ARCHITEKTURA ORIENTOVANÁ NA SLUŽBY

Štěpán Húsek
IT Architekt, SOA; IBM, Stepan_Husek@cz.ibm.com
Tomáš Mayer
IT Specialist; IBM, Tomas_Mayer@cz.ibm.com

ABSTRAKT
Článek shrnuje základní principy, specifikace, techniky a nástroje pro zavedení servisně orientované architektury (SOA). Jeho účelem je shrnout současné trendy a technologie, které v současné době SOA podporují. Je určen pro všechny, kdo se chtějí přehledově seznámit s tímto moderním architektonickým stylem.

KLÍČOVÁ SLOVA
Service Oriented Architecture, Enterprise Service Bus, Web Services, BPEL, Service Component Architecture, Service Data Objects

Historie

Architektura zaměřená na služby má své kořeny v dávné minulosti a je dalším paradigmatem ve vývoji software. Prvním takovým krokem bylo strukturované programování, které přináší procedury a funkce, které slibují přehlednější kód a znuvupoužitelnost kódu. Strukturované programování však zdaleka neposkytlo takovou znuvupoužitelnost kódu, jaká byla požadována. Na základě toho se programovací jazyky vyvíjely a přišlo objektové programování. Objektové programování přináší nový pojem „objekty“, které reprezentují samostatné jednotky obsahující atributy a poskytují operace. Dalším přínosem je dědičnost, kde je možné zdědit určité atributy či operace. Znuvupoužitelnost kódu se zavedením objektů a dědičnosti mnohem zvýšila. Rozdělením aplikace na objekty dostáváme velké množství objektů, které jsou ve velké míře závislé jeden na druhém – těsně vázané a poskytují funkcionalitu na velmi nízké úrovni.
Dalším stupněm ve vývoji programovacích technik a architektury aplikací je rozdělení aplikace na komponenty. Komponenta je část aplikace, která poskytuje předem definovanou funkcionalitu. Výsledkem rozdělení aplikace na komponenty jsou vícevrstvé aplikace. Aplikace složené z komponent splňují nároky na znuvupoužitelnost kódu nicméně postupem času se ukazuje, že existuje mnoho aplikací psaných v různých jazycích běžících na různých platformách a dokonce i v rámci různých organizací. V rámci všeobecné globalizace je tlak ze strany businessu vytvářet nové aplikace poskytující funkcionalitu, kterou již nějaké aplikace v organizaci či mimo organizaci poskytují, ale je napsána v jiném jazyku a běží na jiné platformě. Přepsání aplikací tak, aby splnili veškeré funkcionality a aby běžely na stejné platformě jako nově vyvíjená aplikace by znamenalo velké množství nákladů a času. A právě čas je klíčový faktor, který žene architekturu systému rychle dopředu a tak vzniká architektura zaměřená na služby, která umožňuje využívat komponenty nezávisle na jejich implementačním jazyku a nezávisle na systému, na kterém běží.

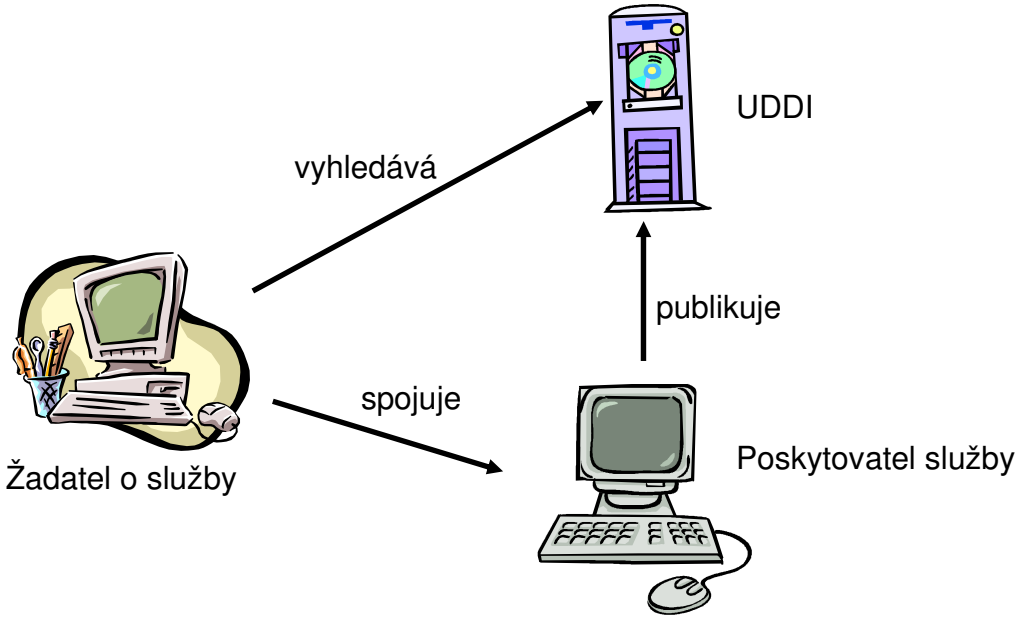
Hlavní principy Architektury orientované na služby

Architektura orientována na služby přináší nový architektonický styl při návrhu a vývoji aplikací. Aplikace se skládají z nezávislých bloků (služeb). Služba je komponenta, která má přesně definované rozhraní a toto rozhraní určuje funkcionalitu, kterou poskytuje. Služby jsou bezestavové a jejich rozhraní je popsané pomocí standardizovaného jazyka (WSDL) a komunikují pomocí standardního komunikačního protokolu (SOAP) po transportním kanálu uvedeném v rozhraní. WSDL a SOAP jsou jedním ze základním specifikací webových služeb - WebServices. Nejčastějším transportním kanálem pro webové služby bývá HTTP nebo HTTPS.
Společnost IBM záměrně nespojuje architekturu zaměřenou na služby (SOA) s webovými službami. Webové služby nejlépe naplňují podstatu architektury orientované na služby, nicméně nepředstavují jediný prostředek k její realizaci. IBM je významným dodavatelem softwarové infrastruktury v podobě aplikačních serverů a v této oblasti podporuje architekturu zaměřenou na služby bez toho, aby vývojář musel definovat webové služby, ale obecně služby, které jsou bezestavové, samopopisné a samoobsažné. Komunikačním protokolem tedy nemusí být SOAP a transportním protokolem nemusí být HTTP či HTTPS. Ve světě J2EE je možné definovat službu jako bezestavovou komponentu (tzv. stateless session bean), která komunikuje pomocí RMI/IIOP, ve světě standardních Java aplikací může služba být reprezentována pomocí tzv. Plain Old Java Objectu (POJO).
Pokud však chceme vytvořit služby, které budeme moci využívat na jiné platformě, je použití webových služeb nezbytně nutné a výše uvedené objekty je možné velice rychle zpřístupnit pomoci rozhraní webových služeb.

Základní operace v architektuře SOA

V úvodu již bylo definováno co je to služba a čím je charakterizována. Nyní se podíváme na to, jak se službami komunikuje a jaké základní typy komunikace se používají.
Máme tedy dva základní druhy služeb a to producenty zpráv a konzumenty zpráv. **Producent** zprávy zprávu vytváří a **konzument** zprávu naopak zpracovává. Zprávou se rozumí data[1], které si producent a konzument vyměňují. Přenášená data jsou XML textová data popsána pomocí XML schématu. To však neplatí pro komunikaci pomocí protokolu RMI/IIOP v J2EE, který nezaručuje interoperabilitu. Zde se předávají serializované Java objekty.
Základní operace v SOA architektuře se popisují pomocí SOA trojúhelníku, který je zobrazen na následujícím obrázku:

[1] V technologii CORBA se přenáší objekty, které v sobě mohou obsahovat i business logiku. K tomu, aby bylo možné logiku vykonávat na různých systémech je nutné mít zaručenou kompatibility na úrovni binárních dat reprezentující kód.



Obrázek 1 - Základní operace

Žadatel nejprve vyhledá podle zadaných kritérií službu v registru služeb (UDDI) a obdrží lokaci služby (obvykle URL). Při znalost lokace služby provede spojení s poskytovatelem služby za pomocí protokolu uvedeného v definici služby (lokaci). Poskytovatel služby (producent) publikuje služby do veřejného registry (UDDI).V současných implementacích se velmi často UDDI registr služeb vynechává. Pak žadatel o službu musí znát lokaci popisu webové služby předem.

Orchestrace služeb

Pokud máme dostatečné množství služeb pak můžeme začít stavět kompozitní aplikace z nezávislých služeb. Služby mohou vzniknout na základě požadavků na implementaci podnikových procesů, kdy cílem implementace je funkční podnikový proces či na základě požadavků na aplikaci či komponentu nebo na základě spojení existujících služeb. Výsledné aplikace složené z více služeb mohou být naimplementovány některým ze současných programovacích jazyků jako je Java či C# nebo můžeme využít grafických nástrojů, které používají jazyk pro procesní choreografii webových služeb WS-BPEL. Výhody použití jazyka pro procesní integraci jsou: rychlá reakce na změny a rychlost implementace. Procesor jazyka WS-BPEL obsahuje korelační a transformační mechanismy, které by si programátor implementující choreografii musel psát sám. Procesor jazyka BPEL může používat externích služeb a naopak může být BPEL proces poskytnut jako služba viz Service Component Architecture.

Integrace aplikací s využitím architektury zaměřené na služby

Integrace podnikových aplikací za pomocí architektury zaměřené na služby (**S**ervice **O**riented **I**ntegration) přináší mnoho výhod a dokáže ušetřit společnostem velké množství nákladů. Až do příchodu webových služeb a architektury zaměřené na služby byla integrace podnikových aplikací velmi nákladnou záležitostí, neboť bylo potřeba vydefinovat standard podle kterého se integrace řídila (vybudovat společný datový slovník, definovat přenosové protokoly, atd..). Výsledkem pokusů o integraci aplikací bylo většinou vytvoření tzv. špagetového efektu, kdy se aplikace integrovaly napřímo a s množstvím m aplikací bylo zapotřebí n^m propojení, kde n je počet propojení.

Při použití návrhové vzoru „hub and spoke[1]“ dochází k vytvoření společného standardu, do kterého všechny aplikace musí konvertovat svá data a vzniká tedy pouze *n* propojek mezi aplikacemi a hubem.
Architektura zaměřená na služby spolu s webovými službami přináší jednotný a standardizovaný komunikační protokol a standard pro popis rozhraní, které aplikace nabízejí. Díky webovým službám a jejich rozšíření je pak implementace této integrace velmi rychlá a nabízí otevřenou architekturu založenou na standardech. Při integraci aplikací pomocí webových služeb vzniká společný datový model, do kterého všechny aplikace konvertují data.

Technologie

V předchozí kapitole byla popsána architektura založená na služby a její výhody a nevýhody. V této kapitole se zaměříme na popis technologií, které s architekturou zaměřenou na služby souvisí.

Web sevices

Webové služby jsou technologií umožňující komunikaci dvou nezávislých komponent či aplikací s využitím současně dostupných technologií (XML) a za pomocí standardních internetových protokolů (HTTP(S)). Webové služby vznikly jako odpověď na technologii CORBA, kde se významní hráči na trhu s technologiemi pro vývoj podnikových neshodli na jednotném standardu. Na specifikaci webových služeb se podíleli významné společnosti v čele se společností IBM a Microsoft. Postupně se do skupiny definující standard zapojily další významné společnosti jako je SUN, Oracle, BEA, SAP, atd.. Zapojení těchto významných hráčů slibuje implementaci standardu v jejich řešení a interoperabilitu při propojování aplikací psaných na technologiích výše uvedených společností.
Webové služby komunikují pomocí protokolu **S**imple **O**bject **A**ccess **P**rotocol který je bezestavový a bezkontextový. Webová služba má jasně popsané rozhraní a to pomocí jazyka pro popis webových služeb **W**ebservice **D**escription **L**anguage. WSDL definuje datové struktury pomocí XML Schématu a operace, které služba poskytuje. Základem obou standardů popisujících webové služby je standard XML.
Základní vlastnosti webových služeb jsou:

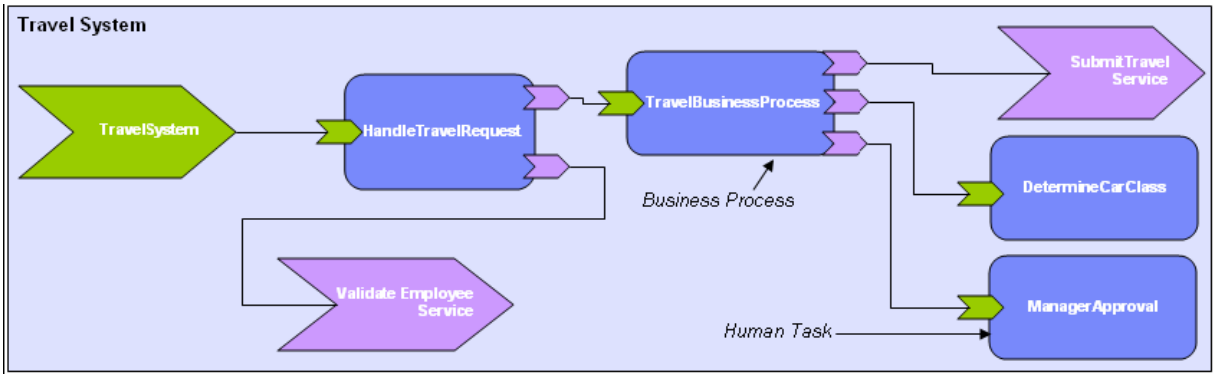
- samopopisnost – služba poskytuje pouze operace, které jsou velmi dobře popsané a definované
- samoobsažnost – přenosový protokol SOAP obsahuje dostačné množství informací o přenášeném obsahu, tzn. je možné jednoduše přiřadit data k operaci
- nezávislost na lokaci – webová služba není závislá ne jejím fyzickém umístění
- nezávislost na platformě a na implementaci - služba komunikuje standardním protokolem, má popsáno své rozhraní a tudíž nezáleží na způsobu implementace
- bezestavovost – služby by neměly držet stav. Stav by měl držet klient.

Service Component Architecture a Service Data Objects

Pro aplikování SOA vznikají specifikace, které tento architektonický styl podporují. Tak také vznikla společná iniciativa společností BEA, IBM, IONA, Oracle, SAP, Siebel Systems, Sybase. Jejím cílem je přinést takové prostředky, které vývojářům usnadní konstrukci aplikací založených právě na SOA. Nosnými specifikacemi, které vznikly v rámci této iniciativy jsou Service Component Architecture a Service Data Objects. Společně umožňují rozdělit SOA řešení do odpovídajích elementů a zjednodušují reprezentaci business logiky a dat. Definují

[1] Hub and spoke vzor představuje složení ze společného středu, do kterého se sbíhají paprsky. Komunikace mezi satelity probíhá vždy přes společný střed namísto přímé komunikace mezi satelity. Původ spojení pochází z jízdního kola (hub – střed kola, spoke – paprsek kola).

standard pro implementaci služeb jako znovupoužitelných komponent, které budou schopny mezi sebou komunikovat a tvořit tak jednotnou aplikaci.
Service Component Architecture poskytuje model pro stavbu systémů a aplikací s použitím SOA, je založena na dosavadních zkušenostech s tímto architektonickým stylem a snaží se o systematický přístup k implementaci služeb na principu komponent, které poskytují své služby prostřednictvím rozhraní orientovaného na služby (*service-oriented interface)* a na druhé straně pro svou činnost využívají rozhraní u jiných komponent prostřednictvím odkazů (*service reference)*. Komponenty mohou přistupovat na externí služby ze systémů prostřednictvím importů, které definují mechanizmus komunikací s touto externí službou.
Na obrázku 2 je pro ilustraci znázorněn jednoduchý příklad takového složení z komponent. Systém pro schvalování služebních cest poskytuje své služby pomocí service-oriented interface TravelSystem. Komponenta HandleTravelRequest provádí validaci žadatele prostřednictvím externí služby (service reference Validate Employee Service) a volá TravelBusinessProcess komponentu. Komponenta je realizovaná jako business process a pro svou činnost potřebuje komponenty a externí službu, které jsou v diagramu úplně vpravo. Mezi nimi je komponenta ManagerApproval, která zajišťuje schválení požadavku pověřenou osobou. Jednotlivé komponenty pak jsou implementovány různými způsoby, některé jsou psané přímo programovacím jazyku, jiné používají nějaký již hotový engine – například BPEL engine, human task engine apod.
SCA se snaží abstrahovat od detailů použité infrastruktury a dokonce přímo od způsobu, jak budou služby volány. Vývoj SOA aplikací lze pak rozdělit do dvou hlavních činností, první je implementace a příprava komponent, druhá pak zahrnuje vlastní skládání a propojování komponent do živoucí aplikace nebo systému. Tyto dvě činnosti jsou ve specifikaci SCA striktně odděleny a umožňují soustředit se při vývoji na skládání služeb do výsledné aplikace bez nutnosti zabývat se implementačními detaily jednotlivých komponent a naopak věnovat se implementaci komponenty jako služby a skládání do aplikace přenechat integračnímu vývojáři.



Obrázek 2 Propojení SCA komponent

Pro sestavení aplikací na základě služeb je potřeba řešit jednotný formát dat, který bude srozumitelný všem spolupracujícím službám. Service Data Objects (SDO) jsou navrženy právě za tímto účelem a sjednocují jak přístup k datům, tak jejich formát - to znamená, jak jsou data reprezentována jednotlivým službám v řešení SOA. SDO představuje jednotný způsob, jak přistupovat k datům, ať se jedná o relační databáze, JCA záznamy, XML, EJB Entity, Cobol copybook a podobně. Definuje způsob, jak provádět vytváření, změnu, mazání dat bez ohledu na konkrétní implementaci a uložení dat. SDO tak odstiňují vývojáře od zbytečně zatěžujících technických detailů přístupu k cílovému datovému zdroji. To je přenecháno tzv. data mediator services, jejímž úkolem je převádět nativní tvar dat do SDO a naopak. Tento přístup také umožňuje založení kanonického datového modelu, který ve

výsledku tvoří centrální datovou základnu sloužící pro komunikaci mezi službami. Při přístupu na konkrétní datové zdroje pak bude použito mapování mezi tímto kanonickým modelem a konkrétními daty v datovém zdroji. Takže při integraci aplikací mluvíme o aplikačně specifických SDO a generických SDO mezi nimiž existuje předpis mapování. Toto mapování může být velmi jednoduché, ale může se také stát, že v rámci mapování budou prováděny složitější transformace s daty jako je spojování, extrakce částí datových polí apod. Kombinace SCA a SDO přinášejí komplexnější přístup k programovým prostředkům v SOA a umožňují tento architektonický styl zavádět jednotně bez ohledu na použité způsoby volání služeb a použitou infrastrukturu.

BPEL

Business Process Execution Language představuje v současné době standard pro popis business procesů a choreografii služeb. Jedná se o standardizovaný popis ve formátu XML, který umožňuje modelovat a provádět business procesy v nástrojích, které akceptují tuto specifikaci. Jak je popsáno v předchozí kapitole, v rámci SCA komponent představuje BPEL jeden druh implementace komponenty. Tak je možné velmi snadno tam, kde je to vhodné, provádět dekompozici poskytovaných služeb do procesů, které jsou pomocí BPEL implementovány.

Vývoj moderních podnikových aplikací

Způsob vývoje podnikových aplikací prošlo značným rozvojem a vyvinuly se různé metody pro řízení, sledování postupu vývoje, pro členění jednotlivých činností. Vytvořily se definované a popsané disciplíny softwarového inženýrství. Pro vývoj softwaru byly definovány a na tisících praktických projektech ověřovány a dolaďovány postupy vývoje. V současné době patří mezi nejrozšířenější „Unified Process“ a „eXtreme Programing“. Architektura orientovaná na služby přináší svým charakterem nový přístup k vývoji aplikací. Skládáním aplikace z volně provázaných komponent, které mají deklarativní nastavení bezpečnosti, transakcí a které přeskupením mohou vytvořit novou aplikaci, je umožněno rozdělit oblasti zájmu. Takže do budoucna je možné mít vývojáře pro implementace služeb, kteří připravují a testují služby a pak integrační vývojáře, kteří nemusí znát jednotlivé programovací jazyky a technické detaily implementací a soustředí se na složení aplikace z SCA komponent. Integrace s ostatními systémy a dokonce celé business procesy jsou zapouzdřeny jako SCA komponenty poskytující služby a jejich výměna na úrovni integrace služeb je mnohem snadnější než změna kódu uvnitř implementace komponent. Lze si tak snadno například představit SCA komponentu, která je implementovaná pomocí EJB. Časem se objeví k dispozici veřejná služba ve formě Web Service, která tuto komponentu může nahradit a má například lepší kvalitu poskytované služby. Vývojové prostředí pomáhá při realizaci jednoduchého přepojení na Webovou službu, která je opět v systému zapouzdřena jako SCA komponenta a není vůbec potřeba upravovat zdrojový kód původní aplikace.

Způsoby pří návrhu podnikových aplikací za pomocí SOA

Stejně jako pro jiné architektonické styly platí, že Service Oriented Architecture má svá specifika a úskalí. I v tomto případě je velmi významné používání architektonických přístupů (šablon) a vyvarování se používání nevhodných architektur. Pro SOA vznikají nové šablony

řešení, jako je například „Requester side caching pattern[1]“ apod. Charakter service oriented architecture dovoluje velmi dobře použít vývoj od programového kódu k sestavení celé aplikace (bottom-up), tak sestavení aplikace a teprve potom implementovat vlastní komponenty (top-down). Zavedení vývojového procesu v SOA je v různých organizacích na různých úrovních stejně tak, jako byla klasifikována vyspělost organizace vývoje pro vývoj klasických informačních systému, jsou zaváděny klasifikace stupně vyspělosti vývoje v SOA, které si všímají organizace vývoje jednotlivých služeb, znovupoužívání již existujících služeb, schopnost plánování služeb, které budou sdílené více projekty, schopnost predikce vývoje a podobně.

Bottom-up

Tento přístup je vhodný, pokud již jsou některé části připraveny nebo pokud používáme již existující služby. Postup je od implementací přes sestavení aplikace pomocí propojení implementovaných SCA komponent.

Top-down

Protože SCA komponenta zapouzdřuje vlastní implementaci, je velmi snadné praktikovat vývoj shora dolů, kdy se nejprve sestaví prototypy SCA komponent a tento skeleton je následně doplňován implementacemi vlastních komponent. To umožňuje velmi rychle získat architecture baseline a následně řídit vývoj metodou rozděl a panuj.

Meet-in-the-middle

V praxi je někdy je účelné oba přístupy kombinovat. Tak je možné postupovat řízeně od modelů business procesů a využít stávající aplikace a to, co poskytují. Pro vzájemné propojení pak je možné použít tzv. „mediatory“, které mají nastavené mapování rozhraní "interface“. Toto mapování interface je zajímavé i v případě změn v interface služeb. Tak je možné si například představit službu, která je poskytována třetím dodavatelem. Pokud tento dodavatel změní interface pro volání služby, pak stačí nastavit správně mapování a není potřeba upravovat kód našeho systému.

Návrh a modelování podnikových procesů

Modelování podnikových procesů je činnost, která se dostává hodně do popředí. Existují specializované společnosti, které disponují experty pro jednotlivé oblasti a používají se specializované nástroje a postupy pro návrh business procesů. Hlavním přínosem modelování a systematického přístupu k návrhu business procesů je možnost dobrého porozumění a možnost zdůvodněné transformace stávajících, zjištění potenciálních míst pro zlepšení procesů a nalezení slabých míst, kde dochází ke ztrátám. Umožňuje také dobře definovat strukturu metrik pro měření výkonu, na základě získaných metrik v reálném prostředí, pak lépe porozumět zákonitostem a případně provést úpravy stávajících procesů.
Jedním z modelovacích nástrojů je IBM WebSphere Business Modeler, který představuje poměrně vyspělý nástroj pro modelování podnikových procesů. Tento nástroj disponuje rejstříkem modelů jako jsou procesní model pro grafickou reprezentaci business procesu, model zdrojů pro modelování podnikových zdrojů (uživatelé, systémy), informační model umožňují pohled na data a jejich použití v rámci business procesů, organizační model pro modelování organizační struktury, analytický model pro definici kličových procesních metrik a atributů a spolu s modelem měření obchodních výsledků je pak možné definovat přímo

[1] Requester Side Caching je vzor, který pomáhá řešit výkonové problémy zavedením cache proxy na klientské straně.

klíčové ukazatele výkonu nasazené a sledované v běžících procesech. Nasbírané hodnoty je pak následně možné použít v modelovacím nástroji pro simulaci a případný optimalizaci procesního modelu (tzv. re-engineering).

Návrh a modelování služeb

Dá se říci, že služby jsou základním kamenem SOA a jejich správným návrhem je možné dosáhnout výhod, které SOA nabízí. Naopak nesprávným návrhem, nevhodným rozvržením služeb a operací, se může projekt dostat do problémů s integrací služeb, údržbou, výkonem, rozšiřitelností... Článek 17) se zabývá základními principy, které pomáhají při definici služeb a jednotlivých operací služeb. Jedná se především o správné rozdělení služeb, které by měly být kohezní a kompletní. Tzn. že v rámci jedné služby budou operace, které spolu souvisí. Upřednostňuje se koheze funkčnosti, která je nejblíže obchodním případům použití. A zároveň, že služby budou co nejvíce kompletní, tzn. budou pokrývat pro danou oblast požadovanou funkcionalitu. To i s výhledem na znovu použitelnost služby budoucími konzumenty služby. Dodržením konvencí pojmenování a dobrým členěním služeb lze dosáhnout velmi dobrého intuitivního popisu rozhraní služby. V rámci modelování služeb je často vhodné použít kromě popisu vlastního interface i další podpůrné modely jako jsou komponentové modely, stavové a sekvenční diagramy.

Vývoj služeb

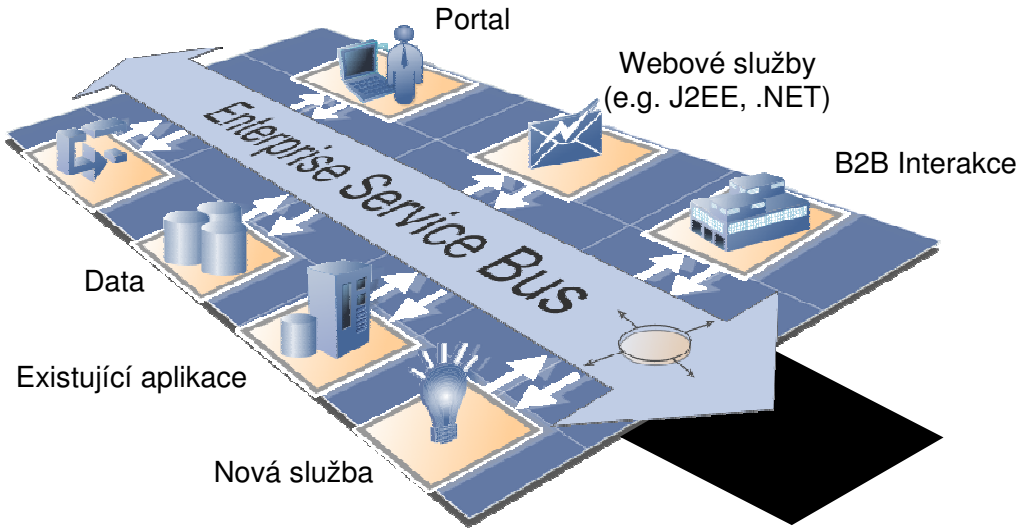
Na vývoj služeb jsou vypracovávány různé postupy a názory. Jedním z nich je Service Oriented Modeling and Architecture (SOMA), která se zabývá způsobem jak identifikovat, specifikovat a realizovat služby viz 5)6) Pro identifikaci služeb se kombinují identifikace ze strany obchodních „use case“ tzv. domain dekompozicí s analýzou služeb, které vzniknou na základě funkcionality stávajících aplikací. Kromě toho je potřeba identifikovat služby, které budou k dispozici pro splnění dalších cílů jako například monitorování, které nebyly identifikovány ani přístupem od obchodních popisů „use case“ ani analýzou stávajících aplikací. Takto získané služby se pak kategorizují a určují se tak vzájemné závislosti a příslušnost k různým vrstvám architektury. Analýza subsystémů má za úkol popsat závislosti a toky mezi jednotlivými subsystémy a definovat objektové modely interface jednotlivých subsystémů.

Infrastruktura pro moderní podnikové aplikace

Podniková sběrnice služeb

Komponenty SCA jsou založeny na nezávislosti na komunikačním protokolu a na infrastruktuře, na které cílově běží. To umožní vyrábět znovupoužitelné komponenty s různou implementací a tzv. binding – způsobem volání těchto služeb. Na druhou stranu je následně potřeba zajistit prostředí, ve kterém budou SCA komponenty běžet, zajistit potřebnou infrastrukturu. Snaha je zavést prostředí, které se bude chovat jako sběrnice a jednotliví přispěvatelé budou moci mezi sebou komunikovat jejím prostřednictvím. Tak vznikla myšlenka Enterprise Service Bus (ESB), inteligentní infrastruktury pro komunikaci, transformaci a směrování. ESB je koncept, který může být realizován různými prostředky a měl by zajišťovat jak synchronní, tak asynchronní volání, poskytovat spolehlivou, bezpečnou a transakční platformu. Pro snadnější realizaci ESB jsou vyráběny specializované produkty, které mají již připraveny základní stavební kameny EBS a jsou dále rozšiřitelné podle specifických požadavků. Kromě toho vznikají šablony, které popisují různé topologie stavby

ESB a různé techniky, které umožní jeho efektivní nasazení v konkrétních podmínkách viz 5) Příkladem takového produktu je IBM WebSphere Enterprise Service Bus.



Obrázek 3 Enterprise Service Bus

Monitorování business procesů

Při provozu podnikové infrastruktury je snaha mít přehled o efektivitě probíhajících procesů, mít možnost definovat prahové hodnoty pro sledování vyjímečných situací, mít zajištěn systém upozornění na tyto události a v neposlední řadě získávat data k dalšímu hodnocení a zpracování. Na základě takto získaných dat lze pak odhalovat slabá místa v procesování a provádět odpovídající korekce z modelu a implementaci business procesů. Pro monitorování je potřeba určit, co je potřeba sledovat a jaké prahové hodnoty považovat za kritické apod. Vzniká tak sada klíčových ukazatelů výkonnosti tzv. Key Performance Indicators (KPIs), která popisuje sledované veličiny a události. Ty pak jsou prostřednictvím nástroje pro monitorování aplikovány na běžící procesy a získaná data dále použita pro analýzu výkonnosti, zjištění anomálií, oznamování situací, které je potřeba hned řešit nebo sledování trendů a reakce na události, které sice ještě nenastaly a situace k nim spěje. Jedním z monitorovacích nástrojů je IBM WebSphere Business Monitor, který je součástí rodiny software pro integraci podnikových procesů. Spolupracuje s WebSphere Business Modelerem, od kterého získává definici KPIs a kterému dodává získaná data z reálného provozu k provádění simulací na modelu a dále spolupracuje s WebSphere Process Serverem, ve kterém běží provoz a kde se sbírají reálná data z aktuálně probíhajících procesů.

Závěr

Servisně orientovaná architektura je architektonický styl, který mění přístup k vývoji podnikových aplikací a slibuje větší flexibilitu podnikové infrastruktury při zajišťování efektivního fungování podniku a efektivitu vývoje. Zkracuje dobu implementace nových služeb a zachovává existující infrastrukturu podniku. Dosavadní praktické zkušenosti zpřesňují postupy vývoje a přinášejí praktické nároky na používané nástroje. Servisně Orientované Architektuře je věnována velká pozornost, svědčí o tom velké množství článků, publikací a nástrojů, které tento směr podporují.

Literatura

1) SOA and Web services; http://www-128.ibm.com/developerworks/webservices
2) WebSphere Enterprise Service Bus; http://www-306.ibm.com/software/integration/wsesb/
3) http://www-306.ibm.com/software/integration/wbimonitor/library/documentation.html
4) Patterns: Service-Oriented Architecture and Web Services; http://www.redbooks.ibm.com/abstracts/sg246303.html?Open
5) Patterns: Implementing an SOA using an Enterprise Service Bus; http://publib-b.boulder.ibm.com/Redbooks.nsf/RedpieceAbstracts/sg246346.html?Open
6) Service-oriented modeling and architecture; http://www-128.ibm.com/developerworks/library/ar-itio4/
7) Service Oriented Architecture Maturity Model; http://www.soaconsultant.com/html/soamm.shtml
8) Introduction to Service Data Objects; http://www-128.ibm.com/developerworks/java/library/j-sdo/
9) Service Component Architecture; http://www-128.ibm.com/developerworks/library/specification/ws-sca/
10) The Business Value of the Service Component Architecture (SCA) and Service Data Objects (SDO) http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-sca/SOAProgrammingModelBusinessValue.pdf
11) Building Systems using a Service Oriented Architecture; http://downloadboulder.ibm.com/ibmdl/pub/software/dw/specs/ws-sca/SCA_White_Paper1_09.pdf
12) Introducing a New Service-Oriented Architecture Maturity Model; http://www.sonicsoftware.com/solutions/learning_center/soa_insights/movin_soa_on_up/index.ssp
13) Service Oriented Unified Process; http://www.soaconsultant.com/pdfs/soa_maturity_part3.pdf
14) Building SOA applications with reusable assets: Reusable assets, recipes, and patterns; http://www-128.ibm.com/developerworks/webservices/library/ws-soa-reuse1/
15) Web Service Orchestration with BPELů http://www.idealliance.org/papers/dx_xml03/papers/04-06-01/04-06-01.html
16) Demystifying Enterprise Service Bus; Technology; http://www.fiorano.com/whitepapers/fiorano_esb.pdf
17) SOA realization: Service design principles; http://www-128.ibm.com/developerworks/webservices/library/ws-soa-design/
18) Juric M.B., Business Process Execution Language for Web Services BPEL and BPEL4WS, Packt Publishing, 2006
19) Erl T., Service-Oriented Architecture (SOA): Concepts, Technology, and Design, Prentice Hall, 2005
20) Erl T., Service-Oriented Architecture : A Field Guide to Integrating XML and Web Services (Paperback), Prentice Hall, 2004
21) SOAP Version 1.2; http://www.w3.org/TR/soap12-part1/
22) BPEL Source; http://www.bpelsource.com/index.html