ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

PROVOZNĚ EKONOMICKÁ FAKULTA

KATEDRA INFORMAČNÍCH TECHNOLOGIÍ

BAKALÁŘSKÁ PRÁCE

Servisně orientovaná architektura (SOA)

Rostislav STŘÍBRNÝ

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Katedra informačních technologií

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Stříbrný Rostislav

Informatika

Název práce

Servisně orientovaná architektura (SOA)

Anglický název

Service-oriented architecture (SOA)

Cíle práce

Bakalářská práce je tematicky zaměřena na problematiku servisně orientované architektury (SOA). Hlavním cílem práce je charakterizovat technologické aspekty tvorby software dle konceptu servisně orientované architektury. Dílčí cíle bakalářské práce jsou:

- analyzovat přístupy k architekturám IT systémů,
- charakterizovat odlišné pohledy na SOA a
- navrhnout a implementovat vlastní řešení softwarové aplikace dle architektury SOA.

Metodika

Metodika řešené problematiky bakalářské práce je založena na studiu a analýze odborných informačních zdrojů. Vlastní řešení je realizováno formou návrhu a implementace jednoduché softwarové aplikace podle zásad architektury SOA. Na základě syntézy teoretických poznatků a výsledků vlastního řešení budou formulovány závěry bakalářské práce.

Harmonogram zpracování

1) Příprava a studium odborných informačních zdrojů, upřesnění dílčích cílů práce a volba postupu řešení: 6/2011
2) Zpracování přehledu řešené problematiky dle informačních zdrojů: 7/2011- 8/2011
3) Vypracování vlastního řešení, diskuze a zhodnocení výsledků: 9/2011 - 10/2011
4) Tvorba finálního dokumentu bakalářské práce: 11/2011 - 2/2012
5) Odevzdání bakalářské práce a teze: 3/2012

Rozsah textové části

30 - 40 stran

Klíčová slova

Service Component Architecture (SCA), Service Data Objects (SDO), Service-Oriented Architecture (SOA), Enterprise Service Bus (ESB), Enterprise Application Integration(EAI)

Doporučené zdroje informací

1. Cummins, Fred. Building the agile enterprise with SOA, BPM and MBM. Amsterdam Boston: MK/OMG Press/Elsevier, 2009. ISBN-13: 978-0-12-374445-6.

2. Erl, Thomas. SOA: servisně orientovaná architektura: kompletní průvodce. Brno: Computer Press, 2009. 1. vydání. 671 s. ISBN 978-80-251-1886-3.

3. Pulier, Eric, and Taylor, Hugh. Understanding Enterprise SOA. Greenwich, Conn: Manning, 2006. ISBN-13: 978-1-932394-59-7.

4. IT Systems. Brno: CCB spol. s r. o. ISSN 1802-002X.

5. Systémová integrace. Praha: ČSSI. [on-line]. ISSN 1804-2716. <http://www.cssi.cz>.

Vedoucí práce

Ulman Miloš, Ing., Ph.D.

Termín odevzdání

březen 2012

doc. Ing. Zdeněk Havlíček, CSc.
Vedoucí katedry

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE
Provozně ekonomická fakulta

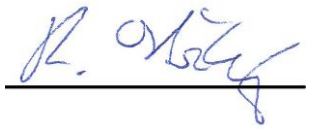
prof. Ing. Jan Hron, DrSc., dr.h.c.
Děkan fakulty

V Praze dne 21.11.2011

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci „Servisně orientovaná architektura (SOA)“ jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu literatury na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 26.3.2012



Poděkování

Rád bych touto cestou poděkoval Ing. Miloši Ulmanovi, Ph.D. za jeho ochotu a rady, které mi poskytoval při tvorbě této práce. Dále pak chci poděkovat své nejbližší rodině, která mě během mého studia podporovala a která mi vždy dodávala nové síly. Především bych pak chtěl poděkovat své manželce Dáše Stříbrné, která mi byla po celou dobu studia neotřesitelnou oporou, a také své mamince Olze Stříbrné, která mě v nelehkých životních podmínkách svědomitě a dobře vychovala.

Servisně orientovaná architektura (SOA)

Service-oriented architecture (SOA)

Souhrn

Předložená práce studuje oblast informačních technologií, která souvisí se servisně orientovanou architekturou. Metodou studia je analýza a syntéza informací z uvedených pramenů. Práce poskytuje přehled historie vzniku SOA včetně charakterizace návazností na vybrané architektonické přístupy. Uvádí základní představy o definici vlastností servisně orientovaného přístupu a základních principech servisní orientace. Věnuje se jednotlivým fázím a způsobům zavádění do praxe a také obecným přístupům k analýze a návrhu servisně orientovaných služeb. Identifikuje a rozebírá omezení, která mají vliv na užití SOA jako celku, a uvádí přehled technologií, které tvoří základ implementace. Je zde uveden přehled vybraných produktů, kterými lze implementaci realizovat. Vlastní práce se zaměřuje na návrh servisně orientované architektury fiktivního podniku, implementaci servisně orientovaných služeb a jejich užití v rámci obchodního procesu. Výsledkem práce je ucelený náhled na servisně orientovanou architekturu a potvrzení obecných principů servisní orientace.

Summary:

Following work studies area of information technologies which is related to service-oriented architecture. Used study method includes analysis and synthesis of information from mentioned sources. Work provides historical overview about SOA creation including characterization of related architectonical approaches. It presents definition of basic features of service-oriented approach as well as basic principles of service orientation. It addresses SOA implementation phases and their details including general approach description of analyzing and designing service-oriented services. It identifies and analyses limitations which are related to SOA as a whole, and provides overview of technologies which constitutes basics for implementation. Work also provides list of chosen SOA-ready products. It focuses on designing service-oriented architecture of fictive company, including implementation of services and business process which both corresponds to SOA. Work outcome is comprehensive outlook on service-oriented architecture and confirmation of generic principles of service orientation.

Klíčová slova: Service Component Architecture (SCA), Service Data Objects (SDO), Service-Oriented Architecture (SOA), Enterprise Service Bus (ESB), Enterprise Application Integration (EAI), SOA Governance.

Keywords: Service Component Architecture (SCA), Service Data Objects (SDO), Service-Oriented Architecture (SOA), Enterprise Service Bus (ESB), Enterprise Application Integration (EAI), SOA Governance.

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

Při bližším zamyšlení se nad významem technologického pokroku se nejprve části z nás vybaví některá zajímavost z oblasti vědy anebo jednoduše některá z hi-tech novinek, která se kolem nás v poslední době objevila. Teprve až s dalším odstupem si začneme uvědomovat rozsah již dosaženého světového pokroku a také významné milníky lidstva, kterých již bylo dosaženo. Nakonec si vybavíme naše individuální vize a touhy, kam by lidstvo jako celek mohlo v budoucnu směřovat, anebo jednoduše výzvy, kterým čelíme již dnes.

Touha tvořit a objevovat je v nás pevně zakotvena a je univerzální vlastností nejenom v západní společnosti. Aby tato touha mohla být naplněna, je nezbytné přistoupit ke konkrétním krokům, které povedou k dosažení našich osobních cílů a vizí. Těmito kroky zajisté mohou být výzkum, vývoj či inovace. V dnešní době je však nezbytná interdisciplinární spolupráce, jelikož již delší dobu není možné pojmout individuálně všechny dostupné informace ze všech oborů. Stejně tak není možné uvědomovat si všechny případné souvislosti ať již mezi dostupnými informacemi v jednotlivých oborech anebo v hloubce jednotlivých témat.

Pro efektivní fungování je tedy bezpochyby nutné, aby tato interdisciplinární spolupráce měla určitý řád. Tento řád může být tvořen oddělením jednotlivých činností - specializací, kategorizací jednotlivých druhů objevů - informací, a v neposlední řadě schopností se ve všech těchto datech orientovat, tedy vyhledávat v nich a tvořit mezi nimi vazby. Takto je například členěna již samotná věda, která se z pohledu oddělení činností dělí na jednotlivé obory. Naopak publikováním a citováním výstupů vědecké činnosti zase dosahujeme jak cíle kategorizovat jednotlivé informace, tak i tvoříme příslušné vazby mezi informacemi.

Tento model sdílení informací je jedním z fungujících způsobů jak provádět značně distribuovanou činnost, a to v celku efektivně. Podobné je to v oboru informačních technologií (IT), kde je také nezbytná interakce mezi oddělenými jedinci, kteří mají omezenou množinu informací. Tentokrát ne však mezi lidmi, ale mezi stroji.

Obor informačních technologií je silně dynamická oblast, kde jen za posledních 10 let došlo k několika technologickým revolucím a zvratům. Navíc s každým rokem přibývá nespočet nových softwarových produktů a vylepšování těch stávajících. Všechny tyto softwarové produkty, či aplikace tvořené na míru, jsou do jisté míry důsledkem obecné snahy o uvedení nových inovativních služeb na trh a tedy odlišení se od konkurence.

Již delší dobu si silní hráči působící na trhu vývoje IT software uvědomují, že pro takto silně heterogenní prostředí je nezbytně nutné nastavit určitá pravidla. Tato pravidla musí umožnit především vzájemnou interakci mezi jednotlivými systémy všech výrobců a také dlouhodobost určitého konceptu z pohledu údržby a dalšího rozvoje těchto produktů. Toho bylo částečně dosaženo standardizací v některých oblastech IT. S postupem času však bylo čím dál jasnější, že pouhá standardizace nemůže stačit, a že bude nutné stanovit takový obecný koncept, který bude určovat směr samotné IT architektury a designu. Postupem času se tedy přechází od základních myšlenek distribuovaného výpočtu (Distributed Computing), přes objektové a komponentové přístupy, Enterprise Application Integration až po standardizaci technologií, kterou vidíme dnes.

Ve světle postupu ve vývoji jednotlivých konceptů vznikaly i odpovídající softwarové produkty, které na daný způsob myšlení navazovaly. Jejich implementace v reálném světě však nenaplnila očekávané výsledky, tedy odstranění či omezení klíčových problémů při vývoji a údržbě IT systémů. Především zůstaly neřešeny otázky spojené se systémovou evidencí již existující funkcionality IT systémů a návaznou schopností z nich budovat rychle, koncepčně a levně nové služby. Řešením těchto problémů se v posledních letech zabývá nový architektonický koncept, servisně orientovaná architektura (SOA).

Tento koncept mimo jiné přináší schopnost uceleně a jednotně řešit problémy spojené s kombinováním funkcionalit v různých heterogenních systémech, čímž lze dosáhnout v uvádění nových služeb na trh vyšší flexibility. Navíc se může jednat o služby poskytované nejenom stávajícími IT systémy uvnitř daného podniku, ale také v kombinaci se službami jiných subjektů. Dalším důležitým aspektem je schopnost výměny či upgrade celých IT systémů, a to dokonce se značně sníženými finančními nároky. Tento předpoklad však vyžaduje striktní dodržení SOA konceptu jako celku, a to napříč celým podnikem.

SOA je ale především koncept. Tento koncept s sebou přináší celou řadu pravidel a povinností, které je nutné dodržet a které mohou do značné míry odporovat již zaběhnutým projektovým či obchodním zvyklostech uvnitř podniku. Proto je při zavádění SOA konceptu nezbytná celková a dlouhodobá podpora jak ze strany odpovědných vedoucích pracovníků, tak i všech ostatních složek podniku, které se podílí na zavádění nových či údržbě stávajících služeb.

2 Cíl práce a metodika

Cílem této bakalářské práce je co nejvíce přehledně a uceleně přednést základní aspekty konceptu servisně orientované architektury, a to s ohledem na již tak dost velkou abstraktnost samotného tématu. Cílem praktické části práce je potom navrhnout a implementovat jednoduchou aplikaci dle architektury SOA.

Celkový proces implementace SOA lze pojmout z pohledu manažersko-obchodního, návrhu IT architektury a designu, anebo z pohledu užitých technologií. S ohledem na požadované cíle se tato práce věnuje primárně oblastem IT architektury a designu, a až druhotně zbylým oblastem.

Koncept servisně orientované architektury je v této práci popsán takovou formou, která přináší hrubý přehled, kam SOA ve světě IT zapadá, a základní rysy a požadavky, které musí být splněny při zavádění SOA do praxe. Z tohoto pohledu je tedy nezbytné začít nejprve uvedením základních požadavků na IT architekturu jako celek a následně pokračovat konkrétněji k samotné servisně orientované architektuře.

Úvodem do tématu je charakteristika SOA a uvedení základních principů servisní orientace služeb. Následuje uvedení jednotlivých fází zavádění SOA v podniku s ohledem na možná rizika, a to především z pohledu řízení tohoto procesu. Dále jsou uvedeny základní teze a postupy týkající se servisně orientované analýzy a návrhu služeb. Dostatečná pozornost je věnována také oblasti technologické. Především se zde jedná o základní charakteristiku vybraných technologií a produktů, které je možné použít při vývoji software dle principů SOA.

Práce dále pokračuje uvedením praktické ukázky, jejímž cílem je implementace jednoduché funkcionality dle principů SOA. Ukázka zahrnuje návrh vhodné IT architektury, výběr softwarových produktů, návrh a implementaci servisně orientovaných služeb a následně propojení těchto služeb za účelem představení výhod servisně orientované architektury.

Metodika řešené problematiky je založena na studiu a analýze odborných informačních zdrojů. Jelikož je zvolená problematika dosti abstraktní a zároveň pokrývá širokou oblast IT, tak je nutné ji nejprve rozčlenit na jednotlivé logické celky, a tyto celky

dále pečlivě strukturovat. Posléze bude možné provést detailnější zkoumání jednotlivých odborných zdrojů s ohledem na aktuálně zpracovávanou oblast. V opačném případě by text nebyl dostatečně tematicky souvislý, a tedy by byl i hůře pochopitelný.

Pro prvotní uchopení tématu je možné se inspirovat rozčleněním, které používají autoři zde citovaných odborných publikací. Je nutné také poznamenat, že autor této práce je z teoretického i praktického hlediska dobře obeznámen s některými oblastmi implementace SOA konceptu v podnikové sféře. Z tohoto důvodu tato práce vychází nejenom ze zmiňovaných odborných zdrojů, ale také z praktických zkušeností autora a jeho vědomostí z této oblasti.

Při tvorbě bude použita analýza konceptu SOA jako celku, návaznost na členění v odborné literatuře a v neposlední řadě syntéza jednotlivých informací plynoucích z těchto pramenů.

Dále je nutné zdůraznit, že téma servisně orientovaná architektura je v odborné literatuře publikováno převážně v anglickém jazyce. Dostupné české překlady či přímo autorská díla často anglické termíny přitom nepoužívají jako slova přejatá, ale zavádí české novotvary. V praxi IT firem však nejsou tyto české překlady příliš používány a není tedy z pohledu autora této práce vhodné se jich striktně držet. Z tohoto důvodu je v textu preferována ta forma daného termínu či obratu, kde její význam je akceptovaný širokou odbornou veřejností. Takto zvolená forma termínu potom je v textu použita výhradně v jednotné podobě. Podpůrným argumentem pro tuto volbu je zároveň fakt, že práce vychází téměř výhradně z anglicky psané literatury.

Slovo podnik je v práci uvedeno v širším významu a vztahuje se na libovolné subjekty, které tvoří samostatnou organizační strukturu a mohou realizovat vlastní řešení v oblasti informačních technologií.

3 Přehled řešené problematiky

Servisně orientovaná architektura (SOA) je koncept, který vznikl na základě potřeby inovovat stávající architektonické přístupy v informačních technologiích. Následující kapitoly tento koncept představují v konkrétních souvislostech a poskytují jeho bližší charakteristiku.

3.1 Přístupy k architektuře informačních systémů

Informační technologie slouží pro potřeby reálného světa a z tohoto důvodu se také neustále rozvíjí a přizpůsobují. S postupem času a společně se zdokonalováním dostupného hardwarového a softwarového vybavení dochází ke změnám možností, které jsou schopny informační technologie poskytovat. Současně dochází ke změnám v chápání, jakými způsoby lze softwarové vybavení modelovat, což se projevuje v různorodých architektonických návrzích informačních systémů a interakcích mezi nimi. Následující kapitoly uvádějí vybrané architektonické přístupy k návrhům informačních systémů, které ovlivnily vznik SOA konceptu, či které jej dále rozvíjejí. Cílem kapitoly je přitom poskytnout základní kontext pro pochopení důvodů vzniku SOA.

3.1.1 Distributed Computing

Jedná se o základní oblast informačních systémů, která je charakterizována vykonáváním počítačového programu v distribuovaném prostředí. Distribuovaný systém se skládá z autonomních počítačových systémů vzájemně komunikujících pomocí počítačové sítě (Emmerich, 2001). Distribuovaný počítačový program je schopen provádění v takovém distribuovaném systému.

3.1.2 Objektově orientovaná analýza a design

Objektově orientovaná analýza a design (OOAD) je způsob tvorby softwarového vybavení modelující počítačové systémy pomocí skupin objektů. Výstupem objektově orientované analýzy je vytvoření datového modelu objektů, který se skládá z dat, metod a vazeb na okolní objekty. Objekty samotné zapouzdřují svá privátní data a umožňují okolním objektům přístup k těmto datům pomocí svých metod.

Objektově orientovaný design aplikuje nefunkční požadavky a omezení podle zvolené vývojové technologie a prostředí. Příkladem takových omezení může být požadavek na podporu distribuovaných transakcí, dobu odezvy při volání metod, vývojová platforma či programovací jazyk.

Objektový přístup je hojně užívaný pro vývoj systémů, jelikož zvyšuje efektivitu a kvalitu především při práci ve větších týmech či při vývoji funkčně podobných aplikací. Pulier a kol. (2006, s. 13) mezi přednosti tohoto přístupu řadí zjednodušení opětovného využívání funkcionality a také zjednodušení změn a adaptace při zpracování dat.

3.1.3 Klient-server architektura a počátky standardizace

Klient-server komunikace spočívá v takové operaci, kdy si klientský program vyžádá data anebo funkcionalitu po jiném (serverovém) programu, typicky vzdáleně přes počítačovou síť (Pulier, Taylor, 2006). Programovací jazyky typicky obsahují předpřipravené knihovny pro síťovou komunikaci a jednotlivé aplikace si tedy mohou zvolit způsob podporované výměny dat po síti. Takové aplikace tedy podporují velmi specifický způsob výměny dat po síti, který není kompatibilní s jinými aplikacemi.

Toto v důsledku vedlo ke snaze standardizovat síťovou komunikaci mezi aplikacemi, což dle Puliera a kol. (2006, s. 16) vedlo k vytvoření specifikací CORBA[1] a následně DCOM[2] jako reakce Microsoftu. Pro rozšířený jazyk Java lze uvést technologií RMI[3], která v pozdějších verzích byla rozšířena o podporu CORBA, a následně specifikaci EJB[4], která je součástí serverové specifikace J2EE[5].

---

[1] Component Object Request Broker Architecture: Specifikace standardizační organizace Object Management Group.

[2] Distributed Component Object Model: Proprietární komunikační technologie společnosti Microsoft.

[3] Remote Method Invocation: Objektově orientovaná komunikační technologie jazyka Java.

[4] Enterprise Java Beans: Specifikace vydávaná v rámci Java Community Process Program pro vývoj řízených, serverových komponent umožňujících modulární tvorbu podnikových aplikací.

[5] Java Platform, Enterprise Edition: Sada specifikací vydávaných v rámci Java Community Process Program pro vývoj podnikových aplikací.

Všechny uvedené způsoby se ovšem vyznačují společnou vlastností, a to je přímé volání vzdálených objektů. To v důsledku vede k vytvoření silné vazby mezi jednotlivými počítačovými systémy a jejich návrhy. Problémy s tím související částečně řeší následující IT koncept.

3.1.4 Enterprise Application Integration

EAI je termín používaný pro plánování, metody a nástroje užívané pro modernizaci, konsolidaci a koordinaci různorodých nezávislých aplikací uvnitř podniku. Umožňuje zvýšení efektivity podniku pomocí výměny dat mezi aplikacemi, které byly vyvinuty pro odlišné operační systémy, databáze či za použití proprietárních technologií. Principem je zpřístupnit existující data a business logiku pro další použití, a to ideálně při minimalizaci nutných zásahů do zdrojové či cílové aplikace.

Tento přístup také odstraňuje část problémů, které jsou spojovány s distribuovanými systémy. Jednotlivé aplikace podporují technologie, které nemusí být podporovány v okolních systémech. Produkty EAI tvoří prostředníka mezi těmito systémy a tím tuto nevýhodu potlačují (Pulier, Taylor, 2006, s. 43). To snižuje náklady nutné na lidské zdroje pro vývoj a testování změn, ale také na následnou údržbu jednotlivých systémů.

EAI produkty typicky umožňují synchronní i asynchronní výměnu zpráv, transakční zpracování, obsahují podporu směrování či pozdržení zpráv, škálovatelnost a vysokou dostupnost. Nevýhodami jsou naopak vysoké investiční náklady, časová prodleva při zavádění do praxe, nízká flexibilita poskytovaných rozhraní a nevýhody spojené s řízením a kontrolou při větším množství propojených služeb. Pulier a kol. (2006, s. XXXVI) mezi další nevýhody tohoto přístupu řadí nutnost udržovat si zaměstnance se specifickými znalostmi konkrétních technologií, které jsou používány v integrovaných aplikacích.

3.1.5 Event-driven architecture

Tento IT architektonický přístup je charakterizován vytvářením, detekcí, konzumací a reakcí na události v počítačových systémech. Událostí je zde myšlena významná změna stavu objektu reálného světa či taková událost, která je významná pro

provádění zamýšlené funkční logiky. IT systémy navržené dle EDA využívají asynchronní výměnu dat pro produkci a konzumaci událostí a zároveň by měly být schopné zpracovávat události v předem nedefinovaném pořadí.

Přístupy EDA a SOA jsou vzájemně komplementární a jejich vhodnou kombinací lze dosáhnout vyšší celkové flexibility podniku. Cummins (2009, s. 208) mezi důležité důvody užití obou přístupů uvádí, že SOA samotná není schopna identifikovat a reagovat na nezvyklé události. Pomocí EDA je naopak možné skládat běžné události do komplexních, ty následně analyzovat a vyhodnocovat. Tato metoda se nazývá Complex-Event Processing (CEP) a skládá se z průběžného monitorování, reportování, zaznamenávání a filtrování událostí, které v určité kombinaci mohou znamenat výskyt události, která je z pohledu dalšího zpracování zajímavá. Následně lze uplatnit flexibilitu SOA pro rychlé upravení či doplnění existujících business procesů v podniku tak, aby byly schopny standardně řešit i nově detekované situace.

3.1.6 Business Process Management

Business Process Management je manažerská disciplína zaměřující se na navrhování a průběžné vylepšování rychlosti, nákladů a kvality obchodních operací (Cummins, 2009, s. 74). Cummins dále uvádí, že tento přístup obnáší analýzu a vylepšování jak manuálních, tak i automatizovaných obchodních procesů, přičemž práce v podniku je typicky vykonávána určitými jednotkami a řízena procesy. Proces je zahájen na základě konkrétní potřeby či cíle, přičemž proces je definován prací, kterou je nutné vykonat, a také kým, kdy a proč má být vykonána (Cummins, 2009, s. 77). Jednotlivé kroky procesu přitom mohou být vykonávány manuálně jednotlivými rolemi pracovníků, anebo plně automaticky některým IT systémem.

Průběžná analýza procesů a jejich pružná optimalizace dává možnost managementu podniku provést strategické omezení nákladů či poskytnout příležitost pro generování nových tržeb. Aby tyto změny byly efektivní, je nutné je provést v zamýšleném časovém období (Pulier, a další, 2006, s. 96). Z pohledu analýzy efektivity procesu je pro management podniku přitom zajímavá možnost spekulativní analýzy návratnosti investovaných prostředků a předpokládaných tržeb hned při úvodním návrhu procesu. Po

implementaci a nasazení je poté naopak zajímavá zpětnovazební analýza na základě reálných dat. Některé softwarové produkty obě tyto možnosti dnes již plně podporují.

Užitím servisně orientovaných služeb při automatizaci obchodního procesu lze nejenom minimalizovat nutné náklady na realizaci, ale především snížit dobu uvedení nových či modifikovaných služeb na trh. To je dáno především autonomností, znovupoužitelností a komponovatelností jednotlivých služeb. Zároveň se zvyšuje schopnost managementu podniku se více soustředit na vlastní obchodní proces a zároveň přitom správně předvídat a chápat dopady požadovaných změn na IT infrastrukturu (Pulier, a další, 2006, s. 96).

3.1.7 Cloud Computing

Tento pojem zahrnuje oblast vývoje a používání počítačových technologií, kde je společným prvkem provoz části IT ve virtuálním prostředí na Internetu. Existuje několik obchodních modelů, jako příklad uvádí Svoboda (2009): IaaS - Infrastruktura jako služba, PaaS - Platforma jako služba a SaaS - Software jako služba. Společným znakem je vysoká dostupnost a obrovská škálovatelnost a elasticita, která umožňuje klientům rychle měnit využívané zdroje podle aktuální potřeby. Zároveň se klient nemusí starat o běžnou údržbu spojenou s výměnou a opravami hardware či o aktualizace poskytovaného softwarového vybavení.

Mezi nevýhody je možné zařadit úzkou vazbu na konkrétního poskytovatele služby (je složité a nákladné přejít k jinému poskytovateli), případný odlišný právní řád poskytovatele a klienta, rizika spojená s bezpečností ukládání citlivých dat mimo podnikovou síť a v neposlední řadě závislost na dostupnosti Internetu.

Cloud Computing není následovníkem SOA a stejně tomu není ani naopak. Oba přístupy se vzájemně doplňují a při vhodné kombinaci užití dokonce poskytují vyšší přidanou hodnotu. Lze například kombinovat služby poskytované v rámci SaaS se službami uvnitř podniku, stejně tak lze provozovat již existující služby nově v rámci IaaS.

3.2 Definice SOA

Následující kapitoly uvádí vybrané pohledy na definici podstaty servisně orientovaná architektury.

3.2.1 Základní vlastnosti

Počátky servisně orientované architektury se vztahují k postupné potřebě zavést nové standardy v oblasti počítačové výměny dat, a to především v období masivního rozšíření a užití Internetu. Mezi klíčové standardy a technologie řadí Pulier a kol. (2006, s. 21) především specifikace založené na XML a komunikační protokoly, které souvisí s univerzální interakcí mezi počítači. Tyto technologie umožňují takovou výměnu zpráv, že při jejich použití jsou komunikující aplikace na sobě mnohem více nezávislé – jsou „loose coupled“ (ve volné vazbě). Zároveň jsou výrazně nezávislé i na použité síťové infrastruktuře, jelikož pro přenos se často využívá rozšířený a podporovaný protokol HTTP[6]. Vzdálené užití služeb probíhá voláním publikovaných webových služeb, tedy zasíláním SOAP[7] zpráv a jejich zpracováním.

Servisně orientovaná architektura rozhodně nesouvisí pouze s technologiemi. Cummins (2009, s. 27) ji například definuje jako nové paradigma business designu se stěžejním zaměřením na vytvoření takové společnosti, která bude podporovat agilnost v oblasti rychlosti, nákladů a kvality. Dále uvádí, že SOA nutně nevyžaduje elektronické technologie, ale automatizace, integrace a modelovaní pomocí těchto technologií je základem pro její optimální implementaci. Plná implementace SOA ovšem také vyžaduje kompletní transformaci podniku.

Hlavním příslibem této nové architektury je schopnost kontinuálního vývoje a optimalizace podniku na základě potřeb a strategie podniku, a ne na základě omezení IT oddělení (Pulier, Taylor, 2006, s. 51). Pulier a kol. (2006, s. 52) dále uvádí, že jednotlivé prvky SOA mohou být přemístěny, nahrazeny či modifikovány, a to díky volné vazbě mezi jednotlivými službami. Dále je možné tyto jednotlivé služby skládat a spojovat do různých forem a za různým účelem, a tím vytvářet nové procesy či složené aplikace. Hlavním příslibem je tedy flexibilita změn.

---

[6] Hypertext Transfer Protocol: Síťový protokol využívaný především v internetovém prostředí. Vývoj protokolu zajišťují organizace IETF a W3C.

[7] SOAP: Specifikace komunikačního protokolu konsorcia W3C.

Erl (2009, s. 42) rozděluje SOA historicky na dvě období: Prvotní SOA a Současná SOA. Prvotní SOA se opírá o standardy a specifikace založené na XML a základních webových službách. Současná SOA staví na prvotním modelu a působením průmyslu a technologického pokroku rozšiřuje své původní ideje (Erl, 2009, s. 43), přičemž podle Erla (2009, s. 45) je základním znakem představa architektury, která podporuje servisní orientaci právě za použití webových služeb.

Tabulka 1 - Základní vlastnosti Současné SOA

Základní vlastnost Současné SOA	Charakteristika vlastnosti
Součást jádra servisně orientované výpočetní platformy.	SOA jde za hranice běžné IT architektury. Představuje servisně orientovaný princip fungování celého podniku.
Zlepšuje kvalitu služby.	Zavádění standardů souvisejících s prováděním služeb bezpečně, spolehlivě, dle požadavků na výkon a datovou integritu.
Zásadní autonomnost služeb.	Nezávislost a soběstačnost služeb formou samosprávy na úrovni výměny zpráv. Zprávy jsou dostatečně inteligentní, aby kontrolovaly způsob, jak je budou zpracovávat přijímající služby.
Je založena na otevřených standardech.	Přenos i zpráva samotná jsou založeny na obecně přijatých standardech, které navíc nevyžadují, aby jednotlivé služby sdílely typový systém. Přemosťuje většinu nestejnorodostí uvnitř podniku a lze tedy vždy zvolit možnost komunikace mezi službami.
Podporuje rozmanitost prodejců.	Standardizovaný systém komunikace umožňuje širší výběr produktů nezávisle na stávajícím softwarovém vybavení podniku.
Stará se o vnitřní spolupráci.	Při tvorbě aplikací lze aplikovat takové principy návrhu, které vybaví služby vlastnostmi, které přirozeně podporují spolupráci. Služby tedy bude možné začlenit do budoucích požadavků na integraci aplikací.

Podporuje zjistitelnost.	Podporuje a doporučuje používání takové formy registru služeb, která podporuje oznamování a zjišťování služeb v podniku i mimo podnik.
Podporuje federace.	Umožňuje zapouzdřit zastaralou a nezastaralou aplikační logiku a vystavit ji pomocí standardizovaného systému komunikace. Umožňuje zavést jednotnost do dříve nesdružených podniků a propojit je.
Podporuje princip architektonické kompozice.	Služby jsou nezávislými jednotkami logiky, ze kterých lze skládat automatizované řídící procesy. To podporuje i zavádění standardů pro podporu flexibilní kompozice (WS-*).
Stará se o základní znovupoužitelnost.	Důraz na návrh služeb dle principu znovupoužitelnosti, i když takový přímý požadavek neexistuje. Služby musí být neutrální k řídícím procesům i řešením automatizace, které je upotřebí.
Zdůrazňuje rozšiřitelnost.	Důraz na takový návrh rozhraní služeb, aby úroveň granularity rozhraní nelimitovala rozšiřování funkcionality. Službu poté lze rozšířit o dodatečnou logiku při minimalizaci negativních dopadů.
Podporuje servisně orientovaný modelový vzor řízení.	Služby lze navrhnout tak, aby vyjadřovaly řídící logiku. Modely BPM a modely entit lze přesněji reprezentovat pomocí uspořádaného sjednocení služeb pro řízení.
Implementuje vrstvy abstrakce.	Umožňují zavádění vrstev abstrakce nastavením služeb jako výhradních přístupových bodů k různým prostředkům a procesní logice. Správným návrhem tedy lze oddělit řídící a aplikační domény podniku. Každý koncový bod poté musí vědět pouze o existenci dalšího, což dovoluje každé doméně vyvíjet se více nezávisle. Toto podporuje schopnost přizpůsobovat se změnám v řízení a technologii.
Podporuje organizační	Působením na řídící reprezentaci služeb, abstrakci služeb a především volnou vazbou mezi řídící logikou a aplikační logikou

flexibilitu.	nabízí potenciál ke zvýšení organizační agilnosti. Jednotlivé automatizované části podniku jsou poté mnohem více na sobě nezávislé, což se nejvíce projevuje během vnitřní reorganizace, sloučení společností, změně v obchodním poli působnosti či při náhradě zavedené technologické platformy v podniku.
Je stavebním blokem.	Často bude SOA reprezentovat pouze jednu z mnoha užitých architektur v podniku a je tedy stavebním blokem poskytujícím služby. Servisně orientovaný podnik naopak považuje SOA za podnikový standard.
Je vývojem.	Využívá úspěšně vlastnosti předchozích distribuovaných architektur a přebírá jejich úspěšně vlastnosti. Liší se od nich návrhem a principy servisní orientace a webových služeb, a také užitím nových návrhových vzorů a technologií. Jmenovitě znovupoužitelnost, zapouzdření, komponovatelnost a abstrakce aplikační a řídící logiky.
Stále dozrává.	Stále existují omezení v rámci funkcionality webových služeb, které limitují nasazení v rámci podniku. Tato omezení jsou ovšem dočasná, jelikož standardizační organizace postupně vytvářejí dodatečné specifikace, které zahrnují jejich řešení.
Je dosažitelným ideálem.	Při zavádění servisně orientované architektury v podniku dojde k hybridnímu přechodnému prostředí, kdy bude společně existovat distribuované prostředí složené z původního a servisně orientovaného řešení. Technologie umožňující implementaci jsou dostupné již dnes a ani výše zmíněný přístup není tedy limitem.

Zdroj: (Erl, 2009, s. 44-56)

Výše uvedené vlastnosti Erl (2009, s. 55) uvádí ve své Formální definici Současné SOA:

- Představuje otevřenou, rozšiřitelnou, federační a komponovatelnou architekturu, která podporuje servisní orientaci a skládá se ze služeb, které jsou

autonomní, schopné kvality (QoS), podporují rozmanitost prodejců, spolupráci, zjistitelnost a jsou implementované jako webové služby.

- Může zavést abstrakci řídící logiky a technologie, což vede k volné vazbě mezi těmito doménami.
- Vyvinula se ze starších platforem, zachovává užitečné vlastnosti tradičních architektur a přichází s různými principy, které se starají o servisní orientaci na podporu servisně orientovaného podniku.
- Je ideálním univerzálním standardem v rámci celého podniku, ale dosažení tohoto stavu vyžaduje přechod a podporu na stále se vyvíjející sady technologií.

3.2.2 Principy servisní orientace služeb

Servisní orientace služeb je základním pojmem, který vychází z potřeby oddělit jednotlivé zájmy na dílčí části, které spolu souvisí (Erl, 2009, s. 242). Tím lze dosáhnout rozdělení komplexních problémů na sadu menších problémů, které již lze izolovaně řešit.

Jedná se vlastně o jinou implementaci již použitého konceptu, na kterém staví objektově orientované programování a programování založené na komponentách (Erl, 2009, s. 242). Implementace samotná se přitom řídí již uvedenými vlastnostmi Současné SOA.

Jednotlivé následně uváděné principy spolu souvisí a vzájemně se podporují (Erl, 2009, s. 267). Některé principy servisní orientace jsou odvozeny z principů objektové orientace (Erl, 2009, s. 268) a některé jsou přirozenými vlastnostmi webových služeb. Mezi takové lze zařadit abstrakci služeb, komponovatelnost, volnou vazbu a dohodu služeb (Erl, 2009, s. 270). Ostatní vlastnosti nevyplývají přímo jako výhody užití webových služeb, a je proto nutné se na ně zaměřit v rámci návrhu a modelovaní služeb.

Tabulka 2 – Běžné principy servisní orientace služeb

Princip služby	Zaměření a cíle
Služby jsou opětovně použitelné.	Služby jsou zkoumány a navrhovány z pohledu znovupoužitelnosti, a to bez ohledu na okamžité příležitosti užití.

Služby sdílejí formální dohodu.	Za účelem interakce služeb nemusí služby sdílet nic jiného než formální dohodu, která definuje všechny služby a podmínky pro výměnu informací.
Služby jsou volně vázané.	Služby musí být navrženy pro interakci, aniž by musely existovat úzké vazby mezi službami.
Služby abstrahují logiku v pozadí.	Jediná část služby, která je viditelná okolnímu prostředí, je vystavena prostřednictvím dohody služeb. Logika v pozadí, která netvoří část dohody služeb, je pro žadatele neviditelná.
Služby jsou komponovatelné.	Služby se mohou slučovat s dalšími službami. To umožňuje reprezentovat logiku na různých úrovních granularity a podporuje znovupoužitelnost a tvorbu vrstev abstrakce.
Služby jsou autonomní.	Logika řízení službou je omezena explicitní hranicí. Služba má kontrolu nad vším uvnitř své hranice a dokončení její úlohy není závislé na jiných službách.
Služby jsou bezstavové.	Služby by neměly být nuceny uchovávat informaci o stavu, jelikož to může ovlivnit jejich schopnost zůstat volně spojené. Návrh služeb proto musí udržovat služby maximálně bezstavové a případně přesunout správu informací o stavu do jiné úrovně.
Služby jsou zjistitelné.	Cílem je zpřístupnit definici služeb, aby je mohli objevit žadatelé služeb, čímž se zvýší jejich znovupoužitelnost.

Zdroj: (Erl, 2009, s. 242-243)

3.2.3 Využitelnost SOA

Z pohledu vývoje v čase může být SOA vnímána jako přístup, který nepřinesl očekávané výsledky. Většinou se ovšem jedná o názory, které souvisí s chybným nastavením očekávání či přímo se špatnou aplikací SOA, kdy hlavním přínosem SOA není zakoupený software, ale především aplikování definovaných principů a postupů (Handy, 2010) a také zavedení politiky, správy a údržby služeb - tzv. Governance (Gartner, 2009).

Vlastní definice principů (Best practices) SOA ovšem přesahuje hranice SOA a je využitelná i v dalších oblastech návrhu a vývoje software. Gartner (2009) uvádí, že výhoda SOA se mnohem šířeji projevuje v kombinaci s jinými přístupy a technologiemi. Realizací obchodních procesů BPM či komplexních událostí v rámci EDA lze zvýšit agilnost a flexibilitu podniku a jeho procesů. Handy (2010) dále uvádí, že z pohledu businessu je dnes nutné vnímat cloud jako prostředí, ve kterém je aplikování SOA nanejvýš vhodné a kde lze také těžit z dosažení vyšší flexibility a nezávislosti jednotlivých služeb.

Právě z těchto důvodů zůstává i nadále SOA stále platnou a žádanou oblastí, což potvrzuje i výzkum provedený společností Forrester (2011).

3.3 Pohled na implementaci SOA v podniku

Tato kapitola uvádí různé pohledy na zavádění servisně orientované architektury a její následné provozování v rámci podniku.

3.3.1 Kategorizace

Zavedení a rutinní užití servisně orientované architektury je dlouhodobý proces, který vyžaduje účast všech rolí v podniku a který přináší celou řadu problémů a omezení, na které je nutné brát zřetel. Úvodní fáze tohoto procesu souvisí s rozhodováním a určením způsobu, jakým bude SOA v dlouhodobém časovém období implementována. Často se zde prolínají vlivy konkrétních potřeb a očekávání managementu, stejně tak i architektonické představy o cílovém řešení. Role jednotlivých pracovníků účastnících se tohoto procesu tedy předurčuje, jakým pohledem budou nahlížet na celý proces implementace. Následující kapitoly analyzují jednotlivé aspekty ovlivňující výslednou servisně orientovanou architekturu podniku.

3.3.2 Fáze zavádění SOA

Každé změně v oblasti informačních technologií v rámci podniku předchází rozhodnutí, zda danou změnu provést či nikoli. V případě prohlubování architektury integrace systémů se navíc jedná o takovou změnu, kde není očekáván okamžitý přímý obchodní přínos, ale naopak jsou zde očekávány snížené náklady z dlouhodobého hlediska.

Zavádění servisně orientované architektury lze rozdělit do jednotlivých etap a fází. Rozdělení uvedené v následujících odstavcích volně navazuje na postup uváděný Pulierem a kol. (2006).

3.3.2.1 Rozhodnutí o zavedení SOA

Zavedení servisně orientované architektury v podniku je dlouhodobým strategickým rozhodnutím managementu, na které se váže i řada negativních dopadů. Klíčových impulzů pro úvahy o zavedení této podpory je několik. Dle Puliera (Pulier, Taylor, 2006) to může být sloučení dvou společností a návazného sjednocování obchodních procesů a IT technologií. Cílem zde je maximalizace synergických efektů. Stejně tak prvotním důvodem může být investice do větší flexibility podniku, ať už při zavádění nových produktů či z důvodu nutnosti hbité reakce na produkty konkurence. V neposlední řadě je nutné vyhodnocení návratnosti investovaných prostředků (ROI – Return Of Investment).

Mezi negativní dopady takto rozsáhlých změn lze zařadit rizika spojená s nutností komplexních změn v pracovních procesech podniku. Dále pak rizika s případným nenaplněným očekáváním či přímo s neúspěchem celého projektu. Z těchto důvodů zmiňuje (Pulier, Taylor, 2006, s. 226) možnost přizvání externích expertů, kteří mají zkušenosti se zaváděním SOA v podnicích a kteří tak mohou snížit míru známých rizik.

Výstupem této fáze je strategické rozhodnutí managementu o zavedení SOA včetně stanovení hrubého plánu. Plán rozdělí implementaci do jednotlivých etap a fází, kde hlavním cílem první (pilotní) etapy je snaha identifikovat a eliminovat negativní dopady rozhodnutí, ověřit životaschopnost projektu a následně upřesnit návratnost dále investovaných prostředků.

3.3.2.2 Základní analýza potřeb a volba strategie zavádění

Pro úspěšnou implementaci je zapotřebí nejprve provést analýzu potřeb, problémů a cílů podniku jak v oblasti IT, tak i v oblasti zaběhlých pracovních procesů a postupů. Cílem je vytvořit úvodní seznam potřebných webových služeb a z nich následně i vizi budoucího servisně orientovaného IT prostředí. Takto identifikované služby je nutné dále seskupit do funkčně logických celků, které budou reprezentovány jednotlivými IT systémy.

Strategie zavádění by měla ovšem obsahovat i schopnost přizpůsobovat plány zavádění změněným podmínkám v podniku. Bloomberg (2010) varuje před tzv. "Checklist" architekturou, kdy vlastní implementace SOA se může stát pouhým vykonáváním stanovených cílů, a to i na úkor agilnosti podniku. Jedním z hlavních cílů SOA je přitom právě zvýšení agilnosti, a proto by strategie měla s tímto specifickým aspektem již od počátku počítat. Z důvodů měřitelnosti výsledků a celkové efektivity je však nutné stanovit alespoň hlavní milníky a cíle tak, aby odpovídali cílovým potřebám podniku (Bloomberg, 2010). Klíčovou vlastností strategie je proto měnitelnost v čase dle změněných potřeb podniku.

Erl (2009, s. 299-308) uvádí dva validní přístupy pro zavádění SOA: Strategie shora-dolů a Agilní strategii. První přístup vede k velmi kvalitní servisní architektuře, jelikož celý návrh je nejprve důkladně analyzován a je tak maximalizována znovupoužitelnost služeb. Nevýhodou je časová a finanční náročnost z důvodu hluboké úvodní analýzy. Druhý přístup je podobný prvnímu s tím rozdílem, že práce na realizaci jednotlivých služeb je zahájena jakmile analýza shora-dolů dodatečně pokročí. Tato strategie splňuje dlouhodobé i krátkodobé cíle, přičemž celkově zvyšuje úsilí spojené se zavedením každé služby. Služby jsou totiž při použití této strategie opětovně revidovány, navrhovány a vytvářeny a je tedy nutné průběžně zajišťovat kompatibilitu s celkovým řešením. Erl (2009, s. 302) dále uvádí přístup zdola-nahoru, který je často využívaným, ale který nevede k vytváření služeb splňujících zásady servisní orientace. Z těchto důvodů označuje tento přístup za krajně nevhodný, jelikož v důsledku vede k nutnosti dodatečné revize modelu služeb a k jejich následné re-implementaci. To způsobuje dodatečné výrazné zvýšení nákladů.

3.3.2.3 Volba technologií pro implementaci

Lze zvolit různé technologie a produkty pro implementaci Současné SOA. Takové rozhodnutí však musí brát v úvahu situaci v konkrétním podniku, jelikož je nutné jak minimalizovat ekonomické a organizační dopady, tak i naplnit definovanou servisně orientovanou vizi.

Mezi významná kritéria rozhodování lze tedy zařadit omezení daná investičním záměrem, stávajícím IT prostředím podniku a vědomostmi a schopnostmi zaměstnanců.

Některá z uvedených omezení lze odstranit jako součást realizace implementace SOA. Příkladem může být upgrade dosud používaných IT systémů na verzi, která je již servisně orientovaná (Pulier, Taylor, 2006, s. 227). Takováto komplexní rozhodnutí jsou tedy stále z velké části na úrovni managementu, jelikož se týkají podniku jako celku.

Z pohledu servisně orientované architektury lze využívat libovolné technologie či produkty, které jsou stavěny na bázi uznávaných standardů. Je nutné ovšem brát v úvahu dodatečné náklady, které vznikají nutností zaměstnávat více pracovníků se znalostí konkrétních produktů a technologií.

V této fázi je nutné uvažovat také nad architekturou z pohledu:

- potřeb na správu a údržbu služeb a jejich verzí,
- požadavků na zabezpečení kvality přenosu zpráv,
- požadavků na zabezpečení integrity a důvěrnosti obsahu zpráv,
- požadovaného sledování chodu služeb a následného vyhodnocování,
- požadované výkonnosti.

3.3.2.4 Pilotní projekt

Kritickou fází je realizace pilotního projektu, který má za cíl ověření procesních a technologických předpokladů a zároveň identifikaci různorodých rizik a negativních dopadů.

Z tohoto důvodu je nutné nejprve vybrat vhodnou oblast podniku, na které bude možné uvedené předpoklady a dopady ověřit. Je vhodné volit z takových oblastí, kde nedochází k přílišným vnějším vlivům a které nejsou příliš komplexní. To usnadní identifikaci problémů souvisejících pouze s pilotním projektem a zároveň usnadní průběžnou interpretaci výsledků. Nevhodné jsou tedy ty části podniku, které jsou zatíženy akutními problémy či častou potřebou změny. Následujícím krokem je vytvoření architektury a designu nového systému pro zvolenou oblast a vlastní implementace.

Závěrem pilotního projektu může být i rozhodnutí o zastavení zavádění SOA. Z tohoto důvodu Pulier (Pulier, Taylor, 2006, s. 185) zdůrazňuje, že tato architektura je stavěna na standardech, a proto je proces zavádění plně reverzibilní.

3.3.2.5 Vyškolení zaměstnanců

Nezbytným předpokladem pro úspěšnou implementaci je pochopení základních principů SOA jednotlivými zaměstnanci a nutnost předejít případným negativním reakcím (Pulier, Taylor, 2006, s. 188-196). Stejně tak důležité je seznámit je se strategickými rozhodnutími a dlouhodobými plány managementu.

3.3.2.6 Postupná implementace

Implementace servisně orientované architektury není jednorázová operace, ale dlouhodobý proces, který v prvních fázích postupně zavádí a standardizuje jednotlivé části IT prostředí podniku. Je sice vhodné postupovat po jednotlivých krocích přesně dle stanoveného plánu, ale zároveň je nezbytné reagovat na nové situace a změny v okolním prostředí. Jelikož servisní orientace poskytuje dodatečnou úroveň flexibility, tak je většinou možné tento dlouhodobý plán rozšiřovat či měnit, a to dokonce bez výrazných negativních dopadů.

3.3.3 Charakteristika vrstev servisně orientovaných služeb

Pro bližší pochopení způsobu modelování servisně orientovaných služeb je nutné nejprve identifikovat jednotlivé kategorie servisně orientovaných služeb. Služby se dělí do jednotlivých vrstev podle svého logického významu, souvislostí s aplikační či řídící logikou a svým umístěním za účelem podpory agilnosti Erl (2009, s.275). Erl (2009, s.277) uvádí následující tři vrstvy služeb:

- vrstva služeb aplikace,
- vrstva služeb řízení,
- vrstva služeb instrumentace.

Vrstva služeb aplikace záměrně abstrahuje logiku, která nesouvisí s řízením, do sady služeb, které lze označit jako služby aplikace (Erl, 2009, s. 278). Takové služby nabízí funkce v určitém kontextu zpracování, jsou nezávislé na řešení, jsou obecné a znovupoužitelné, lze je použít pro přímou integraci s dalšími službami aplikace a mohou se skládat z kombinace služeb vyvinutých interně v rámci podniku či služeb od prodejců,

které byly zakoupeny anebo pronajaty (Erl, 2009, s. 279). Tato vrstva je také často nazývána vrstva služeb užitku anebo služby infrastruktury (Erl, 2009, s. 280).

Vrstva služeb řízení představuje kolekci služeb, jejichž úkolem je vykonání řídící logiky. Služby řízení za tímto účelem využívají a spojují jednotlivé služby aplikace. Erl (2009, s. 282) představuje následující dva typy služeb řízení:

Služby řízení vztahující se k úloze (služby úlohy) zapouzdřují logiku řízení určenou pro konkrétní úlohu řízení nebo pro řídící proces zahrnující dvě a více entit řízení. Tyto služby jsou často málo znovupoužitelné, jelikož se vztahují ke konkrétní úloze.

Služby řízení vztahující se k entitě (služby entity) naopak zapouzdřují logiku zpracování spojenou s určitou entitou řízení. Služby entity jsou často znovupoužitelné a nezávislé na řídícím procesu.

Erl (2009, s.318, s.325) uvádí, že neexistuje jednotný způsob odvození služeb řízení, avšak lze je odvodit z běžných dokumentů modelu řízení, včetně modelů případů užití, definic řídících procesů a modelů entit.

Mezi výhody služeb řízení řadí Erl (2009, s. 315-317) především znovupoužitelnost, přínosy související s vyšší agilností v modelu řízení a dále s přípravou procesu na instrumentaci. Služby řízení jsou nezbytné pro dosažení standardizace servisní orientace v celém podniku.

Vrstva služeb instrumentace neboli vrstva rodičovských řídících procesů sdružuje služby, které umožňují přímé propojení logiky procesu s interakcí služeb a tedy reakce při průběhu práce (Erl, 2009, s. 283). Tento přístup kombinuje modelování procesu řízení se servisně orientovaným modelováním a návrhem. Tyto služby lze také označit jako služby řízených úloh, které umožňují skládání služeb do větších dlouhotrvajících procesů.

3.3.4 Postup při modelování služeb a procesů

Součástí vývojových projektů pro servisně orientovaná řešení jsou fáze servisně orientované analýzy a vytvoření servisně orientovaného návrhu (Erl, 2009, s.296). Obě tyto oblasti představují řadu komplexních úkonů, které svým rozsahem významně přesahují rámec této práce. Z tohoto důvodu je v následujících kapitolách uveden pouze přehled a základní charakteristika jednotlivých činností nutných při modelování servisně

orientovaných služeb. Servisně orientovaný návrh je v následujících kapitolách logicky oddělen do částí pro servisně orientované služby a pro obchodní procesy.

3.3.5 Servisně orientovaná analýza

Výstupem servisně orientované analýzy je identifikace a vytvoření jednotlivých kandidátských služeb včetně kandidátských operací a jejich zařazení do příslušných vrstev služeb. Zároveň v této fázi dochází k rozhodnutí o rozsahu SOA v daném podniku (Erl, 2009, s.296).

Erl (2009, s. 311-313) dělí tuto fázi na 3 hlavní kroky:

- Definice rozsahu analýzy,
- Identifikaci stávajících systémů automatizace,
- Modelování kandidátských služeb.

První krok analýzy nejprve určí rozsah zamýšlené automatizace s ohledem na definované požadavky managementu. Následující krok identifikuje stávající aplikační logiky, které již některou oblast automatizují a které by bylo případně vhodné zapouzdřit a znovupoužít. Třetím krokem je oblast modelování služeb, kde dochází k uspořádání shromážděných informací a jejich postupnému vyhodnocení.

Modelování služeb je základní proces shromažďování a uspořádávání informací řídícího modelu, který je dekomponován do řady samostatných kroků, ve kterých jsou identifikovány kandidátské operace kandidátských služeb (Erl, 2009, s.341).

Samotné modelování kandidátských služeb je proces o 12 krocích, kdy postupně dochází k dekompozici řídících procesů, identifikaci kandidátů operací řídících služeb, abstrakci instrumentační logiky, vytvoření kandidátů řídících služeb, zaměření se na servisní orientaci identifikovaných služeb, identifikaci kandidátských kompozic služeb, přeskupení operací řídících služeb, analýze aplikačních procesních požadavků, identifikaci kandidátů aplikačních služeb, vytvoření kandidátů aplikačních služeb, přezkoumání kandidátů kompozic služeb, přeskupení operací aplikačních služeb a případně i k údržbě inventáře kandidátů služeb (Erl, 2009, s.328-340).

V rámci modelování je nutné dbát na pravidla, která zajišťují znovupoužitelnost a explicitní hranice služeb, což se projevuje ve výsledné kvalitě modelu jako celku. Klíčovým požadavkem pro efektivní modelování obchodních procesů je důkladná znalost logiky obchodního procesu organizace (Erl, 2009, s.348).

3.3.6 Servisně orientovaný návrh služeb

Servisně orientovaný návrh je proces, kterým se od logických služeb odvozují konkrétní fyzické návrhy služeb, které jsou následně seskládány do abstraktních kompozic, jež implementují daný obchodní proces (Erl, 2009, s.367). Mezí cíle návrhu Erl (2009, s. 368) jmenovitě zařazuje: určení základní sady architektonických rozšíření, stanovení hranic architektury, identifikaci požadovaných návrhových standardů, definování abstraktních návrhů rozhraní služeb, identifikaci potenciální kompozice služeb, ustanovení podpory pro principy servisní orientace a prozkoumání podpory pro charakteristiky Současné SOA.

Erl (2009, s.370-371) tento proces dělí na následující hlavní oblasti a dále je charakterizuje (Erl, 2009, 391-505):

3.3.6.1 Volba vrstev služeb a průmyslových specifikací

Součástí servisně orientovaného návrhu je rozhodnutí, které vrstvy služeb budou v podniku implementovány a z jakých průmyslových specifikací bude SOA zkomponována. V první oblasti se jedná především o klíčovou otázku, zda investovat do budování řídících služeb. Mezi klíčové faktory přitom patří výkon, nasazení a správa služeb. Vstupem pro samotné rozhodnutí proto mohou být tato upozornění (Erl, 2009, s. 394-395):

- Stávající konfigurace: Pokud již byly vrstvy služeb standardizovány, je obvykle vhodné se maximálním způsobem pokusit přizpůsobit návrh nových služeb těmto vrstvám.
- Požadované standardy: Při budování nových typů služeb či vrstev se službami je nutné také pro ně definovat návrhové standardy, které budou aplikovány i v budoucnu.

- Výkon kompozice služeb: Je nutné brát na zřetel potřeby z pohledu výkonnosti. Kompozice služeb může s sebou nést značnou zátěž při zpracování, především validaci, deserializaci a serializaci XML dokumentů. Proto je někdy nevhodné aplikovat vícevrstvou konfiguraci služeb. Zároveň je nutné identifikovat kritické služby podniku, které by bylo z důvodů výkonnosti nevhodné realizovat pomocí webových služeb.
- Nasazení služeb: Při návrhu vrstev se službami, které vytvoří znovupoužitelné služby agnostické vzhledem k řešení, může v distribuovaném prostředí způsobovat problém zpoždění vyvolané centrálním uložením služeb. Řešení těchto problémů může vést ke zvýšení nároků na administraci.
- Správa verzí služeb: Jednotlivé znovupoužitelné služby jsou postupem času využívány více žadateli o službu. Po nasazení služby mohou být vyžadována rozšíření doplňující sadu funkcí služby, které povedou ke změně rozhraní služby. V takovém případě je nezbytné už v počátečním návrhu počítat s implementací systému správy verzí služeb.
- Řídící služby a návrhy XSD schémat: Při návrhu je vhodné předcházet potenciálním problémům s kompatibilitou mezi stávající sadou XSD schémat a plánovanými řídícími službami. Vhodné je se zaměřit a revidovat entitně zaměřená schémata vůči plánovaným entitně zaměřeným řídícím službám.
- Údržba řídících služeb: Při postupu dle agilní strategie uvedené v předešlých kapitolách, je nutné průběžně revidovat již existující služby. Vhodným způsobem je vytvoření administrativního procesu navazujícího na zmíněný systém správy verzí.

Další důležitou částí návrhu je volba průmyslových standardů. Erl (2009, s. 402) doporučuje užití specifikace WS-I Basic Profile[8] pro volbu ústředních specifikací SOA a jejich verzí. Především se jedná o verze specifikací XML, XML Schema[9], WSDL[10] a

[8] WS-I Basic Profile: Specifikace konsorcia WS-I, která poskytuje návod na zvýšení interoperabilitu základních specifikací okolo webových služeb.

[9] XML Schema: Specifikace konsorcia W3C pro definování struktury XML dokumentů.

SOAP. SOA samotná klade požadavky a stanoví preference, které dále ovlivňují využití a ustanovení prvků v rámci primárních specifikací.

Následujícím z kroků servisně orientovaného návrhu je volba rozšíření SOA. Lze zde vycházet z toho, které z charakteristik Současné SOA je nutné realizovat. Jednotlivá rozšíření jsou sice volitelná, ale mají významný vliv na poskytované a propagované charakteristiky služeb. Z těchto důvodů je nutné volbu provést před vlastním postoupením k návrhu rozhraní služeb. Naneštěstí jednotlivá rozšíření WS-* jsou různě vyzrálá a stabilizovaná, a také nejsou všechna plně podporována jednotlivými výrobci. Doporučením je proto prvně ověřit dostupnost podpory ze strany stávajících užitých produktů v podniku a stejně tak i rozmyslet dopady na budoucí rozšíření užití navrhovaných služeb mimo rámec podniku.

3.3.6.2 Definice návrhových standardů

Mezikrokem před zahájením vlastní fáze návrhu je dle Erla (2009, s. 409) nutnost stanovit návrhové standardy, které budou aplikovány napříč všemi částmi návrhu a které tak zajistí klíčové vlastnosti SOA: znovupoužitelnost, komponovatelnost a především pak pružnost. Návrhové standardy zahrnují (Erl, 2009, s. 459-466) pravidla pro:

- pojmenovávání koncových bodů služeb, jmen operací služeb, hodnot zpráv;
- aplikaci vhodné úrovně členění rozhraní, což souvisí s granularitou rozhraní a následnou znovupoužitelností a výkonností služeb;
- rozšiřitelnost rozhraní služeb, což souvisí s fyzickým návrhem služby a možnostmi jejího rozšíření dle budoucích požadavků;
- identifikaci potenciálních žadatelů služeb, což obnáší spekulativní analýzu vnějšího užití služby při jejím návrhu;
- použití modulárních WSDL dokumentů, což díky schopnosti importu umožňuje centrální správu a znovupoužitelnost XSD schémat;

[10] Web Services Description Language: Specifikace konsorcia W3C pro definování webových služeb.

- standardizaci užitých jmenných prostorů (namespaces) a dalších atributů vyměňovaných zpráv;
- dokumentaci služeb, což představuje důležitý doplněk umožňující žadatelům služby lépe pochopit a plně se připravit na vzájemnou spolupráci a zároveň tím podporuje zjistitelnost a znovupoužitelnost služby.

3.3.6.3 Návrh služeb

Pojmem návrh služeb představuje transformaci dříve vymodelovaných kandidátů služeb na konkrétní návrhy fyzických služeb. Vstupy do procesu návrhu tvoří kandidáti služeb vytvoření v rámci procesu modelování služeb během servisně orientované analýzy. V souladu s jednotlivými vrstvami služeb je doporučován následující postupný způsob návrhu:

- proces návrhu entitně zaměřených řídících služeb,
- proces návrhu aplikačních služeb,
- proces návrhu úlohově zaměřených řídících služeb,

Erl (2009, s. 408) uvádí, že během vlastního návrhu bude občas zapotřebí iterativně revidovat některé již provedené části návrhu. To je zapříčiněno dodatečnou identifikací potřeb, které v rámci správného návrhu služeb musí být implementovány předešlou vrstvou. Například po vytvoření počáteční sady návrhů aplikačních služeb se během návrhu úlohově zaměřených řídících služeb objeví dodatečná potřeba na další funkci, kterou aplikační vrstva neposkytuje. V takovém případě je nutné návrh aplikační vrstvy revidovat a případně přidat nové operace či dokonce zcela nové služby.

Při návrhu entitně zaměřených služeb může naopak nastat situace, kdy zvolená granularita jednotlivých operací nad danou entitou nevyhovuje z pohledu přílišné obecnosti (operace typu vytvoř, změn či smaž celou entitu), anebo naopak jsou nevýhodné z pohledu výkonu (operace na úrovni jednotlivých atributů entity). Pro správnou výbavu těchto služeb vhodnou sadou generických operací může být nutné provést spekulativní analýzu (Erl, 2009, s. 429). Bloomberg (2010) v tomto ohledu zmiňuje především nutnost zaměřit se během architektonického návrhu na potenciální změny navrhovaných služeb.

Pro každou vytvořenou abstraktní službu je potřeba formálně definovat všechny operace, každou vstupní a výstupní zprávu operace, a související typy XSD schématu používaných pro reprezentaci obsahu zprávy (Erl, 2009, s. 409).

3.3.7 Servisně orientovaný návrh obchodních procesů

Spojením již navržených služeb s logikou obchodních procesů vznikají formální proveditelné definice logiky pracovního toku, které je možné automatizovat. Samotné navrhování procesní služby vyžaduje návrh rozhraní služby a návrh definice procesu (Erl, 2009, s. 505).

Samotný postup návrhu procesní služby je podobný návrhu úkolově zaměřené řídící služby. Rozdíl spočívá v tom, že logika pracovního toku je abstrahována do samostatné definice procesu a je tedy nutné ji řešit zároveň s návrhem procesní služby (Erl, 2009, s. 482). Vstupy do návrhu tvoří všechny předešlé navržené řídící a aplikační služby a samotný obchodní proces, který je potřeba automatizovat. Abstrahování procesní logiky do procesních služeb dále umožňuje zvýšit nezávislost navrhovaných řídících a aplikačních služeb, jelikož procesní služba předpokládá vysoký stupeň správy stavu a tedy zvyšuje schopnost ostatních služeb být bezstavové (Erl, 2009, s. 467).

Návrh definice procesů vede k náležité interpretaci nashromážděných požadavků obchodních procesů a k jejich přesné implementaci. Návrh musí zároveň řešit všechny možné varianty aktivit procesu, včetně všech chybových stavů a neočekávaných situací, a musí na ně vhodně reagovat (Erl, 2009, s. 482). Lze využít dostupné analytické a modelovací nástroje, které umožňují grafickým způsobem vytvářet diagramy řídících procesů, jež představují požadavky na logiku pracovního toku. Následně lze takový diagram využít pro generování do syntaxe jazyka, který je již počítačově zpracovatelný. Erl (2009, s. 468) za tímto účelem uvádí jazyk WS-BPEL[11].

[11] WS-BPEL: Specifikace standardizační organizace OASIS představující jazyk pro automatizaci procesů.

3.4 Technologie

Tato kapitola představuje nejdůležitější technologie používané při implementaci servisně orientované architektury. Kapitola dále představuje technologie a prostředky, které servisní orientaci dále využívají a poskytují tak dodatečnou přidanou hodnotu.

3.4.1 Přehled

Servisně orientovaná architektura je koncept, který může být implementován takovými technologiemi, které vyhoví základním požadavkům a vlastnostem na servisní orientaci. Stávající implementace jsou často realizovány za pomocí specifikací, které jsou založeny na značkovacím jazyku XML a které umožňují standardizovanou výměnu zpráv mezi systémy. Specifikace pro XML, XML Schema, SOAP a WSDL se řadí mezi základní a jsou dnes běžně používané. Funkcionality webových služeb jsou dále rozšiřovány specifikacemi hromadně označovanými jako WS-* (Erl, 2009, s. 507). V následujících kapitolách budou představena rozšíření webových služeb související s adresováním, spolehlivou výměnou zpráv, zásadami a zabezpečením. Pulier a kol. (2006, s. 21) mezi základní specifikace pro SOA řadí také UDDI[12]. Tato práce tuto specifikaci v následujícím textu ovšem již neuvádí, jelikož rozvoj této specifikace byl organizací OASIS ukončen.

3.4.2 Úvod do základních specifikací a technologií

Obvyklá implementace servisně orientované architektury využívá následující základní specifikace pro implementaci webových služeb:

- XML: Obecně rozšířený a podporovaný značkovací jazyk, který představuje standardní formát pro výměnu informací mezi systémy.
- XML Schema: Specifikace pro definování struktury XML dokumentů. Nejvíce rozšířená implementace je v jazyce XSD[13], který je sám založen na XML.

[12] Universal Description Discovery and Integration: Specifikace standardizační organizace OASIS pro registrování a vyhledávání existujících služeb.

[13] XML Schema Definition: Představuje implementaci specifikace XML Schema, která je doporučována organizací W3C.

- SOAP: Specifikace přenosového protokolu využívaného pro přenos zpráv na bázi XML. Obvykle je pro účely přenosu zpráv webových služeb využíván v kombinaci s protokolem HTTP.
- WSDL: Jazyk na bázi XML pro počítačově zpracovatelnou definici webových služeb. Zprávy webových služeb jsou definovány pomocí jazyka XSD a přenos obvykle zajišťuje protokol SOAP.

3.4.3 Rozšíření webových služeb

. Webové služby ve své základní podobě neposkytují vlastnosti, které jsou většinou požadovány při implementaci v podnikové sféře. Webové služby ovšem podporují tzv. volitelná rozšíření, při jejichž využití jsou dané nedostatky odstraněny. Z důvodu omezení rozsahu práce je zde uvedena pouze základní charakteristika vybraných rozšíření webových služeb.

Tabulka 3 – Charakteristika vybraných rozšíření webových služeb

Název rozšíření	Základní charakteristika produktu
WS-Addressing	- Koncept odkazů koncových bodů a koncept hlaviček s informacemi o zprávách. - Vylepšuje inteligenci vloženou do zpráv SOAP a tím zvyšuje autonomii na úrovni zpráv (Erl, 2009, s. 189). - Nepřímo podporuje tvorbu stavových služeb, což jde proti běžnému servisně orientovanému principu bezstavovosti (Erl, 2009, s. 190).
WS-Reliable Messaging	- Umožňuje zvolit úroveň kvality a spolehlivosti přenosu zpráv. Zprávy jsou doručeny dle zvolené kvality nejvíce jednou, alespoň jednou či přesně jednou. - Zprávy odeslané s určitým požadovaným uspořádáním dorazí ve správném pořadí (anebo nastane chyba).
WS-	- Poskytuje prostředky pro připojení vlastností (pravidel, chování, požadavků

Policy	a předvoleb) k webovým službám (Erl, 2009, s. 207), tzv. zásady. Ty mohou být povinné či nepovinné. - Služby poskytují o sobě více informací a přitom si zachovávají nezávislost.
WS-Security[14]	- Ověření identity, autentizace a autorizace původce zprávy. - Zajištění integrity těla zprávy a její důvěrnosti za využití XML-Signature[15] a XML-Encryption[16] (Erl, 2009, s. 222).

Zdroj: autor, 2012

3.4.4 Enterprise Service Bus

Enterprise Service Bus (ESB) představuje koncept sběrnice, která umožňuje propojení aplikací v lokální síti pomocí webových služeb (Cummins, 2009, s. 13). ESB je decentralizovaný EAI middleware s podporou komunikačních standardů, který usnadňuje implementaci servisně orientované architektury. Smyslem integrace aplikací dle EAI konceptu je omezit přímé vazby mezi jednotlivými aplikacemi, což ESB plní vytvořením sdílené sběrnice služeb, která je jednotlivým aplikacím zpřístupněna. ESB typicky poskytuje celou řadu pokročilých funkcí, mezi něž lze zařadit:

- Nezávislost na konkrétním operačním systému či programovacím jazyku;
- Podpora standardizovaných technologií a specifikací s možností podpory proprietárních technologií při napojování na původní systémy;
- Podpora různých návrhových vzorů pro výměnu dat mezi systémy: Například synchronní a asynchronní výměny zpráv na bázi principů request/reply, fire-and-forget či publish/subscribe;
- Podpora pro směrování a adresování zpráv mimo jiné na základě obsahu zpráv či definovaných pravidel a zásad;

[14] WS-Security: Specifikace standardizační organizace OASIS reprezentující rozšíření webových služeb o podporu zabezpečení zpráv.

[15] XML-Signature: Specifikace organizace W3C pro zajištění integrity obsahu zpráv.

[16] XML-Encryption: Specifikace organizace W3C pro zajištění důvěrnosti obsahu zpráv.

- Podpora pro transformace a mediace zpráv: Podpora pro různé technologické či aplikační adaptéry, při transformacích z různých komunikačních protokolů či při mapováních mezi různými formáty zpráv;
- Pokročilé funkce při práci se službami: Validace a transformace zprávy, upřednostnění či pozdržení zpracování zprávy, agregace více služeb do nové služby, obohacování zprávy o dodatečné informace z jiných zdrojů, rozdělování a spojování více zpráv či podpora řešení výjimek;
- Správa služeb včetně jejich nasazení a verzování, monitorování, auditování, logování a měření jejich výkonu;
- Podpora zajištění kvality služeb včetně jejich bezpečnosti, zaručeného doručení zpráv, podpory transakcí či opožděného doručení zpráv v případě výpadku či přehlcení cílové aplikace,
- Realizace nových služeb složených užitím jiných již existujících služeb.

ESB představuje konkrétní prostředek pro implementaci servisně orientované architektury, jehož přidaná hodnota je v možnosti iterativního zavedení servisně orientované integrace systémů. Zavedením jednotné metodiky pro implementaci, testování a nasazování služeb, a zároveň při využití již implementovaných pokročilých funkcí ESB, lze z dlouhodobého hlediska užitím ESB produktů minimalizovat investované prostředky.

Dalším aspektem při implementaci integrace aplikací je užití již existujících a praxí prověřených návrhových vzorů. Pro vnitropodnikovou integraci lze například využít společný (kanonický) datový model, který podporuje abstrakci a volnou vazbu mezi službami. Tento návrhový vzor lze přirovnat k servisně orientovanému principu návrhu entitně zaměřených řídících služeb, přičemž hlavním rozdílem je orientace na strukturu dat napříč celým podnikem. Výhodou je minimalizace vývoje nutných transformací zpráv mezi aplikacemi, kdy pro jednotlivé služby postačuje pouze jednou implementovat transformaci do společného datového modelu, ale také celkové zvýšení flexibility při zavádění změn v jednotlivých aplikacích.

Při masivním nasazení ESB služeb je však nutné uvážit možné výkonnostní problémy způsobované užitím ESB jako centrálního bodu, přes který probíhá celá vnitropodniková komunikace.

3.4.5 Registrace a vyhledávání služeb

Registrace služeb navazuje na servisně orientovanou architekturu, kde cílem je umožnit jednotlivým aplikacím nalézt službu, kterou budou využívat. Za tímto účelem lze využít produkty stavějící na standardu UDDI či pozdějším ebXML[17].

Pro vyhledání vhodné služby je klíčová schopnost službu nalézt na základě požadované funkčnosti. Za tímto účelem je nutné, aby data v registru služeb byla strukturována a byla v jednotném formátu. Je dále nutné poskytovat dodatečná metadata o jednotlivých službách, především pak o chování služby včetně chování v případě chybových stavů.

Registrace a vyhledávání služeb dále umožňuje realizaci pokročilých vlastností, které mohou minimalizovat dopady změn ve velkých implementacích:

- validace a indexace obsahu metadat o službách,
- sledování změn a životního cyklu jednotlivých služeb,
- řízení přístupových práv k jednotlivým službám včetně podpory schvalovacích a obchodních procesů,
- notifikace o změnách registrované služby.

3.4.6 Service Component Architecture

Open SOA Collaboration (2011) je skupina neformálně sdružených předních výrobců software, která vydává specifikace zaměřující se na urychlení vývoje servisně orientovaných řešení. Mezi aktuálně vydávané specifikace patří Service Data Objects (SDO) a Service Component Architecture (SCA).

Cílem řady specifikací SDO je zjednodušit a sjednotit způsoby práce s daty v aplikacích a sjednotit přistupování k datům z různorodých zdrojů, jako jsou relační databáze, XML dokumenty či jiné podnikové systémy.

[17] Electronic Business using Extensible Markup Language: Sada specifikací organizace OASIS, které standardizují registraci služeb a jejich metadat.

Naopak cílem řady specifikací SCA je definovat model tvorby aplikací a systémů, které budou založeny na SOA. Vychází se zde přitom z myšlenky, že aplikace jsou komponovány z nových či již existujících služeb, které svým vhodným propojením splní požadované obchodní cíle. Na základě tohoto předpokladu specifikace formují programovací model, který poskytuje následující výhody:

- Možnost užití v různorodých programovacích jazycích či v jiných prostředcích včetně WS-BPEL;
- Oddělení obchodní logiky od detailů o volané službě;
- Schopnost jednotného způsobu volání služeb bez ohledu na skutečnou implementaci (synchronní či asynchronní výměna zpráv, s odpovědí či bez ní);
- Schopnost využít různorodé standardizované i proprietární komunikační technologie včetně webových služeb, EJB, JMS, JCA, RMI, RPC, CORBA a dalších;
- Schopnost definovat mimo aplikační logiku požadavky na kvalitu služby, včetně požadavků na bezpečnost, transakčnost či kvalitu výměny zpráv;
- Data mohou být reprezentována pomocí zmíněné specifikace SDO.

Hlavní výhodou užití SDO a SCA v servisně orientovaných řešeních je zvýšení produktivity práce vývojářů aplikací, a to sjednocením a celkovým usnadněním přístupu k existujícím službám či při definici služeb nových. Vývojáři se tedy mohou více zaměřit na vlastní obchodní logiku a méně na technologie výměny zpráv. Z tohoto důvodu jednotlivé nástroje pro modelování obchodních procesů dnes již plně integrují podporu SCA technologie.

3.5 Produkty

Tato kapitola představuje vybrané produkty, které podporují servisně orientovanou architekturu či produkty, které jsou dále využity v rámci vlastní práce. Kritérii pro výběr bylo obecné povědomí o daném produktu a zároveň povědomí o výrobci daného produktu. Uvedené produkty jsou seřazeny abecedně podle názvu.

3.5.1 Přehled

Všichni vybraní výrobci poskytují ucelenou obchodní řadu produktů, která umožňuje plnohodnotnou implementaci SOA. Realizaci servisně orientované architektury v podniku lze ovšem provést i za užití jen části produktů z uvedených řad, stejně tak díky standardizaci specifikací i jejich kombinací. Využití produktů jednoho výrobce ovšem přináší garanci funkční kompatibility a především pak návaznost poskytovaných funkcí mezi jednotlivými produkty a tedy i snadnější zavádění do rutinního provozu.

Tabulka 4 – Přehled společností a vybraných řad produktů

Společnost	Produktová řada	Základní charakteristika produktu
Software AG	webMethods	Řada produktů předního výrobce v oblasti SOA. Podpora SOA Governance, BPM či Complex-Event Processing.
Oracle	Oracle Fusion Middleware	Ucelená podpora SOA včetně SOA Governance. Návaznost na další produkty této společnosti.
IBM	IBM WebSphere	Široká podpora SOA včetně registru služeb a pokročilé správy služeb.
Progress Software	SOA Infrastructure	Komplexní SOA řešení zahrnující podporu ESB, BPM či Complex-Event Processing.
SOA Software	Comprehensive SOA Infrastructure	Řešení v oblasti SOA Governance včetně podpory plánování služeb či vyhodnocování návratnosti investic.
WSO2	WSO2 SOA Platform	Ucelená podpora SOA včetně podpory SOA Governance, ESB či BPM. Založeno na Open Source s komerční podporou.
TIBCO	TIBCO ActiveMatrix 3.0	Ucelená podpora SOA. Možnost optimalizace obchodních procesů pomocí zpracování komplexních událostí.
Red Hat	Red Hat JBoss	Jedna z nejmladších platforem pro implementaci

	Enterprise SOA Platform	SOA. Založeno na Open Source s komerční podporou.
Microsoft	Microsoft BizTalk Server	Základní podpora SOA s návaznosti na cloudové prostředí Windows Azure.

Zdroj: autor, 2012

4 Vlastní práce

Následující kapitoly uvádí praktickou ukázku implementace SOA.

4.1 Praktická aplikace podle zásad SOA

Pro vlastní práci byly stanoveny následující cíle:

- navrhnout IT architekturu podniku v souladu se SOA,
- navrhnout a implementovat ukázkové servisně orientované služby a
- implementovat automatizovaný obchodní proces využívající tyto služby.

Vlastní ověření bude probíhat úvodním návrhem minimální IT architektury podniku, která bude v souladu s principy SOA a která bude zároveň v dlouhodobém časovém období poskytovat dostatečnou flexibilitu změn.

Následně budou zvoleny produkty, které budou podporovat zvolené technologie a které z dlouhodobého hlediska zajistí nejenom stabilitu, vysokou dostupnost, škálovatelnost či podporu rutinního provozu, ale také případnou možnost přechodu k jinému produktu či technologii. Technologie a produkty přitom budou zvoleny tak, aby umožnily případné rozšíření pro chod v reálném provozu.

Cílem ukázkové aplikace SOA bude předvedení schopnosti flexibilního užití servisně orientovaných služeb. Za tímto účelem budou navrženy a implementovány ukázkové služby, které budou poskytovat vybrané funkce fiktivní banky. Budou navrženy operace pro správu údajů zákazníka a jeho účtů, možnost transakcí na jednotlivých účtech a podporu konverze měn. Ukázka si neklade za cíl realizaci všech funkčních a nefunkčních požadavků nutných pro reálný chod banky, ale minimální předvedení zvoleného výseku funkcionality banky při splnění principů servisní orientace.

4.2 Návrh IT architektury dle principů SOA

Základním rysem servisně orientované architektury je vytvoření takového prostředí, ve kterém mohou co nejjednodušeji vznikat, měnit se a zanikat služby jednotlivých IT systémů, a zároveň být přitom v co největší míře využívány jednotlivými IT systémy podniku. Životní cyklus jednotlivých služeb by přitom v ideálním případě neměl negativně

ovlivňovat okolní IT systémy, tedy způsobovat nutnost změn v těchto systémech či nedostupnost těchto systémů. Zároveň by jednotlivé služby měly být v co největší míře nezávislé na nevyžádaných změnách, které přímo nesouvisí s funkcionalitou dané služby.

4.2.1 Úvodní rozhodnutí o podobě SOA

Za tímto účelem je nutné nejenom zvolit vhodné produkty a vybudovat technickou infrastrukturu, ale také vytvořit vnitropodnikovou metodologii, která při správné aplikaci minimalizuje problémy spojené se spravováním různorodých IT systémů a služeb.

Základním rysem takové vnitropodnikové metodologie je definice pravidel pro správu služeb a pro návrh služeb (kapitola 3.3.6.2 Definice návrhových standardů). Především se jedná o nastavení standardů pro životní cyklus služeb včetně pravidel pro pojmenovávání jmenných prostorů a jednotlivých verzí služeb. Dále je nutné vynucovat centrální správu služeb a detailní popis funkcí jednotlivých služeb, a to za účelem jejich vyhledání a znovupoužití.

Z pohledu nastavení základních rysů IT architektury je tedy nutné zvolit takové technologie a produkty, které budou splňovat anebo podporovat výše zmíněné základní cíle. Typicky se jedná o produkty označované jako SOA Governance, které se primárně zaměřují na udržovatelnost IT architektury při masivním nasazení a využívání servisně orientovaných služeb. Základní vlastnosti vyspělých produktů jsou:

- podpora centrálního registru služeb,
    - včetně podpory popisu a vyhledatelnosti služeb,
    - včetně podpory definice a údržby životního cyklu a verzování služeb,
- podpora centrální správy nastavení pravidel pro zabezpečení a kvalitu služeb,
- podpora vazeb a závislostí mezi službami, včetně identifikace rizik a dopadů z pohledu dostupnosti, výkonnosti či plánovaných výpadků jednotlivých služeb či IT systémů.

Při návrhu IT architektury je doporučováno nejenom zakoupení konkrétního softwarového produktu podporujícího SOA Governance, ale nejlépe také aplikace již vypracované a praxí ověřené SOA metodologie, která je typicky doprovázena

konzultačními službami. Nasazení SOA je tedy nejenom o volbě vhodných produktů, ale především o volbě vhodného dodavatele, který zajistí analýzu a správné posouzení aktuálního stavu podniku, vyhodnocení a následné doporučení vhodného postupu nasazení SOA.

4.2.2 Návaznost na podporu integrace aplikací

Z pohledu dlouhodobého vývoje je nutné zajistit dostatečnou flexibilitu jednotlivých IT systémů a zároveň minimalizovat náklady spojené s touto potřebou. Mezi vybrané limitující faktory omezující flexibilitu lze zařadit:

- různorodost technologií a přístupů používaných v jednotlivých systémech,
- udržovatelnost systémů z pohledu vyvolaných změn,
- kontrola nad zabezpečením a kvalitou jednotlivých funkcí systémů,
- složitost z pohledu kontinuálního monitorování a vyhodnocování dostupnosti funkcí jednotlivých systémů.

Uvedená omezení lze minimalizovat užitím produktu Enterprise Service Bus. Při volbě architektury je však nutné zvážit, zda je užití ESB z dlouhodobého hlediska účelné. V současnosti stále více koncových aplikací poskytuje své funkce formou webových služeb, které jsou standardně dostupné i pomocí prostředků SOA. Užití ESB ovšem přináší dodatečné výhody spojené s centrální správou služeb (viz. kapitola 3.4.4 Enterprise Service Bus).

Pro účely ukázkové aplikace SOA byl stanoven cíl užít zvolený produkt ESB, přičemž vlastní práce se nezaměřuje na plné využití vlastností ESB, ale omezuje se pouze na předvedení základní synergie ESB se servisně orientovanou architekturou. Za tímto účelem budou vytvořeny jednoduché proxy služby, které zajistí centrální správu a monitoring koncových služeb v rámci podniku.

4.2.3 Volba produktů a technologií pro ukázkovou aplikaci SOA

S ohledem na velikost podniku lze zvolit jak volně dostupné produkty založené na Open Source, tak i uzavřené komerční produkty světových výrobců.

Pro účely ukázkové aplikace SOA byl stanoven cíl užít produktovou řadu WSO2 SOA Platform, která je založena na otevřeném kódu a volně šiřitelné Apache licenci. Tato řada produktů splňuje podmínku podpory SOA Governance a ESB, a zároveň je v případě potřeby rozšiřitelná o další potřebné oblasti. Pro realizaci komponovaného obchodního procesu bylo původně zamýšleno užití produktu WSO2 Business Process Server (WSO2 BPS), ale z následujících důvodů byl dodatečně zvolen produkt IBM WebSphere Process Server (IBM WPS):

- nedostatečná kvalita Open Source produktu WSO2 BPS, kdy i při realizaci jednoduchých obchodních procesů neustále docházelo k chybám při nasazení těchto procesů,
- předvedení schopnosti kombinovat produkty různých výrobců založených na standardizovaných technologiích a předvedení výhod užití SCA (viz. kapitola 3.4.6 Service Component Architecture).

Pro jednotlivé oblasti zamýšlené IT architektury byly zvoleny následující produkty a technologie:

Tabulka 5 – Produkty zvolené pro realizaci ukázkové aplikace SOA

Oblast	Produkt	Technologie
Registrace a správa služeb	WSO2 Governance Registry[18] 4.1.x	-
Servisně orientované služby	WSO2 Application Server 4.1.x	XML, XML Schema, SOAP, WSDL a JAX-WS
Enterprise Service Bus	WSO2 Enterprise Service Bus 4.0.x	XML, XML Schema, SOAP a WSDL
Automatizované obchodní procesy	IBM WebSphere Process Server 6.2	WS-BPEL, SCA, BPELJ

Zdroj: autor, 2012

[18] V textu bude nadále používán zkrácený název WSO2 GREG.

4.3 Ukázková aplikace SOA

Následující kapitoly uvádějí postup pro vytvoření servisně orientovaného prostředí, návrh a implementaci servisně orientovaných služeb a jejich následné užití v produktech WSO2 SOA Platform a IBM WebSphere Process Server.

4.3.1 Vymezení rozsahu práce

Z důvodu omezení délky vlastní práce byla učiněna následující rozhodnutí, která vymezují rozsah ukázkové aplikace SOA:

- jsou maximálně využívány pokročilé možnosti zvolených produktů pro generování zdrojových kódů a nastavení jednotlivých modulů, přičemž:
  - generovaný kód není vložen jako součást práce, ale místo toho jsou popsány kroky vedoucí ke stejnému výsledku,
  - je užito nedoporučovaného postupu návrhu služeb zdola-nahoru (viz. kapitola 3.3.2.2), přičemž zdrojový kód je navržen tak, aby finální webová služba splňovala principy servisní orientace.
- ukázky zdrojového kódu jsou vloženy jako přílohy a práce se na ně odkazuje,
- není uvedeno ověření, že implementované služby splňují jednotlivé principy servisní orientace služeb,
- implementované koncové služby jsou vystaveny pomocí proxy služeb na ESB,
- implementované služby (koncové i proxy) jsou zaregistrovány do WSO2 GREG,
- rozšíření webových služeb nejsou využita (WS-Security, WS-ReliableMessaging a podobně),
- kroky vedoucí k instalaci a nastavení vývojového prostředí nejsou uvedeny, jelikož jsou dostupné v instalačních materiálech jednotlivých produktů,
- obchodní proces implementuje základní ukázkový tok dat, což umožní jak dostatečné předvedení znovupoužitelnosti a komponovatelnosti implementovaných servisně orientovaných služeb, tak i demonstraci výhod užití kombinace technologií SCA a WS-BPEL.

4.3.2 Návrh služeb a jejich rozdělení do modulů

Jednotlivé požadované funkce banky je nejprve nutné analyzovat a vytvořit seznam kandidátských služeb, které bude nutné dále modelovat (viz. kapitola 3.3.5). Vlastní návrh operací jednotlivých služeb je proveden tak, aby byly splněny základní cíle ukázkové aplikace SOA, a to při současném zachování principů servisně orientovaného návrhu služeb (viz. kapitola 3.3.6) a servisní orientace (viz. kapitola 3.2.2).

Služby jsou implementovány v jednotlivých modulech, které představují fiktivní nezávislé IT systémy poskytující danou funkcionalitu:

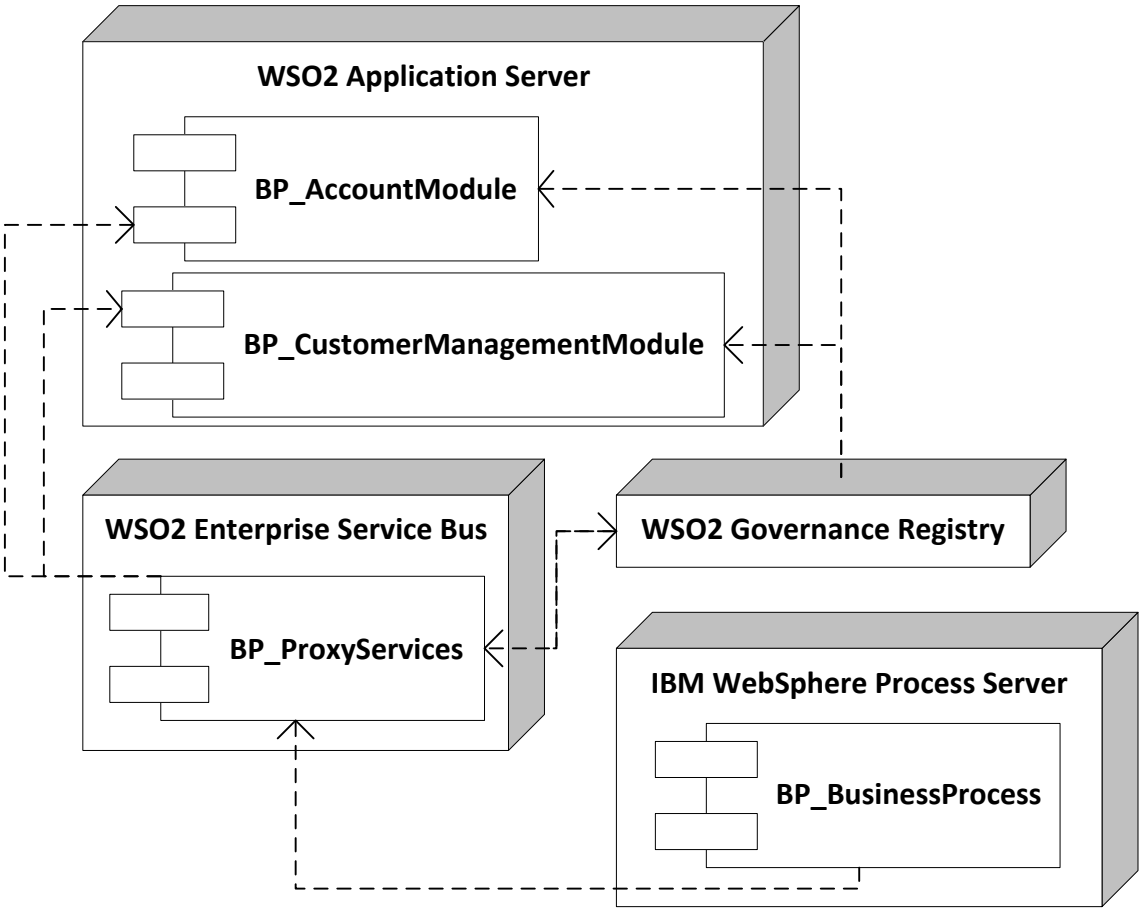
Tabulka 6 – Rozdělení funkcí ukázkové aplikace SOA do jednotlivých modulů

Název modulu	Funkce	Implementace
BP_AccountModule	Správa bankovních účtů	Vytvoření entitně zaměřené řídící služby poskytující základní operace s účtem zákazníka.
	Převody kurzů devizového trhu	Vytvoření úkolově zaměřené řídící služby poskytující operace pro práci s kurzy devizového trhu.
BP_CustomerModule	Správa zákazníků	Vytvoření entitně zaměřené řídící služby poskytující základní operace pro práci s údaji zákazníka.
BP_ProxyServices	Centrální správa služeb	Vytvoření ESB proxy služeb ke koncovým servisně orientovaným službám.
BP_BusinessProcess	Ukázka komponovatelnosti služeb	Vytvoření obchodního procesu za užití vytvořených služeb.

Zdroj: autor, 2012

Vytvořené moduly jsou nasazeny v rámci jednotlivých produktů následovně:

Obrázek 1 - Deployment diagram ukázkové aplikace SOA



Zdroj: autor, 2012

4.3.3 Vývojové prostředí

Jednotlivé produkty jsou velmi paměťově náročně. Z tohoto důvodu je zvoleno následující řešení, které nijak neomezuje výsledné nasazení v případném rutinním provozu a které naopak umožňuje vývoj všech částí řešení současně:

- Jednotlivé moduly jsou umístěny do stejné instance aplikačního serveru WSO2 AS,
- Všechny WSO2 produkty jsou umístěny v rámci jedné instance Java virtuálního stroje (formou OSGi komponent).

- Vysvětlení: WSO2 produkty jsou založeny na specifikacích OSGi Alliance[19] a jsou tedy plně modulární. Lze je tedy instalovat jako nezávislé moduly v rámci OSGi kontejneru a tím zvolit požadovanou kombinaci produktů.
- V rámci vývoje ukázkové aplikace SOA je tato vlastnost plně využita.

4.3.4 Vývoj modulu BP_AccountModule pomocí Axis2

Tento modul realizuje servisně orientované služby pro práci s účty zákazníků banky a pro práci s kurzy devizového trhu. Vlastní implementace je založena na využití prostředků WSO2 Application Server, kdy postačuje implementovat Java třídu a k ní vygenerovat vhodnou Axis2[20] konfiguraci. To lze provést užitím prostředků vývojového prostředí WSO2 Carbon Studio. Při nasazení aplikace je WSO2 AS produktem rozpoznána konfigurace Axis2 a je vygenerována příslušná webová služba.

Třída AccountService.java implementuje funkcionalitu pro práci s účty zákazníka. K tomuto účelu je využito datové třídy Account.java, která reprezentuje data konkrétního účtu. Pro zjednodušení ukázky jsou jednotlivé informace ukládány pouze do paměti, tedy při restartu modulu jsou data ztracena.

Třída CurrencyConvertorService.java implementuje funkcionalitu pro práci s kurzy devizového trhu. Pro zjednodušení ukázky jsou jednotlivé informace ukládány pouze do paměti a samotné devizové kurzy jsou generovány náhodným číslem.

Zdrojové soubory modulu jsou dostupné v kapitole Příloha A – Zdrojové soubory modulu BP_AccountModule.

4.3.4.1 Postup vytvoření konfigurace pro webové služby založené na Axis2

Vytvoření Axis2 konfigurace webových služeb je provedeno následujícím postupem v WSO2 Carbon Studio:

1. Vytvořit nový Carbon Application module s názvem BP_AccountModule

[19] OSGi Alliance – Konsorcium společností vydávající specifikace pro modularizaci software založeného na Java technologii.

[20] Apache Axis2 - Web Services / SOAP / WSDL engine.

2. Vytvořit Java třídy AccountSevice.java a CurrencyConvertorService.java
3. V menu zvolit „New / Axis2 Service / Apache Axis2 Service from class“ a vybrat AccountSevice.java (resp. CurrencyConvertorService.java)
4. Zkontrolovat, že byla služba vytvořena v daném Eclipse projektu
    - Existuje příslušný adresář v artifacts/services/axis2/

Axis2 defaultně generuje jmenné prostory WSDL a XSD schéma webové služby podle Java package jednotlivých tříd. Toto je možné změnit následovně:

5. Ručně editovat soubor resources/META-INF/services.xml umístěný v konfiguračním adresáři dané Axis2 služby
6. Změnit atribut targetNamespace na prvním řádku:
    - <service name="AccountService" targetNamespace="http://pef.czu.cz/soa/account/1.0">
7. Přidat nový element:
    - <schema schemaNamespace="http://pef.czu.cz/soa/account/1.0"/>

4.3.5 Vývoj modulu BP_CustomerModule pomocí JAX-WS

Tento modul realizuje servisně orientovanou službu pro práci s údaji zákazníků. Vlastní implementace je založena na využití prostředků WSO2 Application Server, kdy postačuje implementovat Java třídu společně s JAX-WS anotacemi a následně vložením JAX-WS artefaktu v rámci vývojového prostředí WSO2 Carbon Studio.

Třída Person.java, resp. Customer.java, představuje datový objekt, který reprezentuje konkrétní osobu, resp. zákazníka. XML anotace těchto tříd umožňuje kontrolu nad XML schématem v rámci generovaných JAX-WS webových služeb.

Rozhraní CustomerManagement.java společně s jeho implementací CustomerManagementServiceImpl.java tvoří webovou službu (pomocí JAX-WS anotací), která reprezentuje funkcionalitu pro práci s údaji zákazníka. Pro zjednodušení ukázky jsou jednotlivé informace ukládány pouze do paměti, tedy při restartu modulu jsou data ztracena.

Zdrojové soubory modulu jsou dostupné v kapitole Příloha B – Zdrojové soubory modulu BP_CustomerModule.

4.3.5.1 Postup vytvoření konfigurace pro webové služby založené na JAX-WS

Vytvoření Axis2 konfigurace webových služeb je provedeno následujícím postupem ve vývojovém nástroji WSO2 Carbon Studio:

1. Vytvořit nový Carbon module s názvem BP_CustomerModule
2. Vytvořit Java třídy rozhraní a implementace JAX-WS služby včetně anotací
3. V menu zvolit „New / JAX-WS Service"

4.3.6 Registrace webových služeb do produktu WSO2 GREG

Postup pro registraci webové služby do produktu WSO2 GREG:

1. Nasadit BP_AccountModule a BP_CustomerModule moduly na WSO2 AS
2. Získat URI odkazy na jednotlivé WSDL soubory webových služeb
3. Přihlásit se do konzole produktu WSO2 GREG
4. Zvolit v menu "Add WSDL"
5. Vložit URI odkaz a stisknout OK

4.3.7 Registrace endpointů služeb v produktu WSO2 ESB

Postup pro zaregistrování pojmenovaných endpointů v produktu WSO2 ESB:

1. Přihlásit se do konzole WSO2 ESB
2. Zvolit "Main / Manage / Service Bus / Endpoints / Add endpoint / Address endpoint" a zadat postupně jména koncových služeb a jejich endpointů pro jednotlivé služby. Příklad pro AccountService:
    a. AccountService_1_0
    b. https://localhost:9463/services/AccountService-1-0.AccountService-1-0HttpsSoap11Endpoint

4.3.8 Vývoj modulu BP_ProxyServicesModule

Tento modul implementuje transparentní ESB proxy služby, které dynamicky získávají WSDL soubor (z registru WSO2 GREG) a endpoint na koncovou službu (z WSO2 ESB). Implementace je realizována opět jako Carbon modul ve vývojovém prostředí WSO2 Carbon Studio. Postup vytvoření proxy služeb je pro všechny koncové služby obdobný, a proto je zde uveden pouze příklad pro CustomerManagementProxyService.

Samotné ESB proxy služby přitom splňují následující kritéria:

- nemění vstupní a výstupní strukturu definovanou koncovými službami,
- využívají registr služeb pro nalezení koncové služby, což zajistí:
  - centrální přístupový bod ke koncové službě bez ohledu na její fyzické umístění,
  - centrální správu pravidel pro bezpečnost a kvalitu služeb,
- zajišťují centrální logování a monitorování užití koncových služeb za předpokladu, že všechny klientské aplikace vždy využívají proxy služeb,
- samy jsou publikovány v registru služeb (za účelem případného vyhledání jinými službami).

Zdrojové soubory modulu jsou dostupné v kapitole Příloha C – Zdrojové soubory modulu BP_ProxyServices.

4.3.8.1 Vývoj CustomerManagementProxyService

Pro vytvoření modulu a proxy služeb je nutné postupovat následovně:

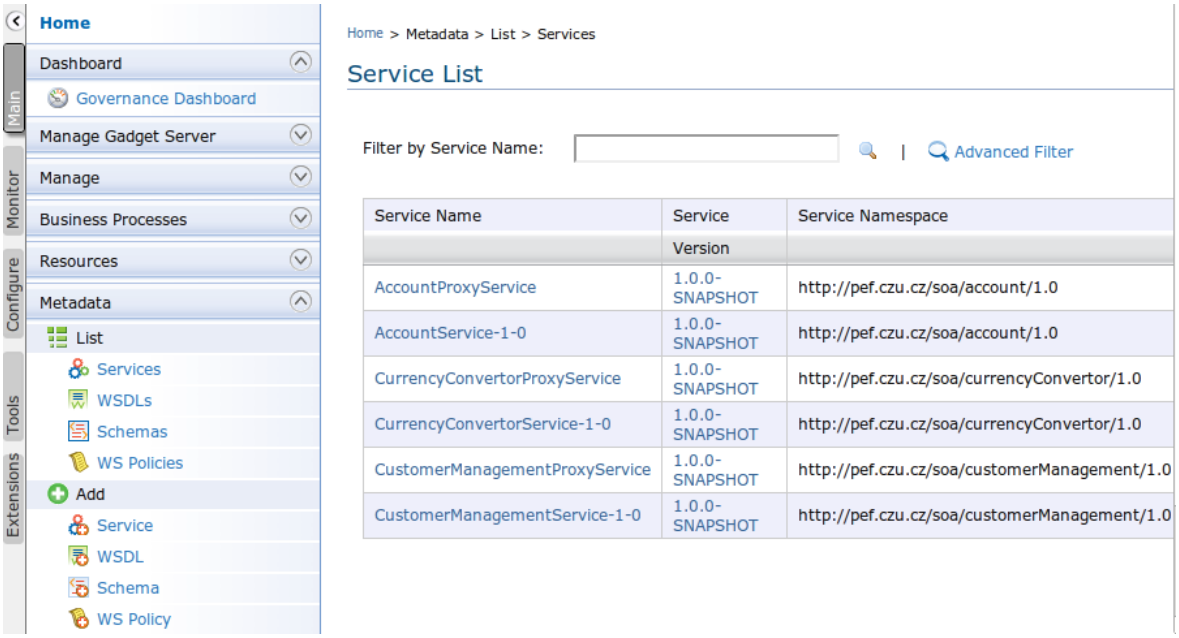
1. Vytvořit Carbon modul BP_ProxyServices.
2. V menu zvolit „New / ProxyService" a šablonu "Log and forward service".
3. V položce WSDL zvolit získání WSDL pomocí klíče "Governance registry key" s hodnotou "gov:trunk/wsdls/cz/czu/pef/soa/customermanagement/_1_0/CustomerManagementService-1-0.wsdl".

4. V položky Endpoint zvolit získání pomocí klíče “Named reference” s hodnotou “CustomerManagement_1_0“.

4.3.9 Registrace ESB proxy webových služeb do WSO2 GREG

Dalším krokem je nasazení BP_ProxyServices modulu do WSO2 ESB produktu a následná registrace jednotlivých ESB proxy webových služeb do produktu WSO2 GREG. Postup je obdobný jako v případě koncových webových služeb. Vystavené ESB proxy služby následně mohou být užity v rámci obchodního procesu.

Obrázek 2 - Seznam koncových a proxy služeb v WSO2 Governance Registry



Zdroj: autor, 2012

4.3.10 Vývoj obchodního procesu BP_BusinessProcess

Tento modul implementuje obchodní proces za užití produktu IBM WPS a technologií SCA a WS-BPEL. Cílem je předvést komponovatelnost a znovupoužitelnost servisně orientovaných služeb při realizaci funkcí podniku. Obchodní proces využije výše implementovaných proxy služeb k vykonání procesu navýšení (resp. snížení) částky na cílovém bankovním účtu. Logika obchodního procesu zároveň zajišťuje, že cílový bankovní účet může být veden v libovolné měně a stejně tak i připisovaná (resp.

odepisovaná) částka může být v libovolné měně. Obchodní proces využije služby CurrencyConvertorService k přepočtu kurzu devizového trhu a výpočtu cílové částky.

Proces by bylo možné dále rozšiřovat a obohacovat o další funkce, například:

- o možnost získávání kurzu ke konkrétnímu datu či pomocí volání jiného (pod)procesu, který by tuto informaci získával komplexnějším způsobem,
- o podporu kontokorentních limitů pro jednotlivé účty a zákazníky na podobném principu,
- o podporu schvalování (pomocí Human Task kroku) při překročení určitých limitů.

Zdrojové soubory modulu jsou dostupné v kapitole Příloha D – Zdrojové soubory modulu BP_BusinessProcess.

Postup vývoje obchodního procesu:

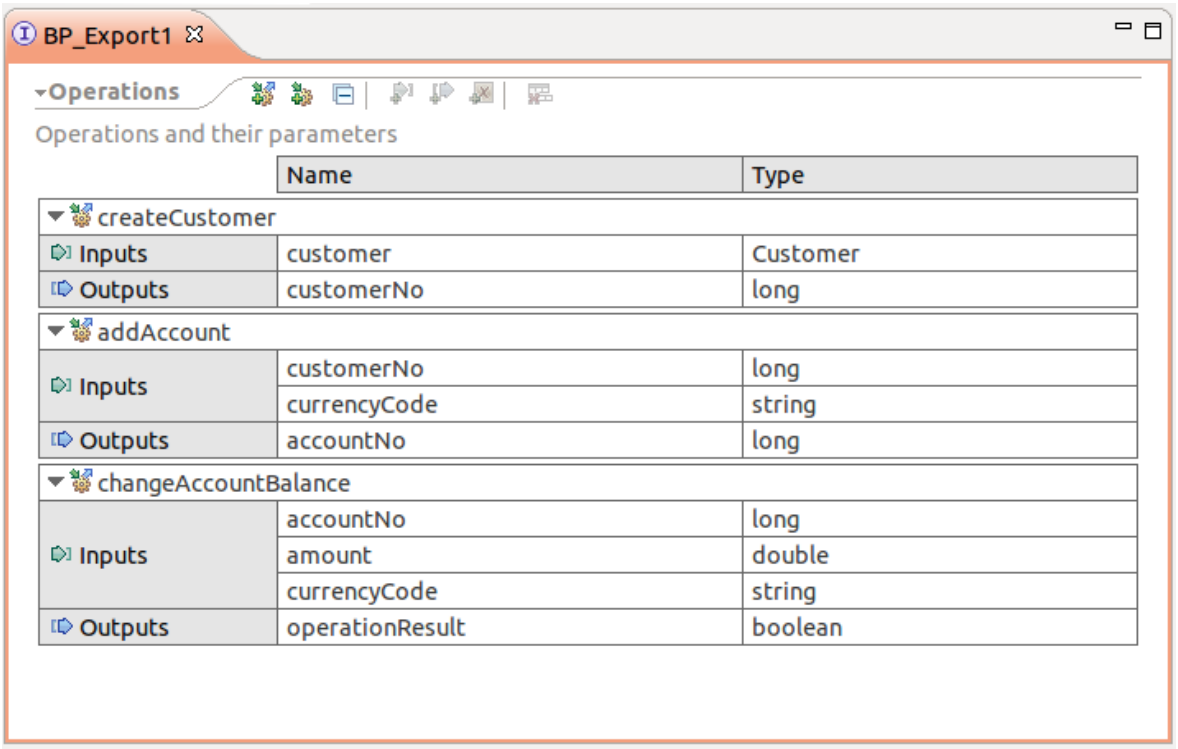
1. Ve vývojovém prostředí IBM WebSphere Integration Developer vytvořit nový projekt typu Module s názvem BP_BusinessProcess
2. Vytvořit nové Business Object-y Customer, Person a Account s následujícími atributy, kardinalitou a vazbami:

Obrázek 3 – Datové struktury použité v modulu BP_BusinessProcess

Zdroj: autor, 2012

3. Vytvořit nové WSDL rozhraní BP_Export1 s těmito operacemi:

Obrázek 4 – Exportní rozhraní modulu BP_BusinessProcess

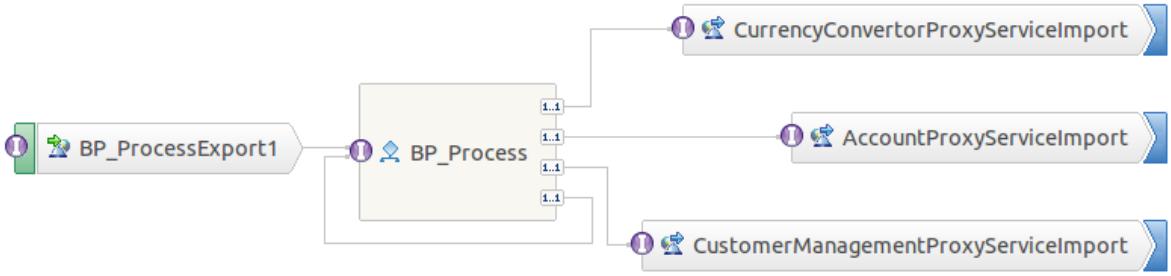


	Name	Type
createCustomer		
Inputs	customer	Customer
Outputs	customerNo	long
addAccount		
Inputs	customerNo	long
	currencyCode	string
Outputs	accountNo	long
changeAccountBalance		
Inputs	accountNo	long
	amount	double
	currencyCode	string
Outputs	operationResult	boolean

Zdroj: autor, 2012

4. Naimportovat do projektu WSDL soubory proxy služeb vystavených na WSO2 ESB.
5. V Assembly Diagramu vytvořit:
    a. Nový proces nazvaný BP_Process
    b. SCA export používající rozhraní BP_Export1
    c. SCA importy používající rozhraní ESB proxy služby

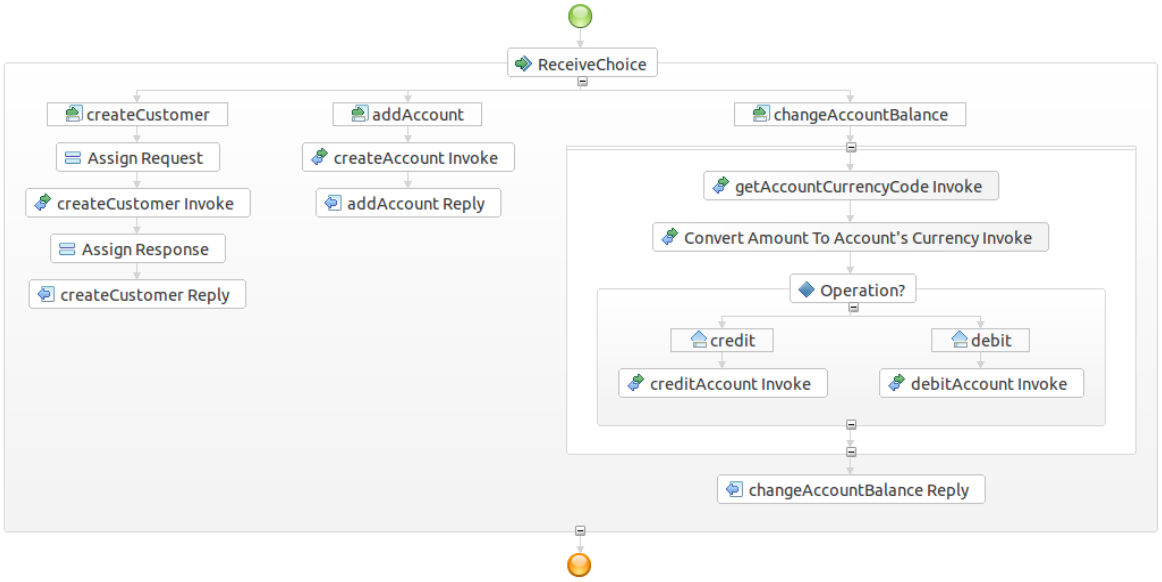
Obrázek 5 – Assembly Diagram modulu BP_BusinessProcess



Zdroj: autor, 2012

6. Implementovat proces BP_Process:

Obrázek 6 – Implementace obchodního procesu BP_BusinessProcess



Zdroj: autor, 2012

Proces changeAccountBalance volá postupně operace:

- AccountPartner getAccountCurrencyCode,
- CurrencyConvertorPartner convert,
- A podle znaménka vstupní částky buď AccountPartner creditAccount anebo AccountPartner debitAccount.

5 Zhodnocení výsledků

Vlastní práce se zaměřila na návrh servisně orientované architektury fiktivní banky a implementaci vybraných servisně orientovaných služeb. Praktické ověření proběhlo za využití produktů typu Application Server, SOA Governance, Enterprise Service Bus a Business Process Server. Zároveň byla ověřena základní vlastnost servisně orientovaných služeb tvořit pomocí kompozice nové služby či přímo obchodní procesy.

Výsledkem vlastní práce bylo zjištění, že při nesprávném návrhu služeb může snadno dojít ke komplikacím, pokud by bylo nutné službu užít i v jiných případech než pro které byla původně zamýšlena. Z tohoto důvodu je nezbytně nutné již při prvotním návrhu služeb striktně dodržovat pravidla servisně orientovaného návrhu tak, jak je uvádí Erl (2009, s. s.328-341).

Jako příklad lze uvést postup při návrhu služby AccountService. První verze této služby obsahovala atribut customerNo (jako vlastníka účtu), což by znamenalo, že vlastníkem účtu může být pouze zákazník. Takový návrh by ovšem z dlouhodobého hlediska také znamenal, že by nebylo možné, aby tato služba byla použita i pro jiné typy vazeb než Účet-Zákazník. Přitom by například mohlo být validní, aby existovala vazba Účet-PotenciálníZákazník anebo Účet-JináBanka. Z tohoto důvodu byl následně atribut změněn z customerNo na accountOwnerNo.

Podobný problém nastal při návrhu služby CustomerService, kdy z počátku byla služba definována tak, aby zákazník mohl mít otevřený pouze jeden účet (kardinalita 1:1). Po opravě kardinality na 1:n ovšem zpětně nastala situace, kdy ve službě AccountService bylo zapotřebí přidat novou operaci getAccountsByOwner (operace vrací 0 až n objektů) a naopak zrušit operaci getAccountByOwner (operace vracela nejvýše 1 objekt).

Uvedené příklady jsou typickým příkladem, kdy již během úvodního návrhu služeb lze správným rozmyšlením principů servisní orientace a hlubokou znalostí řešené problematiky zvýšit potenciál znovupoužitelnosti tvořených služeb.

Vlastní práce proběhla za užití vývojových prostředků WSO2 Carbon Studio a IBM WebSphere Integration Developer, přičemž bylo zároveň nutné, aby byly nastartovány také všechny užité produkty z řady WSO2 a také IBM WPS. Při takovéto zátěži nepostačovala

pro vývoj paměť o kapacitě 4GB RAM a bylo nutné přistoupit k alternativnímu řešení. Užitím OSGi technologie produktů WSO2 bylo dosaženo podstatného snížení nároků na paměť, a to spuštěním všech WSO2 modulů v rámci jedné instance Java virtuálního stroje. V důsledku to znamenalo také ověření vlastnosti volné vazby servisně orientovaných služeb, jak uvádí Pulier a kol. (2006, s. 52), jelikož při přesunu jednotlivých služeb na jiné umístění nebyla zasažena funkcionalita klientských modulů - nemusel se tedy měnit jejich kód.

Při realizaci obchodního procesu bylo zřejmé, že vlastní realizace byla velmi efektivní a jednoduchá. To potvrzuje názor Puliera a kol. (2006, s. 86), že SOA obecně pomáhá zvyšovat efektivitu modelování business procesů.

Během úvodní realizace obchodního procesu byl ovšem identifikován problém s produktem WSO2 BPS, který i při jednoduchých operacích neustále vykazoval chyby a nebylo možné jej proto ve vlastní práci využít. Jako náhrada byl proto zvolen produkt IBM WPS, který již fungoval bezchybně. Zde lze konstatovat, že při správné volbě průmyslových standardů užitých pro SOA lze dosáhnout vyšší interoperability, jak uvádí Erl (2009, s. 396). Dokonce lze takto dosáhnout možnosti záměny celých produktů.

Závěrem lze konstatovat, že vlastní práce úspěšně realizovala a ověřila všechny stanovené cíle.

Závěr

Jedním z cílů této práce bylo poskytnout ucelený přehled o důvodech vzniku konceptu servisně orientované architektury. Úvodní kapitoly se této problematice věnují z pohledu historického vývoje jednotlivých architektonických přístupů při modelování informačních systémů a jejich vzájemné interakce. Za tímto účelem byl představen vývoj základní klient-server komunikace v souvislosti s objektovými přístupy a následné snahy o standardizaci komunikačních technologií. Vyvrcholením těchto snad bylo vytvoření přístupu integrace podnikových aplikací (EAI), který ovšem ve výsledku byl limitován užitím různých proprietárních technologií a v důsledku toho i nízké flexibilitě změn. Tato úvodní část následně uvedla další přístupy, které mohou společně se SOA spolupracovat a vzájemně se podporovat. Mezi vybrané přístupy patří Event-driven architecture, Business Process Management a Cloud Computing.

Druhým cílem práce bylo formulovat základní teze, které by definovaly servisně orientovanou architekturu jako celek. V tomto ohledu bylo zdůrazněno, že se jedná především o obecný koncept, jehož příslibem je schopnost kontinuálního vývoje a optimalizace podniku na základě potřeb a celkové strategie podniku. Byly představeny základní vlastnosti servisně orientované architektury, a následně byla uvedena také její formální definice. Dále byly představeny principy servisní orientace služeb modelovaných v kontextu SOA.

Navazující kapitola se zabývala pohledem na zavádění této architektury v podniku a riziky s tím spojenými. Byly uvedeny jednotlivé fáze a kritéria pro zavádění, a to od úvodních rozhodnutí o způsobu a rozsahu implementace, přes úvodní volbu vhodných technologií a pilotní projekt, až po vyškolení zaměstnanců a vlastní implementaci.

Následně byla uvedena charakteristika jednotlivých logických vrstev služeb, a bylo provedeno členění jednotlivých typů služeb podle jejich účelu a funkcionality. Byl představen způsob postupu při modelování služeb, kde první část tvoří servisně orientovaná analýza následována servisně orientovaným návrhem. První část se věnovala způsobům identifikace kandidátských služeb a jejich operací. Druhá část se věnovala oblastem rozhodování o implementaci různých vrstev služeb, stanovení pravidel pro tvorbu

služeb, volbě konkrétních technologií a na závěr také vlastním principům návrhu jednotlivých typů služeb.

Práce dále uvedla přehled technologií, které tvoří základ dnešní SOA a které ji dále rozšiřují. Byly uvedeny základní vlastnosti rozšíření webových služeb o podporu adresování, spolehlivé výměny zpráv, definice zásad a zabezpečení. Byly představeny vlastnosti produktu Enterprise Service Bus a technologie SCA, které v kombinaci s BPM přinášejí zvýšení efektivity při vývoji a změnách automatizovaných obchodních procesů, a které byly z těchto důvodů také užity v rámci vlastní práce.

V závěru přehledu řešené problematiky byl uveden výčet nejvíce propracovaných softwarových řešení, pomocí kterých lze SOA v podniku implementovat.

Vlastní práce se zaměřila na ukázku implementace jednoduché funkcionality fiktivní banky dle principů SOA. Byla navržena vhodná IT architektura, zvoleny softwarové produkty a provedeno rozčlenění požadovaných funkcí do modulů a služeb. Následně byly tyto služby propojeny vhodným způsobem tak, aby bylo možné prezentovat výhody spojované se servisně orientovanou architekturou. Implementace proběhla za využití produktů typu Application Server, SOA Governance, Enterprise Service Bus a Business Process Server. V závěru práce bylo možné ověřit některé názory citovaných autorů a také výhody spojované s vlastnostmi servisně orientovaných služeb (volnou vazbu a znovupoužitelnost). Závěrem vlastní práce bylo možné konstatovat, že SOA obecně pomáhá zvyšovat efektivitu modelování business procesů a zvyšuje interoperabilitu systémů.

Z pohledu rozsahu je téma servisně orientované architektury velmi rozsáhlé a komplexní. Práce si sice vytkla za cíl charakterizovat pouze základní aspekty SOA a následně implementovat jednoduchou aplikaci, ale i přesto byl výrazně přesažen stanovený rozsah práce. Důvodem byla především nutnost implementace čtyř nezávislých modulů a jejich služeb tak, aby bylo dosaženo stanovených cílů.

Práce samotná nepokrývá celou oblast servisně orientované architektury a i nadále by ji bylo možné v tomto ohledu rozšiřovat. Předně by bylo možné práci rozšířit o detailní rozbor postupů souvisejících s analýzou a návrhem servisně orientovaných služeb. Dále by bylo možné práci rozšířit charakterizací dodatečných specifikací a technologií, které

rozšiřují možnosti webových služeb a tím i zvyšují potenciál nahrazovat stávající proprietární řešení a technologie. V neposlední řadě by bylo možné se zaměřit na další související architektonické přístupy a technologie, které koexistují se současnou vizí SOA. Jako příklad lze uvést zpracování dat v cloudu ve vztahu k flexibilitě servisně orientovaných služeb či analýzu možných synergií s Event-driven architekturou.

Seznam použitých zdrojů

Bloomberg, Jason. 2010. The Dangers of "Checklist" Architecture. *zapthink.* [Online] zapthink, 15. 06 2010. [Citace: 9. 3 2012.] http://www.zapthink.com/2010/06/15/the-dangers-of-checklist-architecture.

Cummins, Fred. 2009. *Building the agile enterprise with SOA, BPM and MBM.* Amsterdam Boston : MK/OMG Press/Elsevier, 2009. str. 306. ISBN: 978-0-12-374445-6.

Emmerich, Wolfgang. 2001. *Engineering distributed objects : Second International Workshop, EDO 2000, Davis, CA, USA, November 2-3, 2000 : revised paper.* [ed.] Stefan Tai. Berlin New York : Springer, 2001. ISBN: 3-540-41792-3.

Erl, Thomas. 2009. *SOA : servisně orientovaná architektura : kompletní průvodce.* Brno : Computer Press, 2009. str. 671. ISBN: 978-80-251-1886-3.

Forrester. 2011. SOA Adoption 2010: Still Important, Still Strong. *Forrester.* [Online] Forrester, 22. 3 2011. [Citace: 10. 3 2012.] http://www.forrester.com/SOA+Adoption+2010+Still+Important+Still+Strong/fulltext/-/E-RES59058.

Gartner, Inc. 2009. Gartner Says SOA Is Evolving Beyond Its Traditional Roots. *Gartner.* [Online] 2. 4 2009. [Citace: 10. 3 2012.] http://www.gartner.com/it/page.jsp?id=927612.

Handy, Alex. 2010. SOA's dead; long live SOA. *SD Times.* [Online] 15. 1 2010. [Citace: 9. 3 2012.] http://www.sdtimes.com/link/34062.

Open SOA Collaboration. *Open SOA Collaboration.* [Online] [Citace: 04. 12 2011.] http://www.osoa.org.

Pulier, Eric a Taylor, Hugh. 2006. *Understanding enterprise SOA.* Greenwich, Conn. London : Manning Pearson Education, 2006. str. 260. ISBN: 978-1-932394-59-7.

Svoboda, Jiří. 2009. *Cloud Computing.* [Document] Praha : ČSSI, Česká společnost pro systémovou integraci, 2009. Systémová integrace, Sv. 2. ISSN: 1210-9479.

Přílohy

Následující přílohy obsahují zdrojové soubory implementovaných modulů. Z důvodu omezení rozsahu jsou uvedeny pouze některé vybrané zdrojové soubory a jejich části.

Příloha A – Zdrojové soubory modulu BP_AccountModule

Tato příloha obsahuje zdrojové soubory modulu BP_AccountModule.

Account.java

Tato třída reprezentuje datový objekt konkrétního účtu.

```
public class Account implements Serializable {
  public Long accountOwnerNo;
  public Long accountNo;
  public String currencyCode;
  public double balance = 0;

  public Account(Long accountOwnerNo, String currencyCode) {
    this.accountOwnerNo = accountOwnerNo;
    this.currencyCode = currencyCode;
  }
}
```

AccountService.java

Tato třída reprezentuje službu pro práci s účty zákazníka.

```
public class AccountService {

  static Hashtable<Long, Account> accountStore = new Hashtable<Long,
Account>();
  private static long accountNoSequence = 0;

  public long createAccount(Long accountOwnerNo, String currencyCode) {
    accountNoSequence++;
    Account newAccount = new Account(accountOwnerNo, currencyCode);
    newAccount.accountNo = accountNoSequence;
    accountStore.put(accountNoSequence, newAccount);
    return accountNoSequence;
  }

  public boolean deleteAccount(long accountNo) {
    return !(accountStore.remove(accountNo) == null);
  }

  public Account getAccount(long accountNo) {
    return accountStore.get(accountNo);
  }
```

```
  public long getAccountOwner(long accountNo) {
    if (checkAccount(accountNo)) {
      return accountStore.get(accountNo).accountOwnerNo;
    }
    throw new RuntimeException("Account does not exist: " + accountNo);
  }

  public String getAccountCurrencyCode(long accountNo) {
    if (checkAccount(accountNo)) {
      return accountStore.get(accountNo).currencyCode;
    }
    throw new RuntimeException("Account does not exist: " + accountNo);
  }

  public Long[] getAccountsByOwner(long accountOwnerNo) {

    List<Long> result = new ArrayList<Long>();
    for (Long accountNo : accountStore.keySet()) {
      if (getAccountOwner(accountNo) == accountOwnerNo)
        result.add(accountNo);
    }
    return (Long[]) result.toArray(new Long[result.size()]);
  }

  public boolean checkAccount(long accountNo) {
    return (accountStore.containsKey(accountNo));
  }

  public double checkAccountBalance(long accountNo) {
    if (checkAccount(accountNo)) {
      return accountStore.get(accountNo).balance;
    }
    throw new RuntimeException("Account does not exist: " + accountNo);
  }

  public boolean creditAccount(long accountNo, double amount) {
    if (checkAccount(accountNo)) {
      Account account = accountStore.get(accountNo);
      account.balance += amount;
      return true;
    }
    throw new RuntimeException("Account does not exist: " + accountNo);
  }

  public boolean changeAccountOwner(long accountNo, long
newAccountOwnerNo) {
    if (checkAccount(accountNo)) {
      Account account = accountStore.get(accountNo);
      account.accountOwnerNo = newAccountOwnerNo;
      return true;
    }
    throw new RuntimeException("Account does not exist: " + accountNo);
  }

  public boolean debitAccount(long accountNo, double amount) {
    if (checkAccount(accountNo)) {
      Account account = accountStore.get(accountNo);
      if (account.balance >= amount) {
```

```
        account.balance -= amount;
        return true;
      }
      return false;
    }
    throw new RuntimeException("Account does not exist: " + accountNo);
  }
}
```

CurrencyConvertorService.java

Tato třída reprezentuje službu pro práci s kurzy devizového trhu.

```
public class CurrencyConvertorService {
  private static Hashtable<String, Double> currencyExchangeRates = new
Hashtable<String, Double>();

  private double getConvertionRatePrivate(Currency srcCurrency, Currency
destCurrency) {
    String key1 = srcCurrency.getCurrencyCode() + "-" +
                  destCurrency.getCurrencyCode();
    String key2 = destCurrency.getCurrencyCode() + "-" +
                  srcCurrency.getCurrencyCode();
    if (currencyExchangeRates.containsKey(key1))
        return currencyExchangeRates.get(key1);
    if (currencyExchangeRates.containsKey(key2))
        return 1.0d / currencyExchangeRates.get(key2);

    Random randomGenerator = new Random();
    double rate = randomGenerator.nextDouble();
    currencyExchangeRates.put(key1, rate);
    return rate;
  }

  public double getConvertionRate(String srcCurrencyCode, String
destCurrencyCode) {
    Currency srcCurrency = Currency.getInstance(srcCurrencyCode);
    Currency destCurrency = Currency.getInstance(destCurrencyCode);

    return getConvertionRatePrivate(srcCurrency, destCurrency);
  }

  public double convert(double amount, String srcCurrencyCode, String
destCurrencyCode) {
    double rate = getConvertionRate(srcCurrencyCode, destCurrencyCode);
    return amount * rate;
  }
}
```

Axis2 konfigurace services.xml služby AccountService

Axis2 konfigurace služby AccountService:

```
<service name="AccountService-1-0"
targetNamespace="http://pef.czu.cz/soa/account/1.0">
    <schema schemaNamespace="http://pef.czu.cz/soa/account/1.0"/>
      <description>
```

```
            This is first version of Account service.
      </description>
      <messageReceivers>
            <messageReceiver mep="http://www.w3.org/2004/08/wsdl/in-only"
class="org.apache.axis2.rpc.receivers.RPCInOnlyMessageReceiver" />
            <messageReceiver  mep="http://www.w3.org/2004/08/wsdl/in-out"
class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"/>
      </messageReceivers>
      <parameter name="ServiceClass"
locked="false">cz.czu.pef.application.service.AccountService</parameter>
</service>
```

Axis2 konfigurace services.xml služby CurrencyConvertorService

Axis2 konfigurace služby CurrencyConvertorService:

```
<service name="CurrencyConvertorService-1-0"
targetNamespace="http://pef.czu.cz/soa/currencyConvertor/1.0">
    <schema
schemaNamespace="http://pef.czu.cz/soa/currencyConvertor/1.0"/>
      <description>
            This is first version of CurrencyConvertor service.
      </description>
      <messageReceivers>
            <messageReceiver mep="http://www.w3.org/2004/08/wsdl/in-only"
class="org.apache.axis2.rpc.receivers.RPCInOnlyMessageReceiver" />
            <messageReceiver  mep="http://www.w3.org/2004/08/wsdl/in-out"
class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"/>
      </messageReceivers>
      <parameter name="ServiceClass"
locked="false">cz.czu.pef.application.service.CurrencyConvertorService</p
arameter>
</service>
```

Příloha B – Zdrojové soubory modulu BP_CustomerModule

Tato příloha obsahuje zdrojové soubory modulu BP_CustomerModule.

Customer.java

Tato třída reprezentuje datový objekt údajů o konkrétním zákazníkovi.

```
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "customerType", namespace =
"http://pef.czu.cz/soa/customerManagement/1.0", propOrder = {
    "customerNo", "owner" })
public class Customer implements Serializable {
  @XmlElement(required = true, nillable = false)
  public long customerNo;

  @XmlElement(required = true, nillable = false)
  public Person owner;

  public void copyNonEmptyObjectValues(Customer customer) {
    if (owner == null)
```

```
      owner = customer.owner;
    else
      owner.copyNonEmptyObjectValues(customer.owner);
  }
}
```

Person.java

Tato třída reprezentuje datový objekt údajů o konkrétní osobě.

```
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "personType", namespace =
"http://pef.czu.cz/soa/customerManagement/1.0", propOrder = {
    "firstName", "surname" })
public class Person implements Serializable {
  private static final long serialVersionUID = -5387226793068054232L;

  @XmlElement(required = true, nillable = false)
  public String firstName;

  @XmlElement(required = true, nillable = false)
  public String surname;

  public void copyNonEmptyObjectValues(Person person) {
    if (person == null)
      return;

    if (person.firstName != null)
      this.firstName = person.firstName;
    if (person.surname != null)
      this.surname = person.surname;
  }
}
```

CustomerManagement.java

Rozhraní služby pro práci s údaji zákazníka.

```
@WebService(targetNamespace =
"http://pef.czu.cz/soa/customerManagement/1.0")
@SOAPBinding(style = SOAPBinding.Style.DOCUMENT, use =
SOAPBinding.Use.LITERAL, parameterStyle =
SOAPBinding.ParameterStyle.WRAPPED)
public interface CustomerManagement {

  @WebMethod(operationName = "createCustomer")
  @WebResult(name = "customerNo")
  public Long createCustomer(@WebParam(name = "owner") Person owner);

  @WebMethod(operationName = "deleteCustomer")
  @WebResult(name = "operationSucceded")
  public boolean deleteCustomer(@WebParam(name = "customerNo") Long
customerNo);

  @WebMethod(operationName = "getCustomer")
  @WebResult(name = "customer")
```

```
  public Customer getCustomer(@WebParam(name = "customerNo") Long
customerNo);

  @WebMethod(operationName = "updateCustomer")
  @WebResult(name = "operationSucceded")
  public boolean updateCustomer(@WebParam(name = "customer") Customer
customer);
}
```

CustomerManagementServiceImpl.java

Implementace služby pro práci s údaji zákazníka.

```
@WebService(serviceName = "CustomerManagementService-1-0", portName =
"CustomerManagementPort", endpointInterface =
"cz.czu.pef.application.service.CustomerManagement", targetNamespace =
"http://pef.czu.cz/soa/customerManagement/1.0"
)
public class CustomerManagementServiceImpl implements CustomerManagement
{
  private static Hashtable<Long, Customer> customers = new
Hashtable<Long, Customer>();
  private static long customerNoSequence = 0;

  @Override
  public Long createCustomer(Person owner) {
    customerNoSequence++;
    Customer newCustomer = new Customer();
    newCustomer.owner = owner;
    newCustomer.customerNo = customerNoSequence;

    // create the entry in the customers
    customers.put(customerNoSequence, newCustomer);
    return customerNoSequence;
  }

  @Override
  public boolean deleteCustomer(Long customerNo) {
    if (customers.remove(customerNo) == null)
      return false;
    return true;
  }

  @Override
  public Customer getCustomer(Long customerNo) {
    return customers.get(customerNo);
  }

  @Override
  public boolean updateCustomer(Customer customer) {
    Customer myCustomer = customers.get(customer.customerNo);
    if (myCustomer == null)
      return false;

    myCustomer.copyNonEmptyObjectValues(customer);
    return true;
  }
```

```
}
```

Příloha C – Zdrojové soubory modulu BP_ProxyServices

Tato příloha obsahuje zdrojové soubory modulu BP_ProxyServices.

AccountProxyService.xml

Definice ESB proxy služby AccountProxyService:

```
<?xml version="1.0" encoding="UTF-8"?>
<proxy xmlns="http://ws.apache.org/ns/synapse"
    name="AccountProxyService" statistics="disable" trace="disable"
transports="https,http">
    <target endpoint="AccountService_1_0">
        <inSequence>
            <log category="INFO" level="full" separator=","/>
        </inSequence>
        <outSequence>
            <send/>
        </outSequence>
    </target>
    <publishWSDL
key="gov:trunk/wsdls/cz/czu/pef/soa/account/_1_0/AccountService-1-
0.wsdl"/>
</proxy>
```

CurrencyConvertorProxyService.xml

Definice ESB proxy služby CurrencyConvertorProxyService:

```
<?xml version="1.0" encoding="UTF-8"?>
<proxy xmlns="http://ws.apache.org/ns/synapse"
    name="CurrencyConvertorProxyService" statistics="enable"
    trace="disable" transports="https,http">
    <target endpoint="CurrencyConvertor_1_0">
        <inSequence>
            <log category="INFO" level="full" separator=","/>
        </inSequence>
        <outSequence>
            <send/>
        </outSequence>
    </target>
    <publishWSDL
key="gov:trunk/wsdls/cz/czu/pef/soa/currencyconvertor/_1_0/CurrencyConver
torService-1-0.wsdl"/>
</proxy>
```

CustomerManagementProxyService.xml

Definice ESB proxy služby CustomerManagementProxyService:

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<proxy xmlns="http://ws.apache.org/ns/synapse"
    name="CustomerManagementProxyService" statistics="disable"
    trace="disable" transports="https,http">
    <target endpoint="CustomerManagement_1_0">
        <inSequence>
            <log category="INFO" level="full" separator=","/>
        </inSequence>
        <outSequence>
            <send/>
        </outSequence>
    </target>
    <publishWSDL
key="gov:trunk/wsdls/cz/czu/pef/soa/customermanagement/_1_0/CustomerManag
ementService-1-0.wsdl">
        <resource

key="gov:/trunk/schemas/cz/czu/pef/soa/customermanagement/_1_0/CustomerMa
nagementService-1-0.xsd"
location="../../../../../../../schemas/cz/czu/pef/soa/customermanagement/
_1_0/CustomerManagementService-1-0.xsd"/>
    </publishWSDL>
</proxy>
```

Příloha D – Zdrojové soubory modulu BP_BusinessProcess

Tato příloha obsahuje zdrojové soubory modulu BP_BusinessProcess. Z důvodu omezení rozsahu zde nebyly uvedeny následující důležité soubory:

- užité WSDL soubory a XML schémata,
- zdrojové soubory jednotlivých SCA komponent,
- výsledný BPEL soubor a k němu přidružené soubory.

BP_Process.component

Tento soubor obsahuje definici komponenty BP_Process. Soubor definuje externí rozhraní komponenty a vazby na komponenty uvnitř modulu:

```
<?xml version="1.0" encoding="UTF-8"?>
<scdl:component xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ns1="http://BP_BusinessProcess/wsdl/exports/BP_Export1"
xmlns:ns2="http://pef.czu.cz/soa/currencyConvertor/1.0"
xmlns:ns3="http://pef.czu.cz/soa/account/1.0"
xmlns:ns4="http://pef.czu.cz/soa/customerManagement/1.0"
xmlns:process="http://www.ibm.com/xmlns/prod/websphere/scdl/business-
process/6.0.0"
xmlns:scdl="http://www.ibm.com/xmlns/prod/websphere/scdl/6.0.0"
xmlns:wsdl="http://www.ibm.com/xmlns/prod/websphere/scdl/wsdl/6.0.0"
displayName="BP_Process" name="BP_Process">
  <interfaces>
    <interface xsi:type="wsdl:WSDLPortType"
preferredInteractionStyle="async" portType="ns1:BP_Export1">
```

```
      <scdl:interfaceQualifier xsi:type="scdl:JoinTransaction"
value="false"/>
    </interface>
  </interfaces>
  <references>
    <reference name="CurrencyConvertorProxyServicePortTypePartner">
      <interface xsi:type="wsdl:WSDLPortType"
portType="ns2:CurrencyConvertorProxyServicePortType"/>
      <scdl:referenceQualifier xsi:type="scdl:SuspendTransaction"
value="false"/>
      <scdl:referenceQualifier xsi:type="scdl:Reliability"/>
      <scdl:referenceQualifier xsi:type="scdl:DeliverAsyncAt"
value="commit"/>
      <wire target="CurrencyConvertorProxyServiceImport"/>
    </reference>
    <reference name="AccountProxyServicePortTypePartner">
      <interface xsi:type="wsdl:WSDLPortType"
portType="ns3:AccountProxyServicePortType"/>
      <scdl:referenceQualifier xsi:type="scdl:SuspendTransaction"
value="false"/>
      <scdl:referenceQualifier xsi:type="scdl:Reliability"/>
      <scdl:referenceQualifier xsi:type="scdl:DeliverAsyncAt"
value="commit"/>
      <wire target="AccountProxyServiceImport"/>
    </reference>
    <reference name="CustomerManagementProxyServicePortTypePartner">
      <interface xsi:type="wsdl:WSDLPortType"
portType="ns4:CustomerManagementProxyServicePortType"/>
      <scdl:referenceQualifier xsi:type="scdl:SuspendTransaction"
value="false"/>
      <scdl:referenceQualifier xsi:type="scdl:Reliability"/>
      <scdl:referenceQualifier xsi:type="scdl:DeliverAsyncAt"
value="commit"/>
      <wire target="CustomerManagementProxyServiceImport"/>
    </reference>
    <reference name="BP_Export1Partner">
      <interface xsi:type="wsdl:WSDLPortType" portType="ns1:BP_Export1"/>
      <scdl:referenceQualifier xsi:type="scdl:SuspendTransaction"
value="false"/>
      <scdl:referenceQualifier xsi:type="scdl:Reliability"/>
      <scdl:referenceQualifier xsi:type="scdl:DeliverAsyncAt"
value="commit"/>
      <wire target="BP_Process"/>
    </reference>
  </references>
  <implementation xsi:type="process:ProcessImplementation">
    <scdl:implementationQualifier xsi:type="scdl:Transaction"
value="global"/>
    <process bpel="/bpel/BP_Process.bpel"/>
  </implementation>

</scdl:component>
```