Web Services

Martin KUBA

Ústav výpočetní techniky, Masarykova univerzita
Botanická 68a, 602 00 Brno
makub@ics.muni.cz

Abstrakt. Webové služby slouží k interakci mezi aplikacemi na počítačové síti internetových rozměrů. Jsou doplňkem k obvyklé interakci mezi živým člověkem a zdrojem statických webových dokumentů či interaktivní webovou aplikací. Webové služby jsou založeny na osvědčených webových standardech, zejména značkovacím jazyku XML. Díky tomu mohou propojovat velmi různorodé systémy, nezávisle na použitých programovacích jazycích, procesorech či operačních systémech. Tento příspěvek vysvětluje základní stavební kameny webových služeb, zejména komunikační protokol SOAP a jazyk pro popis služeb WSDL, a dále se věnuje některým pokročilejším tématům.

Klíčová slova: webové služby, XML, SOAP, WSDL, REST, ESB

1 Úvod

World Wide Web, tak jak jej zná většina jeho uživatelů, je tvořen zdroji (HTML stránkami, obrázky, animacemi atd.), jenž jsou určeny pro zobrazení živým lidem. Programy dokážou tento obsah webu zobrazit člověku, ale nedokážou v něm nalézat informaci, se kterou by mohly dále samy pracovat. Technologie, vyvinuté pro tento „lidský“ web, však mohou být použity i pro zpřístupnění zdrojů, které jsou strojově zpracovatelné. Zásadním prvkem této strojové zpracovatelnosti je značkovací jazyk XML, jenž umožňuje vytvářet dokumenty s jasně danou strukturou, zpracovatelné na všech platformách, tedy ve všech programovacích jazycích, na všech procesorech či operačních systémech. Strojově zpracovatelné webové zdroje, které dokážou XML nejen produkovat, ale i přijímat, se nazývají webové služby (anglicky web services) a jsou tématem tohoto tutoriálu.

Webové služby jsou tedy webem pro stroje, umožňují interakci mezi programy běžícími kdekoliv v Internetu, aniž by musel do jejich konání zasahovat člověk. Mohou umožňovat třeba spolupráci mezi informačními systémy dvou firem, které si mohou vyměňovat informace, i když jsou naprogramovány v různých jazycích, běží na různých systémech a jsou umístěny každý na jiném kontinentu.

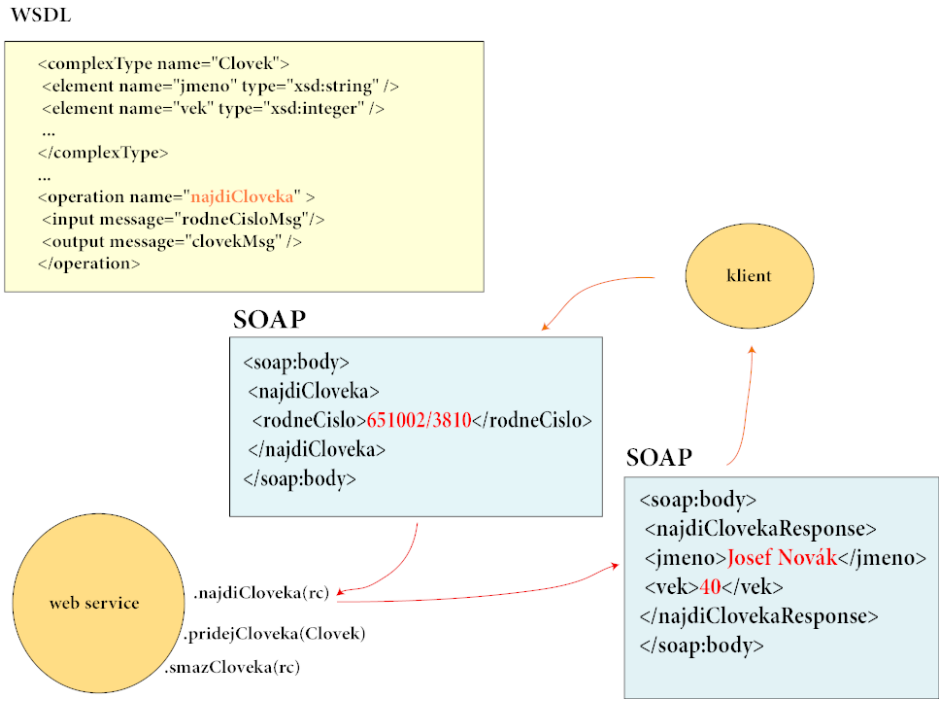
Cílem tohoto textu je pomoci čtenáři se zorientovat v problematice webových služeb. Struktura tutoriálu je následující. Nejdříve je podána oficiální definice webových služeb. Následující tři kapitoly připravují půdu pro popis principu webových služeb, obsahují po řadě: výčet možností komunikace na webu, obecné principy komunikace v distribuovaných systémech a vlastnosti jazyka XML využité

Peter Vojtáš (ed.), DATAKON 2006, Brno, 14.-17. 10. 2006, pp. 93-112.

ve webových službách. Následující dvě kapitoly popisují základní kameny webových služeb, komunikační protokol SOAP a jazyk pro popis služeb WSDL. Po té zmíníme open source nástroje pro práci s webovými službami. Další kapitoly jsou věnovány pokročilým tématům: možnostem nalézání služeb pomocí UDDI a WSIL, džungli dodatečných specifikací WS-*, alternativám k použití SOAP, vztahu gridu a webových služeb, bezpečnosti webových služeb. Poslední kapitola se vrací k problému, co jsou vlastně webové služby, a zmiňuje poslední trend v integraci podnikových aplikací, podnikovou sběrnici služeb ESB.

2 Definice webové služby

Podle dokumentu W3C, sepsaného Pracovní skupinou pro Architekturu webových služeb [15], zní definice webové služby takto:
Webová služba je softwarový systém zkonstruovaný k podpoře interakce mezi stroji přes síť. Má rozhraní popsané ve strojově zpracovatelném formátu (specificky WSDL). Ostatní systémy interagují s webovou službou způsobem předepsaným jejím popisem za pomoci SOAP zpráv, typicky dopravovaných použitím HTTP s XML zápisem v součinnosti s ostatními webovými standardy.



Obrázek 1: Operace poskytované webovou službou jsou popsány jazykem WSDL, komunikace probíhá protokolem SOAP.

Tato definice je kompromisní, ve skutečnosti představy o webových službách sahají od podnikových messagingových systémů zasílajících XML zprávy mezi službami (tedy nic společného s webem), až po služby přístupné přes web server pomocí zasílání čistého XML (tedy žádné SOAP).

Výhody webových služeb oproti jiným systémům vyplývají především z použití XML – nevyskytují se problémy se zápisem českých znaků či znaků z různých jazyků, díky důslednému používání znakové sady UNICODE; interakce je nezávislá na programovacím jazyku, objektové orientovanosti či platformě; vstupní bariéra je nízká; jsou vhodné pro volně vázané systémy, kdy klient a server o sobě musí předpokládat jen naprosté minimum. Nevýhodou je prostorová neúspornost a pomalost syntaktické analýzy XML.

Výše uvedená definice zahrnuje základní myšlenku strojové interakce (tj. bez účasti živého člověka) za pomoci osvědčených webových standardů, ale odvolává se na několik blíže nevysvětlených zkratek (WSDL, SOAP, HTTP, XML), proto si pro lepší porozumění nejdříve vysvětleme, jakými způsoby si aplikace na webu mohou předávat data.

3 Předávání dat mezi aplikacemi na webu

Nejprve se podívejme, jakými způsoby jsou na webu předávána data mezi zúčastněnými stranami.

Web je tvořen dokumenty adresovanými pomocí URL (Uniform Resource Locator), jenž vyjadřují, kde a jakým způsobem lze dokument získat. Dokument nemusí existovat jako statický soubor někde na disku, může být vytvořen dynamicky až v okamžiku, kdy je o něj požádáno. Aplikaci generující dokumenty na straně serveru lze chápat jako službu, která přijímá data ve formě požadavku na dokument, a poskytuje odpověď ve formě dokumentu. Způsobů, jak lze takové aplikaci data předat, je více.

3.1 Předávání dat přes URL

URL má tvar `protokol://uživatel@stroj:port/cesta?dotaz#fragment`.

Data podstatná pro generování dokumentu mohou být obsažena v částech `cesta` nebo `dotaz`. Část `fragment` slouží pro adresování částí dokumentů, a je interpretována na straně prohlížeče, pro předávání dat na server se proto nehodí.

Část `cesta` obecně vyjadřuje cestu nějakou hierarchií, s jednotlivými částmi oddělenými znakem lomítko. Např. cestu adresáři k souboru v souborovém systému, ale může vyjadřovat i cestu k aplikaci na serveru a její vstupní data.

Část `dotaz` může obecně vypadat jakkoliv. Obvyklý tvar pro HTTP protokol jsou dvojice řetězců `parametr=hodnota`, oddělené znaky &, protože tento tvar stanovuje norma jazyka HTML pro předávání hodnot z formulářů.

Zde je na místě vysvětlit lehce kuriózní situaci kolem oddělovacích znaků. Norma jazyka HTML [15] stanovuje v jedné části (konkrétně 17.13) obsahový typ „`application/x-www-form-urlencoded`“, který odděluje parametry znakem `&`, jako formu zápisu pro předávání hodnot z formulářů. Ale v jiné části (v dodatku B.2.2) konstatuje, že znak & je bohužel v SGML (i XML), na kterém je HTML

založeno, speciálním znakem, který musí být nepohodlně zapisován pomocí znakové entity `&` a proto doporučuje používat pro oddělování parametrů v ručně psaných URL znak středník. To však není příliš reálné, aplikace na straně webového serveru nemůže rozlišit, zda parametry vznikly z formuláře, nebo byly napsány ručně, a nemohla by se jednoznačně rozhodnout, který oddělovací znak použít. Proto nástroje pro tvorbu aplikací na straně webového serveru vždy předpokládají znak &.

Ještě se zastavme u zápisu znaků s českou diakritikou. Norma pro zápis URL stanovuje, že lze používat pouze ASCII znaky, ostatní znaky musí být zapsané pomocí znaku procenta a šestnáctkového čísla znaku tj. %XX. Neexistuje však cesta, jak v URL vyjádřit, které kódování znaků do bajtů je použito. Historicky se vžila zásada, že prohlížeče posílají znaky ve stejném kódování, v jakém byla webová stránka obsahující URL, ale starší normy pro zápis URL o kódování mlčí, a novější prosazují kódování UTF-8. V praxi proto nelze české znaky s diakritikou přes URL spolehlivě předávat. Stejná situace je pochopitelně u všech ne-ASCII znaků ve všech jazycích.

Dalším omezením pro data předávaná přes URL je maximální prakticky použitelná délka URL, která byla empiricky stanovena na 4kB. Dále pro předávání dat strukturovaných složitěji než páry řetězců je nutné případ od případu vymyslet vhodný způsob zápisu. Tato omezení vedla ke vzniku další metody předávání dat.

3.2 HTTP metoda POST

Další možností jak předat data aplikaci dynamicky generující webové dokumenty je použít metodu POST protokolu HTTP (Hyper Text Transfer Protocol)[15]. Na tomto místě zmiňme, jak vlastně vypadá požadavek na webový dokument pomocí protokolu HTTP.

Každý HTTP požadavek obsahuje následující části: použitou metodu (GET, POST, PUT, ...), URL požadovaného dokumentu, verzi protokolu HTTP, sadu hlaviček obdobných hlavičkám v e-mailech, prázdný řádek a u některých metod tělo požadavku. Pokud je přítomno tělo požadavku, hlavička `Content-type` určuje typ obsahu těla pomocí standardizovaného seznamu tzv. MIME typů zavedených pro určování obsahu e-mailů, tj. zda se jedná o text, obrázek, video, soubor pro určitou aplikaci nebo nějaký jiný obsah. HTTP odpověď vypadá stejně s tím rozdílem, že místo metody a URL je uveden kód výsledku požadavku (např. „200 OK“ či „301 Moved Permanently”).

Metoda GET neobsahuje tělo, takže data mohou být předána jen v URL nebo hlavičkách. Kvůli omezením URL uvedeným výše omezuje metoda GET předávaná data. Proto byla zavedena metoda POST, která předává data i v těle požadavku.

Typ obsahu těla požadavku při metodě POST může být libovolný, ale aplikace na serveru musí být schopna ho zpracovat. Klasickým typem obsahu je již zmíněný „`application/x-www-form-urlencoded`“, používaný pro předávání obsahu větších formulářů z HTML stránek, který však umožňuje předávat jen data strukturovaná jako páry řetězců. Dalším používaným typem je „`multipart/form-data`“, který umí navíc přibalit binární soubory.

3.3 XML přenášené HTTP

Výše zmíněné typy obsahu HTTP požadavku však stále kladou na předávaná data dvě z omezení známá už z URL, a to malou strukturovanost dat a nemožnost specifikovat kódování ne-ASCII znaků. Pro překonání těchto omezení je třeba použít nějaký jiný typ obsahu. Po vynálezu XML (eXtensible Markup Language) se tímto typem staly XML dokumenty, ve kterých lze zachytit v podstatě libovolně složitě strukturovaná data a které nemají problémy se zápisem znaků z kteréhokoliv jazyka na světě díky používání znakové sady UNICODE.

Nápad zasílat XML protokolem HTTP dostalo samozřejmě víc lidí a firem najednou, a tak souběžně vzniklo více definic, jak má předávané XML vypadat, například XML-RPC, WDDX (Web Distributed Data Exchange), XMI (XML Metadata Exchange) a další. XML společně s dalšími definicemi na něm postavenými, jmenovitě XML Schema, dokáže poměrně přesně zachytit i typovost dat (tj. zda se jedná o řetězce, čísla, časové údaje atd.) a různá omezení kladená aplikacemi na očekávanou strukturu dat.

Logickým požadavkem na přenášená XML data byla možnost vzdáleného volání procedur, a z původního XML-RPC se vyvinul SOAP (Simple Object Access Protocol), za kterým stály firmy Microsoft a IBM. Od okamžiku vzniku SOAP se začíná mluvit o webových službách.

Než se však dostaneme k podrobnému popisu SOAP, bude užitečné se zmínit obecně o komunikaci v distribuovaných systémech, a o některých méně známých zákoutích specifikací kolem XML. Čtenář těchto věcí znalý může následující dvě kapitoly přeskočit

4 Komunikace v distribuovaných systémech

Komunikace mezi částmi distribuovaných systémů, tedy systémů, jejichž části běží na různých počítačích spojených komunikační sítí, se obecně děje zasíláním zpráv (tato část značně čerpá z [15]). Programovat přímo zasílání zpráv je ale komplikované a nepohodlné, proto se ujaly abstrakce označované jako RPC (Remote Procedure Call) – vzdálené volání procedur a RMI (Remote Method Invocation) – vzdálené volání metod.

4.1 RPC – vzdálené volání procedur

Při vzdáleném volání procedur programátor distribuované aplikace používá z jeho pohledu synchronní volání funkce, která na pozadí pošle zprávu s požadavkem na zavolání procedury na vzdáleném systému a čeká, dokud nedostane odpověď. Programátorovi se tedy jeví komunikace stejně jako volání lokální procedury, a po dobu komunikace je volající proces pozastaven. Parametry volané vzdálené procedury a návratové hodnoty se předávají zásadně hodnotou, protože musí být zapsány do komunikačních zpráv. Rozhraní vzdálených procedur se popisují v nějakém jazyku pro definici rozhraní IDL (Interface Definition Language). Z této definice se pomocí nástrojů generuje zástupný kód (stub) v daném programovacím jazyku, který schovává před programátorem komunikaci a volá se jako běžný lokální kód.

Systémy pro RPC musí řešit záležitosti jako reprezentaci čísel při přenosu (známý little-endian a big-endian problém s pořadím bajtů na různých procesorech) a převod znakových sad (EBCDIC vs. ASCII).
Příklady systémů umožňujících RPC jsou DCE RPC nebo SUN RPC.

4.2 RMI – vzdálené volání metod

Objektově-orientované programování se osvědčilo v nedistribuovaných aplikacích. Objekt má stav, metody pro manipulaci stavu (rozhraní) a implementaci. Separace rozhraní a implementace umožňuje vzdálené volání metod, kdy zástupný kód (proxy) má stejné rozhraní jako vzdálený objekt, a provádí skrytě komunikaci se vzdáleným objektem. Vzdálené volání metod je víceméně stejné jako vzdálené volání procedur, potřebuje však předávání odkazů na vzdálené objekty. Příklady systémů umožňujících RMI jsou DCE Remote Objects, CORBA, Java RMI, DCOM.

4.3 Synchronní a asynchronní, transientní a persistentní komunikace

RPC i RMI používají synchronní komunikaci, kdy volající strana zastaví a čeká, dokud neobdrží odpověď. Tento způsob komunikace je jednoduchý pro programátora, avšak není vhodný ve všech systémech, například proto, že volaná strana neběží ve stejný čas jako volající.

Druhou možností je asynchronní komunikace, kdy volající strana pokračuje v práci jakmile pošle komunikační zprávu. Na příchod odpovědi pak musí být upozorněna, například vyvoláním přerušení. Takový způsob komunikace je náročnější na programování, ale v rozlehlých systémech internetové velikosti může být nezbytností.

Komunikaci můžeme také rozlišovat na transientní (pomíjivou) a persistentní (vytrvalou). Při transientní komunikaci je zpráva odeslána, ale může se po cestě ztratit. Naopak při persistentní komunikaci garantuje komunikační systém její doručení, byť by třeba zpráva musela čekat, až dočasně nefungující příjemce začne opět přijímat zprávy.

4.4 SOA – službově orientovaná architektura

RPC a RMI se osvědčily jen v uzavřených systémech podléhajících přísné správě. Při přechodu k systémům na Internetu, kdy jednotlivé části systému jsou vyvíjeny různými subjekty, se ukázala jejich omezení. Prvním omezením je synchronní komunikace, která neškáluje dobře na opravdu rozlehlé systémy, kde zpoždění sítě je významné a předem neodhadnutelné, a kde může dojít k selhání jen některých částí systému. Druhým omezením je těsná vazba mezi klientem a serverem, která brání nezávislé evoluci jednotlivých částí a znesnadňuje zavádění novějších verzí. Řešením se zdá být přechod na tzv. službově orientovanou architekturu, kdy služby mají přesně definované rozhraní, to je ale popsáno pomocí zpráv, které služba může přijímat a odesílat, nikoliv pomocí operací na datových typech.

SOA bývá dávána do kontrastu se systémy založenými na distribuovaných objektech [15]. Je mezi nimi zásadní filozofický rozdíl, trefně vyjádřený v [15] – CD přehrávač poskytuje službu přehrávání CD, kdežto při objektově orientovaném

přístupu by každé CD bylo dodáno s vlastním přehrávačem, ze kterého by nešlo vyjmout. SOA by měla být vhodnější pro rozlehlé systémy než distribuované objekty, protože místo aby se zaměřovala na svázání dat a operací, zaměřuje se na funkci jenž má být vykonána, což lépe odpovídá způsobu, jakým jsou organizovány lidské aktivity [15].

5 Potřebná zákoutí XML

Než se dostaneme k popisu protokolu SOAP a jazyku WSDL, bude užitečné si uvést některé rysy jazyka XML, které jsou ve webových službách využity. Tato kapitola předpokládá, že čtenář má základní znalost XML, tedy ví, že dokumenty jsou tvořeny správně vnořovanými značkami, atributy značek a texty, ale nevyzná se v nadstavbových specifikacích XML Namespaces a XML Schema, jenž obě intenzivně využívají tzv. URI. Vezměme je v logickém pořádku.

5.1 URI aneb proč některá URL nemusí existovat

Většina uživatelů Internetu zná pojem URL (Uniform Resource Locator), který určuje nějaký zdroj jeho umístěním. Kromě URL však existuje ještě URN (Uniform Resource Name), které určuje zdroj jménem, bez uvedení jeho umístění. Mohou být použita např. pro soubory, které existují ve více kopiích a je všeobecně známo, jak některou kopii získat. URL a URN dohromady tvoří množinu tzv. URI (Uniform Resource Identifier) [15]. Jen pro úplnost zmiňme, že URI smí obsahovat pouze ASCII znaky, proto existuje jeho zobecněná forma umožňující používat všechny znaky ze sady UNICODE, a ta se jmenuje IRI (Internationalized Resource Identifiers) [15].

URI se často používají v různých specifikacích kolem XML jako celosvětově jedinečné identifikátory. Předpokládá se totiž, že dva lidé volící nové URI si zvolí různé řetězce, nejlépe proto, že si zvolí URL příslušející webovému serveru, který mají pod kontrolou. Na místě, kde je požadováno URI, lze použít URL, protože URL jsou zvláštním případem URI. V tom případě ale URL nemusí označovat existující dokument. Toto použití URL na místě URI je kontroverzní a pro začátečníky značně matoucí. Je proto dobrým zvykem vytvořit pro každé URL použité jako URI dokument vysvětlující jeho smysl, a umístit ho na adresu určenou oním URL.

5.2 XML Namespaces

XML je metajazyk, umožňující definovat nové značkovací jazyky. Někdy se stává, že je třeba v jednom XML dokumentu smíchat značky z více jazyků. Například v dokumentech popisujících transformaci XML dokumentů jsou smíchány značky jazyka pro popis transformací i značky z cílového jazyka, do kterého se transformuje. Pokud by ve více jazycích byla definována stejná značka, došlo by k nejednoznačnosti.

Proto existuje specifikace XML Namespaces [15], která umožňuje značky z různých jazyků rozlišovat. Značky každého jazyka jsou ve vlastním jmenném prostoru, který je identifikován pomocí jistého URI. Například značky jazyka XHTML jsou ve jmenném prostoru identifikovaném URI

„http://www.w3.org/1999/xhtml“. Značky v dokumentu jsou s jmenným prostorem svázány pomocí prefixu, zapisovaného před jméno značky a odděleného dvojtečkou. Prefixu musí být přiřazeno příslušné URI před jeho použitím pomocí atributu tvaru `xmlns:prefix="URI"`. Samotný prefix není podstatný, důležité je jen URI jemu přidělené, tedy dva různé prefixy vázané se stejným URI označují stejný jmenný prostor. Jméno značky společně s URI jmenného prostoru tvoří tzv. kvalifikované jméno.

Pro ušetření znaků nemusí jeden z jmenných prostorů použitých v nějaké části dokumentu mít definovaný prefix, a všechny značky bez prefixu jsou pak automaticky v tomto prostoru. Tento tzv. implicitní jmenný prostor se definuje pomocí atributu `xmlns="URI"`, a stejně jako u definic prefixů je jeho rozsah platnosti jen v části dokumentu uvnitř značky nesoucí tento atribut, tedy v různých částech jednoho dokumentu můžou být různé implicitní jmenné prostory.

5.3 XML Schema

Jak již bylo zmíněno, XML je metajazyk, a lze s jeho pomocí vytvářet nové značkovací jazyky kladoucí určitá omezení na dokumenty v daném jazyku. Původně byl pro zápis struktury nově definovaných jazyků navržen jazyk DTD (Document Type Definition), ten však měl jinou syntaxi než XML a neumožňoval zadat typová omezení, např. že obsahem nějaké značky smí být pouze celé číslo. Proto byl standardizován nový definiční jazyk XML Schema [15], umožňující popsat strukturu definovaného jazyka a typová omezení. Poskytuje typový systém, který je vhodný pro definice struktury dokumentů, ale který lze bohužel jen velmi obtížně přenést do objektově orientovaných programovacích jazyků. Poskytuje sadu tzv. atomických typů (řetězce, čísla, časové údaje, atd.), seznamy (pole), variantní typy, složené typy a tzv. typy vzniklé omezením, např. číselný interval nebo řetězec odpovídající nějakému regulárnímu výrazu. Zejména vyjádření typů vzniklých omezením v OO programovacích jazycích je problém, který stále není uspokojivě vyřešen. Proto současné nástroje provádějící převod mezi dokumenty odpovídajícími určitému XML schematu a objekty v paměti nikdy neplní svoji úlohu zcela správně.

Výhodou nových jazyků popsaných pomocí XML Schema je, že kontrolu správnosti načítaných dat může místo aplikace provádět už knihovna pro rozbor XML (parser), čímž se tvorba aplikací zjednodušuje.

6 Protokol SOAP

Vyzbrojeni znalostí URI, XML Namespaces a XML Schema se můžeme směle pustit do rozboru webových služeb. Webové služby dle definice komunikují protokolem SOAP, který byl původně navržen firmami Microsoft a IBM jako protokol pro vzdálené volání procedur založený na XML a HTTP. Ty mu měly zajistit lepší nezávislost na platformě a průchodnost skrze firewally než měly ostatní RPC protokoly. Původně zkratka SOAP znamenala „Simple Object Access Protocol“. Tento název byl poměrně nešťastný, protože SOAP není jednoduchý, není určen pro přístup k objektům, ani neumožňuje předávat odkazy na objekty, a lze jej použít i pro jiné účely než RPC. Proto dnes už SOAP oficiálně není zkratka.

Organizace W3C (World Wide Web Consortium) přijala specifikaci SOAP verze 1.1 od IBM a Microsoftu v roce 2000 jako „W3C Note“, tj. vzala ji na vědomí a zveřejnila ji na svých webových stránkách, ale bez další podpory. Zároveň ustanovila pracovní skupinu pro vývoj novější verze, která byla o tři roky později vydána pod názvem SOAP 1.2 jako „W3C Recommendation“, tj. doporučení W3C.

SOAP původně definoval svůj vlastní typový systém, protože jeho vývoj začal dříve, než vzniklo XML Schema. Tento typový systém se ukázal být zdrojem nekončících problémů s kompatibilitou. Pokoušel se totiž definovat, jakým způsobem se určité datové typy v programovacích jazycích zapíšou pomocí XML. Tato cesta se však ukázala být slepá. Už u tak jednoduchého datového typu, jakým je pole řetězců, se ukázalo, že každý programovací jazyk jej chápe po svém. Například v jazyce Java může pole řetězců obsahovat prázdný ukazatel, kdežto v .NET to možné není. Při pokusu přenést pole řetězců z Javy do .NET pak může dojít k neřešitelné chybě. Tato chyba je způsobena právě rozdílným chápáním stejně pojmenovaného datového typu v různých programovacích jazycích.

Řešením se ukázal být opačný filozofický přístup, kdy SOAP neslouží k výměně určitých datových typů, ale k výměně určitých XML dokumentů, protože XML je neutrální půda, na které se různé programovací jazyky mohou potkat. Tedy místo poskytnutí XML rozhraní k existujícímu kódu, jenž nutně vede k problémům s kompatibilitou, se naopak primárně určí jaké XML zprávy je možné zasílat, a implementace se tomu musí přizpůsobit. Z těchto důvodů je nutné přistupovat se skepsí ke všem automatizovaným nástrojům slibujícím vygenerovat webovou službu z existujícího kódu v nějakém programovacím jazyce. S největší pravděpodobností tyto nástroje zavlečou do XML některá specifika daného programovacího jazyka, nejčastěji ve formě skrytých předpokladů o struktuře předávaného XML.

6.1 Anatomie SOAP volání

Volání webové služby pomocí SOAP zprávy znamená přenesení speciálně formátované XML zprávy na server. Tento přenos je obvykle proveden protokolem HTTP pomocí metody POST, ale může být realizován i jinými protokoly, např. SMTP, FTP, JMS atd.

SOAP zpráva je tvořena XML značkou `Envelope`, jenž obsahuje dvě části, nepovinnou hlavičku ve značce `Header` a tělo, označené značkou `Body`, vše v příslušném jmenném prostoru. Tělo musí obsahovat XML, které služba dokáže zpracovat. Při vzdáleném volání procedur tělo obsahuje hlavní značku, pojmenovanou stejně jako procedura, která se má vyvolat. Vnořené značky pak představují parametry volání.

SOAP zpráva přinášející odpověď ze serveru podle konvence obsahuje v těle opět jednu hlavní značku, jejíž název je složen s názvu vyvolané procedury a řetězce `Response`. Vnořené značky pak představují návratové hodnoty, kterých může být obecně více než jedna.

Příklad volání protokolem SOAP vypadá takto:

```
POST / HTTP/1.1
Content-Type: text/xml; charset=utf-8
Content-Length: 411
Connection: close
SOAPAction: ""

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:ns1="urn:mojeURI">
  <SOAP-ENV:Body>
      <ns1:jePrvocislo>
          <cislo>1987</cislo>
      </ns1:jePrvocislo>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Jde o volání operace s názvem `jePrvocilo`, ve jmenném prostoru `urn:mojeURI`, která má jeden vstupní parametr nazvaný `cislo`. Prvních pět řádků jsou HTTP hlavičky. Celá SOAP zpráva je zabalena ve značce `Envelope`, ve které jsou umístěny i definice jmenných prostorů. Samotný obsah volání je uvnitř značky `Body`.

SOAP umožňuje i chybové návratové zprávy, obsahující místo těla značku `Fault`, jenž jsou obdobou výjimek v programovacích jazycích. Chybová zpráva obsahuje tři části, značku `faultcode` určující kde došlo k chybě, značku `faultstring` obsahující chybové hlášení srozumitelné lidem a značku `detail`, která může obsahovat libovolně složitě strukturované XML. Obsah vstupních, výstupních i chybových zpráv lze popsat v jazyce WSDL.

7 WSDL – Web Service Description Language

Rozhraní webové služby je obvykle popsáno XML dokumentem v jazyce WSDL, určeném právě pro popisy rozhraní webových služeb. Jazyk WSDL byl vyvíjen zároveň s protokolem SOAP, jeho verze 1.1 byla přijata jako "W3C Note" v roce 2001, následně utvořená pracovní skupina však zatím stále neukončila práce na jeho další verzi, v srpnu 2006 byla verze 2.0 jen ve stavu "W3C Candidate Recommendation".

WSDL dokument popisuje rozhraní webové služby na syntaktické úrovni, stejně jako .h soubory v jazyce C nebo interface v jazyce Java, tedy jako seznam jmen funkcí/metod, zde zvaných operace, spolu s jmény a typy parametrů a návratových hodnot. WSDL popis je tedy obdobou IDL popisu při vzdáleném volání procedur, a proto existují automatizované nástroje, které z WSDL umí vygenerovat zástupný kód (stub) pro volání dané služby ve zvoleném programovacím jazyce.

WSDL popis kromě popisu rozhraní služby může obsahovat i její umístění, tedy specifikovat protokol a adresu, kde je dosažitelná.

WSDL dokument popisující webovou službu může obsahovat až pět částí. První část (XML značka `types`) definuje použité datové typy pomocí XML Schema. Druhá část (značka `messages`) pomocí těchto definovaných typů definuje komunikační zprávy, které služba akceptuje či vysílá, včetně chybových.

Třetí část (značka `portType`) obsahuje jednu nebo více značek `operation`, definujících pomocí komunikačních zpráv operace, obvykle jako dvojici vstupní a výstupní zprávy s případnými chybovými zprávami (faults), je však možné definovat i operace pouze s vstupní nebo pouze s výstupní zprávou. Každá operace má kromě jména určen i jmenný prostor, aby se zabránilo kolizi stejných jmen. Tato část WSDL popisu je tedy obdobou definice interface v jazyce Java, kde operace odpovídají metodám a deklarované chybové zprávy odpovídají deklarovaným výjimkám. Ve verzi WSDL 2.0 bude tato část přejmenována z `portType` právě na `interface`.

Čtvrtá část (značka `binding`) popisuje jakými komunikačními protokoly je možné službu volat. Ve verzi WSDL 1.1 je standardizován pouze HTTP protokol, ostatní protokoly lze specifikovat pouze po dohodě všech případných zájemců na způsobu jejich označení.

Pátá část (značka `service`) popisuje, kde se služba nalézá, obvykle pomocí URL, na kterém je dostupná protokolem HTTP.

7.1 Styly WSDL – document/literal a ostatní

Z historických důvodů je možné popsat rozhraní webové služby ve WSDL třemi různými způsoby. Je to způsobeno tím, že část `binding` specifikuje pro každou operaci dva atributy, `style` a `use`. Atribut `style` může mít hodnoty `rpc` nebo `document`, atribut `use` může mít hodnoty `literal` nebo `encoded`. To umožňuje čtyři kombinace, pouze tři však dávají smysl. Styl `rpc` je určen pro volání vzdálených procedur, kdy předávané parametry jsou zabaleny do značky určující název volané procedury. Atribut `use` pak určuje, zda se parametry zapíší pomocí typového systému SOAP (hodnota `encoded`) nebo pomocí typového systému XML Schema (hodnota `literal`). Styl `document` je určen pro předávání libovolných XML dokumentů, hodnotou atributu `use` tedy může být jen `literal`.

Podrobný rozbor problematiky stylů WSDL je dostupný v [15], uveďme zde jen výsledek. Z důvodů minimalizace problémů s kompatibilitou mezi různými implementacemi je dobré používat pouze kombinaci `document/literal`, a to i pro RPC volání, u nichž je volání procedury simulováno dokumentem s jednou hlavní značkou se jménem shodným s volanou procedurou. Tento pseudostyl se nazývá `document/literal wrapped`, a je dnes jediným stylem doporučovaným k používání.

7.2 WS-I

Původní specifikace SOAP 1.1 a WSDL 1.1 jsou v mnoha místech vágní, a ponechávají příliš mnoho prostoru pro rozdílné výklady při implementaci. Přidáním verze SOAP 1.2 se zmatek ještě zvýšil, což způsobilo velmi špatnou interoperabilitu mezi různými implementacemi SOAP. Cestou z tohoto zmatku ven bylo ustanovení

organizace WS-I (Web Services Interoperability), která vydává zpřesňující specifikace. Stěžejní specifikací této organizace je WS-I Basic Profile, který zpřesňuje vágní pasáže původních specifikací, a některé možnosti původními specifikacemi povolené zakazuje. Například zakazuje používání WSDL stylu `rpc/encoded`, protože způsoboval problémy, jak bylo popsáno výše v části o protokolu SOAP. Dále povoluje používání pouze SOAP verze 1.1.

Implementace, které se drží specifikace WS-I Basic Profile, mají slušnou šanci být vzájemně interoperabilní, i když plně garantovat to nelze.

7.3 WSDL vždy nejdřív

Jak již bylo zmíněno v části 6 o protokolu SOAP, různé platformy a programovací jazyky se mohou nejlépe potkat na úrovni čistého XML popsaného pomocí XML Schema. Datové typy nejsou přenositelné mezi různými programovacími jazyky, zatímco zpracování XML přenosné je. Proto je doporučovaným postupem začínat tvorbu nové webové služby vždy vytvořením jejího WSDL popisu, tedy "contract first". Tím je dáno rozhraní služby v platformově neutrální formě a je minimalizován prostor pro chyby a nejednoznačnosti.

Existuje dnes mnoho nástrojů ulehčujících tvorbu webových služeb. Některé z nich nabízejí možnost vzít již existující kód a z něj vygenerovat WSDL popis i implementaci serveru služby. Tento postup je však špatný, protože vlastně činí rozhraní služby závislé na její implementaci, tedy "contract last". Při změnách implementace či nástroje pro převod kódu na WSDL může dojít (a dochází) k dramatickým změnám rozhraní webové služby, což znemožňuje funkci klientů služby vytvořených vůči starším verzím. Dobrá rada nad zlato tedy zní – tvorbu webové začínejte vždy od WSDL.

7.4 Změny WSDL

U fungujících webových služeb je čas od času třeba změnit jejich rozhraní. V této situaci je důležité rozhodnout, zda změny znemožňují zpětnou kompatibilitu. Pokud ji uchovávají, což je třeba případ přidání nových operací, je možné službu změnit, protože existující klienti nepřestanou fungovat.

Pokud však zpětnou kompatibilitu zachovat nelze, doporučovaný postup je zachovat stávající službu funkční pro stávající klienty, a vytvořit službu novou v novém jmenném prostoru, a provozovat ji zároveň se starou službou. Obě verze tak mají stejně pojmenované operace, ale díky rozdílným jmenným prostorům nemůže dojít ke kolizi. Tak je možné provozovat vedle sebe více verzí téže služby a uchovat všechny klienty funkční.

8 Open source nástroje

Nástrojů pro tvorbu a práci s webovými službami založenými na SOAP a WSDL je dnes již velké množství, od komerčních po svobodné implementace. V akademické sféře si oblibu vysloužily dvě implementace, Apache Axis implementovaný na platformě Java, a gSOAP implementované v C/C++. Jejich přístup je značně odlišný.

gSOAP je optimalizované na velmi vysoký výkon, pro daný WSDL popis služby vygeneruje vysoce optimalizovaný zdrojový kód v C nebo C++ (je možno zvolit), používající pro zvýšení výkonu pokročilé techniky, např. zásobník syntaktických analyzátorů specializovaných na různé části očekávané zprávy. Naproti tomu Axis umožňuje sestavit SOAP volání i dynamicky za běhu aplikace, a zástupné třídy vygenerované z WSDL jen sestavují konkrétní SOAP volání za pomoci obecných nástrojů. Ve výsledku je proto gSOAP řádově rychlejší než Axis. Nová verze Axis 2 je zhruba dvakrát rychlejší než předchozí verze, ale stále znatelně pomalejší než gSOAP.

9 Vyhledávání služeb – UDDI a WSIL

Nyní již víme, že webové služby komunikují protokolem SOAP a jejich rozhraní je popsáno v jazyce WSDL. Ve WSDL popisu služby je uvedeno, kde je dosažitelná. Ale jakým způsobem nalezneme samotné WSDL, resp. jak zjistíme, že nějaké služba vůbec existuje ? Pro řešení tohoto problému byla navržena dvě řešení.

První řešení se jmenuje UDDI (Universal Description, Discovery and Integration). Je to webová služba, která udržuje jakýsi telefonní seznam webových služeb. Je možné v něm hledat buď podle klíčových slov, nebo podle oborů, například vyhledat webové služby firem zabývajících se hutnictvím. Na rozdíl od SOAP a WSDL, specifikace UDDI nikdy nebyla W3C Recommendation nebo Note. UDDI začalo v roce 2000 jako iniciativa několika firem publikovaná na serveru uddi.org, verze 2 byla definována v červnu 2001 a verze 3 v červenci 2002. Pak byla iniciativa přenesena na půdu organizace OASIS, která v únoru 2005 schválila verzi 3.0.2 jako svůj standard. Počáteční nadšení však během let vyprchalo. Všichni tři hlavní provozovatelé veřejných UDDI rejstříků – IBM, Microsoft a SAP – vypnuli své veřejně přístupné rejstříky v lednu 2006. Uváděný důvod byl, že většina záznamů ve veřejných rejstřících byla špatná až nesmyslná, protože záznamy mohl přidávat kdokoliv. Některé zdroje tvrdí, že UDDI se dále používá vnitropodnikově, toto tvrzení je však těžké ověřit.

Osobní zkušenost autora tohoto textu s UDDI napovídá, že UDDI byla navržena příliš obecně jako rejstřík všech obchodních služeb, například je možné v ní zaznamenat skutečnost, že místní firma rozvážející pizzu má určité telefonní a faxové číslo, tedy informace nijak nesouvisející s webovými službami. Ale účel, který by od UDDI očekávali provozovatelé webových služeb, totiž zaznamenávání a vyhledávání popisů webových služeb, plní nepříliš dobře a velmi komplikovaně.

V neúspěchu UDDI patrně hrál roli i fakt, že služby nalezené ve veřejném seznamu nemají nijak garantovánu svoji kvalitu. Opačný přístup než UDDI volí WSIL (Web Services Inspection Language), jazyk pro popis služeb poskytovaných nějakou firmou či institucí. WSIL dokument je vždy nazván `inspection.wsil` a je umístěn v kořenovém adresáři web serveru příslušné instituce. WSIL tedy počítá s tím, že zájemce si nejdřív vybere firmu poskytující služby, a z WSIL dokumentu pak zjistí, jaké služby poskytuje. WSIL byl definován firmami IBM a Microsoft v listopadu 2001, ale větší úspěch také nezaznamenal.

10 WS-* džungle

Kromě základních specifikací, tj. SOAP, WSDL a WS-I Basic Profile, byly vytvořeny spousty dalších dodatečných specifikací. Pro úsporu místa uveďme jen jejich názvy: WS-Addressing, WS-Agreement, WS-AtomicTransaction, WS-Attachments, WS-BusinessActivity, WS-Choreography, WS-Context, WS-Coordination, WS-CoordinationFramework, WS-Discovery, WS-Enumeration, WS-Eventing, WS-EventNotification, WS-Federation, WS-Management, WS-MessageDelivery, WS-MetadataExchange, WS-Notification, WS-BaseNotification , WS-BrokeredNotification , WS-Topics , WS-Policy, WS-PolicyAssertions, WS-PolicyAttachment, WS-Reliability, WS-ReliableMessaging, WS-ResourceFramework, WS-Resource , WS-ResourceLifetime , WS-ResourceProperties , WS-RenewableReferences , WS-ServiceGroup , WS-BaseFaults , WS-ResourceTransfer, WS-SecureConversation, WS-Security, WS-Transaction, WS-TransactionManagement, WS-Transfer, WS-TransferAddendum, WS-Trust
Tyto specifikace jsou většinou dílem různých firem či sdružení, a mnohé spolu vzájemně soupeří, protože byly vytvořeny vzájemnými konkurenty, např. firmami IBM a Microsoft. Množství těchto dodatečných specifikací je dosti odrazující, a Tim Bray, spolutvůrce XML, to vyjádřil lapidárně v příspěvku ve svém blogu, nazvaném "The loyal WS-Opposition"[15]: "Ať se pokouším sebeusilovněji, stále si myslím, že tato WS-* hromada je nafouklá, neprůhledná a šíleně komplikovaná. Myslím, že bude těžké ji pochopit, těžké ji implementovat, těžké zajistit interoperabilitu a těžké zabezpečit."

Na druhou stranu, webové služby v podnikovém nasazení nemusí používat k přenosu protokol HTTP, a mohou mít požadavky na transakčnost, spolehlivost nebo management. Například WS-Addressing přidává do hlavičky SOAP zpráv informace o tom, odkud a kam zpráva putuje, velmi podobné hlavičkám e-mailů From: a To:, což může být nezbytné, pokud zprávy putují nějakým messagingovým systémem. Specifikace WS-Policy a WS-SecurityPolicy mají za cíl poskytnout rámce pro popisy vlastností služeb, které nelze vyjádřit ve WSDL. WS-Transfer je obdoba HTTP protokolu nad SOAP, poskytující operace Get, Put, Create na dokumenty či jejich části, což může mít smysl v podnikových systémech používajících persistentní messagingové systémy místo protokolu HTTP. WS-Eventing, podporovaný firmami Microsoft a Intel, a konkurenční WS-Notification, podporovaný firmami IBM a HP, poskytují rámce pro zasílání upozornění na události, a podle oznámení z března 2006 budou sloučeny do nové specifikace nazvané WS-EventNotification.

Jistým varovným znakem je, že všechny tyto specifikace jsou vytvářeny komisemi, které se snaží předjímat, co by mohlo být užitečné, místo aby zpětně kodifikovaly v praxi osvědčené postupy. Laskavý čtenář nechť to srovná se specifikacemi nepochybně úspěšných a užitečných HTML a HTTP, které byly kodifikovány až několik let po té, co byly všeobecně používány.

11 Alternativy – REST a POX

Složitost a nepřehlednost WS-* specifikací měla za následek tlak v opačném směru, ke zrušení SOAP úplně. Často citovaným příkladem jsou webové služby firmy Amazon, které jsou poskytovány ve dvou verzích. Jedna verze jsou klasické SOAP/WSDL služby, druhá verze umožňuje jednoduché XML dotazy přes HTTP (zvané někdy POX – Plain Old XML). SOAP verze generuje 20% provozu a jednodušší XML verze 80% provozu [15].

Vzrušená debata nastalá po tomto zjištění nakonec dospěla k názoru, že pro jednoduché aplikace na webu, používající cyklus dotaz-odpověď, jsou SOAP a WS-* kanón na vrabce. Zpracování XML a možnost provádět HTTP dotazy dnes poskytují téměř všechna prostředí, kdežto používání SOAP vyžaduje mnohem specializovanější nástroje. Naopak SOAP a WS-* jsou vhodnější pro podniková prostředí, kde jsou zprávy zpracovávány mezilehlými prostředníky či jejich putování systémem je určováno dynamicky.

Často zmiňovaným pojmem v této souvislosti je REST (REpresentational State Transfer), pojem vytvořený Royem Fieldingem, jedním z autorů HTTP protokolu, v jeho disertační práci [15]. Základní myšlenkou tohoto stylu architektury je, že každý zdroj na webu je identifikován pomocí URI a má reprezentaci, např. XML dokument, obrázek, atd. S touto reprezentací může být nakládáno pomocí pevné sady operací, např. čtyř základních HTTP metod, PUT, GET, POST a DELETE na tomto URI. Nebo obrazněji, webové zdroje jsou podstatná jména, a HTTP metody jsou slovesa určující co se se zdroji má stát. REST velmi zdůrazňuje bezestavovost, s argumentem, že právě ona umožnila škálovat web až na současnou velikost.

REST si získal poměrně agresivní zastánce, kteří sami sebe nazývají RESTafariáni, a prosazují ho jako jediný správný přístup k tvorbě aplikací na webu. Střízlivější pohled napovídá, že REST skutečně má své výhody, zejména umožňuje velmi dobrou škálovatelnost aplikací, nicméně není všelékem. Ne každý zdroj na webu lze vyjádřit jako statickou reprezentaci. Některé zdroje jsou ze své podstaty orientovány na činnost, a tyto zdroje se lépe vyjadřují ve formě služeb než pasivních dokumentů.

Inspirace RESTem je nicméně užitečná. Představme si službu poskytující informace o počasí a teplotě vzduchu v každém městě. Klasický SOAP přístup by byl vytvořit službu s operací getTemperature, jejímž parametrem by bylo požadované město. REST přístup velí vytvořit webovou aplikaci, které se jméno města zadá jako součást URL, např. http://pocasi.cz/teploty/Brno, jenž vrací XML dokument s informací o teplotě. Přicházející specifikace WSDL 2.0 podporuje popis služeb volaných tímto způsobem, tj. s parametry předávanými v URL.

12 Grid a OGSA, životní cyklus a vnitřní stav u webových služeb

Grid je dle definice infrastruktura pro sdílení různorodých zdrojů nepodléhajících centralizované správě, a to infrastruktura všudypřítomná, spolehlivá, bezpečná, standardizovaná, flexibilní a koordinovaná. Původní myšlenka vznikla v prostředí superpočítačů, kde sdílení výkonu procesorů a úložného prostoru mezi institucemi umožňuje řešit větší problémy, než by šlo řešit bez sdílení. Myšlenka sdílení v gridu

pak byla rozšířena i na sdílení jiných zdrojů, např. dat, přístrojů nebo znalostí. Pokusy o praktickou implementaci gridu ale narážely od počátku na heterogenitu různých prostředí, protokolů a programovacích jazyků. Proto autoři myšlenky gridu navrhli v roce 2002 gridovou infrastrukturu orientovat službově a založit ji na webových službách. Tato nová architektura gridu dostala jméno OGSA (Open Grid Services Architecture) [15].

Možnosti webových služeb se jim ale zdály nedostatečné, zejména proto, že webové služby nemají definovaný životní cyklus, tj. nemohou na přání vznikat a zanikat, nemají vnitřní stav, který by bylo možné standardizovaným způsobem získat, a nelze jim zasílat upozornění na události. Navrhli proto realizaci OGSA, ve které bylo možno vytvářet nové instance webových služeb pomocí tzv. továrních (factory) služeb, a tyto instance měly vnitřní stav, vyjádřitelný jako XML soubor, s možností tento stav pozorovat. Tento návrh se setkal s velkým odporem ze strany komunity kolem webových služeb, protože vlastně vytvářel systém distribuovaných objektů nad webovými službami, tedy šel proti podstatě službově orientované architektury.

Proto druhý návrh realizace OGSA zavedl tzv. zdroje (WS-Resource) s vnitřním stavem a omezenou dobou životnosti, manipulovatelné pomocí operací na bezestavových webových službách. Zdroje jsou identifikovány pomocí identifikátorů přenášených v hlavičkách SOAP zpráv. Tato druhý návrh realizace OGSA byl vtělen do rodiny specifikací nazvané WS-ResourceFramework (WSRF), která získala velkou podporu ze strany firmy IBM. Možnost zasílání asynchronních upozornění na události pak řešila druhá rodina specifikací, nazvaná WS-Notification. Tyto dvě skupiny specifikací podstatně přispěly k růstu množství WS-* specifikací.

Firma Microsoft zatím vydala vlastní, konkurenční specifikace, konkrétně WS-Transfer a WS-Eventing. V březnu 2006 pak firmy IBM, HP, Intel a Microsoft společně oznámily plán na sloučení WS-ResourceFramework/WS-Notification s WS-Transfer/WS-Eventing tím způsobem, že vniknou nové specifikace vrstvené nad WS-Transfer/WS-Eventing, zahrnující některé myšlenky z WS-ResourceFramework/WS-Notification. Tyto nové specifikace budou mít názvy WS-ResourceTransfer/WS-EventNotification. Zda se tím situace kolem práce s životním cyklem, vnitřním stavem a zasíláním upozornění na události vyřešila, ukáže budoucnost.

13 Bezpečnost

Bezpečnost webových služeb lze řešit na dvou úrovních, na úrovni zpráv nebo na úrovni transportní vrstvy.

Snadnější, rychlejší a historicky starší je řešení na úrovni transportní vrstvy, kdy SOAP a HTTP komunikace probíhá nad SSL (Secure Socket Layer) šifrovaným a autentizovaným spojením. Toto řešení má jako výhody rychlost a značnou robustnost, protože SSL existuje již dlouhou dobu. Má však také nevýhody. SSL je ze své podstaty dvoubodové spojení, které chrání komunikaci před odposlechem a změnou po cestě, ale neumožňuje digitálně podepisovat zasílaná data, tedy zpětně dokázat, kdo co zaslal (non-repudiation). Jako dvoubodové spojení také neumožňuje použít složitější architektury než je klient-server, tedy zpracování mezilehlými prostředníky.

Druhým řešením bezpečnosti webových služeb je řešení na úrovni SOAP zpráv. SOAP zpráva je XML dokument, je proto možné použít W3C standardy XML Encryption a XML Signature pro šifrování resp. podepisování částí zpráv nebo celých zpráv. Standardizační organizace OASIS vydala specifikaci WS-Security, která stanovuje, jakým způsobem podepisování a šifrování na úrovni SOAP zpráv používat, a jak využít stávající bezpečnostní mechanismy (PKI, Kerberos, hesla atd.) v SOAP zprávách. Specifikace WS-Security má již dvě verze, 1.0 a 1.1, a organizace WS-I pracuje na specifikaci WS-I Security Basic Profile, která by měla pomoci v interoperabilitě mezi různými implementacemi.

Bezpečnost na úrovni zpráv má tedy opačné výhody a nevýhody než bezpečnost na úrovni transportní vrstvy. Umožňuje sice podepisování zpráv a složité scénáře s prostředníky při zpracování zpráv, není však ještě dost vyzrálá a i rychlost zpracování je zatím nižší než u SSL.

14 Co jsou tedy webové služby ?

Na začátku článku byla uvedena definice webové služby, tak jak k ní dospěla pracovní skupina W3C. I v rámci této pracovní skupiny bylo těžké se shodnout na jedné definici [15].

Na jedné straně stojí pohled ze strany aplikací na webu – webové služby by měly být služby poskytované aplikacím přes webová rozhraní, pro ty je nejdůležitější protokol HTTP a použití XML, naopak použití SOAP je nadbytečnou komplikací. Těmto aplikacím vyhovuje RESTový přístup, kdy všechny vše je adresovatelné přes URI, a lze integrovat XML data získaná z více zdrojů. Spolehlivost není příliš důležitá, protože uživatelé jsou zvyklí, že web není spolehlivý.

Na druhé straně stojí pohled podnikových aplikací – webové služby slouží díky XML především jako platformově neutrální rozhraní aplikací, použití protokolu HTTP je nevýhodné, naopak jsou časté komplikované scénáře se zasíláním SOAP zpráv pomocí persistentních messagingových systémů, kdy je navíc cesta zprávy systémem určována dynamicky a zpráva prochází přes řetěz zpracovatelů. Pro tyto aplikace je vhodný SOAP s jeho možnostmi vkládat do hlaviček SOAP zpráv mnoho dodatečných informací. V takovém prostředí nemají služby nic společného s webem, a padly již návrhy, aby z názvu webové služby bylo vypuštěno slovo webové, a mluvilo se raději o SOA [15].

Tyto dva přístupy mají společnou v podstatě jen strojovou interakci pomocí XML zpráv.

14.1 ESB – Enterprise Service Bus

Tématem, které se nedávno vynořilo v oblasti použití webových služeb pro integraci podnikových aplikací, je ESB, česky asi podniková sběrnice služeb. Její koncept představil Roy Shulte z firmy Gartner v prosinci 2002 v publikaci “Predicts 2003: Enterprise Service Buses Emerge“ [15]. Mělo by se jednat o řešení problému propojování (webových) služeb v rámci jedné instituce či firmy flexibilním způsobem. Základní myšlenka je, že pokud všechny podnikové aplikace budou přebudovány podle zásad službově orientované architektury tak, aby poskytovaly

rozhraní ve formě služby komunikující s okolím pomocí XML zpráv, je možné propojit všechny aplikace v podniku jednotnou infrastrukturou zajišťující zasílání XML zpráv. Tato infrastruktura může být konfigurovatelná, a změny v obchodních procesech se pak mohou provádět změnou nastavení komunikační infrastruktury, beze změn služeb samotných.

15 Závěr

Webové služby jsou webem pro stroje, a tak ač zůstávají lidským uživatelům webu skryty, umožňují snadnou komunikaci mezi programy vytvořenými nejrůznějšími způsoby a patřícími různým majitelům. Od původního návrhu RPC systému založeného na XML a HTTP prošly webové služby vývojem, který odhalil, které nápady vedly do slepé uličky (typový systém SOAP, veřejné UDDI seznamy služeb), a které se osvědčily („contract first“ WSDL, orientace na dokumenty).

Zájem o webové služby postupně přerostl v zájem o službově orientovanou architekturu, která má ambice uspět tam, kde předchozí architektury selhaly – při tvorbě skutečně rozlehlých distribuovaných systémů na Internetu, složených z částí vlastněných nezávislými majiteli. Poznatky získané při tvorbě těchto systémů se zpětně promítají i do tvorby vnitropodnikových systémů, kterým orientace na služby umožňuje mnohem flexibilnější změny, než by bylo možné dříve. Orientace na služby, které spolu komunikují zasíláním zpráv, ostatně více odpovídá tomu, jak jsou lidské aktivity organizovány v reálném světě, než tomu bylo u předchozích generací informačních systémů.

Tento tutoriál se pokusil osvětlit základní i pokročilejší témata z oblasti webových služeb. Tato oblast je stále v prudkém vývoji, a bude zajímavé pozorovat, kterými směry se v budoucnu vydá.

Literatura

1. W3C: *Web Services Architecture*. Working Group Note, February 2004 http://www.w3.org/TR/ws-arch/
2. *HTML 4.01 Specification*, W3C Recommendation, http://www.w3.org/TR/html4
3. RFC 2616 – *Hypertext Transfer Protocol – HTTP/1.1*
4. A. Tanenbaum, M. van Steen: *Distributed Systems Principles and Paradigms*. Prentice-Hall, Upper Saddle River, New Jersey, USA, 2002, ISBN 0-13-088893-1
5. RFC 3986 – *Uniform Resource Identifier (URI): Generic Syntax*
6. RFC 3987 – *Internationalized Resource Identifiers (IRIs)*
7. *Namespaces in XML 1.0* (Second Edition), W3C Recommendation, 16 August 2006, http://www.w3.org/TR/xml-names
8. *XML Schema Part 0: Primer* (Second Edition), W3C Recommendation, 28 October 2004, http://www.w3.org/TR/xmlschema-0/

9. Russell Butek: *Which style of WSDL should I use?* IBM developerWorks, 2003, http://www-128.ibm.com/developerworks/webservices/library/ws-whichwsdl/
10. Tim Bray, *The Loyal WS-Opposition*, blog entry, 2004, on-line http://www.tbray.org/ongoing/When/200x/2004/09/18/WS-Oppo
11. Werner Vogels: *Web Services Are Not Distributed Objects*, IEEE Internet Computing, volume 7 (2003), number 6, pages 59-66, http://csdl.computer.org/dl/mags/ic/2003/06/w6059.htm
12. Greg Goth: *Critics Say Web Services Need a REST*, IEEE Distributed Systems Online, volume 5 (2004), number 12, http://csdl2.computer.org/comp/mags/ds/2004/12/oz001.pdf
13. Roy Thomas Fielding: *Representational State Transfer (REST)*, chapter 5 in *Architectural Styles and the Design of Network-based Software Architectures*, PhD Dissertation, University of California at Irvine, 2000, http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm
14. Hao He: *What is Service-Oriented Architecture?*, XML.com 2003, on-line http://webservices.xml.com/pub/a/ws/2003/09/30/soa.html
15. *OASIS Reference Model for Service Oriented Architecture*, Committee Draft 1.0, 7 February 2006, http://www.oasis-open.org/committees/download.php/16587/wd-soa-rm-cd1ED.pdf
16. I. Foster, C. Kesselman, J. Nick, S. Tuecke: *The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration*, 2002, http://www.globus.org/alliance/publications/papers.php#OGSA
17. Martin Keen et al: *Patterns: Implementing an SOA using an Enterprise Service Bus*, IBM Redbook SG24-6346-00, 2004, ISBN 0738490008, http://www.redbooks.ibm.com/redpieces/pdfs/sg246346.pdf

Annotation:

Web Services

This paper is a tutorial of web services. It explains the basics, like the communication protocol SOAP, the language for description of web services' interface WSDL, together with fundamental low level things like Universal Resource Identifiers, XML Schema and XML Namespaces. Also more advanced topics are discussed, like service discovery facilitated by UDDI and WSIL, the ever growing family of additional WS-* specifications, state and lifecycle management introduced by grid services based on web services, web service security, and the latest trend in enterprise application integration, Enterprise Service Bus. Also alternatives to the heavyweight SOAP/WS-* approach are introduced, namely Plain Old XML over HTTP, and REST.