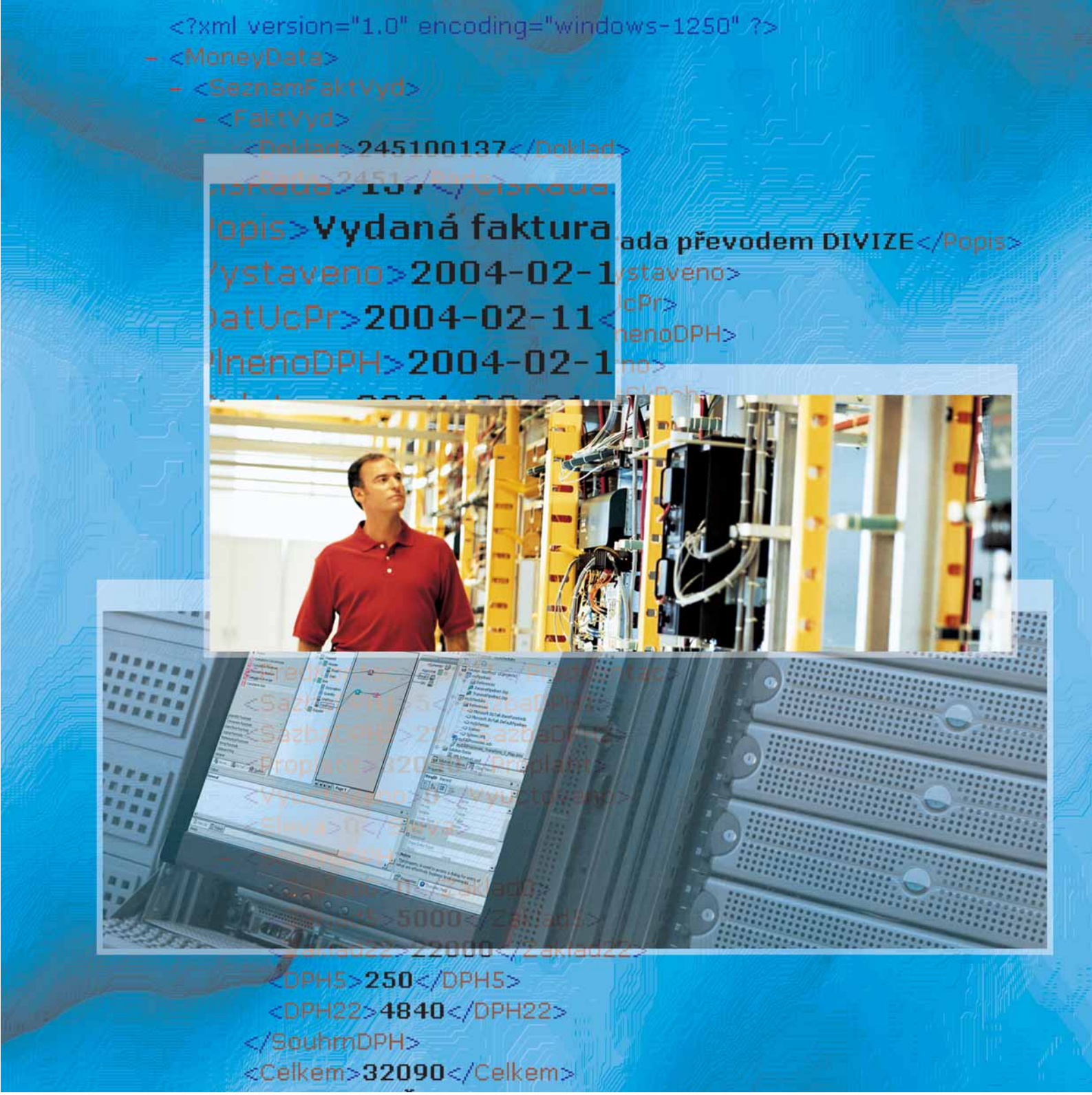


Michael Juřek

Moderní integrace aplikací

Microsoft .net

- 1. Úvod – integrace?
- 2. Moderní koncepce integrační architektury
- 3. Komunikace a přenos zpráv
- 4. Transformace a routování zpráv
- 5. Správa procesů
- 6. Analytické monitorování
- 7. Klienti – interakce s uživatelem
- 8. Místo závěru
- Praktická cvičení

Microsoft®

Věnováno Šárince

Obsah:

Kapitola 1:
Úvod – integrace?

1.1 Proč (ne)číst tuto brožuru

Přesnými statistickými průzkumy bylo zjištěno, že většina čtenářů odborné literatury nečte text na první stránce. Je to možná škoda, neboť text na první stránce vám často pomůže ušetřit několik hodin zbytečně promrhaného drahocenného času anebo vás naopak může motivovat k větší trpělivosti nad textem, který může rozhodnout o úspěchu či neúspěchu důležitého projektu. Dočetli jste až sem? Skvěle!

Tato brožura se zabývá moderními přístupy k integraci aplikací v heterogenním prostředí. Je určena pro softwarové architekty, řídící pracovníky v IT, vedoucí projektů a zkušenější vývojáře. Snaží se klást důraz především na koncepce a metodické postupy při integraci aplikací. Nevěnuje se prázdnému teoretizování, není naplněna marketingovým balastem a není ani podrobnou technickou příručkou. Soustředí se zejména na integraci aplikací uvnitř firmy či organizace (EAI, *enterprise application integration*). Integrace mezi různými subjekty (B2B, *business-to-business*) je zmiňována pouze okrajově – je méně častá a technologicky podobná, liší se spíše v důrazu na bezpečnost a právní aspekty vztahu. V jednotlivých kapitolách se snaží popsat základní metodiku a postupy pro jednotlivé roviny integrace aplikací. Většina kapitol je doplněna praktickým příkladem a nástinem implementace příslušné funkčnosti za pomoci produktu BizTalk Server 2004, podrobný návod k reprodukování praktických příkladů najdete v příloze B. Text by ale měl být užitečný i bez vazby na tento produkt – jako zdroj inspirace, doporučených postupů a jako přehled aktuálních technologických trendů.

Na první stránce se též zpravidla děkuje. Tímto děkuji svému kolegovi Miloši Sobotkovi za přečtení rukopisu. Též děkuji všem ve svém okolí za trpělivost, kterou se mnou v době sepisování brožury měli.

Pokud jste se po přečtení předchozích odstavců rozhodli brožuru odložit, nezoufejte. Budiž vám útěchou, že brožura je zdarma. Také ji můžete předat některému ze svých kolegů či kolegyň, odvážnější muži ji mohou dát manželce nebo přítelkyni k narozeninám. Pokud se rozhodnete, že brožura skončí v koši, rád bych se přimluvil za tříděný recyklovaný odpad.

Pokud jste naopak získali chuť pokračovat ve čtení, udělali jste mi radost, o které se pravděpodobně nikdy nedozvím. A tak alespoň slibuji, že se budu snažit psát text s nadhledem a lehkou ironií, aby nebyl tak „suchý“ jako technické texty bývají a aby byl zajímavý a čtivý. Dejte si ovšem také pozor, aby vás text příliš nepohltil, máte přece také rodinu, partnera/partnerku, koníčky a záliby, občas je vhodné si zasportovat a mít teplou stravu a … život je přece tak krátký. Nebo není?

1.2 Malý pohled do historie

Bylo nebylo – pamětníci mohou tento odstavec přeskočit. Před mnoha lety, zhruba v druhé polovině minulého století začaly firmy a organizace používat počítače. První počítače byly velké rozměry i cenou. Dovolit si je mohly pouze největší podniky a vzhledem k velkým nákladům na pořízení, provoz i vývoj aplikací sloužily pouze pro omezené množství úloh, zpravidla takových které prováděly rutinní opakované výpočty jako třeba výpočet mezd zaměstnanců. V té době mohli potřebu integrace aplikací předvídat pouze opravdoví vizionáři – žádná integrace totiž nebyla nutná. Typický podnik či organizace vlastnil jeden velký systém, který byl centrálně spravován, podléhal velmi tuhé disciplíně pro provoz aplikací (tehdy nazývaných joby) i pro jejich vývoj. Aplikace pracovaly nad jednotnou datovou základnou používající technologie někde mezi děrným štítkem a pevným diskem a všechny aplikace fungovaly na víceméně stejném principu a technologickém základě. Nakonec pár nekomentovaných zkratek pro pamětníky – JSEP, SMEP, OS/390, Cobol, FORTRAN.

Poté přišel technologický zlom, jehož začátek spadá zhruba do druhé poloviny osmdesátých let minulého století, v bývalém Československu s ekonomikou orientovanou na množství vyrobených metrických centů cementu na hlavu ještě o pár let později. V čem tento zlom spočíval? V radikálním zlevnění výpočetní techniky. Nízká cena počítačových systémů zkombinovaná s dostupností moderních vývojových nástrojů přinesla výrazné rozšíření záběru a spektra výpočetně zpracovávaných úloh a zpravidla též živelný a nekoncepční rozvoj počítačového prostředí. Výpočetní střediska postupně transformovaná v IT oddělení zpravidla nedokázala odolat (bezesporu oprávněným) tlakům ostatních oddělení na co nejrychlejší implementaci systémů. Během poměrně krátké doby tak došlo k velkému roztříštění dosavadní jednotnosti prostředí. Ve většině velkých firem byla nasazena řada různých operačních systémů, databázových technologií, programovacích jazyků, vývojových prostředí, komponentových technologií, protokolů atd. Šlo tomuto vývoji zabránit? Téměř jistě ne. Masové využívání výpočetní techniky přineslo v pravém slova smyslu dosud probíhající revoluci ve fungování ekonomiky – význam této doby jednou posoudí historikové. A jak už to u revolucí bývá zvykem, i tato revoluce na závěr přinesla účastníkům velké vystřízlivění a burzovním hráčům navíc trpké zklamání. Rozhodně za ni můžeme být rádi. Je podhoubím pro oblast integrace aplikací a troufnu si tvrdit, že mně i části čtenářů této brožury dala práci a ještě

dlouho jim ji dávat bude, obzvláště pokud si jejím čtením rozšíří svoje vědomosti. Cítíte se již dostatečně motivováni číst dále?

1.3 Důvody pro integraci

Proč je vlastně poptávka po integraci aplikací? Motivačních faktorů je celá řada a z velké většiny jsou dány ekonomickými faktory, umocněnými současným velkým tlakem na snižování a kontrolu výdajů v IT oblasti:

- Snaha o zachování investic – bylo by jistě lákavé zahodit dosavadní heterogenní a roztříštěné aplikace a nahradit je jednou úžasnou integrovanou aplikací se vzorovou architekturou a skvělou vývojovou disciplínou. Nicméně tento fundamentalistický přístup je v praxi nereálný – vyžaduje si příliš mnoho času a prostředků a navíc by vedl ke znehodnocení investic a projektů, které s různou mírou spokojenosti fungují a se kterými jsou spjaty profesní i manažerské kariéry mnoha lidí.
- Úspora lidských zdrojů – roztříštěnost systémů často vede k velké pracnosti jejich obsluhy, vysokým nárokům na zaškolení koncových uživatelů, značnému podílu monotónní lidské práce s tendencemi k vysoké chybovosti a řadě dalších nežádoucích efektů.
- Kontrola nákladů – IT rozpočty jsou dnes pod velkým drobnohledem, a tak není divu, že převažuje snaha o centralizaci výpočetních zdrojů do jednoho místa tak, aby byly jasně dány hranice a vazby mezi aplikacemi. Tím jsou dobře měřitelné jak jejich technické provozní charakteristiky, tak i finanční náročnost. Snaha o návrat ke „zlaté době“ centrálního zpracování dat ve výpočetním centru složeném z levných zaměnitelných komponent je více než zřejmá, ať už se skrývá pod marketingovým slovem *grid* anebo jakýmkoliv jiným.
- Nutnost rychlé implementace obchodních procesů – zostřující se konkurenční prostředí anebo třeba nová legislativa nutí firmy a organizace k rychlému zavádění nových postupů do praxe, přičemž se zpravidla předpokládá využití existujících aplikací a systémů pro tyto nové úkoly. Praktické zavedení nových postupů bývá po organizační i lidské stránce obtížné, pokud je ovšem podpořeno fungujícími integrovanými aplikacemi, šance na úspěšné zavedení změny se výrazně zvyšují.
- Potřeba pružně reagovat na změny vnějších podmínek – málokterý proces zůstává dlouhodobě stabilní, velmi záhy dojde ke změně vnějších podmínek (legislativa, změna situace na trhu, změna vlastníka, výměna managementu apod.), které si vynutí změny existujících procesů. Integrované prostředí s dobrou architekturou dokáže na tyto změny relativně rychle a relativně bezbolestně reagovat, a přispívá tak k efektivnosti a konkurenceschopnosti celé organizace. V zahraniční literatuře se tato schopnost nazývá *business agility,* tento termín však v češtině raději nebudu používat – zavání nedostatkem intelektu nahrazovaného pílí.

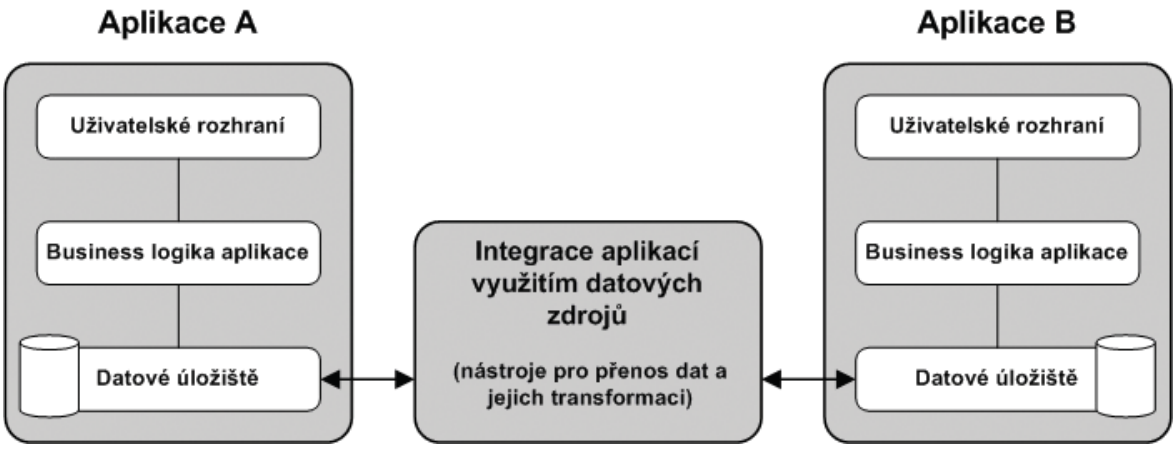
Ač nevzdělán v ekonomické oblasti, napadlo mne celkem pět bodů, ke kterým jistě dokážete přidat i některé vlastní. Tím mám ekonomickou kapitolu za sebou a rád se vrátím k technickým tématům. Jste také rádi?

1.4 Slepé uličky integrace

Upřímně řečeno, vhodnější označení pro tuto část by bylo „Postupy pro integraci, které se hodí v určitých situacích, ale při pokusech o masové nasazení selhávají“. Bohužel jsem podlehl rozšířenému názoru, že titulek musí být krátký, stručný a hlavně musí zaujmout.

Integrace na datové úrovni

Jeden ze zdánlivě dobrých způsobů integrace spočívá ve své podstatě v ignorování existence aplikací. Většina jich totiž funguje víceméně stejně – výsledkem jejich činnosti jsou data uložená v relační databázi, přičemž všechny běžně používané relační databáze fungují na víceméně stejných principech. Vzhledem k tomu, že většina aplikací se bez prostudování dokumentace jeví jako černé skříňky (mnohé i po prostudování dokumentace), je obcházení aplikace a přímá práce s daty velmi lákavá (viz obrázek 1.1). Většinou se přitom používají datové pumpy *(ETL, extraction-transformation-loading),* databázové replikace, export/import a podobné postupy.



Obrázek 1.1: Schéma integrace na datové úrovni

Jde o docela dobré a rychlé taktické řešení, které poměrně rychle a bezbolestně přinese dílčí úspěchy. Problém spočívá ve velmi omezené škálovatelnosti tohoto řešení. V okamžiku, kdy počet různých datových zdrojů přesáhne číslo dva anebo je v rámci zdroje nutné pracovat s větším množstvím tabulek (což lze obojí považovat téměř za jisté), vzniká velmi neprůhledná a špatně udržovatelná struktura. Dalším problémem je možnost konfliktů – pokud určitá data nemají *autoritativní* (někdy nazývaný též *kanonický)* zdroj, je nutné vymýšlet složitou logiku pro řešení možných konfliktů aktualizace dat. Navíc přechod na novou verzi aplikace často znamená radikální změny ve schématu databáze, čímž může být dosavadní vložená práce výrazně znehodnocena. Často se též nelze vyhnout zabudování částí aplikační logiky do transformačního procesu, což je ošidné z hlediska přehlednosti, koncepčnosti a udržovatelnosti řešení. Tím nechci říct, že tento přístup je vždy špatný – hodí se velmi dobře například pro vytváření operačních datových skladů, kde je velký objem dat, je jasný jejich primární zdroj a můžeme si dovolit nezbytnou neaktuálnost *(latenci)* dat v datovém skladu. Pokud ale chceme mít integrační řešení flexibilní, koncepční a pracující v téměř reálném čase, nelze integraci s využitím datových zdrojů v žádném případě doporučit.

Integrace na úrovni uživatelského rozhraní

Integrace na úrovni uživatelského rozhraní je přesně opačným extrémem. Zde aplikace naopak respektujeme jako nebezpečné černé skříňky a hledáme si co nejdelší „klacek“, kterým bychom se jich dotýkali, a tím je klávesnice a obrazovka. Při integraci na úrovni uživatelského rozhraní de facto nahrazujeme koncového uživatele obsluhujícího aplikaci automatizovaným postupem, který čeká na nápovědu od aplikace a reaguje na ni simulací uživatelského vstupu – v anglicky psané literatuře se tento postup nazývá *screen scraping* (viz obrázek 1.2).

Obrázek 1.2: Schéma integrace na úrovni uživatelského rozhraní

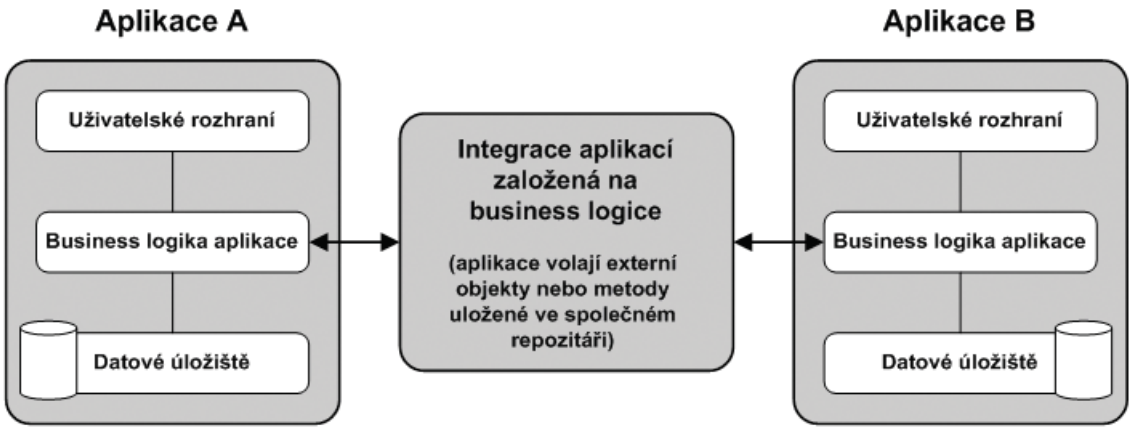
V určitých situacích je tento postup výhodný. Pokud je aplikace již velmi stará, není k dispozici zdrojový kód a dodavatel aplikace již dávno zanikl, jde často o jediný možný postup jak eliminovat nákladnou lidskou práci

potřebnou při předávání údajů mezi aplikacemi (obzvláště, jde-li o pokročilého uživatele schopného zvládnout operaci Kopírovat/Vložit). V drtivé většině situací je ale tento postup naprosto neadekvátní – jde opravdu pouze o krátkodobé nekoncepční řešení, které je kriticky závislé na neměnnosti aplikace, tedy něčem, co lze jenom těžko předpokládat.

Jinou kapitolou je dnes moderní integrace uživatelského rozhraní v portálech. Zde se ovšem jedná spíše o sjednocení uživatelského rozhraní a jeho přiblížení koncovému uživateli, nikoliv o výměnu dat mezi integrovanými aplikacemi. Jde jistě o moderní a racionální trend, které s popisovanou problematikou úzce souvisí, ale není teď pro nás tím hlavním.

Integrace na úrovni obchodní logiky

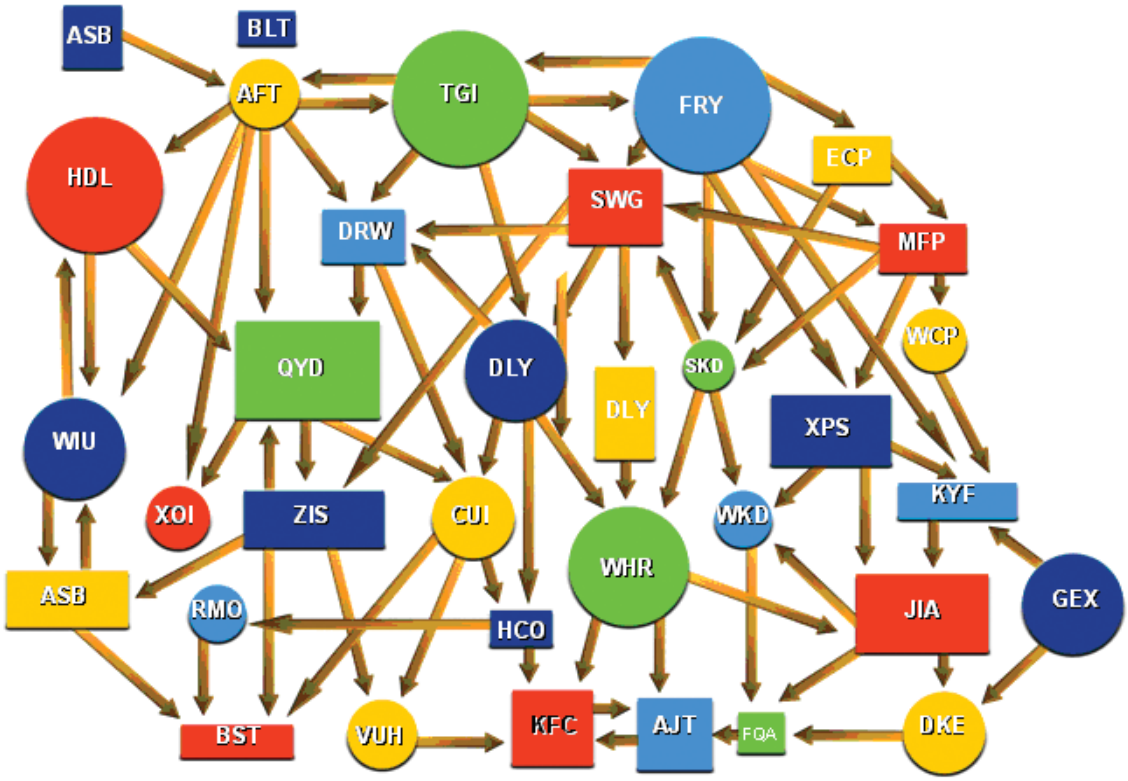
Střední cestou, která ovšem v tomto případě není zlatá, je integrace na bázi komponent obchodní logiky. Spočívá v integraci prostřednictvím vzájemného volání mezi objekty (komponentami) ve střední vrstvě aplikací (viz obrázek 1.3), jde v podstatě o „drátování“ jednotlivých aplikací mezi sebou.



Obrázek 1.3: Schéma integrace na úrovni obchodní logiky

Ačkoliv tento přístup na první pohled vypadá poměrně koncepčně, je jeho praktická realizace z mnoha důvodů velmi obtížná:

- Vyžaduje si zásah do aplikací, což s sebou často přináší nastudování velkého množství dokumentace a prozkoumávání bílých míst, složitou koordinaci s autorem či dodavatelem aplikace, problémy s podporou vzniklého řešení a další.
- Vzájemná závislost a těsná vazba aplikací. Vzhledem k tomu, že dochází ke vzájemnému volání aplikací, stává se dostupnost a správná funkce volající aplikace do velké míry závislou na všech volaných aplikacích – tento nedostatek lze řešit používáním asynchronního volání, což je ale velmi pracné a často pouze jednoúčelové a komplexní řešení, které musí být podporováno řadou servisních aplikací, které celý systém komplikují. Vzájemná závislost a těsná vazba jsou též brzdou aktualizace aplikací – změna v aplikaci si vyžaduje analýzu dopadu změn v ostatních aplikacích, implementaci změn a znovuotestování všech závislých aplikací
- Syndrom N^2 – integrace na úrovni obchodní logiky vede k růstu složitosti s počtem možných vazeb mezi aplikacemi, přičemž tento počet roste s kvadrátem počtu integrovaných systémů (viz obrázek 1.4). Přidání každé další integrované aplikace je tak vždy pracnější a nákladnější než té předchozí – rozhodně ne dobrá strategická vyhlídka.
- Heterogenita – ačkoliv by si to všichni uživatelé přáli, nikdy nedošlo k většinovému používání jediné objektové a komponentové technologie. Ani DCOM/COM+ od Microsoftu, ani komponentové technologie z rodiny J2EE, ani žádná nastupující a „tentokrát už zaručeně ta pravá“ technologie nedosáhly a pravděpodobně nikdy nedosáhnou nadkritického rozšíření. Proto je heterogenita technologií střední vrstvy všudypřítomnou a obtížně překonatelnou realitou – ještě obtížněji než se může na první pohled zdát. Existují sice různé nástroje pro překonání technologických rozdílů (tzv. *bridge*), ale tyto zpravidla nesplňují očekávání zákazníků a jsou navíc často produkovány malými firmami s nejasnou podporou i budoucností. Nejambicióznějším krokem na tomto poli byl pokus o zavedení standardu CORBA. Přestože projekt zaznamenal dílčí úspěchy, dny jeho slávy jsou sečteny z obvyklých důvodů – nadměrná složitost, vysoká pořizovací cena i implementační náklady, nekompatibilita a nedostatečná podpora klíčových výrobců softwaru.



Obrázek 1.4: Syndrom N² při integraci aplikací

Integrace na úrovni ...

Takže si to shrňme: Většina aplikací se skládá ze tří vrstev, z nichž žádná nepředstavuje vhodnou bázi integraci aplikací, a to v zásadě ze stejného důvodu – přílišné křehkosti a velmi špatné schopnosti reagovat na změny. Pesimista by došel k závěru, že žádný přístup k integraci aplikací nefunguje tak jak by se očekávalo a měl by do značné míry oprávněný pocit, že není daleko od pravdy. Optimista by však postřehl, že ještě není se čtením u konce, takže musí existovat ještě nějaký další způsob. O čem by jinak byl zbytek této brožury?

Kapitola 2:

Moderní koncepce integrační architektury

2.1 Těsná vazba – mor integrace

V předchozí části jsme probrali tři možné způsoby integrace aplikací – na úrovni datového zdroje, obchodní logiky a uživatelského rozhraní. Všechny tři způsoby trpí jedním zásadním neduhem – předpokládají sdílení interních záležitostí jednotlivých aplikací. Například pokud chci integrovat aplikaci A s aplikací B na úrovni datové vrstvy, musí aplikace A znát databázové schéma aplikace B a stává se tak na něm závislou. Důsledkem těchto přístupů je pak propletenec obtížně dokumentovatelných závislostí mezi aplikacemi nazývaný těsná vazba *(tight coupling)*.

Moderní integrační architektura jde přesně opačným směrem a preferuje tzv. volnou vazbu *(loose coupling),* jejímž základním východiskem je důraz na minimální množství informací sdílených mezi aplikacemi, přičemž sdílení interních implementačních záležitostí aplikace (databázové schéma, obchodní logika, uživatelské rozhraní) je pak přísně zapovězeno. Tento přístup je formalizován v koncepci architektury orientované na služby *(service oriented architecture, SOA),* které bude věnována celá příští kapitola.

Nejde přitom o nic nového. Podobná myšlenka vznikla již před řadou let ve formě výměny zpráv *(messaging)* mezi aplikacemi. Její nejviditelnější implementací je rodina produktů *MQSeries* od IBM. Přes nesporný úspěch a technickou vyspělost nedošlo k masovému rozšíření této technologie, neboť cena za typickou implementaci byla pro většinu poptávajících příliš vysoká. Důvodem této nákladnosti byla absence podpory na mnoha platformách a dále chybějící standard pro formát vyměňovaných zpráv v prostředí s vysokou heterogenitou. Živou vodou pro reinkarnaci koncepce SOA se stalo až přijetí a rozšiřování XML a webových služeb – série standardů pro komunikaci a výměnu dat v heterogenním prostředí. Vzhledem k dostupnosti a vyzrávání těchto technologií na všech myslitelných platformách se laťka dostala výrazně níž a její přeskočení se stalo realistickým pro mnohem širší okruh zájemců.

Co by tedy měla nabízet ideální integrační architektura? Odpověď snadno získáme popřením nevýhod vyjmenovaných v předchozí kapitole:

- Minimální zásah do existujících aplikací – pokud je při implementaci vyžadován zásah do existujících aplikací, měl by být co nejmenší. Pokud je změna aplikace nezbytná, měly by veškeré změny být lokalizovány na jednom místě (změny „vedle“ aplikace, nikoliv „do“ aplikace).
- Nezávislost a autonomie aplikací – každá aplikace by měla respektovat autonomii ostatních aplikací. Komunikace mezi aplikacemi by měla probíhat způsobem, který nevyžaduje žádné znalosti interní funkce druhé aplikace. Nedostupnost jedné aplikace by měla mít minimální dopad na funkci celého systému, v ideálním případě by neměla zasáhnout žádnou jinou aplikaci *(failure isolation),* přičemž s touto nedostupností se *a priori* počítá. Bezpečnostní selhání či prolomení jedné aplikace neohrozí další *(security isolation)*. Komunikace mezi aplikacemi by měla být natolik obecná, aby mohla zůstat zachována i při změně nebo náhradě některé z aplikací – v ideálním případě by se tato změna nebo náhrada neměla mimo dotyčnou aplikaci jakkoliv projevit.
- Škálovatelnost s počtem aplikací – přidání další aplikace do integračního prostředí by v ideálním případě mělo připomínat připojení dalšího spotřebiče do zásuvky. Složitost a nákladnost tohoto připojení by měla záviset pouze na složitosti a komplexnosti nové aplikace, nikoliv na existujících již integrovaných aplikacích. Jednotlivé aplikace je tak možné rozvíjet odděleně bez větších ohledů na zbytek prostředí.
- Dodržování standardů – aplikace by měly dodržovat existující standardy, které platí *de facto* a zároveň *de iure,* tzn. nejde o „standardy“ protlačované jednotlivými výrobci softwaru ani o standardy, které nikdy nepřekročí rámec zajímavého akademického cvičení. Dodržováním standardů se integrace zjednodušuje tím více, čím různorodější a heterogennější je stávající prostředí.

Vyjmenované body jistě připomínají ráj na zemi. Lidé narození před rokem 1980 si na jeden slibovaný ráj na zemi vzpomínají a dobře také ví, jak skončil, a jistě už předem pojali nedůvěru. Proto je seriózní přiznat, že ideální stav je opravdu nedosažitelný. Architektura orientovaná na služby vás k ráji integrace může alespoň přiblížit. Upřímně řečeno, budete zřejmě vděční i za to, že vás co nejvíce vzdálí od očistce integrace. Věříte mi alespoň, že pro vás nechystám peklo integrace?

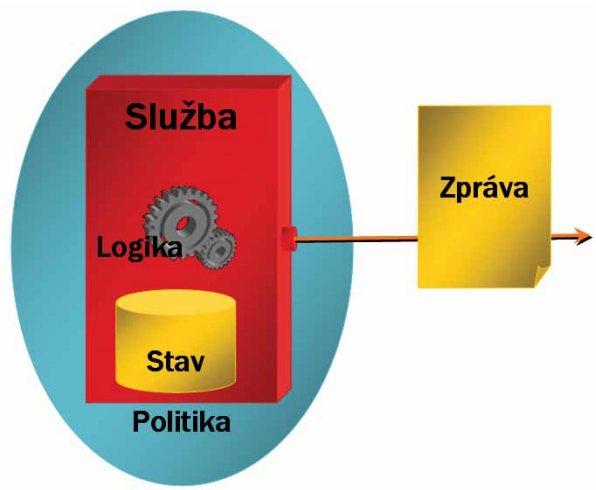
2.2 Koncept architektury orientované na služby (SOA)

Jak jsme si již řekli, integrace na bázi uživatelského rozhraní, datového zdroje ani obchodní logiky nejsou tím pravým ořechovým. Nejlepším dosud objeveným modelem je integrace založená na službách. Schéma této integrace je na obrázku 2.1. Pravděpodobně se vám schéma zdá poněkud alibistické a stále se ptáte: Na jaké úrovni aplikací je tedy SOA postaveno – na uživatelském rozhraní, obchodní logice anebo střední vrstvě? Odpověď je velmi jednoduchá, ale nechtějte po mně, abych prozradil pointu této kapitoly. Pojďme se nyní podívat na vybrané základní myšlenkové konstrukce architektury založené na službách.

Obrázek 2.1: Schéma integrace založené na službách

Služba

Služba *(service)* je autonomní část software, která implementuje logiku v podobě kódu, spravuje svůj stav, komunikuje prostřednictvím zpráv, je řízena politikou a je dostupná po síti. Tolik suchá definice, kterou můžeme znázornit obrázkem 2.2. Jednoduše řečeno, služba je autonomní část distribuované aplikace, od které můžeme očekávat splnění přesně definované úlohy nebo úloh. Ve světě dnešních aplikací může takovou úlohou být například zadání faktury do ERP (enterpise resource planning) systému, aktualizace kontaktních údajů v CRM *(customer relationship management)* databázi, založení nového zaměstnance v HR *(human resources)* systému.



Obrázek 2.2: Vizuální definice služby

Typickou službou ve světě lidí může být například získání výpisu z trestního rejstříku. Rejstřík trestů je instituce, na které lze ukázat nejdůležitější vlastnost služby, a tou je její autonomie. Získání výpisu z trestního rejstříku je přesně definovaná operace – přinesete vyplněný formulář k přepážce tzv. určeného úřadu (obecní úřad, úřad městské části apod.) a výpis pak dostanete zpravidla poštou, při osobní návštěvě rejstříku v Praze též osobně. Co se děje za přepážkou vás nezajímá. Mělo by vám být jedno, zda se změnily vnitřní postupy, interní pravidla anebo bylo vyměněno 90% zaměstnanců. Pokud služba funguje tak jak má, budete v konečném důsledku spokojeni. Pokud by nefungovala,

budete si stěžovat pouze na fakt, že nefunguje – nebude vás zajímat proč a už vůbec nebudete chtít být zatahováni do jejích interních záležitostí.

Stav

Důvodem pro existenci služby je zpravidla stav *(state),* který spravují. Stav lze nejčastěji chápat jako vzletné označení pro informace v databázi, které jsou jádrem většiny dnešních aplikací. Pro ERP aplikaci je stavem seznam všech vydaných a přijatých faktur, účetní kniha, pohledávky, závazky apod. Pro CRM aplikaci je stavem databáze zákazníků, telefonátů, stížností, nabídek. Pro HR aplikaci je stavem seznam všech zaměstnanců, absolvovaných školení a vyplacených mezd.

Služby si střeží svůj stav – je jejich interní záležitostí a důvodem existence. Stav je měněn pomocí vnitřní logiky. Jednou ze zodpovědností služby je udržovat stav konzistentní anebo se o to alespoň co nejvíce snažit – často k tomu používají transakce. Stav by měl být trvalý *(durable),* tj. měl by být odolný vůči co nejvíce „nepředvídatelným“ okolnostem, jako je např. výpadek proudu, havárie disku, vylití Vltavy z břehu, pád asteroidu na Zemi, vítězství komunistů ve volbách. Úkolem služby je zpřístupňovat fragmenty svého stavu externím subjektům, případně je modifikovat na základě externě přijatých požadavků – samozřejmě v souladu se svojí vnitřní logikou.

Stav je takzvaným autoritativním zdrojem dat a jedině služba má interní přístup k tomuto zdroji informací. V okamžiku, kdy jsou data zpřístupněna mimo hranice služby, je nutné je považovat pouze za obraz *(snapshot)* skutečných informací. V našem příkladě s trestním rejstříkem bude stavem pravděpodobně databáze všech zápisů v rejstříku. Výpis je pak pouze jakýmsi potvrzením o stavu databáze v určitý časový okamžik, se skutečným aktuálním stavem již nemá žádnou spojitost. To si bohužel uvědomují též instituce o tento výpis žádající a budou vždy klást podmínku maximálního stáří výpisu. Jakkoliv se tento fakt může zdát triviální, má v SOA dalekosáhlé důsledky. Je nutné mít neustále na paměti fakt, že jakékoliv informace získané od služby jsou pouze obrazem skutečných informací ve více či méně vzdálené minulosti.

Zpráva

Služby komunikují se svým okolím pomocí zpráv *(messages)*. Zprávy se tak v integračních scénářích stávají nejdůležitější součástí. Služba je ve své podstatě navenek definována zprávami, které přijímá a vysílá. Komunikace pomocí zpráv může být buď jednosměrná (služba pouze přijímá zprávy a neodpovídá na ně anebo naopak aktivně vysílá zprávy bez vnějšího podnětu) anebo obousměrná (služba přijme zprávu a očekává se, že na ni odpoví, anebo naopak vyšle zprávu a očekává odpověď odněkud ze svého okolí).

Příkladem zprávy ve světě aplikací může být například nově zadaná faktura, existující faktura, příkaz k úhradě, potvrzení o přijetí platby, upomínka o zaplacení dlužné částky, aktualizované kontaktní údaje zákazníka, změna příjmení zaměstnankyně (nebo zaměstnance, ať nikoho nediskriminujeme). V příkladu s trestním rejstříkem pak služba čeká na příchozí zprávu – souhrn údajů na žádosti o výpis – a po určité době odpovídá odchozí zprávou – výpisem z trestního rejstříku, což je opět souhrn definovaných údajů (platí zde, že čím kratší zpráva, tím lépe, ale toto tvrzení nelze zobecnit).

Služba očekává předem definovaný a zcela jasný formát zprávy, který musí splňovat předepsané náležitosti. Jako formát zprávy se dnes nejčastěji používá standard XML a pro popis zprávy standard XSD, ještě se jim budeme věnovat dále. Nejsou ovšem podmínkou, zpráva může být například ve formátu EDI podle příslušné normy nebo může být textovým souborem ve formátu popsaném v manuálu aplikace. Pokud si můžete vybrat, volte vždy XML vzhledem k bezkonkurenční podpoře napříč platformami.

Ať už tak či onak, nezávislost služeb a jejich „nezájem“ o vše, co je za jejich hranicemi má i svoje důsledky. Služba nemůže předem předpokládat, že zprávy, které obdrží budou korektní. Musí neustále hlídat všechny příchozí zprávy a kontrolovat jejich správnost. Nesprávné zprávy musí vyloučit ze zpracování a předvídatelným způsobem signalizovat chybu, nejlépe tak, aby tento stav byl zachycen používanými monitorovacími aplikacemi. Toto chování se nazývá principem zdravé nedůvěry *(healthy distrust)* a je jedním z pilířů autonomie služeb podobně jako je pasová kontrola pilířem autonomie států. A se špatně vyplněným formulářem výpis z trestního rejstříku též nedostanete.

Vzhledem k tomu, že správný návrh zpráv je klíčovým faktorem úspěchu integračního projektu, budeme mu věnovat celou kapitolu.

Kontrakt

Kontrakt *(contract)* určuje, jakým způsobem je možné se službou komunikovat. Určuje, jaké zprávy jsou přijímány a jaké jsou odesílány, určuje v jakém jsou pořadí, v jakém formátu, jaké náležitosti musí zprávy splňovat, jaký protokol se bude při komunikaci používat, jak bude ověřena autentičnost volajícího apod. Kontrakt je tedy definicí způsobu, jakým je služba ochotná komunikovat se svým okolím. Je určitou formou dohody mezi službou a tím kdo ji volá, přestože jde zpravidla o dohodu jednostranně vnucenou provozovatelem služby.

V případě trestního rejstříku je kontrakt zhruba takovýto: „Volající musí mít originální formulář žádosti o výpis z trestního rejstříku, který je opatřen kolkem ve výši 50 Kč. Tento formulář je osobně nebo prostřednictvím osoby vybavené ověřenou plnou mocí doručen Rejstříku přímo anebo prostřednictvím stanoveného úřadu. Výstupem je výpis z trestního rejstříku na originálním tiskopise opatřený podpisem a kulatým razítkem.“ I zde jde samozřejmě o jednostrannou dohodu mezi občanem a úřadem. Pokud ji občan respektuje, je obsloužen, pokud ne, je jeho žádost odmítnuta a není zpracovávána.

Pro technologické účely je vždy vhodné kontrakt přesně, strukturovaně a jednoznačně popsat. Nejznámějším standardem pro popis kontraktu je WSDL, který přesně říká, jaké operace jsou webovou službou podporovány, jaké formáty XML zpráv jsou používány a v jaké sekvenci, jaký protokol se používá pro komunikaci, jak jsou předávána metadata zpráv pomocí protokolu SOAP apod. Standardům kolem webových služeb je věnována celá příští kapitola.

Politika

Služby se řídí politikou *(policy)*. Politika určuje, jakým způsobem smí být služba po technické i organizační stránce používána a pravidla pro její funkci. Jednou z důležitých částí politiky jsou bezpečnostní zásady jako nutnost šifrování zpráv, povolené šifry včetně délky klíčů, způsob autentizace volajícího, kontrola neporušenosti zpráv, auditování využívání služby apod. Další velkou skupinou jsou operační charakteristiky služby – jaká znaková sada se používá, používaný transportní protokol nebo technologie, zda je logováno využívání služby, jak je zajištěna vysoká dostupnost, zda je omezen čas dostupnosti služby *(service window),* jaká je očekávaná doba odezvy služby apod.

Politika je často formálně vyjádřena v servisním ujednání *(SLA, service level agreement),* které má pro vztahy mezi více organizacemi často charakter přílohy právních dokumentů. SLA se ovšem často vytvářejí i ve formě interních dokumentů uvnitř organizace, kdy určují pravidla pro funkci služby a odstraňují potenciální třecí plochy mezi jednotlivými odděleními a týmy uvnitř firmy či organizace.

Pro ilustraci opět případ trestního rejstříku, kde bychom politiku mohli popsat zhruba takto: „Formulář o výpis z rejstříku musí být předložen osobně nebo prostřednictvím osoby pověřené notářsky ověřenou plnou mocí, která svoji autentičnost prokáže občanským průkazem nebo jiným dokladem totožnosti. Výpis rejstříku je doručen doporučeným dopisem s doručenkou. Žádosti jsou přijímány pouze v pracovní dny od 8:00 do 16:00 hodin. Žádosti doručené osobně jsou vyřizovány v došlém pořadí bezodkladně s kapacitou X žádostí za hodinu. Žádosti doručené poštou mají garantovanou dobu odpovědi X týdnů“. Je pravděpodobné, že Rejstřík nabízí též jiné služby jako např. zápis do rejstříku, které nejsou veřejně přístupné a řídí se jinou politikou (po této službě by též zřejmě byla malá poptávka veřejnosti, na rozdíl od služby vymazání záznamu z rejstříku).

Tajenka z minulého čísla

Po definicích se nyní můžeme vrátit k původní otázce: Na jaké úrovni aplikací je tedy SOA postavena – na uživatelském rozhraní, obchodní logice anebo střední vrstvě? Znáte již odpověď? Správná odpověď zní takto: je v zásadě úplně jedno, jakou úroveň tvůrce rozhraní použije. Pro zpřístupnění služby asi nejčastěji použije určitou formu nadstavby nad obchodní logikou, nicméně klidně může servisní vrstvu postavit i přímo nad databází anebo naopak může služební vrstva překrývat uživatelské rozhraní a používat *screen scraping,* je to interní rozhodnutí zcela v kompetenci tvůrce služby. V souladu s filozofií SOA by tedy správná odpověď na otázku mohla znít: A k čemu to vlastně potřebujete vědět?

2.3 Standardy XML a webových služeb

Od počítačové nepaměti se objevují snahy o standardizaci formátů dat, přičemž tyto snahy vždy narážely na partikulární zájmy jednotlivých velkých hráčů. Zatímco v některých oblastech se podařilo prosadit proprietární *de facto* standardy silou (např. DOC v oblasti textových dokumentů nebo PDF pro výstup z pre-press zpracování), v oblasti výměny dat a komunikace v heterogenním prostředí se dlouho žádný standard nemohl ustavit. Přestože některé technologie dosáhly solidní úrovně rozšíření, jako např. EDI pro výměnu dat nebo CORBA pro komunikaci mezi objektovými technologiemi, žádná z nich nedosáhla univerzálního rozšíření – zpravidla z důvodů přílišné složitosti a malé flexibility, což v konečném důsledku vedlo k vysokým nákladům na jejich implementaci. Právě dlouhé hladovění je prapříčinou obrovského úspěchu a rozšíření technologií kolem XML a webových služeb, protože jde o standardy relativně jednoduché, skutečně heterogenní, technologicky nepředpojaté a bez silné vazby na firmy a zájmové skupiny v IT průmyslu. Pojďme si nyní provést malé defilé zkratek pro standardy, jimž se v integraci nevyhnete. Musím upozornit, že jde o text celkem nudný, který se nebude číst zrovna hladce (o psaní nemluvě, ale to je jenom můj problém). Přesto považuji základní znalost poslání jednotlivých standardů za užitečnou. Pokud se mnou nesouhlasíte, přeskočte rovnou na kapitolu 2.4 – ušetříte si čas a možná zachráníte i trochu sebevědomí.

XML

XML *(extensible markup language)* je značkovací jazyk, jež se stal standardem pro zápis a výměnu dat v heterogenním prostředí. XML zasáhlo svět aplikací jako uragán a můžeme ho dnes najít téměř všude, přestože jde o velmi mladý standard, který se do fáze doporučení dostal až začátkem roku 1998. Jeho starším bratrem je HTML *(hypertext markup language)* pro vytváření webových prezentací, jejich společným otcem je SGML *(standard generalized markup language)* z roku 1986 a dědou z počítačového dávnověku je GML (*generalized markup language)* z roku 1969. Místo dlouhého teoretizování se podívejme na zjednodušený příklad zápisu kontaktních informací pomocí XML jazyka:

```
<?xml version="1.0"?>
<KontaktniUdaje>
   <Jmeno>
     <Krestni>Karel</Krestni>
     <Prijmeni>Novák</Prijmeni>
   </Jmeno>
   <RodneCislo>7003222315</RodneCislo>
</KontaktniUdaje>
```

Na první pohled je vidět základní výhoda XML formátu – čitelnost hierarchické struktury pro člověka. I při absenci technického vzdělání je možné pochopit jádro sdělení vyjádřeného jako XML. Na druhou stranu jde o formát striktní, který se řídí přesnými pravidly. XML dokument splňující všechna pravidla můžeme označit jako správně vytvořený *(well-formed),* takový dokument je pak relativně snadné programově zpracovat a získat z něj potřebné informace. Vzhledem k tomu, že jde o velmi častou úlohu, existuje přehršel dokonalých a komfortních knihoven pro práci s XML dokumenty, nazýváme je XML parsery. Zajímavým aspektem XML dokumentů je fakt, že tatáž informační hodnota může být vyjádřena různým zápisem, např. vložíme-li mezi značky <*KontaktniUdaje*> a <*Jmeno*> prázdné řádky, informační obsah se nezmění. „Čistý" informační obsah XML dokumentu neboli virtuální strom informací bez doprovodných značek a dalšího balastu se nazývá *XML InfoSet*.

Mají XML dokumenty též nějakou nevýhodu? Napadá mne jediná – přílišná rozvleklost. Například fakt, že faktura byla zaplacena vyjádříme v XML jako <*Zaplaceno*>*true*</*Zaplaceno*>. Exaktně řečeno, k zápisu informace o velikosti 1 bitu jsme potřebovali 27 znaků, které na disku zaberou přinejmenším 27 bytů. Tato nevýhoda je sice výrazně kompenzována stále klesající cenou hardwaru (disková a paměťová kapacita, šířka síťového pásma apod.) a dále možností snížit velikost díky dobré komprimovatelnosti na zlomek původní hodnoty (např. při přenosu prostřednictvím modemu). Nicméně je objektivní ji za nevýhodu označit, jakkoliv se její důležitost postupně snižuje. Jde o daň, kterou platíme za čitelnost a interoperabilitu. A jak už to u daní bývá, můžeme si na její výši sice stěžovat, ale nakonec ji stejně musíme zaplatit. Nebo se můžeme odstěhovat do země s nižšími daněmi, u XML ale dosud není kam.

XSD

XSD neboli *XML Schema Definition* (doporučení W3C z května 2001) je zápis požadavků na obsah XML dokumentu, přičemž schéma samotné je též zapsáno pomocí XML dokumentu. Proces porovnání dokumentu a konstatování jeho (ne)dodržení určitého schématu se nazývá validace. Během validace se nevyhodnocuje, zda dokument splňuje všechny syntaktické požadavky na dobře vytvořený dokument, *a priori* se předpokládá, že jsou splněny a dokument je správně vytvořený *(well-formed)*. Pokud dokument validací projde, označí se za platný *(valid)*. XSD není prvním pokusem o kontrolu sémantiky XML dokumentů, předchozí standardy jako DTD nebo XDR nejsou z různých důvodů vhodné. Co nám tedy XSD nabízí? Není toho málo, namátkou:

- Definici základních datových typů – řetězec, datum, celé číslo a řada dalších
- Možnost definice složitých typů – např. typ *Jmeno* je složen z prvků *Krestni* a *Prijmeni,* přičemž oba jsou typu řetězec
- Kontrola pořadí a vzájemné hierarchie – např. *Jmeno* smí obsahovat pouze prvky *Krestni* a *Prijmeni,* přičemž se musí vyskytovat přesně v tomto pořadí
- Kontrola počtu výskytu jednotlivých prvků – např. *Jmeno* musí obsahovat nejvýše jeden prvek *Krestni* a právě jeden prvek *Prijmeni*.
- Hlídání hranic u měřitelných hodnot jako jsou čísla nebo časové údaje – např. věk musí být vyšší než 18.
- Vynucení určitých hodnot prvku – např. pohlaví musí být buď „muž“ nebo „žena“ (omlouvám se všem, které jsem právě diskriminoval)
- Dodržení délky údaje – např. rodné číslo musí být 9 až 10 znaků, cena musí mít maximálně dvě desetinná místa
- Kontrola formátu pomocí regulárních výrazů – např. email musí obsahovat právě jeden znak zavináče a za ním minimálně jednu tečku

Všechny tyto aspekty se sepíšou do dokumentu, který vytvoří XML schéma, se kterým potom pracují různé integrační nástroje. Vytváření schématu ručně je sice možné, ale není příliš praktické, pro jeho vytváření jsou k dispozici kvalitní nástroje. Podívejme se nyní pro ilustraci na příklad vytvořeného schématu, ve kterém se vám možná podaří vyčíst některé z předchozích požadavků:

```
<?xml version="1.0" encoding="utf-16" ?>
<xs:schema xmlns="http://schemas.microsoft.cz/BrozuraBTS/2004"
targetNamespace="http://schemas.microsoft.cz/BrozuraBTS/2004"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="KontaktniUdaje">
     <xs:complexType>
       <xs:sequence>
             <xs:element name="Jmeno">
                   <xs:complexType>
                         <xs:sequence>
                               <xs:element maxOccurs="1" name="Krestni" type="xs:string" />
                               <xs:element minOccurs="1" maxOccurs="1" name="Prijmeni"
type="xs:string" />
                         </xs:sequence>
                   </xs:complexType>
             </xs:element>
             <xs:element minOccurs="1" maxOccurs="1" name="RodneCislo">
                   <xs:simpleType>
                         <xs:restriction base="xs:string">
                               <xs:minLength value="9" />
                               <xs:maxLength value="10" />
                         </xs:restriction>
                   </xs:simpleType>
             </xs:element>
       </xs:sequence>
     </xs:complexType>
```

```
    </xs:element>
</xs:schema>

Dokument platný podle tohoto schématu pak mùže vypadat následovnì:

<?xml version="1.0"?>
<broz:KontaktniUdaje xmlns:broz="http://schemas.microsoft.cz/BrozuraBTS/2004">
    <Jmeno>
      <Krestni>Karel</Krestni>
      <Prijmeni>Novák</Prijmeni>
    </Jmeno>
    <RodneCislo>7003222315</RodneCislo>
</broz:KontaktniUdaje>
```

Možná jste si všimli jedné velmi důležité maličkosti – definice jmenného prostoru *(namespace)*. Vzhledem k tomu, že každé XML schéma v podstatě určuje typ dokumentu, každý definovaný komplexní typ uvnitř schématu je rovněž samostatným typem a všechny tyto typy jsou označeny pouze jednoslovně (např. *Jmeno*), je více než pravděpodobné, že budou vznikat různé typy stejného jména, ale s úplně jiným smyslem. *Jmeno* může být rovněž jméno auta nebo písničky, což by při integraci v prostředí s více typy dokumentů podle XML schémat různého původu působilo nepřekonatelné problémy. Stejně tak třeba jmenný prostor *http://www.w3.org/2001/XMLSchema* zastoupený ve schématu prefixem *xs* odlišuje element v definici schématu od elementu-prvku v databázi chemických prvků. Jmenné prostory se tudíž starají o jedinečnost a nezaměnitelnost typů ve schématech. Aby byla tato jedinečnost zajištěna, stalo se dobrým zvykem používat pro rozlišení jmenných prostorů zaběhnutá jedinečná jména – registrovaná DNS jména tvůrce schématu. Je vhodné označovat jmenné prostory podle notace *http://DNS.jmeno.ve.vlastnictvi.firmy/Aplikace/Verze*, zde tedy např. *http://schemas.microsoft.cz/BrozuraBTS/2004*, ale jakákoliv jiná konvence zaručující jedinečnost jmenných prostorů ve schématu by fungovala stejně dobře. Povšimněte si, že jde pouze o identifikaci, výsledná URL adresa nemusí vůbec nic obsahovat. Nedalo vám to a vyzkoušeli jste si, zda vám nelžu?

XSLT

XSLT 1.0 *(Extensible Stylesheet Language – Transformations)* byl doporučen W3C v listopadu 1999. Původně byl zamýšlen především jako jazyk pro konverzi XML dokumentů pro potřeby jejich zobrazení v koncových zařízeních, internetových prohlížečích apod. – památkou na toto období je slovo *stylesheet* v jeho názvu. Dnes je tento původní účel čím dál výrazněji zastíňován druhým způsobem použití – konverzí a transformací XML dokumentů odpovídajících různým schématům, což je úloha velmi často se vyskytující prakticky ve všech integračních scénářích.

Vezměme si například výše uvedený dokument pro kontaktní údaje:

```
<?xml version="1.0"?>
<broz:KontaktniUdaje xmlns:broz="http://schemas.microsoft.cz/BrozuraBTS/2004">
    <Jmeno>
      <Krestni>Karel</Krestni>
      <Prijmeni>Novák</Prijmeni>
    </Jmeno>
    <RodneCislo>7003222315</RodneCislo>
</broz:KontaktniUdaje>
```

a představme si, že jej pro účely jiné aplikace potřebujeme převést do následující formátu (odpovídajícímu schématu pro kontaktní údaje v jiné aplikaci):

```
<?xml version="1.0"?>
<broz:CONTACTINFO xmlns:broz="http://schemas.microsoft.cz/BrozuraBTS/2004">
    <UNIQUEID>700322315</UNIQUEID>
    <NAME>Karel Novák</NAME>
</broz:CONTACTINFO>
```

Napsat kód provádějící tuto transformaci s pomocí XML parseru v některém procedurálním jazyce (C, C++, C#, Java, Visual Basic, ...) by nebylo vůbec obtížné, ale mezi vývojáři by šlo o velmi nepopulární úlohu, na kterou byste těžko hledali dobrovolníka. Procedurální jazyky jsou na podobné úlohy velmi těžkopádné a pracné, výsledný kód je velmi špatně čitelný a jeho pozdější úpravy jsou noční můrou. Výsledek by navíc byl problematicky přenositelný a znovupoužitelný mezi platformami. Proto vznikl funkční jazyk XSLT vhodný přesně pro tento typ úloh. Opět jenom pro ilustraci, příklad transformace mezi výše uvedenými dokumenty:

```
<?xml version="1.0" encoding="UTF-16" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
   <xsl:output version="1.0" method="xml" />
   <xsl:template match="/">
     <xsl:apply-templates select="/KontaktniUdaje" />
   </xsl:template>
   <xsl:template match="/KontaktniUdaje">
     <broz:CONTACTINFO xmlns:broz="http://schemas.microsoft.cz/BrozuraBTS/2004">
       <UNIQUEID>
             <xsl:value-of select="RodneCislo" />
       </UNIQUEID>
       <NAME>
             <xsl:value-of select="Krestni" />
             <xsl:text> </xsl:text>
             <xsl:value-of select="Prijmeni" />
       </NAME>
     </broz:CONTACTINFO>
   </xsl:template>
</xsl:stylesheet>
```

Nevím, zda jste schopni se v XSLT šabloně orientovat, jde v podstatě o předpis pro záměnu částí dokumentu jinými definovanými částmi. XSLT toho ale umí mnohem více. Podporuje podmínky, aritmetické, logické i řetězcové funkce a celou řadu dalších rysů. Existují též poměrně komfortní nástroje pro vytváření XSLT šablon. Krátce po ustavení XSLT se dokonce našli fanatici, pokoušející se v XSLT vytvářet kompletní aplikace včetně obchodní logiky a uživatelského rozhraní. Dnes se nezdá, že by se z nich mohl stát významný proud, procedurální jazyky jsou zkrátka na některé úlohy lepší. Každopádně v integračních projektech má XSLT nezastupitelnou roli.

WSDL

Kontrakt určuje, jakým způsobem je se službou možné komunikovat. V příkladu s trestním rejstříkem jsme vystačili se slovním popisem, v oblasti softwaru je kontrakt nutno velmi exaktně specifikovat. Kontrakt je totiž smlouvou mezi volajícím a volaným – má-li jejich komunikace hladce fungovat, je nutné aby obě strany rozuměly všem pojmům a termínům stejně, podobně jako v právní smlouvě (nebo alespoň dobré právní smlouvě). Ve světě webových služeb je definován standard pro exaktní popis kontraktu pomocí speciálního XML dokumentu, který se nazývá WSDL (*Web Services Description Language,* březen 2001), který přesně popisuje způsob, kterým webová služba nabízí svoje rozhraní případným volajícím.

Pojďme se podívat jak WSDL dokument vzorové služby vypadá. Pokud jste již periferním viděním tento dlouhý a složitý dokument zaregistrovali, nelekejte se. Jeho jednotlivé části si později vysvětlíme. Pokud byste se během výkladu ztratili, klidně přeskočte rovnou na další standard, neznalost vnitřností WSDL není velkým handicapem. A hlavně, v praxi nebude nutné podobné dokumenty vytvářet. Dnešní vývojářské nástroje umí automaticky generovat WSDL dokumenty pro služby vytvářené v typických objektových technologiích, jako je *.NET Framework* nebo *Java*. A nyní avizované zastrašení WSDL dokumentem. Rozsekal jsem jej podle jednotlivých sekcí na pět částí, aby jej nebyla příliš velká porce najednou a taky se jim bude v tiskárně lépe sázet

```
<?xml version="1.0" encoding="utf-8" ?>
<definitions xmlns:s1=http://schemas.microsoft.cz/BrozuraBTS/2004
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
```

```
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:s0="http://schemas.microsoft.cz/BrozuraBTS/2004/WSKontaktniUdaje"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
targetNamespace="http://schemas.microsoft.cz/BrozuraBTS/2004/WSKontaktniUdaje"
xmlns="http://schemas.xmlsoap.org/wsdl/">
   <types>
     <s:schema elementFormDefault="qualified"
targetNamespace="http://schemas.microsoft.cz/BrozuraBTS/2004/WSKontaktniUdaje">
       <s:element name="rodneCislo" type="s:string" />
     </s:schema>
     <s:schema elementFormDefault="qualified"
targetNamespace="http://schemas.microsoft.cz/BrozuraBTS/2004">
       <s:element name="KontaktniUdaje" type="s1:KontaktniUdaje" />
       <s:complexType name="KontaktniUdaje">
             <s:sequence>
                   <s:element minOccurs="0" maxOccurs="1" form="unqualified" name="Jmeno"
type="s1:KontaktniUdajeJmeno" />
                   <s:element minOccurs="0" maxOccurs="1" form="unqualified" name="RodneCislo"
type="s:string" />
             </s:sequence>
       </s:complexType>
       <s:complexType name="KontaktniUdajeJmeno">
             <s:sequence>
                   <s:element minOccurs="0" maxOccurs="1" form="unqualified" name="Krestni"
type="s:string" />
                   <s:element minOccurs="0" maxOccurs="1" form="unqualified" name="Prijmeni"
type="s:string" />
             </s:sequence>
       </s:complexType>
     </s:schema>
   </types>
```

Kromě předehry v podobě definice jmenných prostorů v prvku *<definitions>* zde vidíme důležitou sekci s definicí všech typů používaných službou, zpravidla se pro ně používá definice pomocí XSD schémat. Tyto typy mohou být též definovány mimo WSDL dokument a připojeny definicí. V našem případě je definován typ *rodneCislo* jednoduchého typu *string* a složené typy *KontaktniUdaje* a *KontaktniUdajeJmeno* přesně odpovídající výše uvedenému XSD schématu.

```
   <message name="VyhledejKontaktniUdajeSoapIn">
     <part name="rodneCislo" element="s0:rodneCislo" />
  </message>
   <message name="VyhledejKontaktniUdajeSoapOut">
     <part name="VyhledejKontaktniUdajeResult" element="s1:KontaktniUdaje" />
   </message>
   <message name="AktualizujKontaktniUdajeSoapIn">
     <part name="kontaktniUdaje" element="s1:KontaktniUdaje" />
  </message>
```

Ve druhé části jsou definovány jednotlivé zprávy, které služba používá při komunikaci. Každou zprávu tvoří jedna nebo více logických částí popsaných nebo jinak definovány v části *types*. V našem případě je situace přehledná a jednoduchá – naše služba používá tři pojmenované druhy zpráv a každá z nich je tvořena pouze jedním typem. Pokud se vám zdá, že druhá a třetí zpráva jsou totožné, máte pravdu, jednu z definic by bylo možné vynechat. Jde o důsledek použití automatického nástroje, který toto zjednodušení neprovedl. Při automatickém generování je pro každé volání i odpověď funkce vygenerována jedna zpráva, ve které počet částí odpovídá počtu parametrů v kódu příslušné funkce.

```
<portType name="WSKontaktniUdajeSoap">
      <operation name="VyhledejKontaktniUdaje">
        <input message="s0:VyhledejKontaktniUdajeSoapIn" />
        <output message="s0:VyhledejKontaktniUdajeSoapOut" />
      </operation>
      <operation name="AktualizujKontaktniUdaje">
        <input message="s0:AktualizujKontaktniUdajeSoapIn" />
      </operation>
  </portType>
```

Třetí část obsahuje definici typů portů, přičemž každý port se skládá z jedné nebo více operací. Port přibližně odpovídá popisu rozhraní v klasickém komponentovém vývoji. V našem případě obsahuje port dvě operace – *VyhledejKontaktniUdaje* a *AktualizujKontaktniUdaje*. Každá operace je kromě svého jména definována až třemi typy zpráv – *input, output* a *fault*, jejichž úkolem je určit typ zprávy pro vstup (volání), výstup (odpověď) a možný výstup při chybě. Všimněte si v kombinaci s definicí zpráv, že operace *VyhledejKontaktniUdaje* má vstupní zprávu složenou z části *rodneCislo* a výstupní zprávu s jedinou částí *KontaktniUdaje*. Jde zjevně o volání, kterým lze zjistit kontaktní údaje osoby na základě vloženého rodného čísla. Tento způsob komunikace se službou se nazývá *request/response*. Naproti tomu operace *AktualizujKontaktniUdaje* má pouze vstupní zprávu s jedinou částí typu *KontaktniUdaje* a slouží – překvapivě – k aktualizaci kontaktních údajů, přičemž se nečeká žádná odpověď. Tento způsob komunikace se nazývá *one-way*. Teoreticky existují ještě další dva způsoby, a to *solicit/response* (výstup a pak vstup) a *notification* (pouze výstup), ale protože je nejde pomocí protokolu HTTP realizovat, nemají zatím praktické uplatnění.

```
   <binding name="WSKontaktniUdajeSoap" type="s0:WSKontaktniUdajeSoap">
     <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
     <operation name="VyhledejKontaktniUdaje">
       <soap:operation
soapAction="http://schemas.microsoft.cz/BrozuraBTS/2004/WSKontaktniUdaje/VyhledejKontaktniUdaje"
style="document" />
       <input>
             <soap:body use="literal" />
       </input>
       <output>
             <soap:body use="literal" />
       </output>
     </operation>
     <operation name="AktualizujKontaktniUdaje">
       <soap:operation
soapAction="http://schemas.microsoft.cz/BrozuraBTS/2004/WSKontaktniUdaje/AktualizujKontaktniUdaje
" style="document" />
       <input>
             <soap:body use="literal" />
       </input>
     </operation>
  </binding>
```

Část *binding*s je poněkud složitější na vysvětlení. Důvodem její existence je skutečnost, že nestačí pouze definovat typ portu prostřednictvím zpráv, je nutné též popsat způsob přenosu zpráv pomocí konkrétní přenosové technologie – v tomto případě prostřednictvím standardu SOAP navázaného na protokol HTTP. Rovněž lze definovat řadu technických detailů, jakým je např. způsob zabalení zprávy. Pro nás je zajímavý atribut *soapAction* pro operaci, který bude hrát velmi důležitou roli v části věnované standardu SOAP.

```
    <service name="WSKontaktniUdaje">
      <port name="WSKontaktniUdajeSoap" binding="s0:WSKontaktniUdajeSoap">
        <soap:address location="http://localhost/mjurek.EAIdemo_Proxy/WSKontaktniUdaje.asmx" />
      </port>
  </service>
</definitions>
```

Poslední část definuje porty a služby. Port je vazba určité *binding* definice (to je typu portu definovaného zprávami navázaného na určitou přenosovou technologii) s určitou adresou, která slouží pro volání služby. Formát adresy je dán navázanou přenosovou technologií, v případě HTTP protokolu je to adresa webové služby zapsaná jako odkaz. Služba slouží ke sdružování portů do logických celků – služeb. V našem případě jde o celkem zbytečnou komplikaci, neboť máme jedinou službu a jediný port, ale co naplat, standard je standard a musí se dodržet za každou cenu.

SOAP

Teď když víme, jak přesně popsat obecnou službu pomocí standardu WSDL, je čas podívat se na konkrétní příklad realizace služby. WSDL sám o sobě nepředepisuje způsob zabalení příchozích a ochozích zpráv při komunikaci. V 99% případů k jejich zabalení použijeme standard SOAP (*Simple Object Access Protocol,* květen 2000), který je základním kamenem webových služeb. SOAP sám je poměrně obecný a přestože nepředepisuje konkrétní přenosový protokol, v 99% případů se pro přenos SOAP zpráv používá protokol HTTP. Zbylé 1% tvoří zprávy poslané e-mailem, na disketě, děrném štítku, holubí poštou, potrubní poštou nebo jiným způsobem. Vrátíme se tedy raději k protokolu HTTP, jehož používání je v SOAP standardu přesně popsáno. Zajímavým prvkem je HTTP hlavička *SOAPAction,* která službě identifikuje operaci volanou jejím klientem:

```
POST /mjurek.EAIdemo_Proxy/WSKontaktniUdaje.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: 183
SOAPAction: "http://schemas.microsoft.cz/BrozuraBTS/2004/WSKontaktniUdaje/VyhledejKontaktniUdaje"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <rodneCislo
xmlns="http://schemas.microsoft.cz/BrozuraBTS/2004/WSKontaktniUdaje">7003222315</rodneCislo>
  </soap:Body>
</soap:Envelope>
```

V našem případě jde tedy o volání operace *VyhledejKontaktniUdaje,* kde požadavkem je zřejmě vyhledání osoby s rodným číslem *7003222315*. SOAP volání tvoří XML dokument s kořenovým prvkem *soap:Envelope* neboli obálkou SOAP zprávy. Obálka obsahuje hlavičky v části *soap:Header* (zde hlavičky nejsou, proto tato část chybí) a část *soap:Body,* která obsahuje vlastní zprávu, v tomto případě rodné číslo poptávané osoby. Zjednodušeně se dá říct, že tělo zprávy je určeno pro aplikaci, zatímco hlavičky jsou určeny její infrastruktuře. Podívejme se nyní na odpověď služby:

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: 395
```

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <KontaktniUdaje xmlns="http://schemas.microsoft.cz/BrozuraBTS/2004">
      <Jmeno>
        <Krestni>Karel</Krestni>
        <Prijmeni>Novák</Prijmeni>
      </Jmeno>
      <RodneCislo xmlns="">7003222315</RodneCislo>
    </KontaktniUdaje>
  </soap:Body>
</soap:Envelope>
```

Struktura zprávy je v zásadě stejná – obálka obsahuje hlavičky a tělo, přičemž tělem je nyní zpráva obsahující odpověď služby, v tomto případě kontaktní údaje Karla Nováka (jakákoliv podobnost se skutečnými osobami je čistě náhodná).

Jakým způsobem přesvědčit službu, aby komunikovala pomocí protokolu SOAP? Jsou v zásadě tři různé možnosti. Některá prostředí, produkty a aplikace mají podporu webových služeb již přímo v krabici, např. *Microsoft .NET Framework,* novější verze SAPu a další. Pokud aplikace používá novější technologie, je možné použít tzv. toolkity, které danou technologii obalí vrstvou webových služeb – do této kategorie patří např. *Microsoft SOAP Toolkit* pro technologie na bázi COM, *Glue* od *Mind Electric* nebo *WASP* od firmy *Systinet* pro systémy na bázi Javy/J2EE a řada dalších. Konečně třetím, vývojářsky nejnáročnějším způsobem je napsání vlastní komunikace za použití libovolného webového serveru a XML parseru. Vyvarujte se ale bezhlavého zpřístupňování stávajících aplikačních rozhraní metodou nejmenší pracnosti (1:1) – „obalování“ aplikace vrstvou webových služeb je dobrou příležitosti pro restrukturalizaci rozhraní do dlouhodobější strategičtější podoby, aby odpovídala zásadám architektury orientované na služby a byla použitelná i při změnách nebo výměně zpřístupňovaného systému.

UDDI

UDDI (*Universal Description, Discovery, and Integration,* verze 2.0 z července 2001) je specifikace SOAP rozhraní webové služby sloužící jako katalog dostupných služeb, schémat a dalších objektů a informací pro integraci. Původní, dnes již naivně působící představa typického scénáře použití z éry dot-com bubliny byla zhruba takováto: „Firma potřebuje koupit 1 milión šroubků. Kontaktuje veřejný UDDI registr a z něj zjistí adresy všech webových služeb dodavatelů šroubků. Poté je postupně zavolá, aby zjistila, který nabízí nejlepší cenu. Nakonec od nejlepšího z nich prostřednictvím webové služby šroubky objedná.“

Veřejné UDDI registry stále existují, i když jsou spíše na okraji zájmu. Mnohem větší pozornost se upírá k privátním UDDI registrům budovaným uvnitř firem pro jejich interní potřebu (UDDI registr je též součástí Windows Serveru 2003). Privátní UDDI registry slouží dvojímu účelu. Během vývoje mohou sloužit jako katalog dostupných schémat a služeb a tím usnadňovat práci vývojářům a architektům. Druhý účel se nazývá *runtime composition* a představuje určitou virtualizaci služeb a jejich abstrakci od topologie sítě. Jde o podobný princip, jako je virtualizace serverů pomocí jejich DNS jmen – stačí si pamatovat jméno, IP adresa serveru je pak dohledána až za běhu programu v „katalogu“ serverů zprostředkovaném DNS službou. Stejně tak UDDI může sloužit jako katalog skutečných adres služeb v dynamických systémech, na které se volající aplikace odkazují neměnným identifikátorem. Tím lze odstranit závislost volajícího na přesné adrese a umožnit scénáře jako je přesun serveru, převzetí funkce serveru, rozkládání zátěže apod. Z tohoto pohledu se UDDI přisuzuje velká budoucnost. Počkejme si, jaká bude skutečnost, do té doby je vhodné alespoň vědět o existenci této specifikace.

Standardy WS-cokoliv

Webové služby prošly překotným vývojem, jehož nechtěným dítětem se stala určitá dávka nekompatibility mezi jednotlivými implementacemi. Jak k ní mohlo dojít? Hodně zjednodušeně například takto: standard SOAP připouští, aby funkčnost X byla implementována způsoby A a B. Firma ASoft si pro svoji implementaci webových služeb zvolí způsob A, firma BSoft způsob B. Obě firmy jsou tak kompatibilní se standardem SOAP, nicméně jejich produkty spolu

nejsou schopny komunikovat prostřednictvím webových služeb. K zastavení tohoto nepříznivého vývoje byla v roce 2002 založena organizace WS-I, jejímž úkolem je dbát na kompatibilitu webových protokolů. Jejím prvním výtvorem je specifikace WS-I Basic Profile 1.0. Vychází ze standardů XML 1.0, SOAP 1.1, WSDL 1.1 a HTTP 1.1, přičemž silně omezuje svobodu výkladu v jejich implementaci, je velmi striktní a neobsahuje příliš mnoho slov „nebo“ (a to nejenom proto, že je v angličtině). Při vytváření webových služeb je tedy daleko nejbezpečnější a nejkompatibilnější striktní dodržování tohoto profilu.

Další předem předpokládanou závadou webových služeb je jejich nekompletnost. Je to poměrně logické, neboť prvotní myšlenka bohudík nebyla „pojďme udělat standard řešící všechny možné otázky integrace“, ale „pojďme udělat minimalistický standard, na kterém se opravdu všichni dokážeme shodnout“. Rozšíření o další funkce je specifikováno v nových standardech, které využívají chytře otevřených zadních vrátek v podobě SOAP hlaviček. Zjednodušená zpráva se SOAP hlavičkami může vypadat např. takto:

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns … >
    <soap:Header>
      <wsa:Action>CreateCoordinationContext</wsa:Action>
      <wsa:To>http://localhost/TestCoordinator</wsa:To>
      <wsrm:Sequence>
        <wsu:Identifier>http://fabrikam123.com/abc</wsu:Identifier>
        <wsrm:MessageNumber>10</wsrm:MessageNumber>
        <wsrm:LastMessage/>
      </wsrm:Sequence>
      <ds:Signature> … </ds:Signature>
    </soap:Header>
   <soap:Body>
     <rodneCislo xmlns= … >7003222315</rodneCislo>
   </soap:Body>
</soap:Envelope>
```

Jednotlivé standardy tedy definují svoje vlastní hlavičky, které jsou zpracovávány infrastrukturou. Standardy jsou přitom modulární a tudíž vzájemně nezávislé, pro každé nasazení si budeme moci vybrat pouze ty standardy, které potřebujeme.

Nejčastějším steskem nad funkcí základních standardů webových služeb je naprosté ignorování bezpečnosti. Vskutku, webové služby se spoléhají v otázce bezpečnosti na přenosový protokol (typicky HTTPS), o jehož existenci nic neví. Tento způsob bezpečnosti se nazývá *point-to-point* a pro řadu scénářů je zcela dostatečný. Stále více jsou ale vyžadovány pokročilejší funkce, jako např. digitální podpis ověřující autentičnost zprávy a její integritu, šifrování zprávy klíčem, který je znám pouze aplikaci a ne infrastruktuře apod. Tyto problémy řeší standardy z rodiny *WS-Security,* které již mají charakter doporučení agentury OASIS a existuje řada jejich implementací.

Druhou nejčastěji poptávanou funkčností je práce s transakcemi. Ani webové služby ani HTTP protokol nemají žádný způsob pro zajištění integrity v rámci více volání a pro přenos transakčního kontextu mezi nimi, přičemž tato funkčnost je ve světě těsné vazby naprosto obvyklá a je tedy logicky poptávána i pro webové služby. Proto vznikl standard *WS-Coordination* pro koordinaci více volání webových služeb se dvěma scénáři použití: *WS-Transaction* slouží k podpoře „klasických“ krátce trvajících ACID transakcí zpravidla realizovaných transakčními monitory, *WS-BusinessActivity* slouží k provádění dlouhotrvajících transakcí s kompenzujícími operacemi v případě nezdaru celé akce. Je nutno říci, že v době psaní tohoto textu (leden 2003) jde o standardy nezralé, jejichž implementace od Microsoftu a IBM lze považovat pouze za experimentální.

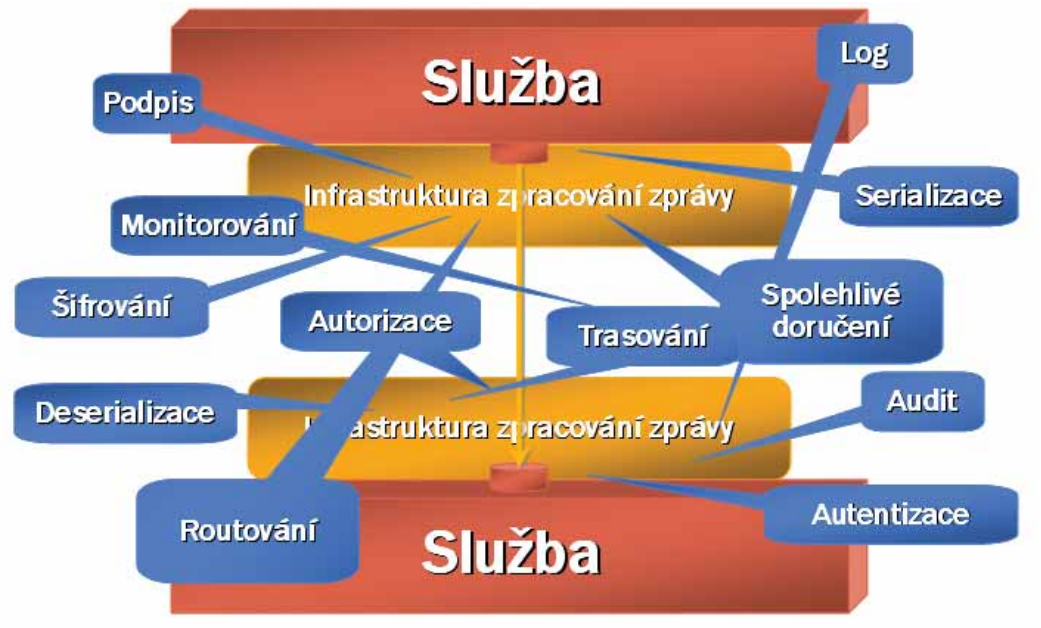
Třetí důležitou chybějící částí je kontrola kvality doručovaných zpráv. Pokud například voláme webovou službu a nedostaneme odpověď, nevíme, zda je to tím, že zpráva s voláním nebyla doručena, anebo zda byla korektně doručena a zpracována, ale někde cestou zpátky se ztratila odpověď – protokol HTTP ani SOAP tyto dvě situace neumí rozlišit. Tato skutečnost nepříjemně zvyšuje požadavky na robustnost implementace služeb, přičemž opět – ve světě integrací s těsnou vazbou se tento problém dá snadno řešit. Proto vznikl standard *WS-ReliableMessaging,* který umožňuje zaručit kvalitu přenosu. Zprávu je tak možno doručit buď bez jakýchkoliv záruk, s garancí doručení alespoň jednou, s garancí doručení právě jednou nebo s garancí doručení právě jednou přesně podle pořadí odeslání. Na závěr ukázka funkce *Kopírovat/Vložit* ve Wordu: Je nutno říci, že v době psaní tohoto textu (leden 2003) jde o standardy nezralé, jejichž implementace od Microsoftu a IBM lze považovat pouze za experimentální.

Oslí můstek

Zdá se vám přechod mezi kapitolami o architektuře a o standardech poněkud nenavazující? Nyní je čas spojit spojit obě kapitoly dohromady krátkou sumarizací. XML je formát pro zprávy předávané mezi službami, přičemž XSD můžeme využít pro kontrolu jejich správnosti a XSLT pro transformaci v případě, že služby poskytují data v různých formátech. WSDL je standard pro popis kontraktu služby komunikující prostřednictvím standardu SOAP. UDDI je katalog služeb, jejich kontraktů, formátů dat a dalších informací. Doplňující standardy WS-I budou sloužit pro infrastrukturní služby na integrační sběrnici nebo brokeru. Zatraceně, co je to integrační sběrnice nebo broker?

2.4 Role integračního brokeru

Architektura orientovaná na služby nabízí vodítka pro vytváření jednotlivých služeb, ale nenabízí pomoc při skládání těchto služeb do většího celku. Pro výběr správného způsobu spojování služeb do komplexního integračního prostředí je nutné uvědomit si v plné šíři složitost problematiky. Předáním zprávy mezi dvěma službami jsme sice teoreticky dosáhli požadovaného výsledku, pro praktické provozování služby v reálném prostředí je to ale zoufale málo. Reálné požadavky na provozování komunikace mezi službami mohou vypadat podobně jako na obrázku 2.3. Je nutno zabezpečit celou řadu důležitých infrastrukturních funkcí od bezpečnostních požadavků (šifrování, autentizace, autorizace) až po provozní záležitosti (monitorování a audit). A právě tyto požadavky jsou určující silou při návrhu celkového uspořádání a architektury integračního prostředí. Projděme si nyní tři základní uspořádání integračního prostředí.

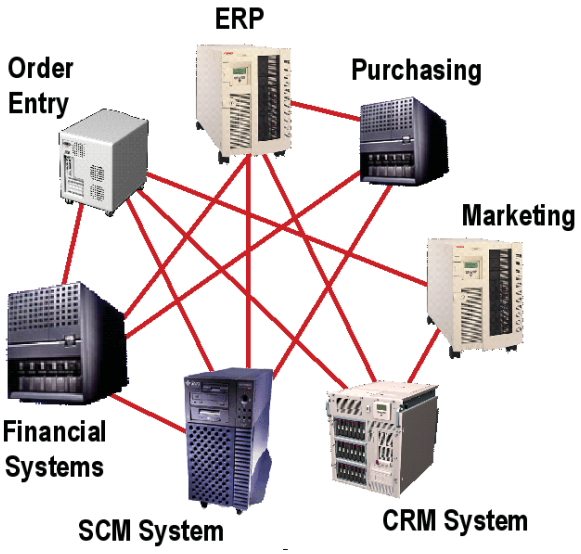


Obrázek 2.3: Infrastrukturní úlohy při komunikaci mezi službami

A. Uspořádání point-to-point

V tomto nejzákladnějším uspořádání jsou jednotlivé systémy propojovány mezi sebou nahodilými vazbami, podobně jako ve schématu na obrázku 2.4. Pokud se vám uspořádání zdá zcela nevhodné, uvedené pouze jako odstrašující příklad před pointou celé kapitoly, máte samozřejmě napůl pravdu. Dalo by se říci, že toto uspořádání přináší řadu taktických výhod, ale též fatální problémy strategického charakteru. Toto uspořádání je často nejsnadnější na prvotní implementaci, je poměrně výkonné bez zbytečné režie a nevyžaduje velkou vstupní investici před dosažením prvních výsledků. Na druhou stranu začíná být brzy nepřehledně složité, je velmi obtížné jej modifikovat a zasahuje výrazně do existujících systémů. Vytváří totiž poměrně těsné vazby a pavučinu obtížně vysledovatelných závislosti mezi jednotlivými systémy. Noční můrou se pak mohou stát dodatečné pokusy o doplnění nezbytných infrastrukturních funkcí, jako je logování veškerých aktivit nebo správa a operační monitorování v provozním prostředí. Požadovanou funkčnost je pak nutné implementovat pro jednotlivé komunikační vazby mezi systémy jednotlivě, což je velmi pracné a nákladné. Nevýhodnost je dána především kvadratickou závislostí nákladů na složitosti, kterou lze připodobnit

k holubí poště. Pokud byste chtěli doručovat poštu pro milión Pražanů prostřednictvím holubů, potřebovali byste 999,999 miliard speciálně vycvičených holubů, na jejichž krmení by nestačil hrubý domácí produkt České republiky. Podobný, i když snad méně dramatický efekt kvadrátu postihuje též integrační model point-to-point, proto jej opravdu nelze (podobně jako holubí poštu) doporučit.

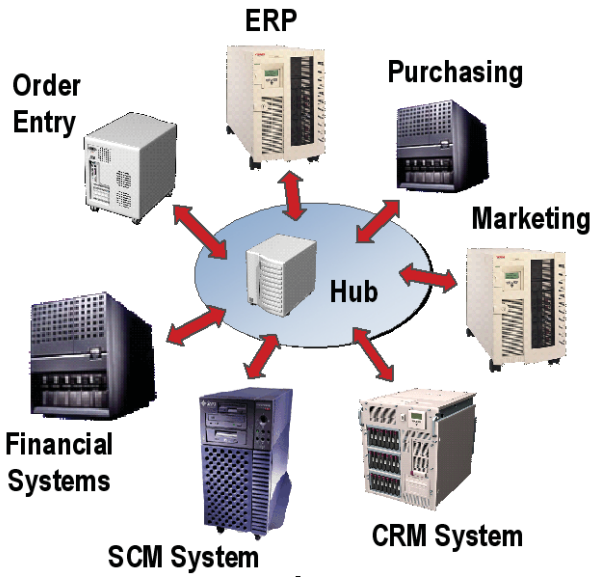


Obrázek 2.4: Uspořádání point-to-point

B. Uspořádání hub-and-spoke

V uspořádání *hub-and-spoke* je mezi jednotlivé systémy vložena dodatečná vrstva v podobě integračního hubu. Integrované systémy pak nikdy nekomunikují mezi sebou přímo, ale vždy prostřednictvím hubu připomínajícího telefonní ústřednu (viz obrázek 2.5). V tomto uspořádání se stává integrace s novými systémy jednodušší, ovšem za cenu vyšší úvodní investice – je třeba navrhnout a uvést do provozu nejprve onen integrační hub, což samozřejmě představuje náklady na software, hardware i lidské zdroje. Odměnou je naopak centralizace mnohých funkcí v podobě hubu, který je prostředníkem mezi systémy a má tudíž naprostou kontrolu nad jejich komunikací. Z toho titulu je možné centrální monitorování, audit, vynucení bezpečnostních požadavků, konverze dokumentů vyměňovaných službami, spolehlivé doručování, překonání výpadku systémů a řada dalších funkcí.

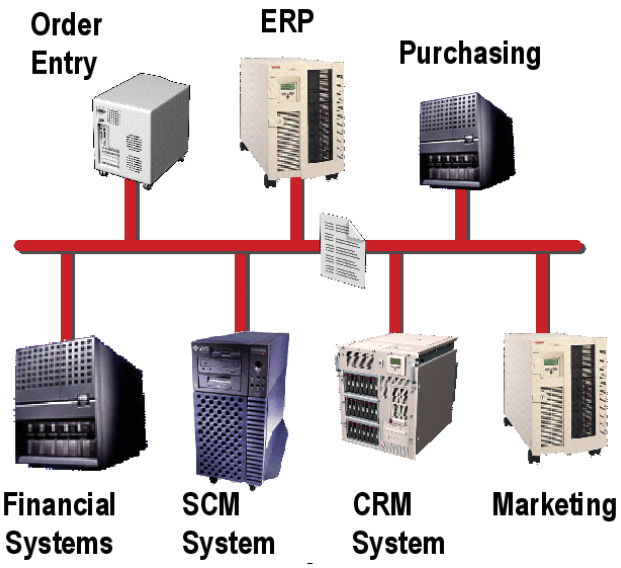
Vazba mezi jednotlivými systémy je zároveň volnější, což znamená menší zásahy do jednotlivých systémů, větší volnost a flexibilitu celého prostředí a snazší možnost náhrady systémů v budoucnu.



Obrázek 2.5: Uspořádání hub-and-spoke

C. Uspořádání sběrnice

Ve třetím možném uspořádání jsou všechny systémy připojeny na virtuální sběrnici, se kterou si vyměňují zprávy. Jednotlivé systémy tudíž nemají povědomost jeden o druhém (viz obrázek 2.6). I když toto uspořádání vypadá na první pohled odlišně od uspořádání hub-and-spoke, ve skutečnosti skýtá mnoho podobností. Vodorovná čára ve schématu samozřejmě není žádný „drát“, ve skutečnosti jde o softwarovou komponentu. Pokud bychom schéma v tomto smyslu překreslili, nebyl by mezi oběma uspořádáními velký rozdíl a podobnost výhod (flexibilita, centralizace, provozovatelnost, volnost spojení) i nevýhod (počáteční investice) rovněž není čistě náhodná.



Obrázek 2.6: Uspořádání integrační sběrnice

V čem je tedy rozdíl mezi hubem a sběrnicí? Ve způsobu navazování vazeb mezi jednotlivými systémy. Obě uspořádání slouží k přenosu zpráv mezi službami. Zprávy přenášené hubem připomínají dopis – mají jasného adresáta i odesilatele, hub zde zastupuje poštovní úřad s případnou další přidanou hodnotou. Naopak sběrnice funguje mnohem anonymněji – na principu *publish/subscribe*. Jednotlivé systémy vysílají na sběrnici svoje zprávy *(publish)* a zároveň mají určitým způsobem objednanou předem určenou množinu zpráv *(subscribe),* které se na sběrnici objevují. Podobně fungují časopisy – představte si, že byste měli u své poštovní doručovatelky předplaceny všechny výtisky časopisu Svět socialismu, ve kterých se vyskytují články o hrdinských činech Leonida Iljiče Brežněva (úmyslně volím témata, která minimalizují možnost žaloby dotčených stran). Integrační sběrnice funguje velmi obdobně, ale s důležitějšími informacemi. Aplikace může mít například předplaceny všechny zprávy obsahující změny kontaktních údajů klientů. Sběrnice tak může tutéž správu doručovat na více míst v závislosti na počtu subskripcí. Samozřejmě je nutné, aby integrované aplikace či služby byly na tento způsob komunikace připraveny, případně je nutné napsat mezivrstvu kódu mezi aplikaci a sběrnici, která vysílání a přijímání zpráv o událostech na sběrnici zprostředkovává.

Co je to integrační broker?

Praktická definice: integrační broker je softwarová komponenta, která zprostředkovává roli hubu nebo sběrnice ve výše uvedených uspořádáních. Debaty o tom, zda je lepší používat uspořádání hubu nebo sběrnice jsou celkem zbytečné – v některých situacích se hodí to, v jiných ono. Většina integračních nástrojů je postavena spíše pro podporu uspořádání typu integrační sběrnice. Důvod je prostý – sběrnice je obecnější koncept a vhodným nastavením podmínek lze v konkrétní situaci nastavit sběrnici tak, že funguje pro danou komunikaci jako hub. Opačný postup, to je zobecnění funkce hubu pro princip sběrnice by byl určitě náročnější a mnohem méně intuitivní.

Činnost integračního brokeru lze zpravidla rozdělit do více logických vrstev. Čím nižší vrstva, tím vyšší podíl technologických, infrastrukturních a provozních funkcí. Čím vyšší vrstva, tím více abstrakce a organizačně-obchodních záležitostí. Rozvrstvení na obrázku 2.7 jsem si poznamenal na jedné prezentaci Gartner Group.

Obrázek 2.7: Rozvrstvení integračního brokeru

Nejnižší vrstva se stará o doručování zpráv mezi jednotlivými integrovanými službami. Tato vrstva se nezajímá o obsah zpráv, jsou pro ni nezajímavými posloupnostmi bytů. Mezi její hlavní úkoly patří spolehlivé doručení zpráv s garantovanou kvalitou doručení, překlenutí výpadku mezi systémy, monitorování rychlosti doručení, diagnostika příslušných problémů, audit zpráv prošlých systémem a další funkce. Používá zpravidla frontu zpráv uchovávanou v relační databázi. Ke komunikaci s integrovanými systémy využívá buď všeobecně rozšířené technologie (např. SOAP, HTTP, FTP, message queue) nebo speciálně vytvořené adaptéry pro rozhraní integrované aplikace (např. SAP adaptér využívající rozhraní ABAP). Tato vrstva též zpravidla obstarává bezpečnostní funkce, jako je autentizace a autorizace komunikujících stran, šifrování přenášených zpráv, digitální podepisování a kontrola podpisu a podobně.

Vrstva transformace a routování dat se již zajímá o vlastní obsah přenášených zpráv. Může mít na starosti kontrolu validity přijatých zpráv. Velmi často též mění obsah zprávy, ať už pouze po stránce formální, jakou je převod mezi znakovými sadami anebo konverze mezi formáty (např. z EDI do XML) nebo po stránce věcné (konverze dokumentů mezi aplikacemi používajícími jinou sémantiku téhož dokumentu). Pokud je integrační broker používán jako hub, je úkolem routovací vrstvy výběr správného příjemce na základě okolností přijaté zprávy (typ dokumentu, původce, autentizační informace aj.). Pokud je broker využíván jako sběrnice, má za úkol vyhodnocení všech možných příjemců na základě pravidel a filtrů jednotlivých předplatných a předání seznamu možných příjemců nižší vrstvě.

Vrstva správy obchodních procesů je zřejmě nejzajímavější, neboť přináší velký skok v úrovni abstrakce. Z hlediska fungování výsledků organizace nejsou zajímavé jednotlivé elementární kroky, ale celé procesy, jenž jsou z nich složeny. Takovým procesem může být třeba zpracování objednávky nebo schválení žádosti. Každý proces se může sestávat z řady dílčích elementárních kroků představovaných výměnou zpráv mezi zúčastněnými systémy. Vrstva správy obchodních procesů se stará o sladění těchto elementárních kroků do vyšších přehledných celků. Má na starosti větvení procesů podle obchodních pravidel, sledování paralelních toků procesu a jejich zpětnou synchronizaci, hlídání mezních časů pro běh procesu *(time-out),* kontrolu odesílání a příjmu zpráv v rámci daného procesu a další podobné záležitosti. Velmi důležitou funkcí vrstvy je též podpora a správa dlouhotrvajících transakcí – pokud by některá z podúloh v transakci selhala, je nutné uvést zúčastněné systémy zpět do konzistentního stavu pomocí kompenzujících úloh (např. vystav fakturu – stornuj fakturu). Samozřejmostí je robustnost procesu – jejich stav je udržován na trvalém médiu a pro případ kritického výpadku systému je zaručeno pokračování procesu po obnovení všech důležitých životních funkcí.

Nejvyšší vrstvou brokeru je vrstva obchodně-analytického monitorování, kde již definitivně opustíme svět technických termínů a vstoupíme do světa obchodních či organizačních ukazatelů. Pozor – jde o oblast, kde může vedoucí IT pracovník dobře „prodat“ užitečnost své práce managementu firmy! Probíhající procesy jsou totiž blízké poslání firmy či organizace a zkoumáním těchto procesů lze poměrně dobře kontrolovat plnění tohoto poslání. Vrstva správy obchodních procesů vytváří velké množství dat, z nichž lze usuzovat na celkové zdraví implementovaného procesu. Jde přitom o čísla, která se pravděpodobně nikdy neobjeví v centrálních datových skladech – datové sklady zpravidla obsahují informace o výsledcích činnosti, nikoliv o procesech vedoucích k těmto výsledkům, přitom právě zdokonalení procesů je přímá cesta k lepším výsledkům. Na základě statistického vyhodnocení procesních dat můžeme zodpovědět na otázky typu: „Jaká je průměrná doba zpracování žádosti v závislosti na místě bydliště žadatele?“ nebo „Jak se mění finanční hodnota objednávek v různých stádiích rozpracovanosti s časem?“. Právě odpovědi na tyto otázky představují vysoký přínos investice do integračního projektu a IT pracovník by neměl přehlédnout jejich vysokou „politickou“ hodnotu.

V dalších částech se budeme hlouběji zabývat funkcí jednotlivých vrstev brokeru, přičemž jejich implementaci si ukážeme na BizTalk serveru 2004, integračním brokeru z dílny Microsoftu. Tipovací soutěž – kterou vrstvou asi začneme?

2.5 Reference

Application Architecture: Conceptual View
http://msdn.microsoft.com/architecture/application/default.aspx?pull=/library/en-us/dnea/html/eaappconland.asp

Standard XML 1.0
http://www.w3.org/XML/core

XML Schema
http://www.w3.org/TR/xmlschema-0/
http://www.w3.org/TR/xmlschema-1/
http://www.w3.org/TR/xmlschema-2/

XSLT 1.0
http://www.w3.org/TR/xslt

WSDL 1.1
http://www.w3.org/TR/wsdl

SOAP 1.1
http://www.w3.org/TR/SOAP11/

UDDI Version 2
http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm#uddiv2

WS-I organizace
http://www.ws-i.org/

WS-I Basic Profile 1.0a
http://www.ws-i.org/Profiles/Basic/2003-08/BasicProfile-1.0a.html

Building Interoperable Web Services: WS-I Basic Profile 1.0
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnsvcinter/html/wsi-bp_msdn_landingpage.asp

WS-Security standard
http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss

WS-Security Drilldown in Web Services Enhancements 2.0
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwse/html/wssecdrill.asp

Rozcestník rozšiřujících WS-* standardů
http://msdn.microsoft.com/webservices/understanding/specs/default.aspx

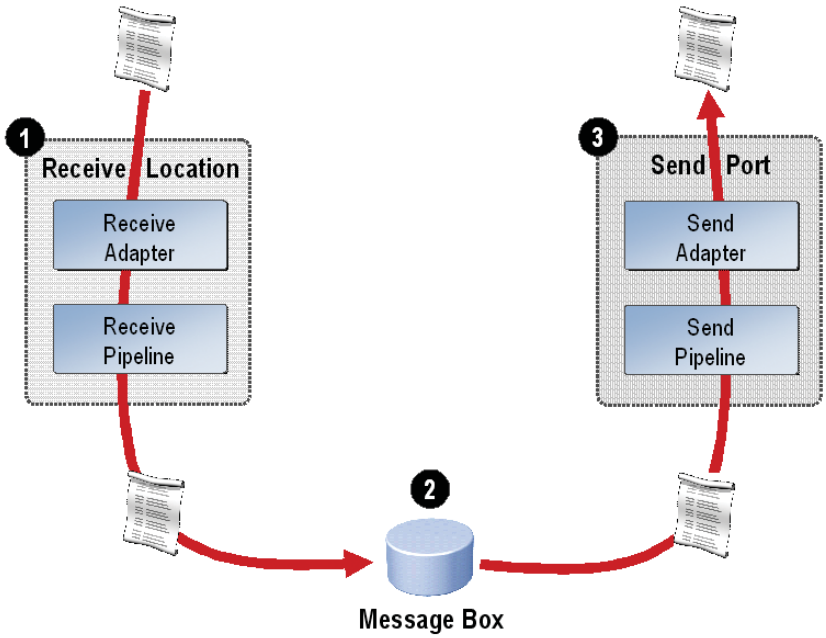
BizTalk Server 2004
http://www.microsoft.com/biztalk/

Kapitola 3:
Komunikace a přenos zpráv

Nejnižší vrstva se stará o doručování zpráv mezi jednotlivými integrovanými službami. Tato vrstva se nezajímá o obsah zpráv, jsou to pro ni nezajímavé posloupnosti bytů. Mezi její hlavní úkoly patří spolehlivé doručení zprávy s garantovanou kvalitou doručení, překlenutí výpadku mezi systémy, monitorování rychlosti doručení, diagnostika případných problémů, audit zpráv prošlých systémem a další funkce. Používá k tomu zpravidla frontu zpráv uchovávanou v relační databázi. Ke komunikaci s integrovanými systémy používá buď všeobecně rozšířené technologie (např. SOAP, HTTP, FTP, *message queue*) nebo speciálně vytvořené adaptéry pro rozhraní integrované aplikace (např. SAP adaptér využívající rozhraní ABAP). Tato vrstva též zpravidla zabezpečuje bezpečnostní funkce, jako je autentizace a autorizace komunikujících stran, šifrování přenášených zpráv, digitální podepisování, kontrolu podpisu a podobně.

3.1 Základní pojmy a koncepce

Integrační broker má škálovatelnou architekturu pro zpracování procházejících zpráv (viz obrázek 3.1). Toto uspořádání zajišťuje tři základní funkce – přijetí zprávy, uložení zprávy do trvalého úložiště a doručení zprávy. Podívejme se nyní na jednotlivé složky architektury.



Obrázek 3.1: Průchod zprávy integračním brokerem

Message box

Message box je v tomto případě relační databáze SQL serveru, používaná jako trvalé úložiště všech zpráv procházejících systémem. V případě požadavku na extrémně vysokou škálovatelnost je možné mít více oddělených databází *message box*. Použití relační databáze pro uložení stavu zpráv a procesů je velmi výhodné, neboť může plně využít samozřejmé vlastnosti databázového stroje – vysokou stabilitu a výkon, možnost transakčních operací nad daty, robustní zálohování a obnovu, technologie pro vysokou dostupnost (clustering), auditing a řadu dalších funkcí. Relační databáze je rovněž dobře monitorovatelná a prozkoumatelná – což je neocenitelné zejména v případě eventuálních provozních problémů. Přestože databáze je vnitřní záležitostí integračního brokeru a přímé zásahy do databáze nejsou předpokládány ani podporovány, je vždy dobré mít tuto možnost v záloze jako KPZ (krabičku poslední záchrany).

Adaptér

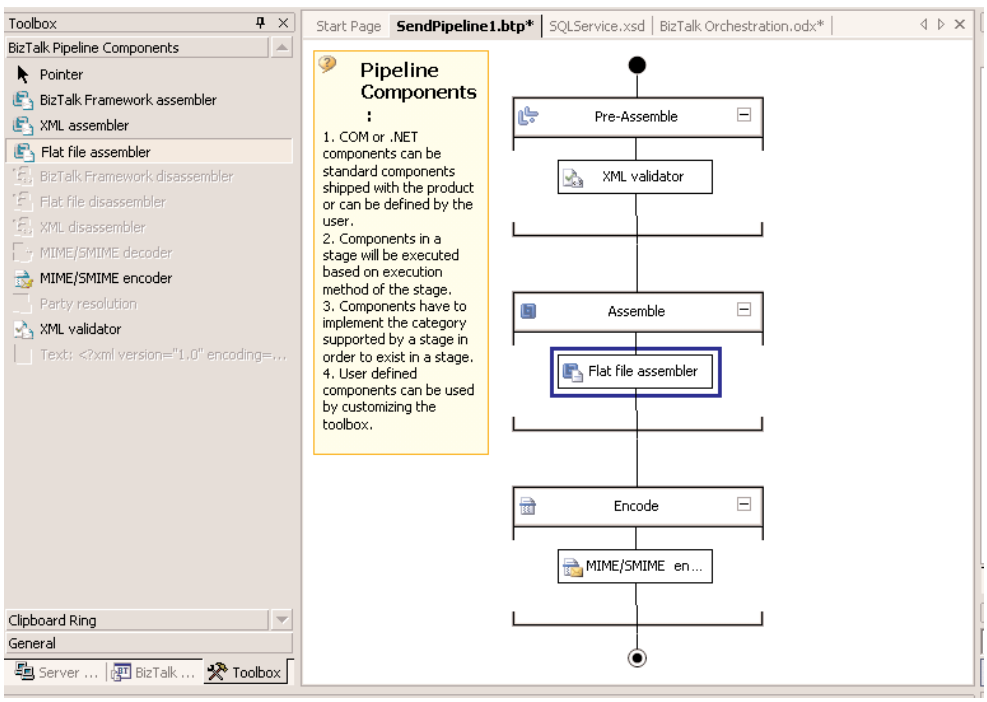
Integrační broker komunikuje s integrovanými aplikacemi či službami prostřednictvím tzv. adaptérů. Adaptér je prvek spojující integrační broker s integrovanými aplikacemi. Koncepce adaptéru slouží k usnadnění života vývojářům i implementátorům, podobně jako třeba jednotná koncepce tiskových ovladačů usnadňuje život běžným uživatelům Windows. Vývojáři adaptérů se mohou soustředit pouze na věci specifické pro danou aplikaci/technologii, BizTalk

Server 2004 jim nabízí tzv. *adapter framework* pro konfiguraci a běh adaptérů. Z této jednotnosti těží též implementátoři, neboť všechny adaptéry se pak stejným způsobem konfigurují a spravují.
Adaptéry lze rozdělit do tří skupin:

- Aplikační adaptéry (vertikální) – spojují integrační broker s vybranou konkrétní aplikací (např. SAP, Siebel, PeopleSoft, Navision, Great Plains). Pokud aplikace implementuje nestandardní rozhraní (např. SAP rozhraní BAPI), potom adaptér obsahuje technické prostředky pro komunikaci s aplikací. Aplikační adaptéry dále obsahují značné know-how o dané aplikaci, zejména o jejích funkcích, které bývají zpřístupněny prostřednictvím předpřipravených portů
- Technologické adaptéry (horizontální) – spojují integrační broker s okolím prostřednictvím technologie nezávislé na konkrétní aplikaci (např. webové služby, souborový systém, SMTP, TCP/IP, LU 6.2, MQSeries, FTP). Tyto adaptéry jsou zpravidla jednodušší a tudíž vyžadují větší konfigurační úsilí na straně brokeru i integrované aplikace.
- Datové adaptéry – jsou na pomezí mezi aplikačními a technologickými adaptéry. Umožňují integračnímu brokeru komunikovat přímo s databází (např. MS SQL, Oracle, DB2). Tyto adaptéry umožňují pohodlný způsob přístupu k datům v operačních databázích, vhodné jsou například pro udržování informací o stavu integrovaných procesů, workflow mimo aplikace apod. Neměli byste však podlehnout svodům „zneužívání" těchto adaptérů k obcházení vlastních aplikací (např. modifikovat data v databázi SAPu). I když jde po technické stránce o schůdné řešení, porušuje zásadním způsobem principy architektury postavené na službách – zbavuje službu její autonomie a umožňuje jinou komunikaci se službou než je výměna zpráv prostřednictvím zveřejněného rozhraní.

Pipeline

Pro přijetí a odeslání se využívá koncepce tzv. proudového zpracování zprávy pomocí myšlenkové konstrukce nazývané *pipeline* (podle slovníku potrubí, ropovod – raději se snad podržím anglického termínu). *Pipeline* slouží ke zpracování přijaté zprávy před jejím uložením do *message boxu* anebo před odesláním po vyjmutí správy z *message boxu*. Je složena z posloupnosti jednotlivých etap ve zpracování zprávy *(stage)* jako je např. dekódování, dešifrování či kontrola digitálního podpisu. V každé etapě je možné zařadit žádnou, jednu nebo více komponent pro danou etapu. Zpráva pak prochází postupně těmito komponentami, přičemž každá komponenta může zprávu modifikovat (např. dešifrovací komponenta), vyvolat chybu (např. komponenta kontrolující správnost příchozí zprávy) nebo zapisovat do kontextu zprávy vlastnosti pro účely jejich pozdějšího vyhledávání nebo dalšího směrování. Jednotlivé *pipeline* vytváří vývojář pomocí techniky *drag&drop* ve Visual Studiu.NET (viz obrázek 3.2). Po zkompilování těchto *pipeline* a jejich nasazení na server jsou tyto *pipeline* připraveny k použití. Výhodou je rozšiřitelnost – kromě předpřipravených *pipeline* komponent je možné vytvářet vlastní komponenty. Například je možné vytvořit komponenty pro kompresi a dekompresi, odstranění diakritiky z textových zpráv apod. Celou řadou komponent pro *pipeline* se ještě budeme zabývat v dalším výkladu.



Obrázek 3.2: Pipeline designer ve Visual Studiu.NET

Receive port

Port pro přijetí zprávy *(receive port)* slouží pro převzetí zprávy integračním brokerem. Je jakousi logickou adresou nebo poštovní schránkou, přes kterou jsou zprávy doručovány – tato virtuální adresa slouží vyšším vrstvám integračního brokeru ke kategorizaci zpráv a k řízení jejich dalšího zpracování. Porty pro přijetí mohou být dvojího druhu – bez odpovědi na přijatou zprávu *(one-way)* nebo s odpovědí *(request-response),* přičemž adaptér nemusí nutně podporovat oba způsoby. Např. adaptér pro MSMQ nepodporuje odpovědi, neboť pro asynchronně přijatou zprávu tato funkčnost postrádá smysl, naopak HTTP adaptér díky specifikaci HTTP protokolu odpovědi umožňuje. Některé adaptéry nemusí dokonce podporovat přijímání zpráv vůbec (např. dodávaný adaptér pro SMTP).

Receive location

Jak již bylo řečeno, port pro doručení je pouhou logickou adresou pro doručení. Skutečná zpráva musí být doručena nějakým konkrétním fyzickým způsobem prostřednictvím určitého konkrétního adaptéru na tzv. *receive location*. Je to stejné jako v komunikaci mezi lidmi. Chcete-li např. dívku pozvat na rande, musíte jí poslat zprávu „pozvání na rande“. Dívka v tomto případě funguje jako *receive port,* přičemž tuto zprávu je nutné doručit prostřednictvím některého technologického adaptéru na konkrétní *receive location,* kterou může být např. poštovní adresa, e-mailová adresa, číslo mobilního telefonu pro poslání SMS (exhibicionisté pošlou MMS), číslo pevné linky pro hlasový hovor a řada dalších více či méně romantických způsobů (naštěstí nejsme v USA, přepsání předchozí věty do bezpohlavního tónu by nebylo snadné). V případě integračního projektu může být portem např. „přijetí žádosti“ a může být spojené se dvěma *receive location* – synchronním přijetím prostřednictvím webové služby a asynchronním přijetím zprávy doručené přes *MQSeries*. Třemi nejdůležitějšími vlastnostmi každého *receive location* jsou použitý adaptér, adresa závislá na zvoleném adaptéru (např. pro FILE adaptér je adresou cesta ke složce, která je monitorována) a *pipeline* použitá pro zpracování přijaté zprávy.

Send port

Send port neboli port pro odeslání slouží pro doručení zprávy cílové službě nebo příjemci. Port plní funkci logické adresy (virtuální schránka, kam zprávy odcházejí) a jeho hlavními vlastnostmi jsou *pipeline* pro zpracování odesílané zprávy, primární transport a někdy též sekundární (náhradní) transport. Transport je pak definován především použitým adaptérem, adresou pro doručení a parametry pro opakování případného neúspěšného přenosu. Převedeno na mezilidské vztahy – portem je „pozvání na rande dívce X“, přičemž pro doručení zprávy zkusíte použít hlasový hovor (maximálně třikrát s hodinovou prodlevou mezi voláními), a pokud je tento primární transport neúspěšný, pošlete SMSku (někdo může volit odeslání téže zprávy na úplně jiný port, ale to nezapadá do našeho příkladu). Ve světě technologií to může být obdobně. O odeslání objednávky se můžete pokusit voláním webové služby a pokud se to nepodaří, odeslat objednávku e-mailem prostřednictvím SMTP protokolu. Podobně jako porty pro přijetí, i porty pro odeslání mohou být dvojího druhu – s čekáním na odpověď *(solicit-response)* anebo bez odpovědi *(one-way),* daný adaptér opět nemusí podporovat oba způsoby. Existuje ještě druhý způsob dělení – na porty statické a dynamické. Zatímco statické porty doručí zprávu vždy na předdefinovanou adresu, dynamické porty nemají pevně stanovenou cílovou adresu. Adresa je totiž přiřazena až při zpracování zprávy. Např. pokud zpracovávaná zpráva obsahuje emailovou adresu zodpovědné osoby, je možné poslat zodpovědné osobě potvrzení – vytvořit dynamický port a cílovou adresu nastavit na základě emailu obsaženého ve zprávě.

Port group

Někdy se může stát, že tutéž zprávu je nutné paralelně poslat na více cílových míst. Za tímto účelem je možné vytvořit skupinu portů *(port group),* která při odesílání představuje jeden logický koncový bod, ale zpráva je ve skutečnosti doručena na více fyzických koncových bodů – portů pro odeslání. Podobnému principu se v terminologii komunikačních serverů (*Microsoft Exchange, Lotus Notes*) zpravidla říká distribuční seznam *(distribution list)*. Existuje ještě jeden způsob doručení na více míst, a to metoda *publish-subscribe*, při které je naprosto oddělen odesilatel a příjemce. Tento způsob je v BizTalku rovněž podporován a budeme se mu věnovat v kapitole 4. Rozdíl mezi oběma způsoby je ve způsobu párování odesilatele s příjemci. *Port group* funguje direktivním způsobem, příjemci jsou jednoznačně určeni „shora“, dobrým příkladem může být distribuce volebních lístků, které jsou doručeny na pevně stanovený seznam voličů. *Publish-subscribe* odděluje odesilatele a příjemce, u kterého předpokládá aktivní přihlášení k odběru. Zde může být příkladem seriózní časopis (nemyslím Strážní věž a jiné reklamní tiskoviny) – redakce vyprodukuje časopis a dá ho do distribuce aniž by se zajímala o jednotlivé předplatitele, předplatitelé se přihlásí k odběru a dostávají onen časopis do schránky. Nelze říct, který ze způsobů je lepší nebo horší, pro každou situaci se hodí jiný a ve skutečnosti mezi nimi není zcela ostrá hranice.

Host

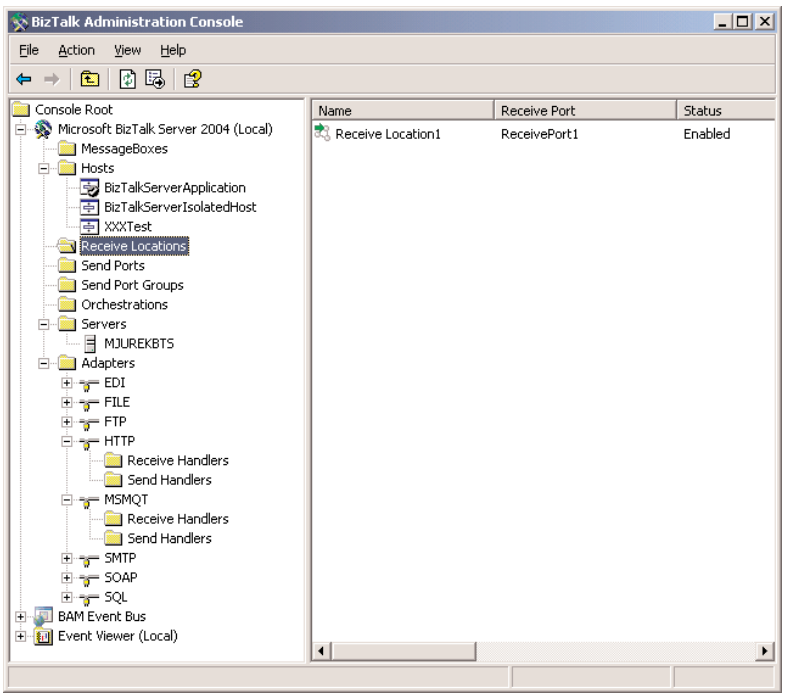
Hostitele *(host)* je možné si představit jako určitou virtuální integrační aplikaci, která slouží jako pojmenovaný kontejner pro jednotlivé prvky integračního řešení, zejména porty a orchestrace (o nich se dozvíme více v kapitole 5). Hostitel sám o sobě nemá žádnou úlohu z hlediska vývoje aplikace. Jeho úkolem je izolace jednotlivých částí řešení pro snadnější správu a zajištění provozovatelnosti řešení, je prostředím, ve kterém naše aplikace běží. Hostitel může mít bezpečnostní úlohu – tvoří hranici důvěry mezi částmi řešení běžícími pod jedním bezpečnostním účtem, který používají při přístupu k externím zdrojům, takže administrátor může dobře omezit jejich práva na nezbytné minimum. Hostitel může též zajišťovat stabilitu řešení – pokud jsou součástí řešení komponenty, o jejichž kvalitě a stabilitě nejste zcela přesvědčeni, můžete je izolovat do samostatného hostitele, aby svojí potenciální nestabilitou neohrozili zbytek integračního řešení. Hostitel je jedním celkem pro sledování výkonnostních charakteristik, případně uplatňování politik na spotřebu zdrojů (zejména paměť a výkon procesoru). Hostitelům jsou též přiřazovány jednotlivé adaptéry, přičemž adaptér přiřazený konkrétnímu hostiteli se nazývá *handler*.

Host instance

Hostitel je pouze myšlenkovou konstrukcí, instance hostitele *(host instance)* je jeho realizací na konkrétním serveru v podobě procesu operačního systému. Při testování zpravidla používáme pouze jeden server a tak pro nás tento pojem není příliš zajímavý. Ve velkých provozních systémech lze instance hostitelů různě kombinovat a dosáhnout tak rozkládání zátěže nebo specializace serverů ve skupině více serverů. Např. je možné mít jednoho hostitele pro příjem zpráv přes webové služby s instancemi na serverech A a B, dalšího hostitele pro řízení procesů s instancemi na serverech A a B, a konečně třetího hostitele pro asynchronní komunikaci se SAPem s jedinou instancí na serveru C, přičemž všechny servery A, B i C ukládají stav na databázovém serveru D. Tím jsme dosáhli vysoké dostupnosti a rozkládání zátěže pro webové služby a řízení procesů, zároveň jsme minimalizovali licenční náklady na adaptér pro SAP (u asynchronní komunikace nepožadujeme vysokou dostupnost), navíc můžeme servery A a B umístit z bezpečnostních důvodů do jiného síťového segmentu než server C.

3.2 Dostupné adaptéry a technologie

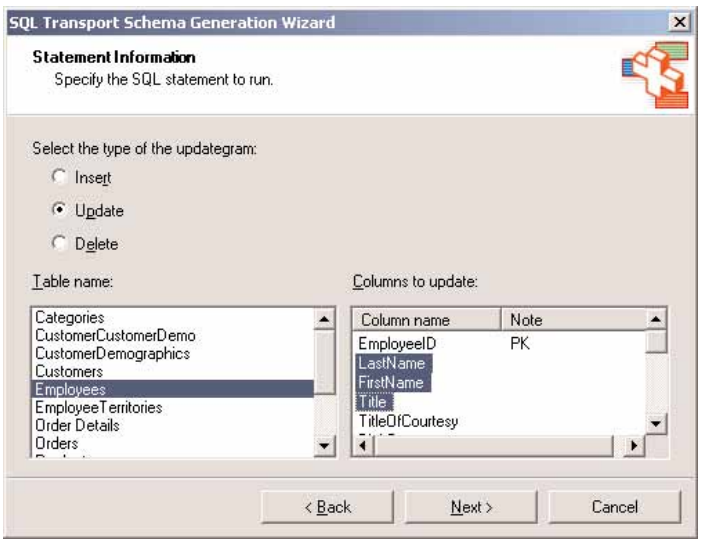
Adaptéry lze dále rozdělit podle jejich původu a dostupnosti. Aplikační adaptéry buď dodává výrobce aplikace (např. Siebel) nebo firma specializovaná na výrobu adaptérů (*pure play adapter vendor,* např. iWay) a v případě SAPu nabízí komerční adaptér též Microsoft. Technologické a datové adaptéry jsou zpravidla dodávány specializovanými firmami, Microsoft nabízí ke stažení adaptér pro IBM MQSeries a osm technologických adaptérů najdete přímo „v krabici“ BizTalku (viz obrázek 3.3).



Obrázek 3.3: Dostupné adaptéry v administrativním nástroji

Pojďme se na jednotlivé adaptéry podívat blíže:

- SOAP adaptér – implementuje komunikaci prostřednictvím protokolu webových služeb SOAP nad protokolem HTTP. Jedná se o nejdůležitější adaptér, neboť webové služby se stávají nosným prvkem mnoha integračních projektů a tento adaptér je tak základem pro interoperabilitu v heterogenním prostředí. K adresaci odchozích zpráv a ke kategorizaci příchozích zpráv je pochopitelně používána URL adresa. Vzhledem k tomu, že řada aplikací dnes podporuje webové služby, může být tento adaptér dobrou alternativou nativních aplikačních adaptérů. Např. se systémem SAP lze komunikovat prostřednictvím podpory webových služeb v tomto produktu, bez nutnosti zakupovat nativní aplikační adaptér (ale se zvýšenými nároky na pracnost a pravděpodobně též s větší režií zpracování – něco za něco, jako v životě).
- MSMQT adaptér – umožňuje asynchronní komunikaci se servery s nainstalovaným Microsoft Message Queueing. MSMQT v BizTalk serveru přitom implementuje protokol MSMQ a částečně nahrazuje MSMQ na počítači s BizTalk serverem. Zprávy jsou v transakci přijímány prostřednictvím virtuálních front přímo do databáze *message box,* odesílané zprávy jsou předávány přednastavenému MSMQ routeru.
- FILE adaptér – nabízí komunikaci prostřednictvím souborového systému. Přijetí zprávy znamená čekání na nový soubor v určené složce, jeho načtení a smazání. Doručení znamená vytvoření nového souboru v určené složce. Vzhledem k tomu, že řada aplikací má funkce exportu/importu do souboru, může jít o velmi bezbolestné a relativně pohodlné řešení. Navíc jsou k dispozici technologie pro integraci souborových systému (např. podpora NFS v doplňku *Services for Unix* nebo pro mainframe v *Host Integration Serveru*), takže tuto techniku je možné použít i v heterogenním prostředí.
- HTTP adaptér – umožňuje odeslání zprávy nebo naopak přijetí zprávy v těle POST požadavku podle specifikace protokolu HTTP. Pro přijetí zprávy nabízí BizTalk svoji URL adresu, na kterou je zpráva doručena, odeslání může být na libovolné URL. Tento adaptér je velmi vhodný do heterogenního prostředí, neboť protokol HTTP je široce podporován napříč platformami.
- SMTP adaptér – poskytuje možnost odeslat zprávu prostřednictvím e-mailu na obvyklou libovolnou adresu, zcela v souladu s protokolem SMTP. Tento adaptér nenabízí přijetí zprávy přes SMTP protokol – neimplementuje vlastní SMTP server. Pokud chcete zprávy v e-mailech přijímat, je třeba využít technologii vašeho SMTP serveru pro zpracování příchozích zpráv a při obsluze události přijetí předat zprávu integračnímu brokeru pomocí vlastního jednoduchého kódu a zdokumentovaného rozhraní.
- FTP adaptér – komunikuje s integrovanými systémy pomocí protokolu FTP. Adaptér umožňuje přijetí zprávy v souborech stahovaných ze specifikovaného FTP serveru v pravidelných časových intervalech za použití příkazu GET anebo naopak odeslání zprávy v souboru na vzdálený FTP server pomocí příkazu PUT.
- SQL adaptér – umožňuje příjem zpráv z SQL Serveru 2000 formou dat vrácených v pravidelném intervalu opakovaným SQL příkazem nebo voláním uložené procedury. Odeslání zprávy je možné formou volání uložené procedury nebo použitím tzv. *updategram* volání. Výhodnou vlastností adaptéru je generování schémat dokumentů na základě metadat (informací o tabulkách, sloupcích, parametrech apod. – viz obrázek 3.4). Tím se výrazně zjednodušuje pracnost této výměny – stačí pouze vytvořit dokument podle daného schématu, o zbytek práce se postará adaptér.



Obrázek 3.4: Generování potřebných dokumentů SQL adaptéru

- EDI – neboli *Electronic Data Interchange* je formát pro komunikaci mezi organizacemi. Tento formát je v současné době silně vytlačován XML, nicméně vzhledem k existujícím rozsáhlým řešením a velkým minulým investicím do této technologie bude EDI používáno ještě dlouhou dobu. EDI v BizTalku podporuje americkou variantu X-12 i evropskou EDIFACT. Adaptér byl vyvinut partnerskou firmou Covast a nabízí základní funkci pro přijímání a odesílání EDI zpráv. Od stejné firmy lze zakoupit EDI adaptér s mnohem širší funkčností.

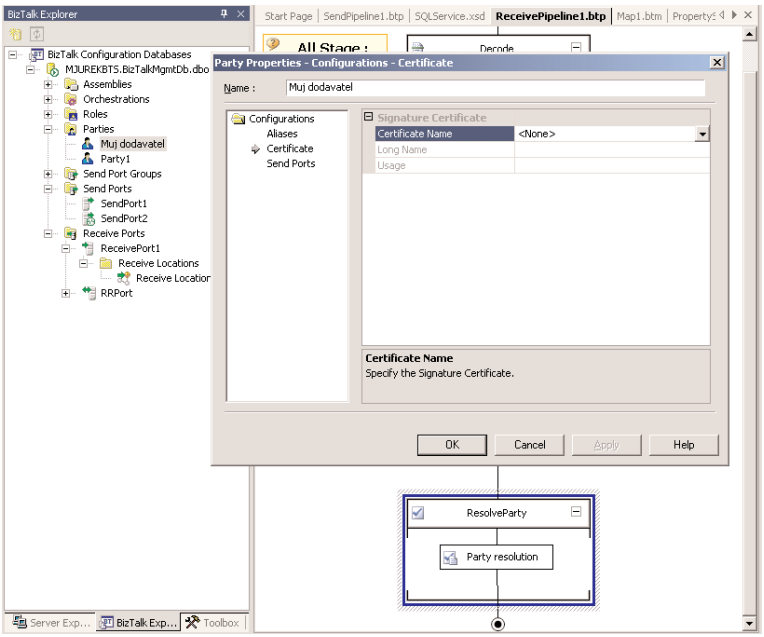
Aktuální seznam všech dostupných adaptérů najdete na adrese *http://www.microsoft.com/biztalk/adapters*. Při výběru vhodného adaptéru pro určitou aplikaci můžete být postaveni před rozhodnutí, zda zakoupit hotový aplikační adaptér anebo si vyvinout vlastní řešení pomocí přibaleného technologického adaptéru (např. SOAP pokud aplikace podporuje webové služby). Nejde o jednoduché rozhodnutí, volíte mezi nákupem *know-how* a jeho získáním prostřednictvím vložené lidské práce. V úvahu je nutné vzít též stabilitu dodavatele, úroveň jeho technické podpory a řadu dalších faktorů.

3.3 Bezpečnost přenosu zpráv

Asi každý intuitivně tuší, co to bezpečnost je, ale zkuste si vzít papír a napsat její definici – mám pocit, že se pěkně zapotíte. Moje oblíbená definice je tato: „Stav, kdy jsou aplikace a ostatní součásti IT prostředí využívány v souladu se svým určením a nemohou být využívány jinak“. Bezpečnost lze rozdělit do řady kategorií, zejména autentizaci, autorizaci, důvěrnost, integritu dat a neodmítnutelnost odpovědnosti. V integračním řešení mohou být tyto funkce implementovány přímo integračním brokerem. Tím sice vzniká žádoucí bezpečnostní vztah mezi integrovanou aplikací a integračním brokerem, ale vyžaduje používání stejných bezpečnostních mechanismů aplikací i brokerem, např. S/MIME – podmínku, která je splněna poměrně zřídka. Druhou a pravděpodobnější možností je bezpečnostní vazba mezi aplikací a adaptérem, což lze splnit zpravidla snadněji, ale vyžadující nevyřčenou důvěru integračního brokeru vůči bezpečnostním funkcím použitého adaptéru. Pojďme se podívat na jednotlivé nabízené možnosti

Autentizace a autorizace adaptéry

Autentizace je nejčastěji záležitostí daného adaptéru, který může autentizační informace předat pro další zpracování v integračním brokeru – po technické stránce jde o zápis autentizačních identifikátorů do seznamu vlastností zprávy. Existují dva autentizační identifikátory. Prvním je bezpečnostní identifikátor *(Security ID, SID),* jehož hodnota je nastavena podle bezpečnostního účtu, pod kterým je zpráva předána integračnímu brokeru (např. v případě SOAP adaptéru je to účet, pod kterým byla webová služba volána). Druhým je identifikátor partnera *(Party ID, PID),* který je doplněn při přijetí v *pipeline* v nepovinné fázi rozlišení partnera *(ResolveParty)*. BizTalk totiž umožňuje udržovat databázi partnerů, která kromě jednoznačného jména může obsahovat též namapované SID nebo certifikát použitý partnerem pro digitální podepsání příchozí zprávy (viz obrázek 3.5). Rozlišení partnera tedy v podstatě znamená převod neosobního a nesrozumitelného identifikátoru na jasné a přehledné jméno, se kterým může logika integračního brokeru dále pracovat. Partnerem samozřejmě nemusí být pouze obchodní partner, může to být třeba interní aplikace spravovaná nezávislou skupinou v rámci IT oddělení.



Obrázek 3.5: Definice partnera pomocí certifikátu

Vlastní autentizační mechanismus je plně v kompetenci každého adaptéru. Například u HTTP adaptéru přijímajícího zprávu mohou být kromě anonymního přijetí zprávy použity autentizace typu *Basic, Digest* a *Windows integrated*. Na druhé straně FTP adaptér dost dobře nemůže podporovat autentizaci – při stažení zprávy (souboru) není možné věrohodně zjistit původce zprávy. Je trochu škoda, že SOAP adaptér BizTalk Serveru 2004 nepodporuje bezpečnostní mechanismy (autentizace, šifrování, digitální podpis) podle standardu *WS-Security*. Podle vyjádření vývojářského týmu se adaptér s podporou těchto technologií dodatečně připravuje.

Výše uvedený přístup s sebou nese určité bezpečnostní riziko – celý systém musí důvěřovat adaptéru, případně *pipeline* komponentě, která tyto informace do kontextu zprávy uložila. Nelze totiž vyloučit, že v systému je zlovolný adaptér nebo komponenta, která tyto informace zfalšovala a falešně tak vůči zbytku systému předstírá identitu původce zprávy. Proti tomuto riziku se lze poměrně dobře bránit. Každý hostitel má vlastnost nazývanou *Authentication trusted,* která říká, zda komponenty systému běžící v tomto hostiteli mají právo trvale zapisovat do autentizačních údajů zprávy. Potenciálně nedůvěryhodné komponenty a adaptéry je tak možné umístit do odděleného hostitele, který není důvěryhodný pro potřeby autentizace, nemůže tyto údaje zapisovat a tedy ani zmást zbytek systému falešnou identitou. Pokud se komponenta v takovémto nedůvěryhodném hostiteli pokusí autentizační údaje zapsat, bude *SecurityID* stejně přenastaveno na účet, pod kterým hostitel běží a *PartyID* nastaveno na hodnotu *SecurityID/Guest*.

BizTalk nemá žádný systematický mechanismus pro autorizaci volajícího. Samozřejmě je možné dělat autorizační rozhodnutí na základě identity odesilatele, a to buď filtrováním na portech pro odeslání (viz kapitola 4) nebo větvením a aplikační logikou v rámci správy procesů (viz kapitola 5). Systematičtější a snadněji spravovatelný přístup je provádění autorizačních rozhodnutí jednotlivými adaptéry v závislosti na použité technologii – např. pro *FILE* adaptér lze snadno využít přístupových práv k složce souborového systému jako velmi dobrého a dobře auditovatelného autorizačního mechanismu.

Šifrování pomocí S/MIME

S/MIME umožňuje standardní šifrování používané např. v elektronické poště. Biztalk podporuje nativně S/MIME a nabízí tak poměrně bezproblémové šifrování zajišťující důvěrnost zpráv vyměňovaných mezi integračním brokerem a aplikací. Princip je celkem zřejmý.

Při odesílání zprávy brokeru použije odesilatel pro zašifrování veřejný klíč z certifikátu brokeru, tento je potom dešifrován certifikátem, který je nastaven ve vlastnostech daného hostitele. Účet, pod kterým hostitel běží, musí mít samozřejmě přístup k certifikátu uloženému v chráněném úložišti, a to včetně privátního klíče. Pro správnou funkci je v *receive location* rovněž nutné zařadit *pipeline* obsahující komponentu *MIME/SMIME decoder*. Naopak broker při odesílání zašifrované zprávy musí mít k dispozici certifikát příjemce (samozřejmě bez privátního klíče) uložený v chráněném úložišti. Port pro odeslání má pak nastavený příslušný šifrovací certifikát a dále musí používat *pipeline* obsahující komponentu *MIME/SMIME encoder*. Rozšifrování na straně příjemce se pak děje obvyklým způsobem a není již záležitostí brokeru a tudíž ani této brožury.

Je možné, že budete chtít používat některé jiné šifrovací standardy než S/MIME, zejména pozvolna dozrávající standard *XML Encryption*. Tyto standardy bohužel nejsou v BizTalk Serveru 2004 zatím nativně podporovány, ale je relativně jednoduché napsat si vlastní pár *pipeline* komponent (encoder a decoder) a využívat je pro vlastní šifrování – přitom můžete jednoduše využívat nakonfigurovaných certifikátů v prostředí BizTalku, stejně jako to dělají S/MIME komponenty.

Digitální podepisování pomocí S/MIME

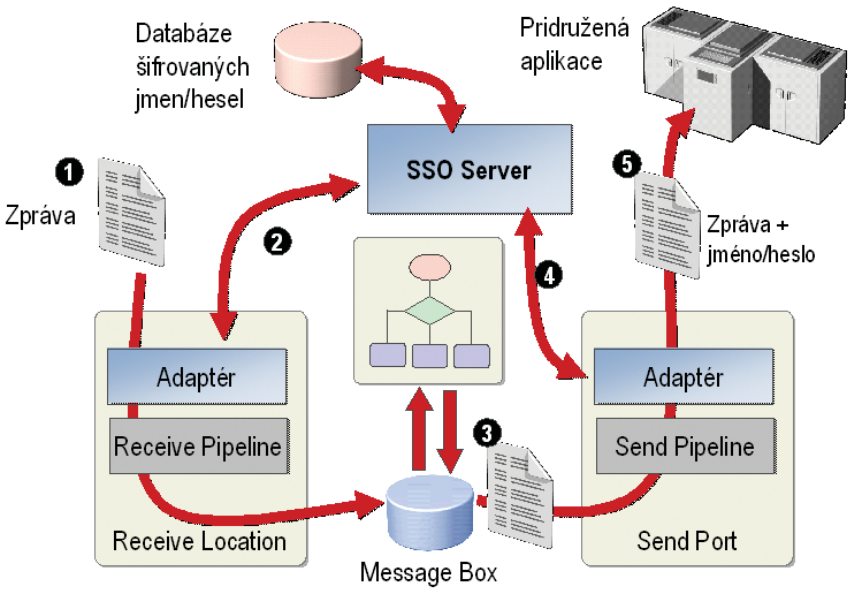
Druhou funkcí standardu S/MIME je digitální podpis, který zaručuje autentizaci odesilatele, kontrolu integrity zprávy a neodmítnutelnost odpovědnosti odesilatele. Podepisování pomocí standardu S/MIME je v Biztalku 2004 implementováno podobně jako šifrování a stejná analogie platí i pro možnou rozšiřitelnost o další standardy, zejména dobře se etablující *XML Signature*.

Kontrola digitálního podpisu odesilatele při přijetí zprávy je snadná. Stačí asociovat příslušné *receive location* s *pipeline* obsahující komponentu *S/MIME decoder*. Tato komponenta může zároveň kontrolovat, zda certifikát není zneplatněný umístěním na CRL *(certificate revocation list)*. Podepisující certifikát může být rovněž použit pro identifikaci odesilatele v B2B *(business-to-business)* scénářích v procesu nazývaném *party resolution,* jehož stručný popis najdete v příloze A.

Naopak pro podepsání odchozích zpráv je nutné nastavit podepisovací certifikát ve vlastnostech skupiny serverů v administrativním nástroji a poté nastavit v příslušném portu pro odeslání pipeline s vloženou komponentou *MIME/SMIME encoder*. Účet, pod kterým hostitel běží, musí mít samozřejmě přístup k certifikátu uloženému v chráněném úložišti, a to včetně privátního klíče.

Enterprise Single Sign-On

Jedním z problémů při integraci v heterogenním prostředí bývá existence mnoha různých přihlašovacích systémů a bezpečnostních domén. Není výjimkou, že uživatel má až desítky jmen a hesel do různých interních systémů, přičemž tato situace je velkou zátěží pro administrátory, uživatele i rozpočet IT oddělení. Naproti tomu v ideálním stavu by uživatel měl mít pouze jedno jediné primární jméno a heslo, které by bylo klíčem ke všem interním systémům, ale změna interních systémů za účelem používání jiných autentizačních schémat je bohužel většinou utopií. V této situaci je částečným řešením technologie *Enterprise Single Sign-on (ESSO),* která je kromě BizTalku implementována též v portálovém řešení *SharePoint Portal Server 2003*. Jde o důmyslné kryptografické řešení, jehož základem je databáze sekundárních přihlašovacích informací zašifrovaných symetrickou šifrou (tzv. *master secret*), takže v případě zcizení databáze jsou přihlašovací informace v ní uložené relativně bezcenné. V této databázi jsou definovány tzv. přidružené aplikace *(affiliate application)* a pro tyto aplikace je pak jednotlivému účtu nebo skupině uživatelů z domény uloženo šifrované jméno a heslo.



Obrázek 3.6: Postup použití ESSO v BizTalk Serveru

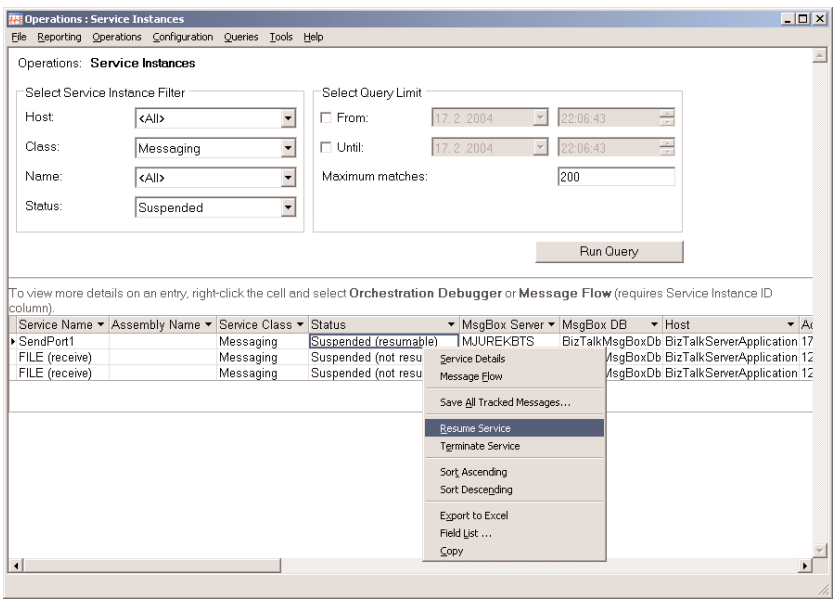
Postup použití ESSO v BizTalku je znázorněné na obrázku 3.6. Základní myšlenka je poměrně jednoduchá – zpráva je doručena adaptéru autentizovaným způsobem, při kterém je impersonifikován přistupující uživatel (tj. kód adaptéru běží pod jeho uživatelským účtem, takže je nepochybně znám jeho bezpečnostní identifikátor SID). Z vestavěných adaptérů tuto funkci podporuje pouze HTTP a SOAP adaptér, naštěstí jde o adaptéry, které budete často používat (krok 1). Podporuje-li adaptér ESSO, požádá SSO server o vydání takzvaného tiketu pro SSO (krok 2). Tento tiket je přiložen ke zprávě jako jedna z jeho vlastností, takže je uložen do *message boxu* a provází zprávu po celou dobu jejího zpracování (krok 3). Pokud je při odesílání použit adaptér podporující ESSO (z vestavěných adaptérů FTP, HTTP a SOAP) se zapnutou podporou a nastavenou hodnotou přidružené aplikace, předloží SSO serveru tiket přidaný ke zprávě a na jeho základě získá příslušné namapované přihlašovací informace pro cílovou aplikaci (krok 4). Tyto informace pak odesílající adaptér použije pro komunikaci s cílovou aplikací. Zdůrazněme, že přihlašovací informace nejsou nikde uloženy v otevřené formě a jsou přístupné pouze příslušnému schválenému adaptéru pouze pro jednu konkrétní přidruženou aplikaci, přičemž je nemůže získat bez toho, aniž by na začátku procesu byl daný uživatel nepochybně autentizován. Tento mechanismus samozřejmě předpokládá určitou důvěru v kvalitu a prověřenost používaných adaptérů, která by ovšem v provozním prostředí měla být samozřejmostí.

3.4 Spolehlivé doručení zpráv

Spolehlivé doručení zpráv je požadavkem tak samozřejmým, že je možné jej snadno přehlédnout. Při vývoji se zpravidla soustředíme na vyšší funkce, doručení zprávy považujeme za cosi normálního a samozřejmého, stejně jako u e-mailu nebo SMS textové zprávy. Při bližším pohledu ale zjistíme, že nejde o jednoduchou problematiku. Kvalitu služby doručení zprávy lze popsat různými charakteristikami:

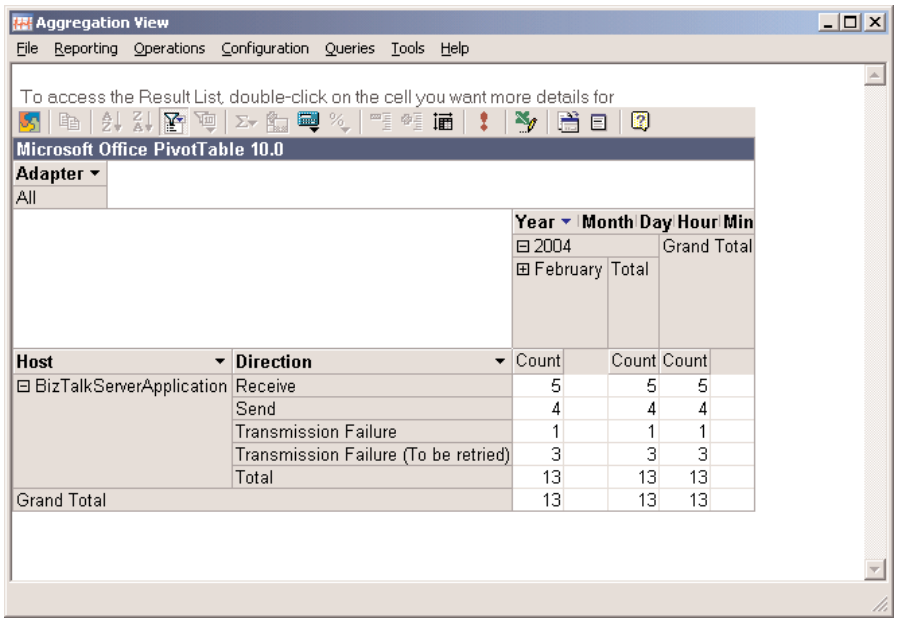
- Záruka doručení alespoň jedenkrát – pokud odesíláme zprávu, chceme mít jistotu, že bude doručena. V případě neúspěšného pokusu o doručení zprávy, např. z důvodu výpadku sítě nebo zastřelení posla nesoucího zprávu je zpráva odeslána znovu, a to tak dlouho, dokud není pozitivně potvrzeno doručení zprávy příjemci. Ve světě technologií se toto potvrzení zpravidla realizuje návratovou hodnotou, např. při doručení protokolem HTTP příjemce vrátí stavový kód v rozmezí 200–299, čímž se doručení zprávy považuje za úspěšné.
- Záruka doručení maximálně jedenkrát – při pokusech o doručení „za každou cenu“ může dojít k doručení zprávy více než jednou, protože odesilatel, který marně čeká na doručení pozitivního potvrzení si nemůže být jist, zda byl problém s doručením vlastní zprávy nebo s doručením onoho pozitivního potvrzení. Neboli neví, zda byl posel se zprávou zastřelen cestou k příjemci anebo až na zpáteční cestě. S tímto problémem se setkáte i v osobním životě – jistě jste již našli ve schránce složenku k zaplacení X s dovětkem „pokud jste již zaplatili, ignorujte tuto upomínku“. Doručení maximálně jedenkrát je technologicky poměrně obtížné a vyžaduje (často chybějící) podporu infrastruktury. Blíže se tímto problémem budeme zabývat v kapitolce o *idempotenci*.
- Záruka doručení ve stejném pořadí jako byly zprávy odeslány – často je třeba, aby zprávy byly doručeny ve stejném pořadí, v jakém byly odeslány. Tento požadavek je bohužel v přímém rozporu s technologickými prvky pro maximalizaci propustnosti (multithreading, asynchronnost), tudíž ho lze zajistit pouze za cenu drastického snížení výkonu. Proto je více než vhodné vytvářet rozhraní služeb tak, aby nespoléhaly na přesné pořadí doručení zpráv. V některých situacích to samozřejmě nelze zajistit, například zpráva „založení zákazníka“ musí vždy předcházet zprávu „nová objednávka od zákazníka“ – pak je vhodné řídit pořadí doručení zpráv na vyšší úrovni – ve správě procesů, kterou se budeme zabývat v kapitole 5.
- Záruka vrácení peněz – ne, to byl jenom vtip.

BizTalk server řeší „záruku doručení alespoň jedenkrát“ pomocí dalších pokusů o doručení v případě, že první pokus o doručení nebyl potvrzen. Ostatní záruky mohou být nabízeny jednotlivými adaptéry, ale nejsou a dost dobře ani nemohou být obecnou funkčností integračního brokeru. Doručení alespoň jedenkrát je zaručeno pomocí funkce znovuodeslání (*retry* mechanismus), která je řízena dvěma parametry portu pro odeslání – počtem opakování a intervalem mezi jednotlivými pokusy. Výchozí nastavení je 3 pokusy s intervalem 5 minut. Pokud nedojde k doručení ani v tomto intervalu, je možné pro odeslání použít tzv. sekundární transport, má-li ho daný port pro odeslání nastavený. Jde o druhý pokus o doručení se všemi náležitostmi včetně *retry* mechanismu, který umožňuje doručit zprávy náhradním způsobem. Např. není-li možná okamžitá komunikace prostřednictvím HTTP protokolu, je možné zprávu poslat asynchronně pomocí *message queue*. Pokud selže i tento sekundární transport nebo pokud sekundární transport není nastaven, je zpráva převedena do pozastaveného stavu *(suspended, resumable)*.



Obrázek 3.7: Sledování pozastavených zpráv v nástroji HAT

Všechny takto pozastavené zprávy je možné sledovat v nástroji HAT (*Health and Activitiy Tracking,* viz obrázek 3.7), kde je možné rozhodnout o dalším osudu zprávy – buď o jejím stornování *(terminate)* nebo o vyvolání dalších pokusů o doručení *(resume)*. Pomocí stejného nástroje je samozřejmě možné sledovat i zprávy doručené v minulosti, a to buď v souhrnném multidimenzionálním pohledu, který pomocí technologie OLAP nabízí nepřeberné množství pohledů na metriky fungování systému (viz obrázek 3.8) anebo lze individuálně dohledat každou zprávu systémem prošlou.



Obrázek 3.8: Souhrnné sledování prošlých zpráv podle času a výsledku

Někdy je nutno integrovat systémy, které nejsou k dispozici 24x7, ale pouze v určité dny nebo v určité denní hodiny (tzv. *service window*). Mimo tento časový úsek může probíhat údržba systému, zpracování rozsáhlých dávkových úloh a podobně. BizTalk na tuto skutečnost pamatuje a umožňuje u každého portu definovat tyto „otevírací hodiny“. Pokud přijde požadavek na doručení zprávy tak říkajíc „po zavíračce“ příslušného portu pro odeslání, musí pokorně čekat ve frontě až do otevření portu další den (na rozdíl od pohostinských zařízení, nátlak na číšníka v případě integračního brokeru nepomáhá). Stejného efektu lze dosáhnout i manuálním pozastavením příslušného portu *(Stop),* čehož lze využít např. při nepravidelných, ale plánovaných odstávkách systému.

Idempotence doručení

Na rozdíl od podobně znějící poruchy, idempotence se neléčí modrou pilulkou. Podle slovníku idempotence znamená vlastnost operátoru, který proveden opakovaně dává stejný výsledek jako při jednom provedení (ÂÂ=Â). Některé naše běžně prováděné úkony jsou přirozeně idempotentní – můžete je provádět opakovaně libovolněkrát a konečný stav je vždy stejný, jako např. naformátování pevného disku, doplnění nádrže benzínu na plný stav, zvýšení denního limitu výběru kreditní karty, zrušení trvalého příkazu k úhradě apod. Pokud tyto operace omylem provedete více než jednou, z hlediska konečného stavu se nic nemění. Větší část úloh idempotentní není – mezi ně patří např. koupení nových bot manželce, vhození chilli papričky do guláše, převod 1000 Kč z účtu A na účet B. Pokud tyto úlohy provedete omylem vícekrát, koncový stav je jiný než při jednom provedení, což je mnohdy krajně nežádoucí.

Jestli si teď říkáte, že vás zbytečně zatěžuji absurdními úvahami, pak věžte, že idempotence volání služeb může být v SOA velkým oříškem. Pokud totiž odesílající strana nedostane potvrzení o doručení zprávy, není jisté, zda zpráva nebyla doručena anebo zda byla doručena, ale potvrzení o přijetí nebylo z důvodu nějakého technologického výpadku doručeno (forma „potvrzení“ je technologicky závislá, může to být například stavový kód odeslané HTTP žádosti). Je to stejná situace, jako když někomu dáte pokyn a on vám neodpoví – nevíte, jestli vás neslyšel anebo slyšel a akceptuje váš požadavek aniž by ho jakkoliv potvrdil anebo ho potvrdil, a vy jste potvrzení přeslechli.

Chcete-li zaručit doručení každé zprávy právě jednou, musí být odeslání i přijetí zprávy součástí nějakým způsobem zajištěné transakce, tzn. že dvě elementární operace odeslání a přijetí se buď uskuteční obě nebo žádná. Takové chování je možné snadno zaručit u databázových operací a adaptérů anebo při používání speciální infrastruktury pro přenos zpráv (MQSeries, Microsoft Message Queueing a jiné). Problémem zůstávají technologie, které v sobě nemají

transakční chování zakomponováno. Tento nedostatek je obzvlášť bolestný u HTTP protokolu a u webových služeb, které v zájmu jednoduchosti (bez které by se nikdy nebyly ujaly) transakce a spolehlivé doručení opomíjejí. U webových služeb se už na horizontu zjevuje řešení v podobě standardu *WS-ReliableMessaging,* tento horizont je ovšem pro potřeby dnešních projektů příliš vzdálen.

Protože se však webových služeb jistě nebudete chtít vzdát, budete muset najít řešení, které zní na první pohled jednoduše – „někdo“ musí mít informace (stav) nutné k jednoznačnému určení, zda zpráva již byla doručena nebo nikoliv. „Někdo“ může být zpráva, která zná část stavu služby, například pokud zpráva představuje příkaz k úhradě z účtu, může si pamatovat zůstatek bankovního účtu a služba ji může provést pouze tehdy, pokud informace o zůstatku v doručené zprávě a skutečný stav souhlasí. Zřejmě vás již napadlo, že nejde o nejlepší řešení – bude fungovat pouze tehdy, pokud máme zaručen exkluzivní přístup ke službě a skutečný stav se v mezičase nemohl změnit, což je situace, která nenastává prakticky nikdy (vzpomeňme si z definice SOA, že stav existující mimo službu je pouze nějakým historickým snímkem minulého stavu, nikoliv skutečným stavem). Druhou možností je uspořádání, kdy „někdo“ je služba sama. V příkladu s příkazem k úhradě by to mohlo znamenat očíslování každého vzniklého příkazu jednoznačným identifikátorem a úpravu služby ve smyslu „U každé příchozí zprávy se podívej na identifikátor. Pokud již zpráva s tímto identifikátorem přišla, ignoruj ji. Pokud nepřišla, zpracuj požadavek a ulož identifikátor do tabulky doručených identifikátorů – to celé v jedné transakci“. Jde o způsob, který pro vás bude zřejmě mnohdy nevyhnutelný. Není sice příliš složitý, ale má svoje stinné stránky – často musíte upravovat kód na straně služby, udržovat tabulku doručených identifikátorů a řada dalších drobných komplikací. Ideální by samozřejmě bylo, aby „někdo“ byla infrastruktura. K tomu je ovšem nutná podpora příslušné technologie, a ta u webových služeb chybí. Znovu se tedy dostáváme ke standardu *WS-ReliableMessaging* a k čekání na jeho běžně rozšířenou implementaci.

Mimochodem, zmiňoval jsem se již o tom, že v Dodatku B najdete k této kapitole praktická cvičení, že?

Kapitola 4:

Transformace a routování zpráv

V předchozí kapitole jsme ke všem zprávám přistupovali jako k hloupým sekvencím bytů. Zpráva pro nás byla neprůhlednou zalepenou obálkou, kterou jsme pouze posunovali systémem z jednoho místa na druhé, nanejvýš jsme s ní prováděli operace, které neměnili vnitřní obsah, jako je šifrování/dešifrování, komprese apod. V této kapitole se posuneme o úroveň výše – budeme se zajímat o její obsah, sledovat její správnost, provádět v ní změny a přidáme též inteligenci směrování zprávy podle okolností a obsahu.

4.1 Schéma zprávy

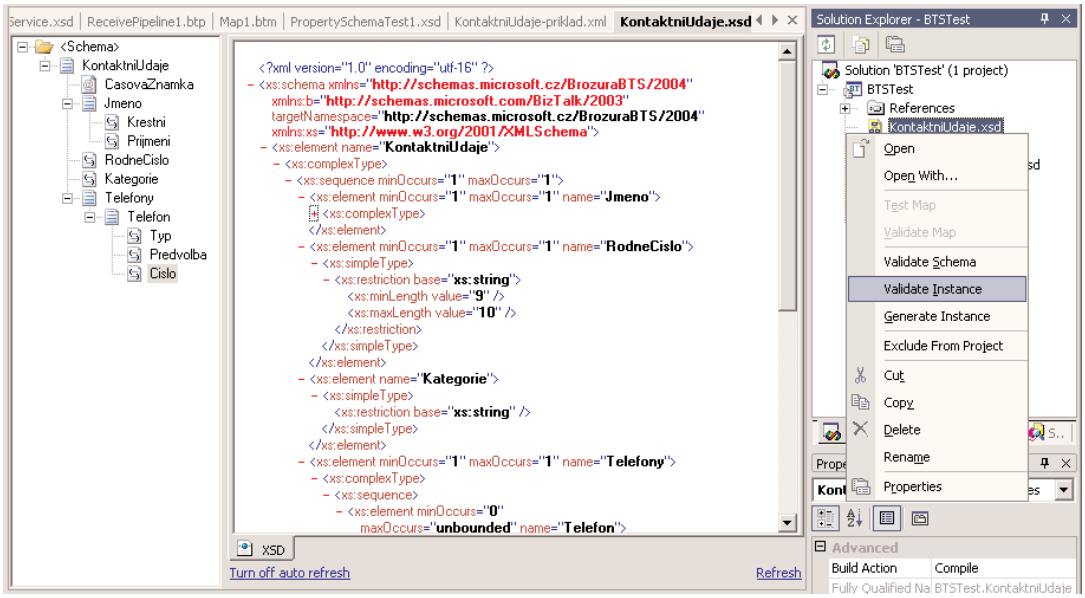
Vytvoření schématu zprávy je zásadní úlohou v integračním projektu a mělo by být výsledkem rozsáhlé analýzy, kterou není možné uspěchat. Tato věta by měla být vytesána do kamene a vsazena na čestné místo nad vaším pracovním stolem. Nemáte-li k dispozici švédskou žulu, zkuste křídový papír a font velikosti 72, tučně a podtrženě. Jde ve své podstatě o analytickou a modelovací úlohu, nezávislou na konkrétním vyjádření schématu (např. XSD), do tohoto vyjádření může být převedeno až dodatečně. Žádný zápis schématu nemůže nahradit dokumentaci, vždy je nutné vytvořit nějaký interní předpis pro dokumentaci schémat zpráv, který může pro zprávu popisující kontaktní údaje vypadat například takto:

Název	A/E/K	Výskyt	Datový typ	Požadavky	Dokumentace
KontaktniUdaje	K	1..1			<dlooooooooouhý text>
> CasovaZnamka	A	1..1	datum a čas		<dlooooooooouhý text>
> Jmeno	K	1..1			<dlooooooooouhý text>
> > Krestni	E	0..1	řetězec		<dlooooooooouhý text>
> > Prijmeni	E	1..1	řetězec	min. 2 znaky	<dlooooooooouhý text>
> RodneCislo	E	1..1	řetězec	9 nebo 10 číslic	<dlooooooooouhý text>
> Kategorie	E	1..1	řetězec	1 velké písmeno	<dlooooooooouhý text>
> Telefony	K	1..1			<dlooooooooouhý text>
> > Telefon	K	0..∞			<dlooooooooouhý text>
> > > Typ	E	1..1	řetězec		<dlooooooooouhý text>
> > > Predvolba	E	0..1	řetězec		<dlooooooooouhý text>
> > > Cislo	E	1..1	řetězec		<dlooooooooouhý text>

Pro jednoduchost a stručnost jsem vynechal další části, jako jsou adresy. V prvním sloupečku je uveden název elementu nebo atributu, počet odrážek určuje úroveň v hierarchii. Ve druhém sloupci je jeho typ. A označuje atribut, E datový element a K kontejner pro datové elementy. Třetí sloupec označuje možný počet výskytů (kardinalitu). Atributy se mohou vyskytovat nejvýše jednou, takže přípustné hodnoty jsou 0..1 nebo 1..1, elementy se mohou opakovat a počet možností je tudíž nepřeberný. Např. element *KontaktniUdaje* musí obsahovat právě jeden element *Telefony,* jenž obsahuje libovolné množství elementů *Telefon,* tedy může být i prázdný. Ve sloupečku pro datový typ je překvapivě uveden datový typ. Požadavky zpřesňují nároky na data, jako např. minimální a maximální počet znaků, dolní a horní hranici číselných typů, výčet možných hodnot, formát řetězce apod. Sloupec dokumentace by ve skutečnosti měl být alespoň odstavcem a hodnoty v něm uvedené berte jako odstrašující příklad. Vám by se něco podobného stát nemohlo, že?

Převedení do XSD

V okamžiku, kdy je schéma vymodelováno, je čas pro jeho formální zápis do XSD dokumentu. Ačkoliv je samozřejmě možné vytvářet XSD schémata „v ruce“ pomocí textového editoru, je mnohem výhodnější použít dostupné grafické editory, které nás oprostí od nutnosti znát rozvleklou a nezáživnou syntaxi XSD. Kvalitní editor XSD dokumentů je i v BizTalk Serveru 2004 (viz obrázek 4.1).



Obrázek 4.1: Editor XSD schématu

Schéma je pak možné vytvářet jako hierarchickou strukturu elementů a atributů, jejichž vlastnosti se nastavují pomocí uživatelsky přívětivého rozhraní, zároveň je možné sledovat vytvářené XSD v textovém tvaru. Pro přehlednost doporučuji u všech prvků nastavovat vlastnosti *minOccurs* a *maxOccurs*, u atributů vlastnost *required*, neboť výchozí hodnoty těchto vlastností nemusí být vždy zřejmé. Editor rovněž umožňuje kontrolovat konzistenci vytvořeného schématu pomocí validace schématu nebo vytvoření vzorové instance dokumentu odpovídající schématu, který může vypadat např. takto:

```
<ns0:KontaktniUdaje CasovaZnamka="1999-05-31T13:20:00.000-05:00"
xmlns:ns0="http://schemas.microsoft.cz/BrozuraBTS/2004/KontaktniUdaje">
    <Jmeno>
      <Krestni>Krestni_0</Krestni>
      <Prijmeni>Pri</Prijmeni>
    </Jmeno>
    <RodneCislo>RodneCislo_0</RodneCislo>
    <Kategorie>Kategorie_0</Kategorie>
    <Telefony>
      <Telefon>
        <Typ>Typ_0</Typ>
        <Predvolba>Predvolba_0</Predvolba>
        <Cislo>Cislo_0</Cislo>
      </Telefon>
      ... kráceno ...
    </Telefony>
</ns0:KontaktniUdaje>
```

Dokument je poté možné doplnit reálnými údaji např. do tvaru

```
<broz:KontaktniUdaje CasovaZnamka="2004-02-27T09:53:12.000+01:00"
xmlns:broz="http://schemas.microsoft.cz/BrozuraBTS/2004/KontaktniUdaje">
    <Jmeno>
      <Krestni>Karel</Krestni>
      <Prijmeni>Novák</Prijmeni>
    </Jmeno>
    <RodneCislo>7003222315</RodneCislo>
```

```
    <Kategorie>C</Kategorie>
    <Telefony>
      <Telefon>
        <Typ>domù</Typ>
        <Predvolba>420</Predvolba>
        <Cislo>235212445</Cislo>
      </Telefon>
      <Telefon>
        <Typ>mobil</Typ>
        <Cislo>666111222</Cislo>
      </Telefon>
    </Telefony>
</broz:KontaktniUdaje>
```

Vytvořený dokument slouží pro testování a kontrolu, můžeme v něm provádět změny a kontrolovat správnou funkci schématu pomocí funkce *Validate Instance*. Bude se hodit i pro další kroky vytváření řešení, pro testování procesů apod. Kompletní příklad vytvoření schématu najdete v příloze B.

Správná definice schématu

Schémata jsou bez nadsázky po technické stránce nejdůležitější částí celého projektu. Schémata jsou pro architekturu založenou na službách tím, čím je pro komponentově nebo objektově orientovanou architekturu správná volba objektového modelu. Pokud se navrhnou špatně, takže je nutné dělat v nich často změny, stává se jakýkoliv rozvoj nebo změna integračního řešení noční můrou. Navrhnou-li se dobře, je možné provádět postupné změny řešení bez dramatických důsledků na funkci celku. Navrhnou-li se bezchybně, nevíme co se bude dít – tohoto výsledku zatím nikdo nikdy nedosáhl. Proto doporučuji se nejprve poohlédnout po existujících schématech v dané oblasti řešení. Např. ve zdravotnictví existují standardy pro výměnu dokumentů HIPAA a HL7. Pokud naleznete vhodný veřejný standard, neváhejte jej použít – je velmi dobrá šance, že takovéto schéma bude lepší a univerzálnější než váš vlastní výtvor, navíc v případě nutnosti výměny dat s externími subjekty se může používání veřejně standardizovaných schémat hodit. Mohou se vám k tomu hodit tzv. akcelerátory řešení dodávané Microsoftem – jejich seznam najdete na *http://www.microsoft.com/biztalk/evaluation/accelerators/,* v současné době jsou k dispozici akcelerátory pro SWIFT, RosettaNet, HIPAA a HL7. Další možností mohou být schémata výrobců existujících programů – pokud již např. účetní systém vyžaduje určité schéma faktury, nemusíte se namáhat s vymýšlením a použijte toto schéma (v ideálním případě budete mít k dispozici dokonce XSD dokument, který potom pouze importujete do integračního řešení). Je také možné, že pro daný problém již existuje velmi dobré v praxi ověřené schéma vlastněné např. konzultační firmou, která má s modelováním problémové oblasti zkušenosti a ráda se nechá na řešení problému najmout.

Často však ani jedna z těchto možností není k dispozici. Přestože budoucí změny nelze vždycky předvídat a schéma bude v budoucnu téměř jistě nutné změnit, je možné alespoň dodržovat určité zásady pro minimalizaci dopadů těchto změn. Uveďme si některé z nich:

- Jednoznačný a neměnný identifikátor – každá zpráva by měla mít jednoznačný identifikátor, který vylučuje její záměnu s jinou zprávou nebo špatnou interpretaci
- Časová známka nebo doba platnosti – každou zprávu je vhodné označit atributem, vypovídajícím o době vzniku zprávy nebo době platnosti zprávy (vzpomínáte na poučku, že stav v podobě zprávy existující mimo službu je pouze historickým obrazem skutečného stavu?). Přestože se vám může zdát, že tuto vlastnost zatím nepotřebujete, je celkem pravděpodobné, že se vám v budoucnu bude hodit. Buď můžete zprávu označit dobou vzniku (datum výroby) nebo dobou platnosti (datum doporučené spotřeby) nebo klidně obojím, tak jak to požaduje Evropská unie u potravin.
- Elementy nebo atributy? – přestože elementy i atributy v XML slouží podobnému účelu, je zvykem používat mnohem flexibilnější elementy k vyjádření vlastních dat a atributy případně používat k zápisu metadat o původu dat, průběhu zpracování apod. V našem případě je jediným atributem časová známka, všechno ostatní jsou datové elementy. K podobnému přístupu se dnes přiklání většina knih o práci s XML.
- Šetřete s definováním povinného výskytu *(minOccurs=1)* – vyžadujte pouze elementy, které jsou opravdu zcela nezbytné. V našem příkladě jsme vyžadovali pouze příjmení, křestní jméno nemusí být ve výjimečných případech známé. Vyžadováním příliš mnoha údajů si navíc můžete „znečistit“ datovou základnu zástupnými údaji jako je ?, xxx, 9999 apod.

- Nevytvářejte příliš specifické struktury – pokud budete příliš konkrétní, zaskočí vás jakákoliv neočekávaná změna daleko pravděpodobněji. Pokud byste například byli definovali (podmiňovací způsob minulý!) fragment pro telefonní čísla následujícím způsobem, zaskočil by vás příchod mobilních telefonů

```
<TelefonDomu>
       <Predvolba>420</Predvolba>
       <Cislo>235212445</Cislo>
</TelefonDomu>
<TelefonZamestnani>
       <Typ>mobil</Typ>
       <Cislo>666111222</Cislo>
</TelefonZamestnani>
```

- Zvažte možnost použití divoké karty – XSD umožňuje definovat speciální elementy *any* a *anyAttribute,* které znamenají zhruba „a nyní už může následovat cokoliv“. Někteří autoři doporučují jejich používání jako prostředek očekávání budoucích změn, které potom nenaruší funkci aplikace používající původní verzi schématu, obdrží-li dokument podle nové verze s přidanými prvky. Na druhou stranu je třeba si uvědomit, že přidáním těchto divokých karet se snižuje schopnost schématu kontrolovat správnost XML dokumentu. Přiznám se, že osobně nejsem přítelem těchto konstrukcí a upřednostňuji spíše disciplinovanější přístup – s každou změnou schématu změnit verzi a tím i jmenný prostor dokumentu. Aplikace používající starší verzi dokumentu je pak možné ošetřit jednoduchou transformací, o které se dočteme v příští podkapitole.
- Správná granularita (velikost zpráv) – neexistuje jednoznačný recept, jaká má být správná velikost zprávy. Přesto jde o velmi důležitou otázku. Hrubým pravidlem může být způsob transakčního zpracování uvnitř služby. Jednotkou nedělitelného transakčního zpracování může být např. objednávka. Pokud byste ji rozdělili do několika zpráv po jednotlivých řádcích, museli byste někde uvnitř služby držet stav příchozích zpráv, být schopni detekovat ztracenou nebo opožděnou zprávu, apod. – nepříjemná práce navíc. Pokud byste naopak zabalili dávku více objednávek do jedné zprávy, museli byste složitěji ošetřovat situaci, kdy některé objednávky v dávce jsou přijaty a jiné z jakéhokoliv důvodu odmítnuty – opět nepříjemná práce navíc. Intuitivní a správnou volbou v tomto případě je tedy: jedna objednávka = jedna zpráva. Kéž by bylo vždycky všechno takhle jednoduché.
- Předvídejte drobné změny – dá se očekávat, že schéma bude v brzké době vystaveno tlakům na změny v důsledku změny vnějších podmínek. Např. je přijat zákon upravující používání osobních údajů a naše schéma pro kontaktní údaje s tímto nepočítá. Pokud jde o trvalou změnu, je možné tuto skutečnost vyřešit změnou schématu, bohužel se všemi nepříznivými důsledky. Změně schématu se lze vyhnout nebo ho alespoň oddálit určitou předvídavostí při vytváření schématu. Můžete do něj předem zanést strukturu pro ukládání těchto nečekaných údajů, např. v podobě poznámek:

```
<Poznamky>
       <Poznamka klic="ZpracovaniOU">true</Poznamka>
       <Poznamka klic="OblibenaBarva">rùžová</Poznamky>
</Poznamky>
```

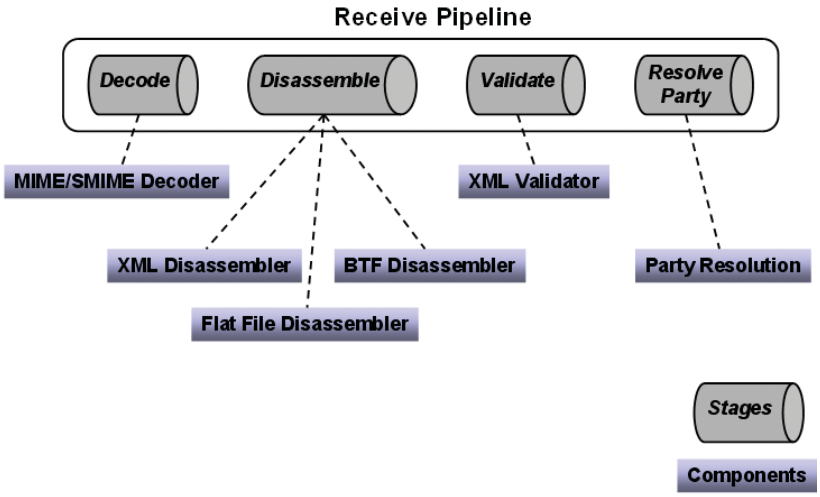
4.2 Konverze zpráv

Při přenosu zpráv mezi dvěma službami je často nutno měnit její obsah, ať už pouze po stránce formální jako je převod mezi znakovými sadami anebo konverze mezi formáty (např. z EDI do XML) nebo po stránce věcné (konverze dokumentů mezi aplikacemi používajícími jinou sémantiku téhož dokumentu).

Translace zprávy

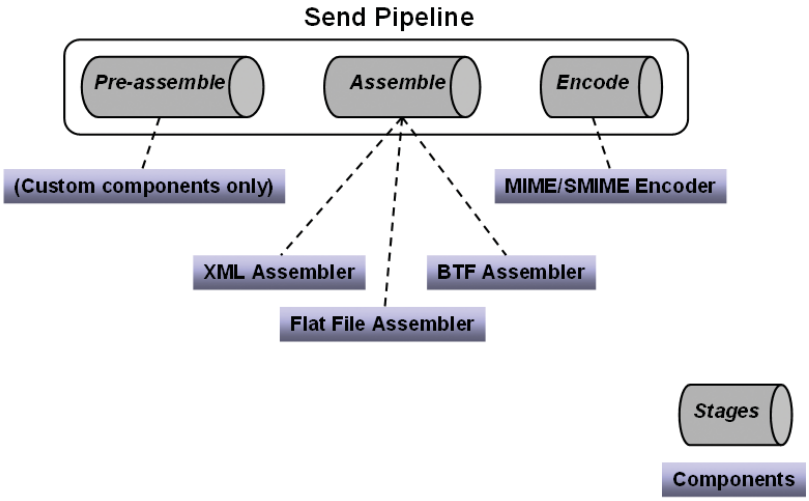
Translace zprávy je formální změna jejího zápisu či formátu. Každou zprávu je možné považovat za abstraktní strom pojmenovaných hodnot.Tento strom hierarchických informací se v terminologii XML nazývá *infoset,* ale je poměrně lhostejné, jak si ho pojmenujeme, není technologicky závislý. V praxi potřebujeme tento strom informací „zhmotnit", převést na určitou konkrétní posloupnost bytů či znaků podle nějakého formátu, ze které je poté možné informační hodnotu rekonstruovat. Právě tato skutečnost vedla ke vzniku XML, které je ve své podstatě jednotným, na platformě nezávislým zápisem informačního stromu. Není proto divu, že BizTalk Server 2004 používá interně XML jako formát reprezentace zpráv. Nicméně v praxi je nutno komunikovat i se systémy používající jiné formáty zápisu, jako například textové soubory s oddělovači, textové soubory s pevnými pozicemi dat, EDI formát, sešity Excelu s příponou XLS

a celou řadu dalších. Proces převodu mezi těmito formáty a interní XML reprezentací se nazývá translace. K translaci dochází uvnitř *pipeline* komponent. Podívejme se nejprve na jednotlivé fáze přijetí zprávy znázorněné na obrázku 4.2.



Obrázek 4.2: Posloupnost fází při přijetí zprávy

První fází je dekódování *(Decode),* ve které se se zprávou zachází jako s celkem. Jediná dodávaná komponenta je *MIME/SMIME Decoder* pro dešifrování a kontrolu digitálního podpisu. Je možné vložit do této fáze i vlastní komponentu, např. pro dekompresi ZIP souborů. Druhou fází je *Disassemble* sloužící k převodu z nativního formátu do interního XML dokumentu, dodávané jsou disassemblery pro XML soubory, textové soubory (s oddělovači nebo pevnou poziční strukturou) a soubory zabalené do hlaviček *BizTalk Framework 1.0*. Některé z nich mohou plnit též další funkce, jako například odstranění hlavičky a patičky nebo rozdělení příchozí zprávy na více zpracovávaných částí (např. rozdělit dávku objednávek na jednotlivé objednávky). Je možné vytvořit též vlastní komponenty pro fázi *Disassemble,* například pro zpracování **.XLS* souborů Excelu nebo **.MDB* souborů Accessu. Ve třetí fázi validace *(Validate)* je možné kontrolovat strukturu podle jednoho nebo více definovaných XSD schémat, případně můžete tuto fázi doplnit o další kontroly. Pokud jakákoliv komponenta v této fázi selže, je zpráva vyhodnocena jako neplatná a její další zpracování je pozastaveno. Poslední fáze slouží pro nalezení identifikátoru partnera *(Resolve Party),* věnovali jsme se jí už v kapitole 3.3 a ještě se jí budeme věnovat v příloze A.

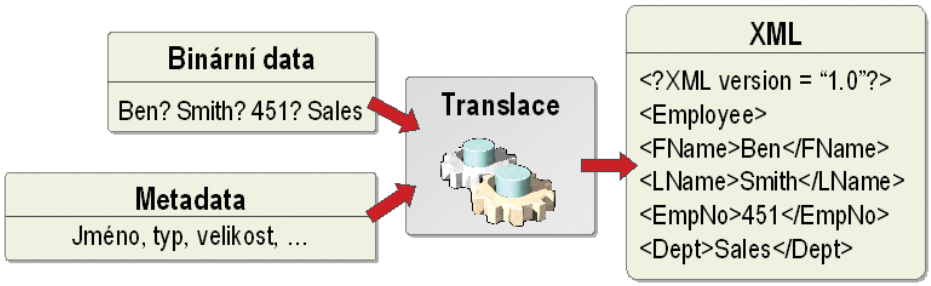


Obrázek 4.3: Posloupnost fází při odeslání zprávy

Zpracování zprávy při jejím odesílání je ze značné části zrcadlovým obrazem přijetí (viz obrázek 4.3).V první fázi *Pre-assemble* můžete umístit svoje vlastní komponenty provádějící libovolné operace se zprávou. Mnohem zajímavější je druhá fáze *Assemble,* ve které dochází k převodu z interního XML formátu do nativních formátů, dodávané jsou opačné komponenty k fázi *Disassemble,* tj. assemblery pro XML soubory, textové soubory

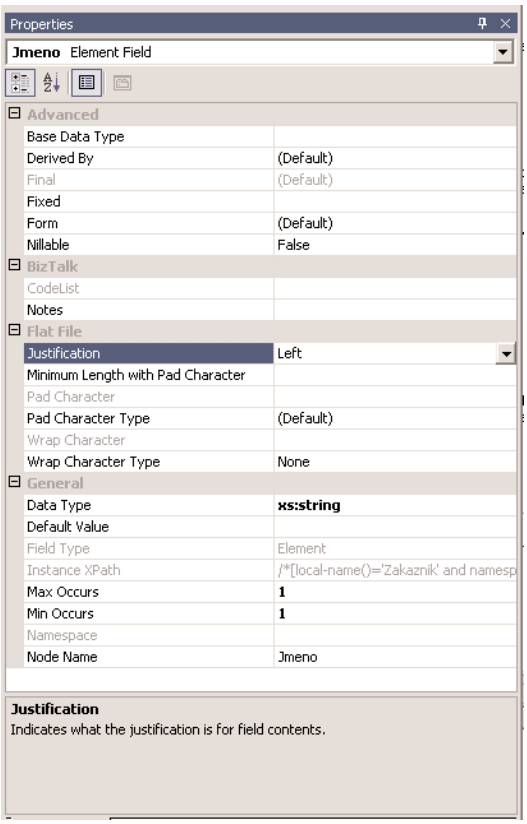
(s oddělovači nebo pevnou poziční strukturou) a soubory zabalené do hlaviček *BizTalk Framework 1.0*. Assemblery mohou plnit též další funkce, např. *XML Assembler* umožňuje do XML dokumentu vložit instrukce pro zpracování *(processing instruction)* nebo úvodní deklaraci dle standardu XML. Ve třetí fázi *Encode* je možné provádět některé další úpravy se zprávou jako s celkem. Jediná dodávaná komponenta pro MIME/SMIME umožňuje kódování podle standardu MIME s možností přidání digitálního podpisu nebo šifrování. A opět se musím opakovat, i tuto fázi je možné obohatit o vlastní komponenty, např. pro kompresi odchozí zprávy do formátu ZIP.

Je zjevné, že k translaci zprávy z/do XML formátu je kromě XSD schématu výsledné zprávy potřeba i řada dalších metadat – viz obrázek 4.4.



Obrázek 4.4: Translace binárních dat do XML formátu

Metadata je třeba někam vhodně uložit. Vzhledem k tomu, že metadata a XSD schéma dokumentu tvoří těžko dělitelný celek, je výhodné použít prvek *annotation* z definice XSD schémat. Editor schémat v BizTalk Serveru 2004 je rozšiřitelný o doplňky zapisující anotace do schématu. Jako příklad může sloužit obrázek 4.5, ze kterého je patrné, že ve vlastnostech elementu pak můžeme nastavovat vlastnosti vztahující se k XSD schématu (např. datový typ a počet výskytů) spolu s vlastnostmi pro příslušnou extenzi – v tomto případě nastavení formátu textového souboru pro zápis (*Flat File* – oddělovače jednotlivých polí, zarovnávání vlevo nebo vpravo, znaky vymezující obsah pole atd.). Toto uspořádání je velmi praktické, kompaktní a intuitivní, formátovací informace jsou přirozenou součástí schématu při dodržení všech standardů.



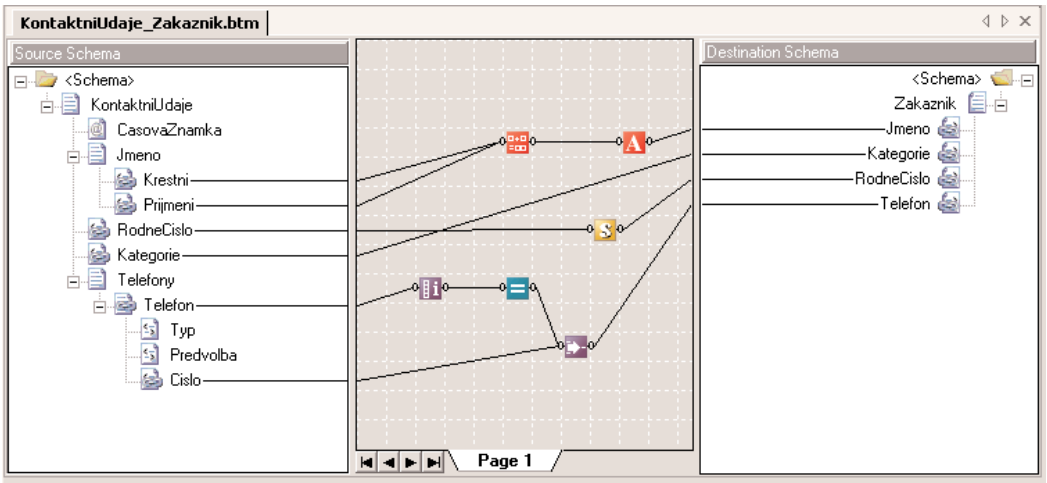
Obrázek 4.5: Schéma dokumentu s anotacemi pro textové soubory

Transformace zprávy

Zatímco translace převáděla mezi jednotlivými formáty (zápisy) zprávy aniž by se měnila informační hodnota, transformace již provádí změny v informacích obsažených ve zprávě. Příklady prováděných změn mohou být:

- kopírování obsahu – beze změny názvu prvku, spojené s přejmenováním prvku, spojené se změnou pozice nebo hloubky hierarchie prvku, ...
- operace s řetězci – převod na velká/malá písmena, spojování řetězců, výběr podřetězce, odstraňování mezer zleva a zprava, ...
- aritmetické operace – sčítání, odečítání, násobení, dělení, zaokrouhlování, odmocnění, sinus, logaritmus, ...
- konverzní funkce – převod mezi datovými typy, převod čísla z hexadecimálního tvaru do desítkového, ...
- agregace – součet, průměr, počet výskytů, minimální a maximální hodnota, ...
- operace s datem a časem – doplnění aktuálního data/času, přidání časového intervalu, ...
- logické a relační operátory – porovnávání velikosti (a>b), zjišťování platnosti typu (b je číslo), logické operace (AND, OR), ...
- získání externí hodnoty – doplnění informací o kontextu zpracování, zjištění databázové hodnoty (dohledání názvu položky v číselníku), ...
- změny struktury – zploštění hierarchické struktury, vytvoření hierarchické struktury z ploché, rozhodování v závislosti na počtu průchodů, ...
- cokoliv dalšího neuvedeného výše

Pro transformace dat je ideální použít standard XSLT probraný v kapitole 2.3. Tento standard umožňuje převádět mezi dvěma XML dokumenty pomocí šablon. Vzhledem k tomu, že BizTalk Server 2004 používá interně formát XML bez ohledu na skutečný formát dokumentu, využití XSLT se doslova nabízí. Proto nás jistě nepřekvapí, že BizTalk Server 2004 obsahuje nástroj pro vytváření transformačních map nazývaný *mapper* (viz obrázek 4.6)



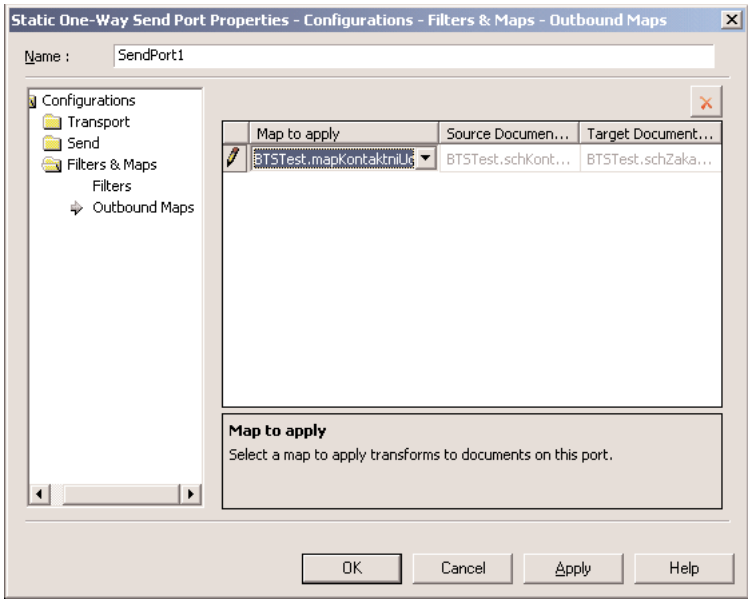
Obrázek 4.6: Vizuální vytváření transformační mapy

Přestože mapper je grafický editor k vytváření XSLT šablon, nelze ho považovat za univerzální XSLT nástroj, je specializovaný pro použití v integračních scénářích. Na levé straně je schéma zdrojového dokumentu, na pravé straně cílový dokument. Plocha uprostřed slouží k vytváření transformačních vazeb. Rovná čára označuje přenos hodnoty a vytváří se „opravdovými profesionály“ opovrhovaným, leč velmi snadným přetažením myší. Barevné kostičky se nazývají poněkud krkolomně functoidy a představují zařazené funkce (XSLT je ve své podstatě funkcionální jazyk).

Místo dlouhého teoretizování si rozeberme výše uvedený příklad. Křestní jméno a příjmení jsou spojeny *(String Concatenate),* přičemž je zároveň vložena mezera a poté jsou převedeny na velká písmena *(Uppercase)*. Kategorie je beze změny zkopírována do odpovídajícího cílového pole. Rodné číslo je upraveno pomocí vlastní funkce *(Scripting),* obsahující vloženou funkci v C# pro umístění lomítka do řetězce rodného čísla (`return rc.Substring(0,6)+"/"+rc.Substring(6);`). Tuto operaci by bylo možné provést též pomocí několika functoidů pro práci s řetězci, ovšem použití vloženého skriptu je o něco jednodušší, a umožňuje nám vyzkoušet si

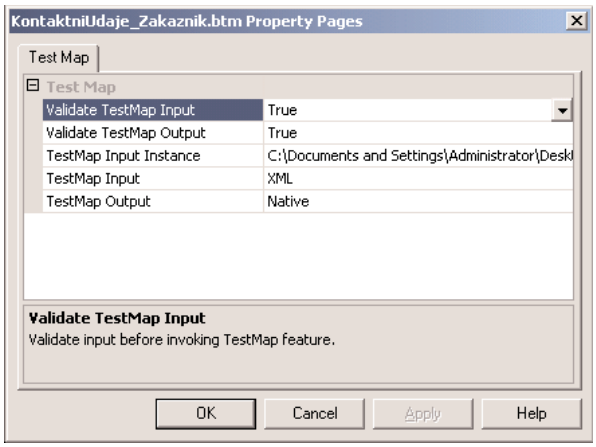
něco nového, dosud nepoznaného. Nejsložitější operace se provádí s telefonním číslem – cílová specifikace umožňuje uložit pouze první uvedené telefonní číslo bez předvolby. Proto použijeme nejprve functoid pro zjištění pořadového čísla prvku *(Iteration)* pro telefon, ve druhém functoidu *(Equal)* porovnáme tuto hodnotu s konstantou 1 – tím získáme hodnotu *true* pouze pro první výskyt prvku *Telefon*. Výsledek tohoto functoidu poté použijeme ve functoidu pro mapování a zploštění struktury (*Value Mapping – Flattening*), který nám vybere hodnotu elementu Cislo pouze pokud je první vstupní údaj na logické hodnotě pravdivosti.

Vytvořenou transformační mapu můžeme využít dvojím způsobem. První možností je při přijetí nebo odeslání zprávy BizTalkem na úrovni portu (viz obrázek 4.7). Při použití mapy na portu pro přijetí se tato možnost jmenuje *Inbound Maps,* na portu pro odeslání se jmenuje *Outbound Maps*. Druhou možností je použití mapy při definici obchodního procesu v kroku *Transform*. To bychom ale zbytečně předbíhali, počkejme si na pátou kapitolu.



Obrázek 4.7: Použití transformační mapy na portu pro odeslání

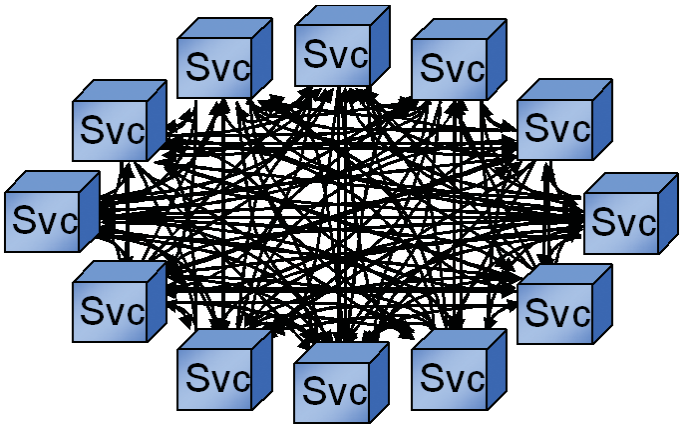
Pro otestování vytvářené mapy nám velmi dobře poslouží funkce *Test Map* v kontextové nabídce mapy. Nejprve je nutné nastavit parametry testování mapy (viz obrázek 4.8), zejména vstupní dokument, který hodláme transformovat. Volitelně můžeme též provádět validaci vstupního a výstupního dokumentu podle jejich schématu, přičemž vstupní a výstupní dokument nemusí být pouze v XML, ale též v nativním tvaru. Funkce testování map je velmi praktická a vývojáři ji při vytváření map jistě mnohokrát ocení.



Obrázek 4.8: Nastavení parametrů pro testování mapy

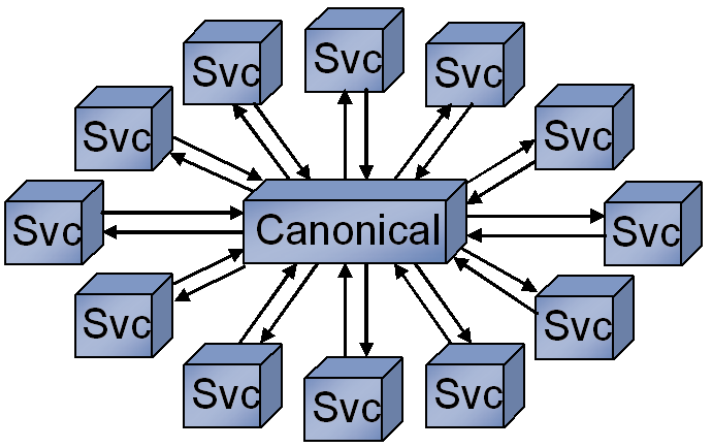
Kanonické schéma

Při integraci je často nutné spojovat větší množství různých systémů, takže jedna logická datová entita (např. zákazník) může být v různých systémech reprezentována různým způsobem a tudíž příslušet mnoha různým schématům. Při vytváření transformací dat je pak nutné vytvořit velké množství různých transformací mezi nimi podobně jako na obrázku 4.9. Pokud oprášíte středoškolskou matematiku, mělo by vám vyjít, že při N různých schématech je počet všech možných jednosměrných transformací mezi nimi roven N*(N-1).



Obrázek 4.9: Transformace dat způsobem „každý s každým“

K překonání tohoto krajně nežádoucího trendu se používá technika takzvaného kanonického schématu, což je jediné referenční schéma pro danou entitu. Mezi jednotlivými schématy se pak dokumenty převádí za použití kanonického schématu jako mezistupně (viz obrázek 4.10). Pokud oprášíte matematiku ze základní školy, mělo by vám vyjít, že při N různých schématech a jednom kanonickém je počet všech potřebných jednosměrných transformací mezi nimi roven 2*N. Srovnáním tedy snadno zjistíme, že již při 4 různých schématech se nám použití kanonického schématu vyplatí.



Obrázek 4.10: Transformace dat s použitím kanonického schématu

Jakým způsobem kanonické schéma definovat? Univerzální správná odpověď bohužel neexistuje. Je zřejmé, že kanonické schéma musí obsahovat všechna data ze všech možných reprezentací entity (matematik by řekl sjednocení). Na druhou stranu není v tomto schématu možné požadovat existenci prvků, které nejsou společné všem schématům (chcete-li, průnik), tyto prvky musí být samozřejmě nepovinné. V ideálním případě by měl být povinným prvkem jednoznačný identifikátor dané entity (u osob by to mohlo být rodné číslo), ale pozor ne vždycky je možné takovýto identifikátor táhnoucí se napříč všemi systémy nalézt! Zkrátka a dobře, platí-li, že správná definice schématu je klíčová pro správnou funkci, u kanonického schématu to platí dvojnásob. Někdy se může poštěstit, že schéma některé aplikace bude možné zároveň použít jako kanonické schéma, ale buďme upřímní, moc pravděpodobné to není.

Kromě snížení celkového počtu transformací má použití kanonických schémat i další výhody. Izoluje aplikace od sebe navzájem, takže při změně schématu v jedné aplikaci stačí aktualizovat pouze dvě transformace, zatímco bez kanonického schématu byste museli upravit 2*(N-1) vzájemných transformací. Kanonický model též pomáhá definovat ideální strategický datový model organizace a oprostit se od krátkodobých schémat vnucených dodavateli jednotlivých aplikací v jejich současné verzi. Na druhou stranu, při použití kanonického schématu je pro převod mezi dvěma dokumenty nutné provést dvě transformace a tudíž spotřebovává dvakrát více výpočetních prostředků. Vzhledem ke klesajícím cenám paměti a zvyšující se rychlosti procesorů ale tento argument jednoznačně ztrácí na důležitosti.

4.3 Routování zpráv

Jednou z velmi důležitých vlastností integračního řešení je tzv. routování zpráv, tedy určení příjemců pro každou přijatou zprávu. K této situaci je možné přistoupit dvojím způsobem. První možností je specifikace určitých pravidel pro příchozí zprávy, na základě kterých je zpráva distribuována příjemcům. Takto funguje většina distribučních procesů v našem reálném životě, například marketingová oddělení firem vezmou svoji supervýhodnou nabídku spolu s výsledkem dotazu do databáze kontaktů a nám všem se pak plní odpadkové koše papírem nebo elektrony. Druhou možností jsou pravidla stanovená příjemci – takto by mohla fungovat např. služba pro odesílání SMS zpráv o dopravní situaci. Každý příjemce si určí jaké zprávy chce dostávat (zácpy, dopravní nehody a umístění radarů v určitém regionu, v určitých denních hodinách). Pokud potom přijde do systému zpráva, je doručena příjemcům, jejichž nastavená pravidla splňuje.

Právě tento druhý způsob je zpravidla používán v implementacích integrační sběrnice, jakou je i BizTalk Server 2004. Jednotlivé integrované systémy nebo uživatele doručují na sběrnici zprávy, které jsou odebírány jednotlivými příjemci na základě pravidel, která si příjemci sami stanoví, stejně jako v našem příměru s SMS zprávami. Popsaný dvoufázový mechanismus se odborně označuje jako *Publish/Subscribe* metafora. Má velmi blízko ke koncepci čisté integrační sběrnice, ve které jednotlivé aplikace či služby vytvářejí neadresné události na sběrnici v podobě zpráv (například aktualizace kontaktních údajů), přičemž sběrnice je doručuje jednotlivým předplatitelům ke zpracování dle jejich vlastního uvážení.

Fáze první – Publish

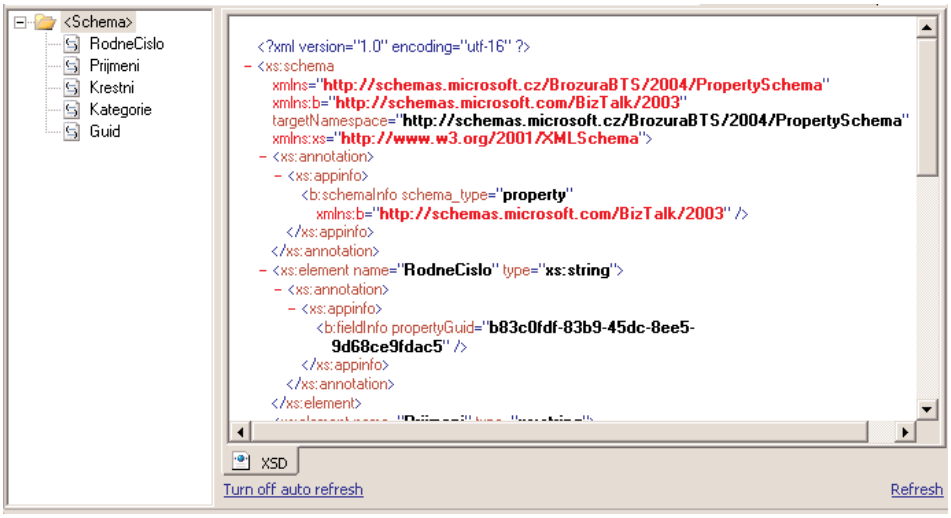
Začněme nudnější první fází. Jak jsme již mohli vidět v kapitole 3.1, je zpráva doručena na port pro přijetí zprávy *(receive port)* prostřednictvím *receive location,* poté sekvenčně prochází jednotlivými fázemi své *receive pipeline,* načež skončí v databázi *message box* (Palackého by moje čeština zjevně nenadchla). Tím bychom mohli tuto podkapitolu ukončit nebýt jednoho velmi význačného prvku, kterým jsou vlastnosti příchozí zprávy. Jediným zajímavým úkolem první fáze je totiž vygenerovat soubor jejich vlastností *(properties),* na základě kterých se v druhé fázi rozhoduje o odeslání zprávy příslušným předplatitelům.

Zpráva sama o sobě je neměnná. Při přijetí je opatřena tzv. kontextem, který si představme jako určitou formu průvodní složky. Pokud se během procesu přijetí zpráva modifikuje, např. při převodu z textového formátu do XML, původní zpráva ve složce zůstává a pouze se přiloží její modifikovaná verze (pamětníci si představí kádrový profil, ostatní třeba zdravotní kartu u svého obvodního lékaře). Dále je zpráva označena různými vlastnostmi a jejich hodnotami, což lze přirovnat k přilepování žlutých samolepících papírků ke složce. Pro rychlou orientaci pak není nutné listovat složkou, stačí prohlédnout přilepené lístky – optimalizace ve světě informačních technologií funguje podezřele podobně jako ve světě lidí. Tyto vlastnosti (samolepící papírky) mohou mít různý původ:

- Vlastnosti (metadata) vytvořené nezávisle na konkrétní technologii – např. podepisovací certifikát, jméno portu použitého pro přijetí zprávy, identifikátor účtu použitého při procesu doručení, jednoznačný identifikátor zprávy apod.
- Vlastnosti (metadata) specifické pro použitou technologii (adaptér) – jméno fronty při použití technologie *Message Queueing,* odesílající server při použití SMTP pošty, typ autentizace při použití protokolu HTTP, jméno souboru při přijetí prostřednictvím protokolu FTP apod.
- Vlastnosti (data) získané z vlastní zprávy – např. IČO odesílatele faktury, celková částka objednávky, rodné číslo žadatele apod.

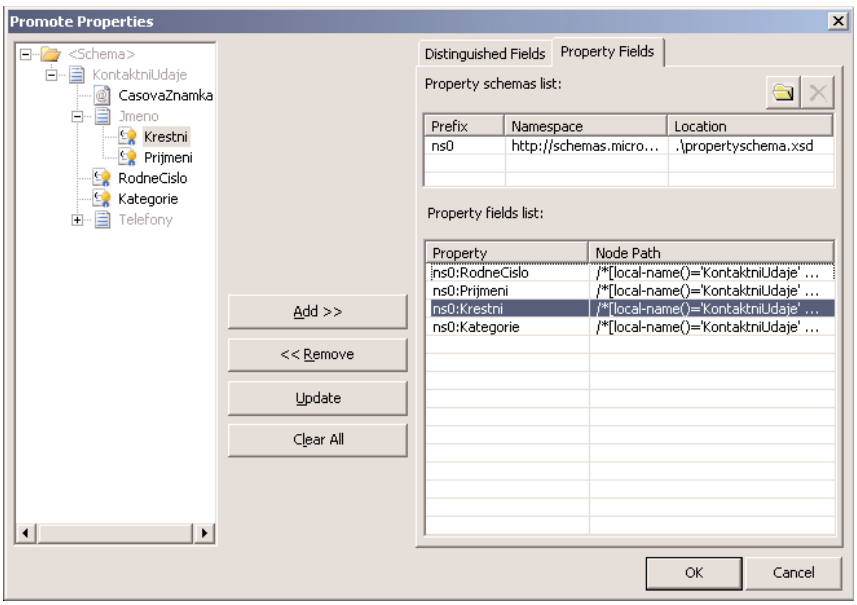
První dvě kategorie vlastností jsou vytvářeny bez našeho vědomí na základě rozhodnutí výrobce BizTalk Serveru (není to Microsoft?), případně adaptérů. Třetí a nejdůležitější skupinu vlastností máme zcela pod vlastní kontrolou. Protože ukládání všech možných vlastností ze zprávy by bylo zbytečným plýtváním, ukládají se pouze takzvané význačné

vlastnosti *(promoted properties)* speciálně označené tvůrcem schématu. Čím více vlastností označíme, tím více možností rozhodování a sledování zprávy, ale i větší režie zpracování zprávy, takže samozřejmě všeho s mírou. Pro vytvoření význačných vlastností je nejprve nutné vytvořit schéma těchto vlastností *(property schema)* – viz obrázek 4.11.



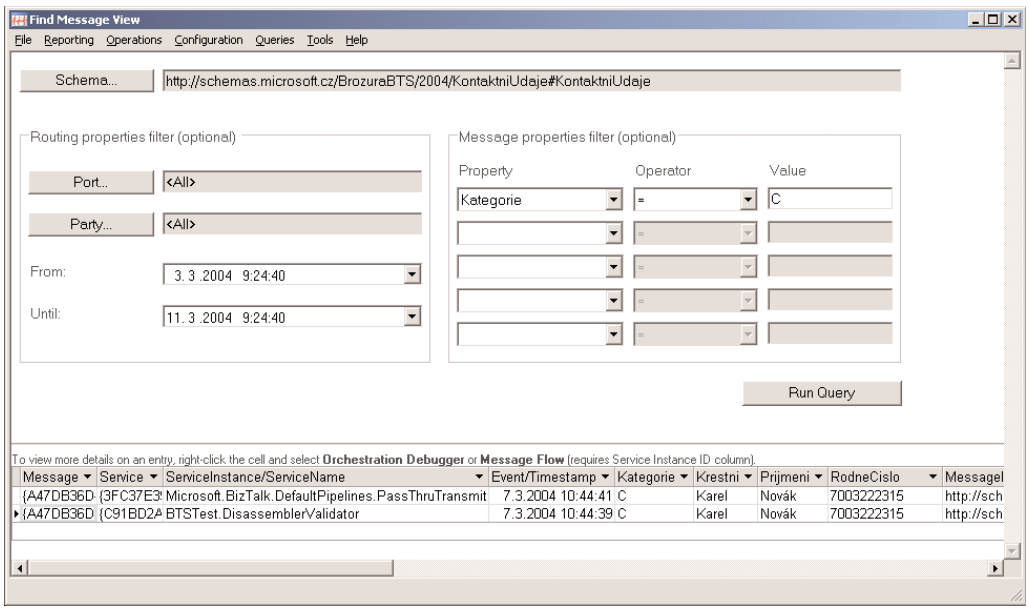
Obrázek 4.11: Schéma význačných vlastností zpráv

Jde o standardní XML schéma s jednoúrovňovou strukturou, ve kterém jsou uvedeny jednotlivé vlastnosti, jejich datové typy a jednoznačné identifikátory. Jsou uloženy ve zvláštním souboru, aby bylo možné sledovat určité vlastnosti napříč různými schématy. Například rodné číslo může být obsaženo v různých dokumentech (XSD schématech) pod různým názvem a v různých jmenných prostorech. Vytvořením zvláštní definice význačných vlastností je možné tyto rozdíly sjednotit. Přiřazení význačných vlastností jednotlivých schémat se nazývá *promotion* – viz obrázek (4.12).



Obrázek 4.12: Přiřazení význačných vlastností schématu

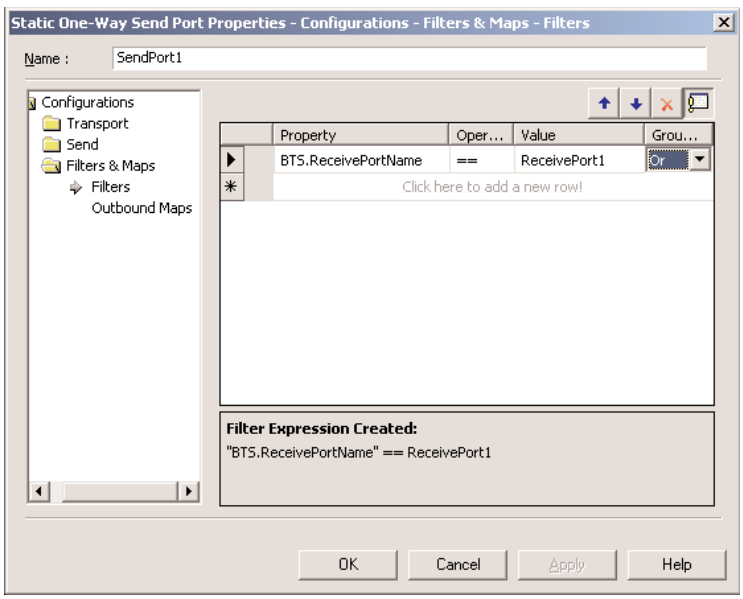
Jde o mapování elementů ze schématu dokumentu na jednotlivé elementy schématu vlastností pomocí anotací v XSD schématu dokumentu. V našem příkladě byly povýšeny vlastnosti křestní jméno, příjmení, rodné číslo a kategorie, které jsou navíc vizuálně označeny zlatou pečetí (pokud máte méně fantazie, tak oranžovým kolečkem a dvěma modrými čarami). Význačné vlastnosti nás příjemně překvapí i v nástroji HAT (*Health and Activity Tracking* – viz obrázek 4.13), neboť díky svému uložení do databáze během zpracování v *receive pipeline* jsou přístupné pro snadné vyhledávání – v našem případě jsme hledali veškeré dokumenty typu *KontaktniUdaje,* které prošly systémem a jejichž vlastnost *Kategorie* je rovna C.



Obrázek 4.13: Sledování význačných vlastností v nástroji HAT

Fáze druhá – Subscribe

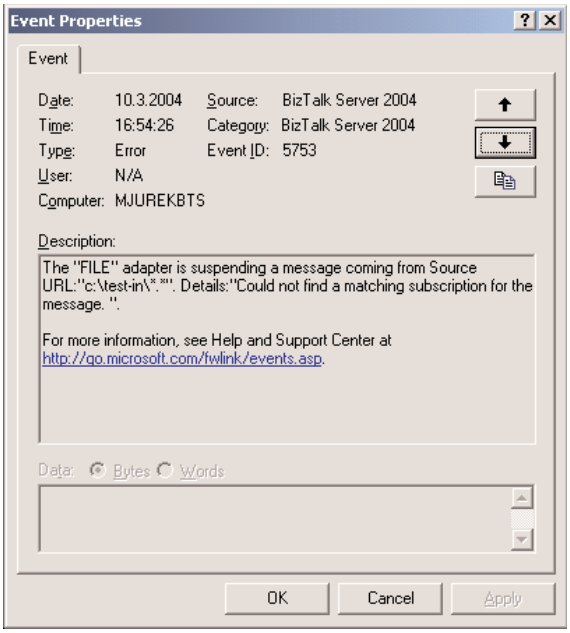
První fáze se dá s trochou nadsázky označit jako příprava podkladů pro rozhodování, které nastane ve fázi druhé, nazývané *subscribe*. Rozhodující roli zde hrají porty pro odeslání *(send port),* případně skupiny těchto portů *(send port group)*. Každý z nich si lze přestavit jako potenciálního předplatitele zpráv doručených do systému. Předplatné se provádí pomocí nastavení filtru (viz obrázek 4.14) určujícího, které zprávy přijaté do *message boxu* jsou poté odeslány tímto portem. Filtr je ve své podstatě podmínkou – logickým výrazem nebo kombinací logických výrazů spojených logickými operátory OR nebo AND. Každý výraz je potom srovnáním určité vlastnosti příchozí zprávy (zapojme krátkodobou paměť a vzpomeňme na vlastnosti ve fázi *publish*) s nějakou hodnotou, případně může být pouhým otestováním existence vlastnosti.



Obrázek 4.14: Nastavení filtru na portu pro odeslání

Tímto způsobem lze realizovat celou škálu scénářů. Nejjednodušším je tzv. *pass-through,* kdy port pro odeslání odebírá bez přemýšlení všechny zprávy, které dorazily na určitý port pro přijetí (viz obrázek 4.14, filtrování se provádí podle vlastnosti *BTS.ReceivePortName*). Komplexní scénář může být například „všechny dokumenty typu objednávka s celkovou částkou vyšší než 1 milión Kč se způsobem platby hotově“.

Někdy se též může stát, že přijatou zprávu najdete v nástroji HAT ve stavu *Suspended (not resumable)* a v protokolu událostí najdete podobnou zprávu jako na obrázku 4.15. Chybová hláška je poměrně samopopisná – do *message boxu* byla doručena zpráva, pro kterou nebyla nalezeno žádné předplatné. BizTalk Server je proto na rozpacích, jak s touto zprávou naložit, proto je zpracování zprávy pozastaveno a je vygenerovaná chyba – jde jednoznačně o nežádoucí situaci. Možná jste se do tohoto stavu dostali jednou špatnou úvahou. Port pro odeslání, který nemá nastavený žádný filtr nemá předplacené všechny zprávy. Ve skutečnosti nemá předplacenou žádnou zprávu. Pokud chcete nějakou zprávu předplatit, vždy musíte nastavit filtr.

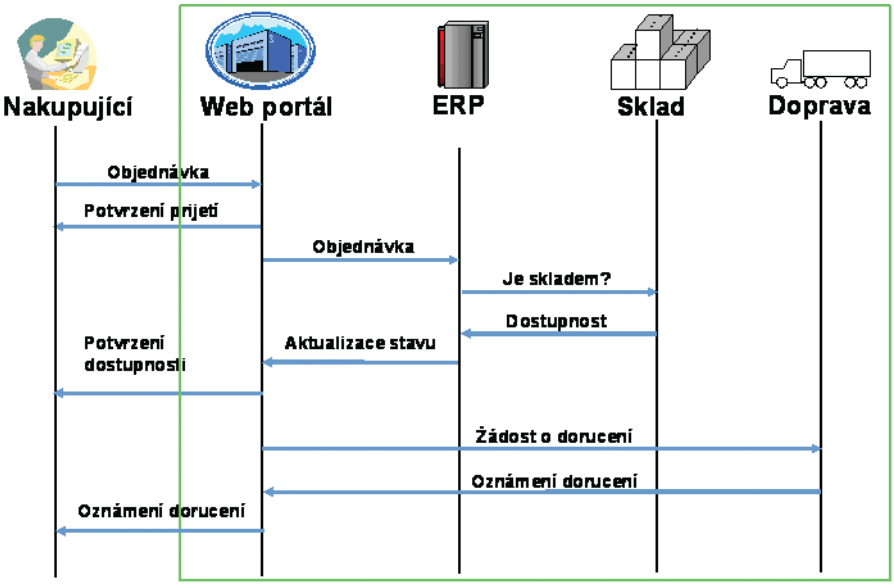


Obrázek 4.15: Každá doručená zpráva musí mít alespoň jednoho předplatitele

Nerad bych se opakoval, nicméně upozorňuji, že praktické příklady k této kapitole naleznete opět v příloze B. V příští kapitole se povzneseme ještě výše. Opustíme svět jedné izolované zprávy a navštívíme svět procesů, tedy jakési koordinované posloupnosti akcí a vyměněných zpráv. Vzrušuje vás ta představa?

Kapitola 5:
Správa procesů

Vrstva správy obchodních a administrativních procesů *(business process management, BPM)* je zřejmě nejzajímavější, neboť přináší velký skok v úrovni abstrakce. Dosud jsme se zabývali pouze elementárními kroky (přijetí žádosti, zápis do účetnictví apod.), které sice rozhodně nejsou nezajímavé nebo nedůležité, ale jsou příliš úzké a detailní. Činnost organizace nebo firmy lze zpravidla popsat určitou sadou procesů vyšší úrovně (zpracování žádosti o změnu kontaktních údajů, realizace objednávky, otevření hypotečního úvěru apod.). Tyto procesy typicky sestávají z jednotek až desítek výše zmíněných elementárních kroků. Příkladem nám může být např. zpracování objednávky zboží přijaté samoobslužným způsobem jakým je internet, který je graficky znázorněn na obrázku 5.1. Při pohledu z výšky jde o jediný proces (vyřízení objednávky), při pohledu zblízka je možné najít deset různých elementárních kroků, a to jsem pro zjednodušení vypustil některé opravdu nepodstatné záležitosti, jakými je například zaplacení objednaného zboží.



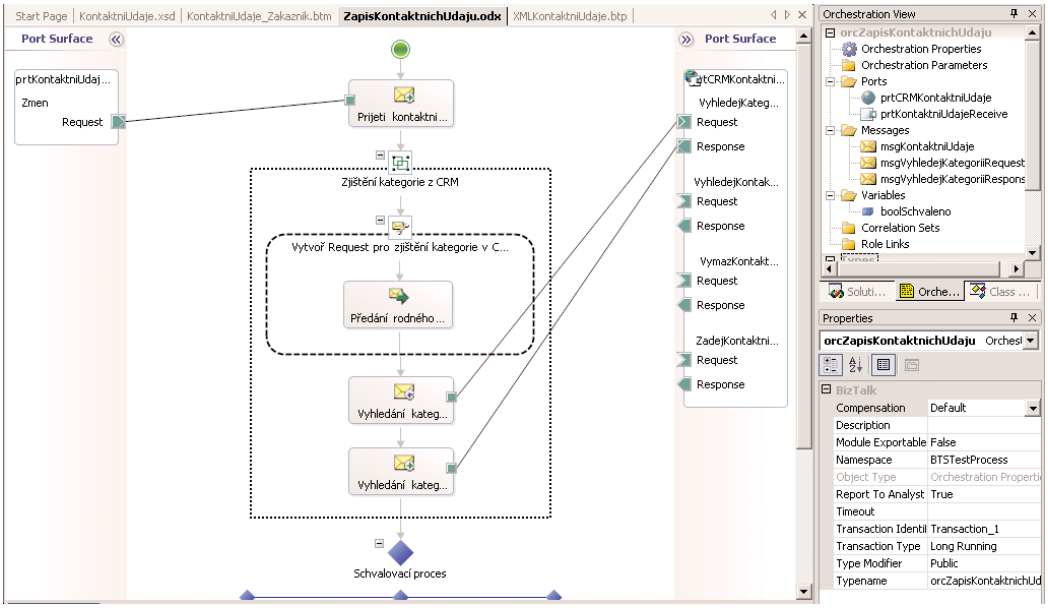
Obrázek 5.1: Posloupnost akcí při zpracování objednávky

Vrstva správy obchodních procesů se stará o řízení konverzace jednotlivých služeb neboli o sladění jednotlivých elementárních kroků do přehledných vyšších celků. Má na starosti větvení procesů podle obchodních pravidel, sledování paralelních toků procesu a jejich zpětnou synchronizaci, hlídání mezních časů pro běh procesu *(time-out)*, kontrolu odesílání a příjmu zpráv v rámci daného procesu, správu transakcí a další technické záležitosti. Samozřejmostí je robustnost procesů – jejich stav je udržován na trvalém médiu a v případě kritického výpadku systému je zaručeno pokračování procesu po obnovení všech důležitých životních funkcí systému.

K čemu je nám vlastně tato vrstva dobrá? Zjednodušeně řečeno, jejím největším přínosem je skutečnost, že není součástí aplikací či služeb, ale nachází se mimo ně. Soustředěním celého procesu do jednoho místa máme k dispozici centrální komunikaci, řízení, monitorování, ladění a ošetřování případných chyb – každá z těchto centralizací má pozitivní vliv na náklady provozování řešení. Jednotlivé integrované aplikace jsou na sobě nezávislé (neví jedna o druhé) a v případě nutnosti změny procesu není nutno měnit jednotlivé integrované aplikace či služby. Pojďme se nyní podívat na základní stavební bloky potřebné pro vytváření reálných procesů. Napadají vás nějaké?

5.1 Stavební bloky správy procesů

V BizTalk Serveru 2004 jsou procesy realizovány pomocí tzv. orchestrací *(orchestration)*. Orchestrace je definicí určitého procesu, např. zpracování objednávky. Orchestrace je zapsána v XML souboru v jazyce XLANG/s. Jde o proprietární jazyk pro interní potřebu BizTalk Serveru 2004. K vytváření orchestrací slouží *orchestration designer* ve Visual Studiu.NET (viz obrázek 5.2).

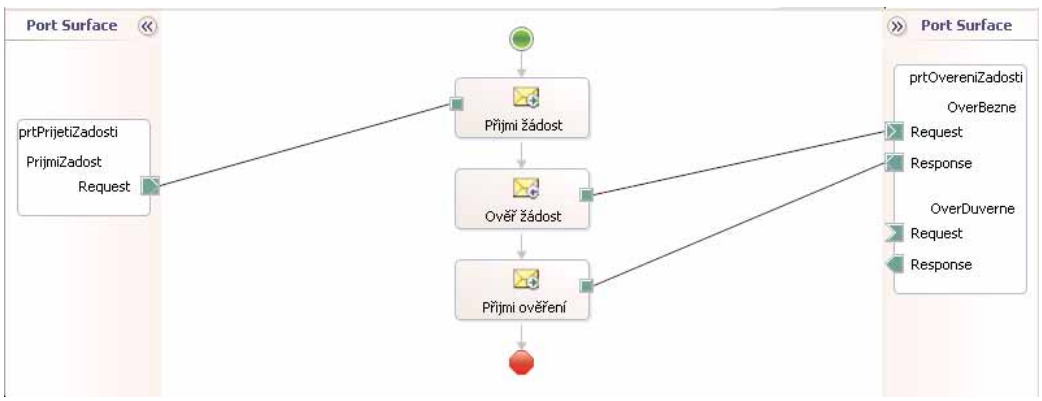


Obrázek 5.2: Orchestration designer ve Visual Studiu.NET

Vytvořená orchestrace je po dokončení zkompilována *(build)* do jediné **.dll* knihovny vytvořené v jazyce IL *(MSIL intermediate language),* což je standardizovaný jazyk používaný pro vytváření aplikací v .NET frameworku. Tato knihovna je poté nasazena *(deploy)* na serveru s nainstalovaným BizTalk Serverem 2004, což obnáší fyzické nakopírování do místa zvaného GAC *(global assembly cache)* a dále analýzu obsažených typů a nakopírování jejich metadat do konfigurační databáze BizTalk Serveru 2004. Poté je možné proces spouštět. Jednotlivé případy spuštěných procesů se nazývají instance *(orchestration instance),* počet spuštěných instancí (např. zpracování konkrétních objednávek) přitom není jakkoliv omezen. Podívejme se nyní hlouběji jaké funkce můžeme od vrstvy obchodních procesů očekávat.

Komunikace s okolím

Komunikace s okolím je samozřejmě prvním předpokladem fungování procesu. Každý proces vyžaduje pro svůj běh určité vstupy a generuje výstupy. Vstupem a výstupem jsou přitom zprávy *(message)* anebo sady zpráv *(multi-part message)*. Základní tvary pro přijetí a odeslání zprávy jsou *Receive* a *Send,* které jsou vždy navázány na určitý port – viz obrázek 5.3. V tomto fragmentu procesu vidíte dva porty *(prtPrijetiZadosti a prtOvereniZadosti)*. Každý port je charakterizován pořadím zpráv (*prtPrijetiZadosti* je jednosměrné přijetí zprávy, *prtOvereniZadosti* je typu *Request-Response*), podporovanými operacemi na portu (zajímavější port *prtOvereniZadosti* má dvě operace *OverBezne* a *OverDuverne*) a typy zpráv předávanými v jednotlivých krocích. Vlastní orchestrace je uprostřed obrázku a obsahuje akce *Send* a *Receive* navázané na příslušné porty. Součástí této vazby je též určení přenášené zprávy včetně jejího typu (zpravidla schématu). U tvaru pro přijetí je nutné uvést, zda příchozí zpráva vytváří novou instanci orchestrace (vlastnost *Activate*) anebo přijímá zprávy potřebné pro již běžící instance – v našem případě je přijetí žádosti zahájením nové orchestrace, zatímco přijetí ověření je pouze pokračováním již běžící orchestrace.



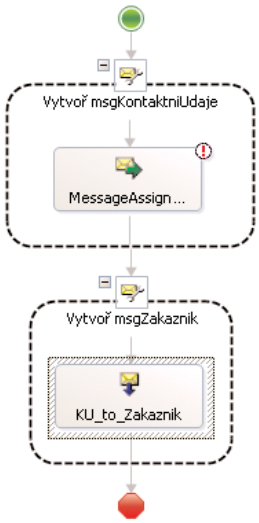
Obrázek 5.3: Přijetí a odeslání zprávy orchestrací

Důležitou součástí je též definování portů. Při jejich vytváření můžete buď přímo definovat příslušný port v podobě adaptéru a adresy (možnost *Specify now*) anebo toto rozhodnutí odložit na pozdější (možnost *Specify later*). Odložení na později je možné s výhodou využít při testování, kdy vlastní orchestraci doplníme XML souborem s definicí jednotlivých portů *(binding file)* a poté použijeme nástroj *BizTalk Deployment Wizard*. Můžeme tak pracovat se stále stejnou zkompilovanou *dll* knihovnou zkombinovanou s různými XML soubory pro testovací prostředí a pro provozní prostředí.

Porty jsou též součástí konceptu *publish/subscribe* popsaného v předchozí kapitole. Každý tvar *Receive* vytvoří interně subskripci pro přijetí všech zpráv z portu, na který byl navázán. Pokud tvar pro přijetí vytváří novou instanci orchestrace, můžeme na něm nastavit filtry pro přijetí podobně jako jsme si ukázali v kapitole 4.3 u portů pro odeslání (viz obrázek 4.14). Pokud tvar pro přijetí nevytváří novou instanci, není možné nastavit filtr – přijetí nebo nepřijetí zprávy se řídí tzv. korelací, o které si ještě povíme později. Tvar pro odeslání publikuje (odešle) svoji zprávu přímo na navázaný port pro odeslání *(send port),* případné filtry nastavené na tomto portu jsou ignorovány. Pokud chcete využít filtrování a mechanismus *publish/subscribe* popsaný v kapitole 4.3 pro odesílání zpráv z orchestrace, je možné nastavit port na typ *Direct,* čímž dojde k přímému předání (opublikování) zprávy do databáze *message box* – zpráva se chová stejně jako by byla přijata portem pro přijetí *(receive port)* popsaným v kapitole 4.3. Přijetí a odeslání zatím mnoho zajímavého nepřineslo, co když třeba chceme zprávu nějakým způsobem modifikovat?

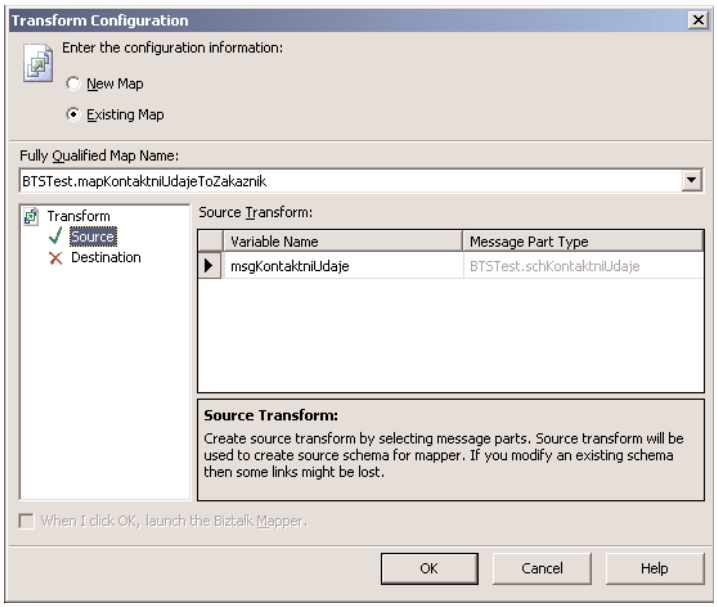
Manipulace se zprávami

Z důvodů přehlednosti procesu a optimalizace režie na ukládání stavu procesu je možnost manipulace se zprávami výrazně omezena. Každá zpráva je téměř neměnná, přesněji řečeno může být měněna pouze během velmi krátké doby omezené tvarem pro vytvoření zprávy *(Construct Message)* – na obrázku 5.4 vidíme dva takovéto tvary ohraničené přerušovanou čarou.



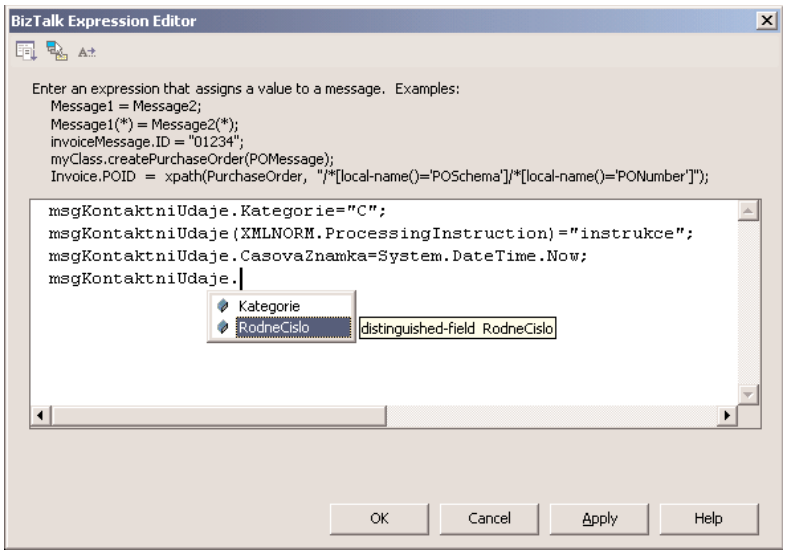
Obrázek 5.4: Vytvoření zpráv dvěma různými způsoby

Jakmile tok procesu opustí tvar pro vytvoření zprávy, dojde k uložení zprávy a zpráva je od této chvíle neměnná. Pokud byste přece jenom chtěli zprávu upravit, musíte použít nový tvar pro vytvoření zprávy, v něm vytvořit novou zprávu stejného typu zkopírováním zprávy původní a uvnitř tvaru pak manipulovat s touto nově vytvořenou zprávou. Zprávu je možné vytvořit trojím způsobem. Prvním je přijetí hotové zprávy, pro které nepotřebujeme žádné další pomůcky. Druhým je transformace existující zprávy pomocí mapy (viz kapitola 4.2). Použijeme zde tvar pro transformaci zpráv *(Transform),* ve kterém je nutné zadat použitou mapu, výchozí zprávu a vytvářenou zprávu (viz obrázek 5.5).



Obrázek 5.5: Vytvoření nové zprávy transformací existující zprávy

Třetím způsobem je vytvoření zprávy pomocí kódu tak jak jsou vývojáři běžně zvyklí. Tento kód smí obsahovat pouze příkazy přiřazení (viz obrázek 5.6). Kód se zadává v editoru výrazů, který nabízí plný komfort Visual Studia.NET. Pokud chcete při práci se zprávou plně využívat komfortu zpracování a zároveň zachytit řadu chyb již při vytváření, můžete důležité a často používané vlastnosti označit jako tzv. *distinguished field,* čímž je získáte k dispozici nejkomfortnějším možným způsobem (na obrázku 5.6 jsou takto odlišeny vlastnosti *Kategorie* a *RodneCislo*). Označení se provádí ve schématu zprávy analogickým způsobem jako vyznačení význačných vlastností probírané v kapitole 4.3. Pokud příslušný element zprávy neoznačíte, máte stále k dispozici méně komfortní, ale flexibilnější možnost používání *XPath* výrazů.



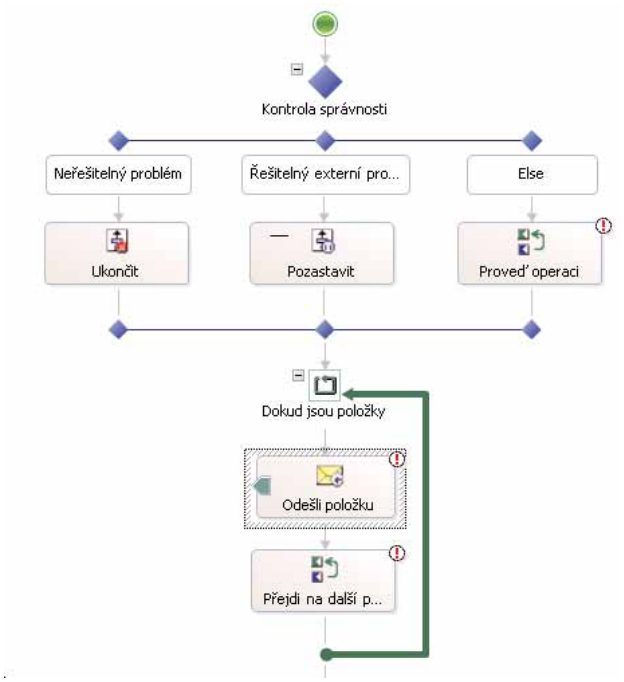
Obrázek 5.6: Vytvoření nové zprávy přiřazením

Řízení toku procesu

Proces, který by byl pouze sekvencí kroků v pevně daném pořadí, by si snad ani nezasloužil své označení. Prakticky žádný reálný proces není takto nudný, reálné procesy se podmíněně rozdělují, mají paralelní větve, které se jinde zase zpátky synchronizují apod. Pojďme se podívat na možnosti řízení toku procesu v BizTalk Serveru 2004.

Základní možností je podmíněné větvení toku, v klasických programovacích jazycích zastoupené konstrukcí typu *IF...ELSE*. Podobně se mohou chovat i procesy – viz obrázek 5.7. Můžete na něm vidět větvení toku pomocí akce *Decide,* který se skládá ze dvou nebo více možných větví. Každá větev s výjimkou poslední má přiřazenou podmínku

(zde jsou pojmenované *Neřešitelný problém* a *Řešitelný externí problém*), jejíž logická hodnota určuje, zda další provádění poběží „po její linii“. Provádí se vždy pouze jediná větev kódu, tudíž záleží na pořadí podmínek. Poslední větev *Else* se provádí pouze pokud nebyla žádná z podmínek splněna. V některých programovacích jazycích existuje ekvivalentní konstrukce *IF … ELSE IF … ELSE IF … ELSE*.

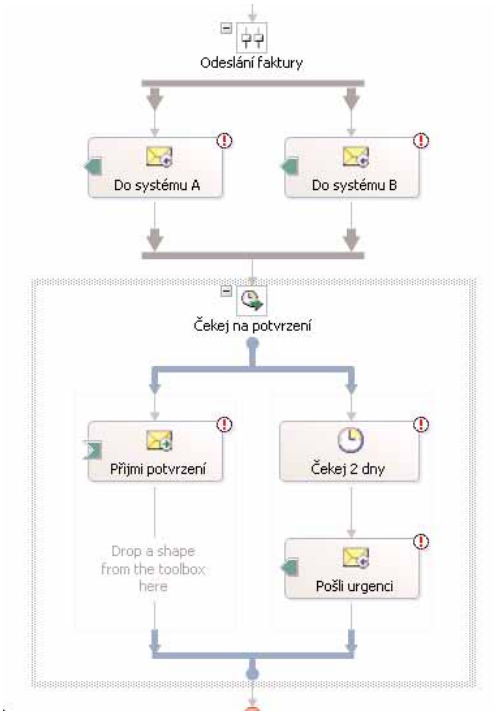


Obrázek 5.7: Různé možnosti řízení toku procesu

Zároveň jsou ve větvích použity další dva tvary pro řízení toku procesů – ukončení procesu *(Terminate)* a pozastavení procesu *(Suspend)*. Oba dva s okamžitou platností zastaví další provádění procesu. Jsou velmi jednoduché, jejich jedinou nastavitelnou vlastností je řetězec, který by měl vysvětlovat důvod provedené akce. Rozdíl mezi nimi je v možnosti pokračování procesu – pozastavený proces může být probuzen ze svého spánku zásahem administrátora, ukončený proces byl skončen jednou provždy. Jejich použití je myslím zřejmé – ukončení se používá v abnormální situaci, na kterou proces není schopen reagovat a která je považována za neřešitelnou, pozastavení dává lidstvu možnost situaci vyřešit a pokračovat v přerušeném procesu jakoby se téměř nic nestalo.

Na obrázku 5.7 vidíte i další tvar pro konstrukci smyčky *(Loop),* která funguje podobně jako učebnicová smyčka s podmínkou na začátku *(WHILE)* v programovacích jazycích. Na začátku má podmínku pro provádění. Pokud je tato splněna je provedeno tělo smyčky a běh se vrátí zpět k vyhodnocování platnosti podmínky. Není-li podmínka splněna, pokračuje se dále bezprostředně za smyčkou.

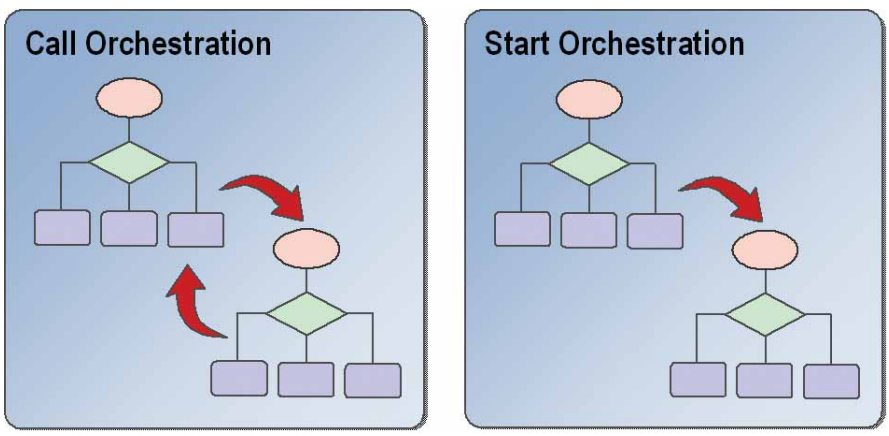
Další možností je rozdělení toku procesu do více větví a jejich opětovné pozdější spojení (viz obrázek 5.8). Existují dvě varianty lišící se právě způsobem synchronizace na konci větvení. První se nazývá *Parallel Actions* (na obrázku jde o odeslání faktury). Způsobí rozdělení do dvou nebo více paralelních větví, které se provádějí vždy všechny, přičemž dříve ukončené větve se na konci synchronizují se svými pomalejšími kolegyněmi. Dále se pokračuje pouze tehdy, když jsou všechny prováděné větve ukončeny. Malý příklad pro přiblížení: moji rodiče chodívali v dobách vlády jedné strany v sobotu brzy ráno nakupovat. Jeden si stoupnul do dlouhé fronty na maso, druhý do fronty na uzeniny. Domů šli společně až tehdy, když každý z nich svoji frontu vystál (úmyslně neříkám nakoupil). Paralelní akce fungují velmi podobně – hodí se v situacích, kdy je třeba provést více na sobě nezávislých akcí, přičemž jejich postupné provádění by bylo zbytečnou ztrátou času. Mimochodem, podobná programátorská úloha je v běžných programovacích jazycích velmi obtížná (pro většinu vývojářů spíše neřešitelná), neboť pro ni v procedurálních jazycích není žádná přirozená konstrukce.



Obrázek 5.8: Rozdělení toku procesů do více větví

Druhou variantou je akce nazývaná *Listen* (na obrázku 5.8 čekání na potvrzení). V tomto tvaru se začnou provádět všechny větve současně, ale dokončí se pouze jediná z nich – ta která nejdříve zvládne udělat první krok. První kroky mohou být pouze tvary vyžadující nějaký vnější podnět – buď přijetí zprávy nebo vypršení časového limitu. Opět malý příklad pro ilustraci. Je noc a jste na tramvajové zastávce. Jste společensky unavení a chcete se dostat co nejdříve domů. Ve vaší hlavě běží zároveň tři pochody – čekání na tramvaj, čekání na taxík a čekání na 8:00 ráno. Pokud jede kolem první noční tramvaj, jedete domů tramvají, je-li to taxík, jedete taxíkem. Pokud do 8:00 nepřijede nic, jdete do práce. Dalo by se říct, že akce *Listen* vždy čeká na nějakou událost a podle toho, jaká událost nastane první, se rozhodne pro další pokračování.

Je možné, že se některý z procesů stane nadměrně složitým a ukáže se žádoucí rozdělit jej na více menších podprocesů. Jindy může být žádoucí definovat pomocný proces využívaný několika hlavními procesy. V těchto situacích může být nápomocné vnořování procesů *(nesting),* vyskytující se ve dvou rozdílných mutacích (viz obrázek 5.9). Vždy jde o rozdělení procesu do více orchestrací, rozdíl je ve vzájemné synchronizaci orchestrací. Volání orchestrace *(Call Orchestration)* je synchronním voláním podprocesu, kdy volající proces čeká na dokončení volaného podprocesu. Spuštění orchestrace *(Start Orchestration)* je asynchronní – volající proces se nezajímá o dokončení a výsledek volaného procesu. Představme si, že procesem by bylo „Zpracování žádosti o úvěr“. Příslušný proces by mohl volat podproces „Kontrola důvěryhodnosti klienta“ (zde je závislý na jeho výsledku) a později startovat proces „Aktualizace úvěrových statistik“, přičemž našemu primárnímu procesu stačí vědět, že aktualizace byla spuštěna, její výsledek je mu však lhostejný

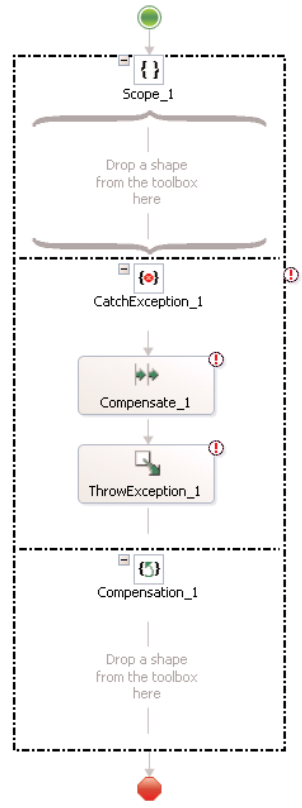


Obrázek 5.9: Rozdělení procesu do více orchestrací

Podpora transakcí a ošetření chyb

Transakce slouží k zachování konzistence dat, zejména pokud dojde k chybě během jejich zpracování. Klasický učebnicový příklad je převod peněz z účtu na účet, při kterém je nejprve třeba odečíst částku z prvního účtu a poté ji převést na druhý účet. Pokud by došlo k selhání po provedení první akce, zůstala by data v nekonzistentním stavu. Použitím transakcí se tomuto scénáři zabraňuje, operace buď proběhne jako celek nebo vůbec. Vlastnosti transakce se v učebnicích označují kyselou zkratkou ACID: *atomicity* – nedělitelné, buď proběhne všechno nebo nic, *consistency* – systém se nachází pouze v konzistentních stavech, *isolation* – transakce není ovlivněna ostatními probíhajícími transakcemi, *durability* – po provedení transakce jsou provedené změny trvalé (což je poměrně relativní termín zpravidla označující zapsání na disk, výbuch nukleární nálože v serverovně transakce zpravidla nepřetrvá). Kéž by byl svět tak jednoduchý jako v učebnicích!

Základním tvarem pro práci s transakcemi je akce pro rozsah *(Scope)* znázorněný na obrázku 5.10. Rozsah sám o sobě vymezuje jakýsi blok či modul uvnitř procesu. V rámci tohoto rozsahu je definována platnost a přístupnost zpráv a proměnných, rozsah je zároveň blokem pro strukturované ošetřování výjimek a slouží jako kontejner pro definici transakcí. Orchestrace jako celek tvoří základní (můžeme říct kořenový) rozsah, ve kterém může být obsažen libovolný počet dalších rozsahů zanořených do libovolné hloubky. Každý rozsah se skládá ze své základní části (těla), libovolného počtu bloků pro ošetření výjimky a nejvýše jednoho bloku pro kompenzaci transakcí.



Obrázek 5.10: Použití tvaru pro rozsah

Podívejme se nejprve na ošetření chyb. Moderní programovací jazyky umožňují tzv. strukturované ošetření výjimek (*structured exception handling,* SEH), zpravidla pomocí konstrukce *TRY ... CATCH*. V bloku *TRY* se provádí výkonný kód, který by měl samozřejmě v ideálním případě běžet bez chyb. Pokud přesto k nějaké chybě (výjimce) dojde, prochází se jednotlivé bloky pro obsluhu výjimek a hledá se blok, který ošetřuje vzniklý typ chyby. Pokud je takový blok nalezen, je proveden kód v něm obsažený. Pokud nalezen není, prochází se řetězově bloky pro obsluhu výjimek ve volajícím kódu a hledá se příslušný blok pro obsluhu výjimky. Pokud není žádný takový blok nalezen, prostředí pro běh programu výjimku zachytí, zpravidla ji oznámí uživateli a ukončí program. Výjimky v orchestraci BizTalk Serveru fungují velmi podobně. Mohou vznikat chybou v kódu, vypršením času transakce, chybou komunikace adaptéru, chybou volaného kódu, ukončením procesu v jiné větvi procesu, jakkoliv nepravděpodobnou chybou v BizTalk Serveru samotném apod. Anebo mohou být vyvolány úmyslně pomocí tvaru vyvolání výjimky (*Throw Exception,* viz též obrázek 5.10), který je vhodné použít, pokud detekujeme nežádoucí situaci, kterou na dané úrovni provádění nejsme schopni napravit. Vyvoláme tedy chybu a doufáme, že ji za nás někdo vyřeší na vyšší úrovni nebo bude alespoň

zaznamenána. Pokud tedy k chybě v orchestraci došlo, hledá se v daném rozsahu blok, který tuto chybu ošetřuje. Není-li takovýto blok nalezen, hledá se v nadřazených rozsazích případně ve volající orchestraci, byla-li použita akce *Call Orchestration*. Pokud není výjimka ošetřena ani tímto způsobem, je odchycena BizTalk Serverem, který příslušnou instanci orchestrace ukončí a zapíše chybové hlášení do protokolu událostí.

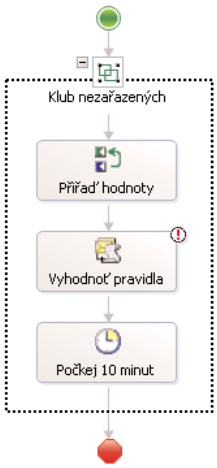
Zpátky k transakcím. Začněme atomárními čili ACID transakcemi z našeho učebnicového příkladu. Práce s nimi je poměrně jednoduchá. Stačí vytvořit rozsah, nastavit jeho typ transakce na *Atomic* a je to. Alternativně můžete stejnou vlastnost nastavit pro celou orchestraci. Jakékoliv operace prováděné uvnitř rozsahu jsou pak prováděny v rámci jedné transakce. Veškerá práce se zprávami, proměnnými, objekty, volání externích funkcí, přijetí nebo odeslání zprávy – vše tvoří jediný nedělitelný celek. Během provádění rozsahu jsou drženy zámky na příslušné objekty (zejména databáze), čímž se potenciálně snižuje propustnost systému. Proto by atomární transakce měly být co nejkratší, nejvýše několik sekund. Pokud během transakce dojde k jakékoliv chybě (selhání kódu, kolaps softwaru, hardwarová chyba apod.), transakční monitor se postará o navrácení všech zainteresovaných zdrojů do původního stavu. Stejný efekt nastane, pokud vyprší čas transakce *(timeout),* který je standardně nastavený na 1 minutu, což berme jako další důvod, proč by měla ACID transakce trvat co nejkratší dobu. Na závěr je nutné přiznat jednu malou nepříjemnost – pokud v transakci používáte prostředky, které nepodporují transakce (např. volání externí .NET komponenty, která nemá podporu pro *Enterprise Services,* nebo použití adaptérů nepodporujících transakce, jakými jsou SOAP, HTTP, FTP nebo FILE), kouzlo transakce bez povšimnutí pomine!

Druhým a zajímavějším typem jsou transakce dlouhotrvající. Vraťme se zpátky k učebnicovému příkladu. Představme si, že peníze převádíte z účtu v bance A na účet v bance B. Je zahájena transakce a peníze jsou odepsány z účtu v bance A. Poté je přes clearingové centrum odeslán pokyn bance B, což může zabrat nějakých 24 hodin, během celé této doby je v bance A otevřena transakce a jsou drženy zámky v databázi … je čas pohádku ukončit, takto by to asi nefungovalo. Místo toho se používá koncept dlouhotrvající transakce. Není to zcela přesné označení, neboť v pravém slova smyslu nejde o transakci, například systém se během ní nachází v nekonzistentních stavech. Ve skutečnosti celá „transakce“ funguje zhruba následovně (pokud pracujete v bankovnictví, omlouvám se za svoji naivní laickou představu). Peníze jsou opravdu odepsány z účtu v bance A a zároveň je odeslán pokyn do clearingového centra. V tuto chvíli je systém v nekonzistentním stavu, neboť peníze takříkajíc visí ve vzduchu. Pokud jsou všechny náležitosti v pořádku, z pohledu banky A vše končí. Pokud bylo např. uvedeno špatné číslo účtu, banka B o tom informuje odesílající banku A prostřednictvím clearingového centra. Banka A spustí proces nazývaný kompenzace *(compensation),* v tomto případě spočívající v připsání peněz zpět na účet klienta banky A. Striktně vzato se systém banky A nepřevede do původního stavu, na vašem účtu se objeví dvě nové položky, jedna plus a druhá minus. Ale systém jako celek se vrátil zpět do konzistentního stavu a to je důležité. Nešlo tedy o opravdovou „učebnicovou“ transakci, přesto se nastavený proces postaral o chování transakci velmi podobné.

Dlouhotrvající transakce jsou v BizTalk Serveru realizovány opět pomocí rozsahu *(Scope)* s typem transakce nastaveným na *Long Running*. V jeho těle se provádí vlastní operace, o níž optimisticky předpokládáme, že se zdaří. K tomuto rozsahu můžete přiřadit kompenzační blok *(Compensation Block)*. Rovněž můžete – ale nemusíte – přiřadit k rozsahu čas vypršení *(Timeout)*. V žádném případě zde nemusíte být tak přísní jako u ACID transakcí, dlouhotrvající transakce nedrží zámky v databázi ani jiné prostředky, proto mohou klidně trvat dny, hodiny i týdny. Dokonce ji v čase nemusíte omezovat vůbec, i když mne momentálně nenapadá jediný důvod, proč byste to dělali. Kdy se kompenzační blok provádí? Pokud dojde k jakékoliv neošetřené chybě v rámci daného rozsahu, pokud uplyne nastavený čas vypršení transakce anebo pokud si kompenzaci vyžádáte pomocí tvaru pro kompenzaci (*Compensate,* viz též obrázek 5.10).

Klub nezařazených akcí

Ještě nám zbývá pár akcí orchestrace, které se nehodí do žádné z předchozích kategorií. Neznamená to, že by byly pouze na okraji zájmu – naopak, patří mezi velmi důležité. Proto jsem je sdružil v klubu nezařazených akcí, jsa inspirován parlamentním klubem nezařazených poslanců. Nemějte obavu, ničím dalším z parlamentu se inspirovat nehodlám. S výjimkou akce *Role Link,* která přesahuje rámec této brožury a byla násilně odsunuta do přílohy A, vidíte všechny čtyři na obrázku 5.11. Ne, nepřepočítal jsem se. Prvním prvkem je akce pro skupinu *(Group),* která nemá žádnou výkonnou funkci. Slouží pro pohodlí návrháře procesu, který může jeho jednotlivé části sdružovat do skupin. Skupinu pak lze vizuálně sbalit nebo rozbalovat, což může být dobrou pomůckou zejména při práci s velkými a rozsáhlými orchestracemi.



Obrázek 5.11: Klub nezařazených akcí

Jednou z nejdůležitějších akcí je výraz *(Expression),* do které můžete zadat libovolné výrazy v jazyce C#. Můžete například volat externí funkci obsaženou v nainstalovaných knihovnách nad .NET Frameworkem nebo manipulovat s hodnotami proměnných definovaných v orchestraci. Nedoporučuje se používat tento tvar k složitějším operacím (např. rozhodovací logice), neboť by se snížila čitelnost a přehlednost vytvářené orchestrace. Výrazy by měly obsahovat pouze jednoduchá volání a přiřazení, vše ostatní by mělo být vyjádřeno jinými procesními tvary, případně zapouzdřeno v nějaké externí funkci nebo externích pravidlech.

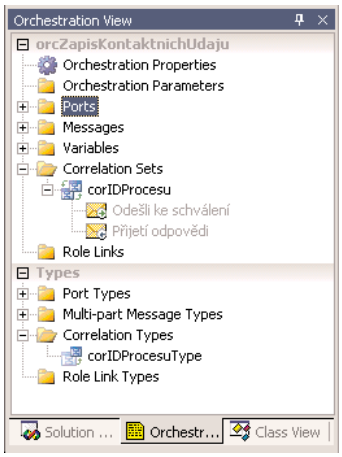
Tím jsme se elegantně dostali k akce pro volání pravidel *(Call Rules),* která umožňuje volat pravidla v separátní komponentě BizTalk Serveru – stroji pro provádění obchodních pravidel (*Business Rules Engine,* BRE). BRE umožňuje oddělit pravidla od procesů a tím zvýšit flexibilitu celého řešení, neboť procesy se zdaleka nemění tak často, jako pravidla pro jejich řízení. Více se tomuto tématu budeme věnovat v kapitole 5.2.

Akce čekání *(Delay)* je velmi jednoduchá. Jejím jediným parametrem je doba čekání, vyjádřená buď časovým intervalem (používá se třída .NET Frameworku *System.TimeSpan*) nebo datem a časem, kdy má být čekání ukončeno. Čekání se nejčastěji používá uvnitř jedné z větví výše probrané akce *Listen,* kdy se čeká na doručení zprávy. Pokud není zpráva doručena do vypršení vymezeného času čekání, je možné na tento fakt procesně reagovat. A nyní něco úplně jiného.

Korelace

Korelace je odborný termín pro spárování příchozí zprávy s běžící instancí orchestrace. V každý okamžik totiž může běžet neomezený počet instancí dané orchestrace, které provádějí stejné akce, ale nad různými daty. Příchozí zprávu je proto nutné přiřadit správné instanci orchestrace. Pro příklad z každodenního života nemusíme chodit daleko. Pokud podáte žádost na soud nebo podobný úřad, je vaší žádosti přiřazeno číslo jednací. Toto číslo je poté používáno na veškeré korespondenci od úřadu směrem k vám, přičemž se od vás očekává, že při korespondenci vůči úřadu budete toto číslo uvádět. Pracovnice v podatelně váš dopis potom podle čísla jednacího přesměruje příslušné výkonné osobě k dalšímu postupu. Pokud číslo jednací neuvedete, bude váš dopis pravděpodobně přesměrován též, ale pravděpodobně za delší dobu a se skřípěním zubů. Korelace v integračním brokeru funguje velmi podobně – pouze skřípění zubů se od serveru nedočkáte, zprávu, kterou není možné jednoznačně korelovat nemilosrdně vyřadí.

V integračním brokeru je nutné korelovat veškeré zprávy přijaté orchestrací se dvěma výjimkami. První jsou zprávy spouštějící danou orchestraci (tato zpráva má vlastnost *Activate* nastavenou na *True*), neboť žádná orchestrace v okamžiku přijetí ještě neexistuje. Druhou jsou zprávy, která jsou přijaty při obousměrné komunikaci na portu pro odeslání *(Solicit-Response),* u nichž se o správné přiřazení k orchestraci postará příslušný adaptér. Základní konstrukcí pro vytvoření korelace je tzv. korelační typ (*Correlation Type* – viz obrázek 5.12). Není ničím jiným než skupinou jedné nebo více význačných vlastností *(promoted property)* zpráv – pokud si chcete osvěžit paměť, vraťte se na chvíli ke kapitole 4.3. Korelačním typem může být nějaký přirozený identifikátor procesu jako třeba výše zmíněné jednací číslo nebo může jít o uměle vytvořený jednoznačný identifikátor (např. *GUID*). Korelační typ by měl opravdu spolehlivě definovat instanci procesu, např. rodné číslo vypadá na první pohled lákavě, ale co když jedna osoba podá více dokumentů nebo se vyskytne duplicita rodných čísel? Pokud si to můžete dovolit, použijte pro jistotu více identifikátorů v korelačním typu (např. rodné číslo a GUID), ale mějte na paměti, že integrované systémy musí tyto identifikátory zachovat a posléze nezměněné předat zpět procesu.



Obrázek 5.12: Definice korelačního typu a množiny

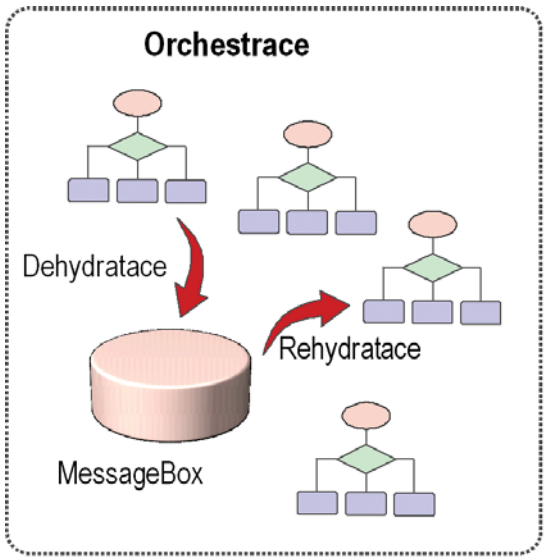
V definici orchestrace pak vytvoříme korelační množinu *(Correlation Set)* určitého korelačního typu. V jedné z akcí pro odeslání (méně častěji pro přijetí) nastavíme korelační množinu (vlastnost *Initializing Correlation Sets*), libovolné množství akcí pro přijetí pak necháme se touto korelační množinou řídit (vlastnost *Follows Correlation Sets*). Pokud máme správně nastavené význačné vlastnosti *(Promoted Properties)* u všech zainteresovaných zpráv, o zbytek se bez našeho přičinění téměř kouzelně postará integrační broker. Vizuální diagram používání korelačních množin při odeslání a přijetí zprávy je možné vidět i v designéru orchestrací (viz obrázek 5.12).

Trvalé uchování stavu procesu

Implementované procesy mohou trvat velmi dlouho, proto se nemůžeme spoléhat na nepřetržitou a bezchybnou celého prostředí, což však sotva může být omluvou pro nedokončení procesu nebo jeho nekorektní pokračování. Proto musí integrační broker velmi bedlivě zacházet se stavem procesu, jeho zprávami a proměnnými a v pravidelných intervalech vše ukládat tak, aby se proces jako celek bezchybně a spolehlivě přesouval mezi konzistentními stavy a po případném výpadku systému plynule pokračoval v provádění od posledního konzistentního stavu.

Kdy se onen konzistentní stav ukládá? Dokumentace vyjmenovává následující možnosti: při ukončení rozsahu s transakčním zpracováním (jde-li o rozsah s atomární transakcí, ukládá se stav v rámci oné transakce), při dosažení bodu přerušení (*breakpoint,* viz dále), při odeslání zprávy mimo atomární transakci, při spuštění jiné orchestrace, při pozastavení provádění orchestrace pomocí akce *Suspend,* při kontrolovaném zastavení procesu serveru, při ukončení daného procesu, při dehydrataci orchestrace (viz dále). Z ukládání proměnných jako součásti stavu procesu vyplývá jedno důležité omezení – musí jít o serializovatelné třídy .NET Frameworku, jinak by BizTalk nevěděl jakým způsobem uložit jejich stav. Omezení se naštěstí nevztahuje na rozsahy s atomárními transakcemi, během kterých se stav nikdy neukládá, takže výše uvedená restrikce není třeba.

Některé bakterie mají schopnost vytvářet tzv. spory. Pokud jsou vystaveny nepříznivým podmínkám, zahustí své buněčné kapaliny a zapouzdří se do podoby spory. Pokud se později dostanou do příznivých podmínek s dostatkem vody, přijme ji zpět a začne opět normálně fungovat. Velmi podobně fungují i instance orchestrací (viz obrázek 5.13), i když silně pochybuji, že architekti produktu hledali inspiraci v říši bakterií. Důvodem není snaha o přežití jako v případě bakterií, ale snaha o úsporu paměti a dalších prostředků serveru. Pokud by totiž všechny běžící instance byly neustále přítomny v paměti, musel by být výrazně omezen počet současně běžících, často dlouhotrvajících orchestrací. Pokud orchestrace čeká na příchozí zprávu, na vypršení akce čekání *(Delay)* nebo na opakování selhané atomární transakce, doba čekání přesáhne nastavený limit a orchestrace se nenachází uprostřed atomární transakce, integrační broker se rozhodne pro dehydrataci orchestrace. Dehydratace spočívá v uložení stavu procesu do databáze *message box* a její dočasné odstranění z paměti. Živou vodou pro dehydrovanou orchestraci je přijetí zprávy určené pro orchestraci nebo vypršení časového intervalu. Dojde k tzv. rehydrataci spočívající v „natažení“ uložené orchestrace do paměti, obnovení jejího stavu a pokračování od místa, kde bylo provádění přerušeno. Výše popsaný mechanismus umožňuje současné provádění v podstatě neomezeného počtu dlouhotrvajících transakcí.

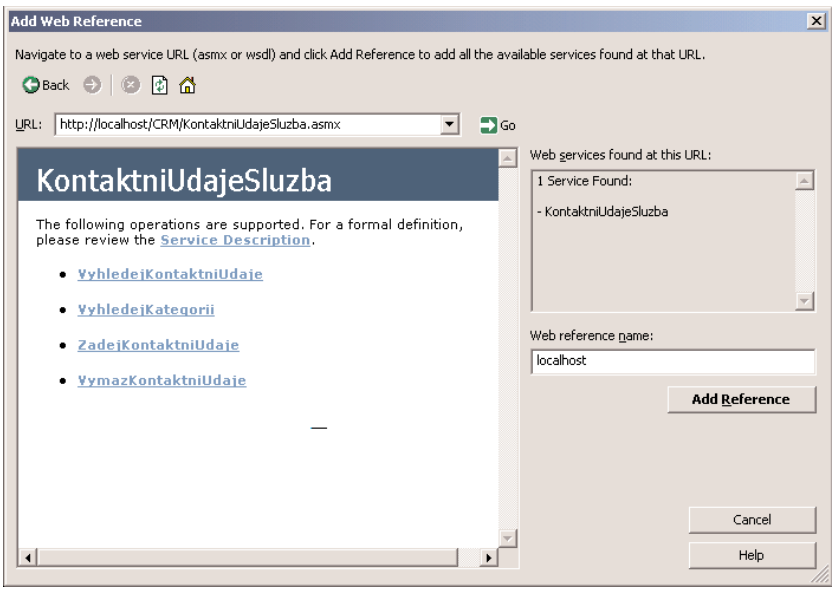


Obrázek 5.13: Dehydratace a rehydratace orchestrací

Podpora webových služeb

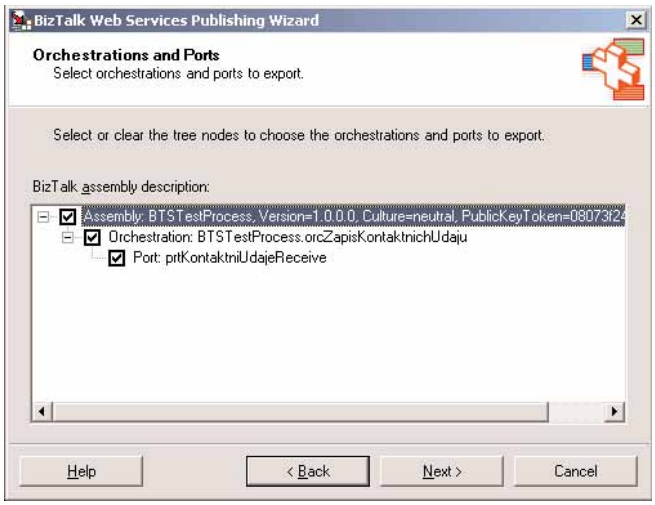
Webové služby (kdo se ztratil, skočte zpět na odstavec 2.3.) mají pro integraci v heterogenním prostředí klíčový význam a pochopitelně se nemohly vyhnout ani procesům. Celou problematiku můžeme rozdělit podle starého přísloví – „buď přijde Mohamed k hoře anebo hora k Mohamedovi“, neboli buď volá proces externí webovou službu anebo externí aplikace či služba volá proces prostřednictvím protokolů webových služeb.

Začněme Mohamedem jdoucím k hoře neboli voláním externí webové služby z naší orchestrace. Vzhledem k tomu, že adaptér pro SOAP je součástí BizTalk Serveru 2004, nic nám nebrání použít standardní postup vytváření zpráv, portů, tvarů pro odeslání a přijetí atd. Vývojáři jsou ale zhýčkané slečinky a ve Visual Studiu.NET jsou zvyklí na časté a bezpracné využívání možnosti volat webové služby prostřednictvím automaticky generované proxy třídy (možnost *Add Web Reference*) a výše popsaný dlouhý postup by jim připadal příliš složitý. Očekávané nářky byly vyslyšeny a *orchestration designer* používá stejný dialog pro komunikaci s webovou službou (viz obrázek 5.14). Stačí pouze zadat URL adresu a jmenný prostor pro generované třídy. Zbytek probíhá automaticky – konkrétně vygenerování portu orchestrace včetně příslušných operací, dále portu pro odeslání včetně SOAP adaptéru s nastavenou adresou a vytvoření potřebných typů sad zpráv *(multi-part message types)* pro jednotlivé operace. Pokud se vrátíte k definici WSDL dokumentu z kapitoly 2.3, zjistíte nápadnou podobnost v termínech WSDL dokumentů a BizTalk Serveru 2004 – podobnost, která jistě nebude čistě náhodná.



Obrázek 5.14: Přidání odkazu na webovou službu

Hora k Mohamedovi neboli zpřístupnění procesu jako webové služby volatelné zvenčí je jenom o málo složitější. Webové služby na platformě .NET jsou implementovány v prostředí ASP.NET. BizTalk Server 2004 obsahuje takzvaný *BizTalk Web Services Publishing Wizard* (viz obrázek 5.15), ve kterém specifikujeme, které orchestrace a které veřejně přístupné porty mají být zveřejněny. Průvodce vygeneruje nový virtuální web včetně projektu pro Visual Studio.NET a veškerý potřebný kód pro fungování webové služby včetně rozhraní věrně kopírujícího operace a zprávy zpřístupněných portů. Pozor, průvodce za nás neudělá jedinou věc – vytvoření portu pro přijetí *(receive port* a *receive location),* který musíte vytvořit sami a správně nastavit relativní cestu k **.asmx* souboru jako jeho adresu.



Obrázek 5.15: Opublikování portu orchestrace jako webové služby

Interoperabilita pomocí BPEL

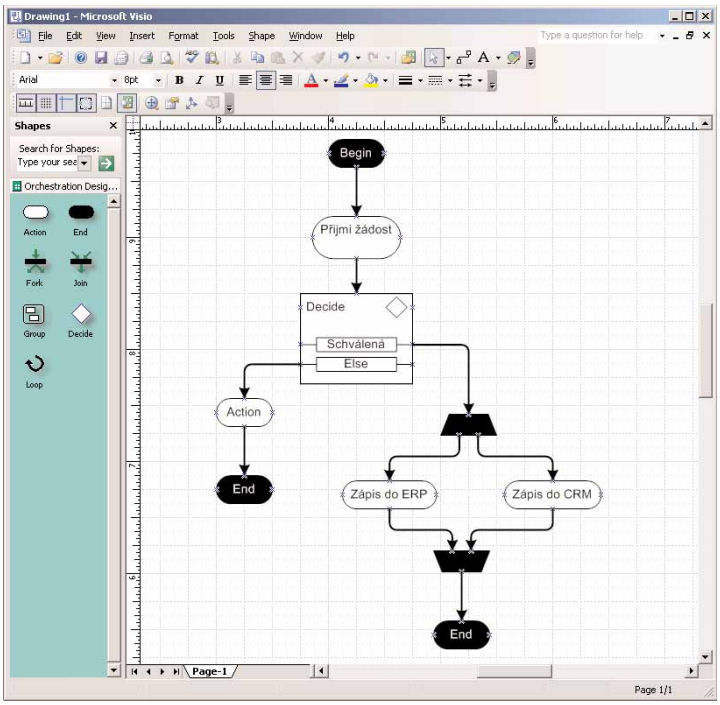
Znalost procesu je často daleko cennější než jeho konkrétní implementace, tudíž může být výhodné popsat obchodní proces standardním a nezávislým způsobem. Softwarové firmy v čele s BEA, IBM a Microsoftem proto vytvořili specifikaci BPEL (*Business Process Execution Language for Web Services,* někdy též zkracováno jako BPEL4WS). Pomocí tohoto standardu je možné definovat obchodní procesy nezávisle na konkrétní implementaci, případně tyto definice procesů přenášet mezi jednotlivými produkty.

Specifikace BPEL je funkčně poměrně minimalistická a dodavatelé integračních nástrojů samozřejmě chtějí nabídnout lepší funkčnost než jejich konkurenti. Proto nástroje pro správu obchodních procesů typicky obsahují širší funkcionalitu než BPEL, např. BizTalk Server 2004 podporuje navíc mapování mezi dokumenty s různými schématy, volání metod lokálních objektů, širokou podporu transakcí a další. Z výše uvedeného důvodu nepoužívá BizTalk Server interně jazyk BPEL, ale vlastní jazyk XLANG/s, který je jeho funkční nadmnožinou.

Proto se při vytváření orchestrace musíme rozhodnout, zda vytvoříme orchestraci BPEL-kompatibilní s omezenou funkčností nebo orchestraci s plnou škálou nabízených vlastností. V každém případě, orchestraci je možné exportovat do BPEL dokumentu a dalších pomocných dokumentů (WSDL rozhraní a XML schémata). Naopak, existující BPEL artefakty můžete importovat a vytvořit tak orchestraci BizTalku. Shrnutí na závěr: přenos implementace procesu pomocí BPEL není reálný, přenos definice procesu pomocí BPEL je plně podporován.

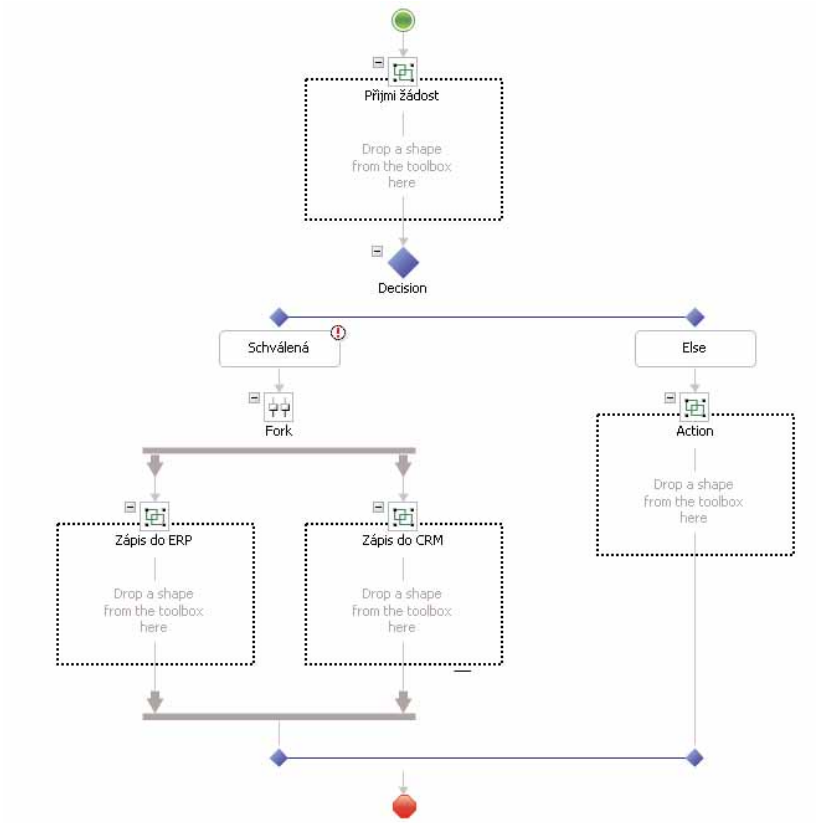
Návrhář orchestrací pro analytiky

Ke správě procesů v BizTalk Serveru existuje zajímavý doplněk v podobě Návrháře orchestrací pro analytiky *(Orchestration Designer for Business Analysts)*. Jedná se o doplněk k programu Microsoft Visio 2002 nebo 2003 (viz obrázek 5.16). Jde o nástroj pro konzultanty a analytiky procesů, kteří nemají ponětí o BizTalk Serveru, jež je pro ně již příliš technicky zaměřeným produktem. S pomocí doplňku pro Visio mohou relativně snadno dokumentovat a navrhovat procesy firmy či organizace. Navíc s pomocí nástroje, který buď již znají nebo pro ně bude vzhledem k podobnosti s ostatními programy z rodiny Microsoft Office snadný na naučení.



Obrázek 5.16: Návrhář orchestrací pro analytiky

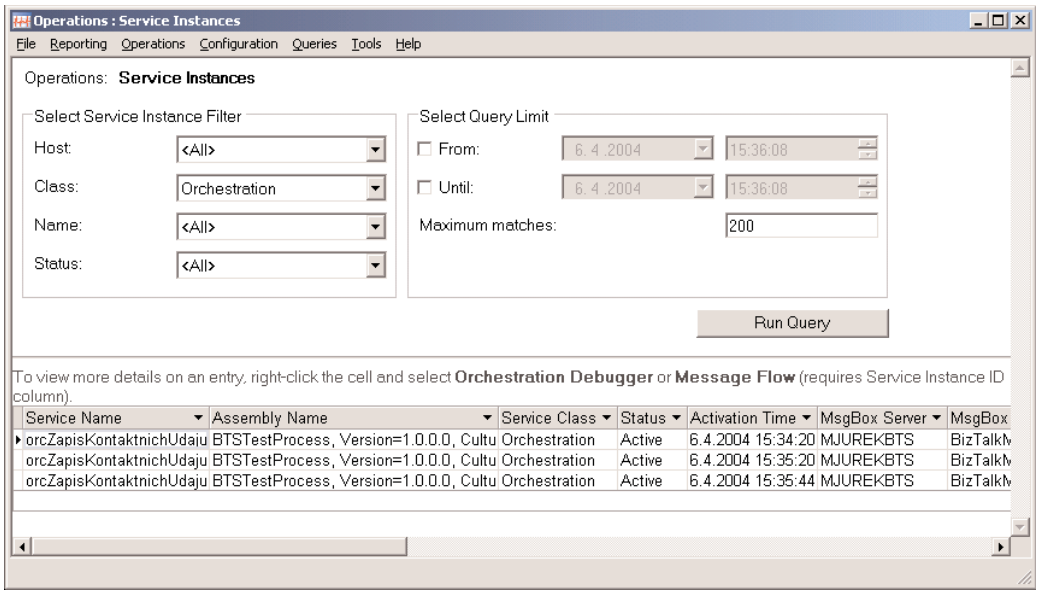
Exportem z programu Visio je možné vytvořit definiční soubor orchestrace BizTalku, který poté může vývojář otevřít ve Visual Studiu.NET (viz obrázek 5.17). Soubor je samozřejmě možné (dokonce nutné) při implementaci upravit. Teoreticky je možný též opačný proces – import definice orchestrace ve formátu **.odx* do Visia, například pro účely dokumentace projektu. Z vlastní zkušenosti však mohu říct, že výsledek není nejlepší a vypadá dosti technokraticky. Což by nám vlastně nemělo vadit, neboť každou implementaci by měla předcházet důkladná analýza, která je zároveň základem pro dokumentaci, takže není nutné ji dodatečně vyrábět, nemám pravdu?



Obrázek 5.17: Návrh procesu přenesený z Visio do Visual Studia.NET

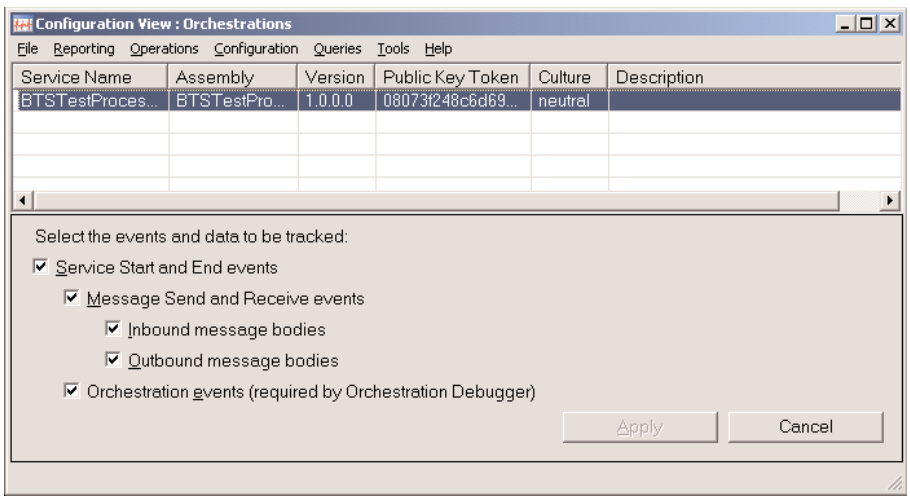
Monitorování a ladění

Je jenom obtížně představitelné, že by někdo navrhnul proces a poté jej slepě uvedl do provozu bez možnosti monitorování jeho běhu. V reálném nasazení je třeba mít v každém okamžiku přehled o běžících procesech a jejich typech, délce jejich běhu, aktuálním stavu apod. BizTalk Server 2004 samozřejmě takové monitorování nabízí. Veškeré události související s během procesu jsou zaznamenávány do databáze, ze které lze získat obrázek aktuální situace. Pomocí nástroje HAT *(Health and Activity Tracking)* můžete pokládat dotazy do monitorovací databáze, filtrovat a třídit výsledky (viz obrázek 5.18).



Obrázek 5.18: Zjištění běžících instancí orchestrací v nástroji HAT

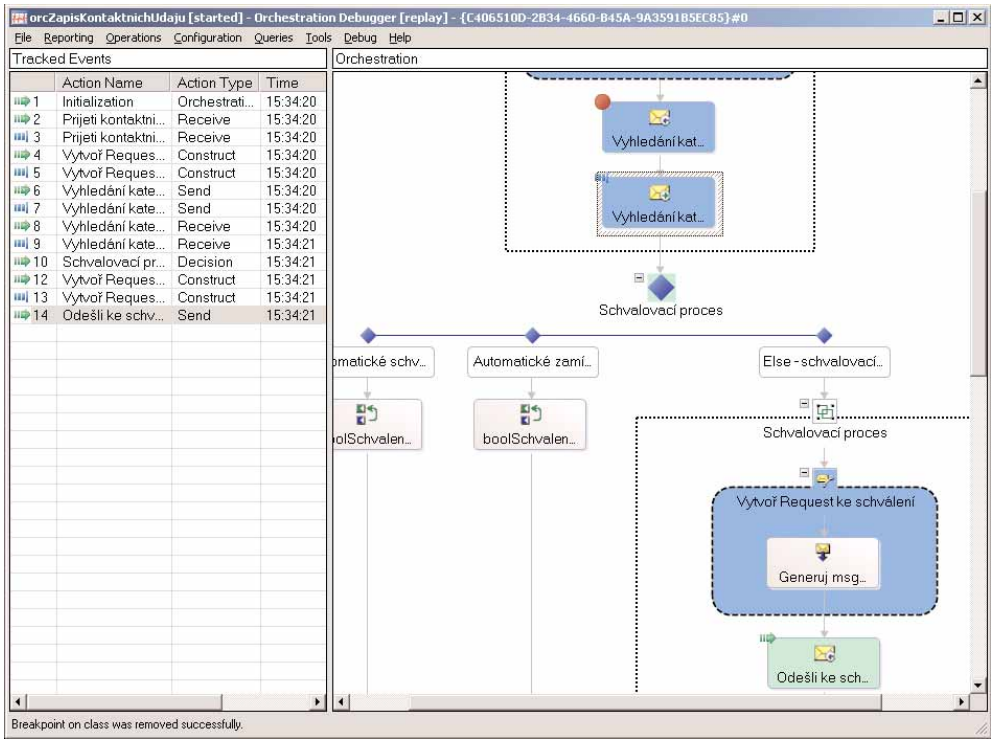
Úroveň monitorování je nejprve nutné nastavit (viz obrázek 5.19). Ve výchozím stavu jsou zaznamenávány pouze informace o provádění jednotlivých kroků. Chcete-li vidět více detailů, zejména obsah zpráv odesílaných nebo přijatých procesem, musíte jejich zaznamenávání povolit. Mějme na paměti, že nic není zadarmo. Za detailnější monitorování probíhajících procesů zaplatíte výrazným zvýšením režie spojené se zaznamenáváním událostí a bezesporu též nárůstem velikosti monitorovací databáze.



Obrázek 5.19: Nastavení úrovně monitorování v nástroji HAT

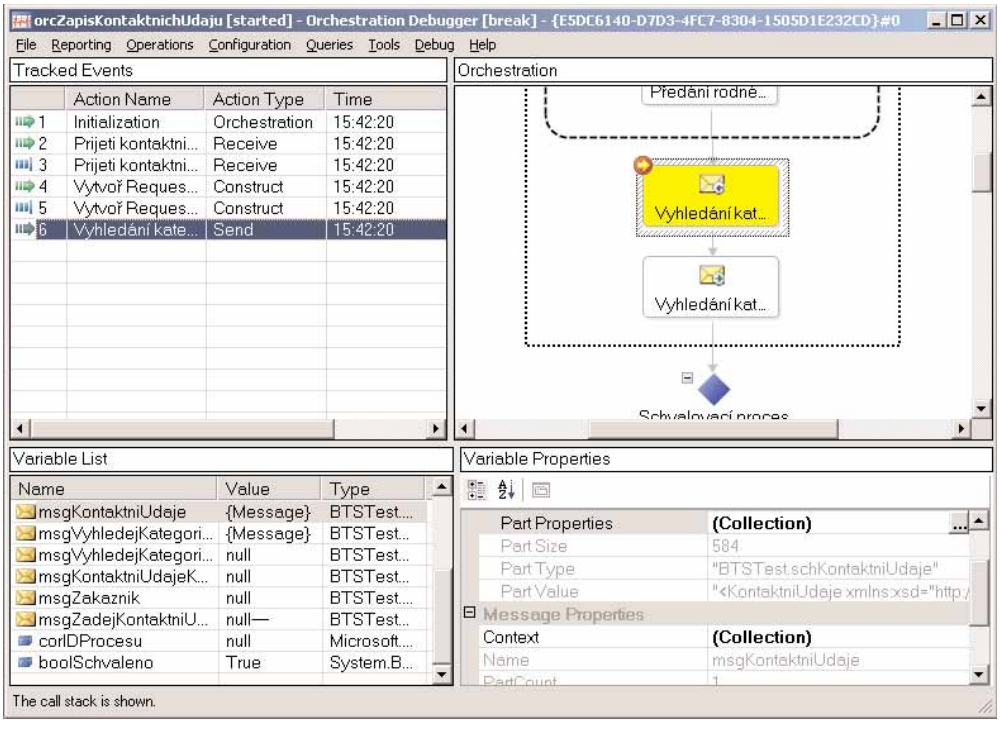
Pokud máte zapnuté zaznamenávání událostí, můžete v nástroji zvaném *Orchestration Debugger* sledovat průběh procesu (viz obrázek 5.20). V levé polovině vidíme zaznamenané časy zahájení a ukončení jednotlivých dosud provedených akcí orchestrace, na levé straně pak grafické vyjádření toku procesu pomocí podbarvení příslušných tvarů. Můžeme zde také nastavit body přerušení pro proces *(breakpoint),* a to buď pro právě otevřenou instanci

orchestrace nebo pro všechny budoucí instance dané orchestrace. Bod přerušení je na obrázku vyznačen podle zažité konvence červeným kolečkem.



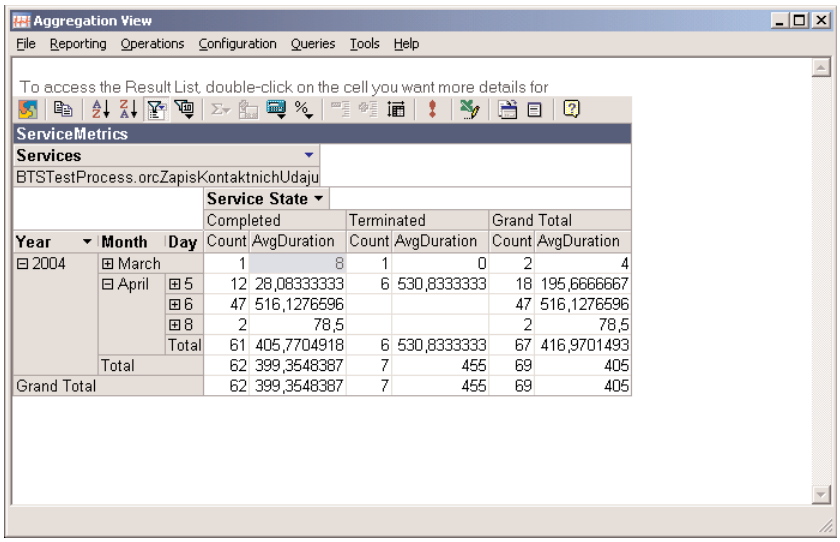
Obrázek 5.20: Sledování průběhu běžící orchestrace

K procesu zastavenému v bodě přerušení pak jednoduše připojíme debugger (viz obrázek 5.21) a můžeme proces ladit velmi komfortním způsobem. Vidíme všechny dosud provedené kroky a jejich časy, přičemž aktuálně prováděný krok je zvýrazněn. Dále máme k dispozici výpis aktuálního stavu zpráv a proměnných. U zpráv si můžeme vypsat jejich obsah, soubor jejích vlastností apod.



Obrázek 5.21: Ladění pozastavené instance orchestrace

Pomocí stejného nástroje je možné sledovat i historické trendy spuštěných procesů v souhrnném multidimenzionálním pohledu, který pomocí technologie OLAP nabízí nepřeberné množství pohledů na metriky fungování systému (viz obrázek 5.22). Je možné vidět počty procesů a dobu jejich běhu v závislosti na čase spuštění, způsobu ukončení, případné vzniklé chybě, hostiteli, použité databázi *message box,* typu a názvu procesu a knihovně, ve které je orchestrace implementována. Co takhle zase něco úplně jiného?



Obrázek 5.22: Sledování historického trendu spuštěných procesů

5.2 Oddělení obchodních a administrativních pravidel

Obchodní a administrativní pravidla řídí základní aspekty veškerých probíhajících procesů. Jsou všudypřítomná, takže si je mnohdy ani neuvědomujeme. Přestože se v odborné literatuře nazývají zpravidla pouze obchodní *(business rules),* nemusí mít vůbec nic společného s obchodem. Uveďme si pár takových pravidel: na stravenky vracíme maximálně 5 Kč, držitelé průkazky tělesně postižených mají slevu 50%, lidem s více než třiceti zápisy v obchodním rejstříku na fakturu neprodáváme, lidé smějí volit pouze v místě svého trvalého bydliště nebo s platným voličským průkazem a tak dále. Obchodní a administrativní pravidla neřídí pouze lidské chování, nezbytně se odrážejí též v softwarových aplikacích. Objevují se zde v podobě podmínek a větvení toku kódu. Tento přístup má řadu nevýhod a zbytečných kroků:

- Při vývoji je nutno pravidla převádět ze srozumitelného a stručného „obchodního“ jazyka analytiků do detailní podoby podmíněných příkazů programovacího jazyka.
- Jakmile k tomuto převodu jednou dojde, ztrácí se vazba na původní pravidla, která již nejsou navenek patrná a zpětně dohledatelná. Pravidla obsažená v kódu již nejsou srozumitelná analytikům.
- Pravidla obsažená v kódu jsou promíchána s dalšími procedurálními funkcemi a nelze je jednoznačně izolovat od zbytku kódu.
- Pokud je pravidlo složitější, je jeho procedurální zápis velmi obtížný – stává se nepřehlednou spletí podmínek spojených logickými operátory do nečitelných struktur.
- Pravidla se mění daleko častěji než kód, který je obklopuje. Nicméně při každé změně pravidel je nutné nasazení nové verze celé aplikace. Tato aktualizace může být organizačně náročná, drahá a mnohdy i riskantní.

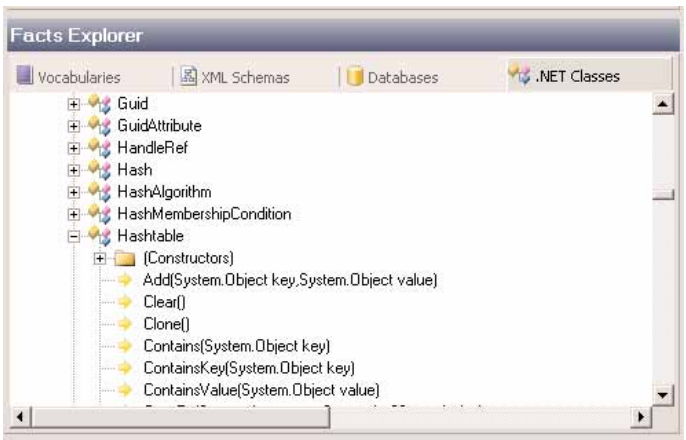
I přes uvedené nevýhody je „pohřbívání“ obchodních pravidel v kódu velmi rozšířené, zejména díky absenci lepších alternativ. Implementace správy procesů mimo aplikace, které jsme se věnovali v kapitole 5.1 situaci vylepšuje pouze částečně – pravidla jsou sice lokalizována uvnitř procesu mimo stávající aplikace, ale řada nevýhod uvedených výše se příliš nezmenšuje. Změna obchodních pravidel ukrytých v orchestraci znamená novou kompilaci knihovny, v níž je obsažena, její nové otestování, a nasazení. Dopad je sice menší než u změny v aplikaci, ale změna procesu kvůli změně pravidla je stále zbytečná. Pravidla, kterými se procesy řídí, se mění výrazně častěji než procesy samotné.

Poslední dobou se začínají prosazovat speciální softwarové komponenty pro vyhodnocování pravidel, často nazývané *business rules engine* (BRE). Jejich použití je poměrně jednoduché. Pokud potřebuje aplikace rozhodnout nějakou složitější situaci (zda schválit úvěr, jakou udělit slevu apod.), nepokouší se provést rozhodnutí sama. Místo toho

shromáždí veškeré podklady nezbytné pro rozhodnutí a předá je vyhodnocovacímu stroji. Ten podle definovaných pravidel fakta posoudí a dospěje k závěru, který předá zpátky aplikaci. Aplikace pak pokračuje v provádění v závislosti na předaném výsledku. Pokud je nutné pravidla změnit, dotýká se tato změna pouze rozhodovacího stroje, nikoliv vlastní aplikace, a to je v tomto případě zřejmě největší výhra. Samozřejmě jsou odstraněny i další výše uvedené nevýhody, jak už to v knihách bývá. Pojďme se teď podívat blíže na základní koncepty a implementaci obchodních pravidel v BizTalk Serveru 2004.

Fakta

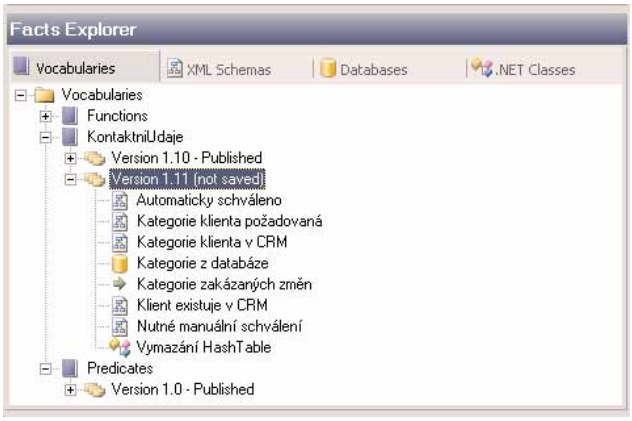
Fakta jsou základními objekty, se kterými pravidla pracují. Představují buď vstupní podklady nutné pro vyhodnocení pravidel (výše dlužné částky, věk, hrubý roční příjem žadatele, požadovaný počet kusů) nebo naopak výstupy pravidel (maximální možná výše úvěru, velikost slevy, schválení žádosti, rozhodnutí o odebrání řidičského průkazu apod.) BRE v BizTalku umí pracovat se čtyřmi různými typy faktů: konstantami (a jejich množinami nebo rozsahy), XML dokumenty platnými podle nějakého schématu, objekty .NET frameworku (resp. jejich členskými proměnnými, vlastnostmi a metodami) a konečně databázovými tabulkami a sloupci. Odpovídají jim operace vstupu a výstupu. Při vstupu dochází k dosazení konstanty, zjištění hodnoty prvku/atributu XML dokumentu, zjištění hodnoty vlastnosti/proměnné/návratové hodnoty metody objektu nebo přečtení řádku z tabulky databáze. Výstup pak představuje aktualizace hodnoty prvku/atributu XML dokumentu, nastavení hodnoty vlastnosti/proměnné/vyvolání metody objektu nebo aktualizace údajů v databázi. Fakta lze vytvářet vizuálně pomocí nástroje *Business Rules Composer* (viz obrázek 5.23).



Obrázek 5.23: Průzkumník faktů pro vytváření obchodních pravidel

Slovníky faktů

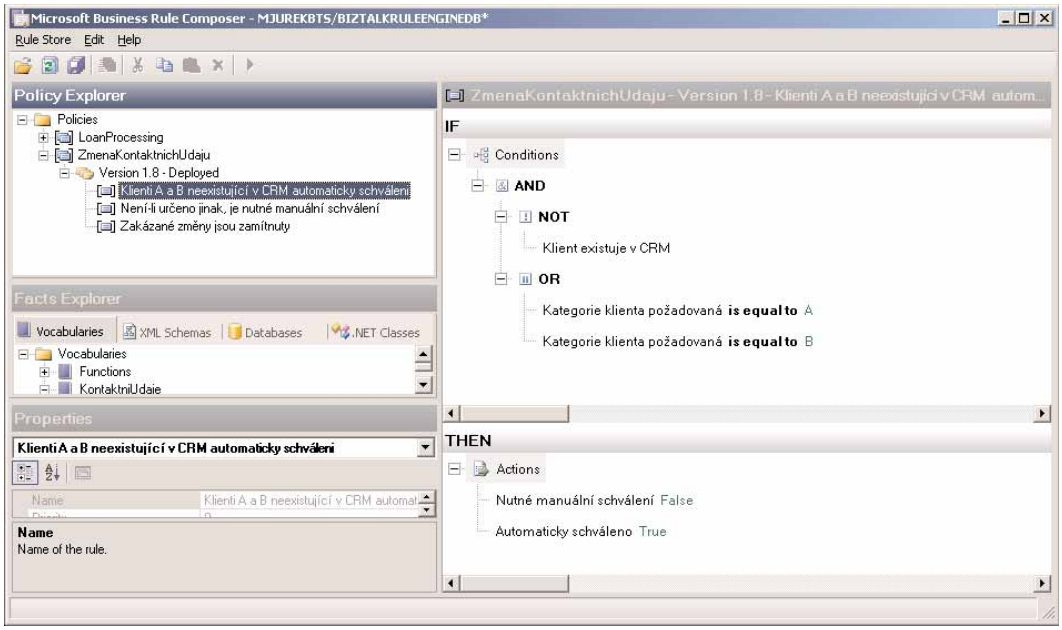
Slovník faktů je slovník obsahující fakta. Hmm, objevné. Tak jinak. Určitě jste někdy viděli právní smlouvu, ve které bylo uvedeno zhruba toto: „Firma X zastoupená panem Y (dále jen dodavatel)“. Ve zbytku textu se pak už mluvilo pouze o dodavateli. Smlouvy někdy dokonce obsahují celou jednu kapitolu, ve které jsou definovány termíny. Slovník faktů *(vocabulary)* plní stejnou funkci. Je souborem pojmenovaných faktů, které používáme při definici pravidel (viz obrázek 5.24). Při vytváření pravidel oceníte zejména jejich uživatelsky příjemné pojmenování. Přece jenom jméno „Automaticky schválené“ je zřejmější než schéma dokumentu a XPath cesta k určitému prvku. Druhou, ještě významnější výhodou je definice faktu na jediném místě. Pokud se některý z faktů změní, stačí změnu provést na jednom místě, není nutné hledat všechny jeho výskyty. Pravidla není nutné vytvářet pomocí faktů ze slovníku, fakta je možné definovat jednorázově pro účely určitého pravidla. Doporučeným postupem je ovšem používání slovníků faktů, určitě tomuto postupu přijdete také na chuť. Hotový slovník faktů je v BizTalk Serveru možné uložit do repositáře BRE, kterým je – jak jinak – databáze SQL Serveru. Druhým krokem k používání slovníku faktů je jeho opublikování, které jej učiní dostupným pro vytváření pravidel. Opublikovaný slovník již nelze modifikovat, pokud v něm chcete provést nějaké změny, musíte zkopírováním vytvořit jeho novou verzi. Existují též dva vestavěné slovníky, jeden pro funkce, které je možné aplikovat na fakta a druhý pro predikáty používané ve vyhodnocování podmínek.



Obrázek 5.24: Slovník faktů pro použití v pravidlech

Politiky a pravidla

Politika slouží k vyhodnocení určité situace v rozhodovacím procesu (výpočet výše úvěru, určení slevy, schválení žádosti). Politika je souborem pravidel s definovanou prioritou (viz obrázek 5.25). Opět je uložena v repositáři a způsob jejího schvalování je tentokrát dokonce trojstupňový. Nejprve se uloží do repositáře, poté se opublikováním zveřejnění pro potřeby testování a vývoje, teprve nasazením se umožní její využití v provozním prostředí. Nasazenou politiku je možné stáhnout, ale není možné ji modifikovat. Pokud chcete politiku změnit, musíte vytvořit novou verzi, uložit, opublikovat, nasadit a případně stáhnout předchozí verzi.



Obrázek 5.25: Politika se skládá z pravidel

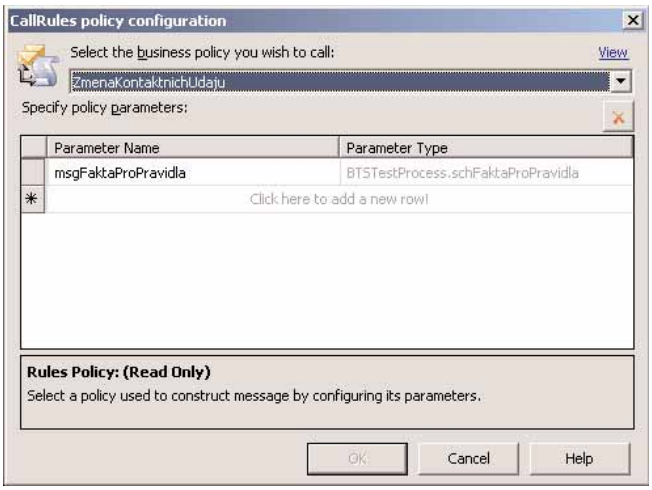
Každé pravidlo je tvořeno podmínkami a akcemi. Pokud jsou podmínky v pravidle splněny, vykonávají se příslušné akce. Podmínky jsou tvořeny tzv. predikáty (*Equal, Between, GreaterThan* apod.) aplikovanými na jednotlivá fakta nebo funkce operující nad těmito fakty. Predikáty jsou pak spojeny logickými operátory *AND, OR* a *NOT*. V příkladě na obrázku jsou podmínky splněny (a tudíž pravidlo použito), pokud není pravdivý fakt *Klient existuje v CRM* a pokud je zároveň fakt *Kategorie klienta požadovaná* roven *A* nebo *B*. Při splnění podmínek se provedou příslušné akce, kterými je typicky nastavení hodnoty některého faktu, méně často systémové akce (odstranění faktu z paměti, zastavení provádění a jiné). V našem případě se hodnota faktu *Nutné manuální schválení* nastaví na nepravdu a hodnota faktu *Automaticky schváleno* na pravdu.

Typická politika obsahuje řadu pravidel, které je nutné nějakým způsobem skloubit. BRE používá třístupňový algoritmus pro vyhodnocení pravidel:

- *Matching* – v první fázi jsou nalezena všechna pravidla, která mají k dispozici nezbytná fakta ke svému vyhodnocení a provádění akcí. Z nich jsou vyloučena pravidla, jejichž podmínka není splněna.
- *Conflict resolution* – v druhé fázi jsou v této množině hledána pravidla, která jsou v konfliktu, tj. provádějí akce nad stejnými fakty. Tato pravidla jsou seřazena sestupně podle klesající priority, což znamená, že akce u pravidel s nejvyšší prioritou jsou prováděny nejdříve. Setříděné akce ze všech postoupivších pravidel tvoří tzv. agendu. Paradoxně tak může akce pravidla s nižší prioritou zvrátit výsledek akce s nižší prioritou.
- *Action* – v poslední fázi jsou prováděny akce z agendy. Zajímavá situace nastává, pokud některá akce změní fakta použitá pro vyhodnocování podmínek pravidel, což způsobí opakování prvních dvou kroků pro dotčená pravidla (tzv. *forward chaining*). Toto chování může být někdy nežádoucí, proto je možné jej kontrolovat pomocí systémových akcí *Assert, Retract* a *Update*.

Nasazení v provozním prostředí

Jakmile je politika hotová, je možné ji použít v provozu. BRE má objektový model, takže v podstatě libovolná aplikace může vytvořit objekt vyhodnocovacího stroje, předat mu potřebná fakta, spustit vyhodnocení a čekat na výsledek. BRE je tudíž možné používat bez jakékoliv vazby na ostatní komponenty BizTalk Serveru. Ovšem nejefektivnější je použití v kombinaci s orchestrací, kde máme k dispozici tvar *Call Rules* (viz obrázek 5.26). Jeho použití je mimořádně jednoduché – jedinou úlohou je výběr politiky k vyhodnocení a výběr zpráv a proměnných, které budou politice předány jako fakta.



Obrázek 5.26: Volání vyhodnocení pravidel z orchestrace

Pro přenos mezi vývojovým, testovacím a provozním prostředím lze použít nástroj *Rule Engine Deployment Wizard,* který vizuálně zpřístupňuje řadu operací s politikami a slovníky faktů. Pomocí tohoto nástroje můžeme publikovat slovníky faktů nebo publikovat, nasazovat a stahovat z nasazení politiky. Možný je rovněž export politiky nebo slovníku faktů z SQL repositáře do XML souboru anebo jeho import opačným směrem.

Testování a monitorování

Před nasazením politiky je vhodné ji nejprve otestovat na připravených vzorových souborech faktů k vyhodnocení. BRE má infrastrukturu pro monitorování těchto testů nazývanou *tracking interceptor*. Testování je možné buď vlastním testovacím kódem nebo přímo v nástroji pro vytváření pravidel. Monitorovací výstup zahrnuje veškeré operace s fakty (přidání a ubrání), vyhodnocování podmínek pravidel, spouštění akcí a aktualizaci agendy akcí.

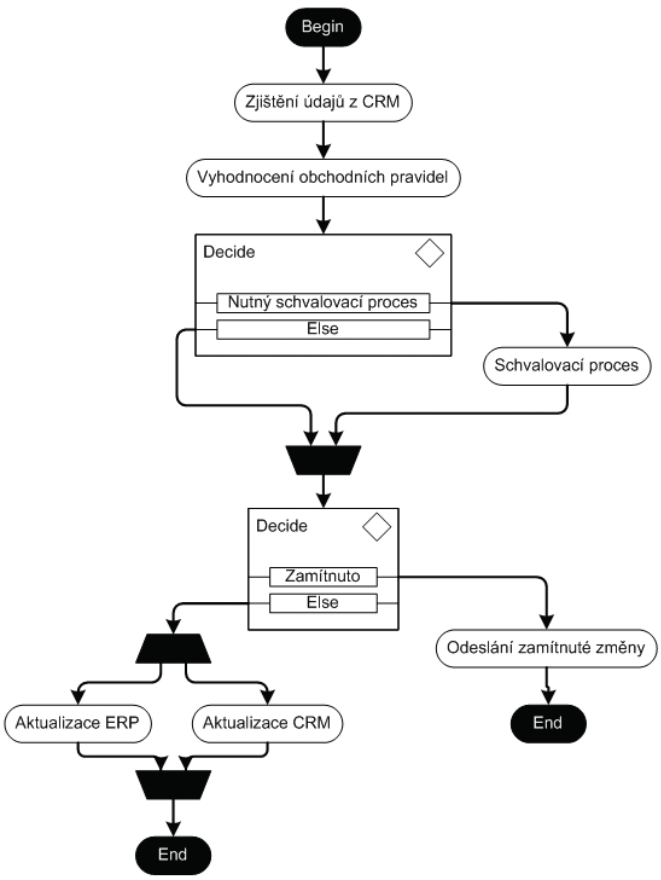
Stejné sledování je možné provádět i za provozu. V nástroji HAT *(Health and Activity Tracking)* je možné pro jednotlivé politiky zapnout monitorování vybraných událostí. Při provádění politik se do monitorovací databáze zaznamenávají záznamy o jednotlivých událostech, takže je zpětně možné dohledat prováděná pravidla a akce. Bohužel v nástroji HAT není k dispozici žádný vhodný dotaz pro jejich prohlížení, tudíž si případné dotazy nebo auditovací pomůcky musíte vyrobit sami. Není už té teorie dost?

5.3 Praktický příklad

Vědomosti týkající se správy procesů a obchodních pravidel využijeme v praktickém příkladě. Budeme implementovat proces zadání nových nebo aktualizace stávajících kontaktních údajů. Tyto údaje jsou uloženy ve více existujících systémech, nejdůležitější z nich je CRM systém. Vzhledem k tomu, že údaje jsou důležitou a citlivou informací, není možné je aktualizovat bezhlavě, je nutné implementovat proces, ve kterém se zkontroluje správnost nových údajů podle obchodních pravidel a rozhodne se o dalším postupu. Další informace:

- Proces aktualizace je přístupný jako webová služba, které je předán XML dokument s aktualizovanými údaji. Jednoznačným identifikátorem klienta je jeho rodné číslo.
- Pro zjištění dalšího postupu je nutné zjistit, zda již klient existuje v databázi CRM systému a jakou má kategorii. CRM systém je vytvořen podle principů architektury orientované na služby, přičemž zjištění kategorie je přístupné jako jedna z operací webové služby.
- Další postup se řídí obchodními pravidly. Není-li klient v CRM systému a má být zařazen do kategorie A nebo B, je možné aktualizaci provést bez schvalovacího procesu. Pokud je klient v CRM systému a je zařazen do kategorie X, není aktualizaci možné provést. Ve všech ostatních případech je nutný schvalovací proces, ve kterém musí někdo aktualizaci schválit.
- Pokud ve schvalovacím procesu nepřijde žádná odpověď během zadaného časového intervalu, považuje se žádost o změnu za zamítnutou.
- Pokud se aktualizace provádí, jsou změny zaneseny do CRM systému (pomocí volání webové služby) a zároveň do ERP systému (jde o historický systém, kterému se předává textový soubor ve formátu odděleném čárkami).
- Pokud aktualizaci není možné provést nebo byla zamítnuta ve schvalovacím procesu, je příslušný dokument odeslán e-mailem správci aplikace.

Celý proces můžeme znázornit diagramem na obrázku 5.27 vytvořeném v již zmíněné nadstavbě Visia zvané *Orchestration Designer for Business Analyst*. Tento soubor je též možné exportovat do **.odx* souboru pro definici orchestrace ve Visual Studiu.

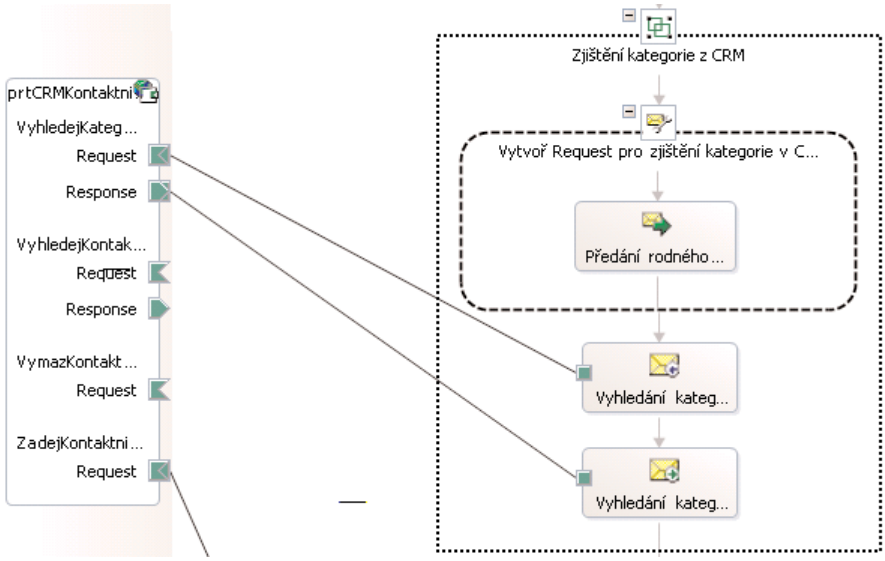


Obrázek 5.27: Proces změny kontaktních údajů

Při implementaci budeme potřebovat celkem čtyři definice schémat – pro kontaktní údaje na vstupu (stejné schéma je použité i pro aktualizaci CRM systému), údaje o zákazníkovi pro ERP systém, dokument pro vyhodnocení obchodních pravidel a dokument pro odeslání do schvalovacího procesu. Dále musíme vytvořit tři transformační mapy, které z dokumentu na vstupu vytvoří transformací ostatní tři potřebné dokumenty.

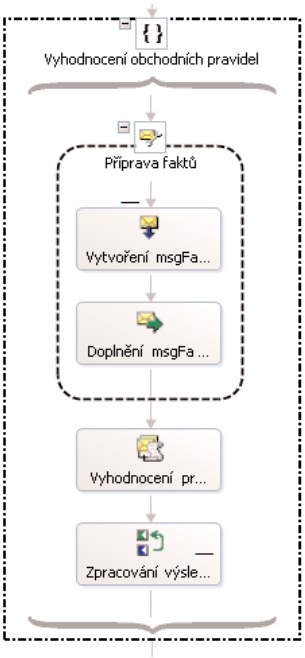
Rovněž budeme potřebovat dva *pipeline* typy. První slouží pro kontrolu vstupního dokumentu podle XSD schématu (kontroluje se např. formát rodného čísla – 9 nebo 10 číslic). Druhý typ *pipeline* slouží pro odeslání do ERP systému, kde je nutné provést převod na textový soubor oddělený čárkami. Pro ostatní vstupní a výstupní operace vystačíme se standardními *pipeline* typy.

V první části jsou přijaty nové kontaktní údaje akcí pro přijetí. Poté je nutno zjistit, zda již tento klient existuje v CRM systému (viz obrázek 5.28). Nejde o složitou operaci, její nejdůležitější částí je vytvoření reference na webovou službu CRM systému pomocí funkce *Add Web Reference,* čímž jsou vytvořeny potřebné typy portů a zpráv pro komunikaci s webovou službu. Volání webové služby spočívá ve vytvoření příslušného portu pro volání služby typu, které obsahuje operaci *VyhledejKategorii* (je typu solicit-response), vytvoření zprávy pro odeslání na webovou službu pomocí akce *Construct Message,* její odeslání akcí *Send* a obdržení odpovědi akcí *Receive*.



Obrázek 5.28: Zjištění kategorie klienta v CRM systému

Nyní je nutné zjistit, jaký bude další postup v závislosti na obchodních pravidlech. K tomu potřebujeme zavolat připravená obchodní pravidla za pomoci tvaru *Call Rules* (viz obrázek 5.29). Před jejich voláním je nutné připravit fakta. V našem příkladě jsou všechna fakta soustředěna do jediné XML zprávy typu *FaktaProPravidla*. Zprávu vytvoříme transformací příchozí zprávy a následným doplněním dalších faktů (existence klienta v CRM systému a jeho kategorie) ze zprávy, která je výsledkem našeho zjišťovacího dotazu do CRM systému.

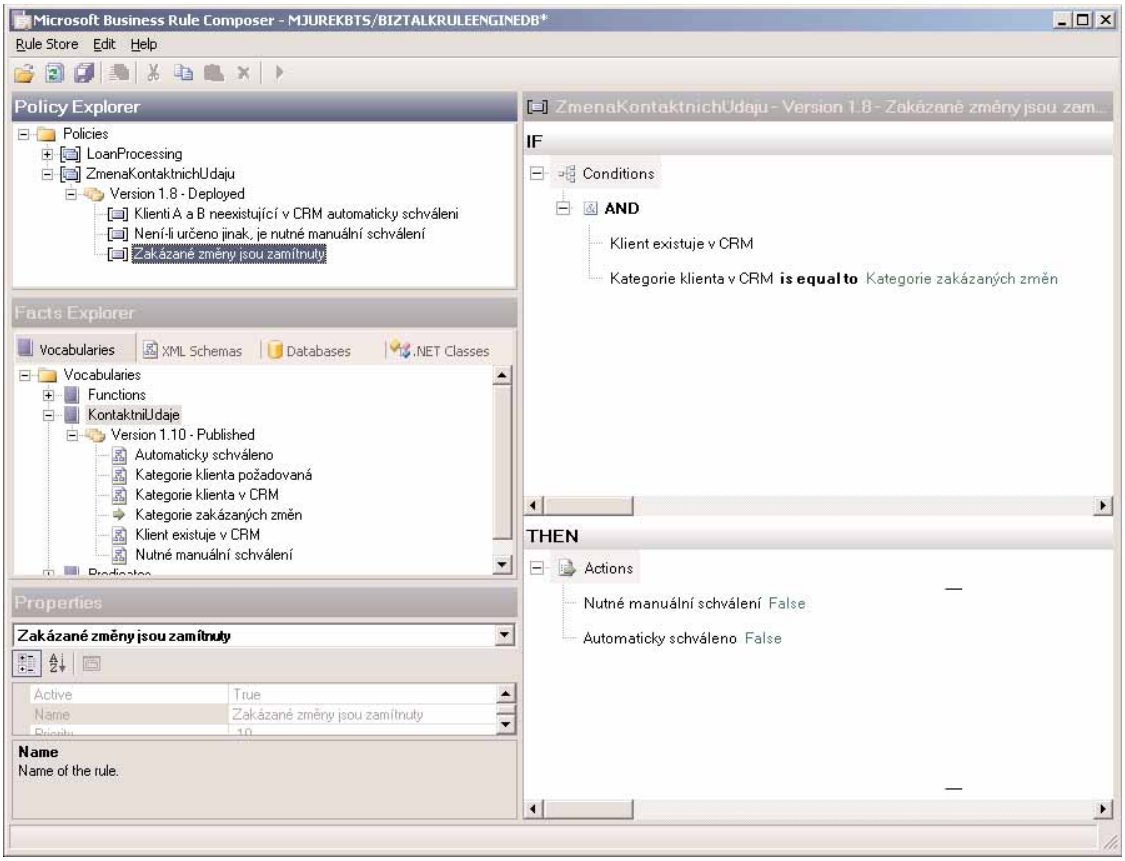


Obrázek 5.29: Příprava faktů a vyhodnocení obchodních pravidel

Vlastní politiku pro schvalování změn kontaktních údajů tvoří tři pravidla (viz obrázek 5.30):

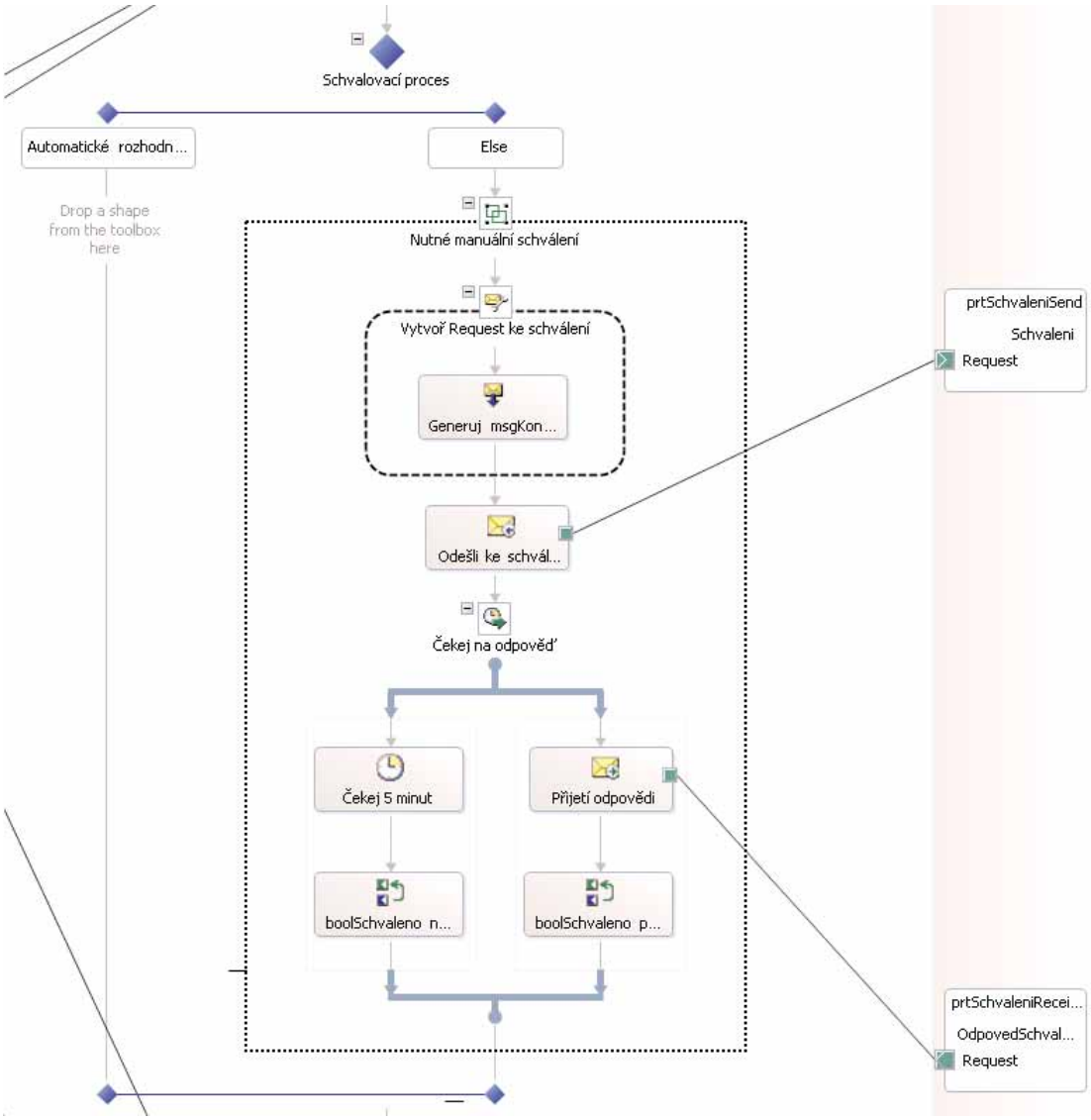
```
Klienti A a B neexistující v CRM automaticky schváleni (priorita 0)
   IF
     NOT (Klient existuje v CRM)
   AND
        Kategorie klienta požadovaná is equal to A
     OR
        Kategorie klienta požadovaná is equal to B
   THEN
     Nutné manuální schválení false
     Automaticky schváleno true

Není-li urèeno jinak, je nutné manuální schválení (priorita 10)
   IF
     1 is equal to 1
   THEN
     Nutné manuální schválení true

Zakázané zmìny jsou zamítnuty (priorita -10)
   IF
     Klient existuje v CRM
   AND
     Kategorie klienta v CRM is equal to Kategorie zakázaných zmìn
   THEN
     Nutné manuální schválení false
     Automaticky schváleno false
```

Tato pravidla jsou celkem bez problémů čitelná (kupodivu, když právě čitelnost externích pravidel je jedním z důvodů, proč je neutopit v kódu). V pravidlech můžete napočítat celkem šest faktů – jednu konstantu (Kategorie zakázaných změn = X), tři části XML dokumentu pro čtení a dvě pro zápis. Za pozornost stojí též přiřazené priority – pravidla se provádí od nejvyšší k nejnižší prioritě, proto jsme výchozí pravidlo umístili jako první a nepřehlasovatelné pravidlo zakázaných změn jako poslední, aby mohlo případně zvrátit výsledky ostatních pravidel. Možná vás též zaujala konstrukce *1 is equal to 1*. Důvodem jejího použití je moje neschopnost najít elegantnější způsob vyjádření vždy pravdivé podmínky. Po provedení pravidel proces pokračuje akcí *Expression,* ve které se hodnoty dvou faktů nastavených pravidly ve formě XML elementů uloží do dvou logických proměnných pro další snadné použití v orchestraci.



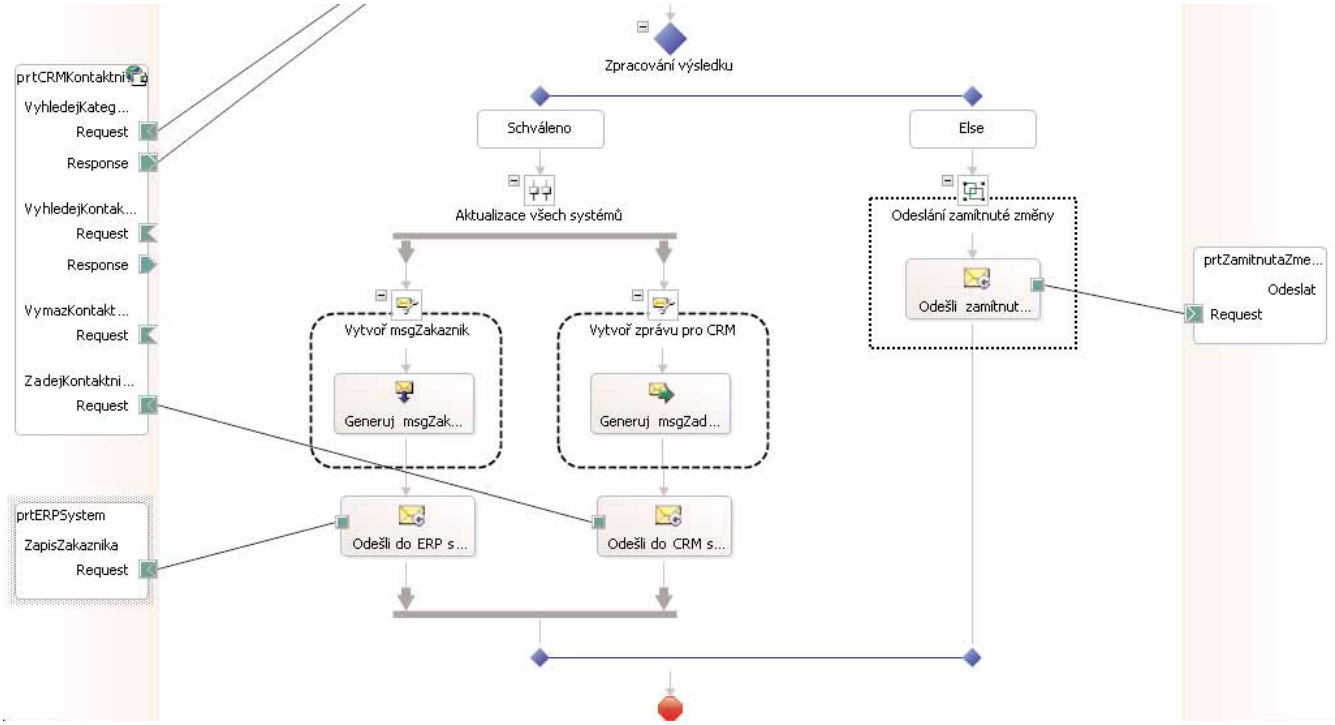
Obrázek 5.30: Pravidla pro rozhodnutí o změně kontaktních údajů

Na toto použití nebudeme dlouho čekat. Hned v dalším kroku se musíme rozhodnout, jak dále pokračovat – bude potřeba schvalovací proces nebo nebude? Odpověď na tuto otázku je uložena v jedné ze zmíněných logických proměnných, takže stačí rozvětvit provádění pomocí akce *Decide* a otestování její hodnoty. Jedna z větví je prázdná. Druhá, zajímavější větev provádí schvalovací proces (viz obrázek 5.31). Nejprve vytvoříme zprávu pro odeslání do schvalovacího procesu, která obsahuje základní údaje o aktualizovaných kontaktních údajích a dále vygenerovaný jednoznačný identifikátor *(GUID)*. Tento identifikátor je použit pro korelaci příchozí odpovědi se správnou instancí orchestrace. Ve schvalovacím procesu je do zprávy pouze doplněna logická hodnota o rozhodnutí (schválení nebo zamítnutí). Pro komunikaci se schvalovacím procesem je použit *message queueing (MSMQ)*. Schvalovací proces je externí služba, na jejíž bezchybnou funkci není možné spoléhat. Je možné, že žádná zpráva o schválení či zamítnutí nepřijde. Proto při čekání na odpověď použijeme akci *Listen* v jehož jedné větvi očekáváme zprávu akcí *Receive* a v druhé větvi čekáme 5 minut, což je samozřejmě nerealisticky málo (ne že bychom si tak rychlé úředníky nepřáli), ale při zkoušení nových technologií budete jistě stejně netrpěliví jako já. Pokud vyprší čas dříve než přijde zpráva, je hodnota logické proměnné pro schválení nastavena na nepravdu. Přijde-li dříve zpráva, je tatáž proměnná nastavená podle XML elementu v odpovědi.



Obrázek 5.31: Podmíněné provedení schvalovacího procesu

Poslední fáze už je čistě výkonná a větví se na základě logické proměnné nastavené v dřívějších fázích procesu (viz obrázek 5.32). Pokud je zápis kontaktních údajů schválen, je nutné aktualizovat kontaktní údaje v ERP systému a CRM systému. Obě akce je pro urychlení možné provádět paralelně. Pro každý systém je nejprve vytvořena příslušná zpráva a poté je odeslána do systému pomocí akce *Send*. CRM systém je zpřístupněn jako webová služba, takže předání zprávy je velmi jednoduché. ERP systém vyžaduje textové soubory pro import zákaznických dat, proto pomocí připravené *pipeline* vytvoříme textový soubor a pro jeho odeslání použijeme *FILE* adaptér. Pokud změna kontaktních údajů nemůže být provedena, je původní doručená zpráva odeslána správci systému pomocí *SMTP* adaptéru.



Obrázek 5.32: Provedení změn nebo odeslání zprávy o zamítnutí

O tom, že veškeré popsané postupy si můžete prakticky vyzkoušet v příloze B jsem se již několikrát zmiňoval, není-liž pravda?

Kapitola 6:

Analytické monitorování

Nejvyšší vrstvou brokeru je vrstva obchodně-analytického monitorování, kde již definitivně opustíme svět technických termínů a vstoupíme do světa obchodních či organizačních ukazatelů. Pozor – jde o oblast, kde může vedoucí IT pracovník dobře „prodat“ užitečnost své práce managementu firmy! Probíhající procesy jsou totiž blízké poslání firmy či organizace a zkoumáním těchto procesů lze poměrně dobře kontrolovat plnění tohoto poslání.

Vrstva správy obchodních procesů vytváří velké množství dat, z nichž lze usuzovat na celkové zdraví implementovaného procesu. Jde přitom o čísla, která se pravděpodobně nikdy neobjeví v klasických datových skladech. Datové sklady zpravidla obsahují informace o výsledcích činnosti, nikoliv o procesech vedoucích k těmto výsledkům. Přitom právě zdokonalení procesů je nejpřímější cesta k lepším výsledkům. Na základě statistického vyhodnocení těchto dat můžeme zodpovědět na otázky jako: „Jaká je průměrná doba zpracování žádosti v závislosti na místě bydliště žadatele?“, „Kolik žádostí bylo dnes přijato a kolik vyřízeno?“ nebo „Jak se mění finanční hodnota objednávek v různých stádiích rozpracovanosti s časem?“. Právě tyto otázky, respektive odpovědi na ně představují vysoký přínos investice do integračního projektu a IT pracovník by neměl přehlédnout jejich vysokou „politickou“ hodnotu.

Moderní integrační brokery zpravidla nabízí nějakou formu analytické nadstavby nad obchodními procesy a BizTalk Server 2004 není výjimkou – vrstva analytického monitorování se v něm nazývá *Business Activity Monitoring (BAM) Framework*. Nejprve se podíváme na obecnou definici analytického modelu procesu a poté na jeho implementaci v BizTalk Serveru 2004.

6.1 Definice analytického modelu

Analytický model pro sledování procesu zahrnuje dvě základní otázky: Jaká fakta jsou sbírána během provádění procesu? Jakým způsobem tato fakta vyhodnotit?

Sběr faktů

Sběr faktů se v terminologii BizTalk Serveru nazývá obchodní aktivita *(business activity)*. Nemusí mít nic společného s obchodem, ale budeme se držet překladu. Pokud si představíme probíhající proces nebo procesy, budou nás pravděpodobně zajímat dva základní typy údajů – milníky a fakta.

Prvním typem údajů jsou takzvané milníky procesu, tedy důležité body, kterými proces prochází. Milník představuje čas, kdy proces dosáhl onoho důležitého bodu anebo konstatování, že proces tohoto bodu zatím ještě nedosáhl (a třeba ani nikdy nedosáhne). Pokud se podíváte na definici orchestrace v příkladu z kapitoly 5.3, můžete v ní najít kolem 25 různých akcí. Řada z nich je pouze technických a není z hlediska celkového procesu důležitá. Pokud byste řekli netechnickému uživateli, že proces právě dosáhl bodu *„Nastavení proměnné boolSchvaleno“,* setkali byste se v lepším případě se shovívavým úsměvem. Při bližším zkoumání můžeme jsem nalezl osm zajímavých milníků procesu:

- Zahájeno – proces byl zahájen
- Vyhodnocen postup – byly shromážděny všechny údaje pro určení dalšího postupu
- Automaticky rozhodnuto – podle obchodních pravidel bylo vyhodnoceno, že o změně není nutné manuálně rozhodovat
- Čeká na manuální rozhodnutí – bylo vyhodnoceno, že je nezbytné manuální rozhodnutí o schválení
- Manuálně rozhodnuto – procesu byl doručen výsledek manuálního rozhodovacího procesu
- Vypršel čas – výsledek manuálního rozhodnutí nebyl doručen v časovém limitu
- Provedeno – změny kontaktních údajů byly provedeny
- Zamítnuto – změny kontaktních údajů byly zamítnuty

Druhým typem údajů jsou fakta – numerické nebo textové údaje důležité pro monitorovaný proces, které má proces nějakým způsobem k dispozici. Prozkoumáním příkladu jsem identifikoval celkem čtyři zajímavá fakta pro proces z kapitoly 5.3:

- Křestní jméno
- Příjmení
- Rodné číslo
- Kategorie

Samozřejmě můžeme (oprávněné) argumentovat, zda křestní jméno je opravdu důležitým údajem, ale bude se nám hodit při dohledávání údajů o provedených procesech. Vše je otázkou správné analýzy závislé na potřebách potenciálních konzumentů analytických údajů.

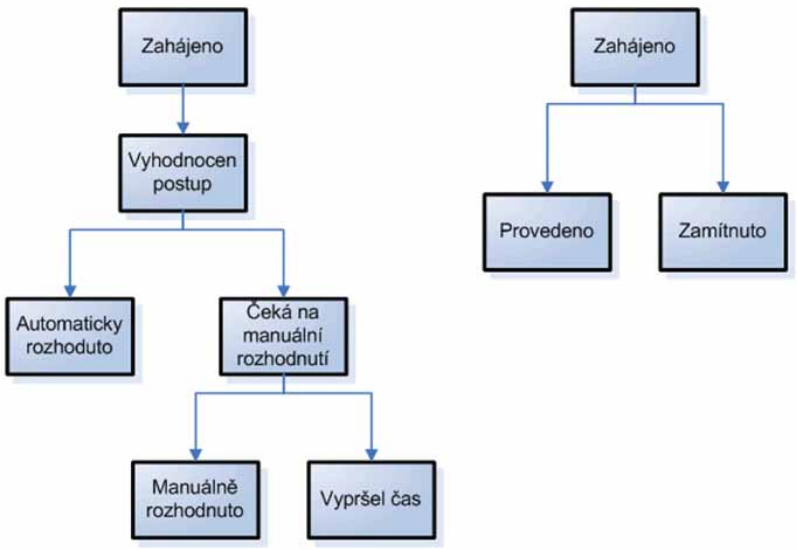
Vyhodnocení faktů

Nasbíraná fakta je třeba nějakým způsobem vyhodnotit – v terminologii BizTalk Serveru se vyhodnocení nazývá *business view*. Tento pohled na data si můžete představit jako multidimenzionální zobrazení věcí, které mne zajímají (metriky problému) v závislosti na okolnostech či parametrech (dimenze problému). Pracujeme zde se stejnými termíny jako v analytických technologiích OLAP *(on-line analytic processing)* a jistě není náhodou, že tyto technologie se poté používají při implementaci analytického monitorování. Pojďme si nyní projít jednotlivé fáze definice pohledu.

Prvním krokem je volba aktivity (nebo aktivit), sloužících jako zdroj dat pro vytvářený pohled, případně výběr relevantních milníků a faktů.

Druhým krokem je určitá příprava milníků a definice vztahů mezi nimi. Můžeme definovat skupiny milníků, např. milník „Ukončen proces“ je dosažen tehdy, je-li dosaženo milníku „Provedeno“ nebo milníku „Zamítnuto“. Skupina milníků se tak sama stává novým, srozumitelně pojmenovaným milníkem. Dále můžeme definovat časové rozdíly mezi milníky. Při N milnících můžeme sestavit až N*(N-1) různých dvojic, ale pouze několik z nich má nějaký smysl. V našem příkladě jsem našel dvě takové kombinace. „Doba manuálního rozhodnutí“ je doba uplynuvší mezi dosažením milníků „Čeká na manuální rozhodnutí“ a „Manuálně rozhodnuto“. „Doba trvání procesu“ je doba mezi milníkem „Zahájeno“ a skupinou milníků „Ukončen proces“ (tvořenou milníky „Provedeno“ a „Zamítnuto“).

Třetím krokem je definice dimenzí. Dimenze si je možné představit jako okolnosti či parametry daného procesu. Pomocí dimenzí se potom analyzují nebo filtrují metriky procesu. Zajímavým typem dimenzí jsou dimenze o postupu procesu *(progress dimension)*. Kdysi jsem v technickém muzeu viděl zajímavý exponát na demonstraci statistického rozdělení. Byla to jakási plexisklová skříň, do které se horem vhazovaly kuličky. Kuličky padaly na kaskádu ostrých hran, po kterých sklouzávaly buď doprava nebo doleva. Dole byly mističky, ve kterých se kuličky hromadily za účelem spočítání. Dimenze o postupu procesu fungují analogicky. Proces si představme jako kuličku vhozenou do horního tvaru. Proces propadává v grafu směrem dolů, ovšem nikoliv náhodně, ale podle nastavených pravidel větvení a vnějších okolností. V každém z tvarů (milníků) potom měříme počet procesů, jenž jimi prošly. Mezi našimi osmi milníky jsem objevil dvě takovéto zajímavé dimenze (viz obrázek 6.1). První sleduje proces z hlediska postupu schvalování. Druhá se soustředí na výsledek celého procesu, jímž je buď aktualizace kontaktních údajů v systémech nebo její zamítnutí.



Obrázek 6.1: Definice dimenzí o postupu procesu

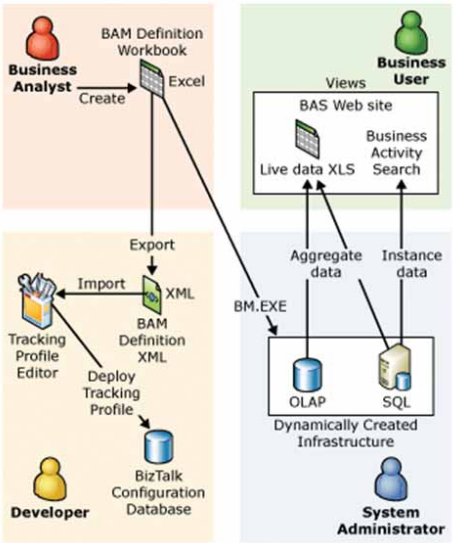
Dalším možným typem jsou časové dimenze, které umožňují proces analyzovat z hlediska času, ve kterém proces prošel určitým milníkem, přičemž je možné určit hierarchii této dimenze (např. rok/kvartál/měsíc/den nebo hodina/minuta). Při ohledání našeho procesu jsem narazil na tři zajímavé časové dimenze – podle milníku „Zahájeno“ ukazující na frekvenci příchodu nových žádostí, podle milníku „Manuálně rozhodnuto“ vypovídající

o pracovním nasazení příslušných schvalovatelů a podle milníku „Ukončen proces“ umožňující sledovat uzavírání žádostí s přímou vazbou na zátěž aktualizovaných systémů. Posledním zajímavým typem dimenzí jsou dimenze datové, sestavené na základě faktů příslušné aktivity. Vzhledem k výběru faktů bude určitě jednou zajímavou dimenzí „Kategorie“ analyzující došlé žádosti podle kategorie žadatele, druhou dimenzí pak může být „Žadatel“ tvořený hierarchií hodnot Příjmení/Křestní jméno/Rodné číslo.

Čtvrtým a posledním krokem je definice měr chilli [čili] metrik, což jsou ukazatele relevantní pro zkoumání kvality a kvantity prováděných procesů. Fantazii se zde meze nekladou, moje vymyslela celkem pět různých metrik: počet procesů, průměrná doba manuálního rozhodnutí, maximální doba manuálního rozhodnutí, průměrná doba trvání procesu a maximální doba trvání procesu. Jak ale všechno naimplementovat?

6.2 Implementace monitorování

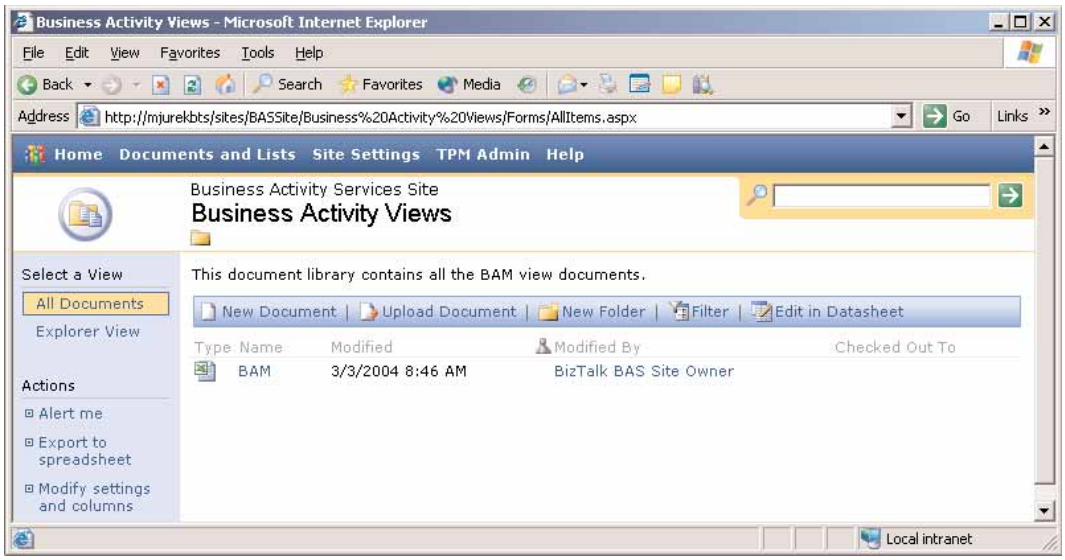
Implementace analytického monitorování má několik kroků, z nichž každý přísluší osobě s jinou úlohou. V celém scénáři můžeme identifikovat celkem čtyři vystupující osoby – analytika, vývojáře, administrátora a koncového uživatele (viz obrázek 6.2). V reálném světě samozřejmě nemusí jít o čtyři různé osoby, často jeden člověk zastává více než jednu roli. Probereme si nyní úlohy jednotlivých rolí v procesu implementace.



Obrázek 6.2: Postup implementace analytického monitorování

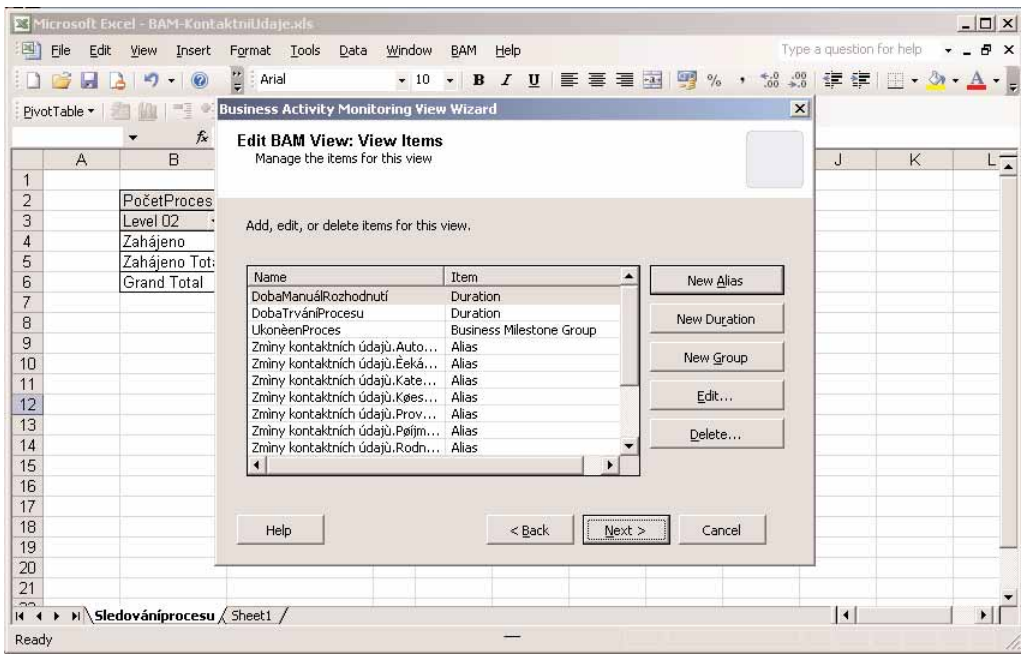
Analytik

Analytik přichází na řadu jako první. Jeho úkolem je vymyslet analytický model podle postupu v kapitole 6.1. Prvním úkolem analytika je definovat jednu nebo více aktivit *(business activity),* tj. milníky procesu a zajímavá fakta hodná sběru. Jeho druhým úkolem je definovat jeden nebo více multidimenzionálních pohledů *(business view),* tj. vztahy mezi milníky, dimenze o postupu procesu, datové dimenze, časové dimenze a míry. Výsledky své analýzy potom musí v nějakém tvaru předat vývojáři a administrátorovi. V případě BizTalk Serveru je pro popis aktivit a pohledů použit standardní formát XML. Od analytika, tedy pravděpodobně technicky neznalého uživatele, těžko můžeme očekávat schopnost práce s XML. Proto je k dispozici doplněk pro *Microsoft Excel* zvaný *BAM Definition Workbook*. Jde o šablonu Excelu s podepsanými makry. Je součástí instalace a je možné ji otevřít libovolným způsobem. V případě plné instalace se nabízí přímá možnost otevření z intranetového portálu nazvaného *Business Activity Services Site,* který používá portálovou technologii *Windows SharePoint Services,* součást Windows Serveru 2003 (viz obrázek 6.3).



Obrázek 6.3: Otevření analytického modelu z webu SharePointu

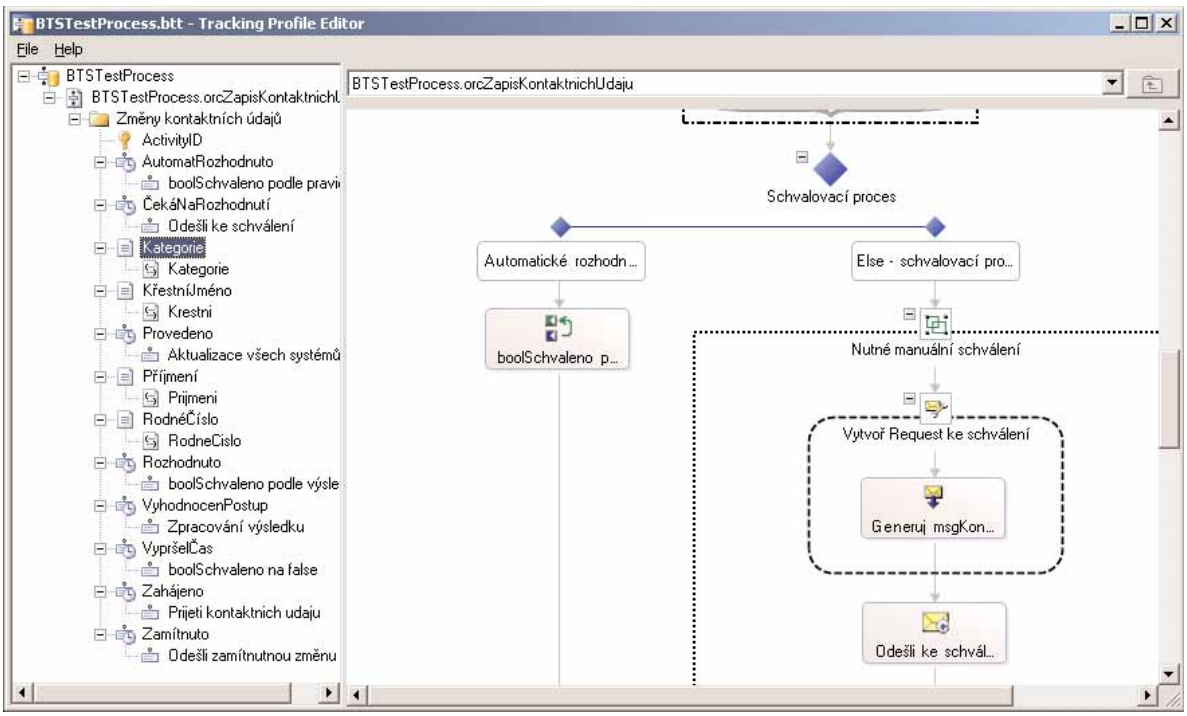
Můžeme sice pochybovat o analytikově schopnosti pracovat s Excelem, nicméně naše šance jsou stále výrazně vyšší než v případě XML. Doplněk Excelu je určitě možné využít přinejmenším k vytvoření vlastní metodiky pro definování aktivit a pohledů. Jeho instalace přidá do hlavní nabídky Excelu tři nové položky. První spustí průvodce pro definování aktivit, druhá průvodce pro definování pohledu (viz obrázek 6.4). Můžeme vytvářet nové položky, měnit nebo mazat existující položky. Vytvářená data jsou uložena na skrytých listech Excelu a mimo průvodce nejsou viditelná. Po ukončení práce můžeme celý sešit Excelu uložit obvyklým způsobem. Doporučené je uložení do knihovny dokumentů portálu *SharePoint* (viz obrázek 6.3). K dispozici máme též kontingenční tabulku k procházení a kontrole našeho modelu s náhodně vygenerovaným vzorkem dat. Konečně posledním krokem naší práce v roli analytika bude odevzdání hotové definice vývojáři a administrátorovi v XML tvaru. K jeho vygenerování použijeme třetí a poslední položku z přidané nabídky Excelu jménem *Export XML ...*



Obrázek 6.4: Definice analytického modelu v Excelu

Vývojář

Práce analytika je tak trochu stavěním vzdušných zámků. Po vytvoření modelu je čas vrátit se zpátky na zem a navázat model na konkrétní orchestraci. Právě v tom spočívá role vývojáře – musí nakonfigurovat orchestraci tak, aby poskytovala veškeré potřebné údaje pro navrhovanou aktivitu. Může se ukázat, že ne všechny původně navrhované součásti modelu jsou prakticky realizovatelné. Vývojář používá nástroj zvaný *Tracking Profile Editor* (viz obrázek 6.5). Nejprve vybere definici orchestrace nasazené na serveru (vidíme ji v pravé části nástroje) a k ní přidá definici aktivity z XML souboru dodaného analytikem (v levé části). Poté graficky přiřadí každému milníku některou z akcí orchestrace a každému z požadovaných faktů část některé ze zpráv doručených nebo odeslaných orchestrací. Vytvořený profil je možné nasadit na server *(Deploy)*. Pokud nemá vývojář k této operaci oprávnění, může sledovací profil uložit do souboru a doručit správci k nasazení na provozní server. Od této chvíle probíhající instance sledované orchestrace ukládají nastavená fakta a časy průchodů jednotlivými milníky do sledovací databáze.



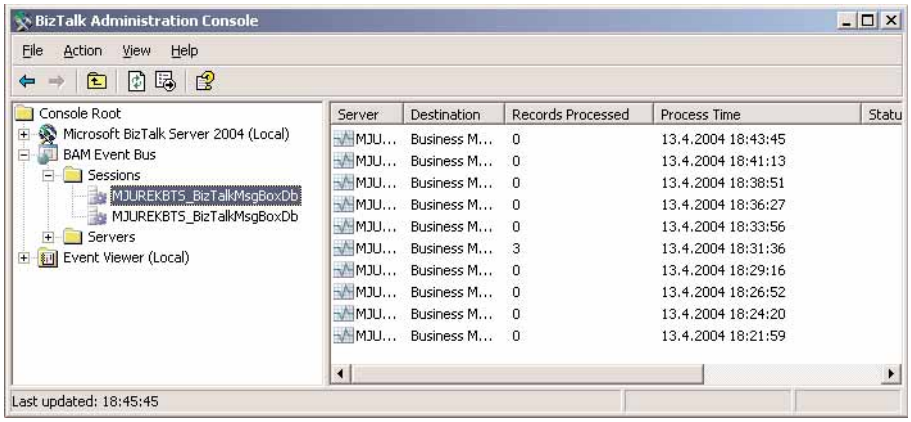
Obrázek 6.5: Přiřazení mezi částmi orchestrace a aktivitou

Administrátor

Administrátor má při implementaci dvě základní úlohy – připravit databáze pro analytické monitorování a poté udržovat celé řešení v chodu. První úkol je relativně snadný – stačí spustit nástroj *bm.exe* a předat mu jako parametr XML soubor dodaný analytikem. Nástroj vytvoří následující objekty:

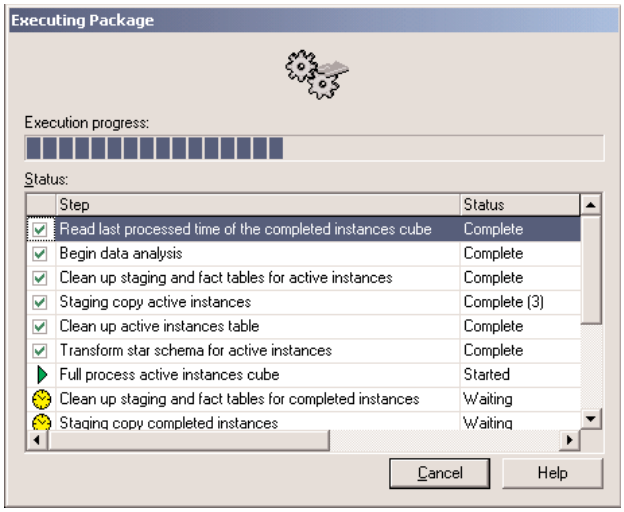
- Tabulky pro definované aktivity v primární databázi pro import sledovaných údajů *(BAMPrimaryImport)*
- Ekvivalentní tabulky v archivní databázi sledovaných údajů *(BAMArchive)*
- Tabulky pro definované pohledy v datovém skladu *(BAMStarSchema)*. Jak jeho název napovídá, jde o vzorový datový sklad s uspořádáním tabulek faktů a dimenzí do hvězdice.
- Kostky pro definované pohledy v analytické databázi OLAP
- Transformační balíček pro DTS *(Data Transformation Services),* který inkrementálně přenáší data z primární databáze údajů do datového skladu s uspořádáním hvězdice a poté inkrementálně zpracuje kostku v analytické databázi
- Transformační balíček pro archivaci neaktuálních údajů z primární do archivní databáze sledovaných údajů

Provozování řešení není nikterak složité. Údaje, které jsou zaznamenány během provádění procesu, jsou zhruba v dvouminutových intervalech dávkovým způsobem přenášeny ze všech serverů ve farmě do primární databáze. Tento proces je možné sledovat v administrativním nástroji (viz obrázek 6.6).



Obrázek 6.6: Monitorování zaznamenávání analytických informací

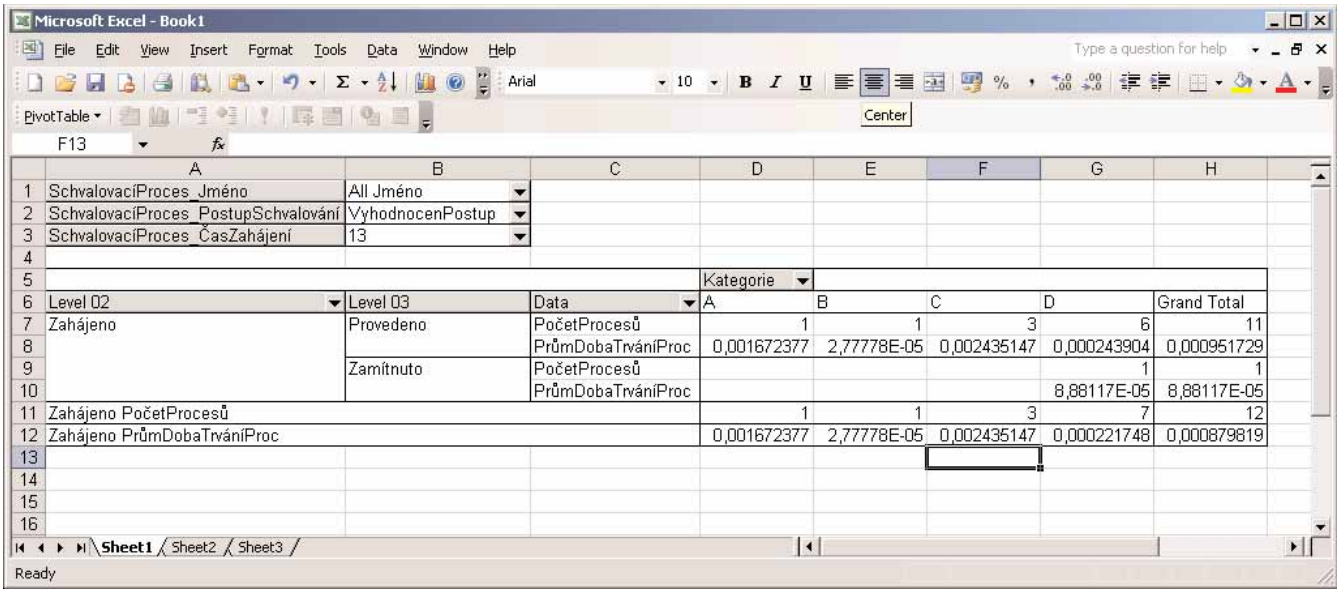
Přenos dat do datového skladu a zpracování příslušných kostek je úkolem vygenerovaného balíčku. Úlohou administrátora je jeho pravidelné spouštění (viz obrázek 6.7) nebo zajištění jeho periodického spouštění, např. pomocí služby *SQL Agent*. Čím častěji se balíček spouští, tím jsou analytické monitorovací informace aktuálnější – samozřejmě za cenu vyšší zátěže serveru během zpracování.



Obrázek 6.7: DTS balíček pro aktualizaci datového skladu a OLAP kostek

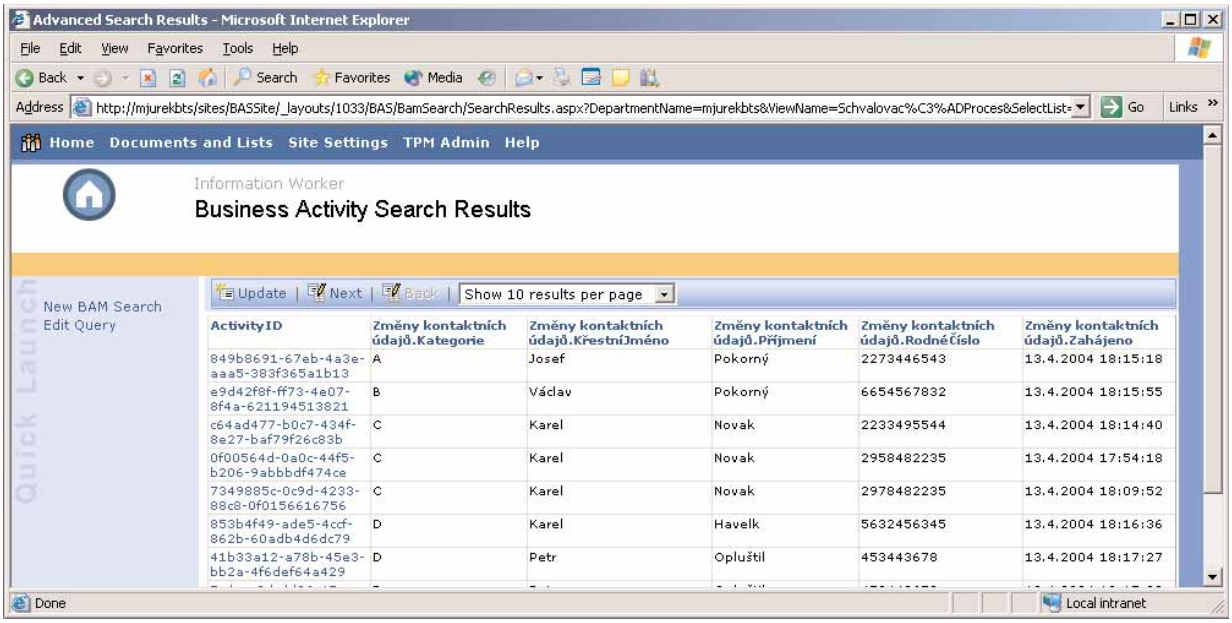
Koncový uživatel

Koncový uživatel vystupuje v roli konzumenta informací v datovém skladě nebo multidimenzionálních kostkách, k čemuž může použít různé nástroje. Standardním nástrojem zkušenějších uživatelů pro práci s OLAP kostkami může být Microsoft Excel a jeho kontingenční tabulky. Na obrázku 6.8 vidíme příklad analýzy procesů. Procesy jsou filtrovány podle času zahájení, zajímají nás metriky pro počet procesů a jejich průměrnou dobu trvání, analyzujeme je podle výsledku procesu (Provedeno/Zamítnuto) a podle kategorie, ve které byl klient zařazen.



Obrázek 6.8: Multidimenzionální analýza v Excelu

Protože jsou veškeré analytické informace dostupné v databázích, je možné vytvořit vlastní řešení, například webovou aplikaci nebo řízený report produktu *SQL Server Reporting Services*. Ke stejnému účelu můžete využít též webovou službu, přes kterou jsou analytické informace přístupné (je ve virtuálním adresáři *BAMQueryWebSvc*). Vzorová aplikace využívající tuto službu se nazývá *Business Activity Search* a je implementována pomocí portálové technologie *Windows SharePoint Services* (viz obrázek 6.9). Umožňuje výběr pohledu a aktivity, volbu sloupců, které se mají zobrazit ve výsledku, a nastavení podmínek pro filtrování výsledku.



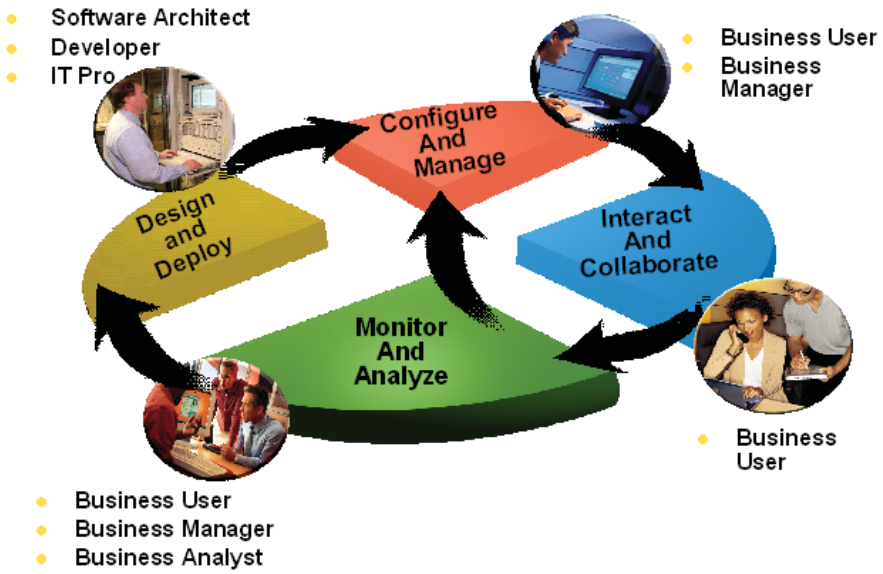
Obrázek 6.9: Dohledání zaznamenaných faktů přes webové rozhraní

Náš příběh se zvolna chýlí ke konci. Probrali jsme všechny čtyři vrstvy definované ve druhé kapitole. Zbývá odpověď na jedinou otázku – jak do integračního řešení zapojit živočichy druhu Homo sapiens?

Kapitola 7:

Klienti – interakce s uživatelem

Pokud otevřete libovolný marketingový materiál s tématem integrace, zcela jistě v něm najdete cosi podobného obrázku 7.1. Nepřehlédnutelným prvkem v něm bývá člověk či koncový uživatel. Ať už si o smyslu podobných diagramů myslíme cokoliv, lidský faktor v integračním řešení pominout nemůžeme. Vždyť většina procesů je vytvářena právě pro lidi.



Obrázek 7.1: Marketingový pohled na problematiku integrace

Proto je předposlední kapitola brožury věnována právě koncovým uživatelům. Ve skutečnosti nás budou zajímat spíše aplikace, které budou koncoví uživatelé používat, pro tyto aplikace budu poněkud volně používat označení „klient“. Nejprve si nastíníme rozsah možných řešení a poté se budeme více věnovat modernějším možnostem – aplikaci *InfoPath,* jež je novým přírůstkem v rodině Office, a portálové technologii *SharePoint*.

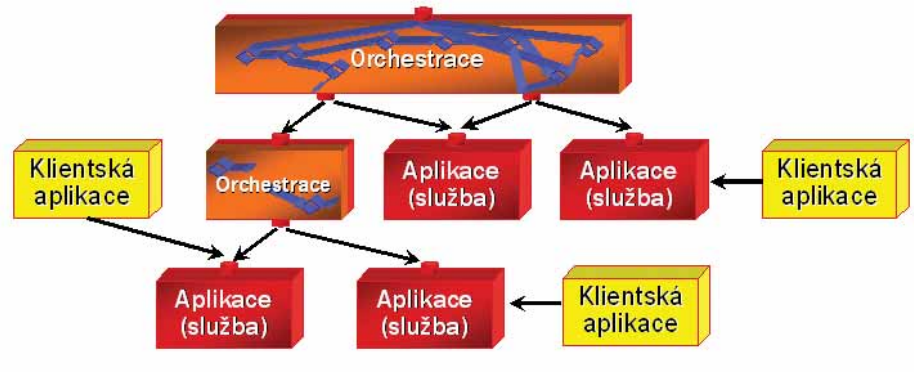
7.1 Nepřeberný výčet možností

Množství různých klientských řešení je opravdu nepřeberné. Liší se zařazením klienta do celkové architektury řešení, použitou technologií pro komunikaci s brokerem, architekturou vlastního klienta a dalšími znaky. Uvedeme si několik možností v každé kategorii, z nichž si pak můžete zkombinovat obraz vlastního řešení ušitého na míru konkrétnímu problému.

Kam klienta zařadit?

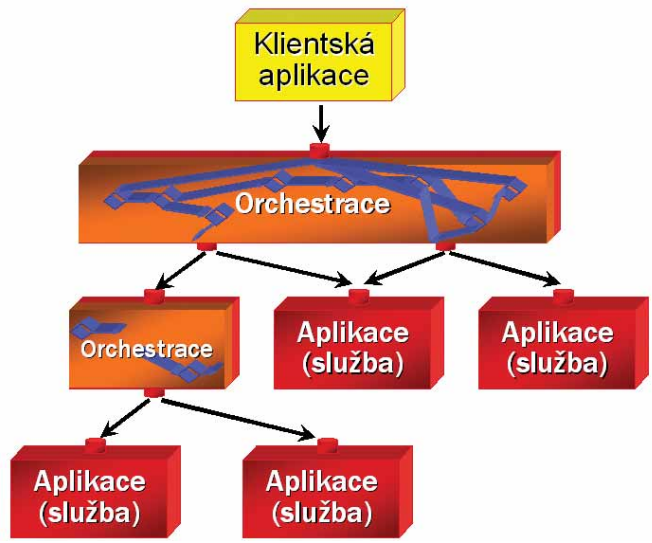
V předchozích kapitolách jsme se klientem nezabývali. Architektura všech našich řešení se skládala z integračního brokeru a integrovaných aplikací (čili služeb v architektuře orientované na služby). Integrační broker vytváří orchestrace, které komunikují s aplikacemi (službami). Orchestrace mohou též volat jiné orchestrace. Do tohoto uspořádání nyní zařadíme klienta.

První extrémní možností je nevytvářet žádné speciální klienty pro integrační proces (viz obrázek 7.2). Klienti pak komunikují přímo s integrovanými aplikacemi – klient ERP systému komunikuje se svým serverem, CRM řešení má vlastního klienta apod. Pro klienty je integrační řešení zcela neviditelné. Integrované aplikace reagují na události v nich vzniklé (přijetí faktury, aktualizace kontaktních údajů apod.) komunikací s integračním brokerem. Broker se postará o vše potřebné – doručení zpráv dalším aplikacím podle metafory *publish/subscribe,* spuštění nové orchestrace apod.



Obrázek 7.2: Klienti pracují s primárními aplikacemi

Druhou krajní možností je napojení klienta přímo na integrační broker (viz obrázek 7.3). Klient pak komunikuje s integračním brokerem, předává mu zprávy, spouští orchestrace apod. Broker komunikuje s integrovanými aplikacemi a stará se o korektní dokončení klientem iniciovaných akcí. Vlastní integrované aplikace se pro klienty stávají zcela neviditelnými. Tento přístup je výhodný například v situaci, kdy integrovaná aplikace je kritická pro chod aplikace, ale má klienta, který již po technologické stránce nevyhovuje požadavkům organizace. Vytvoří tedy nového klienta, kterého od integrované aplikace oddělí integračním brokerem. Jinou situací může být omezenost stávajících aplikací vzhledem k novým požadavkům. Pokud implementovaný proces komunikuje se třemi aplikacemi a vyžaduje v některém kroku schválení uživatelem, může být snadnější implementovat tento krok v integračním brokeru než v některé z integrovaných aplikací.



Obrázek 7.3: Klienti komunikují s integračním brokerem

Většina řešení se bude pohybovat někde mezi těmito dvěma mezními případy. Klienti budou komunikovat jak s integrovanými aplikacemi, tak s integračním brokerem. Pokud navážeme na příklad z kapitoly 5.3, může být způsob komunikace pro zadávání změn v kontaktních údajích následovný: Klient komunikuje přímo s CRM systémem, ze kterého může získat kontaktní informace již existujících uživatelů. Pokud je třeba kontaktní údaje aktualizovat, klient upraví dokument s kontaktními údaji a předá ho integračnímu brokeru, který se postará o aktualizaci údajů ve všech provozních systémech.

Jakou technologii použít pro komunikaci?

Integrační broker komunikuje se svým okolím téměř vždy prostřednictvím adaptérů. Otázku lze proto převést na problém: Který adaptér pro komunikaci s integračním brokerem použít? Následující tabulka přehledně popisuje možnosti základních adaptérů spolu s vysvětlením, co jejich použití znamená pro klientskou aplikaci.

Adaptér	Komunikace směrem k brokeru	Komunikace směrem od brokeru
SOAP	volání vzdálené webové služby	zpracování příchozích SOAP volání
MSMQT	odeslání zprávy pomocí rozhraní služby MSMQ	vyzvednutí zprávy z fronty MSMQ
FILE	vytvoření souboru se zprávou ve vzdáleném adresáři	monitorování adresáře a zpracování zpráv v něm nalezených
HTTP	odeslání zprávy v těle HTTP POST požadavku	zpracování příchozích HTTP POST požadavků se zprávami
SMTP	odeslání e-mailu obsahujícího zprávu (Outlook, SMTP/Sockets, ...)	zpracování příchozího e-mailu se zprávou (Outlook, ...)
FTP	vytvoření souboru se zprávou v adresáři přes FTP	monitorování adresáře přístupného přes FTP a zpracování zpráv v něm nalezených
SQL	vytvoření záznamu v databázové tabulce SQL serveru monitorované brokerem	zpracování záznamů vytvářených nebo modifikovaných brokerem v databázové tabulce SQL serveru

Který z těchto adaptérů použít? To není snadná otázka. Samozřejmě použijeme takový způsob komunikace (a tedy i adaptér), který je pro daný scénář nejvhodnější. Pokud se vám moje odpověď zdá poněkud vyhýbavá, pak věřte že:

- Pokud máte možnost volby, použijte SOAP adaptér. Je jednoduchý na použití, založený na standardech a je po technologické stránce perspektivní.
- Požadujete-li asynchronní komunikaci aplikace s brokerem anebo transakční zpracování zpráv, použijte adaptér pro MSMQT.
- Pro testovací účely je nejjednodušší a nejrychleji nastavitelný FILE adaptér.

Zmínil jsem se, že SOAP adaptér je jednoduchý na použití. Z pohledu klienta komunikace se SOAP adaptérem znamená vytvoření SOAP zprávy a její odeslání na server pomocí protokolu HTTP. Díky dobrému popisu rozhraní webových služeb pomocí standardu WSDL je na klientovi možné opustit svět protokolů a zpráv a místo toho používat maximálně pohodlný objektový model. Tento komfort u ostatních způsobů komunikace pravděpodobně nebudeme mít k dispozici. Moderní vývojové nástroje totiž umožňují z WSDL dokumentu vygenerovat tzv. *proxy* třídy, které zastupují skutečnou webovou službu. Klient se nemusí starat o protokoly a formáty dokumentů a může pracovat se zástupnými objekty podobně jako v následujícím fragmentu kódu:

```
public void ZapisVzoroveKontaktniUdaje()
{
  // Pøiprav tøídu s kontatkními údaji pro zapsání
  KontaktniUdaje ku=new localhost.KontaktniUdaje();
  KontaktniUdajeJmeno jm=new KontaktniUdajeJmeno();
  jm.Krestni="Karel";
  jm.Prijmeni="Novák";
  ku.Jmeno=jm;
  ku.CasovaZnamka=DateTime.Now;
  ku.Kategorie="C";
  ku.RodneCislo="1234567890";
  // Zavolej webovou službu
  KontaktniUdajeSluzba sluzba=new KontaktniUdajeSluzba();
  sluzba.Zmen(ku);
}
```

Jaký typ klienta vytvořit?

Dalším rozhodnutím, které musíme učinit, je volba typu klienta. Vzhledem k rozmanitosti možných způsobů komunikace s brokerem je škála řešení velmi široká a lze říci, že téměř všechny vývojové technologie na všech dostupných platformách jsou použitelné. Samozřejmě každá z nich má svoje výhody a nevýhody. Pro úplnost uvedu možnosti klientů na platformě .NET spolu s jejich výhodami i nevýhodami:

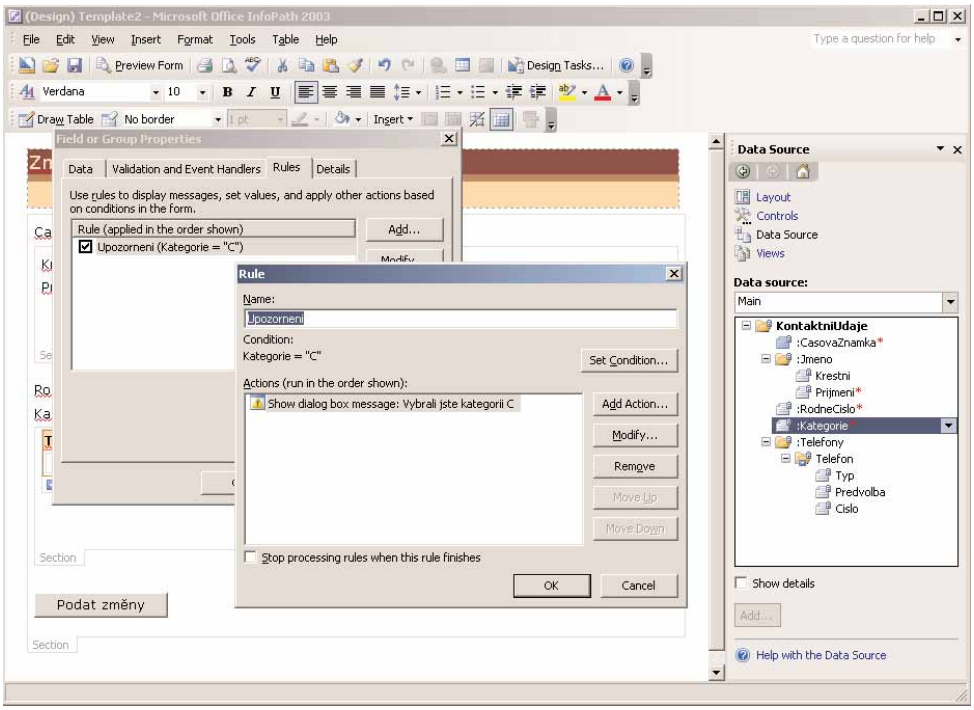
- Webová aplikace v ASP.NET – webové aplikace jsou velmi populární, neboť nevyžadují žádnou instalaci a drahou údržbu na klientovi a jsou bez problémů podporovány širokou škálou klientských operačních systémů. Na druhou stranu nevyužívají potenciál klienta, mají poměrně omezený uživatelský komfort a neumožňují práci v odpojeném režimu *(offline)*.
- Chytrá „tlustá“ aplikace nad .NET frameworkem – v moderních prostředích pro běh programů, jakým je i .NET framework, odpadá nutnost manuální instalace a aktualizace tlustých klientů. Klientské aplikace mohou být plně automaticky nainstalovány a aktualizovány přes URL adresu, čímž se odstraňuje jejich dosavadní největší nevýhoda. Plně využívají možnosti klienta, nabízejí plný uživatelský komfort včetně práce v odpojeném režimu. Na druhou stranu jejich správa je i přes technologický pokrok složitější, zejména díky nutnosti zohlednit operační systém a další komponenty nainstalované na klientovi.
- Tenká aplikace běžící na široké škále mobilních zařízení – uživatelé dnes používají širokou škálu mobilních telefonů, digitálních diářů, pagerů apod. Tato zařízení mají zpravidla jednoduchý prohlížeč neschopný zobrazování plných HTML stránek určených pro desktopové počítače. ASP.NET obsahuje speciální ovládací prvky *mobile controls* schopné generovat stránky pro více než 200 mobilních zařízeních s podporou HTML, cHTML nebo WML/WAP. Hlavní výhodou je široký dosah řešení, základní nevýhodou je velmi nízký komfort uživatelského rozhraní.
- Chytrá „tlustá“ mobilní aplikace nad kompaktním .NET frameworkem – podmnožina .NET frameworku je určena pro zařízení s operačním systémem na bázi *Windows CE*, zejména diáře s *PocketPC 2003* a telefony *SmartPhone 2003*. Tato zařízení se používají buď v bezdrátových sítích (sklady a podobné „terénní“ provozy) anebo připojena přes GPRS technologii mobilních sítí (obchodní cestující apod.). I přes omezený displej a uživatelské ovládání lze na těchto zařízeních vytvořit velmi komfortní aplikace podobné desktopovým. Vývoj je přitom téměř stejně efektivní jako u desktopových aplikací, včetně podpory XML a webových služeb.
- Webový dílec *(web part)* portálové technologie *SharePoint* – portálové technologie používají koncept webových dílců (v .NETu *web part,* v J2EE *portlet*). Jde o objekty vytvářející části uživatelského rozhraní, ze kterých se pak na portále skládají webové stránky. Portálové technologie získávají na popularitě, proto jim budeme věnovat samostatnou kapitolu 7.3. Ve srovnání s klasickými webovými aplikacemi umožňují lepší zapojení koncových uživatelů a větší flexibilitu, jinak jsou si svými výhodami a nevýhodami podobné.
- Formulář aplikace *InfoPath* a další produkty rodiny Office – horkou novinkou ve spektru možností je *InfoPath,* nový přírůstek v rodině Office produktů. Jedná se o formulářovou aplikaci k prohlížení a editaci XML dokumentů umožňující komunikaci prostřednictvím webových služeb. Je ideálním prostředkem pro rychlé vytváření integračních řešení a jejich interakci s koncovým uživatelem. Na druhou stranu není nejvhodnější pro ucelené aplikace typu účetnictví. Blíže se na něj podíváme v kapitole 7.2. Práce s XML a webovými službami je možná též v aplikacích *Excel* a *Word*.

7.2 InfoPath a Office 2003

Aplikace *Office* přestávají být pouhými editory dokumentů, tabulek či prezentací. Strategickým směrem rozvoje je jejich obohacení o funkce, které z nich vytvoří programovatelné klienty interních aplikací, jakými jsou účetní nebo CRM systémy. Pro tyto scénáře je klíčová podpora XML a webových služeb v aplikacích *Office* – ve verzi 2003 byla přidána zejména do aplikací *Word* a *Excel*. Zaměření na standardy podtrhuje také zbrusu nová klientská aplikace *InfoPath*.

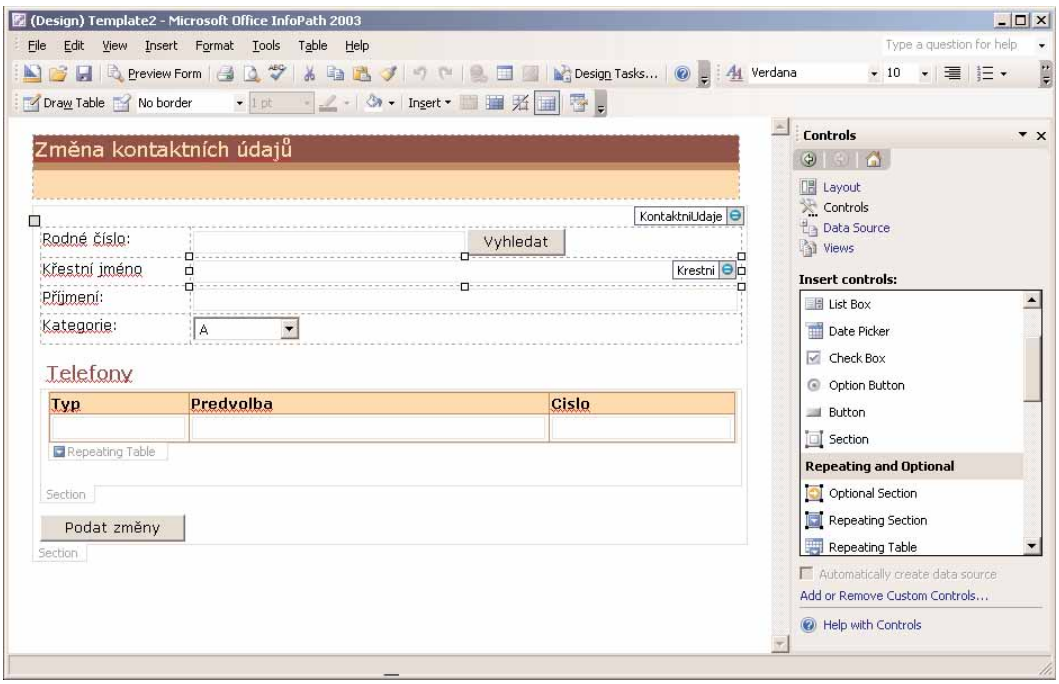
Vytváření aplikací v InfoPath

Microsoft Office InfoPath 2003 je velmi zajímavým produktem, ideálním pro komunikaci s integračním řešením. Je to formulářová aplikace pro práci s XML dokumenty. Při vytváření formuláře musíme zadat datový zdroj *(data source)* – XSD schéma, podle kterého se bude XML dokument vytvářet. Alternativně můžeme schéma nechat vygenerovat ze vzorového XML dokumentu, z WSDL popisu rozhraní webové služby, z databázové tabulky nebo ze seznamu položek v portálové technologii *SharePoint*. *InfoPath* plně respektuje použité schéma a nedovolí nám vytvořit a uložit dokument, který není podle schématu platný. Kromě toho můžeme definovat další restrikce a pravidla neošetřená schématem, případně programově reagovat na událost před a po změně hodnoty položky. Pro elementy a atributy můžeme nastavovat jejich výchozí hodnoty, pomocí pravidel a akcí nad nimi můžete jednoduše deklarativně programovat (viz obrázek 7.4).



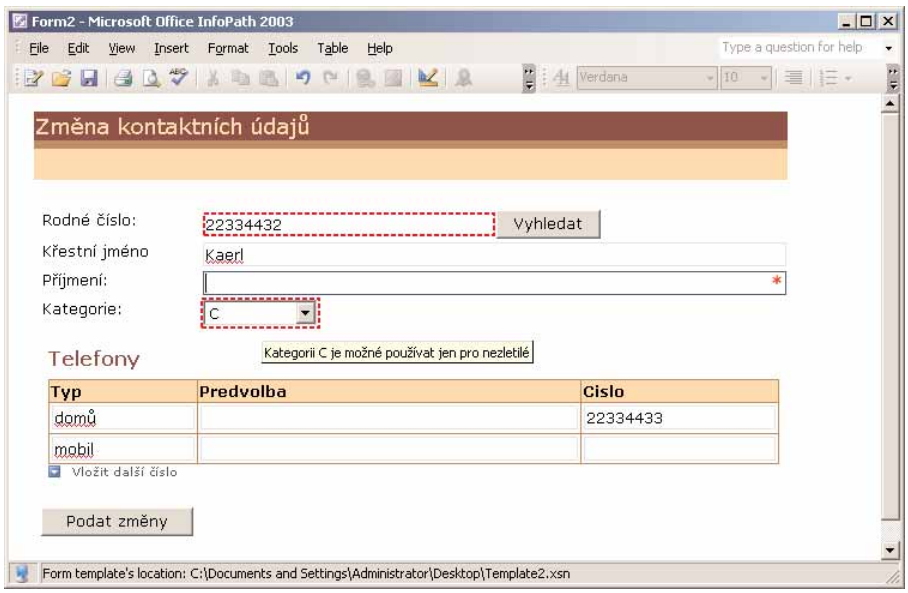
Obrázek 7.4: Práce s definicí XML dokumentu v aplikaci InfoPath

Dalším krokem je vytvoření uživatelského rozhraní. *InfoPath* formulář definuje tzv. pohledy *(Views)* pro různé akce nad daty. Např. můžete mít jeden pohled pro zadávání nové žádosti, druhý pohled pro její schvalování a třetí pohled optimalizovaný pro tisk. Mezi pohledy lze přepínat buď programově nebo pomocí přiřazení uživatelských rolí. Každý pohled je interně reprezentován jako XSLT transformace převádějící XML data formuláře do HTML pohledu na ně. Pohledy se samozřejmě nevytvářejí psaním XSLT transformací, ale umisťováním jednotlivých ovládacích prvků na plochu formuláře a nastavováním jejich vlastností. Škála prvků je velmi široká (viz obrázek 7.5) – od jednoduchých textových vstupů přes kalendáře až po opakující se tabulky nebo rekurzivní stromové struktury. Na tyto vstupní prvky potom mapujeme XML elementy nebo atributy z datových zdrojů. *InfoPath* dále nabízí podmíněné formátování prvků, uplatnění barevného schématu a řadu dalších komfortních možností. *InfoPath* též sdílí infrastrukturu s ostatními aplikacemi z rodiny *Microsoft Office,* takže v něm najdete stejné ovládací prvky a dialogy, kontrolu pravopisu a další obvyklé funkce. Pokud chcete vidět, kam se dá po stránce funkčnosti a vzhledu dojít, prostudujte si vzorové formuláře dodávané s aplikací, stojí rozhodně za to.



Obrázek 7.5: Vytváření uživatelského rozhraní z ovládacích prvků

Důležitou oblastí je práce s datovými zdroji. Datový zdroj není ničím jiným než XML dokumentem odpovídajícím XSD schématu. Každý formulář může obsahovat jeden nebo více datových zdrojů. Jeden z nich je hlavní, ostatní jsou pomocné – mohou obsahovat např. referenční data, jakými jsou seznam krajů, kategorie zboží v katalogu apod. Pomocné datové zdroje se často používají třeba pro vyplňování rozbalovacích seznamů *(drop-down list box)*. Hlavní datový zdroj lze načítat nebo ukládat do XML dokumentu stejným způsobem jako třeba dokumenty v aplikacích *Word* nebo *Excel,* čímž je umožněna práce v režimu odpojeném od sítě *(offline)*. Všechny datové zdroje mohou komunikovat s externími datovými zdroji pomocí datových spojení *(data connection)* s operacemi získání dat *(Receive)* a odeslání dat *(Submit)*. Data do datových zdrojů lze získávat z externího XML dokumentu, z databáze *Microsoft SQL* nebo *Access,* z výsledku volání webové služby nebo ze seznamů portálové technologie *SharePoint*. Odeslání dat je možné provést voláním webové služby, odesláním e-mailové zprávy nebo uložením do knihovny formulářů portálové technologie *SharePoint*.

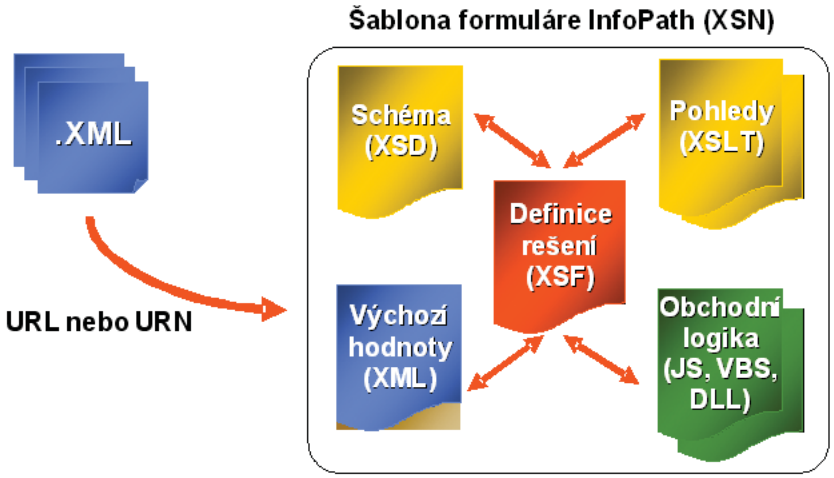


Obrázek 7.6: Příklad vyplňování formuláře aplikace InfoPath

Příklad formuláře vidíte na obrázku 7.6. Jde o formulář pro zadávání změn kontaktních údajů. Jako hlavní datový zdroj používá zpřístupněnou webovou službu našeho vzorového integračního řešení, do kterého se data odešlou pomocí tlačítka „Podat změny“. Jeho stisknutí zavolá webovou službu, předá jí potřebné údaje a zavře formulář. Na obrázku vidíme též některé zajímavé vlastnosti uživatelského rozhraní. Rodné číslo neodpovídá schématu (9 nebo 10 číslic), proto je orámováno červeně. Kategorie neodpovídá kontrolní logice formuláře, je proto rovněž označena červeně a je doplněna textem napovídajícím o původu chyby. Příjmení je povinná, leč nevyplněná položka, proto je u něj symbol červené hvězdičky. A konečně křestní jméno je červeně podtrženo, protože obsahuje překlep objevený kontrolou pravopisu zapnutou na toto pole.

Nasazení formulářů InfoPath

Nasazení řešení je minimálně stejně důležitou oblastí jako vlastní vývoj. Všechny součásti řešení jsou zahrnuty v jediném souboru s příponou XSN. Obsahuje datová schémata všech datových zdrojů (XSD), vytvořené pohledy (XSLT), výchozí hodnoty pro hlavní datový zdroj (XML), kód vytvořený během vývoje (skripty apod.) a definici řešení (nastavení vlastností řešení, datových spojení apod.) – viz obrázek 7.7.



Obrázek 7.7: Složky InfoPath řešení uložené v XSN souboru

Definiční XSN soubor může být buď nainstalován na lokální disk uživatelů nebo může být opublikován na libovolný webový server, kde je přístupný pomocí své URL adresy přes protokol HTTP. Druhá možnost bude zcela jistě výhodnější – umožňuje bezpečnou a snadnou instalaci řešení prostřednictvím hyperlinku, např. z intranetového serveru. Vlastní datový XML dokument uložený formulářem obsahuje instrukci pro zpracování *(processing instruction),* která ukazuje na adresu použitého řešení a jeho verzi:

```
<?xml version="1.0" encoding="UTF-8"?>
<?mso-infoPathSolution solutionVersion="1.0.0.9" ...
            href="http://localhost/formulare/KontaktniUdaje.xsn" ... ?>
<?mso-application progid="InfoPath.Document"?>
<ns1:KontaktniUdaje CasovaZnamka="2004-04-19T12:35:43"
xmlns:ns1="http://schemas.microsoft.cz/BrozuraBTS/2004/KontaktniUdaje">
      <Jmeno>
            ...
      </Telefony>
</ns1:KontaktniUdaje>
```

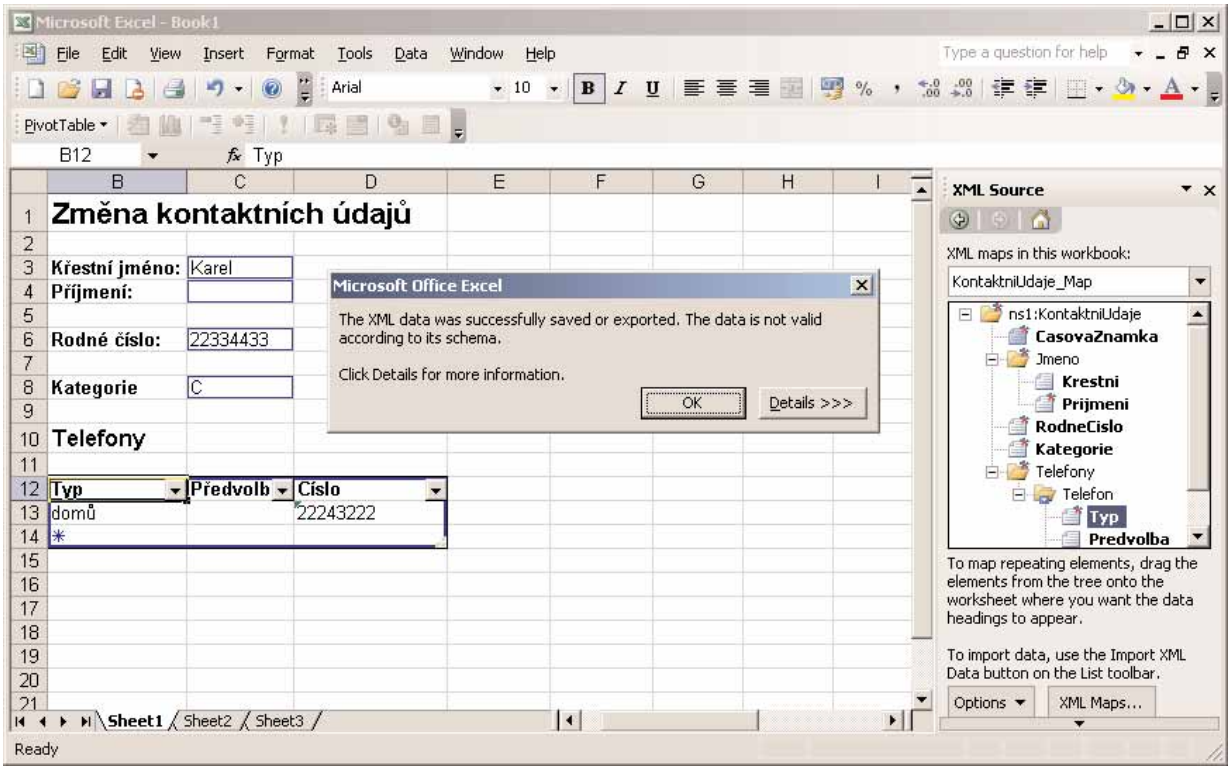
Pokud uživatel s nainstalovanou aplikací *InfoPath* obdrží tento dokument jako přílohu mailu, otevře ho ze sdílené složky nebo intranetového webu, *InfoPath* se připojí na adresu uvedenou v instrukci pro zpracování, stáhne z ní uvedené *InfoPath* řešení a otevře XML dokument s jeho pomocí, takže uživatel může bez jakékoliv instalace začít s daty pracovat. Řešení je navíc drženo v cache adresáři klienta, takže při příštím otevření XML dokumentu vytvořeného stejným řešením jej není nutné stahovat. Pokud je v XML dokumentu uvedena verze řešení vyšší než je verze již staženého řešení, dojde ke stažení nové verze a otevření dokumentu novou verzí – aktualizace řešení je tedy rovněž plně automatická.

Práce s XML a webovými službami v Excelu a Wordu

S XML dokumenty je možné pracovat též v aplikacích *Excel* a *Word* ve verzi 2003. Můžeme plně využívat předností těchto aplikací (např. *Excel* na výpočty vyúčtování výloh a *Word* na práci se smlouvami) a přitom se napojovat na datové zdroje zpřístupněné jako XML nebo webové služby.

Obě aplikace plně podporují XML. Dokumenty v nich vytvořené je možné ukládat buď v nativních formátech **.doc* a **.xls* nebo v podobě XML dokumentu. XML formáty pro plné ukládání dokumentů se nazývají *WordML* a *SpreadsheetML* a mají plně veřejná schémata i dokumentaci. S dokumenty těchto aplikací je tak možné pracovat

jako s jakýmikoliv jinými XML dokumenty, transformovat je, kontrolovat platnost, načítat do objektového modelu apod. Ještě o něco zajímavější možností je práce s uživatelsky definovanými XML schématy. Z obsahu dokumentů je pak možné „vytáhnout“ nebo do nich naopak doplnit údaje XML dokumentů přesně v požadovaném formátu. Na obrázku 7.8 je příklad práce s vlastním XML dokumentem v Excelu. Jednotlivé atributy a elementy schématu jsou mapovány na buňky sešitu pomocí XML map. Dokument je pak možné uložit, zkontrolovat jeho platnost oproti schématu (viz obrázek 7.8), předat integračnímu brokeru apod.



Obrázek 7.8: Práce s vlastním XML schématem v Excelu

V obou aplikacích je též možné používat webové služby. První možností je použití knihoven napsaných nad .NET frameworkem, který webové služby podporuje nativně. Druhou možností je použití toolkitu pro webové služby, který přináší možnost přidání reference na webovou službu i do vývoje pro *Office* v jazyce VBA *(Visual Basic for Applications)*. Nedělá přitom nic jiného než ostatní obdobné nástroje – generuje proxy třídy zastupující webové služby v jazyce VBA. Vývojáři tak mohou obvyklým způsobem používat funkce vygenerovaných objektů VBA místo webových služeb. Podíváme se také na portály?

7.3 Portálové technologie SharePoint

Přestože „portál“ je jedním ze slov zasažených nekontrolovanou inflací, nelze popřít, že portálová řešení rychle získávají na popularitě. Oproti tradičním webovým aplikacím nabízejí řadu výhod:

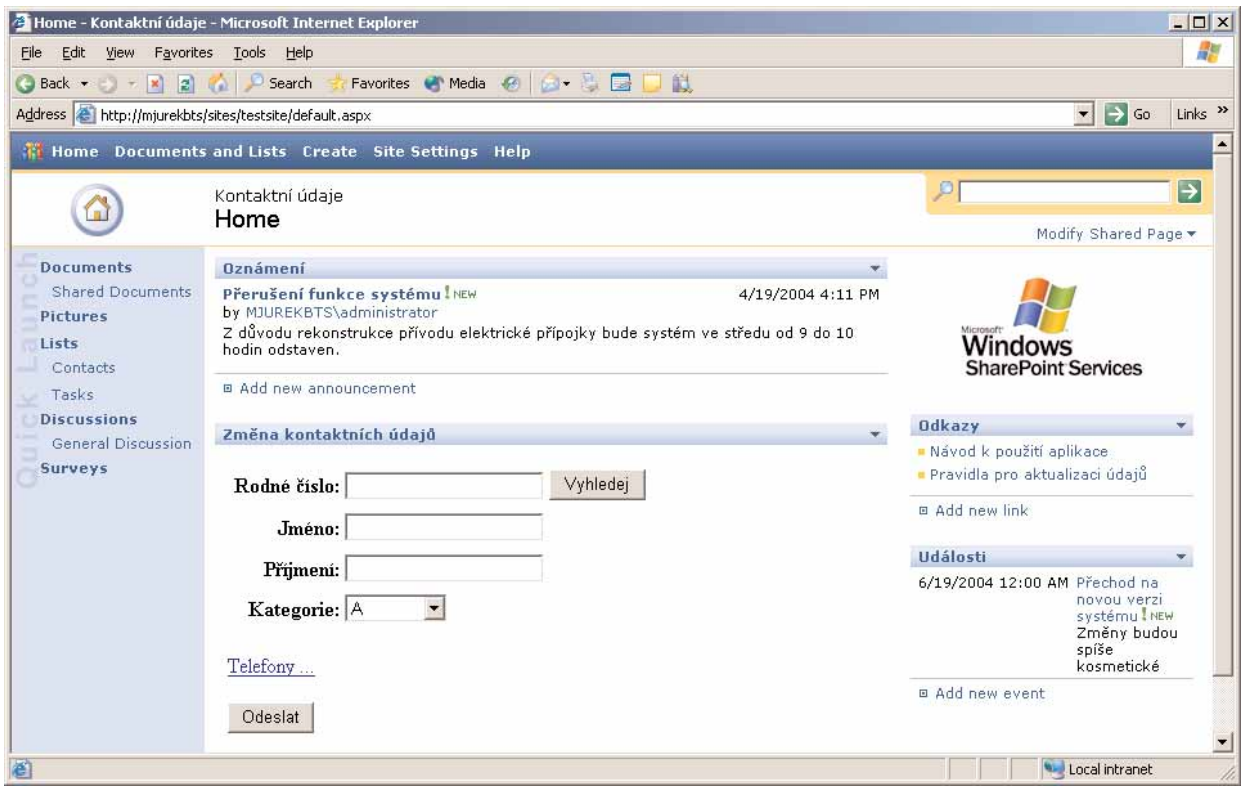
- Rozdělením stránek na jednotlivé webové dílce (*web part, portlet* apod.) dochází ke zřetelnému rozdělení jednotlivých projektů, čímž se snižují náklady na organizaci vývoje. Jednotlivá oddělení mohou pracovat nezávisle a dodávat webové dílce pro centrální řešení.
- Zvyšuje se znovuvyužitelnost kódu. Na portále můžete používat webové dílce dodané výrobcem portálu, vlastní webové dílce nebo webové dílce vyvinuté třetími stranami (např. dodavatelem ERP systému).
- Koncový uživatel je více zapojen do vytváření designu řešení a skládání jeho funkčnosti. Kromě větší zainteresovanosti a ztotožnění se s řešením je přínosem větší samosprávy i menší podíl práce připadající na vývojáře a administrátory.

- Sdílená infrastruktura – vývojáři webových dílců mohou vynechat řadu infrastrukturních úloh. Portál se za ně postará o autentizaci a autorizaci uživatele, personalizaci stránek portálu, jednotnou konfiguraci a její ukládání, jednotnost vizuálního rozhraní apod.
- Portály mají často implementovány zajímavé funkční bloky – jednoduchou správu obsahu *(content management)* a dokumentů *(document management),* deklarativní vytváření mini-aplikací typu „seznam vypůjčených knih“ apod.

BizTalk Server řeší „neviditelnou“ část integrace, proto jsou pro něj portálová řešení nabízející „vizuální“ integraci ideálním doplňkem. Na platformě Microsoft je k dispozici portálová technologie *Windows SharePoint Services* jakou součást operačního systému *Windows Server 2003*. Kompletním portálovým řešením včetně zprávy znalostí, fulltextového vyhledávání, definice cílových skupin a dalších funkcí je pak separátní produkt *SharePoint Portal Server 2003*. Podíváme se proto na některé jejich vlastnosti použitelné ve vazbě na integrační broker.

Integrace do uživatelského rozhraní

Jednou ze základních vlastností portálů je možnost skládání webových stránek z jednotlivých dílců. Na obrázku 7.9 je stránka sestavená z pěti webových dílců – Oznámení, Odkazy, Události, Změna kontaktních údajů a obrázek s logem, který je na rozdíl od čtyřech ostatních dílců bez modrého titulku a vlastního menu.

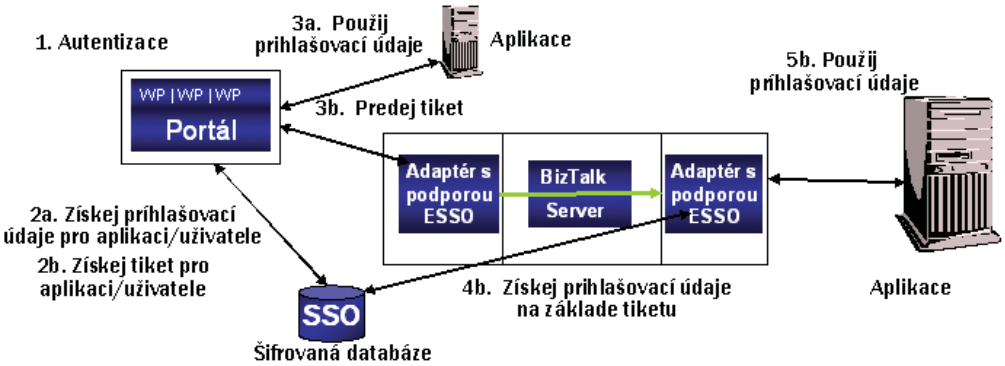


Obrázek 7.9: Vložení webového dílce s vazbou na integrační broker

Vytvoření dílce komunikujícího s integračním brokerem je snadné. Pro komunikaci s brokerem je možné použít libovolnou z technologií popsaných v kapitole 7.1. V konkrétním případě na obrázku 7.9 jsou údaje z formuláře vyplněného ve webovém dílci odeslány ke zpracování integračním brokerem prostřednictvím volání webové služby.

Enterprise Single Sign-On

V kapitole 3.3 jsme se zmínili o funkci *Enterprise Single Sign-On (ESSO),* nabízející jednotné přihlášení uživatele do celého integračního řešení, když sekundární přihlašovací údaje do integrovaných systémů a aplikací jsou uloženy v zašifrované databázi přihlašovacích údajů. Integrační broker i portál používají stejnou infrastrukturu *ESSO*. Portál může k integrovaným systémům přistupovat dvojím způsobem – buď přímo nebo prostřednictvím integračního brokeru (viz obrázek 7.10).



Obrázek 7.10: ESSO použitý v kombinaci portálu a integračního brokeru

První krok (1) v použití *ESSO* je vždy stejný. Uživatel přistoupí na portál a autentizuje se svými primárními přihlašovacími údaji. Při použití prohlížeče *Internet Explorer* je možné využít autentizace integrované s bezpečnostními mechanismy *Windows,* takže jediným okamžikem zadávání přihlašovacích údajů může být prvotní přihlášení při příchodu k počítači. Druhým krokem je kontaktování služby *ESSO* – v případě přímé komunikace (2a) získá portál přímo přihlašovací údaje k aplikaci, v případě komunikace prostřednictvím brokeru (2b) získá tiket pro pozdější získání těchto údajů. Při prvním způsobu jsou přihlašovací údaje použity pro komunikaci s aplikací (3a), při druhém způsobu je tiket předán integračnímu brokeru (3b), který jej uloží do kontextu dále zpracovávané zprávy. Na konci zpracování získá odesílající adaptér přihlašovací údaje od služby *ESSO* na základě tiketu (4b) a použije je pro komunikaci s příslušnou aplikací (5b). Celý proces je z hlediska koncového uživatele zcela transparentní, což je jeho největším přínosem. Z hlediska bezpečnosti je důležité, že přihlašovací údaje do aplikací nejsou nikdy nikde ukládány v otevřeném tvaru.

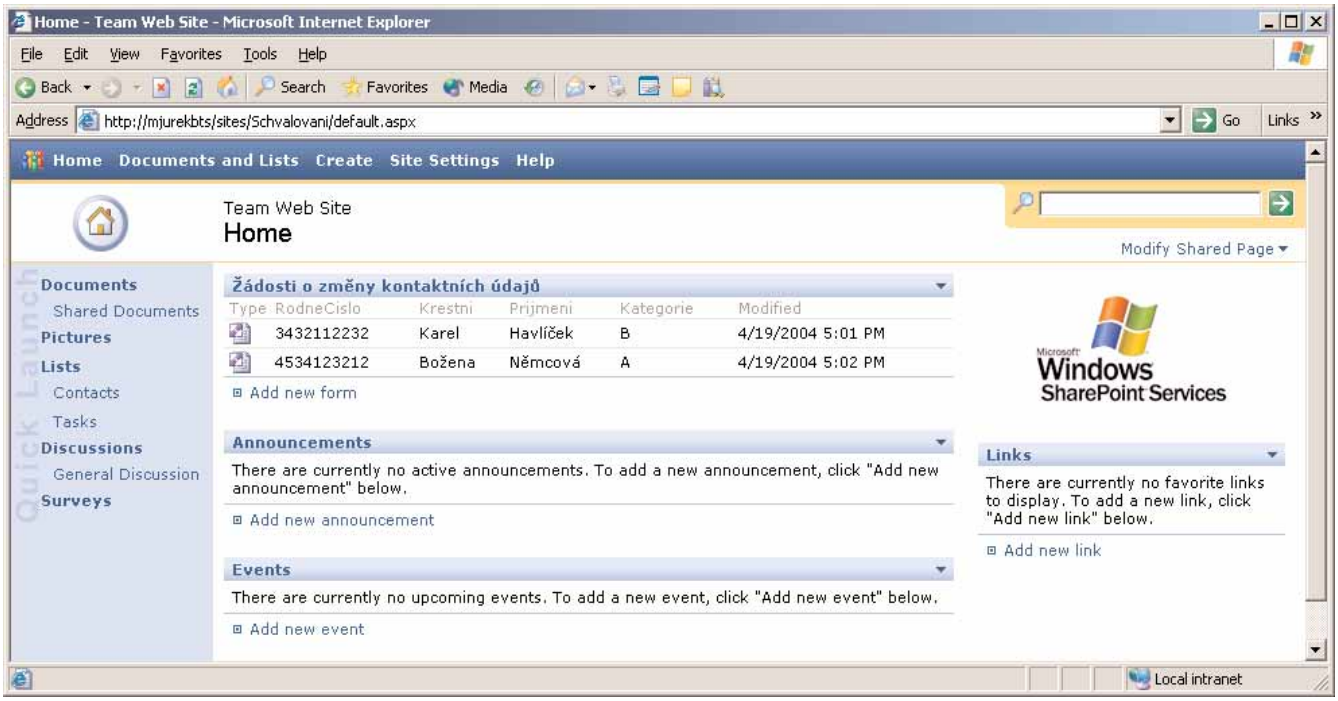
Adaptér pro knihovny SharePoint

SharePoint nabízí funkci knihoven dokumentů. Jsou to virtuální složky souborů doplněných metadaty uložené v relační databázi portálu. Ukládány do nich mohou být jakékoliv soubory. Velmi často jsou to textové dokumenty, sešity tabulkového procesoru nebo prezentace. Speciální variantou využití jsou knihovny XML dokumentů s definovaným *InfoPath* formulářem *(forms library),* tyto knihovny formulářů potom fungují jako určité elektronické šanony s formuláři stejného typu. *SharePoint* umožňuje kompletní programový přístup k nim pomocí webových služeb, proto se nabízí možnost propojit tyto knihovny formulářů s integračním brokerem. Pro snížení pracnosti postupu existuje zajímavé řešení *BizTalk Server Adapter for SharePoint Libraries,* který je včetně zdrojového kódu k dispozici na komunitním webu *GotDotNet* (odkaz ke stažení naleznete v praktických cvičeních).

Pokud adaptér použijeme pro přijetí zprávy, kontroluje v pravidelných intervalech obsah knihovny formulářů. Pokud v nich najde soubory, načte je a jednotlivě předá integračnímu brokeru do databáze *message box*. Poté je z knihovny formulářů buď vymaže nebo přesune do jiné knihovny určené k archivaci přenesených formulářů. Volitelně je možné omezit přijímané dokumenty použitím filtrovaného pohledu definovaného na straně portálu.

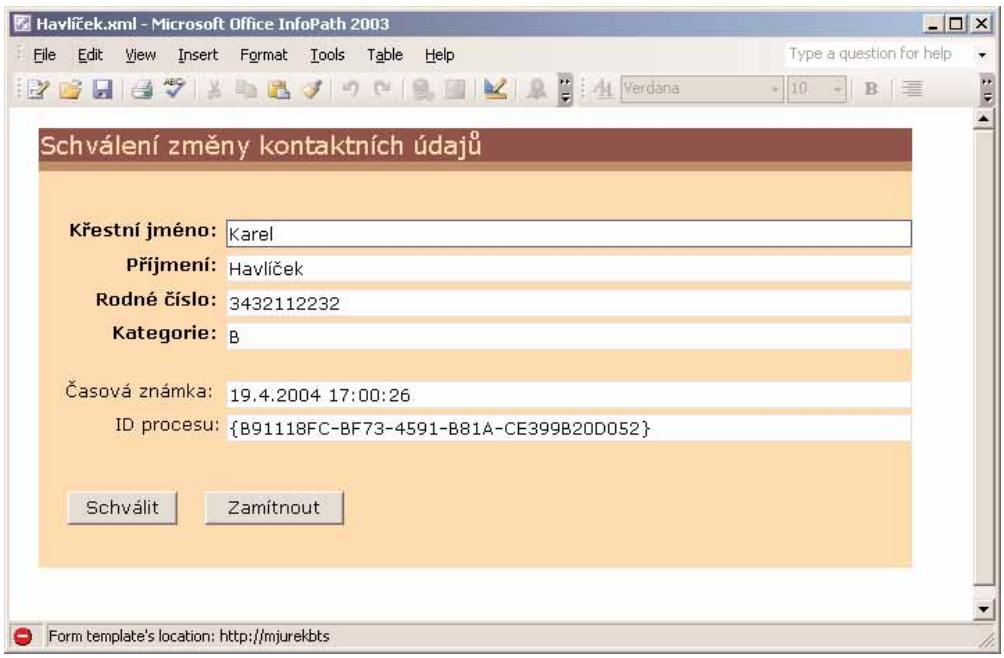
Při odesílání zpráv adaptér uloží příslušný XML dokument do nastavené knihovny formulářů. Přitom podle nastavených pravidel může odvodit jeho jméno a další vlastnosti pro uložení na portálu.

Podívejme se na možnost použití této technologie pro schvalovací proces z příkladu v kapitole 5.3. Zpráva *msgKontaktniUdajeKeSchvaleni* může být přímo z orchestrace odeslána do knihovny formulářů aplikace *SharePoint,* kde čeká na manuální schválení zodpovědným uživatelem (viz obrázek 7.11).



Obrázek 7.11: Dokumenty doručené brokerem do knihovny SharePoint

Při poklepání na formulář se spustí formulářová aplikace *InfoPath* s vyplněnými údaji z XML dokumentu. Koncový uživatel má k dispozici tlačítka pro schválení nebo zamítnutí formuláře (viz obrázek 7.12). Stisknutí na tlačítko způsobí nastavení hodnoty XML elementu vyjadřujícího schválení v příslušném XML dokumentu, jeho uložení zpět do knihovny formulářů na webu a uzavření aplikace *InfoPath*.



Obrázek 7.12: Schválení žádosti o změnu v aplikaci InfoPath

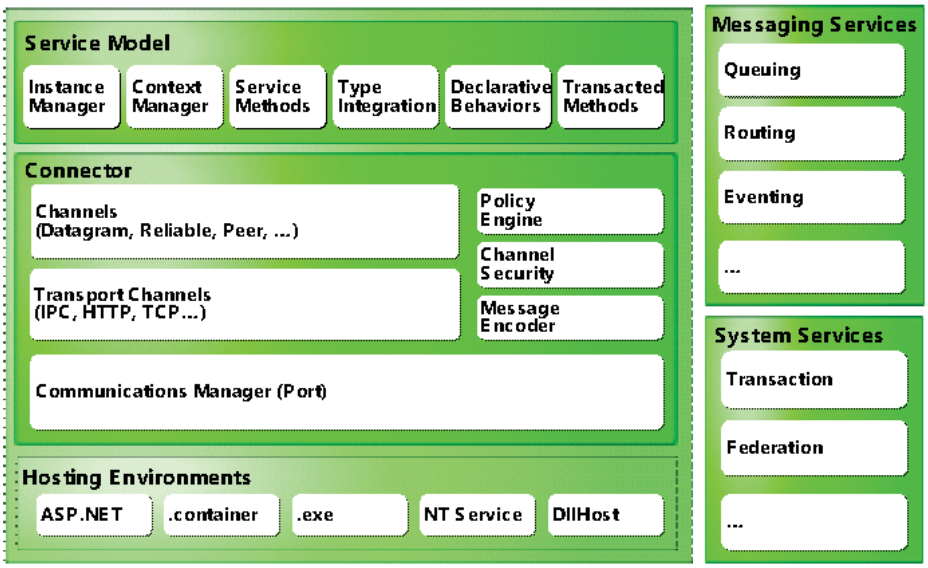
V knihovně formulářů definujeme pohled, který filtruje pouze rozhodnuté žádosti (s nastavenou hodnotou rozhodnutí). V integračním brokeru nastavíme port pro přijetí, který monitoruje dokumenty v novém pohledu na naši formulářovou knihovnu. Tím máme zajištěno přijetí dokumentu zpět integračním brokerem pro další zpracování. Zároveň je dokument vymazán z knihovny dokumentů čekajících na rozhodnutí a místo toho vytvořena jeho kopie v archivní knihovně již rozhodnutých žádostí. Dokument doručený integračnímu brokeru je snadno a automaticky korelován s instancí procesu pomocí jednoznačného identifikátoru. Snadné, rychlé, elegantní a uživatelsky přívětivé – co více bychom si mohli přát?

Kapitola 8:
Místo závěru

V úvodu jsem se pustil do malé disputace o smyslu prvních stran v knihách. Nyní jsme na straně poslední, nepočítám-li přílohy. Těší mne, že jste dočetli až sem na poslední stranu a doufám, že jste se něco dozvěděli a naučili. Shrnující závěry v knihách považuji za zcela zbytečné, proto se do jeho psaní nechci pouštět. Na druhou stranu brožura musí mít nějaký konec, proto jsem se jako třešničku na dort rozhodl zařadit poněkud spekulativní kapitolu o předpokládaném budoucím vývoji.

První očekávanou věcí je standardizace pokročilých webových protokolů, zmíněných v kapitole 2.3. V době psaní této brožury se standard *WS-Security* dostal do fáze doporučení agentury OASIS, organizace WS-I pracuje na novém profilu *Basic Security Profile,* výrobci mají téměř hotové podporované implementace tohoto standardu. Stejný vývoj se dá v dalších měsících očekávat i u dalších standardů, z nichž nejočekávanější jsou *WS-ReliableMessaging* a *WS-Transaction*.

Tyto protokoly budou zahrnuty do infrastruktury operačních systémů a aplikačních serverů. Jejich používání je dnes doménou vývojářů, v budoucnu se stanou samozřejmou infrastrukturu. Podobně jako se dnes vývojáři nemusí starat o souborový systém nebo *TCP/IP* protokol, nebudou se muset starat ani o pokročilé protokoly webových služeb. Velcí výrobci aplikačních serverů (BEA, IBM, Microsoft) zahrnou tyto protokoly do svých produktů. Jejich používání nebude záležitostí vývoje, ale konfigurace aplikací. Microsoft připravuje podporu protokolů *WS-Security, WS-Transaction* a *WS-ReliableMessaging* v komunikačním modulu s kódovým názvem *Indigo* (viz obrázek 8.1). Bude součástí příštího operačního systému s kódovým názvem *Longhorn* a rovněž bude ke stažení pro některé starší verze operačních systémů. Indigo nenahrazuje integrační broker, je vůči němu ve funkci dalšího adaptéru.



Obrázek 8.1: Architektura komunikačního modulu Indigo

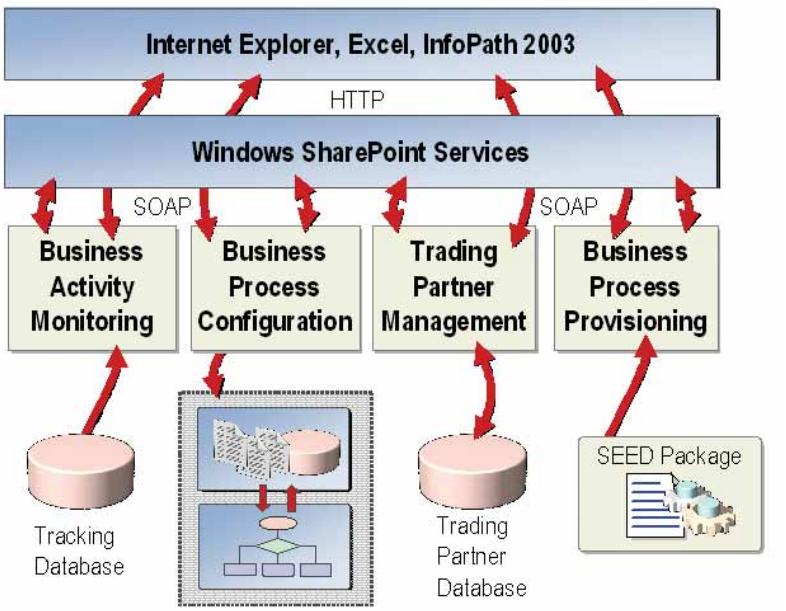
Vyvíjet se bude samozřejmě i BizTalk Server. V krátké době by měl být zdarma k dispozici adaptér pro *MQSeries* a nový SOAP adaptér s podporou *WS-Security,* k nim se přidá komerční adaptér pro systémy SAP. Rozšiřovat se bude i řada adaptérů dodávaných třetími stranami. Zhruba v polovině roku 2005 by se měla objevit nová verze BizTalk Serveru. Podle předběžných informací by měla zajistit kompatibilitu s novými serverovými produkty (zejména Microsoft SQL Server 2005 „Yukon“), přinést řadu vylepšení a nové nástroje v oblasti administrace produktu a přidat nové možnosti pro koncové uživatele *(Business Analytics Monitoring, Business Activity Services)*.

Chtělo by to ještě nějakou chytrou větu na úplný závěr. Bude vadit, když žádná nebude?

Příloha A:

Správa obchodních aktivit

BizTalk Server obsahuje řadu podsystémů pro správu obchodních aktivit (*business activity services,* viz obrázek A.1). Zatímco analytické monitorování je univerzálně použitelné a byla mu věnována celá šestá kapitola, ostatní podsystémy jsou vhodné pouze v situacích, kdy musí integrační broker komunikovat s větším množstvím podobajících se subjektů. Tato situace typicky nastává v B2B *(business-to-business)* scénářích, kdy např. výrobce nebo distributor musí komunikovat s větším množstvím dodavatelů a zpřístupňují jim k tomu standardní rozhraní. Některé vlastnosti lze v omezené míře využít i v interních pobočkových scénářích, kdy centrální integrační broker komunikuje s řadou identických systémů v jednotlivých pobočkách.

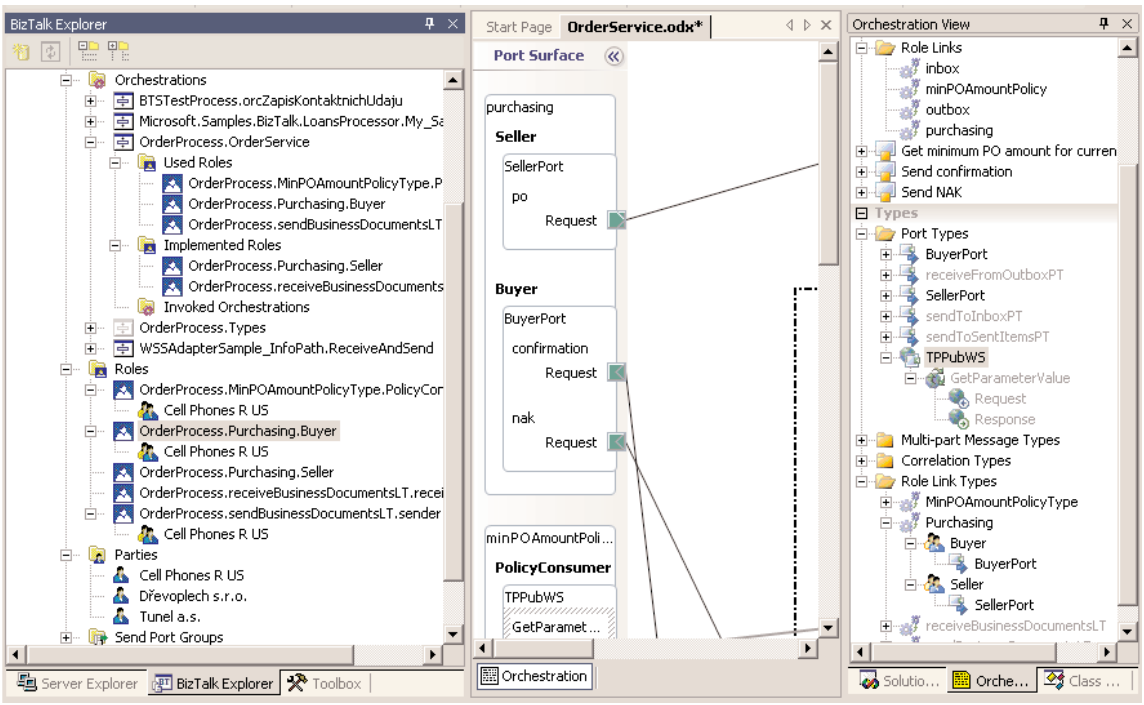


Obrázek A.1: Přehled podsystémů pro správu obchodních aktivit

Partneři a propojení rolí

Jak jsme se již zmínili v kapitole 3.3, umožňuje BizTalk Server definici partnerů (*party,* viz též obrázek 3.5). Partnerem samozřejmě nemusí být pouze obchodní partner, může jím být třeba interní aplikace spravovaná nezávislou skupinou v rámci IT oddělení. Partner je určen během *pipeline* zpracování na portu pro přijetí ve fázi *Resolve Party*. Standardně dodávaná komponenta *PartyResolution* umožňuje rozlišení partnera buď podle účtu, pod kterým se odesilatel zprávy autentizoval nebo podle certifikátu, jímž byla zpráva digitálně podepsána. Můžete si samozřejmě napsat i vlastní komponentu pro rozlišení partnera. Například je možné mít u každého partnera uloženou jeho IP adresu jako jeden z identifikátoru a ve vlastní komponentě pro rozlišení partnera nastavit partnera (*PID* neboli *PartyID*) v závislosti na IP adrese, ze které byla zpráva odeslána.

Význam definice partnerů vynikne v kombinaci s tzv. rolemi *(roles)*. Role jsou skupiny portů, které musí partner implementovat, aby mohl komunikovat s vaší orchestrací. Např. na obrázku A.2 je definován typ propojení rolí *(role link type)* nazvaný *purchasing,* která obsahuje dvě role – prodávajícího *(Seller)* a kupujícího *(Buyer)*. Role prodávajícího obsahuje port pro přijetí objednávky *(po),* role kupujícího obsahuje port s operacemi pro přijetí potvrzení *(confirmation)* nebo zamítnutí *(nak)*. V orchestraci pak můžeme použít propojení rolí *(Role Link)* a jednotlivé porty v něm obsažené propojit s akcemi pro přijetí nebo odeslání v dané orchestraci jako na obrázku A.2.



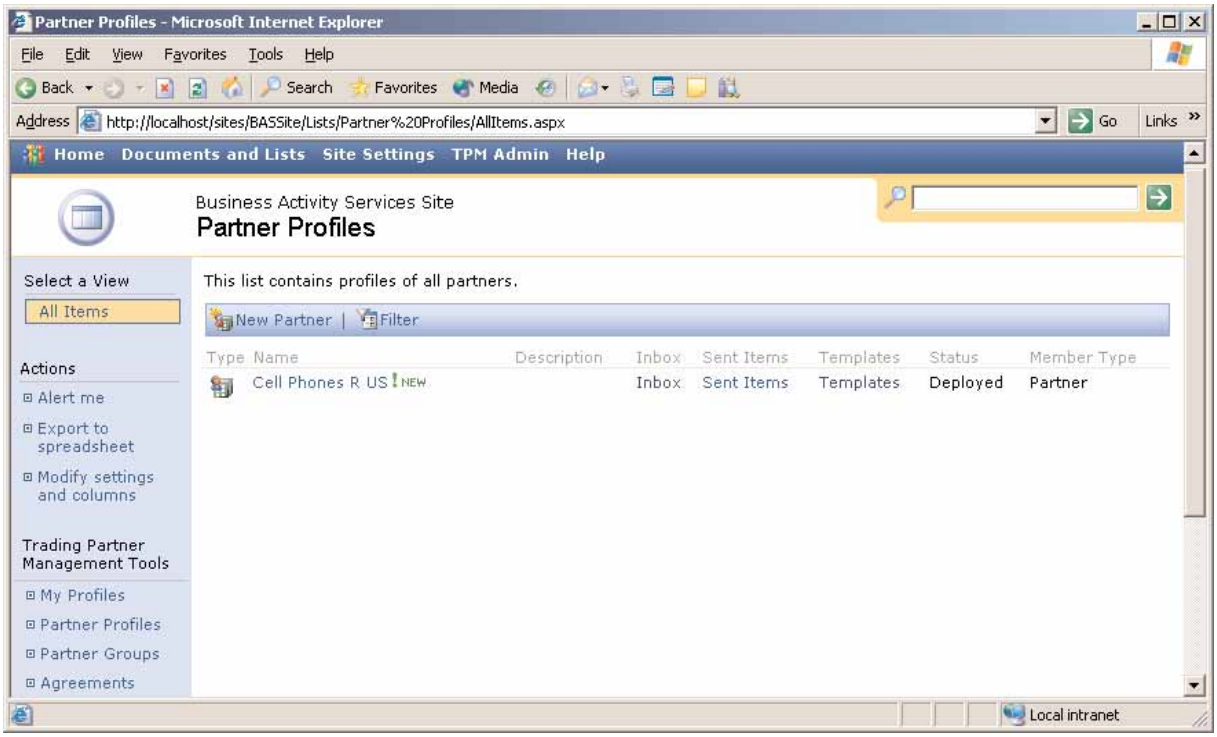
Obrázek A.2: Definice rolí, jejich typů a propojení

V nástroji *BizTalk Explorer* (viz obrázek A.2) je pro každou orchestraci znázorněno, jaké role implementuje *(Implemented Roles)* a jaké role partnerů používá pro komunikaci *(Used Roles)*. Pokud chceme do konfigurace zadat, že určitý partner implementuje některou z rolí, musíme provést tři postupné kroky. Prvním je definice portů pro odeslání vedoucích k danému partnerovi. Druhým je přiřazení těchto portů partnerovi ve vlastnostech partnera. Třetím je přiřazení partnera určité roli, což obnáší navázání přiřazených portů partnera na jednotlivé porty v roli (viz obrázek A.2, kde partner *Cell Phones R US* implementuje roli kupujícího – *buyer*).

Dalo by se tedy říci, že role fungují jako určitá výhybka při komunikaci s více subjekty. Místo aby orchestrace odeslala zprávu na konkrétní port určitého partnera, odešle ho na port v propojení rolí. BizTalk podle údajů v konfigurační databázi identifikuje partnera, nalezne porty, pomocí kterých partner svoji roli implementuje, a odešle zprávu na příslušné porty. Mimochodem, typ propojení rolí plně odpovídá specifikaci *BPEL4WS,* ve které se koncept sloužící k definování a reprezentaci obchodních vztahů označuje jako *service link*.

Správa obchodních partnerů

Počet partnerů, se kterými integrační broker komunikuje, může být velmi vysoký. Automobilka Ford, jedna z vlajkových implementací BizTalk Serveru, má tisíce obchodních partnerů, kteří musí s automobilkou chtě-nechtě komunikovat prostřednictvím jednotného rozhraní. S rostoucím počtem partnerů roste i množství nutné konfigurační dřiny.
Je žádoucí přenést tuto nezáživnou a nikdy nekončící práci z IT oddělení pod křídla jiných oddělení vlastnících implementovaný proces po organizační stránce. Přesně k tomuto účelu slouží subsystém pro správu obchodních partnerů *(Trading Partner Management, TPM),* který udržuje databázi obchodních partnerů pomocí webové portálové technologie SharePoint a formulářové aplikace *InfoPath*. Zodpovědní koncoví uživatelé tak mohou snadno pomocí formulářů vytvářet profily partnerů (viz obrázek A.3). Profily poté slouží k vytvoření příslušného partnera v konfigurační databázi BizTalku, a to včetně instalace a nastavení certifikátů definovaných v profilu partnera.

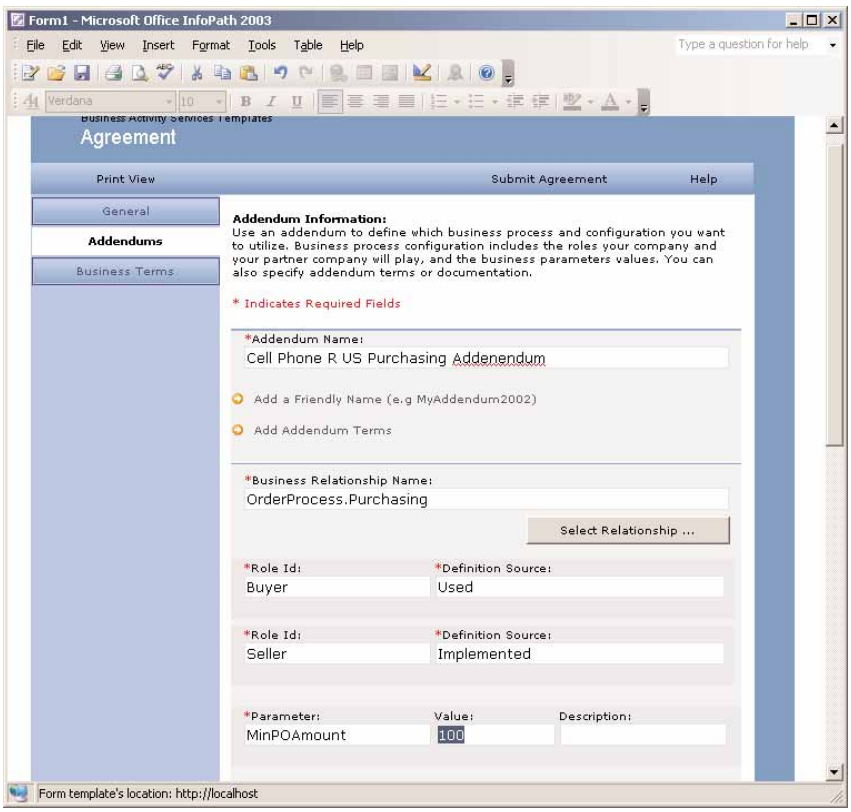


Obrázek A.3: Definice obchodních partnerů pro koncové uživatele

Pro tyto partnery poté zadávají dohody *(agreements)*. Dohoda vyjadřuje vztah mezi námi a partnerem pro určitý implementovaný proces. Obsahuje údaje z profilu partnera – kontaktní údaje, technické informace o protokolu a adaptéru apod. Dále definuje, která strana hraje v této dohodě jakou z implementovaných rolí (prodávající, nakupující atd.). Všechny údaje jsou uloženy v databázi obchodních partnerů.

Konfigurace obchodního procesu

Od koncových uživatelů samozřejmě není možné očekávat schopnosti potřebné pro definici orchestrace. Stejně tak nelze od IT oddělení očekávat neustálou konfiguraci systému na základě stále se měnících podmínek. Není ale nerealistické předpokládat, že koncoví uživatelé budou schopni nastavovat parametry orchestrace. Dobrým příkladem může být prodej zboží – každý partner může mít nastavenou jinou maximální dobu splatnosti faktury, s tímto údajem samozřejmě potřebujeme v orchestraci pracovat. Konfigurace obchodního procesu *(Business Process Configuration)* slouží právě k tomuto účelu – dává koncovým uživatelům do ruky nástroj pro nastavování parametrů jednotlivých partnerů, které pak mohou být využity v orchestraci. Nastavování parametrů se děje pomocí dodatků ke smlouvám *(addendum)*. Princip je ilustrován na obrázku A.4, kde je jako parametr daného partnera zadána minimální přípustná hodnota objednávky.

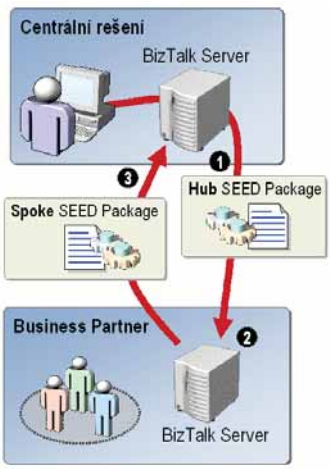


Obrázek A.4: Zadávání parametrů v příloze smlouvy s partnerem

Pokud dohoda s partnerem zahrnuje více orchestrací BizTalk Serveru, vytváří se pro každou z orchestrací nový dodatek. Podsystém pro konfiguraci obchodního procesu je po technické stránce součástí podsystému pro správu obchodních partnerů *(Trading Partner Management),* od kterého jej lze jenom těžko oddělit. V dokumentaci a dalších materiálech bývá uveden odděleně, neboť se předpokládá, že v dalších verzích produktu bude tento systém výrazněji rozšířen a ostřeji oddělen.

Poskytování obchodního procesu

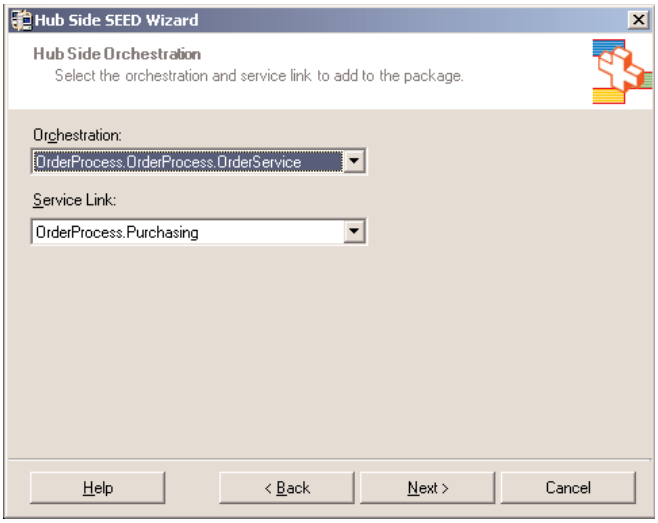
Pokud si organizace nasadí BizTalk Server, ke kterému nechá připojovat svoje partnery, zpravidla se jich neptá, zda se jim toto řešení líbí. Jde většinou o vztah velkého a silného vůči menším a slabším, kde partner buď může vaše centrální řešení akceptovat nebo má zkrátka smůlu. Přestože jde o řešení jednostranně nadiktované, nemůže se centrální organizace dost dobře zbavit práce spojené s připojováním partnerů, konfigurací a testováním tohoto připojení. Ke snížení množství vložené práce a tedy i nákladů slouží funkce poskytování obchodního procesu *(Business Process Provisioning)*. Můžeme si ho představit jako průvodce pro připojení partnera v situaci, kdy partner má rovněž nasazený BizTalk Server (pro tyto účely dokonce existuje jeho velmi levná edice Partner). Celá technologie se nazývá *SEED* (viz obrázek A.5) a její použití sestává ze tří kroků.



Obrázek A.5: Postup připojení partnera pomocí SEED technologie

V prvním kroku se v centrálním místě *(hub)* vytvoří konfigurační balíček *(Hub SEED Package)* K jeho vytvoření slouží průvodce (viz obrázek A.6). Balíček může obsahovat celou řadu ingrediencí:

- Projekt Visual Studia.NET sloužící jako šablona pro vytvoření orchestrace na straně připojovaného partnera
- Zkompilované **.dll* knihovny obsahující orchestrace, *pipeline* komponenty, schémata, mapy, třídy objektů apod.
- Informace o portech pro přijetí na straně centrálního místa – partner s jejich pomocí může komunikovat s centrálním místem pomocí *HTTP* nebo *SOAP* adaptéru
- Testovací zprávy a URL adresa pro jejich
- Soubory pro analytické monitorování *(BAM)* – šablony v Excelu, vyexportované XML definice, sledovací profily navazující aktivity na orchestraci
- Definice *InfoPath* formulářů pro vyměňované zprávy
- Soubory pro správu partnerů – vlastní profil hubu a šablona dohody pro skupinu partnerů, do níž připojovaný partner patří
- Certifikát(y) hubu pro podepisování, šifrování a SSL přístup (samozřejmě bez privátních klíčů)
- Další soubory využívané případnými vlastními rozšířeními technologie SEED



Obrázek A.6: Vytvoření konfiguračního balíčku v centrálním místě

Ve druhém kroku je balíček partnerem nainstalován a otestován v připojujícím se místě *(spoke).* Při instalaci doručeného balíčku dochází k nakonfigurování partnerského BizTalk Serveru. Zpravidla bývá nutné doimplementovat některé části řešení, např. napojit orchestraci na interní systémy partnera. Nakonec se testuje funkčnost komunikace a celého řešení. Po úspěšném otestování je vygenerován další balíček *(Spoke SEED Package),* obsahující informace o konečné konfiguraci partnerského řešení. Může obsahovat následující součásti:

- Informace o portech pro přijetí na straně partnera
- Profil partnera pro systém správy obchodních partnerů
- Výsledky provedeného testování
- Certifikát(y) partnera sloužící pro podepisování, šifrování a SSL přístup (samozřejmě bez privátních klíčů)
- Další soubory, využívané případnými vlastními rozšířeními technologie SEED

Třetím krokem je instalace balíčku doručeného partnerem v centrálním místě. Dochází ke zpracování a nakonfigurování BizTalku v centrálním místě podle doručených informací v balíčku, zejména vytvoření partnera a příslušných portů pro odeslání zrcadlících partnerovy porty pro přijetí.

Příloha B:

Praktická cvičení

Celou brožuru jsem sliboval, že v příloze B najdete praktické příklady. Ne, dnes není apríl (pravděpodobnost 99.73%). Otázka za 1000 Kč: Praktické příklady zde nenajdete, protože

A. Brožura je bez nich levnější, protože má méně stran

B. Je třeba šetřit naše lesy

C. Nestihl jsem příklady dokončit v termínu

D. Příklady by stejně vyžadovaly stahování souborů z Internetu

Ve skutečnosti jsou všechny výše uvedené odpovědi správné. Na adrese *http://msdn.microsoft.cz/docs/BrozuraIntegrace.zip* najdete kompletní příklady ke stažení, které vás krok za krokem provedou všemi scénáři popisovanými v této knize. Konkrétně se můžete těšit na následující menu:

Cvičení 1 – Instalace

Instalace potřebného softwaru

Instalace BizTalk Serveru 2004

Instalace pomocných projektů pro praktická cvičení

Cvičení 2 – Komunikace a přenos zpráv

Vytvoření a konfigurace portu pro přijetí

Vytvoření a konfigurace portu pro odeslání

Implementace scénáře *pass-through*

Pozorování mechanismu *Retry*

Detekce nedoručených zpráv

Sledování zpráv prošlých systémem

Souhrnné statistiky prošlých zpráv

Cvičení 3 – Transformace a routování zpráv

Vytvoření schématu zprávy

Vytvoření transformační mapy

Testování transformace

Vytvoření vlastních pipeline komponent

Definování význačných vlastností

Implementace kompletního scénáře *Publish/Subscribe*

Vyhledávání zpráv podle význačných vlastností

Cvičení 4 – Správa procesů

Definování orchestrace z kapitoly 5.3

Opublikování orchestrace jako webové služby

Vytvoření vstupního InfoPath formuláře

Monitorování běžících instancí orchestrace

Ladění orchestrace

Souhrnné statistiky procesů

Cvičení 5 – Oddělení obchodních pravidel

Definice slovníku faktů

Definice politiky a pravidel

Otestování pravidel

Nasazení pravidel

Volání pravidel z orchestrace

Cvičení 6 – Analytické monitorování

Definování sbíraných údajů (aktivita)

Definice pohledů

Vytvoření databázové infrastruktury

Provázání aktivity s orchestrací

Monitorování sběru dat

Zpracování nasbíraných dat

Analýza dat v MS Excel

Sledování dat na *SharePoint* webu

Cvičení 7 – Interakce s uživatelem

Dokončení vstupního *InfoPath* formuláře

Publikace formuláře na *SharePoint* portál

Instalace adaptéru pro knihovny *SharePoint*

Implementace schvalovacího procesu na *SharePoint* portálu

Názvy produktů a společností uvedené v této brožuře mohou být obchodními značkami jejich vlastníků.
Texty neprošly jazykovou korekturou.

Vydal: Microsoft, s.r.o., Vyskočilova 1461/2a, 140 00 Praha 4, tel.: +420 261 197 111, fax: +420 260 197 100
http://www.microsoft.com/cze, http://www.microsoft.com/slovakia