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 tato publikace učební text do předmětu VPD. 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

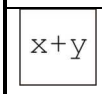
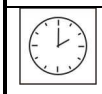
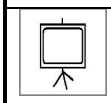
úvod

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	DEF	Definice
 	Reference	 	Zajímavé místo (levá, pravá varianta)
	Souhrn		Rozšiřující látka, informace, znalosti. Nejsou předmětem zkoušky.
	Prezentace pro zobrazení na plátně		

1.3 Cíle a obsah publikace

Tato publikace by měla čtenáři přinést základní informace o architektuře orientované na služby. Více prostoru je věnováno reálné implementaci těchto architektur – webovým službám a jazykům určených pro popis chování služeb.

Druhá kapitola je úvodem do problematiky architektur orientovaných na služby. Tato kapitola vysvětluje základní pojmy a koncepci architektury. Konec této kapitoly je věnován porovnání servisně-orientovaného návrhu s objektově-orientovaným.

Ve třetí kapitole je popsán princip webových služeb a technologie používané v této oblasti.

Čtvrtá kapitola se zabývá popisem služeb, především jazykem pro popis webových služeb, který je založen na XML.

Poslední kapitola je určena jako seznámení s dalšími jazyky popisujícími webové služby. Jsou zde zmíněny WS-CDL a BPEL4WS.

2. Architektura orientovaná na služby

2.1 Úvod

historie SOA

Architektura orientovaná na služby (angl. Service-Oriented Architecture, SOA) a její reálná 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. Sem můžeme zařadit spolehlivé přenosové 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 dosavadní informační systémy.

DEF

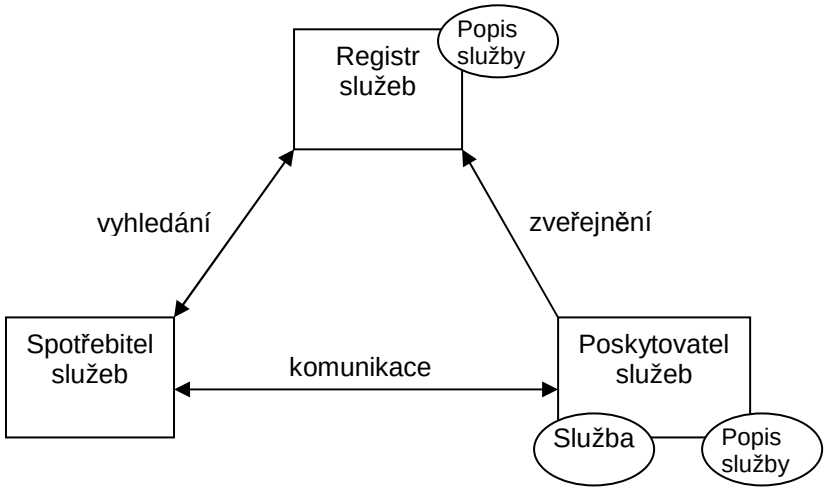
V současné době neexistuje žádná definice, která by přesně vystihovala podstatu SOA. Proto můžeme říci, že SOA je určitým stylem pro vytváření distribuovaných informačních systémů. Jedním dechem je však nutné dodat, že SOA je také nástrojem pro realizaci byznys procesů.

konceptuální model SOA (spotřebitel a poskytovatel služeb)

2.2 Konceptuální model

Tento koncept je založen na interakci mezi dvěma klíčovými entitami: poskytovatelem služeb a spotřebitelem služeb (ang. service provider and service consumer). Model takové interakce zobrazuje obrázek 2.1. Poskytovatel implementuje a nabízí služby. Jak již bylo zmíněno, každá služba je specifikována 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

2.3 Struktura SOA

Abstraktním pohledem na SOA je částečně vrstevnatá architektura, kde každá vrstva představuje jinou úroveň abstrakce. Jak vidíme na obrázku 2.2 výchozí

komponent, služeb a byznys procesů)

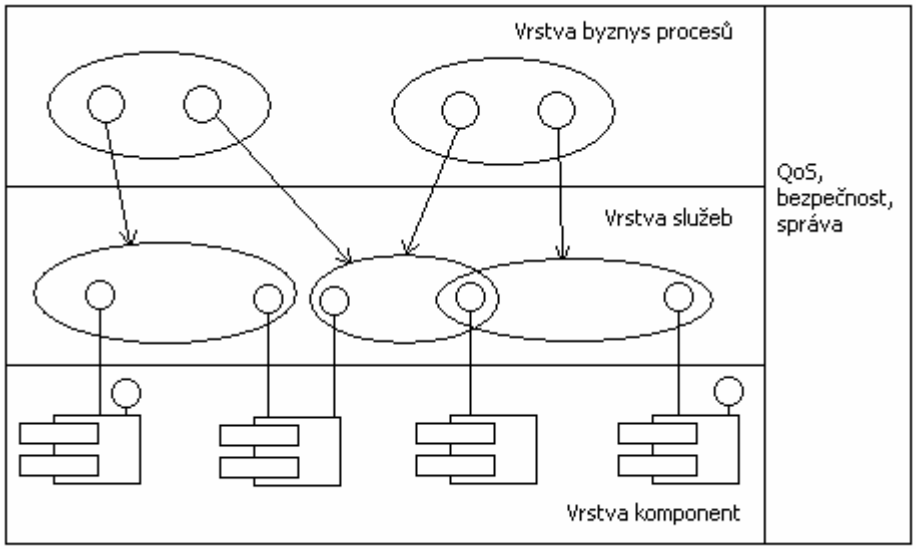
vrstvou je vrstva komponent. Komponenty jsou základní stavební kameny služeb, jsou zodpovědné za realizaci funkčnosti služeb a za zajištění požadované kvality služeb (QoS). Na této úrovni abstrakce jsou komponenty černé skříňky a jejich funkce jsou přístupné pouze přes rozhraní.

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 řídících 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 lze tedy v tomto směru chápat jako jednoduchou samostatnou aplikaci. Proces sestavení služeb do BP je označován jako choreografie služeb.

Napříč těmito vrstvami je řídící vrstva (často bývá rozdělena na několik dílčích vrstev). Tato vrstva podporuje integraci komponent a služeb do větších celků, zajišťuje QoS, správu a monitorování SOA aplikací.

obr. 2.2 Vrstevnatý model SOA



spolupráce služeb: kooperace, agregace, choreografie

2.4 Spolupráce mezi službami

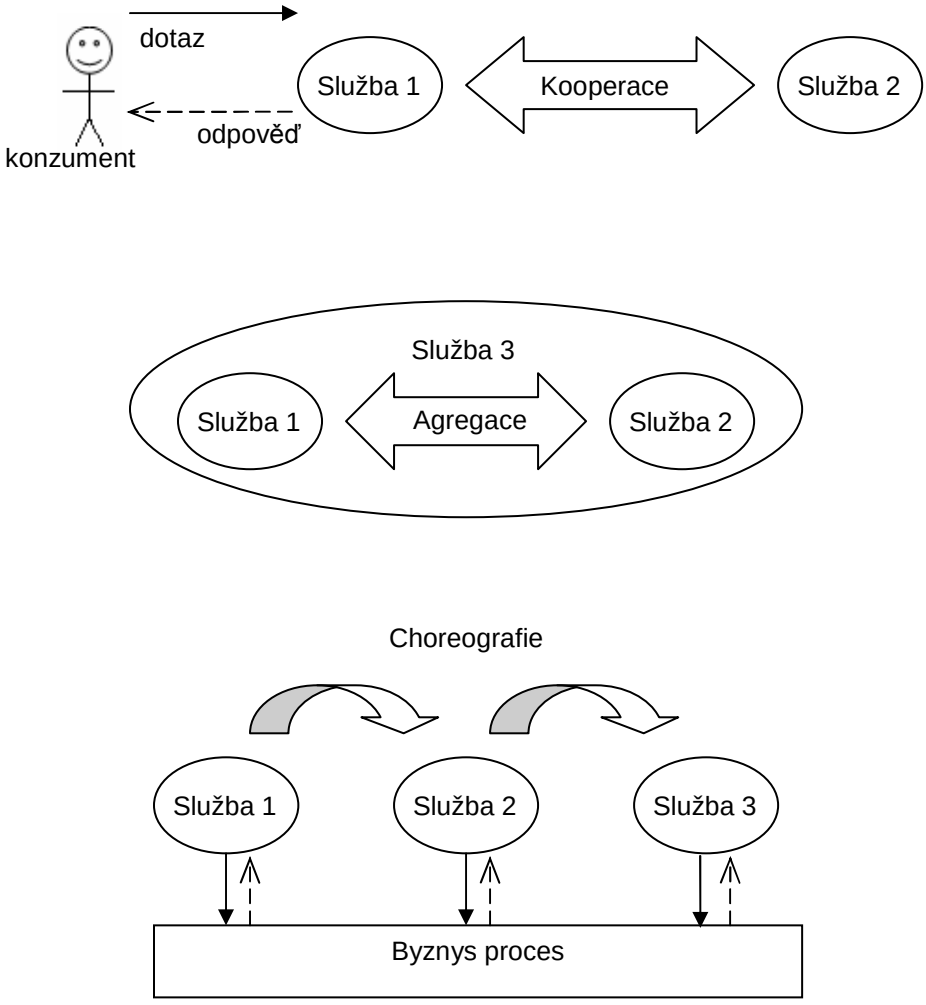
Jak již bylo zmíněno, služby jsou základními jednotkami v SOA a poskytují své prostředky buď přímo cílovému spotřebiteli anebo jiným službám. Služby mezi sebou spolupracují (komunikují) pomocí zasílání zpráv. Obecně lze spolupráci mezi službami rozdělit do tří kategorií: kooperace, agregace a choreografie.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



2.5 Základní vlastnosti servisně orientovaného přístupu

- Volné propojení (loose coupling) – Vztahy mezi službami minimalizují závislosti a pouze poskytují službám informace jedné o druhé.
- Nezávislost – Služby jsou autonomní sebeřídící jednotky.
- Abstrakce – Služby zapouzdřují svoji logiku a okolnímu světu jsou přístupné pouze přes rozhraní.
- Znovupoužitelnost – protože jsou služby specifikovány svým popisem, mohou být navzájem propojovány tzv. ad-hoc. Znovupoužitelnost potom snižuje náklady a zlepšuje údržbu.
- Bezestavovost – Služby se snaží minimalizovat množství uchovávané informace o jiných službách mezi jednotlivými komunikačními cykly.
- Nezávislost na platformě – Služby jsou nezávislé na implementačním

jazyce i na operačním systému.

2.6 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

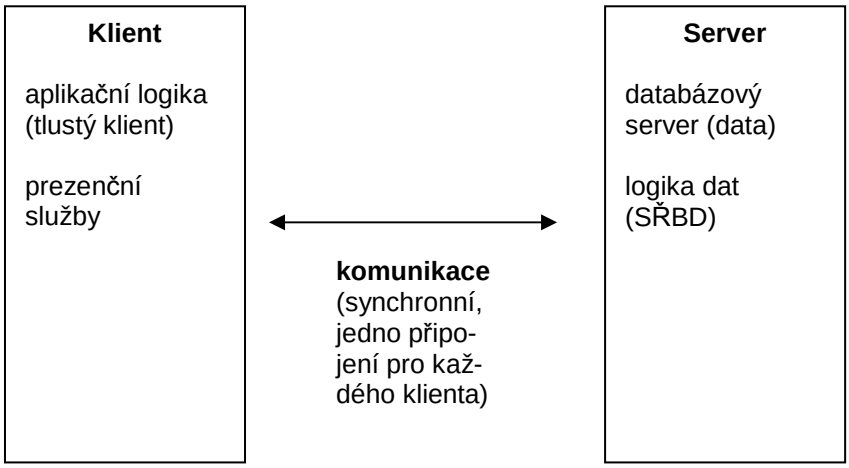
2.7 SOA vs. architektura klient-server

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.

2.7.1 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 (dvou-vrstvá)



2.7.2 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 trigerů 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 čímž 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 bezestavové 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 i podle jejich momentálního vytížení.

2.7.3 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 webových 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 v SO přístupu 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 v SO přístupu je jejich bezestavovost (co možná nejmenší závislost vlastní činnosti na jiných službách), naopak zapouzdření logiky a dat do objektů v OO přístupu způsobuje jejich stavovou závislost.
- Základní vlastností OO přístupu je dědičnost, která v důsledku způsobuje úzké vazby mezi objekty – narozdíl od SO přístupu, který s dědičností na úrovni služeb nepočítá.

3. 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 kapitolach 4.1 (XML), 3.3 (SOAP) a 4.2 (WSDL).

DEF

3.1 Definice WS

Pracovní skupina Web Services Architectures konzorcia 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.

3.2 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 a engine podporující servlety (Servlet je speciální program určený k běhu na straně severu – oproti appletu, který je určený k běhu na klientské straně. WS využívají Java servlety).
- Samopopisnost – Ani spotřebitel služby ani její poskytovatel nemají vlastní mechanismy pro určení významu zpráv. Definice a popis zprávy je posílám společně se samotnou zprávou.
- Vyhledávání služeb v internetu – Pro vyhledávání a následnou komunikaci WS slouží tyto standardy:
    - SOAP (Simple Object Acces Protocol) – Někdy označovaný jako Service-Oriented Architecture Protocol. Více v kapitole 3.3.
    - WSDL (Web Services Description Language) – Popisovací jazyk služeb (kap. 4.2).
    - UDDI (Universal Description, Discovery and Integration) – Mechanismus registrů pro vyhledávání webových služeb.

Jelikož WS jsou implementací SOA, k jejim 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.

SOAP: vlastnosti, struktura

3.3 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.

3.3.1 Vlastnosti SOAPu:

- 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 e-mailem.
- SOAP je bezestavový protokol
- SOAP je nezávislý na operačním systému

3.3.2 Struktura zprávy v SOAPu

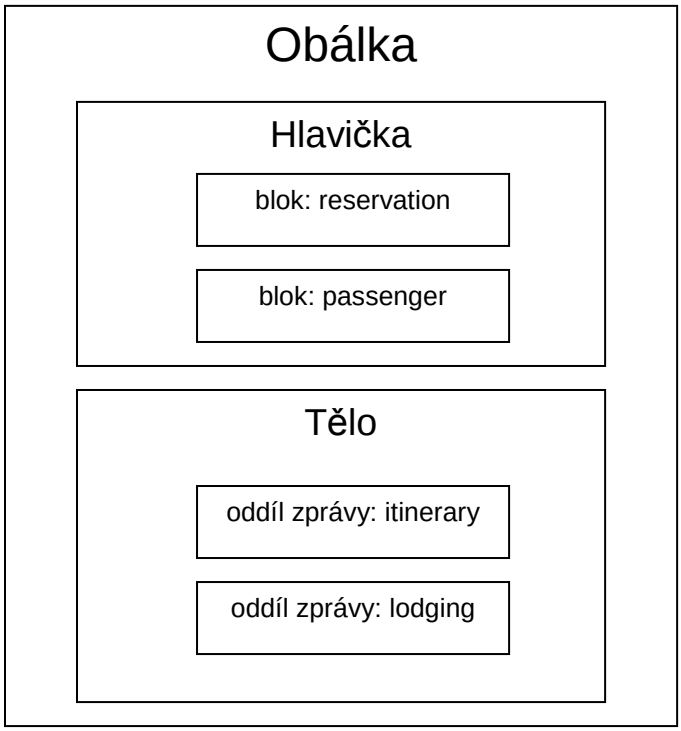
Každá zpráva dodržující podmínky kladené SOAPem je v podstatě balíček (obálka, angl. označení je envelope). Tento balíček zapouzdřuje hlavičku (angl. head) a tělo (angl. body).

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 vyjímek. Většinou jsou v této části uloženy jednoduché zprávy, které slouží k odeslání informací o chybě při výskytu vyjímky.

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. 3.1 Struktura zprávy v SOAPu (vzhledem k příkladu 3.1)



x+y

příklad 3.1 zpráva v SOAPu

```
<?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 3.1 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 buď plně zpracovat (vzhledem k určené roli), nebo ho nezpracovat a vyvolat vyjímku.

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 cíly 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 adresátovy (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. Popis služeb

úvod

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 services description) je jazyk XML.

4.1 XML

Zkratka XML pochází z anglického eXtensible Markup Language. Jak sám název napovídá, XML patří do rodiny značkovacích jazyků. Tyto jazyky (narozdíl od formátovacích jazyků, např. TeX) umožňují označit význam (narozdíl od formátu)

jednotlivých částí popisovaných dat.

historie XML Asi prvním známým značkovacím jazykem byl GML (Generalized Markup Language). Na jeho principu byl v 80. letech minulého století vyvinut jazyk SGML (Standard Generalized Markup Language). Tento jazyk umožňoval definici vlastních značkovacích jazyků - uživatel si dle potřeb mohl vytvořit vlastní sadu značek vhodnou pro daný druh dat (v té době většinou pouze dokumentů). Sady značek jsou vytvářeny pomocí tzv. definice typu dokumentu (angl. Document Type Definition, DTD). Asi nejznámější aplikací SGML je jazyk HTML (Hypertext Markup Language), který se používá pro tvorbu webových stránek. Jelikož koncept SGML je velice komplexní (obsahuje spoustu volitelných parametrů), byla vybrána jeho podmnožina a vytvořen jazyk XML.

4.1.1 Vlastnosti XML

- standardní formát pro výměnu dat – data ve formátu XML mohou být posílána a následně zpracována nezávisle na aplikaci i operačním systému
- otevřený standard – specifikace je volně k dispozici (např. webové stránky W3C)
- jednoduchost – data lze snadno číst i upravovat přímo v XML kódu
- mezinárodní podpora – používá se 32b. znaková sada
- snadná konverze do jiných formátů – pro zobrazení se většinou používají styly – CSS, XSL
- flexibilita a rozšiřitelnost – pokud je potřeba do XML dokumentu přidat nový druh informace, stačí jednoduše vytvořit novou značku

4.1.2 Transformace XML dokumentů

Vzhledem k možnosti vytvářet vlastní značky v XML dokumentech musí existovat nástroje pro převod z jednoho XML formátu do druhého. Takovýmito nástroji jsou např. Extensible Stylesheet Language Transformations (XSLT) a XML Path Language (XPath).

4.1.2.1 XSLT

Transformace v XSLT se provádí pomocí stylů. Jednoduše aplikujeme XSL styl na XML dokument (toto provádí XSLT procesor) a výsledkem je dokument v novém formátu. Tímto způsobem lze v XSLT snadno transformovat XML dokument třeba do HTML formátu. Převod dokumentů v reálném čase naráží na výkonnostní problémy. Postupně se však objevují techniky, které proces několikanásobně zrychlují (např. jednoúčelové kompilátory, kompilované styly).

4.1.2.2 XPath

XPath je samostatný standard W3C, který se používá v několika dalších jazycích včetně XSLT. V XPathu lze zapisovat jednoduché výrazy, které vybírají části XML dokumentu. XML dokument je přitom chápán jako stromová struktura, kde jsou jednotlivé elementy, atributy a text chápány jako uzly.

Výše zmíněné techniky se hojně využívají v prostředí webových služeb, kde se XML dokumenty překládají do jiných XML formátu, jako jsou WSDL a SOAP.

příklad 4.1 XML dokumentu

```
<?xml version="1.0" encoding="UTF-8"?>
<recipe name="bread" prep_time="5 mins" cook_time="3 hours">
  <title>Basic bread</title>
  <ingredient amount="3" unit="cups">
      Flour
  </ingredient>
  <ingredient amount="0.25" unit="ounce">
      Yeast
  </ingredient>
  <ingredient amount="1.5" unit="cups" state="warm">
      Water
  </ingredient>
  <ingredient amount="1" unit="teaspoon">
      Salt
  </ingredient>
  <instructions>
    <step>
      Mix all ingredients together, and knead thoroughly.
    </step>
    <step>
      Cover with a cloth, and leave for one hour in warm room.
    </step>
    <step>
      Knead again, place in a tin, and then bake in the oven.
    </step>
  </instructions>
</recipe>
```

Více informací na stránce http://www.kosek.cz/clanky/swn-xml/index.html (Jiří Kosek: Seriál o XML pro Softwarové noviny)

4.2 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.

4.2.1 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.

4.2.2 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 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 `enpoint` (v tomto smyslu lze chápat jako port). Každý `endpoint` určuje fyzickou adresu, na které je služba přístupná.

příklad 4.2 WSDL dokumentu

```
<?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>
```

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

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 spoluprácí 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 proběhnou 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 webových 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.

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` – obsahuje informace o účastnících spolupráce, může 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 implementovavány. Jinými slovy které role budou naimplementovány 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 oddílu `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 definována buď na úrovni kořenové choreografie nebo v rámci jiné choreografie a jsou prováděna pouze v rámci jiné choreografie (v rámci tzv. zapouzdřující choreografie).

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, ...).
- `exceptionBlock` – Obsahuje podmínky pro vznik vyjímek 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 XLANGu se skládá z popisu dané služby ve WSDL a popisu chování služby jako části byznys procesu. Varianta BPELu 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.

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. atribut `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 oddíly `switch`, `case` a `otherwise`. Oddíl `switch` zapouzdřuje 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 vyjímek má na starost oddíl `faultHandlers`, který může obsahovat několik oddílů `catch`, které podle podmínky zadané ve formě atributu, poznají, o jakou vyjímku se jedná a ošetří ji. V rámci `faultHandlers` může být definován `catchAll`, který slouží jako defaultní ošetření jakékoli vyjímky.

```
<faultHandlers>
  <catch faultName = ...>
    ...
  </catch>
  <catchAll>
    ...
  </cathAll>
</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)

Reference

- Erl, T.: Service-Oriented Architecture: Concepts, Technology, and Design. Prentice Hall PTR (2005), ISBN 0-13-185858-0
- 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)