UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

DIPLOMOVÁ PRÁCA

2006 Alexandra Baltová

Integrácia aplikácií prostredníctvom výmeny správ

DIPLOMOVÁ PRÁCA

Alexandra Baltová

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

Študijný odbor: Informatika
Vedúci diplomovej práce: Mgr. Pavol Mederly

Bratislava 2006

Čestné prehlásenie:

Čestne prehlasujem**,** že predkladanú prácu som vypracovala samostatne pod odborným vedením školiteľa, len s použitím uvedenej literatúry.

.....

Alexandra Baltová

Srdečne ďakujem Mgr. Pavlovi Mederlymu za starostlivé a obetavé vedenie diplomovej práce, za cenné rady a pripomienky pri jej vypracovávaní.
Zároveň ďakujem spolužiakovi Gustávovi Pálošovi za pomoc pri implementovaní JMS.

Abstrakt

Autor:	Alexandra Baltová
Názov práce:	Integrácia aplikácií prostredníctvom výmeny správ
Škola:	Univerzita Komenského, Bratislava
Fakulta:	Fakulta Matematiky, Fyziky a Informatiky
Vedúci diplomovej práce:	Mgr. Pavol Mederly
Rok:	2006
Počet strán:	74

Predmetom tejto práce je analýza a porovnanie troch vybraných produktov, ktoré je možné použiť na realizovanie integrácie aplikácií. Ide o nástroje s otvoreným kódom, ktoré integrujú aplikácie prostredníctvom výmeny správ. Produkty s otvoreným kódom boli zvolené predovšetkým kvôli ich ľahkej dostupnosti a širším možnostiam analýzy.

Práca pozostáva z popisu integrácie prostredníctvom výmeny správ, analýzy konkrétnych produktov a ich vzájomného porovnania. Tieto výsledky sú zároveň ilustrované riešením modelového príkladu v prílohe práce.

Informácie uvedené v práci môžu slúžiť ako podklad pre rozhodnutie o výbere nástroja založeného na výmene správ pre konkrétny projekt integrácie aplikácií. Môžu byť zároveň inšpiráciou pre realizáciu integračného riešenia založeného na vlastnom nástroji.

OBSAH

ZOZNAM OBRÁZKOV

ZOZNAM TABULIEK

Úvod

Integrácia podnikových aplikácií (Enterprise Application Integration) je kombinácia procesov, štandardov, softvérových a hardverových prostriedkov, ktorá realizuje prepojenie podnikových aplikácii tak, aby fungovali ako jeden celok. V tejto práci sa snažíme analyzovať a popísať vybrané riešenia v tejto oblasti. Zamerali sme sa na produkty s otvoreným kódom (open source), ktoré realizujú integráciu aplikácií prostredníctvom výmeny správ. Cieľom práce bolo preskúmať vlastnosti týchto produktov, vzájomne ich porovnať a ilustrovať ich praktické využitie na jednoduchom príklade.

Open source produkty sme zvolili z niekoľkých dôvodov. Ide o produkty, ktoré sú poskytované vrátane zdrojového kódu, čo nám uľahčilo preskúmanie niektorých technických detailov. V porovnaní s komerčnými produktami sú to spravidla jednoduchšie a menej komplexné riešenia. Tieto riešenia sú poskytované zadarmo a sú voľne šíriteľné. Je možné ich modifikovať, prispôsobovať, prípadne použiť len niektorú ich funkčnú časť. V prípade, že by niektorá organizácia mala záujem o použitie týchto produktov v praxi, má možnosť tak urobiť aj bez veľkých investičných nákladov typických pre komerčné riešenia v oblasti EAI.

Pri práci sme vychádzali spočiatku zo štúdia odborných článkov. EAI je veľmi často diskutovanou témou. Našli sme množstvo zaujímavých článkov, ktoré sa venovali tomuto pojmu po opisnej aj názorovej stránke. To znamená, že zahŕňali detailné vysvetlenie tohto pojmu, aj konkrétne skúsenosti pri aplikovaní tohto riešenia. O riešeniach EAI s otvoreným kódom, prípadne o relevantnosti týchto riešení v porovnaní s komerčnými riešeniami sme však našli veľmi málo informácií. Veľmi zaujímavým zdrojom bol projekt OpenEAI [3]. Ide o metodiku, ktorá poskytuje množstvo návodov a inšpirácií. Táto metodika sa nezaoberá len konkrétnymi problémami, ale aj procesom plánovania a realizácie integrácie.

Informácie o skúmaných produktoch sme získali predovšetkým z ich dokumentácie, ktorá je prístupná na internete, ako aj zo zdrojových kódov týchto produktov.

Samotná práca je rozdelená do nasledujúcich častí.

V kapitole 1 definujeme integráciu systémov použitím komunikačného systému, ktorý pre externé systémy slúži ako sprostredkovateľ komunikácie. Rozoberáme problematiku pripojenia systémov ku komunikačnému systému, transformácie údajov a smerovania týchto údajov. V závere kapitoly na abstraktnej úrovni uvádzame príklady riešenia integrovania systémov.

V kapitole 2 uvádzame popisy skúmaných produktov, kde sa sústreďujeme predovšetkým na spomínané tri témy (konektivita, transformácie, smerovanie).

V kapitole 3 uvádzame porovnanie týchto produktov, kde sa sústreďujeme predovšetkým na rôznorodosť prístupov k riešeniu konkrétnych problémov.

V prílohe sa nachádzajú konkrétne riešenia modelového príkladu z prvej kapitoly, ktoré sme vytvorili použitím skúmaných produktov.

Táto práca nemá za úlohu reprodukovať informácie uvedené v dokumentácii k jednotlivým produktom. Jej hlavný prínos spočíva v analýze produktov, ktorá zahŕňa nájdenie spoločných čŕt (konkrétne napríklad v oblasti vnútornej architektúry) a pochopenie a popísanie rozdielov v prístupe k riešeniu konkrétnych problémov.

1. Integrácia použitím výmeny správ

V tejto práci sa zaoberáme produktmi, ktoré umožňujú aplikáciám komunikáciu prostredníctvom výmeny správ a to konkrétne produktmi OpenAdaptor, Proteus a xBus.

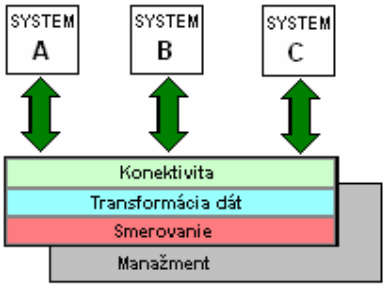
Tieto produkty sme vybrali po prehliadnutí viacerých nástrojov pre integrovanie aplikácií. Tento zoznam sme zredukovali na nástroje, ktoré majú isté spoločné črty a sú v potrebnej mieren zdokumentované, aby bolo možné ich porovnanie. Zároveň sme vylúčili nástroje, ktoré sú len v počiatočnom štáduňiu vývoja a nebola k dispozícii verzia, ktorú by bolo možné použiť. Spoločnými črtami, na základe ktorých boli produkty vyberané, sú:

- podobný prístup k integrácii aplikácií – integrácia prostredníctvom výmeny správ,
- podobné funkcie poskytované integračným riešením – konektivita, transformácie a smerovanie údajov,
- podobný prístup pri vytváraní integračného riešenia – komunikačný systém vytvorený z množiny hotových komponentov,
- prístup k zdrojovému kódu týchto nástrojov.

Integráciu prostredníctvom výmeny správ si môžeme predstaviť nasledovne. Medzi systémy, ktoré chcú vzájomne komunikovať (ďalej externé systémy), je vložený komunikačný systém, ktorý dokáže prijať správu od jedného externého systému a doručiť ju inému externému systému alebo systémom. Tento komunikačný systém poskytuje funkcie týkajúce sa:

- prístupu k údajom, ktoré poskytuje nejaký externý systém,
- transformovania týchto údajov do podoby, ktorá je akceptovateľná iným externým systémom,
- presmerovania na výstup pre tento systém,
- prípadne spracovania výnimiek a zaznamenávania aktivity (log).

Štruktúru integračného riešenia použitím výmeny správ ilustruje obrázok 1. Systémy A – C sú externé systémy, ktoré pripájame k nášmu komunikačnému systému. Samotný komunikačný systém je rozdelený do viacerých vrstiev. Na pripojenie externých systémov slúžia konektory, ktoré sú umiestnené vo vrstve „konektivita". Druhá vrstva poskytuje funkcie pre transformovanie údajov. Tretia vrstva poskytuje funkcie pre smerovanie týchto údajov.



Obrázok 1: štruktúra integračného riešenia použitím výmeny správ

1.1. Konektivita

V prvej časti sa budeme venovať spôsobu, akým je možné externý systém pripojiť k nášmu komunikačnému systému.

Väčšina externých systémov, ktoré pripájame nebolo naprogramovaných tak, aby vedeli priamo komunikovať s komunikačným systémom, ktorý realizuje výmenu správ. Napriek tomu však poskytujú údaje alebo funkcie, ktoré by mohli iné systémy využívať. Údaje môžu ukladať do databáz, prípadne do súborového systému. Funkcie môžu poskytovať cez vhodné rozhranie. Pripojenie konkrétneho systému ku komunikačnému systému teda závisí od spôsobu, akým je samotný systém schopný poskytovať svoje údaje, alebo funkcie. Pripojenie ku komunikačnému systému je realizované pomocou konektorov, ktoré abstrahujú komunikačné detaily jednotlivých systémov. Príkladom tejto abstrakcie je skúmaný produkt xBus. Práve jedna jeho vrstva poskytuje funkcie pre pripojenie systémov, ostatným vrstvám sú technické detaily tejto komunikácie skryté.

1.1.1. Konektor

Konektor je komponent, ktorý implementuje komunikáciu cez určité rozhranie, respektíve protokol. Mnoho systémov komunikuje rovnakým spôsobom, preto je užitočné vytvoriť štandardné konektory, ktoré je možné opakovane použiť na pripojenie systémov ku komunikačnému systému. Skúmané produkty poskytujú konektory pre niektoré konkrétne rozhrania, či protokoly. Všetky skúmané produkty sú poskytované s otvoreným zdrojovým kódom, ktorý je možné modifikovať a tak poskytujú možnosť doprogramovania vlastných konektorov.

1.1.2. Rozdelenie externých systémov

Pozrieme sa na vybrané protokoly a rozhrania, s ktorými vie komunikovať aspoň jeden zo skúmaných produktov. Detaily týkajúce sa konkrétnych produktov opíšeme neskôr.

Na začiatok sa bližšie pozrieme na externé systémy. Tieto systémy rozdelíme podľa dvoch kritérií. Prvé kritérium ich rozdeľuje podľa spôsobu, ako pristupujeme k ich údajom. Druhé kritérium ich rozdeľuje podľa toho, akú reprezentáciu údajov používame pri komunikácii s týmito systémami.

Podľa spôsobu, akým k údajom pristupujeme, môžme externé systémy rozdeliť do dvoch kategórií. Do prvej kategórie patria systémy, ktoré sú zdrojom udalostí. Udalosťou môže byť napríklad umiestnenie správy do sledovaného radu (queue). Do druhej kategórie patria takzvané „Polled Sources“. „Polled Source“ je zdroj údajov, ktorý nie je zdrojom udalostí. Príkladom je súbor umiestnený v rámci súborového systému alebo v rámci FTP serveru, elektronická pošta umiestnená v poštovej schránke, prípadne záznam v databáze. Príklad „Polled Source“ môžeme ilustrovať na súbore, ktorý je umiestnený v súborovom systéme. V prípade, že nejaký systém uloží na disk súbor, komunikačný systém sa o tom nedozvie. Jediná možnosť je pozrieť sa na disk, či tam hľadaný súbor je. Z tohto dôvodu je vhodné pozerať sa na disk opakovane v určitom intervale.

Podľa reprezentácie údajov môžeme externé systémy rozdeliť na systémy, ktorých údaje prijímame, respektíve posielame ako sekvenciu bytov, prípadne znakov a systémy, ktorých údaje prijímame, respektíve posielame v špecifickej štruktúre. Príklad sekvencie bytov, prípadne znakov je súbor v súborovom systéme, na serveri FTP, mailová správa a podobne. Príklad systémov, ktorých údaje prijímame, respektíve posielame v špecifickej štruktúre je databáza, alebo adresárová služba. Tieto systémy väčšinou používajú aj špecifické dátové typy. Preto pri ich používaní potrebujeme namapovať tieto dátové typy na dátové typy vlastné pre náš komunikačný systém. Mapovanie dátových typov môže byť realizované priamo použitím metód, ktoré poskytuje rozhranie príslušného ovládača.

Ako príklad uvedieme niektoré dátové typy používané databázou MySQL a alternatívy týchto typov v programovacom jazyku Java[1]:

MySQL	Java
REAL	double
VARCHAR	String
TEXT	String
BIT	Boolean
DATE	java.sql.Date
TIMESTAMP	java.sql.Timestamp

Tabuľka 1: príklad korešpondujúcich dátových typov

1.1.3. Externé systémy

1.1.3.1. Súborový systém (File System)

Súborový systém slúži na uchovávanie súborov. Rozlišujeme binárne a textové súbory. Súbor je vo všeobecnosti sekvenčná množina blokov alebo znakov. Sekvenčne je spravidla realizované aj jeho čítanie, či zapisovanie.

[1] Rozhranie JDBC poskytuje metódy pre konvertovanie výsledkov príkazu SELECT na dátové typy programovacieho jazyka Java, ako aj konvertovanie dátových typov programovacieho jazyka Java na parametre SQL príkazov.

Potrebné informácie pre konektor sú vo všeobecnosti:

- Umiestnenie súboru v rámci adresárovej štruktúry
- Názov súboru

Meno súboru môže byť uvedené presným názvom alebo vzorom mena súboru, ktorý je vyjadrený napríklad hviezdičkovou konvenciou.

Typ údajov:

V závislosti od použitého riešenia môžeme súbor načítať ako sekvenciu údajov do jedného reťazca, prípadne ho pri načítavaní štrukturovať. Napríklad OpenAdaptor načítava údaje do vlastnej dátovej štruktúry. Pri čítaní textového súboru používateľ určuje počet atribútov, ktoré má v súbore očakávať a oddeľovací znak medzi jednotlivými poliami záznamu. Tieto polia záznamu sú načítané ako hodnoty určených atribútov. V prípade binárneho súboru, ktorý pozostáva zo záznamov rovnakej dĺžky, je potrebné určiť aj štruktúru týchto záznamov.

1.1.3.2. FTP (File Transfer Protocol)

FTP je sieťový protokol, pomocou ktorého môžeme realizovať prevzatie alebo umiestnenie súboru na vzdialenom súborovom systéme. FTP poskytuje dva základné príkazy, a to „PUT“ na umiestnenie súboru na vzdialenom súborovom systéme a príkaz „GET“ na prevzatie súboru zo vzdialeného súborového systému.

Potrebné informácie pre konektor sú vo všeobecnosti:

- Informácie potrebné na pripojenie

Pre pripojenie je potrebná adresa, na ktorej je dostupný server FTP, prihlasovacie meno a heslo, ktoré server vyžaduje pre autentifikovanie používateľa a port na ktorý sa pripájame (štandardne je služba FTP dostupná na porte 21).

- Názov súboru

Typ údajov

Aj v tomto prípade ide o súbor. Tieto informácie sú vo všeobecnosti rovnaké ako v prípade súbor v súborovom systému.

1.1.3.3. Databáza

Pod pojmom databáza rozumieme v tomto prípade relačnú databázu. Údaje tejto databázy sú uložené v relačných tabuľkách. Prístup k týmto údajom je možné realizovať pomocou jazyka SQL, prípadne volaním uložených procedúr.

Potrebné informácie pre konektor sú vo všeobecnosti:

- Informácie potrebné na pripojenie

Potrebné informácie sú príslušný JDBC ovládač, adresa, kde je umiestnený databázový server, port na ktorom poskytuje svoje funkcie, názov databázy,

prihlasovacie meno a heslo pre autentifikovanie používateľa, nastavenie automatického commitu (podrobnosti sú uvedené nižšie).

- Databázový príkaz, prípadne volanie uloženej procedúry

K údajom uloženým v databáze je možné pristupovať pomocou jazyka SQL. Tento jazyk umožňuje čítanie a zapisovanie do relačných tabuliek. Čítanie údajov z databázy je realizované príkazom SELECT, zápis údajov do databázy je realizovaný príkazom INSERT (vytvorenie nového záznamu) alebo UPDATE (aktualizovanie existujúceho záznamu).

Príkaz SQL, ktorý je určený v konfigurácii konektora, je nutné v niektorých prípadoch dynamicky modifikovať. Najmä v prípade zápisu do tabuľky je potrebné prispôsobiť hodnoty parametrov príkazu údajom v prijatej správe. Spôsob modifikácie je závislý od konkrétneho produktu a je uvedený v časti 2.

Napríklad pri použití produktu OpenAdaptor používateľ produktu určuje len vzory príkazov SQL. V týchto vzoroch uvádza namiesto hodnôt parametrov príkazu názvy atribútov DataObject, ktorých hodnoty sa dynamicky doplnia z prijatej správy.

Iná alternatíva prístupu k údajom databázy je volanie uložených procedúr (stored procedure call). Uloženú procedúru si môžeme predstaviť ako hotový príkaz SQL obohatený o rozhodovaciu logiku. Tieto procedúry sú vhodné hlavne pre opakované realizovanie tej istej operácie.

1.1.3.4. LDAP (Lightweight Directory Access Protocol)

LDAP je prístupový protokol k adresárovej štruktúre, ktorá uchováva údaje. Ide o špeciálny druh databázy určený primárne na čítanie. Základný rozdiel je v organizácii usporiadania týchto údajov. Kým v relačných databázach sú údaje uložené v relačných tabuľkách, v prípade LDAP sú údaje uložené v stromovej štruktúre. K týmto údajom je možné pristupovať pomocou príkazov LDAP.

Potrebné informácie pre konektor sú vo všeobecnosti:

- Informácie potrebné na pripojenie

Pre pripojenie je potrebná adresa, kde sa nachádza adresárová služba LDAP a používateľské meno a heslo pre autentifikovanie používateľa.

- Príkaz LDAP

Pri pristupovaní k údajom definujeme uzol, ktorý reprezentuje koreň podstromu, z ktorého začíname vyhľadávanie a filter, ktorý špecifikuje údaje ktoré vyhľadávame. Možné parametre pre vyhľadávanie sú hĺbka, do akej sa pri vyhľadávaní chceme dostať. Môžeme vyhľadávať v definovanom podstrome, prípadne v rámci celého stromu.

Pri vkladaní záznamov do adresárovej štruktúry, prípadne jej modifikovaní používame podobné parametre, ku ktorým pripájame údaje, ktoré aktualizujú obsah adresárovej štruktúry.

1.1.3.5. Elektronická pošta (Mail)

Elektronická pošta slúži na posielanie a prijímanie elektronických správ v rámci siete. Na posielanie správ sa najčastejšie používa protokol SMTP (Simple Mail Transfer Protocol). Správy sú v rámci servera ukladané v radoch a odosielané do siete, alebo v prípade lokálneho prijemcu ukladané priamo do jeho poštového priečinka. Prichádzajúce správy sú uložené na serveri v poštovom priečinku a je možné ich zo servera prevziať napríklad použitím protokolu POP3, alebo IMAP. Formát správy posielanej prostredníctvom elektronickej pošty je často MIME. Správa je rozdelená na telo a hlavičky, ktoré popisujú jednotlivé časti správy. Súčasťou popisu je aj typ tejto časti. Primárne typy tela správy sú: text, multipart, message, application, audio, image, video, other. V tomto prípade čítanie správy pozostáva z pripojenia sa do spomínaného poštového priečinka, prevzatia správy a extrahovania jej častí tela, ktoré sú definované v hlavičkách.

Potrebné informácie pre konektor sú vo všeobecnosti:

- Informácie potrebné na pripojenie

Pre pripojenie je potrebná adresa poštového servera, poštový protokol, používateľské meno a heslo. Server pre odosielanie správ môže, ale nemusí vyžadovať autentifikáciu. Záleží to od konkrétnej konfigurácie.

- Odosielanie správy použitím elektronickej pošty

Odosielanie správ je realizované spravidla použitím protokolu SMTP. Pri odosielaní elektronickej správy potrebujeme poznať adresáta, ktorému túto správu posielame. Ďalšie možné nastavenia sú predmet správy, ďalší adresáti, adresa odosielateľa a podobne. Niektoré z týchto parametrov je možné nastaviť aj dynamicky z obsahu správy. Napríklad OpenAdaptor umožňuje načítanie adresáta zo zadefinovaného atribútu DataObject.

- Prijatie správy použitím elektronickej pošty

Pre prevzatie správy z poštového priečinka máme na výber hlavne medzi protokolmi POP3 a IMAP. Aby sme predišli viacnásobnému spracovaniu jednej správy, je potrebné správu po prečítaní zmazať, označiť ako prečítanú, prípadne ju presunúť do iného poštového priečinka.

1.1.3.6. JMS (Java Message Service)

JMS definuje štandard popisujúci programátorské rozhranie systému na doručovanie správ. Existuje niekoľko implementácií poskytovaných rôznymi dodávateľmi. Rozlišujeme dva komunikačné režimy a to „point-to-point“ a „publish/subscribe“. Prvý komunikačný režim si môžeme predstaviť tak, že správy sú distribuované do radov (Queue). Každý klient sa spravidla pripája práve na jeden rad, z ktorého číta správy. Druhý model je veľmi podobný mailovej konferencii. Do tém (Topic) sa publikujú správy (publish) a tie sú rozdistribuované príjemcom, ktorí sú prihlásení na prijímanie týchto správ (subscribe). Rady a témy sú spravované jedným alebo viacerými komunikačnými servermi.

Potrebné informácie pre konektor sú vo všeobecnosti:

- Informácie potrebné na pripojenie

Potrebné informácie sú umiestnenie komunikačného servera, názov radu alebo témy, používateľské meno a heslo pre autentifikovanie používateľa a komunikačný režim.

- Čítanie údajov

Pri komunikačnom režime „publish/subscribe“ rozlišujeme stále (durable) a nestále (non-durable) prijímanie správ. Nastavenie stáleho prijímania správ znamená, že klient vie dodatočne prijať správy publikované aj v čase, keď nebol k systému JMS pripojený. V prípade, že klient nie je v čase distribuovania správy pripojený ku komunikačnému systému JMS, tak tento systém uchováva kópiu správy, až kým sa klient znova nepripojí.

- Zapisovanie údajov

Pri posielaní údajov komunikačnému serveru môžeme nastaviť režim uchovávania správ. Trvalé uchovávanie (persistent delivery) znamená, že v prípade výpadku komunikačného servera je správa v rade uchovaná.

1.1.3.7. RMI (Remote Method Invocation)

Java RMI umožňuje objektu, ktorý beží v rámci jednej Java Virtual Machine, volanie metód objektov, ktoré bežia v rámci inej Java Virtual Machine. Rozlišujeme server, kde je k dispozícii služba prístupná cez RMI a klienta, kttorý volá metódy tejto služby.

Zo skúmaných produktov poskytuje rozhranie pre RMI len OpenAdaptor. Umožňuje zaregistrovanie služby prístupnej cez RMI, ktorej metóda `processData()` spracováva prijaté údaje. Na základe prijatých DataObject vracia DataObject XML. Na volanie metódy služby prístupnej cez RMI používa OpenAdaptor komponent RMISink.

1.2. Transformácie údajov

V predošlej kapitole sme sa dozvedeli, z akých externých systémov vieme prijímať údaje a akým externým systémom vieme údaje posielať. Rôzne systémy vyžadujú rôzny formát údajov.

Každý z externých systémov väčšinou pracuje s vlastným formátom údajov. Rôzne systémy môžu rôzne chápať jednu entitu a atribúty, ktoré ju definujú. Napríklad jeden záznam „`Student`“ môže v rôznych systémoch obsahovať rôzne atribúty. V prvom systéme môže tento záznam obsahovať atribúty „`meno`“, „`priezvisko`“, „`rodne_cislo`“, „`adresa`“ a „`telefonne_cislo`“. V druhom systéme môže záznam obsahovať atribúty „`meno`“, „`cisloISIC`“ a „`kontakt`“.

Hodnotou prvého atribútu je meno aj priezvisko. Hodnotou atribútu „`kontakt`“ môže byť adresa, prípadne telefónne číslo. To znamená, že:

- záznam popisujúci ten istý element môže v rôznych systémoch obsahovať rôzne atribúty,
- atribúty s rovnakým menom môžu obsahovať rôzny typ informácie.

Ďalšie nezrovnalosti môžu vyplývať priamo z fyzickej reprezentácie údajov. Jeden systém môže pracovať s údajmi vo formáte XML, iný môže údaje uchovávať v relačnej tabuľke databázy.

Priama modifikácia kódu systému, ktorá by prispôsobila výstup tohto systému vstupu iného systému nie je vhodná. Toto riešenie by si vyžadovalo neprimerané úsilie, ktoré by spočívalo v modifikácii viacerých systémov a zároveň by takto vznikli medzi týmito systémami príliš tesné väzby. Z toho vyplýva primárna potreba translačného mechanizmu v rámci nášho komunikačného systému, ktorý dokáže preložiť údaje prijaté z jedného systému do podoby, ktorá je akceptovateľná systémom, ktorému sú tieto údaje určené.

Transformácie môžu byť umiestnené v osobitnej vrstve, tak ako to ilustruje obrázok 1, prípadne môžu byť transformačné funkcie volané konektormi, ktoré realizujú pripojenie systému k nášmu komunikačnému systému.

1.2.1. XML

XML (eXtensible Markup Language) je jazyk, ktorý slúži pre popis údajov a popis ich štruktúry. Bol prijatý ako odporúčanie W3C (World Wide Web Consortium) 10. februára 1998. XML bol vyvinutý hlavne pre výmenu informácií medzi systémami. Dokument XML je textový súbor, ktorého štruktúru si môžeme predstaviť ako dobre uzátvorkovaný výraz. Zátvorkami sú v tomto prípade párové značky, ktoré môžu byť do seba vnorené. Tieto značky nie sú predpísané štandardom XML, značkou môže byť ľubovoľný neprázdny textový reťazec. Vďaka tomuto formátu je dokument XML ľahko prenositeľný medzi rôznymi platformami.

Na definovanie typu dokumentu XML sa používajú takzvané schémy. Schéma presne popisuje obmedzenia štruktúry dokumentu XML. Definuje možné značky, ich typy a vzájomnú organizáciu v rámci dokumentu XML. Schéma akoby zavádzala poriadok do voľnej štruktúry. Existuje niekoľko jazykov schém, medzi najpoužívanejšie patria DTD (Document Type Definition) a schéma XML (XML Schema).

S týmito dokumentmi súvisí aj jedna z najpoužívanejších transformácii a to XSLT (XSL transformation), ktorá prekladá jeden dokument XML na iný dokument XML. Pomocou XSLT môžeme pridávať a odoberať elementy a atribúty, preorganizovať a utriediť elementy a podobne. Použitie transformácie XSLT vyžaduje napísanie takzvaného stylesheetu XSL, ktorý obsahuje transformačné pravidlá. Preto mu budeme venovať viac pozornosti.

1.2.2. XSLT

XSLT je primárne vytvorený ako súčasť XSL, ale je možné ho použiť aj nezávisle od XSL. Transformácia XSLT je popísaná ako dobre štruktúrovaný dokument XML. Na navigovanie v rámci dokumentu XML, teda na lokalizovanie konkrétneho uzlu, používa XSLT výrazy XPath. XPath je jazyk, ktorý slúži na adresovanie častí dokumentu XML.

Dokument XSLT pozostáva z jednej alebo viacerých množín pravidiel, ktoré nazývame „`template`“. Súčasťou tejto množiny je atribút „`match`“, ktorý slúži na asociovanie týchto pravidiel s konkrétnym elementom XML. Hodnotou tohto atribútu je práve spomínaný výraz XPath. Súčasťou množiny pravidiel sú akcie, ktoré treba vykonať. Transformácia teda popisuje pravidlá ako zo zdrojového stromu vytvoriť výsledný strom a to nasledovne:

1) Najprv je vybraný element v pôvodnom strome, ktorý súhlasí so vzorom (výraz XPath) atribútu match.
2) Vo výslednom strome je vytvorená časť stromu, ktorá zodpovedá výsledkom akcií, ktoré sú definované v množine pravidiel.

Výsledný strom je oddelený od pôvodného stromu a môže mať úplne odlišnú štruktúru ako pôvodný strom.

XSLT je jazyk pre popis transformácie. Na realizáciu transformácie je potrebný procesor XSLT, ktorý ako vstup dostane vstupný dokument XML a dokument popisujúci transformáciu a ako výstup vráti výsledný dokument XML. Jedným z procesorov XSLT, ktoré využívajú skúmané produkty, je Xalan. Xalan je produkt, ktorý je poskytovaný ako súčasť Open Source projektu Apache XML.

1.2.3. Príklad transformácie XSL

Transformáciu XSL ilustrujeme na nasledujúcom príklade. Ako pôvodný dokument XML použijeme databázu kníh. Budeme transformovať dokument tak, aby obsahoval len knihy, ktoré boli vydané po roku 1980. Vo výslednom dokumente chceme mať uvedený len názov knihy a autora. Ďalšia požiadavka je, aby boli záznamy utriedené podľa mena autora:

Pôvodný dokument:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<kniznica>
	<kniha>
		<nazov>The Godfather</nazov>
		<autor>Mario Puzzo</autor>
		<isbn>0451167716</isbn>
		<jazyk>English</jazyk>
		<vydavatel>Signet</vydavatel>
		<rok_vydania>1983</rok_vydania>
	</kniha>
	<kniha>
		<nazov>The Hobbit (Collector's Edition)</nazov>
		<autor>J.R.R. Tolkien</autor>
		<isbn>0395177111</isbn>
		<jazyk>English</jazyk>
		<vydavatel>Houghton Mifflin</vydavatel>
```

```
            <rok_vydania>1973</rok_vydania>
      </kniha>
      <kniha>
            <nazov>Dune</nazov>
            <autor>Frank Herbert</autor>
            <isbn>044100590X</isbn>
            <jazyk>English</jazyk>
            <vydavatel>Ace Hardcover</vydavatel>
            <rok_vydania>1999</rok_vydania>
      </kniha>
</kniznica>
```

Príslušný transformačný dokument:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method='xml' version='4.0' encoding='ISO-8859-1'
indent='yes'/>

<xsl:template match="/">                                          (1)
   <kniznica>                                                (2)
   <xsl:for-each select="kniznica/kniha">                         (3)
       <xsl:sort select="autor"/>                            (4)
          <xsl:if test="year &gt; 1980">                          (5)
              <vybrana_kniha>                                     (6)
                <nazov>                                           (7)
                   <xsl:value-of select="nazov"/>                 (8)
                </nazov>                                          (9)
                <meno_autora>                                     (10)
                   <xsl:value-of select="autor"/>                 (11)
                </meno_autora>                                    (12)
              </vybrana_kniha>                                    (13)
          </xsl:if>                                           (14)
   </xsl:for-each>                                            (15)
   </kniznica>                                                    (16)
</xsl:template>                                                   (17)
</xsl:stylesheet>
```

Riadky (1) – (17) tvoria množinu pravidiel (`template`). Hodnotou atribútu `match` je v tomto prípade celý pôvodný dokument.

Riadky (3) – (15) tvoria telo cyklu. Podmienka pre vstup do cyklu je nájdenie značky `<kniha>` vnoreného medzi značkami `<kniznica>` a `</kniznica>`. To znamená, že telo cyklu sa vykoná pre každú knihu.

Riadok (4) obsahuje príkaz `sort` určený na utriedenie elementov výsledného dokumentu podľa hodnoty určenej značky.

Riadky (5) – (14) tvoria podmienku `if`. Vykonanie tela podmienky, príkazov na riadkoch (6) - (13), závisí od splnenia podmienky `test`.

Riadky (2), (6), (7) a (10) obsahujú názvy elementov vo výslednom dokumente.

Riadky (8) a (11) už priamo hovoria o hodnotách elementov, ktoré chceme zapísať do výsledného dokumentu.

Výsledný dokument po transformácii:

```
<kniznica>
	<vybrana_kniha>
		<nazov>Dune</nazov>
		<meno_autora>Frank Herbert</meno_autora>
	</kniha>
	<vybrana_kniha>
		<nazov>The Godfather</nazov>
		<meno_autora>Mario Puzzo</meno_autora>
	</vybrana_kniha>
</kniznica>
```

1.2.4. Iné transformácie

Ďalšie používané transformácie sú vytvorenie kópie správy, prípadne rozdelenie správy na sekvenciu správ a podobne.

Niektoré transformácie nám poskytuje aj samotný programovací jazyk Java, v ktorom sú produkty vyvíjané. Patria medzi ne hlavne metódy na manipuláciu s reťazcom alebo objektom.

Jedným z možných prístupov k otázke reprezentácie údajov je použitie vlastnej dátovej štruktúry, ako je to v prípade produktu OpenAdaptor. OpenAdaptor používa dátovú štruktúru DataObject. Konektory využívajú metódy komponentov DataObject Reader a DataObject Writer. Tieto komponenty realizujú načítanie vstupných údajov do štruktúry DataObject a zapísanie údajov z tejto štruktúry do formátu pre externý systém, ktorému sú posielané. V rámci komunikačného systému komponenty operujú len s touto štruktúrou. Tento spôsob odľahčuje systém od manipulácie s mnohými formátmi údajov. Táto zodpovednosť je odsunutá na okraj systému k samotnému konektoru. Transformácie sú potom viazané na túto štruktúru.

1.3. Smerovanie

V predošlej časti sme sa venovali transformácii údajov a ich reprezentácii v rámci systému. Smerovanie (routovanie) je mechanizmus, ktorý slúži na správne smerovanie správ v rámci komunikačného systému.

„Router“ je smerovací komponent, ktorý na jednej strane prijíma správy a na základe rozhodovacieho mechanizmu ich posiela na jeden alebo viac výstupov. Správy spracováva v poradí, v akom boli do routra prijaté. Tento komponent vo všeobecnosti nemodifikuje obsah správy. Pri definovaní nových správ v systéme teda postačuje pridať alebo modifikovať pravidlá pre smerovanie. Výstup jedného smerovacieho komponentu môže byť súčasne vstupom iného smerovacieho komponentu. Takto môžeme poprepájať viac smerovacích komponentov a zvýšiť rozhodovaciu silu pre smerovanie správ. Existuje viac druhov smerovacích komponentov, väčšinou sa rozdeľujú podľa stratégie smerovania.

1.3.1. Typy smerovacích komponentov

Podľa [1] delíme smerovacie komponenty v rámci integračného riešenia do štyroch kategórií.

- **Fixed Router**

Ide o najjednoduchší smerovací komponent. Obsahuje práve jeden vstup a práve jeden výstup. Takýto komponent poskytuje napríklad produkt Proteus.

- **Content-based Router**

Takýto smerovací komponent sa rozhoduje len na základe vlastností samotnej správy, ako je napríklad typ správy, prípadne hodnota konkrétneho poľa správy. Príkladom takéhoto komponentu je komponent produktu xBus, ktorý smeruje na základe informácie, ktorú extrahoval z údajov po ich prijatí do komunikačného systému.

- **Context-based Router**

Tento smerovací komponent sa rozhoduje na základe podmienok vyplývajúcich z okolia. Napríklad, ak zlyhá spracovanie v jednom komponente, ktorému poslal správu, môže presmerovať správu na iný spracovávajúci komponent a tak sa vyhnúť zlyhaniu spracovávania správy.

- **Dynamic Router**

Tento smerovací komponent sa sám konfiguruje na základe riadiacich správ od potenciálnych príjemcov. Príkladom implementácie Dynamického smerovacieho komponentu môže byť komponent s dodatočným riadiacim kanálom. Cez tento kanál môžu potenciálni príjemcovia správ posielať smerovaciemu komponentu špeciálne správy, ktoré ho oboznámia s ich prítomnosťou, prípadne podmienkami, za akých sú schopní spracovávať správy.

Ako neskôr uvidíme, skúmané produkty pokryli len prvé dve kategórie smerovacích komponentov a to Fixed Router a Content-based Router.

1.4. Manažment

Otázka manažmentu pokrýva nasledujúce témy, ktoré sú úzko poprepájané. Vo všeobecnosti hovoria o možnostiach pre sledovanie udalostí v spustenom systéme. So sledovaním udalostí súvisí dostatočne skoré odhalenie možnej chyby.

1.4.1. Logovanie (Logging)

Logovanie slúži na zaznamenávanie udalostí, ktoré nastali v rámci systému. Jeden záznam zväčša obsahuje časový údaj (timestamp), kedy udalosť nastala a

samotnú udalosť. Logovacie záznamy sú väčšinou zapisované sekvenčne do textového súboru. Tento súbor sa potom nazýva logovací súbor, prípadne log.

Ako príklad logovania môžeme uviesť takzvaný „RemoteLogger“, komponent, ktorý je súčasťou riešenia, ktoré nám ponúka produkt OpenAdaptor. Pre svoju prácu využíva knižnicu Log4j. Táto knižnica je produktom tímu Apache. Umožňuje nastavenie rôznych úrovní logovania a výpis logovacích záznamov do lokálnych aj vzdialených zdrojov, v rôznych formátoch výstupu. Napríklad záznam o chybe je možné poslať elektronickou poštou na definovanú adresu.

1.4.2. Debugovanie (Debugging)

Debugovanie je proces hľadania chýb a ich odstraňovania. Pri výskyte chyby je dôležité túto chybu lokalizovať. Proces debugovania pozostáva z nasledovných krokov:

- rozoznanie chyby,
- identifikovanie zdroja chyby,
- opravenie chyby.

V našom prípade systém pozostáva z externých systémov a komunikačného systému, ktorý tvorí niekoľko komponentov. Zlyhanie v spracovávaní môže byť spôsobené napríklad zlyhaním spracovávania určitým komponentom.

Chybu môžeme identifikovať napríklad z logu.

1.4.3. Konfigurovanie

Konfigurovať systém znamená nastaviť niektoré jeho vlastnosti a tak ho prispôsobiť našim požiadavkám. Parametre, ktoré sa nastavujú, sú závislé od konkrétneho produktu. Súčasťou konfigurácie môže byť vo všeobecnosti nastavenie parametrov pre samotný beh produktu, pre odchytávanie a spracovávanie výnimiek, logovanie a podobne.

1.4.4. Vzdialené manažovanie

Vzdialené manažovanie umožňuje používateľovi manažovať spustený komunikačný systém. Napríklad OpenAdaptor umožňuje pozastaviť, prípadne obnoviť činnosť adaptéra, prípadne získať stav spusteného adaptéra a podobne.

1.5. Príklad pripojenia systémov ku komunikačnému systému

1.5.1. Model

Ku komunikačnému systému pripájame tri externé systémy:

1. **Systém A:** Databáza, ktorá obsahuje relačnú tabuľku „Student“. V tejto tabuľke uchováva študijné oddelenie informácie o študentoch. Tabuľka obsahuje tieto atribúty:
   - `id` – identifikačné číslo záznamu v tabuľke: predpokladáme, že tento atribút je automaticky inkrementovaný (autoincrement),
   - `meno` – krstné meno študenta,
   - `priezvisko` – priezvisko študenta,
   - `adresa` – adresa trvalého bydliska,
   - `rok_nastupu` – rok, v ktorom začal študovať na fakulte.
2. **Systém B:** Poštový server. Prostredníctvom tohto servera môžeme kontaktovať administrátora, ktorý na základe prijatých údajov vytvorí používateľské konto pre nového študenta.
3. **Systém C:** Rad (queue) JMS, ktorý očakáva správu vo formáte XML. Z tohto radu môže napríklad finančný informačný systém zakladať informácie o študentoch.

Priebeh komunikácie:

1. Do systému A bol pridaný nový študent. To znamená, že v relačnej tabuľke „Studnet“ je vytvorený nový záznam.
2. Na základe pridania nového študenta v systéme A:
2.1. Prostredníctvom Poštového serveru (Systém B) pošleme elektronickú správu administrátorovi.
2.2. Prostredníctvom radu JMS (Systém C) pošleme informácie o novom študentovi.

Údaje, s ktorými systémy pracujú:

1. **Systém A:** Databáza uchováva záznamy v relačných tabuľkách. Odpoveďou na selekciu z tabuľky je množina ResultSet, ktorá obsahuje hodnoty atribútov jedného záznamu z tabuľky.
2. **Systém B:** Administrátor generuje názov používateľského konta na základe mena študenta a školského roku, v ktorom začal študovať na fakulte. Do poštovej správy preto stačí poslať hodnoty atribútov „`meno`“, „`priezvisko`“ a „`rok`“. Formát tela správy je reťazec obsahujúci tieto dva údaje.
3. **Systém C:** Do JMS radu posielame údaje v dokumente XML, ktorého štruktúra je nasledovná[2]:

[2]V zátvorkách „{}“ je uvedený názov atribútu databázového záznamu. Hodnotou značky `<meno>` je meno aj priezvisko študenta.

```
<Student>
            <id>{id}</id>
            <meno>{meno} {priezvisko}</meno>
            <adresa>{adresa}</adresa>
            <rok_nastupu>{rok_nastupu}</rok_nastupu>
      </Student>
```

1.5.2. Riešenie

Riešenie bude vo všeobecnosti pozostávať z týchto krokov:

1. Načítanie záznamu zo systému A.
2. Transformovanie načítaných údajov do podoby, ktorú očakáva externý systém, ktorému tieto údaje posielame.
3. Poslanie údajov externému systému, ktorému sú tieto údaje určené.

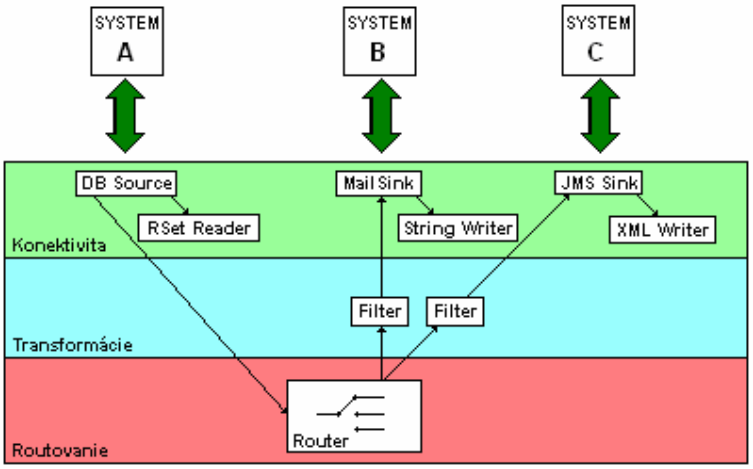
V riešení použijeme komponenty, ktoré nám poskytujú funkcie pre aplikovanie týchto krokov:

1. Konektory sú komponenty, ktoré slúžia ako rozhrania pre komunikáciu s externými systémami (komponenty DB Source, Mail Sink a JMS Sink na obrázku 2).
2. Transformačné komponenty slúžia na transformovanie údajov (Filter, RSet Reader, String Writer, XML Writer).
3. Smerovacie komponenty slúžia na smerovanie údajov v rámci komunikačného systému (Router).

Pri stavbe riešenia použijeme dva prístupy k reprezentácii údajov v rámci komunikačného systému. Prvý prístup sa opiera o použitie vlastnej dátovej štruktúry. Druhý prístup sa opiera o použitie dátovej štruktúry externých systémov.

Výhoda prvého riešenia spočíva v tom, že komponenty použité v rámci komunikačného systému nemusia vedieť manipulovať s viacerými dátovými štruktúrami. Toto riešenie si však vyžaduje dobré zadefinovanie vlastnej dátovej štruktúry a vytvorenie komponentov, ktoré vedia konvertovať inú dátovú štruktúru na nami definovanú dátovú štruktúru a naopak. Dátová štruktúra, ktorú sme pre naše riešenie zvolili, je objekt, ktorý je popísaný nižšie. V tomto objekte sú interpretované údaje aj názov udalosti, ktorú bude táto správa v rámci komunikačného systému reprezentovať.

1.5.2.1. Riešenie, ktoré používa vlastnú dátovú štruktúru



Obrázok 2: riešenie, ktoré používa vlastnú dátovú štruktúru

Najprv si zadefinujeme vlastnú dátovú štruktúru. Zvolili sme štruktúru objekt. (Motivoval nás produkt OpenAdaptor, avšak od jeho štruktúry DataObject sa náš objekt líši.) Každý objekt obsahuje meno a páry „vlastnosť:hodnota“. Meno objektu nazveme podľa udalosti, ktorú bude tento objekt reprezentovať. V našom prípade môže byť toto meno „`novy_student`“.

Objekt bude obsahovať nasledovné páry:

- `nazov_tabulky:Student`
- `id:<id záznamu v DB>`
- `meno:<meno študenta>`
- `priezvisko:<priezvisko študenta>`
- `kontakt:<adresa>`
- `rok_nastupu:<rok>`

Ako sme spomínali, databázy patria k takzvaným „Polled Source“. To znamená, že k tabuľke databázy budeme pristupovať v definovaných intervaloch a sledovať, či pribudol nový záznam. Na to, aby sme vedeli identifikovať nové záznamy, musíme vedieť, ktoré záznamy už komunikačný systém spracoval.

Jeden zo spôsobov, akým je možné identifikovať nový záznam využíva identifikátor záznamu v databáze a jeho vlastnosť automatického zväčšovania. (To znamená, že každý nový záznam v databáze má väčší identifikátor ako všetky záznamy, ktoré boli vložené pred ním.) Každý záznam je jednoznačne definovaný primárnym kľúčom, v našom prípade je to hodnota atribútu „`id`“. Ide o celé číslo, ktoré sa automaticky inkrementuje pri pridaní nového záznamu. Pri spustení komponentu môžeme určiť počiatočnú hodnotu identifikátora, ktorý identifikuje posledný spracovávaný záznam a pri každom výbere záznamu z tabuľky toto číslo zväčšovať.

Výsledkom selekcie z databázy bude jeden prvok množiny „ResultSet“. Na jeho spracovanie použijeme komponent (RSetReader), ktorý túto množinu uloží do našej dátovej štruktúry, ktorá je popísaná vyššie.

Pripravený objekt sa posunie na spracovanie smerovaciemu komponentu (Router). V našom riešení použijeme smerovací komponent, ktorý smeruje správu na základe obsahu (v tomto prípade podľa mena objektu, teda udalosti, ktorú reprezentuje). Tento objekt bude smerovať na dva výstupy pre dva rôzne systémy.

Predposledným krokom bude transformovanie týchto správ do výslednej podoby. Objekt, ktorý bol smerovaný systému B (poštovému serveru) bude modifikovaný tak, že v ňom ostanú len páry „`meno:<meno študenta>`“, „`priezvisko:<priezvisko študenta>`“ a „`rok_studia:<rok>`“. Ostatné páry z neho transformačná funkcia odstráni. Objekt, ktorý bol smerovaný systému C (JMS rad) pretransformujeme tak, že zlúčime páry „`meno:<meno študenta>`“ a „`priezvisko:<priezvisko študenta>`“.

Posledný krok je poslanie údajov systémom B a C, teda príslušným konektorom. Konektor pre systém B použije komponent (StringWriter), ktorý skonvertuje objekt na reťazec. Konektor pre systém C použije komponent (XMLWriter), ktorý skonvertuje objekt na dokument XML, ktorého štruktúra bola popísaná vyššie.

V tomto riešení sme použili tri rôzne komponenty, ktoré dokážu prekladať medzi našou dátovou štruktúrou a iným formátom údajov (ResultSet, String, XML). Z tohto dôvodu sme na jednej strane riešenie skomplikovali, pretože potrebujeme dodatočné komponenty, ktoré využíva konektor pre svoje správne fungovanie. Na druhej strane sme však odľahčili komponenty v ostatných vrstvách od manipulácie s viacerými formátmi údajov.

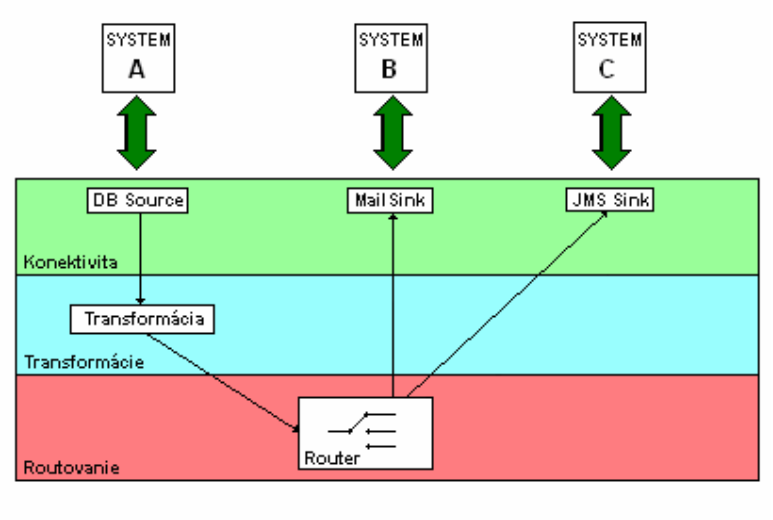
1.5.2.2. Riešenie, ktoré nepoužíva vlastnú dátovú štruktúru

Vo všeobecnosti sa toto riešenie odlišuje len v niektorých krokoch, preto opíšeme hlavné rozdiely.

Pri manipulácii s údajmi v rôznych formátoch je potrebné vedieť z týchto správ extrahovať niektoré informácie (napríklad informácie potrebné pre smerovanie správy). Údaje potom môžeme zabaliť (wrap). Môžeme si to predstaviť ako vloženie údajov do obálky. Na obálku pripíšeme tieto extrahované informácie. Transformácie údajov potom môžeme umiestniť nasledovne:

Transformácia údajov hneď po načítaní zo systému A:

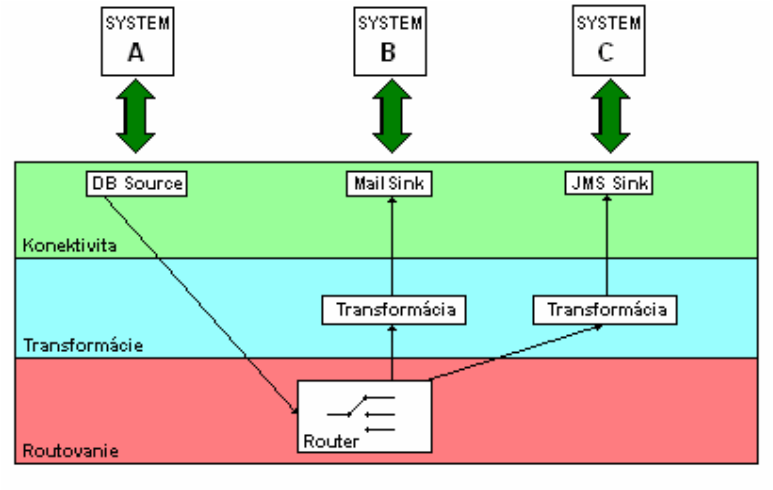
Najprv sa spraví kópia správy. Každá kópia sa transformuje do podoby, ktorú očakávajú príslušné systémy. Tieto správy obalíme a pridáme informáciu pre smerovanie, v tomto prípade názov cieľového systému. Smerovací komponent na základe názvu cieľa presmeruje správu pre konkrétny konektor.



Obrázok 3: transformovanie údajov po načítaní z externého systému

Transformácia údajov až pred zápisom do systémov B a C.

Správu obalíme, pridáme informáciu pre smerovanie, v tomto prípade bude vhodné meno udalosti, ktorú reprezentuje (meno cieľového systému ako v predošlom prípade nie je postačujúce, pretože v tomto kroku máme len jednu kópiu a chceme ju smerovať dvom externým systémom). Smerovací komponent má potom zadefinované, že správu s týmto menom udalosti má presmerovať na dva výstupy. Pred konektormi systémov sú umiestnené transformačné komponenty, ktoré správu odbalia a transformujú na formát, ktorý systémy očakávajú.



Obrázok 4: transformovanie údajov pred zápisom do externých systémov

2. Popis produktov

V tejto kapitole sa budeme venovať konkrétnym skúmaným produktom. Popis sme rozdelili do niekoľkých bodov, ktoré zahŕňajú stručnú charakteristiku aj technické detaily v oblastiach konektivity, transformácií, smerovania a manažovania. Skúmané produkty sú vyvíjané v programovacom jazyku Java a poskytované s otvoreným kódom.

Pre popis skúmaných produktov sme navrhli nasledovné hodnotiace kritériá:

1. **Dodávateľ**
2. **Základná charakteristika**: V tejto časti stručne popíšeme produkt a jeho vlastnosti.
3. **Architektúra:** V tejto časti sa pozrieme predovšetkým na stavebné prvky produktu. Zaujíma nás, aké komponenty daný produkt poskytuje, s akými druhmi údajov vedia tieto komponenty manipulovať a akým spôsobom sú tieto komponenty vzájomne poprepájané.
4. **Konektivita:** V tejto časti sa pozrieme, akým spôsobom je možné realizovať pripojenie vybraných externých systémov. Zaujímajú nás predovšetkým hotové komponenty. Sledujeme spôsob ich použitia a námahu, s akou je tieto komponenty možné použiť.
5. **Transformácie údajov:** V tejto časti sa pozrieme na komponenty, ktoré poskytujú funkcie pre transformovanie údajov. Zaujímajú nás druhy transformácií a spôsob ich použitia.
6. **Smerovanie:** Pozrieme sa, akým spôsobom môžeme smerovať údaje v rámci integračného riešenia. Budeme skúmať hotové smerovacie komponenty, ich použitie, prípadne iné spôsoby smerovania.
7. **Manažovanie:** V tejto časti sa pozrieme na komponenty pre zaznamenávanie udalostí (logovanie), lokalizovanie a odstraňovanie chýb (debugovanie). Pozrieme sa aj na spôsob, akým produkty spracovávajú správy, ktoré nevedia komponenty integračného riešenia priamo spracovať.

2.1. OpenAdaptor

2.1.1. Dodávateľ

Produkt je vyvíjaný pre potreby investičnej banky Dresdner Kleinwort Wasserstein.

2.1.2. Základná charakteristika

OpenAdaptor je produkt, ktorý umožňuje externým systémom komunikovať prostredníctvom posielania správ. Je možné ho použiť na prepojenie:

- systém-systém (produkt poskytuje riešenie, ktoré umožňuje komunikovať externým systémom prostredníctvom výmeny správ), alebo
- systém-„middleware“ (produkt poskytuje vytvorenie spojovacieho prvku medzi externým systémom a iným komunikačným „middleware“).

OpenAdaptor poskytuje množstvo pripravených komponentov pre pripojenie systémov, ako aj komponenty pre transformáciu údajov. Vytvorenie integračného riešenia spočíva vo výbere a pospájaní týchto komponentov do ciest, ktorými budú prechádzať správy.

OpenAdaptor neposkytuje komponenty pre smerovanie správ. Z tohto dôvodu je jeho použitie vhodné predovšetkým v spojení s iným komunikačným „middleware“. (Pomocou tohto produktu vytvoríme adaptér, ktorý bude slúžiť ako spojovací prvok medzi externým systémom a týmto „middleware“.)

2.1.3. Architektúra

Pomocou produktu OpenAdaptoru používateľ pripravuje takzvané adaptéry. Adaptér realizuje výmenu správ pre externé systémy.

2.1.3.1. Komponenty produktu OpenAdaptor a ich vzájomné usporiadanie

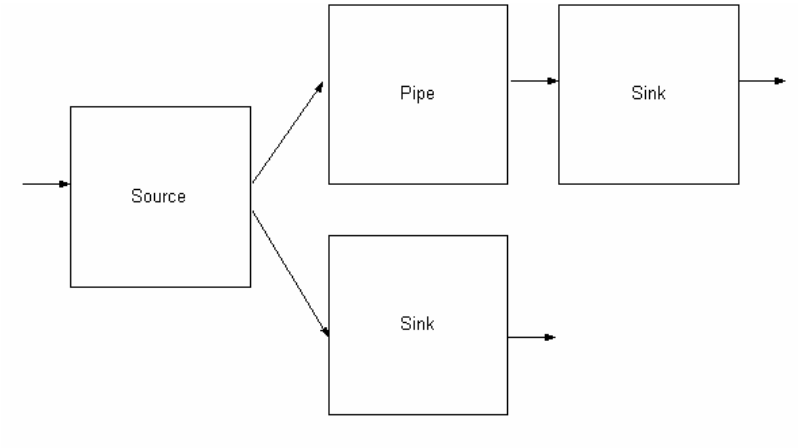
Adaptér obsahuje tri základné typy komponentov[3]:

- „Source“ poskytuje rozhranie, cez ktoré adaptér prijíma údaje od externého systému.
- „Sink“ poskytuje rozhranie, cez ktoré adaptér posiela údaje externému systému.
- „Pipe“ poskytuje transformačné funkcie.

Usporiadanie komponentov v rámci adaptéra si môžeme predstaviť ako orientovaný graf, kde komponenty sú vrcholy grafu, ktoré sú vzájomne pospájané orientovanými hranami. Z každého komponentu „Source“ vedie aspoň jedna cesta do

[3] Pre komponenty „Source“ a „Sink“ používame aj pomenovanie konektor.

aspoň jedného komponentu „Sink“. Vnútorné vrcholy cesty sú výhradne komponenty „Pipes“. Prepojenie komponentov ilustruje obrázok 5.



Obrázok 5: ilustrácia prepojenia komponentov (obrázok je prebraný z [2])

2.1.3.2. Dátová štruktúra, ktorú používa OpenAdaptor na reprezentovanie správ

V rámci adaptéra je používaná vlastná dátová štruktúra, ktorá sa volá DataObject.

Na vstupe (v komponente „Source“) sú údaje z externého systému načítané do DataObject pomocou takzvaných „DOReaders“. Na výstupe (v komponente „Sink“) sú údaje v DataObject skonvertované do príslušnej dátovej štruktúry pre externý systém pomocou takzvaných „DOWriters“. Každá správa v adaptéri je reprezentovaná ako pole inštancií DataObject. Toto pole je štandardne popísané špecifickou schémou XML, ktorá sa nazýva DataObjectXML.

Každý DataObject má definovaný typ a obsahuje množinu atribútov. Jednotlivé atribúty majú definované meno a dátový typ, ktorý obsahujú. Hodnotou atribútu môže byť aj pole iných DataObjects.

Pre ilustráciu uvedieme príklad: záznam „`Student`“ z príkladu v časti 1.5. DataObject, ktorý reprezentuje študenta má definovaný typ „`student`“ a obsahuje atribúty: `id (integer)`, `meno (string)`, `kontakt (string)`, `rok_nastupu (integer)`. Školu reprezentuje DataObject, ktorý má definovaný typ „`fakulta`“. Jeho atribúty budú: `nazov (string)`, `adresa (string)`, `studenti (array of student)`.

Komponenty „DOReaders“ a „DOWriters“, ktoré slúžia na konverziu medzi DataObject a inou štruktúrou údajov, sú využívané v spojení s komponentmi pre konektivitu („Source“ a „Sink“).

OpenAdaptor poskytuje komponenty na spracovanie nasledovných dátových štruktúr:

- záznamy s pevnou dĺžkou polí: FixedWidthStringReader, FixedWidthStringWriter
- záznamy s variabilnou dĺžkou polí, ktoré sú oddelené definovaným separátorom: DelimitedStringReader, DelimitedStringWriter
- jednoduchý dokument XML: XMLStreamReader, XMLStringWriter

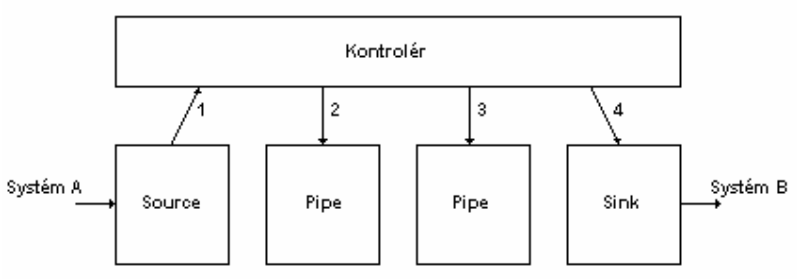
Pri použití iného formátu je nutné vytvoriť vlastný „DOReader“, alebo „DOWriter“.

Pre ilustráciu použitia týchto komponentov uvedieme príklad. V adaptéri máme komponent „FileSource“, ktorý načítava súbor zo súborového systému. Konfigurácia tohto komponentu obsahuje vlastnosti potrebné pre tento komponent (podrobný popis je v časti 2.1.4.) a definovaný „DOReader“. Predpokladajme, že použijeme „DelimitedStringReader“. Najjednoduchšia konfigurácia obsahuje počet atribútov, ktoré bude DataObject obsahovať a hodnotu (ASCII) znaku, ktorý oddeľuje jednotlivé záznamy v súbore. V tomto prípade adaptér vytvorí DataObject typu „Any“ a atribúty s názvami „Att1“,... „AttN“ (kde N je počet atribútov). Štandardný typ týchto atribútov bude „String“. Takto nastavený komponent bude sekvenčne čítať zo súboru záznamy. Každá n-tica záznamov bude tvoriť jeden DataObject (n-tý záznam bude hodnota n-tého atribútu). Správu bude tvoriť pole inštancií DataObject. Tento komponent je možné konfigurovať aj podrobnejšie. Môžeme nastaviť typ DataObject, aj mená a dátové typy jeho atribútov.

2.1.3.3. Riadenie komunikácie

Riadenie v rámci adaptéra koordinuje komponent s názvom „The Controller“ (ďalej „kontrolér“). OpenAdaptor poskytuje implementáciu kontroléra SimpleController. Kontrolér je zodpovedný za začatie vlákien (threadov) zdrojových komponentov, koordinovanie komunikácie medzi komponentmi a manažovanie transakcií. Práca kontroléra vyzerá nasledovne:

Každý komponent „Source“ beží v osobitnom vlákne. Spustenie týchto vlákien koordinuje kontrolér pri svojej inicializácii. Keď komponent „Source“ vygeneruje správu, posunie správu kontrolérovi a ten ju posunie ďalšiemu komponentu v poradí usporiadania komponentov (pipeline) v adaptéri. Prácu kontroléra ilustruje obrázok 6.



Obrázok 6: ilustrácia práce kontroléra

2.1.4. Konektivita

Komponenty „Source“ a „Sink“ reprezentujú rozhrania pre pripojenie externých systémov. OpenAdaptor poskytuje komponenty pre množstvo externých systémov (osobitný komponent pre každý druh externého systému). V závislosti od použitého komponentu používateľ nastavuje tomuto komponentu potrebné vlastnosti v konfiguračnom súbore. Tieto vlastnosti súvisia predovšetkým s informáciami pre pripojenie (umiestnenie, autentifikačné údaje, komunikačný protokol) a so štruktúrou údajov externého systému.

Konektory

2.1.4.1. Súbor

V tejto kapitole sa pozrieme na komponenty, ktoré realizujú čítanie, respektíve zapisovanie súboru v rámci súborového systému.

Názvy komponentov: FileSource, FileBufferSource, FilePollSource, BinaryFileSource, FileSink, BinaryFileSink, MultiFileSink

Pri čítaní súboru máme niekoľko možností v závislosti od použitého komponentu:

- **FileSource** je komponent, pomocou ktorého adaptér realizuje jednoduché čítanie textového súboru. Potrebná vlastnosť, ktorú používateľ konfiguruje je názov súboru.
- **FileBinarySource** je komponent, pomocou ktorého adaptér realizuje čítanie binárnych súborov. V konfigurácii je potrebné navyše určiť štruktúru binárneho súboru. (Pre každú dátovú položku v zázname, z ktorých sa súbor skladá, určujeme jej názov, typ a dĺžku.)
- **FilePollSource** je rozšírením predošlých komponentov (FileSource, FileBinarySource). Umožňuje čítanie súboru opakovane v určenom intervale. Jedna možnosť je čítať súbor vždy od začiatku, iná možnosť je čítanie len novo pridaných záznamov v súbore (komponent si pamätá pozíciu posledného spracovaného záznamu).

V konfigurácii spomínaných komponentov môže používateľ uviesť meno súboru aj v podobe URL (vrátane cesty a protokolu). V tomto prípade OpenAdaptor použije knižnice java.net.URL.

Pre zápis súboru máme tiež k dispozícii viac komponentov:

a. **FileSink** je komponent, pomocou ktorého adaptér realizuje zápis súboru. Pri zápise súboru je potrebné určiť aj akcie, ktoré sa majú vykonať v prípade, že tento súbor v súborovom systéme existuje, prípadne ak súbor prekročil nastavenú veľkosť. Máme možnosť do súboru pridávať záznamy, celý ho prepísať, prípadne premenovať starý súbor (zálohovať ho).

b. **FileBinarySink** je komponent, pomocou ktorého môžeme údaje zapisovať do binárneho súboru. Rovnako, ako v prípade FileBinarySource je potrebné určiť štruktúru záznamov.
c. **MultiFileSink** je komponent, ktorý umožňuje zapisovať obsah správy (t.j. pole inštancií DataObject potenciálne rôznych typov) do viacerých súborov. Adaptér môže zapisovať každý typ DataObject do osobitného súboru.

Uvedené komponenty je možné použiť v spojení s generátorom názvov súborov.

- **FilenameGenerators** je množina komponentov, ktoré umožňujú generovanie mien súborov, ktoré majú byť spracované adaptérom. Napríklad komponent „IncrementalFilenameGenerator“ generuje názvy súborov inkrementovaním hodnoty obsiahnutej vo vzore názvu. „DirectoryListFilenameGenerator“ vracia názvy všetkých súborov v adresári. Vzor mena súboru je regulárny výraz.

2.1.4.2. FTP

V tejto kapitole sa pozrieme na komponenty, ktoré nám umožňujú pracovať so súborom umiestneným v rámci servera FTP.

Názvy komponentov: FTPSink, FTPSource

Komponenty FTP využívajú funkcie knižníc iných dodávateľov. OpenAdaptor umožňuje zvoliť medzi knižnicami ApacheFTP, SunFTP, prípadne JSch, ktorý implementuje bezpečné SFTP. (Príslušná knižnica sa volí v konfiguračnom súbore ako hodnota FTPLibrary.)

Pri pripájaní na server FTP používateľ produktu určuje základné vlastnosti ako sú umiestnenie servera FTP (hostname), port, na ktorom beží služba FTP, používateľské meno a heslo.

Pri čítaní súboru je potrebná znalosť umiestnenia tohto súboru v rámci servera FTP (directory) a jeho názov. Pre názov používa OpenAdaptor hviezdičkovú konvenciu[4].

2.1.4.3. Elektronická pošta

V tejto kapitole sa pozrieme na komponenty, ktoré realizujú prístup k poštovému priečinku na prevzatie správy a posielanie správ prostredníctvom poštového servera.

Názvy komponentov: MailSink, MailSource

Pre pripojenie používateľ určuje základné informácie ako umiestnenie poštového servera, protokol, používateľské meno a heslo.

[4] Komponenty FileSource a FileSink nám túto možnosť neposkytujú, avšak umožňujú použitie FilenameGenerators.

Ďalšou možnosťou je nastavenie typu MIME, t.j. typu obsahu, ktorý posielame, prípadne preberáme. Pri pripájaní k poštovému priečinku používateľ určuje aj maximálny počet pokusov o čítanie (MaxPolls).

- **MailSource** je komponent, ktorý realizuje prevzatie správ z poštového servera. Používateľ si môže vybrať medzi protokolmi POP3 a IMAP.
- **MailSink** je komponent, ktorý realizuje posielanie elektronických správ. Používateľ nastavuje informácie o adresátovi, odosielateľovi, prípadne predmet elektronickej správy, ktorú posielame. Adresát aj predmet správy môže byť dynamicky doplnený priamo z údajov obsiahnutých v DataObject. Napríklad pri použití dynamického adresáta používateľ v konfigurácii neuvádza poštovú adresu, ale názov atribútu DataObject, z ktorého sa tento údaj doplní.

2.1.4.4. Databázy

Pre prístup k údajom, ktoré sú umiestnené v relačných tabuľkách v rámci databázového servera si môžeme zvoliť dva spôsoby. Prvý spôsob je práca s údajmi použitím príkazov SQL. Druhý spôsob je volanie uloženej procedúry v rámci databázy. Prvý spôsob je realizovateľný použitím štandardných konektorov SQLSource a SQLSink.

Názvy komponentov: SQLSource, SQLSink, InformixSink, JDBCSink, OracleSink

Pre pripojenie sú potrebné informácie ako umiestnenie databázy, názov databázy, používateľské meno a heslo.

- **SQLSource, SQLSink** sú komponenty, ktoré pristupujú k údajom v databáze použitím príkazov SQL. Používateľ produktu pripraví vzor príkazov SQL, kde ako hodnoty parametrov tohto príkazu môže použiť atribúty DataObject. Takto pripravený príkaz SQL spracuje komponent „SQLGenerator“, ktorý tieto parametre nahradí hodnotami určených atribútov DataObject. Výsledkom selekcie z databázy je množina „ResultSet“. Pre správne načítanie do DataObject je potrebné konvertovať dátové typy elementov databázy na dátové typy atribútov DataObject. Na to slúži komponent „DOTypeMap“. OpenAdaptor používa štandardne mapu „DefaultDOTypeMap“, ale poskytuje aj možnosť implementovania vlastnej. K databáze konektor pristupuje v intervaloch (Poll). Pre identifikovanie nových záznamov v databáze využíva celočíselný identifikátor záznamov. Pamätá si posledný načítaný identifikátor a po každom načítaní záznamu ho aktualizuje. Tento spôsob sme použili v príklade v časti 1.5. OpenAdaptor umožňuje v rámci selekcie z databázy dodatočne aktualizovať záznamy v databáze (updatesql). Toto umožňuje používateľovi označovať v databáze prečítané záznamy.
- **JDBCSink, InformixSink, OracleSink,** sú komponenty, ktoré realizujú spracovanie inštancií DataObject volaním uloženej procedúry.

2.1.4.5. LDAP

Konektory LDAP umožňujú pristupovať k údajom, ktoré sú uložené v rámci adresárovej štruktúry LDAP.

Názvy komponentov: LDAPSink, LDAPSource

Pri použití komponentu pre čítanie z adresárovej štruktúry používateľ určuje základné vlastnosti pre pripojenie ako sú adresa, používateľské meno a heslo.

Pri čítaní údajov z adresárovej štruktúry používateľ určuje uzol podstromu, z ktorého začína vyhľadávanie, hĺbku v rámci ktorej chceme vyhľadávať a samotný reťazec (filter), ktorý reprezentuje vyhľadávané údaje.

Modifikovanie adresárovej štruktúry nie je možné implementovať bez doprogramovania. Komponent „LDAPSink“ využíva komponent „LDAPEntryModification“, ktorý musí v sebe zahŕňať všetky detaily modifikácie. Používateľ produktu teda musí modifikovať tento objekt a prispôsobiť ho konkrétnej adresárovej štruktúre.

2.1.4.6. JMS

Konektory JMS realizujú výmenu správ s radom (queue), alebo témou (topic) v rámci systému implementujúceho štandard Java Message Service (JMS).

Názvy komponentov: JMSSink, JMSSource

Pre pripojenie k radu alebo téme používateľ určuje základné informácie, ako sú adresa, používateľské meno, heslo a názov radu alebo témy. Pre zápis určuje aj vlastnosti, ako sú priorita publikovanej správy, prípadne nastavenie trvalého uloženia správ (persistent delivery).

2.1.4.7. RMI

RMI slúži na volanie metód vzdialených objektov.

Názvy komponentov: RMISink, RMISource

Pre pripojenie používateľ produktu určuje základné informácie, ako sú adresa RMIRegistry a názov služby.

OpenAdaptor používateľovi umožňuje zaregistrovanie RMI služby implementujúcej rozhranie RMIPubSub, ktorej metóda spracováva prijaté údaje. Na volanie metódy služby používa OpenAdaptor komponent RMISink. Používateľ určuje základné informácie, ako sú umiestnenie a názov služby.

2.1.5. Transformácie údajov

Na transformáciu údajov slúžia komponenty „Pipes“.

Názvy komponentov: FilterPipe, XSLTransformPipe, ValueModifyPipe, ValueAliasPipe, TransformerPipe, AliasPipe, DOAttributeAddPipe, DOWrapperPipe, DOUnwrapperPipe, ArraySplitterPipe, DOMParserPipe

Transformácie údajov sú viazané na vlastnú dátovú štruktúru DataObject. Uvedené komponenty poskytujú nasledovné možnosti:

- **XSLTransformPipe**: Transformácie XSLT je možné aplikovať na štruktúru XML.
- **FilterPipe**: Tento komponent filtruje DataObject na základe typu objektov, prípadne na základe hodnôt atribútov. (Komponent odstráni tie inštancie DataObject, ktoré nezodpovedajú určeným podmienkam.)
- **ValueModifyPipe, ValueAliasPipe, AliasPipe, TransformerPipe:** Tieto komponenty realizujú modifikovanie, prípadne kombinovanie hodnôt atribútov DataObject.
  Príklady:
  - Používateľ chce pridať informáciu k hodnote atribútu DataObject. Hodnotou atribútu je rodné číslo. Pred toto číslo chceme pridať reťazec „RČ“. Používateľ nastaví akciu „Prepend“, a hodnotu atribútu `{RČ }`. (V tomto príklade sme použili komponent ValueModifyPipe.)
  - Používateľ chce nahradiť hodnoty atribútov DataObject, ktoré obsahujú prázdne reťazce hodnotou „NULL“. Použije komponent ValueAliasPipe, kde nastaví, ktorý typ DataObject a ktorý atribút DataObject chce modifikovať, pôvodnú a novú hodnotu.
- **DOAttributeAddPipe**: Tento komponent realizuje pridávanie atribútov (komponent) do inštancie DataObject. Používateľ zvolí typ DataObject, ktorému chce pridať atribút, potom zvolí meno, hodnotu a typ nového atribútu. (Typ atribútu je dátový typ, napr. String, Float a podobne.)
- **DOWrapperPipe, DOUnwrapperPipe:** Tieto komponenty realizujú zabalenie DataObject zo správ dovnútra iného DataObject a naopak. Údaje, ktoré má obsahovať „obálka“ určuje používateľ v konfiguračnom súbore.
- **ArraySplitterPipe**: Tento komponent realizuje rozdelenie poľa inštancií DataObject zo správy na sekvenciu správ (jedna správa bude obsahovať jednu inštanciu DataObject). Jedna správa nemusí obsahovať len jeden objekt, ale môže obsahovať aj pole objektov. Pomocou tohto komponentu vieme toto pole rozdeliť na osobitné správy.
- **HTMLTidyPipe, DOMParserPipe:** Tieto komponenty realizujú spracovanie HTML, XML z reťazca[5] do DOM-stromu.

2.1.6. Smerovanie správ

OpenAdaptor neposkytuje komponent, ktorý by realizoval smerovanie správy. Avšak usporiadanie jednotlivých komponentov čiastočne umožňuje smerovanie správ. Podrobnosti o usporiadaní komponentov sú v časti 2.1.3. Prepojenie medzi komponentmi vytvára cesty, ktorými prechádzajú správy.

2.1.7. Manažment

OpenAdaptor poskytuje riadiaci komponent, ktorý koordinuje posun jednotlivých správ medzi komponentmi. Tento komponent spolupracuje s komponentom „RemoteLogger“ a „RemoteController“.

[5] Atribút DataObject môže byť typu “String“.

RemoteLogger je komponent, ktorý slúži na zaznamenávanie udalostí v rámci adaptéra. Využíva knižnicu Log4j.

OpenAdaptor umožňuje nastavenie logovania vyplnením konfiguračného súboru pre OpenAdaptor, prípadne priamym nastavením Log4j.
OpenAdaptor nám umožňuje nastaviť nasledovné vlastnosti:

- úroveň logovania správ – Od úrovne logovania závisí rýchlosť spracovávania. Čím viac záznamov logujeme, tým je toto logovanie časovo náročnejšie. Úrovne logovania sú zoradené podľa počtu záznamov zostupne (Debug, Info, Warn, Error, Fatal).
- logovanie záznamov s časovým údajom – Záznam bude obsahovať časový údaj (timestamp), kedy bol logovací záznam vytvorený.
- logovanie s informáciou o vlákne – Každý logovací záznam bude obsahovať informáciu o vlákne, v ktorom nastala zaznamenaná udalosť.
- logovanie záznamov s menom komponentu – Každý logovací záznam bude obsahovať meno komponentu, ktorý inicioval zaznamenanú udalosť.

RemoteController je komponent, ktorý slúži na dynamické riadenie spusteného adaptéra. Tento komponent umožňuje pozastaviť, obnoviť, prerušiť činnosť adaptéra, nastaviť úroveň logovania, alebo zistiť stav komponentov. OpenAdaptor umožňuje používateľovi vytvoriť vlastnú funkciu pre riadenie. Toto rozhranie je prístupné napríklad priamo cez protokol HTTP.

V prípade, že niektorý komponent nevie spracovať správu, môže ju preposlať komponentu „MessageHospital“. **MessageHospital** je len rozhranie, nie konkrétny sklad pre správy. (Konkrétna implementácia môže byť realizovaná napríklad prostredníctvom databázy.)

Zhrnutie

OpenAdaptor poskytuje veľa možností v oblasti konektivity aj transformácií. Umožňuje zaznamenávať aktivitu v rámci adaptéra. Avšak neposkytuje žiadne smerovacie komponenty.

2.2. Proteus

2.2.1. Dodávateľ

Info-Scape

2.2.2. Základná charakteristika

Proteus je produkt, ktorý umožňuje externým systémom komunikovať prostredníctvom výmeny správ. Poskytuje komponenty pre pripojenie externých systémov, transformáciu údajov a smerovanie správ.

2.2.3. Architektúra

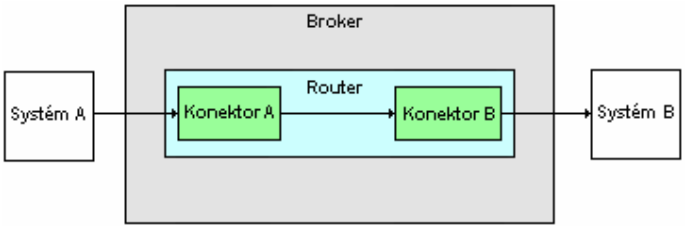
Použitím produktu Proteus používateľ vytvára komunikačný systém, ktorý je tvorený množinou smerovacích komponentov. Všetky tieto komponenty sú definované v rámci centrálnej štruktúry (Broker).

2.2.3.1. Komponenty produktu Proteus a ich vzájomné usporiadanie

Proteus rozlišuje komponenty pre konektivitu, transformovanie a smerovanie. Na rozdiel od predošlého produktu sú však komponenty pre konektivitu úzko spojené so smerovacími komponentmi a komponenty pre transformáciu sú úzko prepojené s komponentmi pre konektivitu. Každý komponent „Source“ je vstupom niektorého smerovacieho komponentu a každý komponent „Sink“ je výstupom niektorého smerovacieho komponentu. To znamená, že tieto komponenty nemožno použiť ako samostatne stojace komponenty, ale len v spojení s niektorým smerovacím komponentom. Podobne je to aj s transformačnými komponentmi, ktoré sú priamo naviazané na konkrétny konektor.

Smerovacie komponenty môžu byť vzájomne prepojené. To znamená, že na výstupe jedného smerovacieho komponentu nemusí byť len konektor pre pripojenie cieľového externého systému, ale aj vstup iného smerovacieho komponentu.

Pre objasnenie usporiadania komponentov uvádzame jednoduchý príklad: Prepájame dva systémy (Systém A, Systém B). Na pripojenie systémov použijeme komponenty pre konektivitu (Konektor A, Konektor B). V rámci komunikačného systému použijeme jednoduchý „Router“, ktorý má jeden vstup a jeden výstup. Vstupom tohto smerovacieho komponentu je konektor A a výstupom je konektor B. Tento príklad ilustruje obrázok 7.



Obrázok 7: prepojenie externých systémov použitím produktu Proteus

2.2.3.2. Dátová štruktúra, ktorú používa Proteus na reprezentovanie správ

V rámci komunikačného systému sú všetky správy reprezentované použitím všeobecného objektu (java.lang.Object). V súčasnosti je väčšina správ typu reťazec, a ako neskôr uvidíme, aj transformácie a smerovanie (s výnimkou jednoduchého smerovacieho komponentu) vyžadujú objekt typu reťazec.

2.2.3.3. Riadenie komunikácie

Riadenie v rámci komunikačného systému, ktorý je vytvorený produktom Proteus vyzerá nasledovne: Komunikačný systém je tvorený množinou smerovacích komponentov, ktoré sú definované v rámci jednej centrálnej štruktúry. Pri inicializácii komunikačného systému sú vytvorené osobitné vlákna pre každý smerovací komponent. Výnimku v tomto prípade tvoria smerovacie komponenty, ktoré sú používané ako výstup iného smerovacieho komponentu. Tieto nie sú spustené v rámci vlastného vlákna. Smerovací komponent sa snaží prijať správu zo svojho vstupu (komponent „Source“) a smerovať ju podľa smerovacích pravidiel na výstup (komponent „Sink“, prípadne vstup iného smerovacieho komponentu).

2.2.4. Konektivita

Komponenty „Source“ a „Sink“ reprezentujú rozhrania pre pripojenie externých systémov. Proteus poskytuje množstvo komponentov pre rôzne druhy externých systémov. Podobne, ako je to pri produkte OpenAdaptor, definícia rozhrania spočíva vo výbere komponentu a nastavení príslušných vlastností (umiestnenie systému, komunikačný protokol, autentifikačné údaje a podobne).

Konektory

2.2.4.1. FTP

V tejto časti sa pozrieme na komponenty, ktoré nám umožňujú pracovať so súborom umiestneným na serveri FTP.

Názvy komponentov: FTPSink, FTPSource

Komponenty FTP využívajú funkcie knižnice EDTFTPJ spoločnosti Enterprise Distributed Technologies.

Pre pripojenie k serveru FTP používateľ určuje základné vlastnosti (umiestnenie servera, port, používateľské meno a heslo).

- **FTPSource** je komponent pre stiahnutie súboru zo servera FTP. Používateľ určuje meno súborov (v hviezdičkovej konvencii) a dodatočnú vlastnosť „SendLines“. Ak je táto vlastnosť nastavená, každý riadok súboru je načítaný ako osobitná správa. Tento komponent nám umožňuje prečítať viac súborov (ak meno súboru zodpovedá viacerým súborom).
- **FTPSink** je komponent, ktorý realizuje uloženie súboru na server FTP. Používateľ určuje názov súboru, ktorý sa má vytvoriť na serveri FTP.

2.2.4.2. Databázy

Proteus umožňuje pristupovať k údajom v databáze použitím príkazov SQL.

Názvy komponentov: DBSink, DBSource

Komponenty využívajú funkcie knižnice Java (java.sql).

Pre pripojenie k databáze používateľ určuje základné vlastnosti (umiestnenie, názov databázy, používateľské meno a heslo) v tvare URL a ovládač JDBC.

- **DBSink** je komponent, pomocou ktorého používateľ posiela údaje databáze. Používateľ pripraví v konfigurácii vzor príkazu SQL INSERT. Tento vzor je spracovaný mapovacím komponentom.
- **DBSource** je komponent, pomocou ktorého používateľ prijíma údaje z databázy. Používateľ určuje, v akých intervaloch má komunikačný systém pristupovať k údajom a rovnako ako pri použití DBSink vytvára vzory príkazov SQL. Na identifikovanie záznamov, ktoré majú byť spracované, Proteus nevyužíva identifikátory záznamov ako OpenAdaptor. Informácie o prečítaných záznamoch ukladá priamo v databáze. Tento spôsob si vyžaduje pridanie stĺpca v tabuľke, ktorý bude obsahovať informáciu o prečítaní záznamu, alebo vytvorenie pomocnej tabuľky. Tento komponent potom pracuje v dvoch krokoch. V prvom kroku načíta záznam z tabuľky, v druhom kroku tento záznam aktualizuje (pridá informáciu, že záznam bol prečítaný).

Tieto konektory používajú mapovacie komponenty. Mapovací komponent poskytuje funkciu, ktorá pri prijatí vzoru príkazu SQL (ktorý môže obsahovať na niektorej pozícii znak „?“) a správy vráti príkaz SQL. Napríklad StringMapper vloží do príkazu SQL celú prichádzajúcu správu.

Pri použití DB2DBMapper má používateľ širšie možnosti. Môže využiť štruktúru „keyPositions“, ktorá slúži na mapovanie údajov z výslednej množiny selekcie na parametre príkazu SQL, ktorým sa aktualizujú údaje v databáze a to nasledovne: Vzor príkazu SQL, ktorý slúži na aktualizovanie záznamu v databáze môže obsahovať jeden alebo viac otáznikov. Štruktúra „keypositions“ sa skladá z párov čísel. Prvé číslo určuje pozíciu hodnoty atribútu v množine, ktorú sme získali

selekciou. Druhé číslo určuje pozíciu otáznika, ktorý sa má touto hodnotou nahradiť. Príklad použitia tejto štruktúry je v prílohe C.

2.2.4.3. JMS

Konektor JMS využíva funkcie knižnice Java (javax.jms).

Názvy komponentov: JMSSink, JMSSource

JMSSink, JMSSource sú komponenty, ktoré umožňujú používateľovi posielať správy do radu alebo téme JMS a prijímať správy z radu, alebo témy JMS. Používateľ určuje základné informácie, ako sú umiestnenie, používateľské meno a heslo a názov radu alebo témy.

2.2.5. Transformácie

Transformačné komponenty sú v produkte Proteus používané takzvanými filtrami. Používateľ môže určiť filter pre ľubovoľný konektor. Filter sa používa na transformovanie údajov po prijatí konektorom „Source“ alebo transformovanie údajov pred ich poslaním externému systému konektorom „Sink“. Použitie filtra vyzerá nasledovne: Používateľ najprv zadefinuje konektor. Potom zadefinuje nový komponent (typu FilteredSource), ktorý bude v sebe obsahovať názov transformačného komponentu a názov pôvodného konektora. Definícia vstupu smerovacieho komponentu bude obsahovať meno tohto nového komponentu.

Proteus rozlišuje tri základné typy filtrov podľa umiestnenia transformácií:

- **FilteredSource** používateľ používa na transformovanie údajov po prijatí do komunikačného systému z externého systému.
- **FilteredSink** používateľ používa na transformovanie údajov pred poslaním externému systému.
- **ExpandedSource** používateľ používa na transformovanie údajov po prijatí do komunikačného systému z externého systému, ak použitý filter vracia viac ako jednu správu. Tento komponent ukladá správy do dočasnej pamäte (cache), z ktorej ich postupne prijíma smerovací komponent a ďalej spracováva.

Proteus poskytuje len tri transformačné komponenty. Tieto komponenty vedia transformovať len reťazec, preto je na správu pred transformáciou aplikovaná metóda `java.lang.Object.toString`.

- **Splitter** je komponent, ktorý rozdelí správu na viac správ použitím separátora. Separátor určuje používateľ.
- **UpCaser** je komponent, ktorý transformuje správu tak, že všetky malé písmená v správe nahradí veľkými písmenami (`toUpperCase()`). Tento komponent je len vzorom pre použitie transformačných metód (metód Java, prípadne vlastných).
- **XSLTer** je komponent ktorý na správu použije transformáciu XSLT. Proteus používa Xalan ako službu XSLT. Používateľ určí meno, prípadne umiestnenie transformačného stylesheetu.

2.2.6. Smerovanie

Proteus poskytuje dva základné typy smerovacích komponentov.

- **StraightRouter** je smerovací komponent, ktorý má jeden vstup a jeden alebo viacej výstupov. Správy smeruje priamo zo vstupu na výstup. To znamená, že tento komponent neobsahuje rozhodovaciu logiku pre smerovanie správ.

- **SwitchedRouter** je smerovací komponent, ktorý patrí do kategórie „ContentBased Router“ (viac v časti 1.3.1.). To znamená, že správy smeruje na základe ich obsahu. Tento komponent má jeden vstup a niekoľko výstupov. Používateľ pri definovaní tohto komponentu definuje niekoľko podmienok. Ku každej podmienke priradí názov výstupu. Správa je presmerovaná na výstup, ktorý je definovaný v rámci podmienky, ktorá bola splnená ako prvá. (V tomto prípade je dôležitá správna sekvencia podmienok.)

Proteus rozlišuje tri druhy „SwitchedRouter“. Líšia sa predovšetkým vo formáte podmienky.

- **RegexSwitch** obsahuje podmienku v tvare regulárneho výrazu. To znamená, že v správe hľadá výraz, ktorý zodpovedá regulárnemu výrazu v podmienke.
- **XMLSwitch** smeruje na základe hodnoty určenej značky XML. To znamená, že pri smerovaní porovnáva hodnotu určenej značky XML v správe s hodnotou, ktorú obsahuje podmienka.
- **SubstringSwitch** smeruje na základe hodnoty podreťazca. To znamená, že v správe vyhľadáva reťazec, ktorý zodpovedá reťazcu v podmienke.

2.2.7. Manažment

Proteus neposkytuje komponenty, ktoré by umožňovali zaznamenávanie udalostí (log), prípadne lokalizovanie chyby (debug).

Poskytuje len testovací komponent, ktorý môže uľahčiť proces nasadenia komunikačného systému. Tento komponent umožňuje otestovanie rozhraní pre jednotlivé externé systémy. (Pri jeho použití nemusíme rozhranie použiť v spojení so smerovacím komponentom.)

Zhrnutie

Proteus poskytuje málo konektorov pre pripojenie externých systémov a len tri transformačné komponenty. Neumožňuje zaznamenávať aktivity, ktoré nastali pri spracovávaní správ komunikačným systémom. Na druhej strane poskytuje viac druhov smerovacích komponentov.

2.3. xBus

2.3.1. Dodávateľ

Stefan Fleckenstein

2.3.2. Základná charakteristika

xBus je možné použiť ako komunikačný systém, ktorý umožňuje externým systémom komunikovať prostredníctvom výmeny správ.

Poskytuje komponenty pre pripojenie externých systémov, transformovanie údajov aj smerovanie správ.

2.3.3. Architektúra

2.3.3.1. Komponenty, ktoré poskytuje xBus

xBus obsahuje tri základné druhy komponentov:

- **konektory**, ktoré slúžia ako rozhranie pre prijímanie, prípadne posielanie údajov externému systému,
- **transformačné komponenty**, ktoré slúžia na vytvorenie správy a transformovanie údajov,
- **smerovací komponent**, ktorý slúži na smerovanie správ.

Komponenty sú usporiadané v troch abstraktných vrstvách. Technická vrstva obsahuje konektory, protokolová vrstva obsahuje transformačné komponenty a aplikačná vrstva obsahuje smerovací komponent.

2.3.3.2. Dátové štruktúry

Komponenty, ktoré sú umiestnené v protokolovej vrstve, realizujú konštrukciu správ a transformácie. Každá správa v systéme je inštancia príslušnej triedy, ktorá obsahuje prijaté údaje a dodatočné informácie (identifikátor správy, informácia pre smerovanie a podobne). Produkt xBus poskytuje komponenty pre spracovanie viacerých štruktúr údajov. Vo všeobecnosti rozlišuje tri základné typy údajov:

- XMLMessage
- TextMessage
- ObjectMessage

Implementáciou týchto troch typov sú nasledovné typy správ, ktoré xBus používa:

- XMLMessage, XBUSXMLMessage, SOAPMessage, RecordTypeMessage, CSVMessage, ByteArrayListMessage, SimpleTextMessage, SimpleObjectMessage

Napríklad do RecordTypeMessage je možné načítať údaje, v ktorých:

- Každý riadok obsahuje práve jeden záznam.
- Polia tohto záznamu majú pevnú dĺžku.

xBus načíta jednotlivé polia záznamov ako reťazec a interne ich štrukturuje v dokumente XML.

ByteArrayListMessage je veľmi podobný predošlému typu. Avšak polia záznamu nie sú načítané ako reťazec, ale ako pole bytov. (Jeden záznam je interpretovaný ako jedno pole bytov.[6])

2.3.3.3. Riadenie komunikácie

Spustenie procesu spracovávania správ pozostáva predovšetkým v aktivovaní konektorov, ktoré prijímajú údaje z externých systémov. Aktivovaný konektor sa pokúša prijať údaje. Po úspešnom prijatí údajov vytvorí inštancie komponentov, ktoré sú potrebné pre ich spracovanie (úspešné odoslanie externému systému, prípadne systémom). Úspešné spracovanie údajov zahŕňa prijatie potvrdenia o doručení, alebo odpovede (údajov).

Spracovávanie správ je možné spustiť dvoma hlavnými spôsobmi:

- **Jednorázové spracovanie**: Komunikačný systém postupne[7] aktivuje jednotlivé konektory, ktoré slúžia na príjem údajov, tieto údaje spracuje a ukončí spracovávanie.
- **Kontinuálne spracovávanie**: Komunikačný system aktivuje vlákna pre konektory, ktoré slúžia na príjem údajov. Každé vlákno potom funguje samostatne, t.j. vytvára si vlastné inštancie komponentov.

2.3.4. Konektivita

Komponenty pre konektivitu sú zaradené v logickej vrstve „technical layer“. Rovnako ako predchádzajúce produkty, aj xBus rozlišuje konektory, ktoré prijímajú údaje z externého systému a konektory, ktoré posielajú údaje externým systémom. xBus rozlišuje konektory na jednorázové (Receiver) a kontinuálne prijímanie údajov (ReceiverThread)[8]. Spoločnou vlastnosťou, ktorú používateľ určuje ku každému použitému konektoru je štruktúra údajov, ktoré konektor prijíma z externého systému, respektíve posiela externému systému.

---

[6] Tento typ je vhodné použiť napríklad vtedy, ak niektoré pole záznamu obsahuje znak „koniec riadku“. RecordTypeMessage by tento znak identifikoval ako koniec záznamu, čo by viedlo k nesprávnemu načítaniu údajov.

[7] Podľa sekvencie určenej konfiguráciou. Aktivuje jeden konektor, ktorý následne spracuje údaje. Potom aktivuje ďalší konektor v poradí. Hlavný rozdiel oproti kontinuálnemu spracovávaniu je, že konektory nie sú aktivované ako samostatné vlákna.

[8] Podrobnosti ohľadom spustenia spracovávania správ sú uvedené v časti architektúra.

V tabuľke 2 je uvedený prehľad externých systémov, ktoré môžu vracať údaje ako odpoveď[9].

Konektor	Súbor	FTP	MQ	DB	LDAP	El.Pošta	Java
Odpoveď (údaje)				x	x		x

Tabuľka 2: prehľad možností externých systémov vracať odpoveď

Konektory

2.3.4.1. Súbor

xBus poskytuje niekoľko konektorov pre súbor. Používateľ určuje názov a cestu k súboru.

Názvy komponentov: FileReceiver[Thread][10], FileSender, FileLineReaderReceiver[Thread], FileLineWriterSender, FileByteArrayListReceiver, FileByteArrayListSender

- **FileReceiver[Thread], FileSender** sú komponenty, ktoré realizujú načítanie/zapísanie celého súboru do/z reťazca.
- **FileLineReaderReceiver[Thread], FileLineWriterSender** sú komponenty, ktoré realizujú načítavanie/zapisovanie po záznamoch. To znamená, že pri prijímaní údajov je súbor otvorený a postupne sú z neho načítavané záznamy (riadky). Pri zapisovaní do súboru je súbor najprv otvorený a postupne sú do neho pridávané záznamy.
- **FileByteArrayListReceiver[Thread],FileByteArrayListSender** sú komponenty, ktoré fungujú podobne ako FileReceiver, FileSender. Líšia sa tým, že údaje nenačítavajú do/z reťazca, ale do zoznamu polí bytov.

2.3.4.2. FTP

xBus poskytuje jeden konektor pre prijímanie údajov a jeden pre posielanie údajov. Tieto konektory využívajú knižnicu org.apache.commons.net.ftp.

Názvy komponentov: FTPReceiver, FTPSender

Používateľ určuje základné informácie, ako sú umiestnenie servera FTP, autentifikačné údaje, názov a cesta k súboru.

2.3.4.3. JMS

xBus poskytuje jeden konektor pre prijímanie údajov z radu správ a jeden konektor pre posielanie údajov radu správ. Tieto konektory využívajú rozhranie javax.jms.

[9] Pri komunikačnom režime Invoke je možné, aby konektor, ktorý inicioval komunikáciu, dostal odpoveď v podobe údajov od externého systému, ktorému boli údaje určené. Podrobnosti sú uvedené v časti 2.3.6.
[10] Toto označenie určuje dva konektory: FileReceiver a FileReceiverThread.

Názvy komponentov: MQReceiver[Thread], MQSender

Používateľ určuje základné informácie, ako sú umiestnenie radu správ a pomenovanie radu správ, pripadne autentifikačné údaje.

2.3.4.4. Databáza

xBus poskytuje jediný konektor pre databázy a to DatabaseSender. Je možné ho použiť pre ľubovoľnú databázu s ovládačom JDBC. Tento konektor realizuje prístup k údajom databázy posielaním príkazov SQL. Príkaz SQL musí byť uvedený v podobe špecializovaného dokumentu XML. Odpoveď príkazu SQL je tiež dokument XML. xBus neposkytuje komponenty, ktoré by realizovali dynamickú upravu príkazu SQL. V prípade, že chceme realizovať selekciu z databázy na základe správy v komunikačnom systéme, je potrebné použiť medzikrok, ktorým bude tento dokument XML upravený[11]. xBus neposkytuje ani metódy, ktorými by vedel identifikovať nové záznamy v databáze. Vďaka stratégii Invoke však môžeme použiť podobný prístup ako pri použití produktu Proteus[12].

Názov komponentu: DatabaseSender

2.3.4.5. LDAP

Podobne ako pri relačných databázach, xBus poskytuje len konektor, ktorý posiela údaje adresárovej štruktúre. Príkazy aj odpoveď sú v podobe dokumentu XML.

Názvy komponentov: LDAPSender

2.3.4.6. Elektronická pošta

xBus poskytuje konektory, ktoré realizujú posielanie a prijímanie elektronických správ.

Názvy komponentov: POP3Receiver[Thread], POP3XMLReceiver[Thread], SMTPSender

- **POP3Receiver[Thread], POP3XMLReceiver[Thread]** sú komponenty, ktoré realizujú prijímanie správ z určeného poštového priečinka použitím protokolu POP3. Prvý komponent zo správy načíta len obsah jej tela (body), druhý komponent načítava aj údaje obsiahnuté v hlavičke tejto správy. Tieto údaje štrukturuje v podobe dokumentu XML.

- **SMTPSender** je konektor, ktorý realizuje odosielanie elektronických správ použitím protokolu SMTP.

---

[11] Na rozdiel od produktu OpenAdaptor alebo xBus, v tomto prípade používateľ neurčuje vzory príkazov SQL, v ktorých by mohli byť hodnoty niektorých parametrov dynamicky doplnené z prijatých údajov.

[12] Výsledkom komunikačného režimu Invoke je odpoveď. Použitím smerovacích pravidiel tak môžeme po selekcii údajov z tabuľky realizovať aj zápis do tabuľky v databáze. Podrobnosti ohľadom stratégie Invoke sú uvedené v časti 2.3.6. Stratégia na identifikovanie zatiaľ nespracovaných údajov je popísaná v časti 2.2.4.2.

2.3.5. Transformácie údajov

Transformačné komponenty v produkte xBus môžeme rozdeliť do troch kategórií.

Do prvej kategórie patria komponenty, ktoré z prijatých údajov vytvoria správu a komponenty, ktoré zo správy vyberú údaje, ktoré sú poslané externému systému. Tieto komponenty sú spomenuté v časti 2.2.3.2.

Názvy komponentov: XMLMessage, TextMessage, ObjectMessage, ByteArrayListMessage, CSVMessage, RecordTypeMessage, SimpleTextMessage, SimpleObjectMessage, SOAPMessage, XBUSXMLMessage, XMLWrapperTransformer, XMLUnwrapperTransformer

XMLWrapperTransformer, **XMLUnwrapperTransformer** sú komponenty ktoré obalia, prípadne vybalia údaje do/z špecifického formátu XBUSXMLMessage. Ide o typ správy XML, ktorá obsahuje pôvodné údaje a dodatočnú hlavičku.

Do druhej kategórie patria komponenty, ktoré vedia transformovať údaje medzi rôznymi štruktúrami údajov. (Napríklad transformovanie reťazca na objekt, správy XML na dokument CSV a podobne.)

Názvy komponentov: XSLTTransformer, RecordTypeTransformer, XMLParserTransformer, XMLSerializerTransformer

- **XSLTTransformer** je komponent, ktorý realizuje transformáciu XSL, teda transformáciu medzi správami XML (XMLMessage).
- **RecordTypeTransformer** je komponent, ktorý realizuje transformáciu medzi RecordTypeMessages. (Aj táto transformácia je určená transformáciou XSLT.)
- **XMLParserTransformer** je komponent, ktorý realizuje transformovanie reťazca do org.w3c.Document.
- **XMLSerializerTransformer** je komponent, ktorý realizuje transformovanie org.w3c.Document do reťazca.

Do tretej kategórie patrí jediný komponent, pomocou ktorého však môže používateľ realizovať viaceré transformácie.

Názvy komponentov: JavaTransformer

JavaTransformer je komponent, pomocou ktorého vieme správu modifikovať volaním metódy Java.

2.3.6. Smerovanie údajov

Smerovanie údajov realizuje smerovací komponent. Smeruje na základe údajov, ktoré boli extrahované z údajov pri vytváraní správy. Tento komponent poskytuje dve stratégie pre smerovanie správ.

Invoke je komunikačný režim, v ktorom sú odpoveďou na prijaté údaje nové údaje. Túto stratégiu si môžeme predstaviť ako volanie metódy objektu. Iniciátor zavolá metódu objektu, ktorému pošle vstupné parametre. Odpoveďou sú údaje, ktoré zodpovedajú výstupu tejto metódy.

Distribute je komunikačný režim, v ktorom je odpoveďou potvrdenie o prijatí údajov (ACK). Túto stratégiu si môžeme predstaviť tak, že konektor A príjme údaje z externého systému A. Tieto údaje sú presmerované konektoru B, ktorý ich posiela systému B. V prípade, že sa konektoru B podarí úspešne doručiť údaje, konektoru A sa vráti odpoveď, ktorá ho informuje o tom, že údaje boli doručené systému B.

Produkt xBus v oboch prípadoch informuje konektor o doručení údajov externému systému. Nie všetky externé systémy však takúto odpoveď očakávajú, prípadne ju vedia spracovať. V mnohých prípadoch je teda informácia o doručení len informáciou, ktorú príjme konektor. (Na základe neúspešného doručenia môže zopakovať príjem údajov.)

Smerovací komponent môže jednu správu smerovať na viac výstupov, podľa určených smerovacích pravidiel. Každé smerovacie pravidlo môže mať určenú inú stratégiu smerovania.

2.3.7. Manažment

xBus poskytuje dve možnosti pre sledovanie udalostí v rámci komunikačného systému.

- Journaling je metóda, ktorá realizuje logovanie udalostí, ktoré súvisia s prijatím údajov z externého systému a posielaním údajov externému systému. Poskytuje dve možnosti zapisovania logovacích záznamov: Umožňuje zapisovať záznamy do databázy alebo do súboru. Prvý spôsob vyžaduje existenciu databázy, v ktorej budú uložené záznamy.
  Záznamy sú vytvárané pre každú udalosť, ktorá súvisí s prijatím alebo posielaním údajov mimo komunikačného systému. Každý záznam obsahuje informácie ako sú identifikátor správy, časové údaje udalosti, výsledok tejto udalosti a podobne.
- Tracing je tiež metóda, ktorá realizuje logovanie udalostí. Na rozdiel od predošlej metódy však nezaznamenáva len údaje súvisiace s príjmom a odosielaním údajov. Táto metóda umožňuje niekoľko úrovní zaznamenávania udalostí (Debug, Info, Warning, Error, Always). Táto metóda umožňuje zapisovanie záznamov len na obrazovku alebo do súboru.

V prípade, že nastane problém pri spracovávaní nejakej správy, je túto správu vhodné z komunikačného systému odstrániť[13]. xBus poskytuje možnosť odstrániť tieto správy do takzvaného skladu vymazaných správ (DeletedMessageStore)[14].

Zhrnutie

xBus poskytuje pomerne veľa konektorov na pripojenie externých systémov aj množstvo transformačných komponentov. Smerovací komponent smeruje správu len na základe informácií, ktoré boli extrahované pri konštrukcii správy. Umožňuje zaznamenávanie aktivít v rámci komunikačného systému.

[13] Systém by sa túto správu pokúšal spracovávať znova a znova. Preto je takéto správy vhodné z komunikačného systému odstrániť.
[14] Zmazanie týchto správ väčšinou postačuje, avšak takýmto spôsobom nestratíme prehľad o tom, ktoré správy boli spracované a ktoré nie.

3. Porovnanie

V tejto kapitole sa budeme venovať vzájomnému porovnaniu troch vyššie popísaných produktov, a to OpenAdaptor, Proteus a xBus. Tieto produkty sú poskytované s otvoreným kódom a implementované v programovacom jazyku Java.

Produkty je možné použiť na realizovanie komunikácie systémov použitím výmeny správ.

Súčasťou produktov sú hotové komponenty, ktoré umožňujú pripojenie systémov ku komunikačnému systému, rôzne transformácie údajov, smerovanie správ v rámci komunikačného systému, prípadne komponenty poskytujúce funkcie pre manažovanie. Tieto komponenty je možné použiť jednoduchým zadefinovaním v konfiguračnom súbore (prípadne súboroch). Definujeme vlastnosti jednotlivých komponentov, ako aj ich vzájomné prepojenie v rámci komunikačného systému.

Porovnanie sme rozdelili do piatich kategórií. Prvá kategória porovnáva architektúru skúmaných produktov a dátovú štruktúru, s ktorou tieto produkty pracujú. Ďalšie štyri kategórie zodpovedajú vrstvám modelu komunikačného systému, ktorý ilustruje obrázok 1.

3.1. Architektúra

V tejto kategórii porovnávame usporiadanie komponentov v komunikačnom systéme a ich vzájomnú komunikáciu. Toto porovnanie budeme robiť hlavne vzhľadom na model komunikačného systému, ktorý je uvedený v kapitole 1. Na úvod sa pozrieme na dátové štruktúry, s ktorými tieto produkty pracujú.

3.1.1. Dátové štruktúry

Vlastnú dátovú štruktúru používa len OpenAdaptor. Táto štruktúra sa nazýva DataObject. Údaje prijaté z externého systému sú spracované do štruktúry DataObject a pri posielaní externému systému sú tieto údaje konvertované zo štruktúry DataObject do formátu, ktorý je akceptovateľný cieľovým systémom. V rámci komunikačného systému si komponenty medzi sebou vymieňajú len pole inštancií DataObject. Ako sme spomínali, toto riešenie odľahčuje ostatné komunikačné vrstvy od manipulácie s viacerými formátmi údajov. Avšak tento spôsob si vyžaduje existenciu komponentov, ktoré dokážu konvertovať rôzne formáty údajov do tejto štruktúry a opačne. Podrobnosti ohľadom konkrétnych komponentov sú spomenuté v časti 2.1.3.2.

xBus z prijatých údajov vytvára správy. Produkt rozlišuje niekoľko typov správ. Konkrétny typ správy určuje používateľ v definícii konektora v závislosti od štruktúry údajov externého systému. Správa v systéme je objekt, ktorý obsahuje pôvodné údaje a dodatočné informácie, ako sú napríklad informácie potrebné pre smerovanie správy. Základné rozdelenie typov správ, ktoré xBus rozlišuje, je spomenuté v časti 2.3.3.2.

Iným riešením je reprezentácia údajov použitím ľubovoľného objektu. Toto riešenie používa produkt Proteus.

3.1.2. Prehľad komponentov

V našom modeli sme uviedli rozdelenie komunikačného systému do vrstiev. Jedna vrstva poskytovala funkcie pre pripojenie externých systémov ku komunikačnému systému, druhá vrstva poskytovala funkcie pre transformáciu údajov a tretia poskytovala funkcie pre smerovanie týchto údajov.

V tabuľke 3 uvádzame prehľad pokrytia týchto oblastí skúmanými produktmi:

Názov produktu	Konektivita	Transformácie	Smerovanie	Manažovanie
OpenAdaptor	x	x		
Proteus	x	x	x	x
xBus	x	x	x	x

Tabuľka 3: prehľad komponentov skúmaných produktov

Vzájomné usporiadanie komponentov v rámci komunikačného systému:

Produkt OpenAdaptor poskytuje dva druhy komponentov.

- Prvý druh je „Source“ a „Sink“, ktoré poskytujú funkcie pre konektivitu. V našom modeli by sme ich mohli umiestniť do vrstvy „konektivita”.
- Druhý druh komponentov sú „Pipes“. Tieto komponenty poskytujú funkcie pre transformáciu údajov a v našom modeli by sme ich mohli umiestniť do vrstvy „transformácie“.

OpenAdaptor priamo neposkytuje komponenty pre smerovanie správ. Ale vzájomné prepojenie jednotlivých komponentov určuje presné smerovanie správ v rámci komunikačného systému.

Architektúra produktu xBus je logicky rozdelená do troch vrstiev:

- „technical layer“, ktorá zodpovedá našej vrstve pre konektivitu,
- „protocol layer“, ktorá zodpovedá vrstve pre transformácie,
- „application layer“, ktorá zodpovedá vrstve pre smerovanie správ.

Každá z týchto vrstiev obsahuje komponenty, ktoré poskytujú príslušné funkcie. Každý komponent je umiestnený v konkrétnej vrstve, ktorá určuje jeho umiestnenie v sekvencii spracovávania.

Prechod správy medzi vrstvami vyzerá nasledovne: Každá správa je z externého systému načítaná konektorom, ktorý je umiestnený v „technical layer“. Na spracovanie údajov sú použité komponenty z vrstvy „protocol layer“ (vytvorenie správy, extrahovanie informácií pre smerovanie, prípadne transformovanie). Potom je správa presmerovaná smerovacím komponentom, ktorý je umiestnený vo vrstve „application layer“.

Produkt Proteus rozlišuje podobne ako OpenAdaptor

- „Source“ a „Sink“, ktoré poskytujú funkcie pre konektivitu a
- „Filters“, ktoré poskytujú funkcie pre transformácie.
- „Routers“, ktoré poskytujú funkcie pre smerovanie správ.

Filozofia vzájomného prepojenia týchto komponentov je odlišná od predošlých dvoch produktov. Source a Sink nie sú samostatné komponenty, ale sú súčasťou smerovacieho komponentu. Tento rozdiel uvidíme hlavne pri porovnaní riadenia v rámci komunikačného systému. Usporiadanie komponentov je nasledovné:

Centrálna štruktúra je tvorená množinou smerovacích komponentov. Každý takýto komponent má vstup a jeden alebo viac výstupov. Vstupom tohto komponentu môže byť konektor na pripojenie externého systému alebo výstup iného smerovacieho komponentu. Výstupom je konektor pre pripojenie externého systému alebo vstup iného smerovacieho komponentu. (Ako vidíme, tieto smerovacie komponenty môžu byť vzájomne poprepájané.)

Z popisu vidíme, že v podstate všetky produkty rozlišujú medzi komponentmi na pripojenie systémov, transformačnými komponentmi a (s výnimkou OpenAdaptoru) smerovacími komponentmi. Porovnania týchto komponentov sú popísané nižšie.

3.1.3. Riadenie v rámci komunikačného systému

Pri nasadzovaní integračného riešenia používajú všetky tri produkty podobný prístup. Najprv používateľ zadefinuje komponenty, ktoré chce používať a ich vzájomné prepojenie. Potom je produkt spustený v súlade s konfiguráciou.

V produkte OpenAdaptor realizuje riadenie komunikácie komponent „The Controller“. Tento kontrolér koordinuje spúšťanie vlákien pre jednotlivé komponenty. Na začiatku spustí vlákna pre všetky komponenty „Source“. Každý z týchto komponentov beží v samostatnom vlákne. V prípade, že zdrojový komponent prijme správu zo systému, pošle ju kontroléru. Ten potom riadi jej spracovanie a smeruje ju ďalším komponentom v určenom poradí.

Komunikačný systém vytvorený produktom Proteus je tvorený množinou smerovacích komponentov. Každý z týchto komponentov je spustený v rámci samostatného vlákna (s výnimkou smerovacieho komponentu, ktorého vstup je výstup iného smerovacieho komponentu). Toto vlákno prijíma správy z komponentu „Source“ a smeruje ich (podľa smerovacích pravidiel) na výstup, ktorý môže byť „Sink“ pre externý systém, alebo vstup iného smerovacieho komponentu.

xBus, podobne ako OpenAdaptor, spúšťa osobitné vlákna pre komponenty „Source“. V rámci týchto vlákien sú vytvárané inštancie komponentov z iných vrstiev, ktoré spracovávajú vytvorenú správu.

Ako vidíme, produkty majú spoločnú črtu. Komponent, ktorý prijíma správy je spustený v rámci samostatného vlákna. Takto môže komunikačný systém naraz prijímať správy z viacerých externých systémov. Správa je načítaná aj spracovávaná v rámci vlákna. A to je zodpovedné aj za jej doručenie. Toto vysvetľuje analógiu medzi na prvý pohľad rozdielnym komunikačným režimom produktu xBus a produktmi OpenAdaptor a Proteus. Všimnime si (v časti 2.3.6.), že produkt xBus poskytuje práve komunikačný režim „Distribute“. Tento režim zahŕňa doručenie odpovede konektoru, ktorý prijal správu. Tento režim je vlastne analógiou komunikačných režimov produktov OpenAdaptor a Proteus. Keďže správa je odosielaná v rámci toho istého vlákna, akým bola prijatá, túto odpoveď dostane konektor vždy.

3.2. Konektivita

V tejto kapitole sa budeme venovať konektivite. Budeme porovnávať komponenty, ktoré slúžia na pripojenie systémov ku komunikačnému systému a spôsob ich použitia. Porovnávanie budeme robiť vzhľadom na konkrétne zdroje údajov.

3.2.1. Konektory skúmaných produktov

Všetky skúmané produkty poskytujú hotové rozhrania, ktoré je možné implementovať vyplnením konfiguračného súboru. Tieto rozhrania sú vo všeobecnosti rozdelené na rozhrania pre externé systémy, z ktorých prijímame správy a na rozhrania pre externé systémy, ktorým posielame správy.

Najprv zjednotíme terminológiu. Prvé dva skúmané produkty, OpenAdaptor a Proteus, používajú pre pomenovanie rozhrania pre systém, ktorému posielame správy výraz „Sink“ a pre rozhranie pre systém, z ktorého prijímame správy, výraz „Source“. xBus namiesto „Sink“ používa výraz „Sender“ a namiesto „Source“ výraz „Receiver“. Pre zjednodušenie budem ďalej používať výrazy „Sink“ a „Source“ (s výnimkou konkrétnych názvov komponentov).

Množiny konektorov jednotlivých produktov sa odlišujú. Niektoré produkty umožňujú pripojiť ku komunikačnému systému viac typov externých systémov, iné menej. Pri skúmaní sme sa obmedzili na niekoľko konkrétnych externých systémov. Prehľad konektorov pre tieto systémy je uvedený v tabuľke 4.

Názov Produktu	Súbor	FTP	Databáza	LDAP	El. pošta	JMS	RMI
OpenAdaptor	x	x	x	x	x	x	x
Proteus		x	x			x	
xBus	x	x	x	x	x	x	

Tabuľka 4: prehľad zdrojov, s ktorými vieme použitím produktov komunikovať

3.2.1.1. Súbor

Komponenty pre manipuláciu so súborom, ktorý je umiestnený v rámci lokálneho alebo pripojeného súborového systému, poskytujú produkty OpenAdaptor a xBus.

Obidva produkty umožňujú pristupovať k súborovému systému v definovaných intervaloch.

3.2.1.2. FTP

OpenAdaptor umožňuje, na rozdiel od ostatných produktov, výber medzi knižnicami ApacheFTP, SunFTP, prípadne knižnicou pre bezpečné SFTP. xBus využíva knižnicu ApacheFTP. Proteus využíva knižnicu EDFTPJ.

OpenAdaptor a Proteus umožňujú pristupovať k systému FTP v intervaloch. Produkt xBus nepracuje s FTP ako s „PolledSource“. To znamená, že umožňuje čítať z FTP len vtedy, ak je tento komponent spustený v režime jednorazového spracovania (toto spracovanie je popísané v časti 2.3.4.).

3.2.1.3. Databázy

Všetky produkty poskytujú komponenty pre prístup k údajom v databáze pomocou príkazov SQL. Produkt OpenAdaptor navyše umožňuje pristupovať k údajom v databáze pomocou volania uložených procedúr.

Formát príkazov SQL

Pri použití produktu OpenAdaptor alebo Proteus používateľ definuje len vzory príkazov SQL. Príkaz SQL je vytvorený zo vzoru, ktorý môže obsahovať parametre, ktoré sa doplnia hodnotami z údajov prijatej správy.

Produkt xBus načítava príkazy SQL z dokumentu XML. Tento dokument má určený špecifický formát. Výsledok príkazu je tiež dokument XML so špecifickým formátom.

Pristupovanie k databáze

OpenAdaptor a Proteus pristupujú k databáze v definovaných časových intervaloch, pretože s databázou pracujú ako s PolledSource. Výnimku v tomto prípade tvorí produkt xBus.

Produkt xBus pre prístup k databáze poskytuje jedine „Sink“ komponent (DatabaseSender), ktorý realizuje posielanie príkazov SQL databáze. Pri selekcii údajov z databázy je nutné použiť stratégiu smerovania „Invoke“[15]. Pristupovanie k databáze v intervaloch by bolo možné realizovať napríklad použitím súboru, v ktorom sa nachádza príkaz SQL [16].

[15] Pri použití “Invoke” konektor prijme údaje ako odpoveď. Táto stratégia je popísaná v časti 2.3.6.
[16] Používateľ použije konektor “FileReceiverThread”. Tento konektor bude pristupovať v intervaloch k súboru, ktorého obsahom môže byť dokument XML s príkazmi SQL.

Identifikovanie nových záznamov pri selekcii z databázy

Proteus umožňuje použiť spôsob, ktorý vyžaduje čiastočnú modifikáciu databázy. Priamo v databáze označuje prečítané záznamy. Tento spôsob je možné použiť vďaka tomu, že Proteus umožňuje pri každom čítaní z databázy aktualizovať záznam v databáze.

OpenAdaptor tiež umožňuje pri každom čítaní z databázy aktualizovať záznam v databáze. Avšak poskytuje aj iný spôsob, ktorý využíva identifikátor záznamu v databáze (celočíselný údaj s vlastnosťou autoincrement).

Produkt xBus priamo neposkytuje možnosť rozlíšenia nových záznamov.

3.2.1.4. LDAP

xBus používa pre prístup k adresárovej štruktúre veľmi podobný spôsob ako pri relačných databázach. Požiadavka aj výsledok spracovania príkazu je v dokumente XML.

OpenAdaptor poskytuje komponenty pre čítanie aj modifikovanie adresárovej štruktúry. Avšak komponent pre modifikovanie adresárovej štruktúry používa objekt, ktorý v sebe musí zahŕňať všetky podrobnosti modifikovania adresárovej štruktúry (presnú množinu atribútov a podobne), ktoré je nutné doprogramovať. To znamená, že LDAPSink nie je možné použiť bez dodatočného programovania.

3.2.1.5. Elektronická pošta

Konektory pre elektronickú poštu poskytujú len produkty OpenAdaptor a xBus.

Odosielanie elektronických správ

Pre odosielanie správ používajú konektory obidvoch produktov protokol SMTP. Používateľ určuje v konfigurácii tohto konektora adresáta správy, obsah správy a podobne. OpenAdaptor umožňuje dynamické nastavenie týchto polí (priamo z prijatých údajov).

Príjmanie elektronických správ

Pre prijímanie správ používa konektor produktu xBus protokol POP3. OpenAdaptor umožňuje aj použitie protokolu IMAP.

Konektory týchto produktov umožňujú pristupovať k poštovému priečinku v určených intervaloch. (Aj elektronická pošta sa radí medzi „PolledSources“.)

Po prečítaní správy je vhodné túto správu z poštovej schránky vymazať, aby nedošlo k jej opakovanému spracovaniu. xBus vymazáva správu automaticky po prečítaní. OpenAdaptor je v tomto smere benevolentnejší. Používateľ môže zvoliť

vymazanie správy, označenie správy, že bola prečítaná, prípadne presunutie správy do iného poštového priečinku[17].

3.2.1.6. JMS

JMS je štandard popisujúci programátorské rozhranie systému na doručovanie správ, ale konkrétne implementácie sa môžu líšiť v závislosti od poskytovateľa JMS produktu (server).

Všetky skúmané produkty poskytujú komponenty, ktoré využívajú rozhranie javax.jms.

3.2.1.7. RMI

Konektory pre RMI poskytuje len OpenAdaptor. Podrobnosti ohľadom RMI komponentu tohto produktu sú uvedené v jeho popise.

Zhrnutie

OpenAdaptor je jediný zo skúmaných produktov, ktorý používa v rámci integračného riešenia vlastnú dátovú štruktúru.

OpenAdaptor poskytuje najviac konektorov pre rôzne externé systémy. Produkt xBus obsahuje menej konektorov. Na rozdiel od ostatných produktov však neposkytuje dostatok možností pri použití databáz. Vyžaduje si pripravené príkazy SQL v dokumente XML. V prípade, že chceme niektoré parametre definovať dynamicky, musíme použiť medzikrok na vytvorenie tohto dokumentu XML. Proteus obsahuje konektory len pre tri typy externých systémov.

3.3. Transformácie údajov

V oblasti tranformácií sme skúmali, akým spôsobom je možné modifikovať údaje. Transformácie môžu slúžiť na zmenu obsahu, zmenu štruktúry, rozdelenie správy na viac správ a podobne. Toto porovnanie rozdelíme do troch kategórií a to nasledovne:

- umiestnenie transformácií v rámci integračného riešenia,
- formát údajov, ktorý transformačné funkcie podporujú,
- samotné transformácie.

Prehľad štruktúr údajov a transformačných komponentov je uvedený v nasledujúcich tabuľkách 5 a 6.

[17] Presunutie správy do iného priečinka je možné realizovať len pri použití protokolu IMAP. Tento protokol umožňuje štrukturovanie poštovej schránky na priečinky, štandardne sa pripája na priečinok s názvom INBOX.

Názov produktu	Štruktúra údajov
OpenAdaptor	Používa vlastnú dátovú štruktúru DataObject.
Proteus	Používa všeobecný objekt java.lang.Object.
xBus	Rozlišuje dokumenty XML (štrukturované údaje), Text (reťazec), Objekt (všetky ostatné typy).

Tabuľka 5: prehľad štruktúr údajov, ktoré produkty používajú v rámci komunikačného systému

Názov produktu	Transf. medzi formátmi	XSLT	Iná modifikácia údajov	Rozdelenie údajov	Kópia údajov	Java metódy	Zabalenie údajov
OpenAdaptor	x	x	x	x	x		x
Proteus		x		x		x	
xBus	x	x	x			x	x

Tabuľka 6: prehľad možných transformácií

3.3.1. Umiestnenie transformácií v rámci integračného riešenia

Umiestnenie transformácií zodpovedá vo všetkých skúmaných produktoch druhej vrstve modelu komunikačného systému, ktorý znázorňuje obrázok 1.

OpenAdaptor používa pre transformovanie údajov komponenty, nazývané „Pipes“, ktoré sú umiestnené medzi „Source“ a „Sink“, teda medzi komponenty, ktoré realizujú pripojenie systémov ku komunikačnému systému.

V produkte Proteus je transformačný komponent naviazaný priamo na konektor, ktorý realizuje pripojenie systému ku komunikačnému systému. Transformáciu je možné realizovať hneď po načítaní údajov z externého systému, prípadne tesne pred poslaním údajov externému systému.

V rámci produktu xBus poskytujú transformačné funkcie komponenty, ktoré sú umiestnené v protokolovej vrstve. Vykonávané sú teda po prijatí údajov konektormi, respektíve pred poslaním údajov externému systému.

3.3.2. Formát údajov, ktorý transformačné funkcie podporujú

OpenAdaptor sa výrazne odlišuje od ostatných produktov práve tým, že v rámci komunikačného systému používa vlastnú dátovú štruktúru DataObject. Transformácie sú viazané na túto štruktúru.

V produkte Proteus sú správy reprezentované všeobecným Java objektom (java.lang.Object). Avšak v súčasnosti dostupné transformačné komponenty

transformujú len údaje reprezentované v podobe reťazca. Pri použití týchto transformácií je všeobecný objekt konvertovaný na reťazec[18].

xBus poskytuje komponenty, ktoré sú viazané na konkrétny typ správy, ale aj komponenty, ktoré dokážu transformovať medzi týmito typmi.

3.3.3. Transformácie

Niektoré transformácie je možné kategorizovať spoločne pre všetky produkty.

3.3.3.1. Transformácie medzi rôznymi formátmi údajov

Pozrieme sa na transformačné komponenty, ktoré transformujú medzi rôznymi formátmi údajov (napríklad XML, CSV, objekt, reťazec, ...).

Komponenty produktu OpenAdaptor, ktoré realizujú pripojenie externých systémov, využívajú priamo transformačné komponenty, ktoré realizujú načítanie údajov do DataObject a opačne.

Proteus používa len všeobecný objekt. Neobsahuje komponenty, ktoré realizujú preklad medzi rôznymi formátmi udajov.

xBus definuje transformátory, ktoré realizujú preklad medzi rôznymi typmi správ (takmer pre každú dvojicu typov).

3.3.3.2. XSLT

Ďalším typom transformačných komponentov sú také, ktoré určitým spôsobom modifikujú údaje (preusporiadanie, vymazanie niektorých hodnôt a podobne). Na preklad dokumentov XML sa používa transformácia XSL. Tento nástroj využívajú všetky skúmané produkty. Príslušný transformačný komponent načítava transformačné pravidlá (v podobe stylesheetu), ktoré sú pripravené používateľom produktu.

OpenAdaptor používa v rámci komunikačného systému DataObject, ale transformáciu XSL nevie priamo aplikovať na túto štruktúru. Hodnotou atribútu DataObject však môže byť dokument XML. (OpenAdaptor poskytuje konektor - FileBufferSource, ktorý umožňuje uložiť celý súbor do jediného atribútu DataObject). Použitím transformačného komponentu (DOMParserPipe) je možné tento reťazec skonvertovať do DOM stromu. Na tento strom vieme aplikovať transformáciu XSLT.

Proteus používa všeobecný objekt, avšak pred samotnou transformáciou tento objekt skonvertuje do reťazca (toString()). Tento reťazec pošle priamo na spracovanie

---

[18] Tento spôsob vyžaduje naozaj veľmi dobrú znalosť (zo strany implementátora riešenia) typu údajov, na ktoré chce transformácie aplikovať. Proteus neposkytuje možnosť logovania aktivít v rámci komunikačného systému, ani odstránenie správ, ktoré nevie spracovať. V tomto prípade by došlo k chybe v spracovávaní, s ktorou by sa komunikačný systém nevedel vysporiadať.

pre procesor XSLT Xalan. Táto metóda samozrejme funguje len vtedy, ak je tento objekt naozaj možné skonvertovať do reťazca predstavujúceho dokument XML.

Aj xBus využíva Xalan ako procesor XSLT. Transformácie XSLT využíva aj pri prekladoch medzi niektorými typmi správ.

3.3.3.3. Iná modifikácia údajov

V prípade produktu OpenAdaptor súvisia ďalšie transformácie práve so štruktúrou DataObject. Umožňuje rôznym spôsobom preusporiadať, prípadne modifikovať jeho atribúty.

xBus a Proteus umožňujú pre transformovanie údajov použiť ľubovoľnú metódu Java. Proteus obsahuje jednu vzorovú triedu, ktorá poskytuje transformovanie pomocou metódy Java toUpperCase(). Slúži ako model pre použitie vlastných transformácií. xBus poskytuje priamo JavaTransformer, ktorý umožňuje modifikovanie údajov volaním metódy jazyka Java.

3.3.3.4. Rozdelenie údajov

Rozdelenie správy na viac správ umožňuje OpenAdaptor a Proteus. OpenAdaptor umožňuje rozdeliť pole inštancií DataObject na postupnosť správ (jedna správa bude obsahovať jednu inštanciu DataObject). Proteus umožňuje rozdeliť správu reprezentovanú reťazcom na viac správ použitím definovaného separátora. xBus neposkytuje takýto transformačný komponent.

3.3.3.5. Kópia údajov

Vytvorenie kópie údajov umožňuje len produkt OpenAdaptor. (Vytvorenie kópie údajov súvisí s usporiadaním a prepojením komponentov, ktoré je popísané v časti 2.1.3.1.)

3.3.3.6. Obalenie a odbalenie údajov

Posledným typom transformácie, ktorý uvedieme je obalenie správy (pridanie informácií k správe). Túto transformáciu sme uviedli v modelovej situácii. Ako sme spomínali, táto transformácia je dôležitá hlavne kvôli pridávaniu informácií, ktoré sú potrebné pre smerovanie správy.

Produkt xBus používa komponent, ktorý správu obalí do takzvaného XBUSXMLMessage. Výsledná správa je dokument XML, ku ktorému je pridaná hlavička.

OpenAdaptor umožňuje zabaliť DataObject do iného DataObjectu. V tomto prípade si môžeme priamo v konfigurácii definovať atribúty, ktoré majú byť použité ako obálka. Pôvodný DataObject bude umiestnený ako hodnota jedného atribútu.

Zhrnutie

Najmenej transformácií poskytuje produkt Proteus. Všetky pracujú výlučne s reťazcami napriek tomu, že Proteus používa na reprezentáciu údajov ľubovoľný objekt.

Produkt xBus poskytuje množstvo transformácii pre preklad medzi rôznymi typmi správ. Každá z transformácií, ktoré používa, je viazaná na jeden, prípadne viac typov týchto správ.

OpenAdaptor používa transformátory na načítavanie do vlastnej štruktúry DataObject a množstvo transformácií, ktoré sú viazané na túto štruktúru.

3.4. Smerovanie správ

V oblasti smerovania správ sme skúmali hotové komponenty, ktoré je možné použiť pre smerovanie správ.

3.4.1. Smerovacie komponenty

Názov produktu	„Fixed-Router“	„Content-Based Router“	Iný typ „Router“
OpenAdaptor			
Proteus	x	x	
xBus		x	

Tabuľka 7: prehľad smerovacích komponentov

V tabuľke 7 je prehľad smerovacích komponentov skúmaných produktov. Komponent „Fixed-Router“ poskytuje len produkt Proteus[19]. Napriek tomu, tento typ smerovania môže používateľ realizovať použitím všetkých skúmaných produktov. V prípade produktu xBus určením jediného výstupu pre správy[20]. Dokonca aj v prípade produktu OpenAdaptor, ktorý neposkytuje smerovacie komponenty, je takéto smerovanie realizované jednoduchým definovaním prepojenia medzi jeho komponentmi.

Produkty xBus a Proteus poskytujú takzvané „Content-Based Routers“, ktoré smerujú správu na základe obsahu (rozdelenie routrov je popísané v časti 1.5.1.).

Proteus smeruje na základe obsahu správy. Tento komponent má zadefinovanú sadu podmienok. Ku každej podmienke je priradený výstup. Správa je smerovaná na výstup, ktorý zodpovedá podmienke, ktorú správa splnila.

[19] Tento na prvý pohľad neužitočný komponent má v produkte Proteus veľký význam. Každé rozhranie pre pripojenie externého systému je definované v rámci niektorého smerovacieho komponentu. To znamená, že bez existencie tohto typu komponentu by sme nemohli vytvoriť jednoduché prepojenie dvoch externých systémov.

[20] Ak chce používateľ smerovať správy určeného typu na jediný výstup a tak simulovať prácu komponentu “Fixed-Router”, zvolí také smerovacie pravidlo, aby každá prijatá správa tohto typu splnila podmienku smerovacieho pravidla.

xBus smeruje na základe informácie pre smerovanie, ktorú extrahoval z údajov pri ich prijatí v transformačnej vrstve.

Smerovací komponent xBus rozdeľuje stratégiu smerovania na dva typy a to Distribute a Invoke. Stratégia Invoke zahŕňa čakanie odpovede po presmerovaní. Odpoveďou je v tomto prípada správa, ktorú môže smerovací komponent presmerovať ďalej. Stratégia smerovania sa uvádza pre každé smerovacie pravidlo osobitne.

Produkt Proteus umožňuje zadefinovanie viacerých smerovacích komponentov, ktoré môžu byť vzájomne poprepájané a to tak, že výstup jedného môže byť vstupom iného.

3.5. Manažment

V tejto kapitole porovnávame komponenty, ktoré slúžia na sledovanie a zaznamenávanie aktivít v rámci systému (log, journal), prípadne komponenty, ktoré nám pomáhajú lokalizovať, prípadne odstraňovať problémy, ktoré nastali pri komunikácii systémov (debug). Pozrieme sa aj na to, akým spôsobom zaobchádzajú produkty so správami, ktoré nevedia z nejakého dôvodu spracovať.

V tabuľke 8 je uvedený stručný prehľad jednotlivých produktov:

Názov produktu	Logovanie	Debugovanie	Spracovanie výnimiek
OpenAdaptor	x	x	x
Proteus			
xBus	x	x	x

Tabuľka 8: prehľad schopností produktov v oblasti manažovania

Produkt Proteus nám neumožňuje ani zaznamenávanie aktivity, ani lokalizovanie chýb. Poskytuje len testovací komponent, ktorý umožňuje otestovať rozhranie pre komunikáciu s konkrétnym systémom. Tento komponent pošle testovanému konektoru „Sink“ jednoduchú správu „Hello World“, prípadne prijme údaje z testovaného konektora „Source“.

Produkty OpenAdaptor a xBus umožňujú logovanie záznamov.

OpenAdaptor využíva funkcie knižnice Log4j. Môžeme definovať úroveň logovania aj lokáciu logovacích záznamov (logovací záznam je možné poslať aj prostredníctvom elektronickej pošty určenému adresátovi).

Logovanie produktu xBus je realizované podpornými funkciami, ktoré nie sú zaradené do žiadnej z jeho vrstiev. Na rozdiel od OpenAdaptoru rozlišuje dva druhy logovania záznamov. Prvý sa nazýva „Tracing“, druhý „Journaling“. Prvý spôsob je analógiou logovania v produkte OpenAdaptor z toho hľadiska, že umožňuje nastavenie úrovne zaznamenávania. Výstup týchto záznamov však vie smerovať len na konzolu (obrazovku) alebo do súboru. Druhý spôsob umožňuje zaznamenávanie udalosti prijatia údajov do komunikačného systému a poslania údajov externému systému. Tieto logovacie záznamy je možné ukladať do súboru alebo do logovacej tabuľky umiestnenej v databáze.

Všimnime si, že logovanie (úroveň Debug) v produkte OpenAdaptor a Tracing (úroveň Debug) v produkte xBus umožňuje aj lokalizovanie problému. Ak vieme zaznamenať každú udalosť v systéme, vieme z týchto záznamov vyčítať, kde bola správa prvýkrat nesprávne spracovaná, prípadne kde jej spracovávanie skončilo.

Niekedy môže nastať situácia, že v rámci nášho komunikačného systému nevieme niektoré správy spracovať. Táto situácia môže nastať napríklad pri zlyhaní externého zdroja, prípadne ak má správa iný formát, ako daný komponent očakáva. Kým správa nie je úspešne spracovaná, ostáva v komunikačnom systéme. Najvhodnejšie je takúto správu presunúť do nejakého skladu nespracovaných správ. Tento môže obsahovať funkcie, ktoré vedia ako správu ďalej spracovať. Produkt OpenAdaptor používa takzvané MessageHospitals, produkt xBus používa Deleted Message Store. Už podľa názvu vidíme, že v prípade produktu xBus ide naozaj len o akýsi sklad vymazaných správ. MessageHospital v prípade OpenAdaptoru je len rozhranie, nie konkrétny sklad pre správy. (Konkrétna implementácia môže byť realizovaná napríklad prostredníctvom databázy.)

Zhrnutie

Proteus ako jediný neposkytuje žiadne možnosti v oblasti manažovania. OpenAdaptor a xBus poskytujú v skúmaných oblastiach približne rovnaké možnosti.

Záver

Cieľom tejto práce bolo analyzovať a porovnať vybrané produkty, ktoré realizujú integráciu aplikácií. Venovali sme sa trom konkrétnym open source riešeniam, ktoré je možné použiť na integráciu aplikácií prostredníctvom výmeny správ.

Prieskumom sme našli viaceré prístupy k integrovaniu aplikácií. Okrem prostriedkov popísaných v tejto práci ide napríklad o modelovanie a spúšťanie podnikových procesov použitím technológie BPEL (Business Proces Execution Language). Avšak z časových dôvodov nebolo možné rozumne preskúmať viaceré z nich. Preto sme sa sústredili na jeden typ produktov, aby sme mohli príslušné riešenia preskúmať detailnejšie.

V práci sme najprv teoreticky popísali komunikačný systém, ktorý realizuje integrovanie aplikácií. Potom sme preskúmali detaily jednotlivých produktov. V tejto oblasti sme vychádzali predovšetkým z dokumentácie k produktom. V mnohých prípadoch však dokumentácia nebola postačujúcim zdrojom informácií a preto sme proces prieskumu kombinovali aj s analýzou zdrojových kódov. Práve v tomto prípade sa ukázala voľba open source produktov veľmi dobrým riešením. Nakoniec sme tieto produkty v jednotlivých oblastiach porovnali. Doplnkom práce je realizácia jednoduchého príkladu použitím jednotlivých produktov. Tento príklad ilustruje základné rozdiely v prístupoch týchto na prvý pohľad veľmi podobných produktoch.

Táto práca neobsahuje návody, ako používať tieto produkty, ale mala by poskytnúť prehľad možností týchto produktov v konkrétnych oblastiach integrácie. Získaný prehľad môže slúžiť ako podklad pre rozhodnutie o použití nástroja založeného na výmene správ, resp. pre výber konkrétneho nástroja v tejto oblasti. Práca je zároveň inšpiráciou pre realizáciu vlastného integračného riešenia a to buď kombinovaním komponentov týchto produktov, alebo vytvorením nového nástroja.

Pokračovaním tejto práce by mohlo byť vytvorenie reálneho integračného riešenia použitím niektorého skúmaného produktu. Zaujímavé by bolo aj porovnanie ďalších prístupov k riešeniu integrácie aplikácii (ako je napr. spomínaná technológia BPEL), prípadne porovnanie nástrojov typu open source s komerčnými riešeniami. Ďalšou možnosťou pokračovania je aplikovanie získaných myšlienok a vzorov na napísanie vlastného produktu, ktorý by realizoval integráciu prostredníctvom výmeny správ.

Slovník pojmov

Adaptér – komunikačný systém vytvorený použitím produktu OpenAdaptor
ASCII (**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange) – kódovanie znakov založené na anglickej abecede, používané na reprezentovanie textu v počítačoch
Databáza – štruktúrovaný úložný priestor pre údaje rôznych typov
Debugovanie – metodický proces na lokalizovanie a odstraňovanie chýb
DOM (Document Object Model) – popis spôsobu akým je možné objektovo reprezentovať dokumenty XML
DTD (Document Type Definition) – schéma, ktorá je používaná na presné definovanie štruktúry dokumentu XML
EAI (Enterprise Application Integration) – proces integrácie podnikových aplikácií
ERP (Enterprise Resource Planning) - označenie podnikový systém, ktoré v sebe zahŕňa všetky aspekty činnosti spoločnosti, vrátane plánovania, výroby, predaja a marketingu
FTP (File Transfer Protocol) – protokol, ktorý je používaný pre výmenu súborov v rámci siete, ktorá podporuje protokol TCP/IP
IMAP (Internet Message Access Protocol) – protokol, ktorý umožňuje lokálnemu klientovy pristupovať k elektronickej pošte na vzdialenom serveri
Komunikačný systém – systém, ktorý umožňuje externým systémom komunikovať prostredníctvom výmeny správ
JMS (Java Message Service) – štandard popisujúci programátorské rozhranie systému na doručovanie správ
Klient – systém, ktorý využíva služby, ktoré sú poskytované serverom
Konektor – komponent v komunikačnom systéme, ktorý reprezentuje rozhranie pre komunikáciu s externým systémom
LDAP (Lightweight Directory Access Protocol) – protokol pre vyhľadávanie, respektíve modifikovanie údajov v rámci adresárovej služby, ktorá je dostupná cez protokol TCP/IP
Logovanie – chronologické zaznamenávanie udalostí, ktoré nastali pri vykonávaní v rámci systému
Middleware – systém, ktorý slúži ako sprostredkovateľ komunikácie medzi viacerými aplikačnými komponentmi
MIME (Multipurpose Internet Mail Extension) – internetový štandard pre formát elektronických správ
Open source – tento pojem označuje vo všeobecnosti akúkoľvek informáciu dostupnú verejnosti, za podmienky, že možnosť jej slobodného šírenia zostane zachovaná
Ovládač (Driver) – počítačový program, ktorý poskytuje funkcie na ovládanie softvérového alebo hardvérového komponentu
Pipe – druh komponentu produktu OpenAdaptor, ktorý poskytuje predovšetkým transformačné funkcie
Platforma – označenie charakteristickej množiny hardvérových, resp. softvérových komponentov, z ktorých je postavený počítač (platforma môže označovať

charakteristický operačný systém, prípadne hardvér, napríklad platforma Windows vs Unix alebo x86 vs SPARX)
Point-to-point - v JMS: komunikačný režim, v ktorom sú správy ukladané do radov
POP3 (Post Office Protocol version 3) **-** protokol, ktorý umožňuje lokálnemu klientovy preberať elektronickú poštu zo vzdialeného servera
Publish/subscribe – v JMS: komunikačný režim, v ktorom sú prijaté správy distribuované definovaným príjemcom
Rad (QUEUE) – programová súčasť pre ukladanie správ
RMI (Remote Method Invocation)
Rozhranie (označované ako API – Application Programming Interface) – rozhranie systému, cez ktoré poskytuje svoje funkcie
Server – systém, ktorý poskytuje služby klientom
Sink – rozhranie, cez ktoré komunikačný systém posiela údaje externému systému
SMTP (Simple Mail Transfer Protocol) **–** štandard pre prenos správ v rámci Internetu
Source – rozhranie, cez ktoré komunikačný systém prijíma údaje od externého systému
SQL (Structured Query Language) **–** najpoužívanejší počítačový jazyk, ktorý slúži na vytváranie, modifikovanie a vyberanie údajov z relačných databáz
Súbor (File) – štruktúra, ktorá je používaná na uchovávanie informácií v rámci počítaču
Systém – zoskupenie hardvérových a softverových komponentov, ktorý slúži na uchovávanie, alebo spracovávanie informácií
Transakcia – zložená operácia, ktorá pozostáva z atomických operácií, ktoré musia byť vykonané všetky, alebo žiadna z nich
Uložená procedúra (Stored Procedure) – príkaz SQL obohatený o rozhodovaciu logiku
URL (Uniform Resource Locator) **–** lokátor zdroja, ktorý pozostáva z adresy zdroja aj spôsobu prístupu k tomuto zdroju
Vlákno (thread) – spôsob, akým môžu programy rozdeliť spracovávanie do dvoch alebo viacerých simultáne vykonávaných úloh v rámci jedného procesu
W3C (The World Wide Web Consortium) – medzinárodné združenie, ktoré sa venuje vyvíjaniu webových štandardov
XML (Extensible Markup Language) **–** jazyk pre popis údajov, používaný ako štandard pre výmenu informácií medzi rôznymi systémami
XPath (XML Path Laguage) **–** jazyk, ktorý je primárne používaný na adresovanie častí dokumentu XML
XML Schema - schéma, ktorá je používaná na presné definovanie štruktúry dokumentu XML
XSLT (Extensible Stylesheet Lannguage Transformation) – jazyk, ktorý sa používa na definovanie transformácie medzi dokumentami XML

Zoznam literatúry

[1] Gregor Hohpe and Bobby Woolf. *Enterprise Integration Patterns.* Addison-Wesley Proffesional 2004.

[2] *OpenAdaptor* [online]. Domovská stránka. Formát HTML. Dostupné na internete. <http://www.openadaptor.org>

[3] *OpenEAI Project* [online]. Domovská stránka. Formát HTML. Dostupné na internete. <http://www.openeai.org>

[4] *xBus* [online]. Domovská stránka. Formát HTML. Dostupné na internete. <http://xbus.sourceforge.net>

[5] *Proteus* [online]. Domovská stránka. Formát HTML. Dostupné na internete. <http://www.info-scape.net/proteus>

[6] Henry Peyret a Michael Goulde v spolupráci s Andrew Parker. *Are Open Source Integration Solutions Mature?* [online]. Analyst Corner 2005. Formát HTML. Dostupné na internete. <http://www2.cio.com/analyst/report3741.html>

[7] Dave Rosenberg. *Open Source EAI* [online]. InfoWorld 2005. Formát HTML. Dostupné na internete. <http://weblog.infoworld.com/openresource/archives/2005/07/open_source_eai.html>

[8] *Open Source Initiativ* [online]. Domovská stránka. Formát HTML. Dostupné na internete. <http://www.opensource.org>

[9] Eric Austvold. *Five Minutes with Eric Austvold* [online]. Formát HTML. Dostupné na internete. <http://www.siebel.com/customer-case-studies/interview.shtm?year=2003&vol=3&issue=tech&numQ=9&lname=austvold&template=1>

[10] *Enterprise Application Integration* [online]. Formát HTML. Dostupné na internete. <http://www.openpsa.net/eai.htm>

[11] Ian Cartwright and Gregor Hohpe. *Open Source Middleware: Ready for Prime Time?* [online]. Formát PDF. Dostupné na internete. <http://www.cutter.com/offers/eau0512.pdf>

[12] David Reilly. *Introduction to Java RMI* [online]. 2000. Formát HTML. Dostupné na internete. <http://www.javacoffeebreak.com/articles/javarmi/javarmi.html>

[13] *Mapping SQL and Java data types* [online]. Formát HTML. Dostupné na internete.<http://www.service-architecture.com/database/articles/mapping_sql_and_java_data_types.html>

[14] *World Wide Web Consortium* [online]. Domovská stránka. Dostupné na internete. <http://www.w3.org>

[15] Carlos E. Perez. *Open Source Written in Java* [online]. 2006. Formát HTML. Dostupné na internete. <http://www.manageability.org/blog/stuff/open-source-messaging-integration-transformation-routing-java/view>

[16] Eric van der Vlist. *Two Open Source project for EAI* [online]. 2003. Formát HTML. Dostupné na internete. <http://www.xmlhack.com/read.php?item=1917>

[17] Matt Assay. *Open source EAI: The next Big Thing?* [online]. 2005. Formát HTML. Dostupné na internete. <http://asay.blogspot.com/2005/06/open-source-eai-next-big-thing.html>

A. Všeobecne o riešeniach

Produkty sú vytvorené programovacím jazykom Java. Pri realizovaní riešení sme použili nasledovné softvérové komponenty:

- platformu linux x86 (Distribúcia Debian r3.1stable jadro 2.4.27),
- java software development kit (j2sdk1.4.2_11),
- databázový server MySQL (mysql4.1),
- poštový server (exim3), server ftp (proftpd 1.2.10),
- ovládač mysql pre java (mysql-connector-java-3.1.12),
- ovládač pre JMS SunAS.

Riešenie bolo realizované na nasledovnej hardvérovej konfigurácii:

- procesor IntelPentium 4 3.0GHz
- fyzická pamäť 1GB RAM
- pevný disk s kapacitou 80GB (využitý priestor približne 1GB).

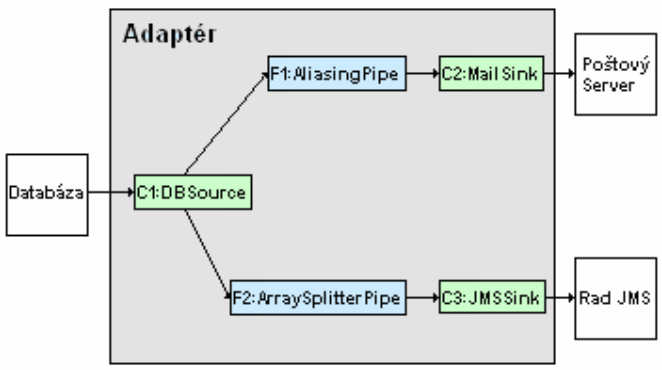
B. Riešenie použitím produktu OpenAdaptor

I. Priebeh inštalácie, požiadavky produktu

Inštalácia prebiehala bez komplikácií podľa návodu na inštaláciu. OpenAdaptor používa lokálnu premennú `$CLASSPATH`. V tejto premennej musia byť zadefinované všetky knižnice, ktoré produkt používa. Pre JMS sme použili vlastnú knižnicu, ktorú stačilo pridať do tejto premennej.

II. Popis riešenia

Riešenie pozostáva z jedného komponentu „Source“ pre databázu, dvoch konektorov „Sink“ pre JMS a elektronickú poštu a dvoch filtrov pre výstupné konektory. Toto riešenie ilustruje nasledovný obrázok.



Obrázok 8: rozloženie komponentov v adaptéri

Adaptér v pravidelných intervaloch pristupuje cez rozhranie „DBSource“ k údajom v databáze. Pokúša sa načítať záznamy z databázy s najmenším nespracovaným identifikátorom. V premennej `PK` si pamätá identifikátor posledného spracovaného záznamu. Pred každou selekciou údajov z databázy najprv aktualizuje túto premennú a to tak, že v tabuľke vyhľadá najmenší identifikátor, ktorý je väčší ako hodnota tejto premennej. Prijaté údaje posiela dvom transformačným komponentom. Po transformovaní sú údaje poslané poštovému serveru cez rozhranie „MailSink“ a do radu JMS cez rozhranie „JMSSink“.

Komponent DBSource načítal množinu záznamov do poľa inštancií DataObject, kde počet atribútov DataObject zodpovedal počtu načítaných stĺpcov databázy.

Komponent „AliasingPipe“ vytvorí nový typ DataObject, ktorý obsahuje len vybrané atribúty pôvodného typu. To znamená, že po spracovaní je správa tvorená poľom inštancií DataObject nového typu.

Komponent „ArraySplitterPipe“ rozdelí pole inštancií DataObject na viac správ a to tak, aby práve jedna správa obsahovala informácie z jedného záznamu databázy. Takto zaručíme, že do radu JMS budeme posielať správy, z ktorých každá bude obsahovať údaje o jednom študentovi.

Komponent „MailSink“ používa komponent „DelimitedStringWriter“, kde ako oddeľovací znak sme použili medzeru.

Komponent „JMSSink“ používa komponent „XMLStringWriter“, ktorý údaje formátuje v podobe dokumentu XML.

III. Spustenie riešenia

Spracovávanie správ sme spúšťali príkazom

```
java org.openadaptor.adaptor.RunAdaptor A1.props A1
```

kde `adapter1.props` je názov konfiguračného súboru a `A1` je názov adaptéra.

IV. Komplikácie

- V konfigurácii elektronickej pošty nemôže byť uvedená adresa vo formáte `užívateľ@IP`.
- Pri použití inkrementovania identifikátora, ktorú tvorcovia použili v ilustračnom príklade, adaptér nefungoval správne, ak neexistoval niektorý identifikátor (napríklad zmazanie záznamu, kde id je 4 spôsobí, že adaptér vie spracovať len záznamy 1, 2, 3). Použili sme odlišné riešenie. Hodnotu identifikátora určuje minimálna hodnota množiny identifikátorov.
- V prípade, že sme do premennej `$CLASSPATH` pridali rohranie javax, ktoré potreboval produkt Proteus, riešenie prestalo fungovať. Tento problém sme vyriešili osobitnou definíciou premennej `$CLASSPATH` pre OpenAdaptor.
- Nenašli sme komponent, ktorý by vedel zreťaziť hodnotu dvoch atribútov. Tento problém sa nám nepodarilo vyriešiť. Preto sme nerealizovali transformáciu, ktorá spája atribúty „`meno`“ a „`priezvisko`“ do jediného atribútu „`meno`“.

V. Konfiguračné súbory

Konfiguračný súbor produktu OpenAdaptor: `adapter1.props`

```
### konektory

A1.Component1.Name = C1
A1.Component2.Name = C2
A1.Component3.Name = C3

### filtre

A1.Component4.Name = F1
A1.Component5.Name = F2

### definovane prepojenia

A1.C1.LinkTo1 = F1
A1.F1.LinkTo3 = C2
A1.C1.LinkTo2 = F2
A1.F2.LinkTo4 = C3

### konektor Source pre databazu

A1.C1.ClassName = org.openadaptor.adaptor.jdbc.PollingSQLSource
```

```
A1.C1.Database = slnce
A1.C1.JdbcUrl = jdbc:mysql://localhost/
A1.C1.JdbcDriver = com.mysql.jdbc.Driver
A1.C1.UserName = root
A1.C1.Password = qwerty
A1.C1.PrimaryKeyRegExp = PK
A1.C1.NextPrimaryKeySQL = SELECT id FROM Student WHERE (id >= PK+1) ORDER BY
id
A1.C1.SelectSQL = SELECT * FROM Student WHERE id = PK

### filter pre mail

A1.F1.ClassName = org.openadaptor.adaptor.standard.AliasingPipeSegment
A1.F1.Type1 = ANY Student
A1.F1.Type1.Alias1 = meno meno
A1.F1.Type1.Alias2 = priezvisko priezvisko
A1.F1.Type1.Alias3 = rok_nastupu rok_nastupu

### konektor Sink pre mail

A1.C2.ClassName = org.openadaptor.adaptor.mail.MailSink
A1.C2.DOStringWriter=org.openadaptor.dostrings.DelimitedStringWriter
A1.C2.To = slnce@localhost
A1.C2.From = openadaptor@localhost
A1.C2.Mailhost = localhost
A1.C2.StaticSubject = true
A1.C2.FieldDelimiter = 32
A1.C2.Subject = novy student

### filter pre JMS

A1.F2.ClassName = org.openadaptor.adaptor.standard.ArraySplitterPipe

### konektor Sink pre JMS

A1.C3.ClassName = org.openadaptor.adaptor.jms.JMSSink
A1.C3.Queue.JndiProvider = iiop://158.195.16.196:3700
A1.C3.Queue.JndiFactory = com.sun.appserv.naming.S1ASCtxFactory
A1.C3.Queue.Factory = baltovaConnectionFactory
A1.C3.Queue.Subject = baltovaQueue
```

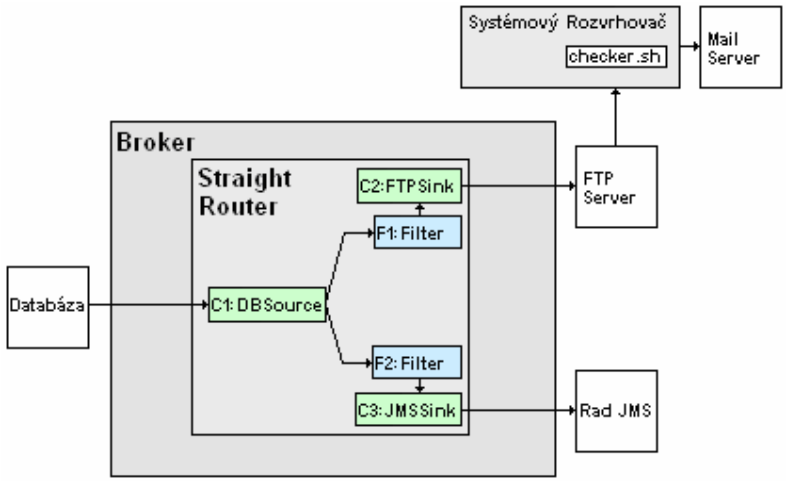
C. Riešenie použitím produktu Proteus

I. Priebeh inštalácie, požiadavky produktu

Inštalácia prebiehala bez komplikácií podľa návodu na inštaláciu. Proteus používa lokálnu premennú `$CLASSPATH`. V tejto premennej musia byť zadefinované všetky knižnice, ktoré produkt používa. Pre JMS sme použili vlastnú knižnicu, ktorú stačilo pridať do tejto premennej.

II. Popis riešenia

V tomto prípade sme museli mierne modifikovať riešenie príkladu. Proteus totiž nemá konektor pre elektronickú poštu. Ako alternatívu sme zvolili umiestnenie súboru, ktorý obsahuje údaje elektronickej správy na server FTP. Poslanie elektronickej pošty je realizované jednoduchým skriptom, ktorý je spúšťaný systémovým rozvrhovačom. Riešenie pozostáva z jedného komponentu „Source“ pre databázu, dvoch konektorov „Sink“ pre JMS a FTP a dvoch filtrov pre výstupné konektory. Toto riešenie ilustruje nasledovný obrázok.



Obrázok 9: rozloženie komponentov v „Brokeri“ produktu Proteus

Router v pravidelných intervaloch pristupuje cez rozhranie „DBSource“ k údajom v databáze. Pokúša sa načítať práve jeden záznam z databázy, ktorý ešte nebol spracovaný. V tomto prípade sme do databázy pridali novú tabuľku „`Pomocna`“, ktorá obsahuje stĺpec „`student_id`“, kde Proteus uchováva identifikátory všetkých spracovaných záznamov tabuľky „`Student`“. Pri úspešnom načítaní záznamu aktualizuje záznam v databáze (pridá identifikátor tohto záznamu do pomocnej tabuľky). Prijaté údaje smeruje na dva výstupy a to filtre pre rozhrania FTPSink a JMSSink. Po transformovaní sú údaje zapísané do súboru cez rozhranie „FTP“ a poslané do radu JMS cez rozhranie „JMSSink“.

Proteus neobsahoval vhodné filtre pre naše riešenie, preto sme si tieto filtre doprogramovali. Komponent DBSource spolupracuje s mapovacím komponentom, ktorý mapuje výsledok selekcie z databázy. V prípade použitia komponentu „StringMapper“ sme však docielili, že načítal do reťazca len hodnotu jedného stĺpca

databázy. Toto nebol uspokojivý výsledok a preto sme použili „DB2DBMapper“, ktorý vrátil množinu ResultSet a filtre sme aplikovali na túto množinu.

III. Spustenie riešenia

Spracovávanie správ sme spúšťali príkazom

```
broker broker1.xml
```

kde `broker1.xml` je názov konfiguračného súboru.

IV. Komplikácie

- Proteus neposkytuje konektory pre elektronickú poštu. Túto komplikáciu sme vyriešili použitím rozvrhovača, ktorý sa v pravidelných intervaloch pokúša načítať definovaný súbor, a ak tento súbor existuje, odošle elektronickú poštu a súbor vymaže.
- Pri čítaní z databázy nedokázal Proteus do reťazca načítať hodnotu viac ako jedného stĺpca databázy. Neposkytoval ani filtre, ktoré by dokázali transformovať načítanú množinu ResultSet. Tieto filtre sme preto doprogramovali.
- Pri zapisovaní do súboru môžeme určiť, koľko záznamov sa má do neho zapísať. Ak sme nastavili túto hodnotu väčšiu, ako bolo reálne prijatých záznamov, tak Proteus neuzavrel súbor a nezapísal ho na disk. Urobil to až po prijatí dodatočných záznamov. Na prvý pohľad to vyzeralo ako komplikácia, pretože nás zaujímalo, či nestratí načítané záznamy v prípade ukončenia. Po preskúmaní sme zistili, že záznamy ukladá do dočasného súboru. Pri spustení sa vždy pozrie, či tento súbor existuje.
- V dokumentácii je povedané, že komponent StraightRouter môže smerovať správu len na jeden výstup. Nie je to pravda. Môže mať definovaných viac výstupov a na každý výstup smeruje jednu kópiu správy.

V. Konfiguračné súbory

1. Konfiguračný súbor produktu Proteus: `broker1.xml`

```
<?xml version="1.0"?>
<broker>
  <source name="C1">
    <class>com.infoscape.proteus.db.DBSource</class>
    <driver>com.mysql.jdbc.Driver</driver>
    <url>jdbc:mysql://localhost/slnce?user=root&amp;password=qwerty
    </url>
    <sql>
select S.meno, S.priezvisko, S.adresa, S.rok_nastupu, S.id FROM
Student as S WHERE S.id NOT IN (SELECT student_id FROM pomocna)
    </sql>
    <mapper>
      <class>com.infoscape.proteus.mapper.DB2DBMapper</class>
    </mapper>
    <updatesql>
insert into pomocna (student_id) values (?)
    </updatesql>
    <keyPositions>
```

```
      <position>
        <source>5</source>
        <target>1</target>
      </position>
    </keyPositions>
  </source>
    <sink name="C2">
    <class>com.infoscape.proteus.ftp.FTPSink</class>
    <server>localhost</server>
    <user>slnce</user>
    <password>slniecko</password>
    <tmpDir>/home/slnce/ftpdata</tmpDir>
    <putMask>mail01</putMask>
    <batchSize>20</batchSize>
  </sink>
  <sink name="F1">
   <type>filtered</type>
   <sink>ftp1</sink>
   <filter>
     <class>com.infoscape.proteus.filter.Moja</class>
   </filter>
  </sink>
  <sink name="C3">
    <type>jms</type>
    <contextFactory>com.sun.appserv.naming.S1ASCtxFactory</contextFactory>
    <connectionFactory>baltovaConnectionFactory</connectionFactory>
    <url>iiop://158.195.16.196:3700/</url>
    <queue>baltovaQueue</queue>
  </sink>
  <sink name="f2">
   <type>filtered</type>
   <sink>F2</sink>
   <filter>
     <class>com.infoscape.proteus.filter.Moja1</class>
   </filter>
  </sink>
  <router>
    <class>com.infoscape.proteus.router.StraightRouter</class>
    <source>db1</source>
    <sink>fltftp1</sink>
    <sink>fltjms1</sink>
  </router>
</broker>
```

2. Filter pre transformovanie údajov pre FTP: `F1.java`

```
import java.util.*;
import java.sql.ResultSet;

public class F1 implements Filter {

  public Object filter(Object msg) throws Exception {
    ResultSet msg1 = (ResultSet)msg;
    return msg1.getString("Meno") + ' ' +  msg1.getString("priezvisko") + '
' + msg1.getString("rok_nastupu");
  }
}
```

3. Filter pre transformovanie údajov pre JMS: `F2.java`

```
import java.util.*;
import java.sql.ResultSet;

public class F2 implements Filter {
```

```
  public Object filter(Object msg) throws Exception {
    ResultSet msg1 = (ResultSet)msg;
    return "<Student>\n<id>" + msg1.getString("id")+ "</id>\n<meno>" +
msg1.getString("meno") + ' ' + msg1.getString("priezvisko") +
"</meno>\n<adresa>" + msg1.getString("adresa") + "</adresa>\n<rok
_nastupu>" + msg1.getString("rok_nastupu") + "</rok_nastupu>\n</Student>";
  }
}
```

4. Skript, ktorý realizuje odosielanie elektronickej pošty na základe výskytu súboru: `checker.sh`

```
#!/bin/bash
#
FILES=(`ls /home/proteus|grep mail01`)
for FILE in ${FILES[*]}; do
        cat /home/proteus/$FILE|mailx -s proteus slnce@localhost
        echo "sent $FILE"
        rm -f /home/proteus/$FILE
done
```

5. Príslušný záznam v rozvrhovači crontab

```
* * * * /home/proteus/scripts/./checker >>/home/proteus/scripts/checker.log
```

D. Riešenie použitím produktu xBus

I. Priebeh inštalácie, požiadavky produktu

Inštalácia prebiehala presne podľa pokynov bez akýchkoľvek komplikácií. xBus nepoužíva lokálnu premennú `$CLASSPATH`. Všetky knižnice, ktoré využíva, musia byť skopírované do adresára `XBUS_HOME/lib`, kde `XBUS_HOME` je adresár, v ktorom je nainštalovaný produkt xBus.

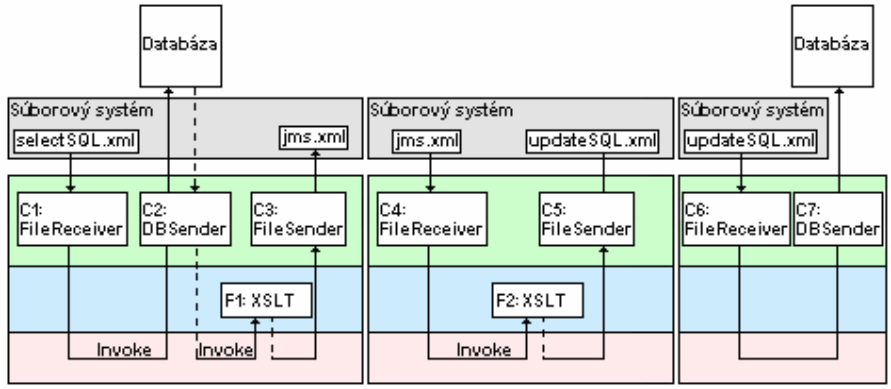
II. Popis riešenia

Produkt xBus nepoužíva konektor „Source“ pre databázu. Preto sme použili medzikrok. Iniciátor selekcie z databázy je súbor umiestnený v lokálnom súborovom systéme, ktorý obsahuje príkazy SQL v požadovanom formáte XML.

Výsledok selekcie bol znova dokument XML. Formát tohto dokumentu nespĺňal naše požiadavky, preto sme pre jeho skonvertovanie použili transformáciu XSLT (`XBUS_HOME/xsl/F1.xsl`). Pri čítaní údajov zo súboru vo formáte XML bola potrebná schéma XML (`XBUS_HOME/Schema/student.xsd`). Nad takto popísaným dokumentom bolo možné aplikovať ľubovoľné transformácie. Použili sme spolu tri transformácie:

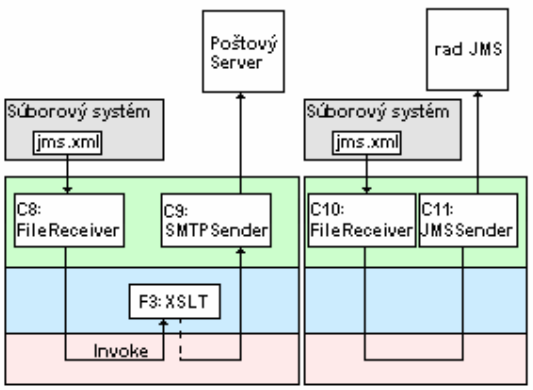
- F1: výsledky z databázy sme transformovali do formátu XML, ktorý zodpovedá formátu pre rad JMS
- F2: vytvorenie príkazov pre aktualizovanie záznamu v databáze,
- F3: vytvorenie elektronickej správy.

Postup pri spracovávaní ilustrujú nasledovné obrázky:



Obrázok 10: postup spracovávania produktom xBus (1):

načítanie údajov z databázy, zapísanie nového príkazu SQL a aktualizovanie záznamu v databáze



Obrázok 11: postup spracovávania produktom xBus (2):

odoslanie údajov prostredníctvom elektronickej pošty a poslanie údajov do radu JMS

III. Spustenie riešenia

Spracovávanie správ sme spúšťali príkazom

```
receiversingle C1 C4 C6 C8 C10
```

kde parametre sú názvy konektorov „Source“.

xBus postupne aktivoval tieto konektory. Spracovávanie druhého začalo po úspešnom ukončení prvého a podobne. Po úspešnom ukončení spracovávania konektorov, produkt ukončil spracovávanie.

To znamená, že jedno spustenie začalo načítaním jedného záznamu z databázy a ukončilo sa po úspešnom spracovaní tohto záznamu, ktoré zahŕňalo odoslanie údajov prostredníctvom elektronickej pošty, odoslanie údajov do radu JMS a aktualizovanie záznamu v databáze (poznačením, že bol úspešne prečítaný).

Aby sme zaručili pravidelné spustenie spracovávania, použili sme rozvrhovač, ktorý spúšťa tento produkt v definovaných intervaloch.

IV. Komplikácie

- Načítavaný dokument XML musel byť popísaný schémou XML, čo predstavuje prácu navyše.
- Chybové správy, ktoré sa vyskytli pri spracovávaní, nie vždy zodpovedali reálnej chybe.
- Transformácia XSLT nefungovala, ak transformačný dokument nedefinoval aspoň jednu značku XML v novom dokumente. Preto je upravený formát elektronickej správy nasledovne:
  ```
  <Údaje>meno priezvisko rok_nastupu</Udaje>
  ```

V. Konfiguračné súbory

1. Konfiguračný súbor produktu xBus: `standard.conf`

```
#-----------------------------------------------------------------------
# Moj: Riesenie modeloveho prikladu
#-----------------------------------------------------------------------

#--- Definitions of interfaces -------------------------------------

###- Nacitanie suboru, ktory obsahuje v dokumente XML prikazy SQL

System_C1_Message=SimpleTextMessage
System_C1_Receiver=FileReceiver
System_C1_Filename=$XBUS_HOME$/sample/selectSQL.xml
System_C1_FinalResolution=Preserve
System_C1_OnEmpty=Ignore
System_C1_OnError=Preserve
System_C1_Journal.Send=true

###- Selekcia z databazy

System_C2_Message=XMLMessage
System_C2_Sender=DatabaseSender
System_C2_DBConnection=MojDB
System_C2_Journal.Send=true

###- Transformacia, ktora ulozi vysledok z databazy v pozadovanom formate

System_F1_Message=XMLMessage
System_F1_Sender=PingSender
System_F1_Journal.Send=true

###- Zapisanie suboru, ktory obsahuje vysledok z databazy v dokumente XML

System_C3_Message=SimpleTextMessage
System_C3_Sender=FileSender
System_C3_Filename=$XBUS_HOME$/sample/jms.xml
System_C3_ConflictResolution=Overwrite
System_C3_Journal.Send=true

###- Nacitanie suboru, ktory obsahuje vysledok z databazy v dokumente XML

System_C4_Message=XMLMessage
System_C4_Receiver=FileReceiver
System_C4_Filename=$XBUS_HOME$/sample/jms.xml
System_C4_FinalResolution=Preserve
System_C4_OnEmpty=Ignore
System_C4_OnError=Rename
System_C4_XMLSchema=moja.xsd
System_C4_Journal.Send=true

###- Transformovanie dokumentu XML do podoby pre potreby DatabaseSenderu

System_F2_Message=XMLMessage
System_F2_Sender=PingSender
System_F2_Journal.Send=true

###- Zapisanie noveho dokumentu XML s prikazmi SQL

System_C5_Message=SimpleTextMessage
System_C5_Sender=FileSender
System_C5_Filename=$XBUS_HOME$/sample/updateSQL.xml
System_C5_ConflictResolution=Overwrite
System_C5_Journal.Send=true
```

```
###- Nacitanie prikazov SQL

System_C6_Message=SimpleTextMessage
System_C6_Receiver=FileReceiver
System_C6_Filename=$XBUS_HOME$/sample/updateSQL.xml
System_C6_FinalResolution=Preserve
System_C6_OnEmpty=Ignore
System_C6_OnError=Preserve
System_C6_Journal.Send=true

###- Aktualizovanie zaznamu v databaze

System_C7_Message=XMLMessage
System_C7_Sender=DatabaseSender
System_C7_DBConnection=C7
System_C7_Journal.Send=true

###- Nacitanie vysledkov z databazy pre poslanie elektronickej posty

System_C8_Message=XMLMessage
System_C8_Receiver=FileReceiver
System_C8_Filename=$XBUS_HOME$/sample/jms.xml
System_C8_FinalResolution=Preserve
System_C8_OnEmpty=Ignore
System_C8_OnError=Rename
System_C8_XMLSchema=student.xsd
System_C8_Journal.Send=true

###- Transformovanie dokumentu XML pre zaslanie posty

System_F3_Message=XMLMessage
System_F3_Sender=PingSender
System_F3_Journal.Send=true

###- Poslanie elektronickej posty

System_C9_Message=SimpleTextMessage
System_C9_Sender=SMTPSender
System_C9_Host=localhost
System_C9_Subject=Novy_student
System_C9_ToAddress1=slnce@localhost
System_C9_FromAddress=xbus@localhost
System_C9_FromName=xbus
System_C9_User=xbus@localhost
System_C9_Journal.send=true
System_C9_Password=xbus

###- Nacitanie vysledkov z databazy pre JMS

System_C10_Message=SimpleTextMessage
System_C10_Receiver=FileReceiver
System_C10_Filename=$XBUS_HOME$/sample/jms.xml
System_C10_FinalResolution=Preserve
System_C10_OnEmpty=Ignore
System_C10_OnError=Rename
System_C10_Journal.Send=true

###- Poslanie dokumentu XML do radu JMS

System_C11_Message=SimpleTextMessage
System_C11_Sender=MQSender
System_C11_Queuename=baltovaQueue
#System_C11_connection=MQ
System_C11_Journal.send=true

#--- Definitions for routing ----------------------------------------
```

```
#nacitanie z db, zapisanie do suboru
Router_C1_Invoke1=C2
Router_C1_Invoke2=F1
Router_C1_Invoke3=C3

#vytvorenie SQL prikazu
Router_C4_Invoke1=F2
Router_C4_Invoke2=C5

#poznacenie zaznamu v db
Router_C6_Invoke1=C7

#poslanie mailu
Router_C8_Invoke1=F3
Router_C8_Invoke2=C9

#poslanie do queue
Router_C10_Invoke1=C11

#--- Definitions for transforming ----------------------------------

XSLTStylesheet_C1_F1=F1.xsl
XSLTStylesheet_C4_F2=F2.xsl
XSLTStylesheet_C8_F3=F3.xsl

#-----------------------------------------------------------------------
# Definitions for connections
#-----------------------------------------------------------------------

DBConnection_C2_Driver=com.mysql.jdbc.Driver
DBConnection_C2_URL=jdbc:mysql://localhost:3306/slnce
DBConnection_C2_MaxRows=1
DBConnection_C2_User=root
DBConnection_C2_Password=qwerty
DBConnection_C2_AutoCommit=false

DBConnection_C7_Driver=com.mysql.jdbc.Driver
DBConnection_C7_URL=jdbc:mysql://localhost:3306/slnce
DBConnection_C7_MaxRows=1
DBConnection_C7_User=root
DBConnection_C7_Password=qwerty
DBConnection_C7_AutoCommit=false

Connection_MQ_QueueConnectionFactory=baltovaConnectionFactory
Connection_MQ_URL=iiop://158.195.16.196:3700
Connection_MQ_ContextFactory=com.sun.appserv.naming.S1ASCtxFactory

#-----------------------------------------------------------------------
# Some basic definitions
#-----------------------------------------------------------------------

#--- Tracing -----------------------------------------------------------

#Trace_Levels: 4=DEBUG, 3=INFO, 2=WARN, 1=ERROR, 0=ALWAYS
Base_Trace_Level=3
Base_Trace_Tracer=ConsoleTrace
#Base_Trace_Tracer=FileTrace
Base_Trace_MaxLength=999999
Base_Trace_Filename=trace.log

#--- Journaling --------------------------------------------------------

Base Journal Implementation=FileJournal
Base_Journal_MessageLength=100
```

2. Dokument XML, ktorý obsahuje príkaz SQL SELECT: `selectSQL.xml`

```
<skuska>
  <Statement>SELECT S.id, S.meno, S.priezvisko, S.adresa, S.rok_nastupu FROM
Student AS S WHERE S.id NOT IN (SELECT student_id FROM pomocna) </Statement>

</skuska>
```

3. Dokument XML, ktorý je vytvorený filtrom F2: `updateSQL.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<Skuska>
<Statement>INSERT INTO pomocna (student_id) VALUES ('5')</Statement>
</Skuska>
```

4. Schéma popisujúca dokument XML: `student.xsd`

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs='http://www.w3.org/2001/XMLSchema'>

 <xs:element name="Student">
  <xs:complexType>
   <xs:sequence>
     <xs:element ref="id" minOccurs='1' maxOccurs='unbounded'/>
     <xs:element ref="meno" minOccurs='1' maxOccurs='unbounded'/>
     <xs:element ref="adresa" minOccurs='1' maxOccurs='unbounded'/>
     <xs:element ref="rok_nastupu" minOccurs='1' maxOccurs='unbounded'/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
<xs:element name="id" type='xs:string'/>
<xs:element name="meno" type='xs:string'/>
<xs:element name="adresa" type='xs:string'/>
<xs:element name="rok nastupu" type='xs:string'/>
</xs:schema>
```

5. Transformácia medzi dokumentom XML, ktorý obsahuje výsledok selekcie z databázy a dokumentom XML pre rad JMS: `F1.xsl`

```
<?xml version ="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method='xml' version='4.0' encoding='ISO-8859-1' indent='yes'/>

<xsl:template match="/">
<xsl:for-each select="MojDB/Result/Record">
  <Student>
    <id><xsl:value-of select="S.id"/></id>
    <meno><xsl:value-of select="S.meno"/> <xsl:value-of
select="S.priezvisko"/></meno>
    <adresa><xsl:value-of select="S.adresa"/></adresa>
    <rok_nastupu><xsl:value-of select="S.rok_nastupu"/></rok_nastupu>
  </Student>
</xsl:for-each>
</xsl:template>
</xsl:stylesheet>
```

6. Transformácia dokumentu XML do dokumentu XML, ktorý obsahuje aktualizačný príkaz pre databázu: `F2.xsl`

```
<xsl:transform version = '1.0'
xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>

<xsl:template match="Student">
  <Skuska>
    <Statement>INSERT INTO pomocna (student_id) VALUES ('<xsl:value-of
select="id" />')</Statement>
  </Skuska>
</xsl:template>

</xsl:transform>
```

7. Transformácia dokumentu XML do formátu elektronickej správy: `F3.xsl`

```
<xsl:transform version = '1.0'
xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>

  <xsl:template match="Student">
    <Novy_student>
      <xsl:value-of select="meno" />
      <xsl:value-of select="rok_nastupu" />
    </Novy_student>
  </xsl:template>

</xsl:transform>
```

8. Skript, ktorý realizuje spúšťanie spracovávania správ produktom xBus: `runner`

```
#!/bin/bash
receiversingle.sh C1 C4 C7 C8 C10
```

9. Príslušný záznam v rozvrhovači crontab

```
* * * * * /home/xbus/scripts/./runner >> /home/xbus/scripts/runner.log
```