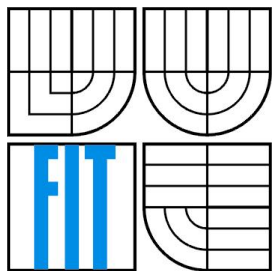


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ  
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ  
ÚSTAV INFORMAČNÍCH SYSTÉMŮ  
FACULTY OF INFORMATION TECHNOLOGY  
DEPARTMENT OF INFORMATION SYSTEMS

ARCHITEKTURA ORIENTO VANÁ NA SLUŽBY,  
NÁVRH ORIENTO VANÝ NA SLUŽBY,  
WEBOVÉ SLUŽBY

Ing. Petr Weiss, Mgr. Marek Rychlý

BRNO 2007

# Obsah

1. Úvod.....	3
1.1 Upozornění o účelu této publikace.....	3
1.2 Struktura publikace.....	3
1.3 Cíle a obsah práce.....	4
2. Architektura orientovaná na služby.....	5
2.1 Úvod.....	5
2.2 Koncepční předchůdci SOA.....	5
2.3 Konceptuální model.....	5
2.4 Struktura SOA.....	6
2.5 Spolupráce mezi službami.....	7
2.6 Základní vlastnosti servisně orientovaného přístupu.....	8
2.7 Hlavní výhody SOA z pohledu vývoje a uplatnění výsledného produktu.....	9
2.8 SOA vs. Architektura klient-server.....	9
2.8.1 Architektura klient-server.....	9
2.8.2 Rozdíly.....	10
3. Servisně orientovaná analýza a návrh.....	11
3.1 Servisně-orientovaný přístup a objektově-orientovaný přístup.....	11
3.2 BPM.....	11
3.3 EAf.....	12
3.4 Choreografie a orchestrace služeb.....	13
3.5 Vývojový cyklus SOAD.....	13
4. Webové služby.....	16
4.1 Definice WS.....	16
4.2 Vlastnosti WS.....	16
4.3 WSDL.....	17
4.3.1 Abstraktní popis.....	17
4.3.2 Konkrétní popis.....	17
4.4 SOAP.....	18
4.4.1 Vlastnosti SOAP.....	18
4.4.2 Struktura zprávy v SOAP.....	18
4.5 UDDI.....	20
5. Další jazyky webových služeb.....	22
5.1 Web Services Choreography Description Language.....	22
5.1.1 Struktura WS-CDL dokumentu.....	22
5.2 Business Process Execution Language.....	24
5.2.1 Struktura definice procesu ve BPEL4WS.....	24
6. Implementace webových služeb.....	27
6.1 Rámce a technologie pro implementaci webových služeb.....	27
6.1.1 Microsoft .NET.....	27
6.1.2 SUN Java Enterprise Edition (Java EE).....	27
6.1.3 Další rámce a technologie.....	29
6.2 Java API for XML Web Services (JAX-WS).....	29
6.2.1 Implementace poskytovatele WS pomocí JAX-WS.....	29
6.2.2 Implementace spotřebitele WS pomocí JAX-WS.....	30
6.2.3 Kombinace JAX-WS s doplňujícími technologiemi.....	31
6.3 Bezpečnost webových služeb v Java EE.....	32
7. Závěr.....	33

# 1. Úvod

upozornění o  
účelu této  
publikace

## 1.1 Upozornění o účelu této publikace

Autorský zákon, přesněji řečeno zákon 121/2000 Sb., ze dne 7. dubna 2000 o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů ve svém oddíle 3 praví.

citace  
autorského  
zákona

ODDÍL 3  
Bezúplatné zákonné licence

§ 31  
Citace

Do práva autorského nezasahuje ten, kdo

- a) cituje ve svém díle v odůvodněné míře výňatky ze zveřejněných děl jiných autorů,
- b) **zařadí do svého samostatného díla vědeckého, kritického, odborného nebo do díla určeného k vyučovacím účelům, pro objasnění jeho obsahu, drobná celá zveřejněná díla,**
- c) **užije zveřejněné dílo v přednášce výlučně k účelům vědeckým nebo vyučovacím či k jiným vzdělávacím účelům;**

vždy je však nutno uvést jméno autora, nejde-li o dílo anonymní, nebo jméno osoby, pod jejímž jménem se dílo uvádí na veřejnost, a dále název díla a pramen.

Ve smyslu tohoto zákona je obsahem této publikace souhrn náplně přednášek předmětů uvedených na titulní straně. Díla zde citována a výňatky z děl jiných autorů jsou určeny **výlučně k vyučovacím a vzdělávacím účelům** zejména ve smyslu odstavce b) a c) předchozího paragrafu.



**Použití obsahu pro jiné účely může znamenat porušení autorského zákona.**

## 1.2 Struktura publikace



















úvod

Studijní opora má strukturu publikace s číslovanými kapitolami. V levém pomocném sloupečku se vyskytují pomocné informace ve tvaru:

- hesel, která slouží k rychlé orientaci a vyhledávání a,
- piktogramů, označujících typické části textu.



Význam použitých piktogramů je uveden v následující tabulce:

piktogram	význam	piktogram	význam
	Počítačové cvičení, příklad		Správné řešení
	Otázka, příklad k řešení	  	Obtížná část. Obtížnost 3 je nejvyšší.
	Příklad		Důležitá část
	Slovo autora, komentář		Cíl
	Potřebný čas pro studium, doplněno číslicí přes hodiny		Definice
 	Reference	 	Zajímavé místo (levá, pravá varianta)
	Souhrn		Rozšiřující látka, informace, znalosti. Nejsou předmětem zkoušky.



### 1.3 Cíle a obsah práce

Tato práce by měla čtenáři přinést základní informace o architektuře orientované na služby a její reálné implementaci – webových službách. Práce má následující strukturu:

Druhá kapitola je úvodem do problematiky architektur orientovaných na služby. Tato kapitola vysvětluje základní pojmy a koncepci architektury, dále se věnuje porovnání servisně-orientovaného návrhu s objektově-orientovaným návrhem.

Třetí kapitola se zabývá návrhem a modelováním systému založeného na SOA.

Ve čtvrté kapitole jsou popsány principy webových služeb a jednotlivé technologie používané (popisné jazyky a protokoly) v této oblasti.

Pátá kapitola slouží, jako seznámí se s dalšími jazyky popisujícími webové služby. Jsou zmíněny WS-CDL a BPEL4WS.

Šestá a poslední kapitola výkladu popisuje způsoby implementace webových služeb v softwarových produktech pomocí nástrojů platforem Microsoft .NET a SUN Java Enterprise Edition.

## 2. Architektura orientovaná na služby

### 2.1 Úvod

**historie SOA** *Architektura orientovaná na služby* (angl. *Service-Oriented Architecture*, SOA) a její implementace – *webové služby* (angl. *Web Services*, WS) patří v dnešní době k nejdiskutovanějším tématům na poli distribuovaných informačních systémů. Ačkoli myšlenka SOA není ve své podstatě nová (na podobném principu pracovaly již např. koncepty CORBA nebo Microsoft (D)COM) zaznamenává velký rozvoj až v posledních letech a to především díky dvěma faktorům. Prvním z nich je dostatečná technická podpora. Do této oblasti můžeme zařadit spolehlivé sítě, kvalitní komunikační protokoly i dostatečný výpočetní výkon dnešních počítačů. Druhým faktorem jsou požadavky firem na takové IT prostředky, které budou mnohem více svázány s obchodními záměry a s procesy probíhajícími uvnitř firmy, než poskytují dosavadní informační systémy.



V současné době neexistuje žádná definice, která by přesně vystihovala podstatu SOA. Proto můžeme říci, že SOA je určitým stylem pro vytváření distribuovaných informačních systémů. Jedním dechem je však nutné dodat, že SOA je také nástrojem pro realizaci byznys procesů.

### 2.2 Koncepční předchůdci SOA

Koncepčními předchůdci současných architektur orientovaných na služby byly systémy založené na použití komponent z DCOM (*Distributed Component Object Model*) nebo ORB (*Object Request Broker*) z architektury CORBA (*Common Object Request Broker Architecture*).

Prvně jmenovaný předchůdce – DCOM – je rozšíření Komponentového Objektového Modelu (zkr. COM). DCOM byl představen v roce 1996 společností Microsoft a byl navržen pro komunikaci dílčích komponent napříč různými komunikačními sítěmi (mimo jiné i sítí Internet, protokolem HTTP).

CORBA je architektura vyvinutá pod záštitou OMG (Object Management Group). Tato architektura je v podstatě middleware umožňující libovolné aplikaci založené na CORBA konceptu komunikovat s libovolnou jinou aplikací stejného typu nezávisle na operačním systému, programovacím jazyce nebo komunikační síti.



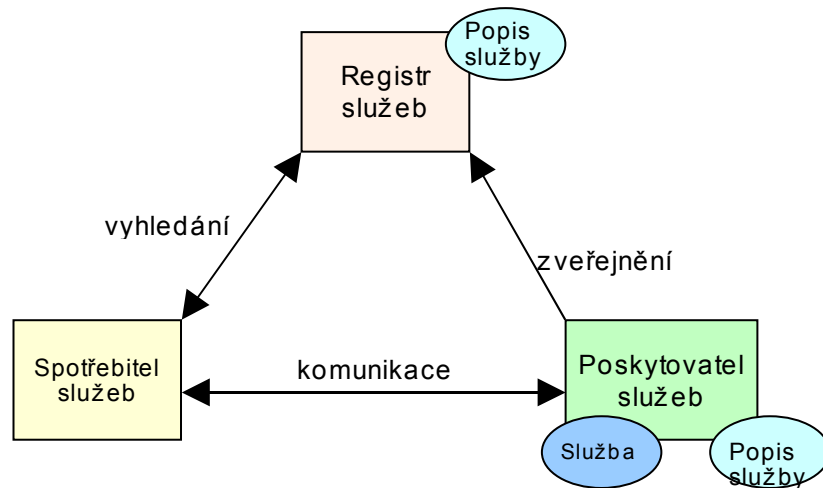
Další informace o DCOM resp. CORBA jsou k nalezení na <http://www.microsoft.com/com/default.mspix> resp. na <http://www.corba.org>

**konceptuální model SOA (spotřebitel a poskytovatel služeb)**

### 2.3 Konceptuální model

Tento koncept je založen na interakci mezi dvěma klíčovými entitami: *poskytovatelem služeb* a *spotřebitelem služeb* (angl. *service provider* a *service consumer*). Model takové interakce zobrazuje obrázek 2.1. Poskytovatel implementuje a nabízí služby. Jak již bylo zmíněno, každá služba je specifikovaná svým popisem. Na základě tohoto popisu spotřebitel vyhledá odpovídající službu v registru služeb a naváže s ní komunikaci.

obr. 2.1  
Konceptuální  
model SOA



SOA jako  
částečně  
vrstevnatá  
architektura  
(vrstva  
komponent,  
služeb a  
byznys  
procesů)

## 2.4 Struktura SOA

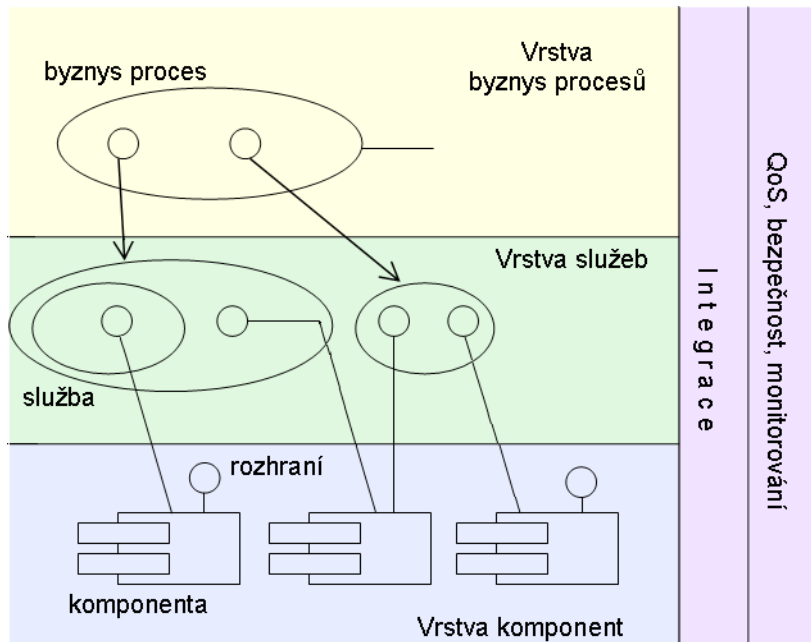
Abstraktním pohledem na SOA je částečně vrstevnatá architektura, kde každá vrstva představuje jinou úroveň abstrakce. Jak vidíme na obrázku 2.2 výchozí vrstvou je vrstva komponent. *Komponenty* jsou základní stavební kameny služeb, jsou zodpovědné za realizaci funkčnosti služeb a za zajištění požadované kvality služeb (QoS). Na této úrovni abstrakce jsou komponenty černé skříňky a jejich funkce jsou přístupné pouze přes rozhraní.

Na vyšší úrovni jsou rozhraní jednotlivých komponent sjednocena do služeb. *Služba* se dá chápat jako mechanismus, který za běhu sestavuje komponenty a vhodným způsobem jim přeposílá požadavky, které jsou kladeny na samotnou službu. Služba má podobně jako komponenty rozhraní a veškeré funkce služby jsou přístupné pouze přes toto rozhraní. Na základě rozhraní a dalších řídicích informací vzniká *popis služby* (angl. *service description*). Tento popis se uplatňuje při vyhledávání a komunikaci mezi službami (více v kapitole 4).

Nejvyšší vrstvou hierarchie zobrazené na obr. 2.2 je vrstva byznys procesů. *Byznys proces* (dále jen BP) je posloupnost kroků, která respektuje určitá byznys pravidla a vede k zisku (hmotnému i nehmotnému). V kontextu SOA je BP reprezentován sekvencí provedení několika služeb. BP jde tedy v tomto směru chápat jako jednoduchou samostatnou aplikaci. Proces sestavení služeb do BP je označován jako choreografie služeb.

Napříč těmito vrstvami jsou dvě řídicí vrstvy. První z nich podporuje integraci komponent a služeb do větších celků, druhá zajišťuje QoS, správu a monitorování SOA aplikací.

obr. 2.2  
Vrstevnatý  
model SOA  
převzato z [1]



spolupráce  
služeb:  
kooperace,  
agregace,  
choreografie

## 2.5 Spolupráce mezi službami

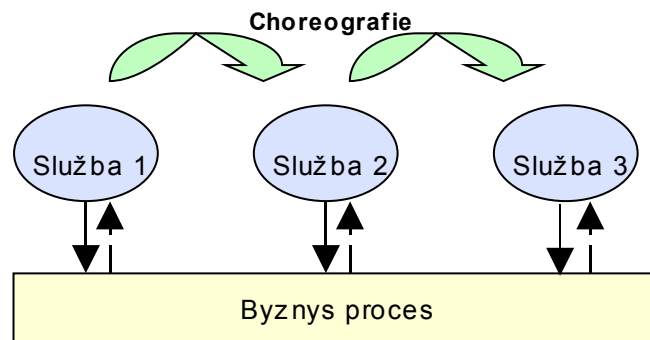
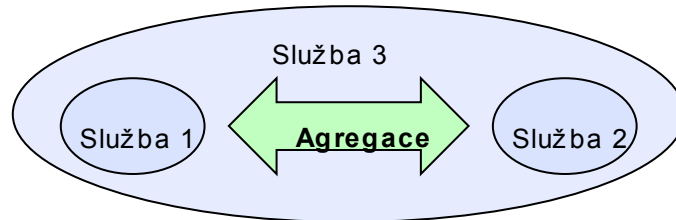
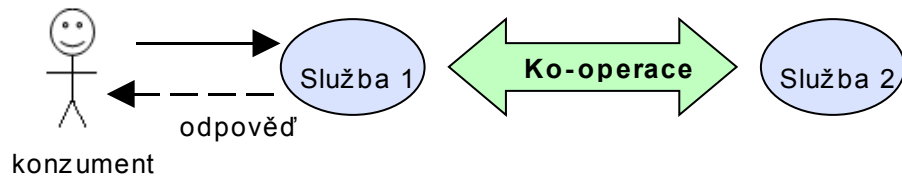
Jak již bylo zmíněno, služby jsou základními jednotkami v SOA a poskytují své prostředky buď přímo cílovému spotřebiteli anebo jiným službám. Služby mezi sebou spolupracují (komunikují) pomocí zasílání zpráv. Obecně lze spolupráci mezi službami rozdělit do tří kategorií: kooperace, agregace a choreografie. Každá kategorie je dána množinou komunikačních pravidel (protokolů) a účel, za jakým služby spolupracují. Schéma spolupráce služeb v jednotlivých případech je naznačeno na obrázku 2.3:

*Kooperace služeb popisuje situaci, kdy jedna služba využívá prostředky jiné služby, aby mohla plně realizovat funkce, které nabízí.*

*Agregace služeb je způsob, jakým lze ze dvou (nebo více) služeb sestavit novou službu. Výsledná služba potom nabízí kombinaci funkcí dílčích služeb.*

*Choreografie zprostředkovává službám spolupráci napříč organizacemi. Zúčastněné služby potom spolupracují za účelem provedení byznys procesu.*

obr. 2.3  
Spolupráce  
služeb



## 2.6 Základní vlastnosti servisně orientovaného přístupu

- *Volné propojení* – Vztahy mezi službami minimalizují závislosti a pouze poskytují službám informace službám jedné o druhé. Tato vlastnost umožňuje snadný vývoj a udržování SOA systémů (např. aktualizace nějaké služby znamená pouze její nahrazení jinou službou se stejným rozhraním).
- *Nezávislost* – Služby jsou autonomní sebe-řídící jednotky.
- *Abstrakce* – Služby zapouzdří svoji logiku a okolnímu světu jsou přístupné pouze přes rozhraní.
- *Znovupoužitelnost* je žádoucí vedlejší efekt návrhu a implementace a potvrzuje správnou implementaci služeb. Dobře navržená SOA se snaží minimalizovat počet potřebných služeb a snaží se, aby danou službu využíval co největší počet aplikací.
- *Bezstavovost* – Služby se pro vnějšího pozorovatele během své činnosti nenacházejí v žádném stavu. Služby se snaží minimalizovat množství uchovávané informace mezi jednotlivými komunikačními cykly (Toto se týká především dotazovaných služeb. Služby vyvolávající komunikaci samozřejmě potřebují uchovávat určité informace a čekají na odezvu svých



dotazů).

- *Nezávislost na platformě* – Služby jsou nezávislé na implementačním jazyce i na operačním systému.

## 2.7 Hlavní výhody SOA z pohledu vývoje a uplatnění výsledného produktu

Jednou z hlavních výhod systémů založených na SOA je jich způsob vývoje. Takovéto systémy většinou nevznikají na „zelené louce“, ale vhodnou dekompozicí stávajícího systému. Z toho plyne ušetření na vývoji nového systému a využití těch prostředků, do kterých bylo již dříve investováno.

Další výhodou těchto systémů je jejich snadná rozšiřitelnost. Vývoj nových služeb je mnohem rychlejší a levnější než více či méně složité úpravy stávajícího systému. Jedním z důvodů je i to, že služby jsou nezávislé na platformě a integraci nových služeb tak ovlivňuje pouze jejich specifikace.

Poslední výhodou systémů založených na SOA je jejich flexibilita. Znovupoužitelnost služeb a jednoduchá rozšiřitelnost dovolují systému snadné přizpůsobení novým funkčním požadavkům – snadná choreografie služeb do nových BP.

srovnání SOA  
s  
architekturou  
klient-server

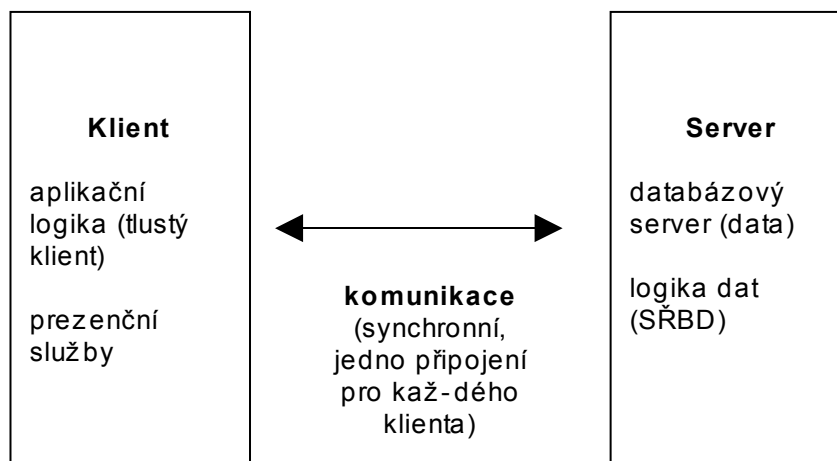
## 2.8 SOA vs. Architektura klient-server

Konceptuální model představený v kapitole 2.2 může svádět k tomu, že SOA je určitou podmnožinou architektury klient-server. Tato kapitola popisuje základní rozdíly mezi oběma koncepty.

### 2.8.1 Architektura klient-server

Počátky této architektury sahají do osmdesátých let minulého století, kdy byla využívána tzv. jednovrstvá architektura klient-server. Ta se skládala na jedné straně z terminálu (tenký klient bez aplikační logiky) a na straně druhé ze serveru, který zajišťoval aplikační logiku a spravoval data. Dalším krokem ve vývoji byla dvouvrstvá architektura (viz obr. 2.4).

obr. 2.4  
Architektura  
klient-server  
(dvouvrstvá)



### 2.8.2 Rozdíly

Reálná implementace takovéto architektury se skládá ze serveru a několika klientů, kteří se k serveru připojují. Klienti (tlustí klienti) obsahují aplikační logiku, zatím co servery ukrývají logiku související s daty a byznys pravidly (většinou v podobě spouští, angl. triggers, a uložených procedur). Veškerá data jsou uložena na straně serveru, s čímž souvisí větší zatížení (jak serveru, tak komunikace mezi serverem a klienty) při zpracovávání požadavků od klientů. V SOA je situace odlišná. Každá entita, která je schopna komunikovat podle předepsaných pravidel (protokolů) je považována za spotřebitele služeb, přičemž každý spotřebitel je obecně považován také za službu. Proces zpracování je v SOA silně distribuovaný. Každá služba poskytuje omezenou množinu funkcí, s tím jsou spojeny menší nároky na zdroje. Navíc v SOA je do zpráv, které si služby posílají, umístěna část „výpočetní“ logiky. To vede k nezávislé a bezstavové povaze služeb.

Dalším rozdílem mezi architekturou klient-server a SOA je rozložení zátěže. V systému založeném na SOA se většinou nachází více služeb poskytujících stejné funkce. Spotřebitel si potom vybírá služby na základě jejich momentálního vytížení.

## 3. Servisně orientovaná analýza a návrh

### Úvod

Základním krokem ve vývoji každého systému je jeho návrh. V kontextu SOA mluvíme o tzv. *servisně orientované analýze a návrhu* (angl. *Service-Oriented Analysis and Design, SOAD*). Stávající návrhové a modelovací techniky, které nabízejí osvědčené a kvalitní postupy jako jsou *objektově-orientovaná analýza a návrh* (angl. *Object-Oriented Analysis and Design, OOAD*), *rámce pro podnikovou architekturu* (angl. *Enterprise Architecture (EA) frameworks, EAf*) a *modelování byznys procesů* (angl. *Business Process Modeling, BPM*) poskytují pouze částečnou podporu pro celý proces vývoje SOA systémů. SOAD je tím pádem určitým hybridním přístupem, který kombinuje výše zmíněné disciplíny s novými prvky (např. choreografie služeb).

### 3.1 Servisně-orientovaný přístup a objektově-orientovaný přístup

V názvu této kapitoly je záměrně použita spojka „a“ místo spojky „vs.“. Toto rozlišení zdůrazňuje, že oba přístupy nejsou výhradně konkurenční. Ve skutečnosti objektově orientovaný (zkr. OO) přístup se často používá při návrhu a implementaci aplikační logiky uvnitř služeb.

Následující výčet aspektů slouží k porovnání obou přístupů:

- Servisně orientovaný (zkr. SO) přístup je založený na volném provázání jednotlivých entit (služeb). Naopak OO přístup klade důraz spíše na přesně definované vztahy mezi třídami, což vede k mnohem těsnějším vazbám mezi entitami (objekty).
- Oba přístupy mají podobný pohled na abstrakci základních entit vytvořenou pro komunikaci mezi těmito entitami. Abstrakci v OO přístupu vytvářejí přesně definovaná rozhraní objektů. V SO přístupu tuto úlohu většinou plní nějaký druh popisného dokumentu (u webových služeb to je popis služby vytvořený v WSDL)
- SO přístup předpokládá, že rozsah činností, které daná služba nabízí, se může měnit. OO přístup definuje objekty, které jsou mnohem více specifické v daném oboru působnosti.
- Aktivita služeb je vyvolána až příchodem nějaké zprávy. Zprávy obsahují kromě dat i logiku zpracování a určitým způsobem řídí činnost služeb. V OO přístupu jsou data svázána s logikou do objektů.

Jednou ze základních vlastností služeb je jejich bezstavovost, naopak zapouzdření logiky a dat do objektů v OO přístupu způsobuje jejich stavovou závislost.

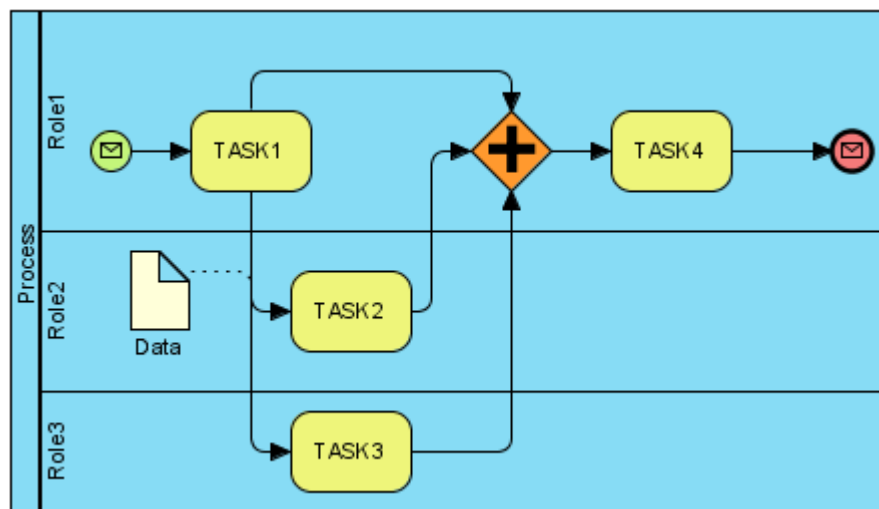
### 3.2 BPM

Před samotným úvodem do BPM je nutné vysvětlit pojem byznys procesu. BP je posloupnost činností, které společně realizují podnikatelský nebo strategický cíl, obvykle v kontextu struktury dané organizace. BPM je potom množina metod, přístupů a nástrojů používaných ke specifikaci a analýze byznys procesů. BPM na jedné straně vytváří abstrakci procesu a na druhé straně poskytuje prostředky pro specifikaci korektních procesů a analýzu jejich vlastností. K samotnému sestavení abstraktního modelu používá BPM několik různých cest, např. *Business Process Modeling Notation (BPMN)*, *Event-Driven Process Chains (EPC)* nebo v poslední době (v souvislosti s rozvojem webových služeb) stále častěji využívaný *Business Process Execution Language (BPEL)*. Tento jazyk specifikuje BP především z hlediska interakcí jeho jednotlivých částí (nejčastěji webových služeb). Více o

BPEL v kapitole 5.2.

Na obrázku 3.1 je zobrazen příklad BP zapsaného v BPMN. Na provedení procesu *Process* se podílí tři role: *Role1*, *Role2* a *Role3*. *Process* je odstartován událostí vyvolanou přijetím zprávy (zelené kolečko s obálkou). Po této události je *Role1* zodpovědná za provedení úkolu *TASK1*. Po dokončení *TASK1* zahájí *Role2* resp. *Role3* provádění *TASK2* resp. *TASK3*. Tyto úkoly jsou prováděny paralelně a *Role1* čeká na jejich dokončení. Po synchronizaci (oranžový kosočtverec s černým křížem) se provede *TASK4* a proces je ukončen odesláním zprávy (červené kolečko s obálkou). Všimněme si, že volání *TASK2* a *TASK3* je spojeno s přenosem dat *Data* z *TASK1* do těchto úkolů,

obr. 3.1  
Příklad  
byznys  
proces  
diagramu



Všechny současné přístupy BPM mohou být výchozím krokem v SOAD, přesto plná síla BPM se uplatní až ve fázi sestavování služeb do byznys procesů, protože v počátečních krocích návrhu je nutná určitá inicializace (např. identifikace jednotlivých služeb), pro kterou nemá BPM prostředky.

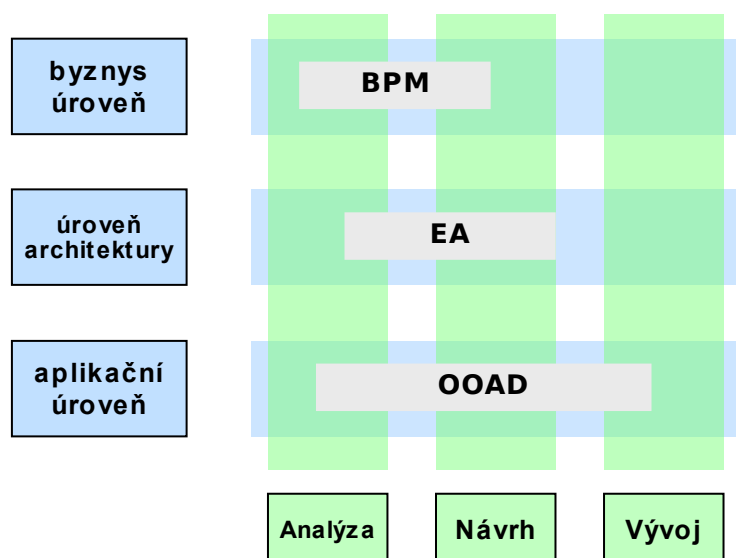
### 3.3 EAF

Pod pojmem podniková architektura se skrývá množina nástrojů, které popisují stávající (a případně budoucí) strukturu a chování procesů v rámci organizace. Procesy se dělí do čtyř úrovní: obchodní, informační, aplikační a technické. Cílem EA je optimalizovat stávající procesy tak, aby jejich prováděním bylo co nejefektivněji dosaženo strategických cílů dané organizace. K tomu se používají tzv. rámce pro podnikovou architekturu. Tyto rámce podle svého účelu (tj. pro jaký typ organizace byly vytvořeny) zavádějí referenční architektury, které se aplikují na strukturu dané organizace, aby se dosáhlo požadovaných změn.

Jelikož současná SOA řešení nevznikají na zelené louce, nýbrž dekompozicí a integrací stávajících systémů, jeví se jako jeden z možných kroků v SOAD právě cesta rozvíjení EAF. Nevýhoda použití EAF spočívá především v jejich jednostranném zaměření, např. rámec určený především pro optimalizaci technických procesů nemůže být použit na úrovni služeb nebo obchodních procesů. Současná problematika SOA, resp. SOAD, je tak rozsáhlá, že nemůže být celá pokryta jedním EAF.

Na obrázku 3.2 je zobrazeno v jakých fázích SOAD se uplatní výše zmíněné přístupy. Horizontální osa popisuje jednotlivé fáze životního cyklu projektu. Vertikální osa vymezuje jednotlivé úrovně abstrakce, kde se uplatní modelovací techniky.

obr. 3.2  
Uplatnění  
OOAD, Eaf a  
BPM na  
jednotlivých  
úrovních  
abstrakce v  
SOAD  
převzato z [8]



### 3.4 Choreografie a orchestrace služeb

Oba pojmy – *choreografie* a *orchestrace* – souvisí s modelováním vzájemné spolupráce mezi službami. Odlišují se tím, kde spočívá logika, která řídí spolupráci mezi službami. U choreografie je logika rozdělena mezi všechny zúčastněné služby. Pro vytvoření modelu choreografie je nutné nejdříve popsat interakce mezi všemi spolupracujícími službami a následně popsat vztahy mezi těmito interakcemi. Choreografie nesleduje akce, které provádí poskytovatel služeb, aby zajistil provedení nabízené služby. Modely choreografie webových služeb lze vytvářet např. pomocí *Web Services Choreography Description Language (WS-CDL)*, viz kapitola 5.1.

Zatímco choreografie se zabývá více cílem spolupráce než jednotlivými službami, orchestrace popisuje jaké úkony (na úrovni služeb) provádí poskytovatel služby, aby zajistil provedení dané služby. Logika v tomto případě spočívá pouze na straně poskytovatele služby. Pro modelování orchestrace je nutné popsat interakce, které má poskytovatel (resp. služba) se všemi zúčastněnými stranami a akce, které probíhají v rámci poskytovatele.

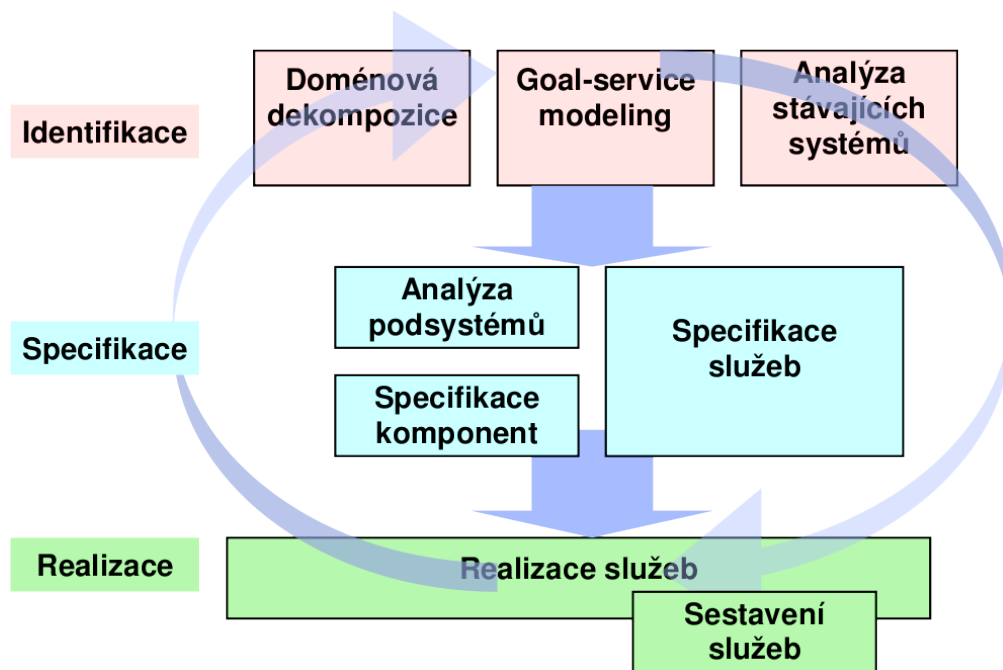
### 3.5 Vývojový cyklus SOAD

SOAD umožňuje zapouzdřit techniky pro modelování, analýzu a návrh konkrétních prvků v jednotlivých vrstvách architektury (viz. kapitola 2.4) a další aktivity nutné pro vytvoření systému založeného na SOA. Přesněji řečeno SOAD pokrývá následující:

- Identifikaci, specifikaci a realizaci komponent, služeb a choreografie služeb.
- Specifikaci požadavků z pohledu spotřebitele a poskytovatele služeb
- Vývoj aplikací používaných v různých kontextech napříč několika podniky

Obrázek 3.3 zobrazuje cyklus vývoje systému postaveného na SOA. Popis jednotlivých fází SOAD následuje.

obr. 3.3  
Fáze SOAD  
převzato z [2]



### Identifikace služeb

Pro identifikaci služeb se používá kombinace tří přístupů: analýz *shora-dolů* (angl. *top-down*) a *zdola-nahoru* (angl. *bottom-up*) a *modelování cílových služeb* (angl. *goal-service modeling*).

Prvním krokem v identifikaci služeb bývá analýza *shora-dolů*. Tento přístup (někdy označovaný jako doménová dekompozice) se zabývá byznys procesy probíhajícími v daném podniku a vztahy mezi nimi. Tyto procesy jsou rozloženy na jednodušší procesy a podprocesy a následně jsou vyjádřeny v podobě byznys případů použití. Tyto případy použití jsou vhodným prostředkem pro specifikaci obchodních služeb<sup>1</sup>.

Dalším krokem je analýza *zdola-nahoru*. V této fázi se analyzují stávající řešení (existující aplikace, podsystémy, programové moduly, uživatelská rozhraní, atd.) a hledají se vhodné kandidáti na komponenty a primitivní služby<sup>1</sup>.

Po analýze stávajícího systému přichází poslední krok, modelování cílových služeb. Tento přístup má za úkol odhalit služby, které nebyly identifikovány v předchozích krocích, upřesnit identifikované služby a případně redukovat počet služeb s ohledem na požadované byznys cíle, výkonnost, bezpečnost a další ukazatele.

### Klasifikace služeb

Dalším krokem v SOAD, po identifikaci služeb, je klasifikace služeb. Služby mohou být tvořeny komponentami nebo jinými službami. Na základě těchto vlastností je vytvořena určitá hierarchie služeb. Klasifikace pak pomáhá při sestavování nových služeb ve vytvořené hierarchii a zamezuje vytvoření velkého počtu primitivních služeb – takový systém by byl neefektivní jak z hlediska správy tak výkonnosti.

### Analýza podsystémů

Analýza podsystémů zpracovává byznys případy použití vzniklé doménovou dekompozicí. Základem zpracování je vytvoření objektových modelů pro funkční jednotky a podsystémy, ze kterých budou vytvořeny komponenty.

<sup>1</sup> Obchodní služby jsou tvořeny primitivními službami a poskytují větší míru abstrakce při přechodu mezi procesy a službami.

## **Specifikace komponent**

V této fázi probíhá návrh struktury a komunikace komponent. Komponenty jsou sestavované z podsystémů navržených v předchozím kroku.

## **Sestavení služeb**

Jak již bylo zmíněno, každý podsystém je reprezentovaný nějakou komponentou, která zpřístupňuje jeho funkce. Tyto komponenty jsou následně sestavovány do služeb. Fáze sestavení služeb také souvisí s přiřazením komponent a služeb do příslušných vrstev architektury.

## **Realizace služeb**

V tomto kroku se rozhodujeme, jestli pro splnění požadavků je vhodnější použít již existující službu (pokud taková existuje) nabízenou nějakým poskytovatelem služeb nebo zda vytvoříme službu novou. Nová služba může být vytvořena "od začátku" nebo sestavena z existujících řešení. Na této úrovni se také zabýváme bezpečností, správou a monitorováním služeb.

V praxi se většinou výše zmíněné kroky provádějí paralelně za účelem zkrácení doby vývoje požadovaného systému. Přesněji řečeno analýza shora-dolů (modelování a dekompozice byznys procesů, analýza podsystémů) probíhá paralelně s přístupem zdola-nahoru (analýza stávajících systémů, realizace služeb). Paralelně k těmto dvěma přístupům probíhá ještě modelování cílových služeb, které se snaží provázat služby a požadované byznys cíle.

## 4. Webové služby

### technologie webových služeb

*Webové služby (Web Services, WS)* jsou neznámější a nejpoužívanější reálnou implementací architektury orientované na služby. Jak už sám název napovídá, jedná se o koncept, který ke své činnosti používá internet. Přesněji WS jsou postaveny na následujících technologiích:

- *eXtensible Markup Language (XML)*
- *Simple Object Access Protocol (SOAP)*
- *Universal Description, Discovery and Integration (UDDI)*
- *Web Services Description Language (WSDL)*

Tyto technologie jsou popsány v kapitolách 4.3 (WSDL), 4.4 (SOAP) a 4.5 (UDDI).

### DEF

#### 4.1 Definice WS

Pracovní skupina Web Services Architectures konsorcia W3C zavedla následující pracovní definici: Základní myšlenka spočívá ve spojení dvou dnes široce rozšířených technologií. Těmito technologiemi jsou XML (univerzální jazyk pro popis dat) a HTTP (transportní protokol podporovaný většinou dnešních webových prohlížečů a serverů). Webová služba je potom aplikace identifikovatelná pomocí URI (angl. Uniform Resource Identifier) a její rozhraní a způsob komunikace lze definovat a poskytovat pomocí XML. Webová služba podporuje přímou interakci s jinými softwarovými agenty, kteří ke komunikaci používají zprávy ve formátu XML zasílané prostřednictvím internetových protokolů.

#### 4.2 Vlastnosti WS

- *Nezávislost, samostatnost* – Pro využívání webových služeb není na klientské straně potřeba žádný speciální software, jak už bylo zmíněno, stačí pouze podpora XML a HTTP. Na straně serveru je nutný webový server, v případě implementace např. pomocí Java EE je to aplikační server implementující webový kontejner pro HTTP servlety (servlet je speciální program zpracovávající klientské požadavky na straně severu – oproti appletu, který je určený k běhu na klientské straně).
- *Samo-popisnost* – Ani spotřebitel služby ani její poskytovatel nemají vlastní mechanismy pro určení významu zpráv. Definice a popis zprávy je posílán společně se samotnou zprávou.
- *Vyhledávání služeb v internetu* – Pro vyhledávání a následnou komunikaci WS slouží tyto standardy:
  - SOAP (Simple Object Access Protocol) – Někdy označovaný jako Service-Oriented Architecture Protocol. Více v kapitole 4.4.
  - WSDL (Web Service Description Language) – Popisný jazyk služeb (kap. 4.3).
  - UDDI (Universal Description, Discovery and Integration) – Mechanismus registrů pro vyhledávání webových služeb.

Jelikož WS jsou implementací SOA, k jejich vlastnostem patří i vlastnosti z kapitoly 2.5. Webové služby lze stejně jako služby SOA sestavovat do vyšších celků. Pomocí WSDL a UDDI lze tento proces do značné míry automatizovat.

Jednou ze základních vlastností SOA je volné propojení mezi službami. Aby byla



tato vlastnost splněna, musí být spotřebitelé a poskytovatelé služeb navzájem co možná nejméně závislí. Tuto vlastnost umožňuje právě popis služeb. Každý spotřebitel služeb vytváří zprávy, které bude posílat cílové službě (poskytovateli služeb), právě na základě popisu cílové služby. Daný spotřebitel tímto docílí, že zpráva bude poskytovatelem přijata a správně interpretována.



Popis služby (angl. service description) určuje, jakým způsobem bude konzument nějaké služby komunikovat s poskytovatelem této služby. Specifikuje formát dotazu a odpovědi. Takovýto popis může také obsahovat seznam podmínek, které musí být splněny před/po komunikaci nebo určujících úroveň QoS.

Základním prostředkem pro vytváření popisu webových služeb (angl. web service description) je jazyk XML.



Více informací naleznete na <http://www.w3.org/XML>

## 4.3 WSDL

Organizace W3C zavedla WSDL jako standard pro popis webových služeb. V současné době je ve verzi 2.0. WSDL, potažmo popis služby, slouží k zodpovězení následujících otázek:

- Jaké funkce poskytuje daná služba?
- Kde je daná služba uložena?
- Jak může být s danou službou navázána komunikace?

WSDL je XML formát pro popis webových služeb. V kontextu WSDL je každá služba chápána jako množina koncových bodů (angl. service endpoints). V těchto bodech služba komunikuje se svým okolím pomocí zasílání zpráv (pro jednoduchost si lze koncový bod představit jako rozhraní služby). WSDL poskytuje formální definici koncových bodů. WSDL dokument se sestává ze dvou částí: abstraktního a konkrétního popisu koncového bodu.

### 4.3.1 Abstraktní popis

Abstraktní popis udržuje integritu popisu služby. Obsahuje informace charakterizující rozhraní služby bez ohledu na technologie, kterými je (bude) služba implementována, operační systém, ve kterém bude služba pracovat, a způsobu komunikace.

Abstraktní popis obsahuje následující tři základní oddíly: `interface`, `operation` a `message`. Oddíl `interface` poskytuje abstraktní pohled na rozhraní služby – v rámci tohoto oddílu jsou vyjmenovány poskytované operace (pro jednoduchost si lze tyto operace představit jako veřejné metody objektů v klasickém objektově orientovaném přístupu). Každá operace má samozřejmě vstupní a výstupní parametry. Jelikož webové služby komunikují výlučně jen pomocí zpráv, jsou tyto parametry v podstatě zprávami. Oddíl `operation` tedy zahrnuje prvky `input` a `output`, které tyto zprávy specifikují.

### 4.3.2 Konkrétní popis

Tento popis slouží k navázání logiky popsané v abstraktním popisu na reálnou implementaci a k navázání komunikace na konkrétní protokol.

Konkrétní popis se skládá ze tří základních oddílů: `binding`, `endpoint` a `service`. Oddíl `binding` popisuje požadavky služby pro navázání konkrétního spojení. Jinými slovy `binding` reprezentuje nějakou technologii, kterou služba



**příklad 4.2**  
**WSDL**  
**dokument**  
převzato z [7]

využije pro komunikaci. Webové služby využívají SOAP, ale koncept webových služeb umožňuje použití i jiných technologií. `Binding` může být aplikován přímo na jednotlivé `operation` nebo na celé `interface`. Oddíl `service` seskupuje prvky `endpoint` (v tomto smyslu lze chápat jako port). Každý `endpoint` určuje fyzickou adresu, na které je služba přístupná.

```
<?xml version="1.0" encoding="UTF-8"?>
<description targetNamespace="http://example.com/bank"
  xmlns="http://www.w3.org/2006/01/wsdl"
  xmlns:ns1="http://example.com/bank">

  <interface name="ns1:Bank">
    ...
    <operation name="withdrawFunds">
      <input messageLabel=""/>
      <output messageLabel=""/>
    </operation>
  </interface>
  <binding name="ns1:BankSOAPBinding">
    ...
    <operation ref="withdrawFunds">
      ...
    </operation>
  </binding>
  <service name="ns1:BankService" interface="tns:Bank">
    <endpoint binding="ns1:BankSOAPBinding">
      ...
    </endpoint>
  </service>
</description>
```



**SOAP:**  
**vlastnosti,**  
**struktura**

Více informací na stránce <http://www.w3.org/TR/wsdl20>. (W3C: Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language)

## 4.4 SOAP

Jelikož veškerá komunikace mezi službami je založená na posílání zpráv, musely být zavedeny takové standardy, aby služby mezi sebou komunikovaly jednotným způsobem. Takovýmto standardem se stal SOAP. SOAP je protokol umožňující spotřebiteli služeb komunikovat s jejich poskytovatelem. Tento protokol je nezávislý na typu sítě, podporuje zprávy ve formátu XML a v současné době je ve specifikaci 1.2 od organizace W3C.

### 4.4.1 Vlastnosti SOAP:

- SOAP je vyvíjen se zaměřením na jeho jednoduchost a snadnou rozšiřitelnost
- SOAP je nezávislý na transportních protokolech. HTTP je jen jedním z podporovaných protokolů. SOAP zprávy lze posílat třeba i emailem.
- SOAP je bezstavový protokol
- SOAP je nezávislý na operačním systému

### 4.4.2 Struktura zprávy v SOAP

Každá zpráva dodržující podmínky kladené SOAP je v podstatě *balíček (obálka, angl. envelope)*. Tento balíček obsahuje hlavičku (angl. *head*) a *tělo (angl. body)*.

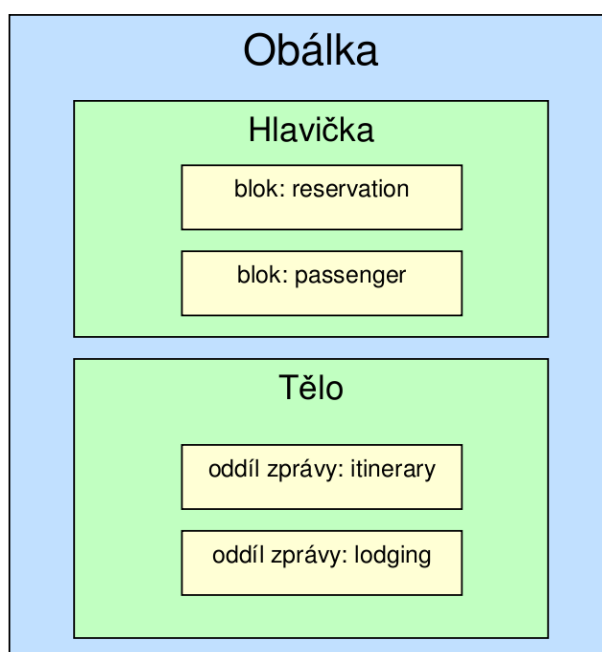
Hlavička se skládá z několika bloků, které obsahují metainformace. Hlavička je

nepovinná (tzn. může být vynechána). Metainformace v sobě ukrývají část komunikační logiky a obecně umožňují zavádět nová rozšíření. Typicky hlavička obsahuje nutné informace pro všechny služby, které mohou zprávu obdržet. Cílová služba potom na základě těchto informací rozhodne o způsobu zpracování zprávy.

Tělo obsahuje samotná data (ve formátu XML). Tělo může také obsahovat sekci pro chyby (angl. *faults*), která obsahuje logiku pro zpracování výjimek. Většinou jsou v této části uloženy jednoduché zprávy, které slouží k odeslání informací o chybě při výskytu výjimky.

SOAP zahrnuje také prostředky pro posílání dat, která jsou těžko popsatelná pomocí XML (binární data, např. obrázky). Takováto data se posílají jako *přílohy* (angl. *attachments*).

**obr. 4.1**  
**Struktura**  
**zprávy**  
**v SOAP**  
převzato z [4]



x+y

**příklad 4.1**  
**zpráva**  
**v SOAP**  
převzato z [6]

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-
envelope">

  <env:Header>

    <m:reservation
      xmlns:m="http://travelcompany.example.org/reservation"
      env:role="http://www.w3.org/2003/05/soap-env/role/next"
      env:mustUnderstand="true">
      <m:dateAndTime>
        2001-11-29T13:20:00.000-05:00
      </m:dateAndTime>
    </m:reservation>

    <n:passenger
      xmlns:n="http://mycompany.example.com/employees"
      env:role="http://www.w3.org/2003/05/soap-env/role/next"
      env:mustUnderstand="true">
      <n:name>John Smith</n:name>
    </n:passenger>
```

```

</env:Header>

<env:Body>

  <p:itinerary
    xmlns:p="http://travelcompany.example.org/
      reservation/travel">

    <p:departure>
      <p:departing>New York</p:departing>
      <p:arriving>Los Angeles</p:arriving>
      <p:departureDate>2001-12-14</p:departureDate>
      <p:departureTime>late afternoon</p:departureTime>
      <p:seatPreference>aisle</p:seatPreference>
    </p:departure>

    <p:return>
      <p:departing>Los Angeles</p:departing>
      <p:arriving>New York</p:arriving>
      <p:departureDate>2001-12-20</p:departureDate>
      <p:departureTime>mid-morning</p:departureTime>
      <p:seatPreference/>
    </p:return>

  </p:itinerary>

  <q:lodging
    xmlns:q=
      "http://travelcompany.example.org/reservation/hotels">
    <q:preference>none</q:preference>
  </q:lodging>

</env:Body>
</env:Envelope>

```

Ve výše uvedeném příkladu 4.1 hlavička (Header) zprávy obsahuje dva bloky. Každý z nich je definován ve vlastním XML jmenném prostoru a obsahuje data týkající se samotného obsahu zprávy, který je obsažen v těle zprávy (oddíl Body). Atribut `env:role`, resp. jeho hodnota, určuje roli uzlů (angl. SOAP intermediary nodes), kterými bude zpráva procházet na cestě k cílovému adresátovi. Položka `env:mustUnderstand="true"` znamená, že uzel musí daný blok plně zpracovat (vzhledem k určené roli), nebo ho nezpracovat a vyvolat výjimku.

Rozhodnutí která data budou uložena v hlavičce a která v těle se provádí až ve fázi vývoje aplikace. Klíčovým faktorem při tomto rozhodování je, že hlavičku zpracovávají uzly po cestě k cíli a na základě dat v hlavičce mohou tyto uzly poskytovat „přidané“ služby.

Oddíl `env:Body` (resp. oddíly `p:itinerary` a `q:lodging`) slouží pro zápis informací, které posílá odesílatel (angl. initial SOAP sender) koncovému adresátovi (angl. ultimate SOAP receiver).



Více informací na stránce <http://www.w3.org/TR/2003/REC-soap12-part0-20030624/>. (W3C: SOAP Version 1.2 Part 0: Primer)

## 4.5 UDDI

Zkratka UDDI skrývá dva významy. Za prvé UDDI byla firma zastřešovaná firmou

Ariba, Microsoftem a IBM, která však v roce 2002 přešla pod konsorcium OASIS. Za druhé UDDI jako specifikace popisuje mechanismus adresáře služeb. V podstatě UDDI není nic jiného než další webová služba, která udržuje jakýsi „telefonní seznam“ webových služeb. Je zde možné vyhledávat podle mnoha různých kritérií (klíčová slova, obory, apod.).

Jelikož UDDI registr byl plně veřejný, záznamy do rejstříků mohli přidávat prakticky kdokoli, což vedlo k faktu, že většina záznamů byla špatná až nesmyslná a v lednu roku 2006 všichni tři hlavní provozovatelé – IBM, Microsoft a SAP – své veřejně přístupné rejstříky vypnuli. Specifikace UDDI nebyla nikdy standardizována a v dnešní době se používá maximálně vnitropodnikově.

## 5. Další jazyky webových služeb

### 5.1 Web Services Choreography Description Language

Web Services Choreography Description Language (WS-CDL) patří do rodiny standardů, která se označuje WS-\*. Tyto standardy se zabývají vzájemnou spoluprací služeb. Kromě WS-CDL sem patří ještě např. WS-Policy, WS-Security, WS-Coordination. WS-CDL je postavena na XML (a souvisejících technologiích, např. XPath) a podporuje WSDL. V současné době je WS-CDL k dispozici ve verzi 1.0.

DEF

*Choreografie* popisuje interakce mezi službami, závislosti mezi těmito interakcemi, včetně řízení toku závislostí (tzn. nějaká interakce musí proběhnout dříve než jiná), časových omezení atd. WS-CDL slouží k vytváření předpisů definujících podmínky a omezení vztahujících se na komunikaci pomocí zpráv v rámci spolupráce nějaké skupiny služeb se záměrem dosáhnout určitý cíl. Tyto předpisy popisují chování celé skupiny služeb z globálního pohledu. Každý účastník spolupráce potom na základě takového předpisu vytváří dílčí řešení cílového problému. Celkové řešení je vytvořeno kombinací těchto dílčích řešení. Výhodou tohoto globálního přístupu je, že zatímco způsob řešení v dílčích (lokálních) systémech se může měnit, globální předpisy zůstávají stejné a součinnost všech zúčastněných je zachována.

DEF

struktura  
WS-CDL  
dokumentu

legenda:

\* značí 0-n  
výskytů oddílu

+ značí 1-n  
výskytů oddílu

? značí  
nepovinnost  
atributu

```
<package
  name="NCName"
  author="xsd:string"?
  version="xsd:string"?
  targetNamespace="uri"
  xmlns="http://www.w3.org/2005/10/cdl">
```

```
<informationType/*
<variable/*
<token/*
<roleType/*
<relationshipType/*
<participantType/*
<channelType/*
<choreography/*
```

```
</package>
```

#### 5.1.1 Struktura WS-CDL dokumentu

Celý dokument je označován jako balíček (*package*). Atributy balíčku *name*, *author* a *version* určují autorství dokumentu. Atribut *targetNamespace* určuje jmenný prostor použitých datových typů v dokumentu. Následující oddíly popisují použitou choreografii:

- *informationType* – definuje typy informací v rámci choreografie. Vytváří abstrakci nad WSDL – neodkazuje se přímo na datové typy
- *variable* – obsahují informace o účastnících spolupráce, mohou být několika typů:
  - proměnné pro výměnu informací – obsahují obsah zpráv, které mají být odeslány resp. které byly přijaty
  - proměnné pro uchování stavu – ukládají stavy jednotlivých rolí (z hlediska globálního pohledu na chování). Např. jednotlivé role se

mohou nacházet v různých stavech podle druhu přijatých/odeslaných zpráv

- proměnné informačního kanálu – tyto proměnné ukládají informace o adrese (URL), kam se mají zasílat zprávy, nebo informace o politice kanálu (bezpečnost, spolehlivost, atd.)
- `token` – `token` je v podstatě alias sloužící k zpřístupnění částí některých oddílů `variable`
- `roleType` – definuje výčet rolí, ve kterých se může daný účastník spolupráce nacházet v rámci choreografie (např. jedna služba může být chápána jako dodavatel i jako odběratel)
- `relationshipType` – definuje vztah (způsob chování) mezi dvěma a více rolemi
- `participantType` – seskupuje ty části chování daného účastníka spolupráce, které musí být implementovány. Jinými slovy, ty role, které budou implementovány v rámci jednoho účastníka.
- `channelType` – určuje způsob spolupráci mezi jednotlivými účastníky, specifikuje, jak a jaké informace budou předávány. V podstatě popisuje informační kanál, který bude vytvořen mezi dvěma účastníky a kterým se budou posílat potřebné informace (zprávy).
- `choreography` – tento oddíl definuje pravidla, která budou použita při výměně zpráv. Jeho struktura je podrobněji popsána v následujícím odstavci

### Choreography

V celém balíčku (`package`) může být definováno několik oddílů `choreography`. Právě jeden z nich je označován jako kořenový. Takový oddíl nesdílí svůj obsah a je zpracovaný (prováděný) primárně. Jak bylo zmíněno jednotlivé `choreography` mohou sdílet svůj obsah jiným `choreography`. Choreografie, které sdílí svůj obsah (chování) jsou definovány buď na úrovni kořenové choreografie, nebo v rámci jiné choreografie a jsou prováděny pouze v rámci jiné choreografie.

**DEF**  
struktura  
oddílu  
`choreography`

```
<choreography name="ncname" root="true"|"false">
  <relationship type="qname" />+
  variableDefinitions?
  Choreography-Notation*
  Activity-Notation
  <exceptionBlock name="ncname">
    WorkUnit-Notation+
  </exceptionBlock>?
  <finalizerBlock name="ncname">
    WorkUnit-Notation
  </finalizerBlock>*
</choreography>
```

Význam jednotlivých atributů a oddílů je následující:

- `choreography`
  - `name` – název choreografie
  - `root` – určuje zda se jedná o kořenovou choreografii
- `relationship` – definuje které vztahy (`relationshipType`) bude choreografie popisovat

- `variableDefinitions` – výčet proměnných použitých v dané choreografii
- `Choreography-Notation` – prostor pro definice vnořených choreografií
- `Activity-Notation` – definuje provedení akce (např. interakce, zápis hodnoty do proměnné, atd.)
- `exceptionBlock` – obsahuje podmínky pro vznik výjimek a způsob jejich ošetření
- `finalizerBlock` – specifikuje „dokončovací akce“ po provedení choreografie (např. při neúspěšném provedení choreografie může obsahovat roll-back)
- `WorkUnit-Notation` – určuje podmínky a způsoby provedení dané operace. V rámci tohoto oddílu bývá typicky uveden Activity-Notation.



Více informací na stránce <http://www.w3.org/TR/2005/CR-ws-cdl-10-20051109/>. (W3C: Web Services Choreography Description Language Version 1.0)

## 5.2 Business Process Execution Language

Business Process Execution Language (BPEL) je jazyk pro popis chování byznys procesů. Základ toho jazyka tvoří WSFL a XLANG. WSFL je zkratka pro Web Services Flow Language, jazyk vytvořený IBM pro popis kompozice webových služeb. XLANG je rozšíření WSDL. Zjednodušeně se dá říct, že popis služby v XLANG se skládá z popisu dané služby ve WSDL a popisu chování služby jako části byznys procesu. Varianta BPEL pro webové služby se označuje BPEL4WS (někdy také WS-BPEL). Tento jazyk rozšiřuje základní interakční schéma mezi službami o podporu byznys transakcí. Byl vyvinut organizací OASIS a v současné době je ve verzi 2.0.



**DEF**  
struktura  
definice  
byznys  
procesu  
v BPEL4WS

### 5.2.1 Struktura definice procesu ve BPEL4WS

Dokument definice procesu ve BPEL4WS má následující kostru:

```
<process>
  <partnerLinks>
    ...
  </partnerLink >
  <variables>
    ...
  </variables>
  <faultHandlers>
    ...
  </faultHandlers >
  <sequence>
    <receive ...>
    <invoke ...>
    <reply ...>
    ...
  </sequence>
  ...
</process>
```

Atributy kořenového oddílu `process` slouží k pojmenování dokumentu a k definici jmenných prostorů použitých v dokumentu.

Oddíl `partnerLinks` je určený pro výčet partnerských služeb, podílejících se na popisovaném byznys procesu. Partnerská služba může vystupovat jako klient



vzhledem k nějaké službě byznys procesu, tj. vyvolávat služby procesu, nebo může být vyvolána byznys procesem. Tuto vlastnost určuje atribut `myRole` (pokud partnerská služba volá službu byznys procesu) resp. `partnerRole` (v případě že proces volá danou službu). Pro použití těchto konstrukcí (resp. celého BPEL4WS) je nutné upravit popis služby vytvořený ve WSDL. Úprava se provádí vložením oddílu `partnerLinkType` do kořenového oddílu `description`. Tento oddíl obsahuje seznam rolí, které může služba hrát v daném byznys procesu. Zjednodušeně řečeno, pokud ve výčtu partnerských služeb v BPEL4W je nějaká role (`partnerRole`), potom musí existovat nějaká služba, která je může tuto roli hrát (podle popisu v WSDL).

Pro ukládání informací (např. stavové informace vzhledem k workflow logice, příchozí zprávy pro pozdější zpracování) jsou v oddílu `variables` definovány proměnné. Typ ukládané informace musí být předem specifikovaný.

Oddíl `sequence` dovoluje zapsat posloupnost aktivit, které mají být provedeny. Těmito aktivitami jsou `receive`, `assing`, `invoke` a `reply`. Jejich popis následuje.

- `invoke` – identifikuje operaci partnerské služby, která má být vyvolána
- `receive` – slouží k získání informací, které služba byznys procesu obdrží po navázání spojení s partnerskou službou. V tomto případě se byznys služba chová jako poskytovatel a čeká na vyvolání od partnerské služby.
- `reply` – zjednodušeně řečeno je určen pro sestavení odpovědi pro dotazující se partnerskou službu
- `assign` – tento oddíl slouží pro přiřazování hodnoty (případně i části hodnoty – pokud se jedná o zprávu) jedné proměnné do druhé proměnné. Celý proces přiřazení se zapisuje pomocí oddílů `copy`, `from` a `to`.

```
<assign>
  <copy>
    <from variable = ...>
    <to variable = ... >
  </copy>
</assign>
```

Pro vytváření podmínek slouží konstrukce `switch`, `case` a `otherwise`. Oddíl `switch` seskupí jednotlivé případy podmínek (`case`). Provedení obsahu oddílu `case` je podmíněno splněním podmínky, která je jako atribut tohoto oddílu. Pokud není splněna ani jedna podmínka ze všech `case`, provede se obsah oddílu `otherwise` (pokud je oddíl přítomen).

```
<switch>
  <case condition = ... >
    ...
  </case>
  <otherwise>
    ...
  </otherwise>
</switch>
```

Správu výjimek má na starost oddíl `faultHandlers`, který může obsahovat několik oddílů `catch`, které podle podmínky zadané ve formě atributu poznají o jakou výjimku se jedná a ošetří ji. V rámci `faultHandlers` může být definován `catchAll`, který slouží jako implicitní ošetření jakékoli výjimky.

```
<faultHandlers>
  <catch faultName = ...>
```

```
...
</catch>
<catchAll>
...
</catchAll>
</faultHandlers>
```



Více informací na stránce <http://www.oasis-open.org/committees/download.php/10347/wsbpel-specification-draft-120204.htm>.  
(OASIS: Web Services Business Process Execution Language Version 2.0)

## 6. Implementace webových služeb

### Úvod

Podle definice jsou webové služby nezávislé na architektuře (hardware), platformě (operační systém, virtuální stroj) nebo implementační technologii. Jedná se však jen o nezávislost specifikace webových služeb, která zajišťuje vzájemnou kompatibilitu různých implementací při komunikaci (volání služeb). Samotné implementace softwarových produktů používajících webové služby jsou plně závislé na použitých implementačních technologiích a jejich omezeních (např. nepřenositelnost řešení na jinou platformu nebo architekturu).

V této kapitole představíme stručný přehled rámců pro implementaci webových služeb. Podrobněji se budeme věnovat knihovnám JAX-WS, které poskytují rámec pro implementaci webových služeb na platformě Java, Enterprise Edition (Java EE).

### 6.1 Rámce a technologie pro implementaci webových služeb

Různé programovací jazyky a vývojové platformy poskytují různé prostředky pro implementaci webových služeb. Tyto prostředky můžeme rozdělit na dva druhy:

1. **knihovny implementující technologie** webových služeb (HTTP, SOAP, XML, UDDI, WSDL, atd.),
2. **knihovny poskytující rámce** pro snadnou implementaci softwarových produktů používajících webových služeb (ASP.NET, JAX-WS, atd.).

#### 6.1.1 Microsoft .NET

Firma Microsoft podporuje práci s webovými službami v rámci platformy .NET. Balík *.NET Framework SDK* implementuje potřebné knihovny i rámec pro tvorbu aplikací, který je dobře integrován v nástroji *Visual Studio .NET*. Webové služby jsou rovněž klíčovou součástí rámce *Compact Framework* pro tvorbu aplikací pro mobilní zařízení, a konzumenti webových služeb jsou podporováni v balíku *Microsoft Office* pomocí *Office Toolkit for Web Services*. Rozšířené technologie webových služeb, jako je např. *Web Service Security* (WS-Security), jsou implementovány v balíku *Web Services Enhancements* (WSE), rozšíření balíku *.NET Framework SDK*.

Samotné webové služby jsou implementovány pomocí *ASP.NET*, jako *.aspx* soubory. Následuje ukázka implementace webové služby Math:



**příklad 6.1**  
**webová**  
**služba**  
**MathService**  
**v jazyce**  
**VisualBasic**  
**v ASP.NET**

```
<%@ WebService Language="VB" Class="MathService">
```

```
Imports System  
Imports System.Web.Services
```

```
Public Class MathService : Inherits WebService
```

```
    <WebMethod>  
    Public Function Add(A As Integer, B As Integer) As Integer  
        Return A + B  
    End Function
```

```
End Class
```

#### 6.1.2 SUN Java Enterprise Edition (Java EE)

Také firma SUN, v rámci platformy *Java Enterprise Edition* (Java EE), se snaží poskytnout co nejjednodušší a přitom bohaté prostředky pro implementaci aplikací

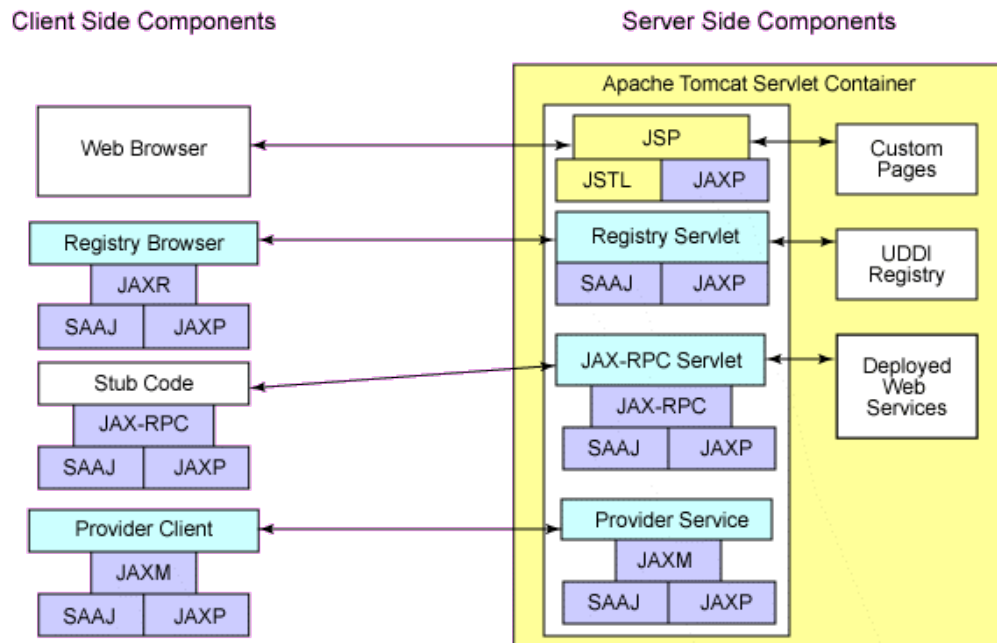
používajících webové služby.

Dřívější verze Java EE (označované jako J2EE) se zaměřovaly především na implementaci jednotlivých technologií webových služeb v balíku *Java Web Services Developers Pack* (WSDP). Dílčí technologie pak programátor používal pro zpracování volání webových služeb v rámci klasické webové aplikace. Webová služba pak byla tvořena HTTP servletem (komponentou webového kontejneru aplikačního serveru zpracovávající obecné požadavky zaslané pomocí HTTP protokolu). Servlet převzal HTTP požadavek a pomocí knihoven pro protokol SOAP a formáty XML analyzoval a interpretoval volání webové služby, které předal příslušnému objektu třídy implementující operace webové služby jako volání její metody.

**Balík WSDP** zahrnuje knihovny poskytující své rozhraní v jazyce Java:

- *Java API for XML Processing* (JAXP) implementující technologie W3C pro práci s XML (tj. SAX, DOM a XSLT),
- *Java API for XML Messaging* (JAXM) umožňuje aplikacím posílat a přijímat asynchronní zprávy v podobě XML dokumentů, přičemž obsluha a kódování zpráv jsou založeny na specifikaci *ebXML Transport, Routing, and Packaging specification*.
- *SOAP with Attachments API for Java* (SAAJ) umožňuje vývojářům aplikací vytvářet a spotřebovat zprávy v souladu se specifikací SOAP 1.1, včetně implementace SOAP příloh (původně bylo SAAJ spojeno s JAXM 1.0, ale od JAXM 1.1 je nezávislou samostatnou knihovnou),
- *Java API for XML-based RPC* (JAX-RPC) implementuje vzdálené volání procedur (RPC) založené na XML podle SOAP 1.1, zahrnuje také nástroje pro práci s *Web Services Definition Language* (WSDL),
- *Java API for XML Registries* (JAXR) implementuje podporu přístup k XML registrům, které jsou používány pro uchovávání informací o publikované webových službách (popisy služeb ve WSDL), např. pro nejčastěji používaný *Universal Description, Discovery, and Integration (UDDI) registry*.

**obr. 6.1**  
**Komponenty WSDL ve vrstvách aplikace**  
převzato z [11]



V současných verzích platformy Java EE je výše uvedený postup konstrukce webové služby zachován (nejen z důvodů zpětné kompatibility a nezavádění duplicitních řešení), ale také výrazně usnadněn pomocí Java API for XML Web Services (JAX-WS). Toto řešení však bude popsáno podrobněji v následující kapitole.

### 6.1.3 Další rámce a technologie

Existuje také nemalé množství knihoven a rámců pro implementaci aplikací s webovými službami. Jedná se o různé knihovny pro konkrétní programovací jazyky, např. Action Web Service pro aplikace v Ruby on Rails, nebo Apache Axis implementaci protokolu SOAP.

## 6.2 Java API for XML Web Services (JAX-WS)

Knihovna JAX-WS poskytuje prostředky pro realizaci XML webových služeb a jejich klientů na platformě Java EE. JAX-WS využívá podpůrné knihovny z balíku WSDP (viz sekce 6.1.2), ty poskytují přenos XML zpráv SOAP protokolem přes HTTP, produkují a přijímají „message-oriented“ a „RPC-oriented“ volání webových služeb, apod. Díky dodržení standardů podpůrnými třídami a díky specifikaci webových služeb JAX-WS respektuje nezávislost na platformě (lze komunikovat s libovolně implementovanými službami, tzn. se službami neimplementovanými v Javě, pomocí JAX-WS, apod.).

**Poskytovatel** služby zapíše webové služby jako Java třídy s poskytovanými metodami označenými pomocí anotací, WSDL popis je pak generován automaticky z definice anotovaného rozhraní třídy, stejně jako obslužný kód vázající vytvořenou Java třídu na podpůrné knihovny WSDP.

**Spotřebitel** vytvoří pomocí JAX-WS lokální proxy pro vzdálenou webovou službu (opět pomocí anotací) a tu transparentně používá. Tato proxy má stejné rozhraní jako třída implementující webovou službu u poskytovatele (převod volání metod proxy na SOAP zprávy je automatický podle WSDL popisu, který musí mít spotřebitel k dispozici).

### 6.2.1 Implementace poskytovatele WS pomocí JAX-WS

Implementace poskytovatele webové služby použitím anotací  
`@javax.jws.WebService` a `@javax.jws.WebMethod`:

- webová služba je realizována jako rozhraní/třída s anotací `@WebService` (tzv. service endpoint interface/service endpoint implementation (SEI)),
- implicitní je definice rozhraní služby společně s implementací (ve třídě), přímo u implementace metod třídy pomocí anotace `@WebMethod`,
- parametr „`endpointInterface`“ umožňuje oddělit definici rozhraní služby, přičemž rozhraní musí rozšiřovat rozhraní „`Remote`“ a označit metody pomocí anotace `@WebMethod`,
- třída implementující službu nesmí být „`final`“ nebo „`abstract`“ a musí mít implicitní bezparametrický konstruktor,
- metody s anotací `@PostConstruct` a `@PreDestroy` mohou obsloužit vznik a zánik instance webové služby (objektu třídy, jeho pravý konstruktor nelze použít, je využíván rámcem JAX-WS),
- metody rozhraní/třídy pro webovou službu (anotace `@WebMethod`) musí být „`public`“ a nesmí být „`static`“ nebo „`final`“,
- typy parametrů a návratových hodnot metod rozhraní/třídy pro webovou

službu jsou omezeny *Java Architecture for XML Data Binding* (JAXB),

- z třídy (tj. z implementace služby) lze generovat popis služby voláním `wsgen -d <output dir> -classpath <cp dir> <SEI class>`

x+y

**příklad 6.2**  
**poskytovatel**  
**webové**  
**služby math**  
**v JAX-WS**

```
import javax.jws.*

@WebService()
public class math {

    private double memory = 0;
    private String ver = "Math Service v0.1";

    @WebMethod(operationName = "getMemory")
    public double getMemory() { return this.memory; }

    @WebMethod(operationName = "setMemory")
    @Oneway
    public void
        setMemory(@WebParam(name = "newValue") double newValue)
        { this.memory = newValue; }

    @WebMethod(operationName = "addToMemory")
    public double
        addToMemory(@WebParam(name = "addValue") double addValue)
        { return this.memory = this.memory + addValue; }

    @WebMethod(operationName = "getVersion")
    public String getVersion() { return this.ver; }

}
```

**Postup zprovoznění poskytovatele webové služby:**

- tvorba kódu třídy s implementací služby,
- kompilace třídy s implementací služby,
- použití nástroje „wsgen“ pro generování popisu služby,
- zabalení zkompilované implementace a popisu do WAR archivu,
- umístění WAR archivu na aplikační server (aplikační server automaticky generuje popis služby vyžadovaný spotřebitelem).

Většina aplikačních serverů poskytuje pomocné akce:

- **Apache Tomcat** publikuje informace o webové službě na adrese <http://localhost:8080/ws-demo/math?Tester> a WSDL popis na adrese <http://localhost:8080/ws-demo/math?wsdl>
- **SUN** umožňuje testování webové služby na adrese <http://localhost:8080/ws-demo/math?Tester>

### 6.2.2 Implementace spotřebitele WS pomocí JAX-WS

Implementace spotřebitele webové služby použitím anotace `@WebServiceRef`:

- klient deklaruje odkaz na webovou službu anotací `@WebServiceRef`, kde

parametr „wsdlLocation“ udává URI na WSDL popis odkazované služby,

- anotace se použije s deklarací proměnné zastupující objekt poskytovatele proxy, bez přiřazení hodnot a s typem proměnné, tj. třídou objektu, nastavenou na třídu implementující službu (u poskytovatele) doplněnou v názvu slovem „Service“,
- objekt poskytovatele proxy se použije k získání proxy (portu služby), použije se metoda „getNamePort“, kde slovo „Name“ je název třídy implementující službu}, proxy se používá jako klasický objekt třídy implementující službu, tzn. práce s webovou službou zastupovanou proxy je plně transparentní, jako práce s lokálním objektem třídy implementující webovou službu,
- z WSDL lze generovat zdroje pro referenci služby v kódu klienta voláním `wsimport -d <output dir> <WSDL file>`

x+y

**příklad 6.3  
spotřebitel  
webové  
služby math  
v JAX-WS**

```
import javax.xml.ws.WebServiceRef;
import cz.vutbr.fit.rychly.wsDemo.service.MathService;
import cz.vutbr.fit.rychly.wsDemo.service.math;

public class MathClient {

    @WebServiceRef
    (wsdlLocation="http://localhost:8080/ws-demo/math?wsdl")
    static MathService service;

    public static void main(String[] args) {
        try { (new MathClient).doTest(args); }
        catch(Exception e) { e.printStackTrace(); }
    }

    public void doTest(String[] args) {
        try {
            System.out.println(
                "retrieving the port from" + service);
            Math port = service.getMathPort();
            System.out.println(
                "invoking the operation on the port");
            Sestem.out.println(port.getMemory());
        } catch(Exception e) { e.printStackTrace(); }
    }
}
```

**Postup zprovoznění spotřebitele webové služby:**

- tvorba kódu tříd spotřebovávajících webovou službu,
- použití nástroje „wsimport“ s WSDL pro generování zdrojů,
- kompilace tříd s spotřebovávajících webovou službu,
- spuštění spotřebitele webové služby.

Celý proces může být usnadněn vývojovým prostředím (IDE).

### 6.2.3 Kombinace JAX-WS s doplňujícími technologiemi

Speciální použití webových služeb:

- *Streaming API for XML (StAX)* umožňuje postupné zpřístupnění XML dokumentu pomocí webové služby (toto není Document Object Model (DOM)),
- *SOAP with Attachments API for Java (SAAJ)* poskytuje volání s přílohami.

### 6.3 Bezpečnost webových služeb v Java EE

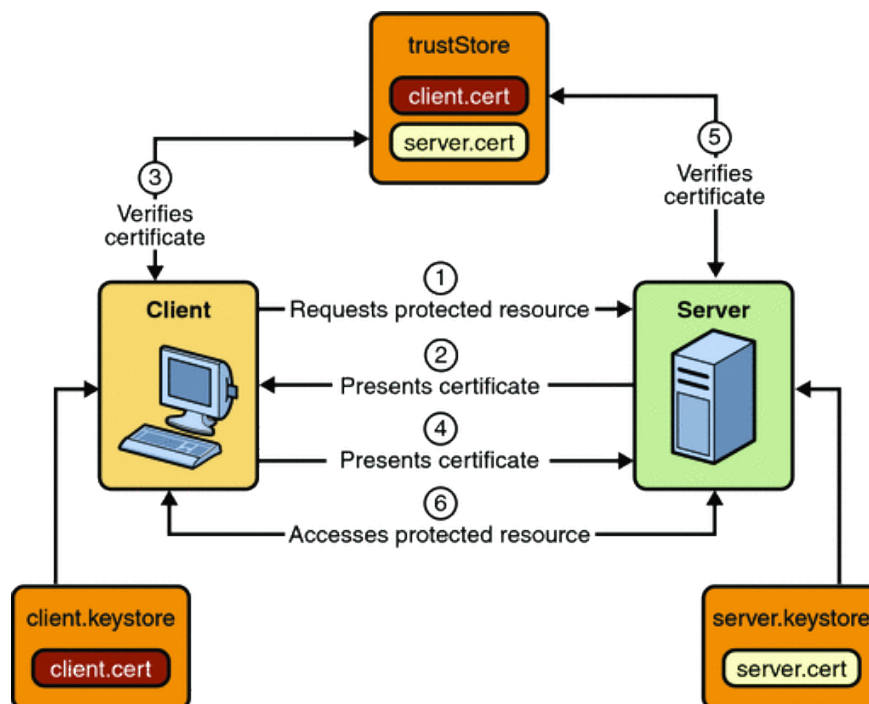
Webová služba je v Java EE implementována jako webová komponenta, proto se její zabezpečení provádí stejnými prostředky, jako zabezpečení jiných webových komponent (servletů a JSP stránek překládaných na servlety).

Autentizaci a autorizaci použití webových komponent provedeme následovně:

- anotace `@javax.annotation.security.RolesAllowed` umožňuje specifikovat role, které mohou přistupovat ke službě (SEI třídě) nebo jejím metodám,
- anotace `@code{javax.annotation.security.DeclareRoles}` umožňuje nastavit role, ve kterých bude služba (SEI třída) vystupovat, to mohou být užitečné při (oboustranném) použití certifikátů},
- aplikační server podle anotací spouštěných tříd a podle konfigurace serveru sám autentizuje uživatele a autorizuje je pro přístup k webovým komponentám s definovanými rolemi (mohou se použít autentizační mechanismy Basic, Digest, Form, Client-Cert, a jiné).

Následující obrázek ukazuje postup autentizace stran při použití systému certifikátů Public key infrastructure (PKI):

obr. 6.2  
Zabezpečení  
webové  
komponenty  
pomocí  
certifikátů  
PKI  
převzato z [12]



Více informací např. v Java EE 5 tutoriálu [12] na stránkách <http://java.sun.com/javase/5/docs/tutorial/doc/>



## 7. Závěr

Architektura orientovaná na služby (Service Oriented Architecture, SOA) patří v současné době mezi nejvíce diskutované moderní technologie návrhu informačních systémů. Umožňuje vhodně dekomponovat a rozmístit informační systém, respektuje základní pravidla softwarového inženýrství, jako jsou modularita, zapouzdřenost, znovupoužitelnost, nezávislost na platformě, atd.

Tato studijní opora se snažila vhodným způsobem prezentovat základy SOA, analýzu a návrh informačních systémů pomocí SOA a ve stručnosti představit webové služby (WebServices, WS), standardizovanou a jednu z nejrozšířenějších implementací SOA.

**Poděkování** Tato práce byla podporována grantem FRVŠ MŠMT číslo FR2233/2007/G1 „Modelování architektur založených na službách“.

## Reference



- [1] Arsanjani, A.: *Service-oriented modeling and architecture*. Dokument je dostupný na URL <http://www-128.ibm.com/developerworks/webservices/library/ws-soa-design1/> (únor 2006)
- [2] Benatallah, B., Dijkman, R., Dumas, M. and Maamar, Z.: *Service Composition: Concepts, Techniques, Tools, and Trends*. V: Stojanovic, Z. and Dahanayake, A. (Eds): *Service-Oriented Software System Engineering*. 2005, Idea Group, ISBN 1-59140-426-6.
- [3] Endrei, M., et al.: *Patterns: Service-Oriented Architecture and Web Services*. IBM Redbooks (2004), ISBN 0-738-45317-X. Dokument je dostupný na URL <http://www.redbooks.ibm.com/redbooks/pdfs/sg246303.pdf> (únor 2006)
- [4] Erl, T.: *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall PTR, 2005, ISBN 0-13-185858-0.
- [5] Newcomer, E.: *Understanding Web Services: XML, WSDL, SOAP, and UDDI*. Addison-Wesley, 2002, ISBN 0-201-75081-3.
- [6] W3C: *SOAP Version 1.2 Part 0: Primer*. Dokument je dostupný na URL <http://www.w3.org/TR/2003/REC-soap12-part0-20030624/> (únor 2006)
- [7] W3C: *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*. Dokument je dostupný na URL <http://www.w3.org/TR/wsdl20> (únor 2006)
- [8] Zimmermann O., Krogdahl P., Gee C.: *Elements of Service-Oriented Analysis and Design*. IBM developerWorks. Dokument je dostupný na URL <http://www-128.ibm.com/developerworks/webservices/library/ws-soad1/> (prosinec 2006)
- [9] Žemlička, M., Král, J.: *Service-oriented architectures as a new paradigm: myths, problems, challenges, benefits*. V: Pour J. (Ed.): *Systems Integration 2004, 2004*, Oeconomica, Praha, ISBN 80-245-0701-3, str. 261-268.
- [10] Kuba, M.: *Web Services*. V: *DATAKON 2006, Proceedings of the Annual Database Conference*. Brno : 2006. ISBN 80-210-4102-1, str. 93-112.
- [11] James McCarthy. *Get ahead with Java Web services. Get up to speed on the Java Web Services Developer Pack*. Dokument je dostupný na URL <http://www.ibm.com/developerworks/webservices/library/ws-jwsdp/> (prosinec 2006)
- [12] SUN. *The Java™ EE 5 Tutorial*. 2007: Sun Microsystems, 4150 Network Circle, Santa Clara, CA 95054, U.S.A.