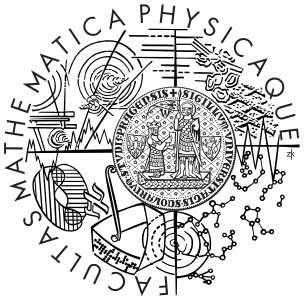
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE



Peter Zelenay

Přednosti a nevýhody protokolu SOAP v praktickém využití

Katedra softwarového inženýrství

Vedoucí bakalářské práce: Prof. RNDr. Jaroslav Král, DrSc.

Studijní program: Informatika (ISS)

2009

Moja vďaka patrí najmä mojim rodičom za to, že mi vytvorili perfektné podmienky pre štúdium a že ma trpezlivo podporovali po celý čas. Chcel by som poďakovať aj Prof. RNDr. Jaroslavu Královi, DrSc. za množstvo vedomostí a praktických skúseností získaných počas štúdia, ktoré určujú moje ďalšie kroky.

Prohlašuji, že jsem svou diplomovou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze dne 7. augusta 2009 Peter Zeleny

Obsah

Název práce: Přednosti a nevýhody protokolu SOAP v praktickém využití
Autor: Peter Zelenay
Katedra (ústav): Katedra softwarového inženýrství
Vedoucí bakalářské práce: Prof. RNDr. Jaroslav Král, DrSc.
e-mail vedoucího: Jaroslav.Kral@mff.cuni.cz

Abstrakt: Prostředí výpočtových systémů v organizácích se vyvinulo z monolitických systémů do sítě množstva menších prvkú spojených do distribuovaných systémú. Tyto prvky potřebují schopnost srozumiteľně vzájemně komunikovat v heterogenním prostředí pořád sa měnící infrastruktury a v kontextu neustálých změn požadavků. Tato práce identifikuje možnosti uplatnění protokolu SOAP v roli komunikačního nosiče v danem prostredí. SOAP jako jednoduchý ale hlavně dobře rozšířitelný protokol postavěný na jazyce XML slibuje široké uplatnění v řešení této problematiky.

Klíčová slova: SOAP, systémová integrace, webové služby

Title: Advantages and deficiences of the SOAP protocol in practical use
Author: Peter Zelenay
Department: Department of Software Engineering
Supervisor: Prof. RNDr. Jaroslav Král, DrSc.
Supervisor's e-mail address: Jaroslav.Kral@mff.cuni.cz

Abstract: Environment of computer systems in organization evolved from monolithic systems to network of many smaller nodes connected in to distributed systems. These nodes need the ability to comprehensible communicate with each other in heterogeneous environment of infrastructure that is still changing and in context of permanent changes of requirements. This work identifies possible application of SOAP as a communication carrier in given environment. SOAP as a simple and very extensible protocol based upon XML promises wide application in solving this problem.

Keywords: SOAP, system integration, web services

Kapitola 1

Úvod

V dobe, ktorú z hľadiska vývoja organizácií charakterizujú pojmy ako fúzia, konzorcium, outsourcing alebo offshoring vzniká potreba prepojenia organizácií na rôznych úrovniach a oblasť výpočtových systémov nieje výnimkou. Požiadavky na vzájomnú kooperáciu úplne odlišných systémov nie sú nové, ale v súčasnosti naberajú na dôležitosti. Samotná komunikácia ako nástroj výmeny informácií nieje dostatočná a musí spĺňať alebo podporovať kľúčové vlastnosti súčasných systémov akými je škálovateľnosť, bezpečnosť, stabilita, tolerancia k poruchám, vysoká dostupnosť. Okrem protokolu SOAP existuje množstvo ďalších prístupov riešenia tohoto problému. Niektoré preverené množstvom aplikácií ako napríklad CORBA, DCOM alebo RMI; iné založené na moderných trendoch a princípoch ako je JSON alebo RESTful web services.

Kapitola 2

Úvod do problematiky

2.1 XML, XML Infoset a menné priestory XML

XML (Extensible Markup Language) je všeobecná špecifikácia jazyka na reprezentáciu štrukturovaných dát. Jedná sa o značkovací (*markup*) jazyk, ktorý umožňuje vytváranie vlastných formátov reprezentácie dát. Pri jeho návrhu bol kladený dôraz na jeho jednoduchosť, škálovateľnosť, možnosť strojového spracovania dokumentu XML. V súčastnosti je to jeden z najrozšírenejších spôsobov reprezentácie dát pri ich prenose medzi rôznymi prostrediami. Presná popis jazyka je na tomto mieste zbytočná a úplná špecifikácia jazyka je k dispozícii ako otvorený štandard [6]. Pripomeniem len, že XML dokument sa skladá z pomenovaných elementov a každý element môže obsahovať text, alebo vnorené elementy. Element môže ešte obsahovať atribúty, čo je dvojica mena atribútu a jeho hodnoty.

Z historických dôvodov je popis jazyka XML priamo spätý s konkrétnou syntaxou a popisuje konkrétny dokument namiesto dátového modelu, ktorý tento dokument popisuje. Takýto prístup síce zjednodušil jeho prijatie, ale pre formálnu špecifikáciu ďalších štandardov tento prístup nevyhovuje. Špecifikácia **XML Infoset** tento problém rieši. Zavádza vyššiu úroveň abstrakcie pre dátový model v XML dokumente. Jeho model je stromová štruktúra, kde má každý uzol definované vlastnosti, ktoré popisujú jeho typ a ďalšie nadväzujúce uzly. Takáto reprezentácia umožňuje lepšie spracovanie pomocou programov. Samotný XML dokument je potom označovaný ako jedna konkrétna reprezentácia inštancie XML Infoset. Abstrakcia zároveň umož-

ňuje iné reprezentácie ako napríklad Fast Infoset. Pre jednoduchosť bude pojem XML dokument a predstavovať inštanciu XML Infoset alebo jeho konkrétnu reprezentáciu vo forme XML dokumentu podľa kontextu alebo ak nešpecifikujem inak.

XML dokument môže obsahovať elementy a atribúty z viacerých značkovacích jazykov postavených na XML. Takto je možné využiť existujúce jazyky bez potreby špecifikovania vlastných. Problém môže nastať pri použití rovnakého mena pre element prípadne atribút vo viacerých značkovacích jazykoch. Úlohou **menných priestorov XML** je presne takýmto prípadom zabrániť. Tento koncept umožňuje priradiť element alebo atribút k mennému priestoru. Menný priestor je identifikovaný URI (jednotný identifikátor zdroja). Táto dvojica: lokálne meno a menný priestor zaručujú jednoznačnú identifikáciu elementu a atribútu v dokumente XML. Pri zápise v dokumente XML sa pre skrátenie používajú prefixy pre menné priestory.

Vo výpise na strane 22 je XML dokument, ktorý obsahuje elementy z viacerých menných priestorov. Menný priestor pre správu SOAP má prefix *soap* a menný priestor špecifického jazyka tretej strany má prefix *m*.

2.2 Distribuované systémy

Už prvotnej epoche vývoja informačných systémov, kedy hlavným prvkom boli rozsiahle monolitické systémy sa začali objavovať požiadavky na ich vzájomnú komunikáciu a kooperáciu. Zároveň vedľa výhod akými je napríklad jednoduchší návrh a prehľadnejšia správa bezpečnostnej politiky, kedže jeden systém má väčšinou jednu sadu bezpečnostných zásad majú monolitické systémy aj nedostatky, ktoré sa snažia vyriešiť distribuované systémy. Distribuovaný systém je systém, ktorý sa skladá z viacerých nezávislých uzlov a navonok sa tvári ako jeden ucelený systém. [5] Výhodami konceptu distribuovaných systémov je:

- **Škálovateľnosť** - Schopnosť poskytovať rovnakú kvalitu služby so vzrastajúcim počtom operácií a dát s ktorými sa pracuje.
- **Spoľahlivosť** - Systém ako celok môže pracovať aj v prípade, že nejaká jeho časť zlyhá.
- **Výkonnosť** - Výkon distribuovaného systému môže presiahnuť technologické limity jeho jednotlivých uzlov.

Jednotlivé uzly distribuovaného systému, ktoré navonok pracujú ako jeden systém sa skladajú z troch logických častí.

- **Platforma** - je individuálne prostredie, v ktorom vykonáva svoju funkciu aplikačná vrstva distribuovaného systému. Jednou z motivácii distribuovaných systémov je umožniť komunikáciu a kooperáciu uzlov v heterogénnom prostredí. Každý uzol môže byť postavený na úplne odlišnej platforme z hľadiska hardvéru, operačného systému, architektonického návrhu prípadne použitého programovacieho jazyka a jeho prostredia.

- **Middleware** - je softvérová vrstva, ktorá spája komponenty a aplikácie. Umožňuje komunikáciu medzi jednotlivými uzlami systému transparentne od toho, či sa uzly nachádzajú na jednom fyzickom zariadený, rovnakom operačnom systéme, či sú postavené na rovnakej platforme. Skrýva celkovú heterogenitu prostredia, v ktorom je distribuovaný systém postavený. Uzly spolu komunikujú rovnakým spôsobom ak sa nachádzajú na jednom počítači alebo je každý v inej geografickej lokácii a sú postavené na úplne odlišných platformách a architektonických princípoch.

- **Aplikačná vrstva** uzlu je samotná komponenta uzlu, ktorá obsahuje aplikačnú logiku ako súčasť distribuovaného systému. Môže komunikovať s ďalšími komponentami prostredníctvom *middleware* a súčasne využívať zdroje špecifické pre danú platformu.

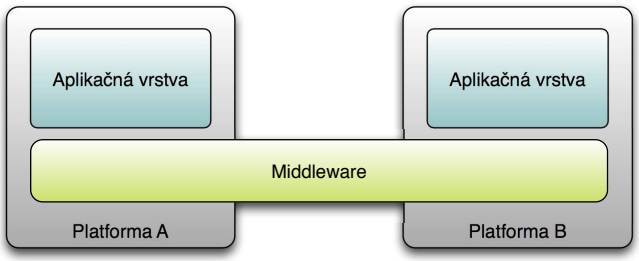


Diagram 2.1: Distirbuovaný systém

2.3 Servisne orientovaná architektúra

Servisne orientovaná architektúra (Service-oriented architecture; SOA) je sada pravidiel a princípov riadiacich proces vývoja a integrácie počítačových systémov. Táto architektúra zapuzdruje funkčnosti systému ako služby. Tieto služby sú vonkajším rozhraním jednotlivého systému prístupné ďalším systémom v organizácii prípadne tretím stranám. SOA vyžaduje slabú viazanosť služieb s operačným systémom a platformou na ktorej je implementácia služby realizovaná. Každá služba implementuje jednu akciu. Ich vzájomná interakcia prebieha na transportnom protokole, ktorý pomocou meta-dát presne špecifikuje akým spôsobom si služby dáta predávajú. Spájaním vstupov a výstupov viacerých služieb alebo ich koordináciou viacerých služieb je možné vytvoriť novú komponentu, ktorá poskytuje dodatočnú funkčnosť. Tento proces sa označuje ako **orchestration**. Novo vzniknutá komponenta môže byť znova sprístupnená ako služba a byť použitá v ďalšom procese. SOA vyžaduje dostatočne silné prostriedky na zápis meta-dát, ktoré formálne špecifikujú charakteristiky služieb a formát dáta, ktorý je potrebný pre ich komunikáciu.

2.4 Vymedzenie problému

Máme informačné a počítačové prostredie nejakej organizácie. Toto prostredie sa skladá z množstva jednotlivých systémov. Tieto systémy sú rôzne zo všetkých možných hľadísk. Jednotlivé elementy prostredia môžu byť monolitické systémy, časti alebo celý distribuovaný systém. Každý element môže z hardvérového pohľadu prestavovať inú architektúru. Elementy môžu používať rôzne operačné systémy. Existuje množstvo platforiem nad ktorou sú vytvorené jednotlivé systémy. Systémy sú vyvíjané v rôznych programovacích prostrediach a v rôznych časových obdobiach. Tieto systémy sú umiestnené v rôznych geografických lokáciách a nie sú vzájomné prepojené len v rámci jednej podsiete z pohľadu protokolu TCP. Dodatočne niektoré komunikujú so systémami iných organizácií mimo infraštruktúry danej organizácie. V takomto heterogénnom prostredí, ktoré v čase postupne vznikalo a rástlo s vývojom organizácie a zmenou potrieb, ktoré boli na neho kladené, máme rozhodnúť a navrhnúť architektonický rámec pre dodatočné prepojenie existujúcich systémov, nahradenie niektorého zastaralého systému novým, začlenenie nového systému alebo vytvorenie nového rozhrania pre komunikáciu s partnerskými organizáciami.

Možných scenárov, ktoré môžu za týchto podmienok nastať je obrovské množstvo. Táto práca si nekladie za cieľ nájsť dokonalý návrh, ktorý bude najlepší pre všetky prípady ani dokázať, že protokol SOAP je najlepšia voľba v každom scenári. V práci rozoberiem potreby, ktoré sú na architektúru v rôznych životných fázach systému kladené. Tieto kroky sa budú najmä vzťahovať na výber middleware a konkrétne SOAP ako jeho súčasť.

Kapitola 3

Protokol SOAP

Hlavným cieľom SOAP je jednoduchosť a rozšíriteľnosť.[8]

Simple Object Access ProtoCol (SOAP) je protokol pre výmenu informácií v štruktúrovanej podobe. Formálna špecifikácia definuje formát a dátovú štruktúru správy ako XML Infoset, ktorej najčastejšia reprezentácia je formát XML [8]. Špecifikácia definuje protokol ako jednosmerný a bezstavový, ale jeho hlavným cieľom je vytvoriť bázu pre komplexnejší spôsob komunikácie či už medzi dvoma uzlami ako po vzore *request/response* alebo komplexnejšiu schému komunikácie medzi viacerými uzlami v distribuovanom prostredí. Špecifikácia priamo nedefinuje spôsob komunikácie, takže na prenos správ protokolu SOAP môžu byť použité rôzne transportné protokoly.

3.1 Formát správy

Správa protokolu sa skladá z dvoch častí zabalených do obálky (SOAP envelope; *soap:Envelope*). Viz. diagram 3.1. Obsah týchto časti už nieje špecifikáciou definovaný, ale ovplyvňuje spôsob akým je správa predávaná a spracovaná.

Prvá časť správy je hlavička (SOAP Header; *soap:Header*). Táto časť je voliteľná a je určená na prenos informácií, ktoré priamo nesúvisia s hlavným prenášaným obsahom. Tieto informácie majú najmä kontrolný charakter. Vzhľadom na to, že SOAP nieje spojený s konkrétnym transportným protokolom, tak prenos správy môže prebiehať vo viacerých krokoch a správa môže medzi prvotným odosielateľom správy (Initial SOAP sender) a koncovým príjemcom (Ultimate SOAP receiver) prejsť cez viacero medzičlánkov (SOAP intermediary). Všetky články sa súhrne označujú ako SOAP

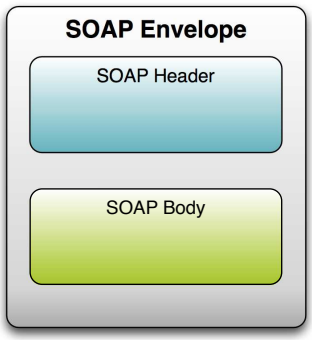


Diagram 3.1: Štruktúra správy SOAP

uzly. Vďaka tomuto návrhu, môže správa počas prenosu zmeniť transportnú vrstvu medzi jednotlivými medzičlánkami. Hlavička správy slúži ako nosič dodatočných metainformácií pre medzičlánky a príjemcu. Jednotlivé elementy tejto časti je možné meniť, pridávať a mazať na každom článku počas transportu správy.

Druhá časť správy je telo (SOAP Body; *soap:Body*), ktorá je nosičom hlavného obsahu správy určenej pre koncového príjemcu. Táto časť správy je povinná.

3.1.1 Dátový model SOAP

Dátový model SOAP (SOAP Data Model) definovaný v [9] reprezentuje dátové štruktúry aplikačnej vrstvy a hodnoty ako orientovaný graf s pomenovanými hranami. Jednotlivé jeho komponenty sú popísané v tejto sekcii. Použitie tohoto modelu nieje povinné a jednotlivý uzol SOAP nemusí implementovať tento model spolu s voliteľnými špecifikáciami pre SOAP kódovanie a SOAP RPC, ktoré sú vysvetlené v ďalších častiach tejto kapitoly.

Orientované **hrany grafu** *začínajú* a *končia* v uzloch grafu. Ak hrana začína v uzle, označuje sa z pohľadu toho uzla za **odchodziu**; ak hrana v uzle končí, tak sa z pohľadu toho uzla označuje za **príchodziu**. Hrany sa môžu odlišovať ich **menom** alebo **pozíciou**. *Pozícia* je úplne ostré usporiadanie hrán na danom uzle. Meno je plne kvalifikovaným menom z pohľadu XML; skladá sa z menného priestoru a lokálnej časti. Dve hrany sú si ekvivalentné, ak ich meno je zhodné a menný priestor chýba u oboch hrán alebo sú ich

menné priestory rovnaké. **Uzly grafu** môžu mať ľubovoľný počet odchodzích hrán. Ak nemá uzol žiadnu odchodziu hranu, tak obsahuje a reprezentuje **jednoduchú hodnotu**. Ak má uzol aspoň jednu odchodziu hranu, tak sa jedná o **zloženú hodnotu**. Ak sú všetky jeho hrany odlíšené len *menom*, tak sa jedná o hodnotu **struct**. Ak sú všetky jeho hrany odlíšené pozíciou, tak sa jedná o hodnou **pole (array)**. Odchodzie hrany v tomto prípade nesmú byť pomenované. *Uzol* grafu, ktorý má jednu príchodziu hranu je označený ako *singe-referenced*, uzol s viacerými príchodzími hranami je z tohoto pohľadu *multi-referenced*.

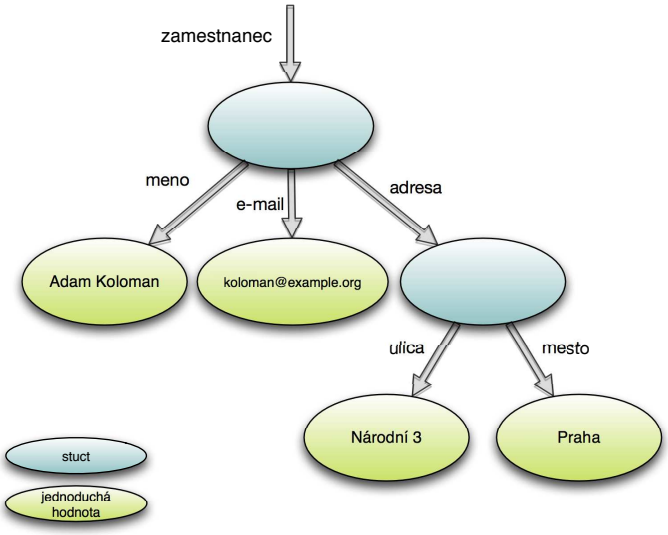


Diagram 3.2: Príklad dátového modelu

3.1.2 SOAP XML

Reprezentácia SOAP správy pomocou XML dokumentu je základný spôsob zápisu a výmeny správ SOAP. Výpis 3.1 zobrazuje jednu takúto správu. Pre označenie obálky, hlavičky a tela správy sa používajú elementy envelope, header, body z menného priestoru XML (XML namespace) `http://shemas.xmlsoap.org/soap/envelope` (prefix: soap). Ako vidno v hlavičke a tele dokumentu sú použité ďalšie menné priestory špecifické pre konkrétnu aplikáciu ako aj dodatočné atribúty z menného priestoru SOAP.

3.2 Kódovanie

Komunikácia v heterogénnom prostredí so sebou prináša známy problém s kompatibilitou prenášaných dát. Typová reprezentácia dát jedného systému nieje kompatibilná s reprezentáciou dát v druhom systéme. Táto inkompatibilita môže byť spôsobená rozličným návrhom architektúry aplikácie, ale rozdielnou platformou použitou pre vývoj systému. Riešenie ponúka použitie jednotného formátu pre reprezentáciu dát, ktorý je použitý ako prostredník pri výmene dát medzi aplikáciami s rôznymi dátovými reprezentáciami. Každý zúčastnený systém má vlastný adaptér, ktorý mapuje dáta medzi svojou špecifickou reprezentáciou dát a spoločnou dátovou reprezentáciou.

V prípade správy SOAP reprezentovanej ako validný XML dokument je zavedený atribút *soap:encodingStyle*. Tento atribút je voliteľný typu a identifikuje sadu pravidiel, ktoré boli použité pre serializáciu SOAP správy alebo jej časti v [8]. Typ atribútu je *xs:anyURI*. Atribút je možné použiť na elementoch hlavičky, elementoch tela alebo na ich pod-elementoch. Tento atribút definuje kódovanie pre element, v ktorom je obsiahnutý, a pre všetky jeho pod-elementy, ak nieje v niektorom z nich prítomný ďalší atribút *soap:encodingStyle*. V tomto prípade sa kódovanie pre tento pod-element a celú jeho vetvu riadi podľa jeho hodnoty. Hodnota môže byť ľubovoľné URI a určuje ju aplikačná vrstva a jej návrh. Pravidlá ako spracovať takto definované dáta sa musia určiť mimo tohoto komunikačného protokolu. Vo výpise 3.1 je príklad vlastného kódovania v hlavičke. Ak atribút nieje použitý, nieje špecifikované žiadne štandardné kódovanie.

SOAP špecifikácia [9] definuje jednej konkrétny spôsob kódovania SOAP správy, ktorej dátový model zodpovedá modelu popísanému v kapitole 3.1.1. Serializačné pravidlá podľa tejto špecifikácie sú identifikované URI `http://www.w3.org/2003/05/soap-encoding`.

3.3 Doručovanie správ

Správa môže pri doručovaní prechádzať cez viacero uzlov SOAP. Jednotlivé uzly sú identifikované pomocou URI. V prípade, že je ako transportný protokol použitý HTTP sa URI mapuje ako URL. Pri prijatí správy uzlom, ktorý nieje jej koncovým príjemcom je len prostredníkom a môže spracovať len obsah hlavičky. O spracovaní elementu hlavičky rozhoduje atribút soap:role. Jeho hodnota je definovaná ako xs:anyURI. SOAP element musí spracovať element hlavičky, ak vyhodnotí, že zodpovedá roli identifikovanej touto URI.

uzol/rola	atribút neexistuje	*žiaden*	*ďalší*	*koncový uzol*
odosielateľ	N/A	N/A	N/A	N/A
prostredník	nie	nie	áno	nie
príjemca	áno	nie	áno	áno

Tabuľka 3.1: Vyhodnotenie atribútu *soap:role*

Postup vyhodnotenia nieje súčasťou špecifikácie. SOAP špecifikuje tri role:

- **žiaden**

  http://www.w3.org/2003/05/soap-envelope/role/none

- **ďalší**

  http://www.w3.org/2003/05/soap-envelope/role/next

- **koncový uzol**

  http://www.w3.org/2003/05/soap-envelope/role/ultimateReceiver

Každý uzol, ktorý dostane element hlavičky s atribútom soap:role hodnotu *ďalší* musí byť daný element schopný spracovať, pretože tejto roli musí zodpovedať každý SOAP uzol. Koncový uzol z definície tiež zodpovedá roli *ďalší*.

Použitie atribútu soap:role je nepovinné a jeho absencia je identická s hodnotou atribútu *koncový uzol*. Tejto roli zodpovedá len uzol, ktorý je koncovým príjemcom správy.

Hodnota *žiaden* pre rolu v elemente hlavičky správy znamená, že žiaden uzol SOAP by nemal obsah elementu spracovať. Na obsah elementu sa však môžu odkazovať iné elementy v hlavičke, ktoré už spracovať možné je.

Telo správy *soap:Body* nemá atribút *soap:role*, pretože telo je vždy určené pre rolu *koncový uzol*. V tomto zmysle je vlastne telo len špecifický prípad elementu hlavičky určeného len koncovému uzlu.

Tabuľka 3.1 sumarizuje štandardizované role vo vzťahu k uzlom. **Áno** znamená, že uzol zodpovedá danej roli.

Atribút **soap:mustUnderstand** je nepovinne a určený pre elementy hlavičky správy. Určuje ďalšie kroky pre spracovanie elementu. Typ atribútu je *xs:boolean*. Tento atribút ma za cieľ zabezpečiť, že elementy hlavičky, ktoré sú dôležité pre aplikáciu nebudú ignorované uzlami SOAP. Hodnota atribútu *true* znamená, že uzol musí spracovať element presne podľa jeho špecifikácie.

V prípade, že táto podmienka nieje splnená, musí uzol generovať chybu. Neschopnosť uzlu spracovať povinný element hlavičky spôsobí, že sa celkové spracovanie SOAP správy zastaví a správa nieje preposlaná na následujúci uzol.

Telo správy *soap:Body* nemá atribút *soap:mustUnderstand*, pretože telo musí byť spracované príjemcom.

SOAP verzia 1.2 pridáva dodatočný voliteľný atribút pre elementy hlavičky **soap:relay**. Typ atribútu je *xs:boolean*. Hodnota *true* definuje, že element hlavičky musí byť preposlaný ďalej, ak ho súčasný uzol nespracuje.

Pravidlá pre spracovanie SOAP správ vyžadujú, že elementy hlavičky, ktoré sú spracované, musia byť zo správy odstránené. (Samozrejme ľubovoľná hlavička môže byť uzlom pridaná v nezmenenej forme.) Ak je element hlavičky určený danej roli, ale nieje spracovaný, tak musí byť zo správy pred odoslaním odstránený.

Najčastejším dôvodom nespracovania elementu hlavičky je to, že jej daný uzol prosto nerozumie. V špecifických prípadoch môže byť vhodné, aby sa elementy hlavičky predávali aj cez uzly, ktoré daný uzol spracovať nemôžu, ale spĺňajú rolu, pre ktorú je element určený. Napríklad rola *ďalší*, ktorú splňuje každý uzol. Označenie elementu atribútmi *soap:role="next"*, *soap:relay="true"* a *soap:mustUnderstand="false"* zabezpečí, že element spracuje len uzol, ktorý to dokáže a iné uzly ho len prepošlú ďalej. Funkcia tohoto atribútu sa dala využiť aj v predchádzajúcej špecifikácii vytvorením novej role, ktorú by následne splňovali len uzly schopné danú hlavičku spracovať. Tento atribút to značne uľahčuje.

3.4 Transportný protokol

Prenos správ medzi uzlami SOAP môže prebiehať na rôznych transportných protokoloch. Sada pravidiel pre prenos SOAP správ pomocou konkrétneho protokolu sa nazýva **väzba** (binding). Väzba definuje ako je správa vo forme inštancie XML Infoset serializovaná do konkrétnej formy, ktorá je následne prenesená danou transportnou vrstvou ďalšiemu uzlu, kde je zo serializovanej formy vytvorená originálna inštancia XML infoset bez straty informácie. Typickým príkladom takejto väzby je SOAP HTTP binding. Je to prenos správy pomocou protokolu HTTP, kde je správa serializovaná ako validný dokument XML. Správa je následne prenesená napríklad pomocou metódy HTTP POST k ďalšiemu uzlu.

Väzba zároveň umožňuje poskytovať **dodatky** (feature), ktoré potrebuje daná SOAP aplikácia. *Dodatok* je sada funkčností potrebných pri komunikácii medzi dvoma uzlami SOAP. Dodatok je identifikovaný pomocou URI, takže je zabezpečené, e všetky uzly pracujú s rovnakou sémantikou dodatku. V prípade komunikácie na princípe *request/response* je príkladom takýchto funkčností potreba spojiť dotaz s odpoveďou. Ďalej môže ísť o potrebu zabezpečenej komunikácie cez šifrovaný kanál a podobne.

Niektoré *dodatky* môžu byť priamo implementované v transportnom protokole. V prípade väzby na protokol HTTP je takouto funkčnosťou spojenie dotazu s odpoveďou, ktorá je mu vlastná a SOAP ju môže jednoducho využiť. V prípade iného transportného protokolu môže byť potrebné takúto funkčnosť realizovať pomocou elementov v hlavičke správy na úrovni SOAP modulu.

3.4.1 SOAP HTTP Binding

Väzba SOAP na transportný protokol je súčasťou špecifikácie SOAP verzie 1.2. Táto časť je voliteľná a uzol SOAP ju nemusí implementovať. Ak uzol túto časť špecifikácie splňuje, označuje sa ako uzol, ktorý je v súlade s väzbou SOAP HTTP. Identifikácia cieľového uzla pomocou URI je realizovaná podľa špecifikácie HTTP/1.1 [11]. Spojenie prebieha na nižšej vrstve TCP/IP. Odosielateľ odošle HTTP dotaz a od príjemcu obdrží HTTP odpoveď. Protokol HTTP automaticky spája odpoveď s dotazom. Táto funkcia je využitá pri komunikačnom scenári RPC (Kapitola 3.5.2).

Implementácia väzby SOAP HTTP musí podporovať dodatok **SOAP Web Method Feature** aby umožnil uzlom zvoliť *Web-metódu* pri komunikácii (POST, GET). Tieto dve metódy reprezentujú dva rôzne spôsoby komunikácie v prostredí World Wide Web. Metóda HTTP POST umožňuje prenos dát v tele dotazu s cieľom modifikovať alebo vytvoriť zdroj identifikovaný URI ktorému je správa určená. Dotaz pomocou metódy HTTP GET získava reprezentáciu zdroja identifikovaného cieľovou URI. Z aplikačného hľadiska je možné použiť každú metódu na splnenie oboch cieľov a špecifikácia ani explicitne nevyžaduje dodržiavanie pravidiel. Špecifikácia však dôrazne odporúča použitie metódy HTTP GET len na doručenie správy, ktorá má za cieľ získanie informácie bez zmeny stavu zdroja. Takáto komunikácia sa označuje ako *bezpečná* a *idempotentná*. Na metódu HTTP POST sa nevzťahuje takéto doporučenie a môže byť použitá vždy.

3.4.2 SOAP Email Binding

Príkladom ďalšieho protokolu, ktorý môže byť použitý ako transportná vrstva je SMTP, teda posielanie správ pomocou emailu [10]. Väzba SOAP Email je nenormatívna a má bližšie k demonštrácii možností rozšírenia protokolu SOAP ako k praktickému využitiu. Hlavnými odlišnosťami od väzby SOAP HTTP je absencia garancie doručenia a automatického párovania dotazu-odpovede. Tieto doplnkové funkčnosti musia byť implementované na úrovni SOAP pomocou elementov v hlavičke a zložitejšieho komunikačného scenára v prípade, že by sa mali napríklad začleniť do infraštruktúry používajúcu ako transportný protokol HTTP.

3.5 Komunikácia

SOAP je jednoduchý rámec pre prenos informácie medzi dvoma uzlami. Príkladom je jednoduché odoslanie SOAP správy obsahujúcej len telo správy priamo koncovému príjemcovi bez toho, aby príjemca očakával prijatie ďalšej správy, alebo musel odoslať ďalšiu správu.

Zložitejšie komunikačné scenáre sa definujú ako **dodatok** k protokolu. *SOAP Message Exchange Pattern (MEP)* slúži ako šablóna pre špecifikáciu takéhoto scenára. Okrem formálnych požiadaviek na špecifikáciu scenára MEP musí obsahovať následujúce body:

- Popis životného cyklu správ.
- Popis vzťahov medzi viacerými správami.
- Popis normálneho a neočakávaného ukončenia komunikácie.
- Všetky požiadavky na vytvorenie dodatočných správ
- Pravidlá na doručovanie a spracovanie chybových stavov SOAP, ktoré vzniknú počas komunikácie.

Na základe tejto charakteristiky sa v praxi ujali dva spôsoby komunikácie pomocou protokolu SOAP. Každý reprezentujúci jednu zo spomenutých variant.

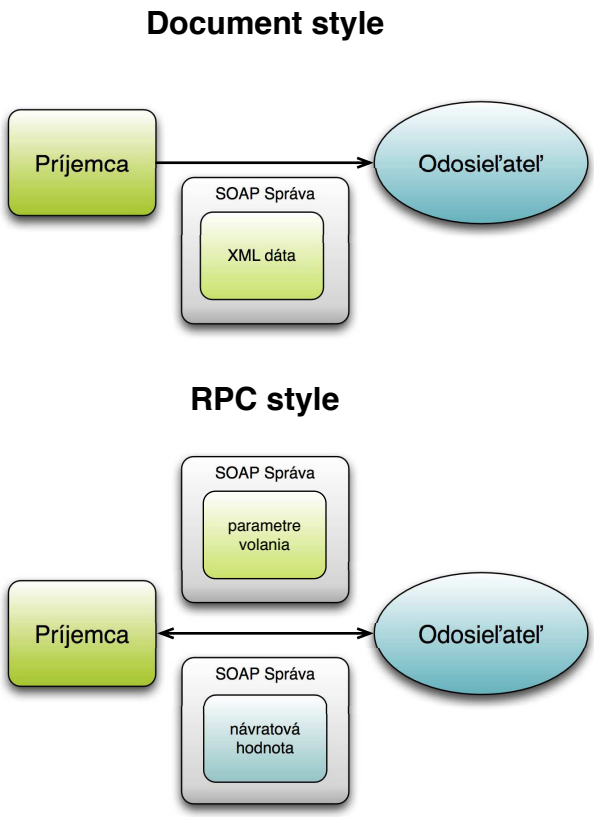


Diagram 3.3: Komunikačné scenáre

3.5.1 Document style

Pri použití tohoto spôsobu komunikácie sa telo SOAP správy skladá z ľubovoľnej XML inštancie. Obsah nemusí zodpovedať žiadnej predom danej XML štruktúre a jedinou požiadavkou, ktorá je na túto časť kladená vyplýva z podstaty SOAP protokolu ako takého a to je nutnosť tvoriť spolu so zbytkom správy validný XML dokument. Vzhľadom na voľné podmienky pre štruktúru tela správy ju príjemca pri spracovaní na úrovni protokolu SOAP nevaliduje a nemodifikuje. Telo je ako celok predaný na vyššiu - aplikačnú úroveň k následnému spracovaniu. Príjem správy nevyžaduje odoslanie ďalšej správy ako odpoveď.

3.5.2 RPC style

Jedným z cieľov návrhu protokolu SOAP je aj podpora pre vzdialené volanie metód (Remote Procedure Call, *RPC*). V dodatku špecifikácie je súčasne ako vzorový návrh komunikačného scenára MEP definovaný SOAP RPC ako nadstavba protokolu SOAP pre vzdialené volanie metód. Predstavuje šablonu pre reprezentáciu volania metód a ich návratovej hodnoty ako správy SOAP. Následujúce informácie umožňujú vytvoriť vzdialené volanie metódy:

- URI identifikujúce cieľ volania. Cieľom volania je v tomto prípade uzol SOAP. Spôsob prenosu URI a samotná identifikácia uzlu je v réžii väzby na transportný protokol. Niektoré väzby môžu mať prenášať URI ako súčasť hlavičky správy, alebo v prípade väzby SOAP HTTP je v hlavičke HTTP dotazu.
- Identifikáciu metódy, ktorá má byť vykonaná. Je to kombinácia menného priestoru špecifikujúceho službu a meno metódy, ktorú túto služba poskytuje.
- Typová identifikácia a hodnoty argumentov volanej metódy.

SOAP RPC volanie môže obsahovať aj dodatočné informácie v hlavičke. Pri použitií komunikačného scenára sa očakáva odoslanie odpovede. Obe správy odoslané od klienta, alebo poskytovateľa služby musia spĺňať následujúce podmienky:

- Formálna signatúra volanej metódy je prenášaná ako telo správy. Dodatočné informácie sú uložené v hlavičke správy.
- Volanie metódy je modelované ako jeden *struct*.
- *struct*, ktorý modeluje volanie obsahuje uzly pre každý argument volanej metódy. Meno uzlu je totožné s menom argumentu a obsahuje hodnota argumentu s ktorou má byť metóda vykonaná na volanom uzle.

Odpoveďou na RPC volanie cez SOAP je ďalšia SOAP správa, ktorá ako telo obsahuje ďalší *struct*, ktorý predstavuje návratový model volania. Tento model obsahuje uzol pre každý návratový argument, ktorý nesie jeho hodnotu.

```
<?xml version='1.0' ?>
<soap::Envelope
 xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
 <soap:Header>
  <t:transaction
     xmlns:t="http://example.com/transaction"
     soap:encodingStyle="http://example.com/encoding"
     soap:mustUnderstand="true">5</t:transaction>
  </soap:Header>
 <soap:Body>
  <m:orderItem
     soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding"
     xmlns:m="http://example.com/shop">
     <m:item>
      <m:code>ADJ908</m:code>
     </m:item>
  </m:orderItem>
 </soap:Body>
</soap:Envelope>
```

Výpis 3.1: RPC volanie

Aj keď použitie SOAP na RPC nieje zviazané so žiadnou transportnou väzbou, je vhodné aby transportná vrstva podporovala komunikáciu na princípe *request/response*. Absencia tejto vlastnosti na transportnom protokole sa dá samozrejme implementovať na úrovni spracovania SOAP správy.

Ako najlepší príklad vhodnej transportnej väzby je SOAP HTTP, kde sa správa volania metódy zabalí ako HTTP dotaz a po vykonaní tejto operácie na cieľovom uzle vráti SOAP správu obsahujúcu návratové argumenty volania ako HTTP odpoveď.

Výpisy 3.1 a 3.2 zobrazujú príklad serializovaných správ do dokumentu XML pre volanie metódy cez SOAP RPC. V hlavičke je použitý dodatočný element ktorý špecifikuje transakčné spracovanie.

3.6 Chybové stavy

V prípade, že nastane chybový stav pri spracovaní správy, SOAP definuje model ako má uzol postupovať. Spracovanie chyby pozostáva z vygenerovania chybovej správy a doručenia tejto správy ďalším zainteresovaným uzlom (pôvodca správy, ďalší uzol). Samotný spôsob doručenia u vygenerovanej

```
<?xml version='1.0' ?>
<soap::Envelope
 xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
 <soap:Header>
  <t:transaction
     xmlns:t="http://example.com/transaction"
     soap:encodingStyle="http://example.com/encoding"
     soap:mustUnderstand="true">5</t:transaction>
  </soap:Header>
 <soap:Body>
  <m:orderItemResponse
     soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding"
     xmlns:m="http://example.com/shop">
          <m:code>ordered</m:code>
  </m:orderItem>
 </soap:Body>
</soap:Envelope>
```

Výpis 3.2: RPC odpoveď

správy nieje súčasťou protokolu SOAP a je závislé na použitom komunikačnom protokole. Tento protokol definuje ako a kedy sa chybová správa doručí.

Telo chybovej správy obsahuje len jeden jediný element *soap:Fault*. Tento element obsahuje dva povinné pod-elementy *soap:Code* a *soap:Fault*. Nepovinné pod-elementy *soap:Detail* obsahujúci špecifické informácie o chybe z aplikačnej vrstvy a *soap:Node*, ktorý cez URI identifikuje uzol SOAP, ktorý túto chybu vygeneroval. Pod-element *soap:Code* sa sám skladá z ďalšieho povinného pod-elementu *sopa:Value* a prípadne nepovinného pod-elementu *soap:Subcode*. Hodnoty elementov sú definované v špecifikácii [8]

Výpis 3.3 ukazuje príklad chybovej správy. Hodnota elementu *soap:Value* je typu *xs:QName* a v tomto prípade znamená, že za chybu je zodpovedný odosielateľ. Ďalšími možnými hodnotami sú: *soap:VersionMismatch*; *soap:MustUnderstand* a *soap:Receiver*. Element *soap:Subcode* je nepovinný umožňuje bližšie špecifikovať vzniknutú chybu. Tento element je navrhnutý ako hierarchický; ako pod-element môže vedľa povinného elementu *soap:Value* voliteľne obsahovať aj vnorený element *soap:Subcode*. Takto je umožnené odosielať chybové kódy viacerých vrstiev aplikácie v jednej jedinej správe. Počas spracovania správy môže nastať chyba už pri spracovaní hlavičky správy. Jedným z monžných príčin je, že element hlavičky označený atribú-

```
<?xml version='1.0' ?>
<soap::Envelope
 xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
 xmlns:rpc="http://www.w3.org/2003/05/soap-rpcpc">
 <soap:Body>
  <soap::Fault>
   <soap:Code>
     <soap:Value>soap:Sender</soap:Value>
     <soap:Subcode>
        <soap:Value>rpc:BadArguments</soap:Value>
     </soap:Subcode>
   </soap:Code>
   <soap:Detail>
    <e:myFaultDetails
        xmlns:e="http://example.com/faults">
        <e:message>Item not found</e:message>
        <e:errorcode>12</e:errorcode>
    </e:myFaultDetails>
   </soap:Detail>
  </soap:Fault>
 </soap:Body>
</soap:Envelope>
```

Výpis 3.3: Chybová správa

tom *soap:mustUnderstand="true"* nieje schopný aktuálny uzol spracovať. Chybová správa omamujúca túto udalosť má rovnako štruktúrované telo, ale hlavička by naviac mala obsahovať element *soap:NotUnderstood*. Tento element má identifikovať konkrétnu časť pôvodnej hlavičky, ktorá chybu spôsobila.

V prípade viacerých elementov hlavičky, ktoré uzol nemôže spracovať by chybová správa obsahovala jeden element *soap:NotUnderstood* pre každý taký element hlavičky identifikovaný rôznymi *qname* atribútmi.

```
...
 <soap:Header>
    <soap:NotUnderstood qname="t:transaction"
       xmlns:t="http://example.com/transaction"/>
 </soap:Header>
...
```

Výpis 3.4: Chybová správa 2

Kapitola 4

Webové služby

Webová služba je softvérový systém navrhnutý poskytovať vzájomnú komunikáciu medzi dvoma strojmi cez sieť. Jej rozhranie je popísané pomocou XML dokumentu. Ďalšie systémy s Webovou službou komunikujú cez toto rozhranie pomocou správ, ktoré sú tiež vo formáte XML dokumentu. V praxi je rozhranie popísané pomocou jazyka WSDL a komunikácia prebieha pomocou protokolu SOAP nad transportnou vrstvou HTTP. [2]

Architektúra Webovej služby rozlišuje tri charakteristické funkčnosti:

- Výmena správ
- Popis Webových služieb
- Publikovanie a sprístupnenie popisu Webových služieb

Operácie poskytujúce tieto funkčnosti pracujú s dvoma konceptami. Jedným je softvérový **modul** Webovej služby, ktorý je samotná implementácia služby a jej **popis** (Web service description) v podobe meta-dát. *Modul* Webovej služby musí byť schopný komunikovať z ďalšími časťami architektúry pomocou daného komunikačného protokolu (SOAP). Formát na *popis* Webovej služby je najčastejšie WSDL (Web Service Description Language) [12]. Pomocou XML Dokumentu definuje *popis* Webovej služby:

- Dátové typy a ich reprezentáciu
- Štruktúru správy ktorou Webová služba komunikuje
- Konkrétny MEP (Message Exchange Pattern - Kapitola 3.5), ktorý sa má pri komunikácii použiť.

- Transportný protokol pre prenos správ. Transportný protokol je napríklad HTTP.

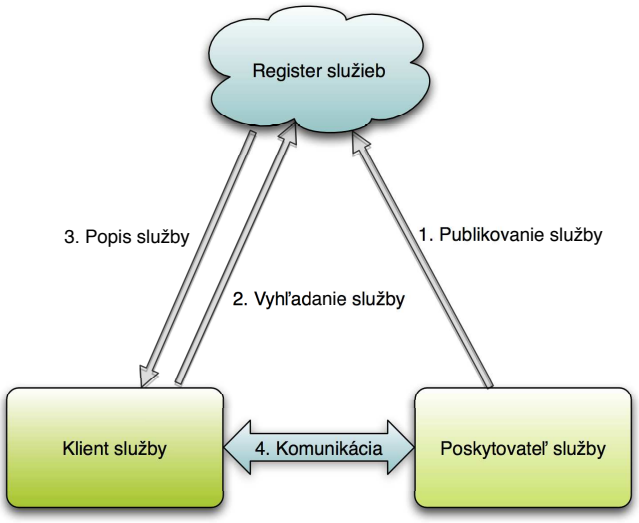


Diagram 4.1: Architektúra Webových služieb

Diagram 4.1 znázorňuje postup interakcie troch rolí, ktoré vystupujú v architektúre Webových služieb. Tieto role sú:

- **Klient služby** inicializuje vykonanie služby
- **Poskytovateľ služby** spracuje požiadavku Webovej služby
- **Register služieb** je voliteľným prostredníkom, ktorý umožňuje publikovanie publikovanie popisu Webovej služby poskytovateľom. Klient môže použiť register k nájdeniu služby. Keďže je táto komponenta voliteľná architektúra umožňuje použitie vlastných spôsobov na spárovanie klienta a poskytovateľa služby.

4.1 Komunikačné protokoly SOA

SOA ako taká je len metodika alebo architektonický návrh. Nieje spojená so žiadnou konkrétnou technológiou alebo protokolom. SOA je vlastne vytváranie rozhraní na vyššej úrovni nad doménovým modelom. V určitom

bode procesu vývoja systému je predsa len potrebné rozhodnúť o konkrétnych štandardoch, implementáciách a postupoch, ktoré budú tento návrh realizovať.

V tejto kapitole porovnáme tri prístupy v komunikácii medzi službami. Prvý prístup je založený na výmene správy v jednoduchom XML formáte POX (Plain Old XML), ktorý je špecifický pre daný systém. Druhý protokol je použitie princípu *RESTfull webovej služby*. Tretím je použitie SOAP ako komunikačného protokolu.

4.1.1 POX

Použitie vlastného jednoduchého formátu XML na výmenu informácií medzi službami nad transportným protokolom HTTP, označované ako POX-over-HTTP [13].

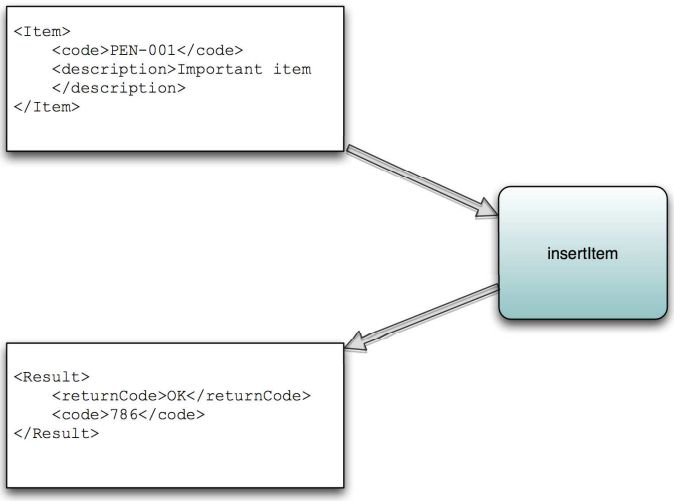


Diagram 4.2: Príklad komunikácie pomocou POX

Diagram 4.2 ukazuje príklad interakcie so službou *insertItem* pomocou vlastného formátu XML pre zápis objektov a aj návratových stavov. Z pohľadu implementácie je možné spracovanie takejto správy na tri kroky:

- Deserializovanie XML dotazu
- Vykonanie aplikačnej logiky

- Pripravenie a serializovanie odpovede

Implementácia klienta a služby pre takúto architektúru webovej služby je veľmi jednoduché a okrem knižnice na spracovanie XML dokumentu nevyžaduje žiadne dodatočné knižnice. Klient a poskytovateľ služby môžu byť implementovaní na ľubovoľnej platforme, ktorá umožňuje komunikáciu cez HTTP. Vzhľadom na jednoduchosť protokolu je jeho réžia nízka. Jednoduchosť tohoto postupu sa môže prejaviť zároveň aj ako jeho slabá stránka, pretože dodatočné požiadavky ako je napríklad podpora transakcií alebo autentifikácia by musela byť riešená na vyššej, aplikačnej úrovni.

4.1.2 RESTfull webová služba

REST (Representational State Transfer) je štýl softvérovej architektúry navrhnutý pre distribuované systémy. Jeho orientácia je hlavne dátova a nie procedurálna ako je tomu pri XML-RPC alebo SOAP. REST definuje rozhranie pre prístup k zdrojom. Zdrojom môžu byť aplikácie alebo dáta.

Označením **RESTfull** webová služba sa rozumie jenoduchá webová služba postavená na protokole HTTP a splňujúca princípy REST. *Popis* webovej služby sa skladá z:

- URI identifikúcu službu/zdroj
- Reprecentácie dát pre komunikáciu (JSON, POX, YAML...)
- Sady operácií zo sady HTTP metód (POST, GET, PUT, DELETE)

URI identifikuje nielen službu aku takú, ale aj každý jendotlivý prvok. Ak by napríklad URI `http://example.org/items` zodpovedalo sade objektov v katalógu, tak URI `http://example.org/items/365` zodpovedá konkrétnemu objektu v tejto sade. Jednotlivé metódy majú následujúci efekt:

- **GET** vráti ako odpoveď zoznam prvkov sady, alebo konkrétny prvok.
- **PUT** modifikuje existujúci prvok prípadne vytvor nový identifikovaný danou URI.
- **POST** vytvorí nový prvok v sade.
- **DELETE** Zmaže celý zdroj identifikovaný danou URI.

Požiadavky na implementáciu takejto webovej služby sú podobné ako v predchádzajúcej stati. Oba systémy sa odlišujú hlavne filozofiou ktorou pristupujú k dátam. RESTfull webová služba sa viac zameriava na dáta a naopak POX sa viac orientuje na procedurálne spracovanie dát. Ani jeden systém nieje založený na štandarde, čo má za následok menšiu mieru formalizmu a väčšiu intuitívnosť pri návrhu a vývoji takýchto aplikácií.

Táto voľnosť umožňuje agilnejší vývoj služieb a je vhodná najmä pre menšie projekty v uzatvorenom prostredí, kde je existuje centrálna autorita, ktorá môže určiť dodatočné pravidlá. Naopak, v prípade rozsiahleho federalizovaného systému s množstvom navzájom nezávislých účastníkov vyústi použitie takéhoto prístupu buď k nezdaru projektu, prechodu na existujúci formalizovaný komunikačný protokol akým je SOAP alebo vytvoreniu špecifikácie vlastného protokolu pre komunikáciu a výmenu dát.

4.1.3 SOAP webová služba

Pri návrhu webovej služby, ktorá má používať ako komunikačný protokol SOAP (SOAP webová služba) je potrebné urobiť množstvo architektonických rozhodní, ktoré ovplyvňujú celkovú povahu služby.

Už pri **návrhu služby** a jej špecifikácii pomocou *WSDL* je možné použiť dva rôzne prístupy **top-down** a **bottom-up**. Top-down prístup je postup vytvárania špecifikácie z existujúceho dokumentu popisujúceho doménový model. Pri tomto postupe najskôr vznikne WSDL dokument a následne sa z neho môžu pomocou dodatočných nástrojov generovať kostry tried pre implementáciu v konkrétnom jazyku a platforme. Naopak bottom-up prístup má ako základ už existujúcu komponentu konkrétneho programovacieho jazyka na nejakej platforme a z neho sa pomocou dodatočných nástrojov vytvorí WSDL popisujúcu túto službu. Z pohľadu náročnosti je top-down postup obtiažnejší, pretože vytvorenie WSDL dokumentu je netriviálna záležitosť aj s dostupnými nástrojmi. Príklad časti takej špecifikácie je v prílohe A. Vytvorenie WSDL pomocou bottom-up prístupu uľahčujú nástroje, ktoré dokážu zo zdrojového kódu vytvoriť WSDL špecifikáciu. Tento postup môže naraziť na problém, kde napríklad rozhranie používa nepodporované dátové typy v signatúre metód. Riešením tohoto problému môže byť aplikovanie návrhového vzoru fasáda na toto rozhranie.

Ďalším rozhodnutím pri návrhu služby je voľba **transportnej** vrstvy. SOAP umožňuje použitie viacerých transportných protokolov:

- **HTTP** je hlavný transportný protokol z hľadiska nasadenia v praxi,

podpory nástrojov a je zahrnutý aj v štandarde ako referenčná implementácia.

- **MOM** Message-Oriented Middleware akým je napríklad Apache ActiveMQ je nový druhov protokolov navrhnuté na mieru pre distribuované systémy a poskytujú podporu pre rôzne MEP.

- **SMTP** Simple Mail Transfer Protocol je zahrnutý v špecifikácii protokolu, ale nieje rozsiahle používaný. Použitie je obmedzené na jednosmernú výmenu správ, pretože implementácia modulu pre RPC (request-response) MEP je zložitá na tomto jednosmernom transportnom protokole.

Všeobecne sa odporúča použitie len jednej transportnej vrstvy v celom prostredí. Kombinácia protokolov je podporovaná štandardom, ale prakticky môže nastať množstvo problémov. Ako u bolo spomenuté, protokol HTTP je najpoužívanejší, ale aj použitie MOM môže ponúknuť viaceré výhody oproti HTTP. Základné charakteristiky MOM ako je napríklad perzistentné úložisko pre správy môžu byť použité k dosiahnutiu niektorých cieľov samotného návrhu distribuovaného systému.

4.1.4 Voľba formátu SOAP správy

Pri návrhu služby, ktorá bude používať SOAP je potrebné zvoliť správnu kombináciu spôsobu komunikácie a kódovania. Možné kombinácie sú:

- **RPC/Encoded**

- **RPC/Literal**

- **Document/Literal**

Variantu *Document/Encoding* v súčasnosti nepodporuje špecifikácia WSDL.

Pri použití varianty *RPC/Encoded* je celá Webová služba špecifikovaná pomocou jazyka WSDL vrátane argumentov a návratových hodnôt. Toto zabezpečuje pevnú väzbu implementácie a popisu služby.

Silné stránky tohoto prístupu:

- WSDL popis v procedurálnom formáte RPC

- Podpora množstva nástrojov

Pri použití varianty *RPC/Encoded* je celá Webová služba špecifikovaná pomocou jazyka WSDL vrátane argumentov a návratových hodnôt. Toto zabezpečuje pevnú väzbu implementácie a popisu služby.

Silné stránky tohoto prístupu:

- WSDL popis v procedurálnom formáte RPC
- Podpora množstva nástrojov

Nevýhody:

- Použitie SOAP kódovania (Encoded) má za následok, že správa SOAP obsahuje informácie o použitých typoch priamo v sebe. Toto výrazne zväčšuje správu a výraznejšie zaťažuje transportnú vrstvu. (xsi:type="xsd:int", xsi:type="xsd:string", xsi:type="xsd:double"...)
- Typová informácia je uložená priamo v správe a nie napríklad vo forme XML Schema, takže je validácia omnoho zložitejšia.

Príklad takejto správy je v prílohe C.

Varianta *RPC/Literal* je v podstate totožná s predchádzajúcou. Jediný rozdiel je v použití vlastného kódovania na serializovanie dát.

Silné stránky:

- WSDL popis je stále v procedurálnom formáte RPC
- Menšia veľkosť správ, pretože redundantné informácie o typoch dát sú presunuté do samostatného dokumentu XML Schema.
- Možnosť jednoduchej validácie formátu dát proti existujúcej XML Schéme.

Nevýhody:

- Silná väzba medzi medzi službou a klientom z princípu návrhu RPC.

Varianta **Document/Literal** nešpecifikuje žiadne podmienky pre štruktúru dokumentu. V tejto variante klient pošle dokument, ktorý musí poskytovateľ služby namapovať na objekty, volania operácií a ich argumenty.

Silné stránky:

- WSDL popis je stále v procedurálnom formáte RPC

- Menšia veľkosť správ, pretože redundantné informácie o typoch dát sú presunuté do samostatného dokumentu XML Schema.
- Možnosť jednoduchej validácie formátu celej správy proti existujúcej XML Schéme.
- Rozširovanie je možné menšími modifikáciami XML schema bez straty spätnej kompatibility.

Nevýhody:

- Slabá väzba komplikuje identifikáciu volaných operáciu a objektov.

Príklad takejto správy je v prílohe D.

Kapitola 5

Záver práce

Úspešne som analyzoval rôzne charakteristiky protokolu SOAP a vyčlenil som špecifické prípady, kedy je vhodné SOAP použiť. Ukázalo sa, že z hľadiska komplexnosti systémov sa SOAP viac hodí na rozsiahle systémy alebo prípady, kedy sú jednotlivé uzly distribuovaného systému vzdialené či už organizačne alebo geograficky. Pri menších systémoch je výhodnejšie použiť menej robustné komunikačné protokoly a architektúry.

V budúcnosti vidím veľkú možnosť uplatnenia SOAP v systémoch ESB (Enterprise service bus) a EAI (Enterprise application integration), kde môže fungovať ako hlavný komunikačný kanál.

Literatúra

[1] Mlýnková I., Pokorný J., Richta K., Toman K., a Toman V. *Technologie XML*, Karolinum, 2006.

[2] Zimmermann O., Tomlinson M., Peuser S.: *Perspective on Web Services*, Springer, 2005.

[3] Hansen M.: *SOA Using Java Web Services*, Prentice Hall, 2007.

[4] Binildas CA, Malhar Barai, Vincenzo Caselli: *Service Oriented Architecture with Java*, Packt Publishing, 2008.

[5] Maar ten van Steen: Distributed Systems - Principles and Paradigms
`http://www.cs.vu.nl/~steen/courses/ds-slides/notes.01.pdf`

[6] *Extensible Markup Language (XML) 1.0 (Fifth Edition)*
`http://www.w3.org/TR/2008/REC-xml-20081126/`

[7] *SOAP Version 1.2 Part 0: Primer (Second Edition)*
`http://www.w3.org/TR/2007/REC-soap12-part0-20070427/`

[8] *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)*
`http://www.w3.org/TR/2007/REC-soap12-part1-20070427/`

[9] *SOAP Version 1.2 Part 2: Adjuncts (Second Edition)*
`http://www.w3.org/TR/2007/REC-soap12-part2-20070427/`

[10] *SOAP Version 1.2 Email Binding*
`http://www.w3.org/TR/2002/NOTE-soap12-email-20020703`

[11] R. Fielding, J. Gettys, J. C. Mogul, H. Frystyk Nielsen, P. Leach, L. Masinter and T. Berners-Lee, Editors. IETF: *Hypertext Transfer Protocol – HTTP/1.1*, June 1999.

`http://www.ietf.org/rfc/rfc2616.txt`

[12] *Web Services Description Language (WSDL) 1.1*

`http://www.w3.org/TR/2001/NOTE-wsdl-20010315`

[13] Wikipedia:*Plain Old XML — Wikipedia The Free Encyclopedia*, 2009

`http://en.wikipedia.org/w/index.php?title=Plain_Old_XML&oldid=268137155`

Prílohy

Príloha A

Príklad wsdl

```
<?xml version="1.0"?>
<wsdl:definitions name="EndorsementSearch"
   targetNamespace="http://namespaces.snowboard-info.com"
   xmlns:es="http://www.snowboard-info.com/EndorsementSearch.wsdl"
...
   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
   <wsdl:types>
     <xsd:schema targetNamespace="http://namespaces.snowboard-info.com"
        xmlns:xsd="http://www.w3.org/1999/XMLSchema">
       <xsd:element name="GetEndorsingBoarder">
         <xsd:complexType>
           <xsd:sequence>
             <xsd:element name="manufacturer" type="string"/>
             <xsd:element name="model" type="string"/>
           </xsd:sequence>
         </xsd:complexType>
       </xsd:element>
       <xsd:element name="GetEndorsingBoarderResponse">
         <xsd:complexType>
           <xsd:all>
             <xsd:element name="endorsingBoarder" type="string"/>
           </xsd:all>
         </xsd:complexType>
       </xsd:element>
...
     </xsd:schema>
   </wsdl:types>
   <wsdl:message name="GetEndorsingBoarderRequest">
     <wsdl:part name="body" element="esxsd:GetEndorsingBoarder"/>
   </wsdl:message>
   <wsdl:message name="GetEndorsingBoarderResponse">
     <wsdl:part name="body" element="esxsd:GetEndorsingBoarderResponse"/>
   </wsdl:message>
   <wsdl:portType name="GetEndorsingBoarderPortType">
     <wsdl:operation name="GetEndorsingBoarder">
       <wsdl:input message="es:GetEndorsingBoarderRequest"/>
       <wsdl:output message="es:GetEndorsingBoarderResponse"/>
       <wsdl:fault message="es:GetEndorsingBoarderFault"/>
     </wsdl:operation>
   </wsdl:portType>
   <wsdl:binding name="EndorsementSearchSoapBinding"
                 type="es:GetEndorsingBoarderPortType">
...
```

Príloha B

Soap-envelope schema

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:tns="http://www.w3.org/2003/05/soap-envelope"
           targetNamespace="http://www.w3.org/2003/05/soap-envelope"
                  elementFormDefault="qualified" >
  <xs:import namespace="http://www.w3.org/XML/1998/namespace"
             schemaLocation="http://www.w3.org/2001/xml.xsd"/>
  <!-- Envelope, header and body -->
  <xs:element name="Envelope" type="tns:Envelope" />
  <xs:complexType name="Envelope" >
    <xs:sequence>
      <xs:element ref="tns:Header" minOccurs="0" />
      <xs:element ref="tns:Body" minOccurs="1" />
    </xs:sequence>
    <xs:anyAttribute namespace="##other" processContents="lax" />
  </xs:complexType>
  <xs:element name="Header" type="tns:Header" />
  <xs:complexType name="Header" >
    <xs:annotation>
          <xs:documentation>
          Elements replacing the wildcard MUST be namespace qualified,
          but can be in the targetNamespace
          </xs:documentation>
        </xs:annotation>
    <xs:sequence>
      <xs:any namespace="##any" processContents="lax" minOccurs="0"
                        maxOccurs="unbounded"  />
    </xs:sequence>
    <xs:anyAttribute namespace="##other" processContents="lax" />
  </xs:complexType>
  <xs:element name="Body" type="tns:Body" />
  <xs:complexType name="Body" >
    <xs:sequence>
      <xs:any namespace="##any" processContents="lax" minOccurs="0"
                  maxOccurs="unbounded" />
    </xs:sequence>
    <xs:anyAttribute namespace="##other" processContents="lax" />
  </xs:complexType>
  <!-- Global Attributes.  The following attributes are intended to be
  usable via qualified attribute names on any complex type referencing
  them.  -->
  <xs:attribute name="mustUnderstand" type="xs:boolean" default="0" />
  <xs:attribute name="relay" type="xs:boolean" default="0" />
  <xs:attribute name="role" type="xs:anyURI" />
  <!-- 'encodingStyle' indicates any canonicalization conventions
  followed in the contents of the containing element.  For example, the
  value 'http://www.w3.org/2003/05/soap-encoding' indicates the pattern
  described in the SOAP Version 1.2 Part 2: Adjuncts Recommendation -->
  <xs:attribute name="encodingStyle" type="xs:anyURI" />
...
```

Príloha C

Soap RPC/Encoded komunikácia

```
==========Request Sample==========
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:tns="http://eherenow.com/RPC"
 xmlns:types="http://eherenow.com/RPC/encodedTypes"
 xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <q1:GetUserName xmlns:q1="http://www.eherenow.com">
      <Where xsi:type="xsd:string">string</Where>
    </q1:GetUserName>
  </soap:Body>
</soap:Envelope>

==========Response Sample==========
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tns="http://eherenow.com/RPC"
xmlns:types="http://eherenow.com/RPC/encodedTypes"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <q2:GetUserNameResponse xmlns:q2="http://www.eherenow.com">
      <GetUserNameResult href="#id1" />
    </q2:GetUserNameResponse>
    <types:UserName id="id1" xsi:type="types:UserName">
      <Name xsi:type="xsd:string">string</Name>
      <Domain xsi:type="xsd:string">string</Domain>
    </types:UserName>
  </soap:Body>
</soap:Envelope>
```

Príloha D

Soap Document/Literal komunikácia

```
==========Request Sample ==========
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
      <GetUserName xmlns="http://www.eherenow.com">
         <Where>string</Where>
      </GetUserName>
   </soap:Body>
</soap:Envelope>
========== Response Sample ==========
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
      <GetUserNameResponse xmlns="http://www.eherenow.com">
         <GetUserNameResult>
            <Name>string</Name>
            <Domain>string</Domain>
         </GetUserNameResult>
      </GetUserNameResponse>
   </soap:Body>
</soap:Envelope>
```