akcie *(IBM, 2009)*

```
public class PropertySetAction implements IHandler {

   public void addHandlerListener(IHandlerListener arg0) { // TODO Auto-generated }
   public void dispose() {   // TODO Auto-generated method stub }

   public Object execute(ExecutionEvent event) throws ExecutionException {
      if (event.getTrigger() instanceof PropertyChangeEvent) {
         PropertyBroker broker = PropertyBrokerFactory.getBroker();
         PropertyChangeEvent pce = (PropertyChangeEvent) event.getTrigger();
         PropertyValue pv = pce.getPropertyValue();
                Wire wire = pce.getWireDefinition();

         //Tu príde náš kód
      }  return null;
   }
   public boolean isEnabled() { return true;}
   public boolean isHandled() { return false;}
   public void removeHandlerListener(IHandlerListener arg0) {  // TODO Auto-generated }
}
```

Eclipse komponent v CAE

Komponent vložíme do palety v CAE obdobne ako pri NSF komponentoch, no predtým ho musíme mať uložený v projekte update site. Klikneme pravým tlačidlom na paletu a vyberieme z menu možnosť *Add Components -> Components from Update Site.* Zobrazí sa dialógové okno, vyberieme či pôjde o lokálny alebo vzdialený projekt a zadáme cestu k nemu. Po potvrdení sa do palety z projektu update site načítajú Eclipse komponenty a môžeme ich začať používať v našej aplikácii.

4.6 Skladanie kompozitných aplikácií

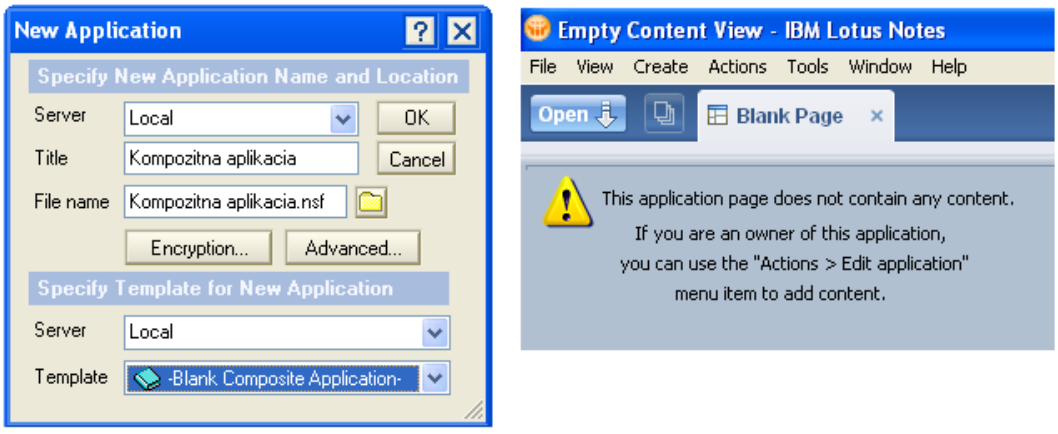
Skladanie komponentov *(assembling)* do aplikácie a ich následné spájanie môžeme vykonávať omnoho neskôr ako samotný vývoj komponentov. Je to potenciálne úloha aj pre koncových používateľov aplikácií. K editoru CAE, kde sa vykonáva fáza skladania, majú prístup Notes používatelia s právami na editáciu. Používatelia si môžu dizajnovať vlastné aplikácie a spájať do kontextu informácie, ktoré práve potrebujú. *Look-and-feel* týchto aplikácií je dobré udržať konzistentný s ostatnými aplikáciami organizácie.

Skladanie kompozitnej aplikácie má nasledovné kroky:

1. Vytvorenie entity *composite application* v Lotus Notes. Predtým je potrebné vytvoriť databázu, ktorá bude obsahovať túto kompozitnú aplikáciu.
2. Pridávanie komponentov do kompozitnej aplikácie. Aplikácia je na začiatku prázdna a komponenty sa pridávajú použitím Composite Application Editora.
3. Spájanie komponentov v kompozitnej aplikácii. Je to finálny krok, keď sa pospájajú vlastnosti a akcie dokopy pomocou nástroja Wiring v CAE.

Nová kompozitná aplikácia

Novú (prázdnu) kompozitnú aplikáciu vytvoríme z prostredia klienta Lotus Notes. Vyberieme z hlavného menu príkaz *File->Application->New*.



Obrázok 17 Tvorba novej, prázdnej kompozitnej aplikácie *(2009)*

Otvorí sa dialógové okno, kde do relevantných polí zadáme informácie o aplikácii. Postupne vyplníme názov servera, názov aplikácie, cestu k súboru a vyberieme ako šablónu aplikácie možnosť *Blank Composite Application.* Otvorí sa novovytvorená kompozitná aplikácia, neobsahuje však žiaden komponent. Ak otvoríme túto aplikáciu v Domino Designer, zistíme, že ide o štandardnú NSF aplikáciu, ku ktorej je pridaný súbor ca.xml. Zároveň je spôsob otvárania nastavený na *Launch as Composite application.*

Pridávanie komponentov a ich rozloženie

Komponenty sa do kompozitnej aplikácie pridávajú v CAE z pravej lišty. Do aplikácie sa umiesťujú spôsobom *drag and drop*. Komponenty môžu byť v aplikácii rozložené podľa rôznych vzorov, v závislosti od celkového zamerania aplikácie a podľa funkcií jednotlivých komponentov.

Pracovný priestor s centrálnym komponentom (radiating dashboard) je rozloženie komponentov typické tým, že stredobodom aplikácie je jediný komponent. Tento býva najčastejšie *doménovo-centrický* a ponúka prácu na hlavnej pracovnej aktivite, ako je prezeranie, editácia a spracovávanie jednotlivých dátových položiek. Ostatné komponenty sú rozložené okolo centrického komponentu a ponúkajú kontextové informácie k spomenutej centrálnej aktivite. Často zaberajú minimum priestoru na obrazovke, až kým nie sú potrebné k práci.

Rozloženie komponentov typu *zoznam (list)* sa používa na zhustenie informácií o mnohých dátových položkách. Centrálny komponent zobrazuje zoznam položiek s rôznymi možnosťami triedenia a usporadúvania dát. Ostatné komponenty zobrazujú sumarizačné informácie o práve zobrazených dátach, ako sú celkový počet prvkov alebo priemer niektorej hodnoty. Takéto rozloženie prezentuje informácie o množine dát ako o jednom celku.

Špeciálnym prípadom rozloženia komponentov typu *Zoznam* je rozloženie typu *výber (selection).* Zo všetkých položiek sa vyberú podľa zvolených kritérií len

niektoré a tak periférne komponenty zobrazujú detailnejšie informácie. Je to rozloženie komponentov na polceste k zobrazeniu informácie v úplnom detaile.

Komponent detailu býva najčastejšie riešený formou vyskakujúceho okna, aby celkovom rozložení nezaberal veľa miesta a zároveň aby používateľ mohol prezerať a editovať kompletnú dátovú položku.

Nástroj Wiring v CAE

Dva komponenty zviažeme spojením pomocou nástroja Wiring v CAE. Jeden komponent bude zdroj *(source)*, ktorý ponúka vlastnosť na prepojenie. Tá bude spojená s akciou v druhom komponente. V CAE zvolíme v ľavej lište s komponentmi aplikácie požadovaný zdrojový komponent a klikneme na neho pravým tlačidlom. Otvorí sa roletové menu a vyberieme možnosť *Wiring*. Zobrazia sa nám všetky komponenty na aktuálnej stránke a pri každom komponente jeho zverejnené akcie.

Obrázok 18 Nástroj Wiring na vytváranie spojení v CAE *(Zdroj: IBM, 2008)*

Spojenie vytvárame tak, že vyberieme jednu vlastnosť zo zdrojového komponentu. Klikneme na ňu, potiahneme a začne sa zobrazovať čiara zakončená

šípkou. Natiahneme šípku k zvolenej akcii v druhom komponente a pustíme tlačidlo myši. Týmto spôsobom sa vytvorí spojenie, čo je vizuálne indikované čiarou medzi komponentmi a zmenou ikonky pri použitej vlastnosti a tiež pri použitej akcii. Ak chceme prepojiť dva komponenty na rôznych stránkach, druhú stránku vložíme tlačidlom *Include another page*. Spájať sa dajú len vlastnosti a akcie s rovnakým dátovým typom. Toto obmedzenie sa od verzie 8.0.1 dá vypnúť tlačidlom *Disable strict type checking*.

Techniky spájania

Pre stránky s malým počtom komponentov vo všeobecnosti nie je nutné vymýšľať zložité techniky spájania, stačí nám *impromptu* spájanie. Keď pridáme nový komponent, tak ho zakaždým prepojíme so zdrojom očakávanej informácie. Pre veľké množstvo stránok je ale tento systém prakticky nepoužiteľný. Mnoho akcií vyvoláva kaskádovité zverejnenie ďalších vlastností. Niektoré sa zverejnia len pri určitých splnených podmienkach a iné pri zmene niektorej hodnoty. V princípe je možné spojeniami vytvoriť aj rekurzívne alebo nekonečné slučky.

Iný prípad je, ak sa v aplikácii nachádza jeden centrálny komponent. Spojenia najčastejšie vychádzajú z neho a spájajú ho s periférnymi komponentmi. Tok dát je od stredu smerom k okrajom, čo sprehľadňuje spôsob fungovania aplikácie.

Najzložitejšie je spájať viacstránkové aplikácie. Ak chceme prenášať informácie medzi stránkami, nestačí iba poslať hodnotu, ale je nutné poslať aj celý kontext. Na toto slúži komponent typu aggregation. Ten prijme vlastnosť, pozbiera celkový kontext a prepošle ho druhému aggregation komponentu na zvolenej stránke. Tam sa kontext poskladá a rozpošlú sa hodnoty príslušným komponentom.

4.7 Testovanie

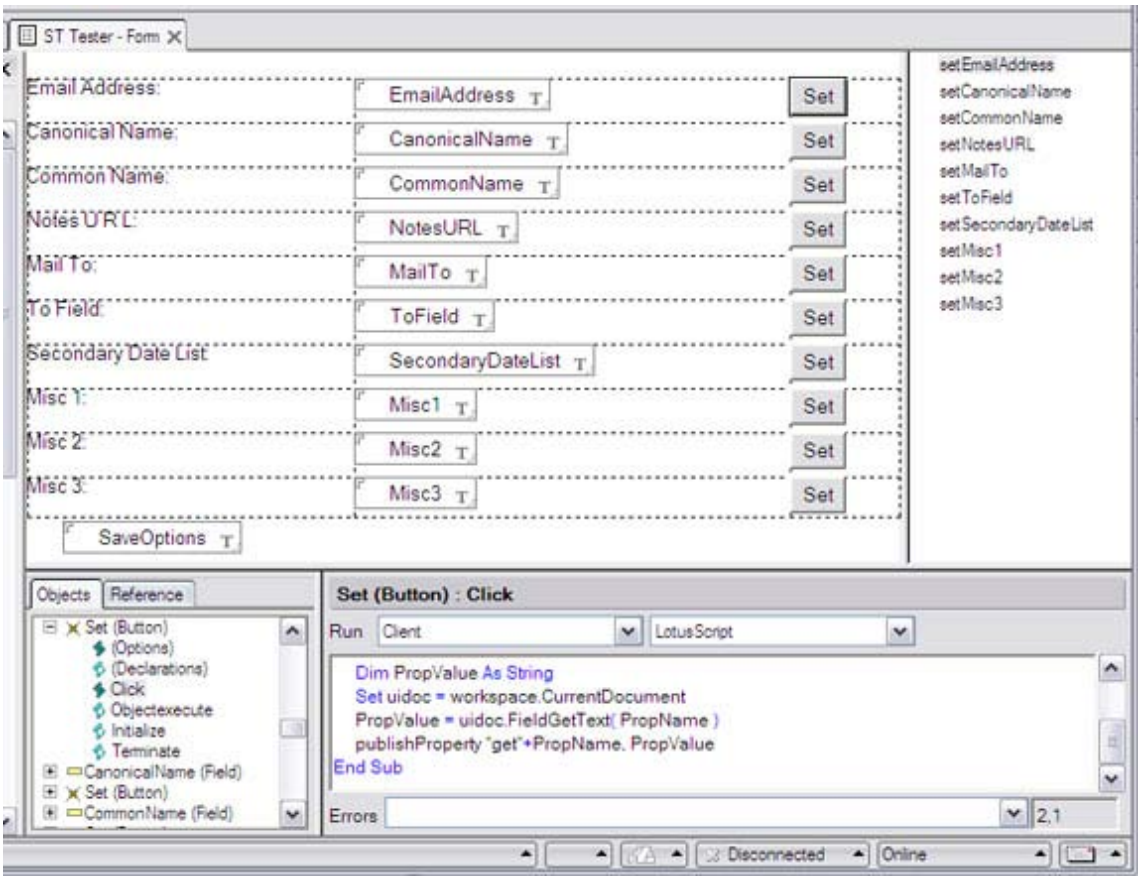
Tvorba kompozitnej aplikácie prebieha v dvoch fázach. Najprv sa vytvoria komponenty a neskôr sa poskladajú do aplikácie. Keďže sa komponenty nevytvárajú pre konkrétnych ľudí a pre špecifickú aplikáciu, je nutné predvídať všetky možné scenáre použitia. Jednotlivé komponenty sa musia správať konzistentne, predvídateľne a musia náležite spracovať aj nekorektné vstupné hodnoty. Musia mať dobrú dokumentáciu, ktorá popisuje rozhranie komponentu. Nevyhnutným predpokladom pre splnenie týchto podmienok je testovanie komponentov. Dôkladne otestovaná funkčnosť komponentu znamená menej problémov pri jeho nasadzovaní.

Prístup

Komponenty spolu nekomunikujú na úrovni programového kódu a preto na testovanie nie sú vhodné tradičné metódy. Vo vnútri komponentov je možné jednotlivé časti kódu testovať klasickým spôsobom, pre Lotus Notes pomocou knižníc LotusScriptu a pre Eclipse napríklad modulom *JUnit*. Ale samotný komponent je najlepšie testovať v prostredí kompozitnej aplikácie.

Testovací komponent

V testovacej aplikácii sú navzájom prepojené dva komponenty. Jeden je náš komponent, ktorý chceme testovať a druhý je špeciálny testovací komponent. Ten má definované vlastnosti a akcie pre všetky dátové typy a ponúka používateľské prostredie pre nastavovanie a zobrazovanie ich hodnôt. Ak sa zverejní vlastnosť v našom komponente, testovací komponent hodnotu prijme zodpovedajúcou akciou a zobrazí. Naopak, v testovacom komponente môžeme nastaviť a zverejniť hodnotu, ktorú prijme akcia definovaná v našom komponente. Takto okamžite vidíme všetky vstupné a výstupné hodnoty a zároveň máme možnosť poslať ktorejkoľvek akcii testovacie dáta. Testovanie môže byť automatizované testovacím skriptom. Týmto spôsobom je možné v krátkom čase skontrolovať všetky hraničné hodnoty a celú funkcionalitu komponentu do detailov. Ak je nutné testovať interakciu dvoch komponentov, je vhodné najprv testovať každý zvlášť a následne medzi sebou navzájom.



Obrázok 19 Testovací komponent v Domino Designeri (*Zdroj: Grant, 2008*)

Testovanie vlastností a akcií

Štandardné komponenty Lotus Notes deklarujú nasledovné menné priestory a dátové typy:

- *http://com.ibm.propertybroker.standardtypes*: emailAddress, canonicalName, commonName, NotesURL, MailTo, toField
- *http://w3.ibm.com/xmlns/ibmww/sw/datatype*: secondaryDateList
- *http://www.w3.org/2001/XMLSchema*: string

Podľa tohto zoznamu vytvoríme jednu vlastnosť a jednu akciu pre každý typ. Potom vytvoríme WSDL súbor, ktorý tieto entity popisuje. Mennou konvenciou je pridať prefix *set* pre vlastnosti a *get* pre akcie.

Plán testovania

Každý komponent by mal mať podrobnú a presnú špecifikáciu svojich vlastností a akcií. Dobrý plán testovania kontroluje všetky tieto oblasti a zároveň zisťuje, ako sa správa komponent pri neočakávaných vstupoch. V princípe sa tak dá testovanie rozdeliť na pozitívne a negatívne.

Pri pozitívnom testovaní zisťujeme, či pre korektný vstup komponent reaguje podľa očakávaní. Môže sa to týkať zadaných hodnôt alebo aj práce s používateľským prostredím, ako napríklad kliknutie na tlačidlo. Pri negatívnom testovaní zisťujeme, či komponent korektne spracuje všetky nevhodné vstupy. Skúšame vkladať hodnoty, ktoré sú prázdne, nesprávne, zle formátované, objemovo príliš veľké alebo obsahujú neštandardné znaky. Pri testovaní používateľského rozhrania skúšame klikať na tlačidlá v nesprávnom poradí.

Celkový plán musí zahŕňať aj procesy mimo rozsahu testovania jedného komponentu. Žiadne testovanie nie je dokonalé a preto je potrebné zaviesť spôsob, ako zistiť príčinu chyby, ktorá nastala v komplikovanej aplikácii. Ak je to možné, systém by mal vytvárať a uchovávať log súbory. Z nich sa dajú vybrať hodnoty, ktoré spôsobili chybu a použiť ich pri testovaní komponentov v kontrolovanom prostredí. Ak sa týmto spôsobom podarí chybu reprodukovať, nie je nutné hľadať príčiny zlyhania v celom komplikovanom systéme. Zároveň je vhodné následne na jej základe upraviť testovací skript, aby kontroloval aj chyby tohto typu.

4.8 Nasadzovanie kompozitných aplikácií

Nasadzovanie kompozitných aplikácií do produkčného prostredia je jednou zo záverečných fáz vývoja aplikácie. Neskôr nasleduje ešte udržiavanie, podpora a vylepšovanie aplikácie. Nasadzovanie je však fázou, po ktorej sa aplikácia používa priamo v ostrej prevádzke. Pohodlné nasadzovanie umožňuje architektúra klient – server. Vďaka nej nie je nutné byť fyzicky prítomný pri klientskej stanici, keď prebieha inštalácia a aktualizácia aplikácie. Tieto dve činnosti sa vykonávajú pomocou mechanizmu inštalácie takzvaných *update site* projektov.

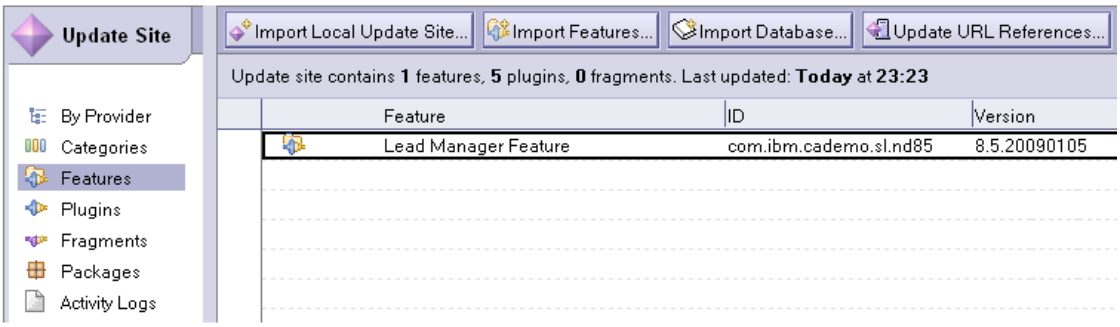
Eclipse Update Site

Je to eclipsovský projekt na distribúciu a inštaláciu v ňom uložených pluginov (komponentov kompozitnej aplikácie). Môže byť umiestnený na HTTP serveri alebo v lokálnom adresári. Lokálny adresár je vhodný pre vývoj a testovanie. Keď nastane produkčná fáza projektu, odporúča sa použiť vzdialený *update site.*

Ak je projekt update site uložený lokálne (odkaz na neho môže vyzerať ako *file:/C:/test/projekt1/site.xml*), musí byť na rovnakej ceste prítomný aj adresár *updateSite*. V momente keď je komponent pridávaný z update site, automaticky sa v súbore *ca.xml* nastaví vlastnosť *url.feature* na aktuálne URL. Potom keď sa v Lotus Notes prvýkrát spúšťa kompozitná aplikácia, nastáva proces obstarávania pluginov použitím danej URL. Všetky nové pluginy sú uložené do zdieľaných */eclipse* adresárov a je možné ich použiť ako komponenty.

Tvorba NSF Update Site

Lotus Notes umožňuje vytvoriť NSF update site projekt, ktorým sa do prostredia inštaluje dodatková funkčnosť na báze Eclipse. Projekt je vlastne štandardná NSF databáza, ktorá sa vytvára použitím pokročilej šablóny *Eclipse Update Site* (*updatesite.ntf).* Šablóna dynamicky poskytne súbor *site.xml* a následne je možné vložiť požadované pluginy.



Obrázok 20 Ukážka NSF update site *(2009)*

Obrázok 21 Ukážka pluginov jedného feature uložených v NSF update site (2009)

Oproti klasickému projektu má NSF update site niekoľko výhod:

- Databáze je možné nastaviť prístupové práva.
- Databázu je možné jednoducho distribuovať cez Domino replikáciu.
- Ľahko čitateľné notesovské dokumenty poskytujú inteligentnú analýzu celej databázy update site. Napríklad je prístup k jednotlivým pluginom vo feature zjednodušený cez hyperlinky v rámci dokumentu (Obrázok 21).
- Šablóna updatesite.ntf obsahuje nástroje na automatickú globálnu modifikáciu URL prepojení v pluginoch vo vnútri JAR súborov.

Do NSF update site sa dajú vkladať jednotlivé feature, viacero lokálnych Eclipse update site projektov a aj iný NSF update site. Počas trvania projektu tak do jedného update site môžu pribúdať pluginy v ľubovoľnom poradí.

Kompozitná aplikácia v rámci jedného NSF

Kompozitné aplikácie vo všeobecnosti spájajú komponenty z viacerých zdrojov a z viacerých fyzických umiestnení. Je však možné umiestniť všetky komponenty do jedného NSF súboru a v ňom zároveň definovať kompozitnú aplikáciu. Príkladom kompozitných aplikácií, ktoré odkazujú na komponenty uložené v tej istej databáze sú aplikácie *Notes Mail* a *Contacts* vo verzii 8. Takúto aplikáciu vytvoríme splnením nasledovných krokov:

1. Vytvoríme štandardný projekt NSF update site.
2. V Domino Designer nastavíme spúšťanie aplikácie ako Composite Application.
3. Pridáme všetky NSF komponenty a vložíme všetky projekty update site použitých komponentov na báze Eclipse.
4. Otvoríme aplikáciu v CAE a zložíme ju z komponentov pochádzajúcich z interného update site.

Namiesto odkazov v klasickej forme (*replica ID* alebo cesta k súboru) sa používajú nové Notes URL s replica ID „0000000000000000“. To oznamuje klientovi Lotus Notes 8, že má otvoriť NSF komponenty v tej istej databáze ako je umiestnený súbor ca.xml – definícia kompozitnej aplikácie. Nie je tak potrebné zakaždým meniť ca.xml, keď sa aplikácia presunie na iný server. Niekoľko ďalších špeciálnych URL, ktoré odkazujú na komponenty relatívne k používateľovi alebo k databáze sú:

- *notes:///0000000000000000* – otvára rodičovskú databázu,
- *notes:///0000000000000E00* – otvára databázu *Mail* aktuálneho používateľa,
- *notes:///0000000000000E01* – databáza *Contacts* aktuálneho používateľa.

Záverečným krokom je uloženie aplikácie na Domino server. Celú kompozitnú aplikáciu sme tak vložili do jediného NSF súboru, ktorý je prístupný na serveri cez štandardný bezpečnostný model ACL, po spustení si sám inštaluje všetky potrebné pluginy a je možné ho replikovať na lokálne offline použitie. Nasadzovanie kompozitnej aplikácie je tak výrazne zjednodušené.

5 Implementácia

5.1 OpenNTF.org

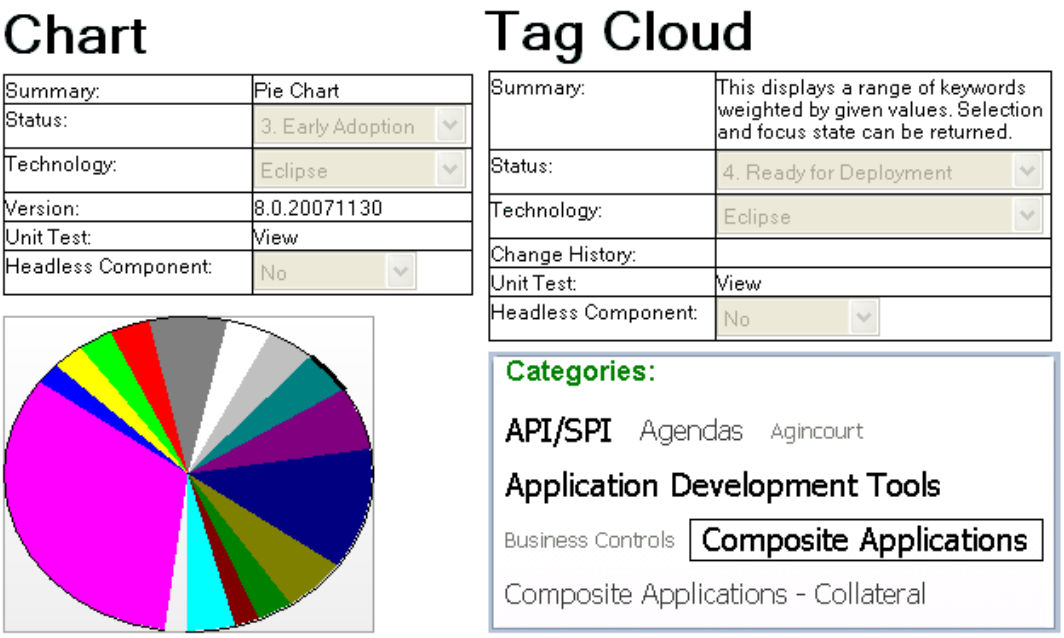
OpenNTF.org je stránka spájajúca Lotus Notes/Domino komunitu, ktorá vyvíja slobodný softvér. Ten zahŕňa kompletné projekty, šablóny, praktiky a overené postupy. Komunita má vyše 50 000 registrovaných používateľov, 200 projektov s otvoreným kódom a 10 000 stiahnutí mesačne. Na rok 2009 sa plánuje oficiálna spolupráca s IBM v súvislosti s otvorenými komponentmi a kompozitnými aplikáciami (IBM, 2009). Časť stránky sa nazýva OpenNTF Catalog. Cieľom katalógu je poskytnúť celú škálu vysokokvalitných príkladov a komponentov. Katalóg je rozdelený na niekoľko častí podľa zamerania príspevkov a tak je pohodlné vyhľadať riešenie vhodné pre konkrétnu situáciu. Príklady v katalógu musia vždy obsahovať celý zdrojový kód, podrobný popis, obrázky fungovania aplikácie a musia byť otestované komunitou. Ak je komponent zaradený do OpenNTF katalógu, vypovedá to o jeho vysokej kvalite. Pre vývojára je najlepšie vystaviť svoj projekt pod drobnohľad komunity. Ostatní používatelia môžu projekty komentovať, oznamovať chyby, navrhovať nové funkcie a projekt sa tak vyvíja rýchlejšie a stáva sa kvalitnejším.

5.2 Toolkit

Najvyzretejším príspevkom do katalógu býva takzvaný *toolkit*. Toolkit je množina súvisiacich a znovupoužiteľných entít (napríklad tried) s užitočnou a dostatočne všeobecnou funkcionalitou. Toolkit nevynucuje použitie niektorého konkrétneho dizajnu v aplikácii, iba ponúka funkcionalitu, aby nemusel vývojár písať tie isté základné triedy. Tvorba toolkitu je ťažšia ako tvorba aplikácií, pretože ak má byť toolkit prospešný, musí byť aplikovateľný vo viacerých systémoch. Zostavovateľ toolkitu navyše nikdy nevie detaily aplikácií, v ktorých budú jednotlivé časti použité. Musí sa vyhnúť závislostiam a nič špeciálne o nasadení toolkitu nepredpokladať. Štatistiky sťahovanosti na OpenNTF.org indikujú veľký záujem o knižnice komponentov.

Composite application component library

Composite Applications Component Library 2.0 je knižnica komponentov vyvinutá pre interné účely IBM a neskôr zverejnená pre všeobecné použitie. Keďže je otvorená, začínajúci vývojár môže kód podrobne skúmať a použiť ho aj pre svoje účely. Obsahuje aj niekoľko hotových vizualizačných komponentov. Sú uložené v knižnici pod spoločnou hlavičkou *Visual components*. Najbližšie k téme práce má komponent *pie chart.* Ide o jednoduchý koláčový graf so základnou funkcionalitou. Prijímané dáta sú vo formáte váženého zoznamu hodnôt. Váha určuje šírku jednotlivých častí grafu. Keď sa niektorá z častí zvolí, rozpošle sa s ňou asociovaná hodnota (Grant, 2008). Z iných komponentov spomenieme *Tag Cloud*, ktorý zobrazuje slová zo vstupu a prideľuje im rôznu veľkosť písma podľa početnosti. Spomenuté príklady nie sú veľmi pôsobivé, ale ich výhodou je, že sú to hotové komponenty a pri nasadení nie je potrebné nič meniť. Zároveň majú veľmi dobre popísané vstupné a výstupné hodnoty, ako aj odporúčania k správnemu použitiu. Projekt sa dá stiahnuť na adrese *http://www.openntf.org/Projects/pmt.nsf/ProjectLookup/Composite%20Application%20Component%20Library.*

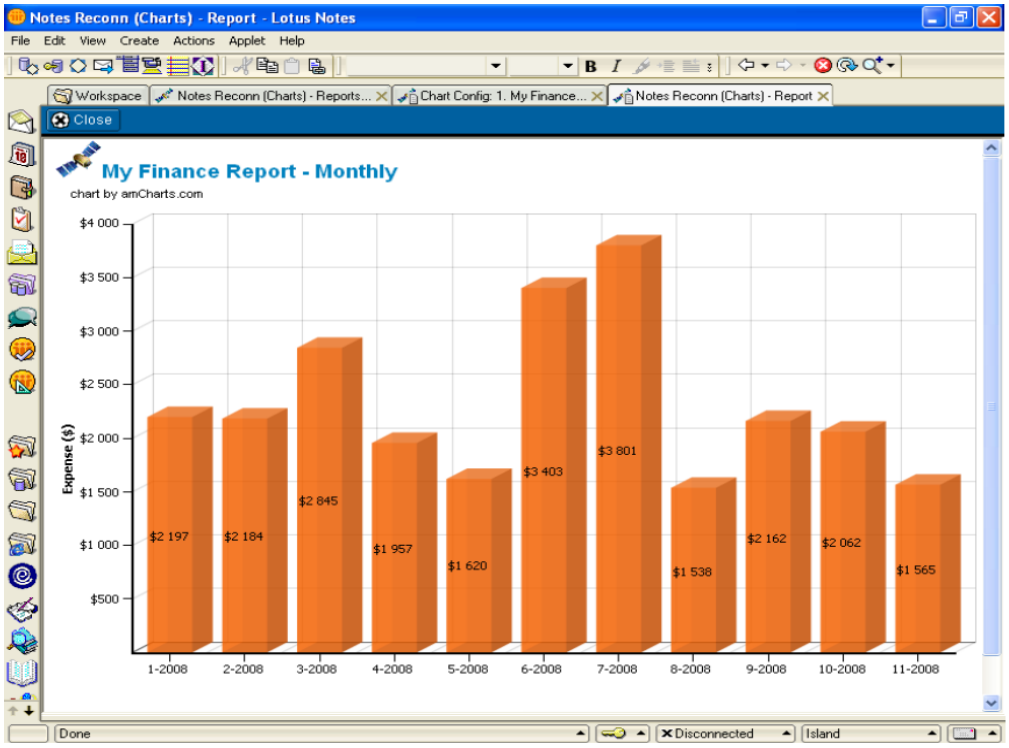


Obrázok 22 Popis a obrázky komponentov Chart a Tag Cloud *(Zdroj: IBM, 2008)*

5.3 Tvorba záverečných správ

Notes Reconn

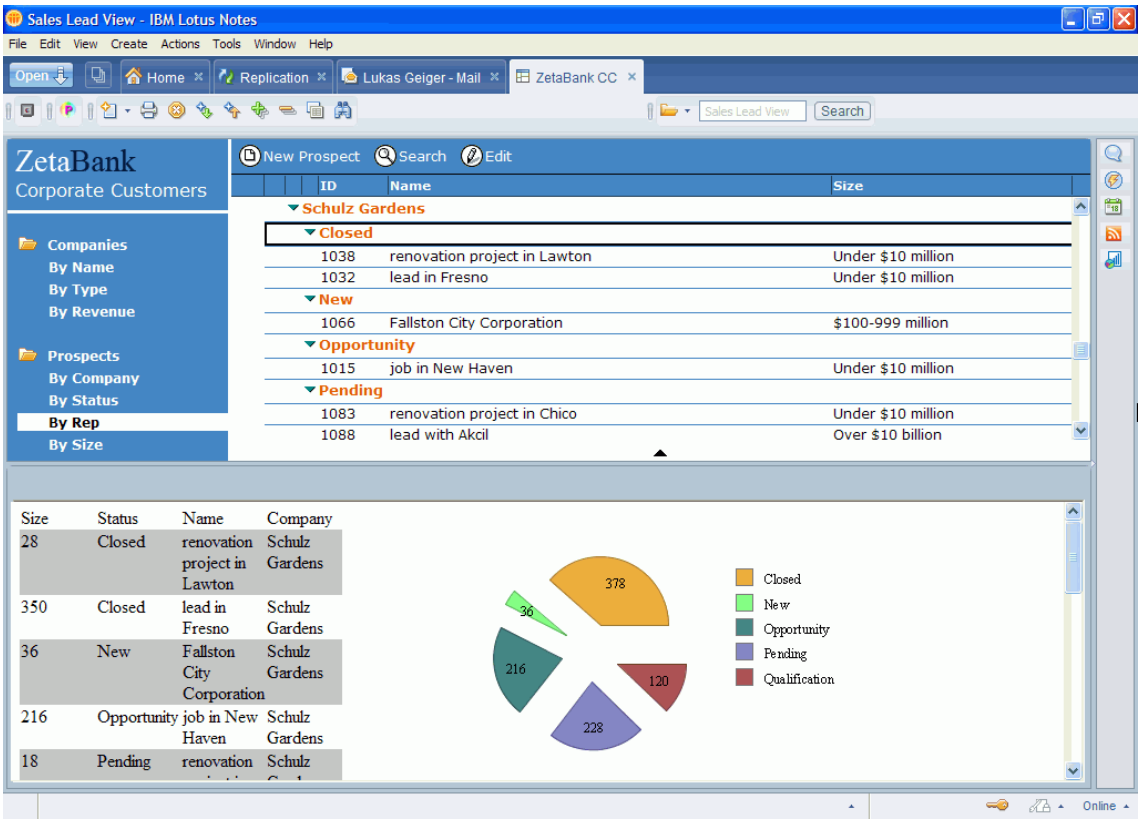
Tento projekt slúži na vytváranie záverečných správ a grafov v prostredí Lotus Notes, prístup je možný aj cez internetový prehliadač. Pre nás je zaujímavé práve vykresľovanie grafov, projekt používa dva rozličné prístupy: export do Excelu a export do XML, ktoré je následne spracované technológiou Flash. Export do Excelu je kvalitne spracovaný, ale z hľadiska bezpečnosti ho neodporúčame. Navyše sa tento princíp nedá osamostatniť do komponentu. Flash grafy sú zobrazované v integrovanom internetovom prehliadači, ktorý ale už ako komponent vystupovať môže. Ako platforma na kreslenie grafov slúži demoverzia produktu firmy *amCharts*, ktorej logo sa zobrazí na vykreslenom grafe. Ak chce vývojár použiť túto časť projektu ako komponent, musí prekonať technologickú prekážku – vytváranie a odosielanie XML súboru. V mnohom môžu pomôcť predprogramované triedy Notes Reconn. Projekt je vedený na *http://www.openntf.org/Projects/pmt.nsf/ProjectLookup/Notes%20Reconn.*



Obrázok 23 Časť záverečnej správy v Notes Reconn (2009)

Business Intelligence and Reporting Tools

Business Intelligence and Reporting Tools (BIRT) je komplexný systém na tvorbu správ *(reports)* vo forme štruktúrovaného dokumentu, ktorý zobrazuje dáta z externého zdroja, napríklad databázy (Anderson, 2006). V kontexte systémov na podporu rozhodovania je v určitých prípadoch vhodné mať vypracovanú správu na prezentáciu. BIRT je postavený na jadre Eclipse a preto je vhodným kandidátom na transformáciu na komponent. IBM ponúka na ukážku už predprogramovaný BIRT komponent. Plusom je, že je to odľahčená verzia pôvodne veľmi komplexnej aplikácie. Všetky nastavenia sú uložené v Advanced properties komponentu, vrátane špecifikovania súboru šablóny pre správu. Na druhej strane musíme vytknúť zle definované komunikačné rozhranie, ktoré znemožňuje použitie komponentu na verziách klienta vyšších ako 8.0. Komponent je k dispozícii, otázna je jeho použiteľnosť bez výrazných predchádzajúcich zmien. Stránka: *http://www.ibm.com/developerworks /blogs/page/CompApps?entry=birt_component_in_composite_application.*



Obrázok 24 Ukážka použitia komponentu BIRT *(Zdroj: IBM, 2008)*

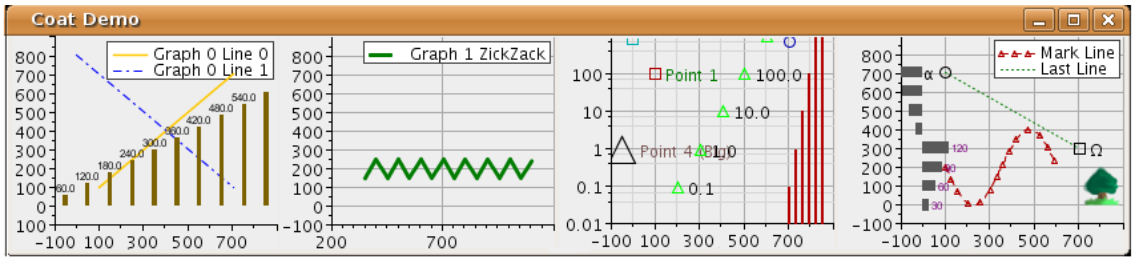
5.4 Grafy

NGraph

Projekt NGraph zobrazuje kategorizované pohľady v Lotus Notes do triviálneho grafu (chart). Tak ako projekt Notes Reconn, grafy sa zobrazujú v okne internetového prehliadača. Technológia používa DOJO 0.4 Javascript framework. Ako vstupný dátový formát používa buď XML, alebo pre človeka ľahko čitateľný textový formát JSON. Práve použitie jednoduchého textového formátu na výmenu informácií robí z inak priemerného projektu zaujímavú voľbu pre tvorbu komponentov, kde je táto zrozumiteľnosť v tesnej korelácii so znovupoužiteľnosťou. Projekt je vedený na stránke *http://www.openntf.org/Projects/pmt.nsf/ProjectLookup/NGraph*.

COAT

Projekt COAT je kompaktný vizualizačný balík na real-time vykresľovanie 2D grafov (charts) napísaný v Jave. Jeho hlavná nevýhoda – nedostatok pokročilých vizualizačných metód paradoxne zabezpečuje existenciu jeho hlavnej výhody – real-time zobrazovanie. Všetky grafy sú vlastne iba jednoduché body, čiary a stĺpce, čo znamená ich celkovú nenáročnosť na výpočtový výkon procesora pri aktualizácii. V kompozitnej aplikácii by COAT komponent slúžil na zobrazenie aktuálneho kontextu v jednoduchej, ale dynamickej forme a pri tlači by sa použil iný, sofistikovanejší systém na tvorbu statických grafov. Zdroj: *http://coat.sourceforge.net.*

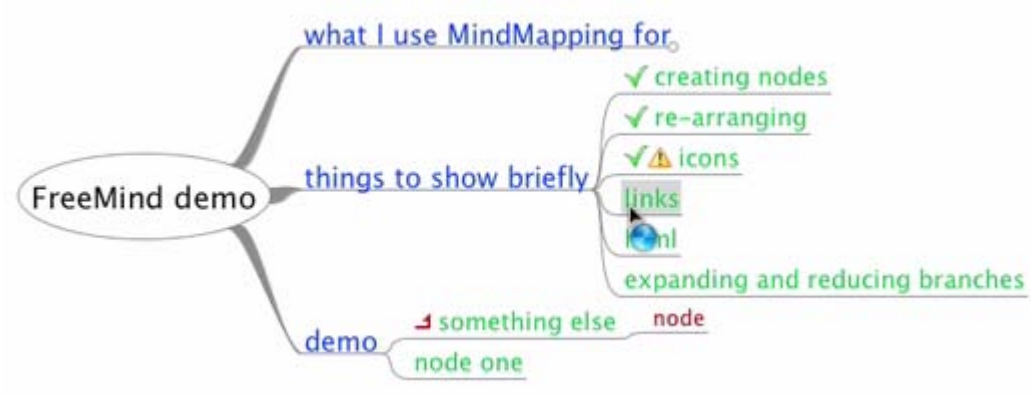


Obrázok 25 Ukážka grafov systému COAT *(Zdroj: coat.sourceforge.net, 2009)*

5.5 Myšlienkové mapy

FreeMind

FreeMind je softvér napísaný v Jave, ktorý slúži na vytváranie myšlienkových máp *(mind maps).* Je to silný a ľahko použiteľný nástroj s dobrým ovládaním. Ak by sme ale do komponentu chceli vybrať iba zobrazovacie jadro, vyžadovalo by si to veľa programátorských prác. Tento príklad slúži skôr na ukážku výpovednej sily tohto modelu zobrazovania. Stránka: *http://freemind.sourceforge.net.*



Obrázok 26 Ukážka projektu Freemind *(Zdroj: freemind.sourceforge.net, 2009)*

HyperTree Java Library

Projekt *HyperTree Java Library* poskytuje triedy na kreslenie vizualizácii hyperbolických stromov. Projekt je napísaný v Jave a preto ľahšie použiteľný v systémoch založených na Eclipse. Stránka: *http://hypertree.sourceforge.net.*

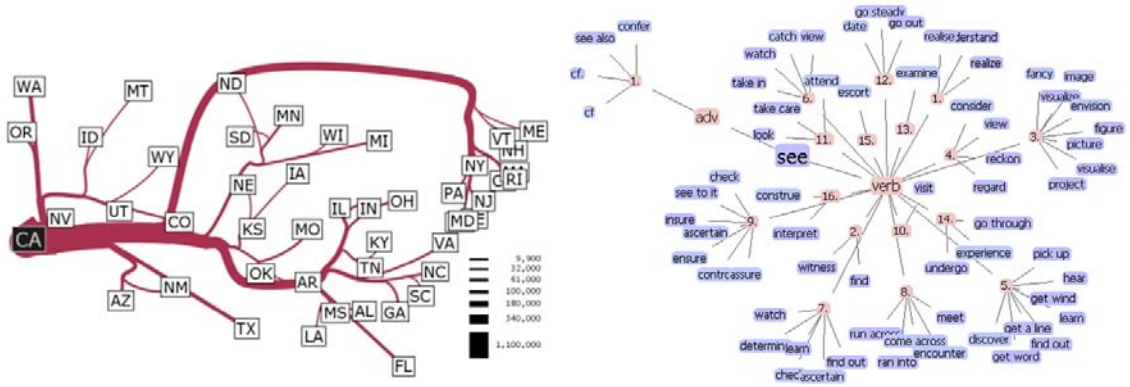
5.6 Všeobecné projekty

JGraph

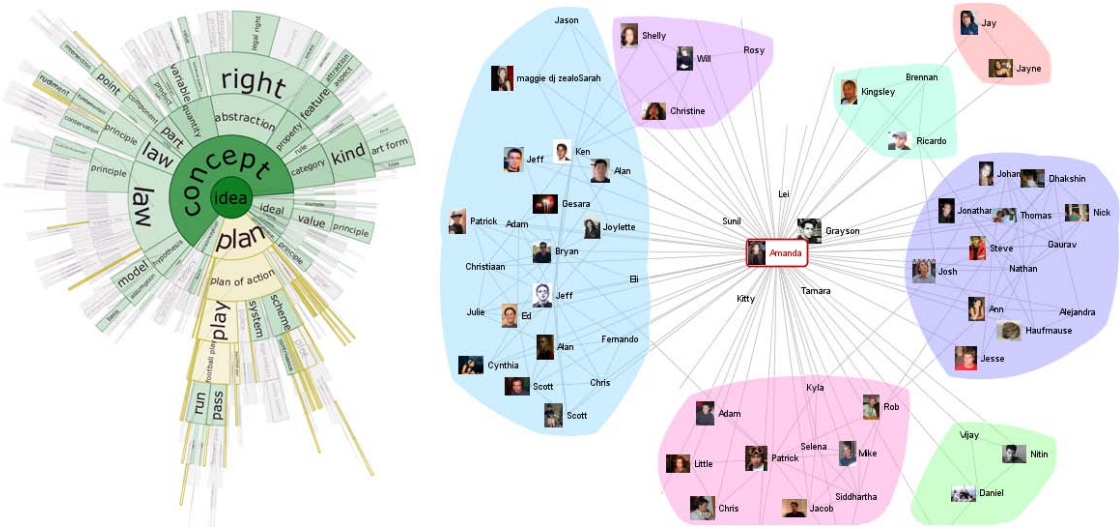
JGraph je plnohodnotná otvorená knižnica vizualizácií grafov *(graph)* napísaná v Jave. Projekt má integrované API, takže je pripravený na externý prístup. Ako komponent je možné nasadiť vizualizáciu pomocou JGraph na takmer ľubovoľný účel. Stránka open source projektu je *http://www.jgraph.com/jgraph.html.*

Prefuse

Prefuse je rozšíriteľný softvérový framework na tvorbu interaktívnych vizualizácií dát s použitím programovacieho jazyka Java (konkrétne využíva knižnicu Java 2D). Je možné ho použiť na tvorbu samostatných aplikácií, ale aj na vývoj komponentov pre väčšie systémy. Prefuse dáta modeluje, vizualizuje a umožňuje následnú interakciu s nimi. Devízou frameworku je veľké množstvo implementácií a príkladov, ktoré je možné študovať. Hlavná stránka projektu je *http://prefuse.org.*



Obrázok 27 Príklady implementácie Prefuse. Vľavo projekt Flow Map Layout *(Zdroj: http://graphics.stanford.edu/papers/flow_map_layout, 2009)*, vpravo NearWord Visual Synonyms *(Zdroj: http://www.intsysr.com/nearword.htm, 2009).*



Obrázok 28 Ďalšie ukážky Prefuse: DocuBurst a Vizster *(Zdroj: prefuse.org, 2009).*

6 Záver

Dostupnosť včasných a presných informácií je dôležitá pri správnych rozhodnutiach. Techniky vizualizácie napomáhajú v procese rozhodovania, pretože umožňujú skoré odhalenie nastupujúceho vývoja. Je to vďaka schopnostiam ľudského vizuálneho vnímania, ktorý ľahko rozpoznáva známe vzorce aj vo veľkom množstve vhodne zobrazených dát. Kontextová vizualizácia má veľké uplatnenie v podnikovom prostredí. Technologický pokrok ju dovoľuje pridať do existujúcich aplikácií, v ktorých sú následne kvantitatívne a kvalitatívne parametre údajov prehľadnejšie. Jedným z takýchto systémov je IBM Lotus Notes. Prináša model kompozitných aplikácií, kde jeden komponent uchováva dáta a druhý ich môže vizualizovať. Tematika kompozitných aplikácií úzko súvisí s architektúrou orientovanou na služby, ktorá propaguje myšlienky všeobecnosti a znovupoužiteľnosti. Komponent vyvinutý podľa zásad spomenutej architektúry je možné bezo zmeny nasadiť do mnohých systémov na spoločnej platforme.

V súlade s cieľmi diplomovej práce sme preskúmali uvedenú problematiku. Rozoberáme systémy na podporu rozhodovania, tému vizualizácie a detailne popisujeme oblasť kompozitných aplikácií na platforme Lotus Notes. Najdôležitejším prínosom práce je podrobný návod na tvorbu komponentov. Po jeho osvojení je jednoduché vyvíjať kvalitné komponenty pre platformu Lotus Notes. Návod pokrýva všetky oblasti vývoja komponentov, od začiatočného plánovania, cez dizajn až po tvorbu NSF a Eclipse komponentov. Uvádzame aj popis skladania komponentov do kompozitných aplikácií, ich testovanie a nasadzovanie do produkčného prostredia.

V celej práci rezonuje myšlienka znovu používať existujúce riešenia. Súčasťou práce je prehľad projektov, ktoré graficky zobrazujú dáta na platforme Lotus Notes a prehľad zaujímavých externých vizualizačných riešení programovaných v jazyku Java. Každý projekt sme analyzovali a uvádzame, ako je vhodné ho transformovať na vizualizačný komponent v systéme na podporu rozhodovania. Cieľom

prehľadu nie je podať vyčerpávajúci výpočet vizualizačných projektov, skôr uviesť zaujímavé existujúce riešenia. Vyberáme otvorené projekty, aby vývojár nemusel začínať úplne od začiatku, ale mohol použiť časti hotového kódu.

Kompozitné aplikácie na platforme Lotus Notes sú podporované iba od augusta 2007 a komunita, ktorá sa nimi zaoberá, nie je veľmi početná. Okrem propagačných materiálov spoločnosti IBM neexistuje k téme žiadna literatúra napísaná v slovenskom jazyku. Ilustruje to predovšetkým zastúpenie zdrojov v anglickom jazyku v zozname použitej literatúry. Podarilo sa nám tak splniť sekundárny cieľ práce – zmapovať a sprostredkovať problematiku aj pre slovenských vývojárov.

Koncept práce naznačuje otvorenosť ďalšiemu výskumu a vývoju v spracovávanej oblasti. Najväčšie rezervy vidíme v obmedzení formátu vymieňaných dát na jednoduchý reťazec. Dopracovať by sa dal komponent typu XML exchange na riadenie výmeny informácií medzi komponentmi vo forme štruktúrovaných XML dokumentov. Toto obmedzenie by sa dalo vyriešiť aj zásahmi priamo do jadra systému, no tento typ riešenia je v rukách vývojárov Lotus Notes.

Zoznam použitej literatúry

Abdollah, Tina. 2007. Building a successful SOA project. *developerWorks.* [Online] developerWorks, 2007. [Dátum: 19. 3 2009.] http://www.ibm.com/developerworks/library/ar-buildsoa/.

Anderson, Tyler. 2006. Extract information from databases using BIRT and. *developerWorks.* [Online] 2006. [Dátum: 12. 12 2008.] https://www6.software.ibm.com/developerworks/education/os-ecl-birt/index.html.

Auriemma, Stephen. 2008. Get started creating composite applications with this hands-on exercise. *The VIEW.* Júl/August 2008, Zv. XIV, 4, s. 11-30.

—. 2008. What you need to know about Notes 8 composite applications. *The VIEW.* Júl/August 2008, Zv. XIV, 4, s. 3-10.

Behrends, Eric, Harwood, Robert F a Drschiwiski, Deanna. 2008. Eclipse fundametals. *IBM Composite Applications wiki.* [Online] IBM, 2008. [Dátum: 3. 11 2008.] http://www-10.lotus.com/ldd/compappwiki.nsf/dx/eclipse-fundamentals.

Benz, Brian a Oliver, Rocky. 2005. *Mistrovství v programování Lotus Notes a Domino.* Brno : CP Books, 2005. s. 966. ISBN 80-251-0750-7.

Carlson, Pierre a Carter, Keith. 2008. Build Java components for Notes 8 composite applications with IBM Lotus Expeditor Toolkit 6.1.2. *The VIEW.* 2008, Zv. XIV, 6, s. 9-38.

—. 2007. IBM Lotus Expeditor Toolkit: Get started building applications for Expeditor 6.1.1 and Sametime 7.5.1. *The VIEW.* September/Október 2007, Zv. XIII, 5, s. 31-46.

Carrier, Nicole, a iní. 2008. *The business case for enterprise mashups.* [Dokument] Somers : IBM Corporation Software Group, 2008. EPW14002-USEN-00.

Claus, Volker a Schwill, Andreas. 1991. *Lexikón informatiky.* Bratislava : SPN, 1991. s. 544. ISBN 80-08-00755-9.

Darmawan, Budi, Rintoul, David a Anglin, Howard. 2008. *Managing Composite Applications: An Operator's View.* Austin : International Technical Support Organization, 2008. REDP-4319-00.

Educause. 2007. *7 things you should know about... Data Visualization.* [Dokument] Boulder : Educause, 2007.

Eppler, M.J. a Burkhard, R.A. 2005. Knowledge Visualization in Encyclopedia of Knowledge Management. *Idea Group.* [Online] 2005.

Friendly, M. a Denis, D. 2006. Milestones in the History of Thematic Cartography, Statistical Graphics and Data Visualisation. *math.yorku.ca.* [Online] 2006. www.math.yorku.ca/SCS/Gallery/milestone/.

Gamma, Erich, a iní. 1994. *Design Patterns, Elements of Reusable Object-Oriented Software.* s.l. : Addison Wesley, 1994. s. 416. ISBN 0-201-63361-2.

Grant, Jo. 2008. Composite applications component library v2.0. *developerWorks.* [Online] developerWorks, 2008. [Dátum: 2. December 2008.] http://www.ibm.com/developerworks/lotus/library/notes8-library/.

—. 2008. Moving to Notes 8 - What every Notes professional must know about the Eclipse Rich Client Platform. *The VIEW.* Január/Február 2008, Zv. XIV, 1, s. 3-26.

Gucer, Vasfi, a iní. 1999. *Using Tivoli Decision Support Guides.* Austin : International Technical Support Organization, 1999. SG24-5506-00.

Harris, Robert L. 1999. *Information Graphics: A comprehensive Illustrated Reference.* New York a Oxford : Oxford University Press, 1999. ISBN 0-19-5135326.

Harwood, Robert F. a Drschiwiski, Deanna. 2009. Business value. *IBM Composite Applications wiki.* [Online] IBM, 2009. [Dátum: 6. 4 2009.] http://www-10.lotus.com/ldd/compappwiki.nsf/dx/business-value.

IBM. 2009. Lotus Domino and Notes Information Center. *Publib.boulder.ibm.com.* [Online] IBM, 2009. [Dátum: 6. Január 2009.] http://publib.boulder.ibm.com/infocenter/domhelp/v8r0/index.jsp.

Lee-Loy, Sheldon a Fang, Jane. 2008. Advanced charting in BIRT. *developerWorks.* [Online] 2008. [Dátum: 15. 12 2008.] https://www6.software.ibm.com/developerworks/education/os-eclipse-birt-advanced/index.html.

Lengler, Ralph a Eppler, Martin J. 2006. *Towards A Periodic Table of Visualization Methods for Management.* [Dokument] Lugano : Institute of Corporate Communication, University of Lugano, 2006.

Lotus Software. 2008. *Composite Applications in Notes - Benefits and Technical Overview.* [Dokument] Austin : IBM Corporation, 2008.

—. 2007. *IBM Lotus Notes and Domino software: building on the evolution of people-centric applications.* [Dokument] Cambridge : IBM Corporation, 2007. LOB10852-USEN-00.

Mindzora, Joanne a Brent, Karen. 2007. *IBM Lotus Notes and Domino 8 Reviewer's Guide.* Austin : International Technical Support Organization, 2007.

Moravec, Luboš. 2003. *Lotus Notes - Uživatelská příručka.* Brno : Computer Press, 2003. s. 250. ISBN 80-251-0027-8.

National Forum on Education Statistics. 2006. *Forum Guide to Decision Support Systems: A Resource for Educators.* Washington, DC : U.S. Department of Education, 2006. NFES 2006–807.

Paleta, Petr. 2003. *Co programátory ve škole neučí.* Brno : Computer Press, 2003. s. 337. ISBN 80-251-0073-1.

Peh, Diana, Hannemann, Alethea a Hague, Nola. 2006. *BIRT: A Field Guide to Reporting.* Boston : Addison-Wesley, 2006. ISBN 0-321-44259-8.

PistolStar, Inc. 2009. *Myths & Realities of Lotus Notes and Domino 8.5 Security.* [Dokument] Amherst : PistolStar, Inc., 2009.

Rippen, Michael. 2005. *Decision Support Systems.* [Dokument] Oxon : Tessella Support Services plc, 2005.

Rodriguez, Juan R., a iní. 2007. *Building Composite Applications.* Poughkeepsie : International Technical Support Organization, 2007. ISBN 0738489417.

Schneider, Robert, a iní. 2007. *Building Composite Applications in Lotus Expeditor V6.1.* Austin : International Technical Support Organization, 2007. s. 134 . REDP-4241-00.

Speed, Tim, a iní. 2007. *Lotus Notes Domino 8 Upgrader's Guide.* Birmingham : Publishing Ltd., 2007. ISBN 978-1-847192-74-5.

Tomlyn, Craig, Rasid, Abdul a Larsen, Yan. 2007. Hints and tips for using the Business Intelligence. *developerWorks.* [Online] 2007. [Dátum: 12. 12 2008.] http://www.ibm.com/developerworks/data/library/techarticle/dm-0708.

Výboh, M. 2003. *Vizualizácia algoritmov.* Bratislava, 2003. s. 59.

Weathersby, Jason, a iní. 2006. *Integrating and Extending BIRT.* Boston : Addison-Wesley, 2006. ISBN 0-321-44385-3.

Wikipedia. 2009. Lotus Notes. *en.wikipedia.org.* [Online] Wikipedia, 2009. [Dátum: 12. Február 2009.] http://en.wikipedia.org/wiki/Lotus_notes.

—. 2009. Model–view–controller. *en.wikipedia.org.* [Online] Wikipedia, 2009. [Dátum: 16. 2 2009.] http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller.

Wolpert, Craig a Grant, Jo. 2008. Design patterns. *developerWorks.* [Online] developerWorks, 2008. [Dátum: 5. Január 2009.] http://www.ibm.com/developerworks/lotus/library/notes8-patterns/?S_TACT=105AGX13&S_CMP=ART.

—. 2008. Designing composite applications: Component design. *developerWorks.* [Online] developerWorks, 2008. [Dátum: 15. 12 2008.] http://www.ibm.com/developerworks/lotus/library/notes8-component-design/.

—. 2008. Developing composite applications: Composite application assembly, part 1. *developerWorks.* [Online] developerWorks, 2008. [Dátum: 17. Január 2009.] http://www.ibm.com/developerworks/lotus/library/notes8-assembly-pt1/?S_TACT=105AGX13&S_CMP=ART.

—. 2008. Developing composite applications: Composite application assembly, part 2. *developerWorks.* [Online] developerWorks, 2008. [Dátum: 22. Január 2009.] http://www.ibm.com/developerworks/lotus/library/notes8-assembly-pt2/?S_TACT=105AGX13&S_CMP=ART.

—. 2008. IBM Lotus Notes components. *developerWorks.* [Online] developerWorks, 2008. [Dátum: 14. Január 2009.] http://www.ibm.com/developerworks/lotus/library/notes8-components/?S_TACT=105AGX13&S_CMP=ART.

—. 2008. Unit testing. *developerWorks.* [Online] developerWorks, 2008. [Dátum: 7. Január 2009.] http://www.ibm.com/developerworks/lotus/library/notes8-unit-test/?S_TACT=105AGX13&S_CMP=ART.

Príloha

Periodic Table of Visualization Methods

http://www.visual-literacy.org/periodic_table/periodic_table.html