Masarykova univerzita
Fakulta informatiky

Webové služby v architektuře REST na platformě .NET

Bakalářská práce

Tomáš Pažourek

Brno, jaro 2012

Prohlášení

Prohlašuji, že tato bakalářská práce je mým původním autorským dílem, které jsem vypracoval samostatně. Všechny zdroje, prameny a literaturu, které jsem při vypracování používal nebo z nich čerpal, v práci řádně cituji s uvedením úplného odkazu na příslušný zdroj.

Brno, 18. května 2012
Tomáš Pažourek

Vedoucí práce: Mgr. Filip Jurnečka

Poděkování

Tímto bych rád poděkoval vedoucímu této bakalářské práce, Mgr. Filipovi Jurnečkovi, za odborné vedení, připomínky, praktické rady a celkově za veškerý čas, který mi věnoval. Děkuji také svým přátelům a rodině za podporu během studia a přípravy práce.

Shrnutí

Práce rozebírá architekturu REST z teoretického i praktického hlediska. Popisuje její základní pojmy a myšlenky ve vztahu z webovým službám. Představuje implementaci architektury na platformě .NET a zabývá se dalšími otázkami a problémy okolo vývoje a konzumace těchto služeb. Práce také obsahuje implementaci vzorové RESTové služby a přikládá materiály pro výuku daného tématu.

The thesis analyzes REST architecture from both theoretical and practical point of view. It describes the basic concepts and ideas in relation to web services. It also presents information on the implementation of REST architecture on .NET platform and discusses other issues and problems of development and consumption of these services. The work also includes a sample implementation of RESTful service and study materials on the topic.

Klíčová slova

webové služby, web service, REST, Representational State Transfer, hypermédia, .NET Framework, WS, SOAP, Windows Communication Foundation, WCF

Obsah

Úvod

Rozsáhlá síť *World Wide Web* má v dnešní době mnohem více použití než jen distribuci webových stránek. Díky jejímu nesmírnému rozšíření se stává univerzální platformou pro komunikaci elektronických zařízení. Softwarový systém navržený pro podporu spolupráce zařízení na síti se nazývá *Webová služba (WS)*. [1]

U webových služeb je kladen velký důraz na interoperabilitu. Díky WS mezi sebou mohou komunikovat systémy z naprosto odlišných prostředí, naprogramované v různých jazycích, běžící na rozdílných platformách i zařízeních. Z tohoto důvodu jsou u webových služeb tendence k používání známých a standardizovaných protokolů a formátů. Ve zkratce je možné říci, že webové služby poskytují jednotný a velmi univerzální komunikační prostředek.

Representational state transfer, zkráceně *REST*, je moderním termínem v této oblasti. Jedná se o sadu principů a doporučení pro návrh architektury webových služeb. Služby, které tyto principy splňují, se označují slovem *RESTful* nebo českým termínem *RESTové*. Základním pojmem v architektuře REST je zdroj. Zdrojem jsou v tomto kontextu jakákoliv data, která mají svůj jednotný identifikátor *URI (Uniform Resource Identifier)*. Pro manipulaci s těmito zdroji služby používají standardní operace protokolu HTTP. Architektura REST je takto v kontrastu s architekturou *SOAP (Simple Object Access Protocol)*, která je orientována primárně na akce a činnosti.

Téma architektury REST v kontextu webových služeb jsem si vybral, protože jde o jeden ze současných trendů vyskytujících se čím dál častěji v mnoha moderních aplikacích. Rozsáhlé webové aplikace a služby, např. *Twitter*[1] nebo *Amazon S3*[2] nabízí kromě hlavního uživatelského rozhraní také rozhraní RESTové. Důvodem je, dle mého názoru, hlavní výhoda RESTových služeb, a to jednoduchá spolupráce s externími aplikacemi

1. `https://dev.twitter.com/docs/api`
2. `http://docs.amazonwebservices.com/AmazonS3/latest/API/`

a systémy, například mobilními telefony a tablety. Vzhledem k tomu, že jsou RESTové služby velmi snadno konzumovatelné, jsou velmi dobrým prostředkem pro komunikaci s co největším spektrem klientů. Z těchto důvodů může být REST využit i u uzavřených komerčních systémů. RESTové API je poté buď doplňkem už existujícího rozhraní, nebo i primárním rozhraním použitým pro veškerou komunikaci zařízení v systému.

Má práce se zaměřuje na vývoj těchto služeb na platformě .NET. Hlavním důvodem je právě velké rozšíření této platformy v komerční sféře. Oficiální nástroje pro tvorbu webových služeb v architektuře REST pro tuto platformu jsou stále ve vývoji. Poslední verze *Windows Communication Foundation 4* například podporu architektury REST podstatně rozšiřuje. [2] Právě kvůli stálému vývoji jsou, podle mého názoru, informační zdroje pro vývoj RESTových služeb na platformě .NET roztříštěné. Chybí ucelený text respektující aktuální novinky, řešící běžné problémy při vývoji, který se zabývá širším kontextem.

Na architekturu REST nabízím pohled z několika úhlů. V práci se věnuji teoretickému výkladu architektury založeném především na práci Roye T. Fieldinga, jednoho z průkopníků této architektury. Výklad doplňuji o příklady z praktického použití v současném webu. Dále rozebírám otázky a problémy týkající obecně návrhu RESTových služeb. Podrobněji se věnuji také implementaci RESTových služeb ve Windows Communication Foundation. Dále předvádím různé nástroje, které mohou být užitečné při vývoji, testování, nebo konzumaci RESTových služeb.

Součástí práce je i přiložená aplikace – diskusní služba s RESTovým rozhraním. Tato služba může sloužit buď jako jednoduchý vzor jiným službám nebo jako pomůcka pro porozumění jednotlivým principům RESTových služeb. K práci přikládám také připravené studijní materiály použitelné pro výuku tohoto tématu. Patří sem podklady pro přednášku a úloha pro praktické cvičení.

Kapitola 1

Architektura REST

V následující kapitole představím architektonický styl *Representational state transfer* (*REST*). Vysvětlím základní pojmy této architektury, její požadavky a omezení. Dále je v této kapitole rozebrán úzký vztah architektonického stylu REST a protokolu HTTP.

1.1 Historie

Historie architektury REST je spojena s disertační prací Roye T. Fieldinga, spoluzakladatele projektu *Apache HTTP Server* a jednoho z hlavních autorů webového protokolu HTTP. [3] Fielding pomohl navrhnout první webové služby a zabýval se výzkumem a vysvětlením proč a jak síť Web funguje. V jeho práci jsou formulovány základní požadavky na architekturu REST a představeny výhody, které z dodržení požadavků plynou, jako například interoperabilita a škálovatelnost. [4]

Architektonický styl REST byl vyvíjen pro distribuované prostředí, pro potřeby čím dál tím více se rozšiřujícího World Wide Webu okolo roku 2000. World Wide Web proto můžeme považovat za největší implementaci tohoto stylu. Ačkoliv některé z principů této architektury jsou u něj zanedbány, pravděpodobně nenajdeme systém tak dokonale propojených hypermédií, jako je Web.

1.2 Orientace na zdroje

REST patří mezi architektury orientované na zdroje, zkráceně ROA (*resource oriented architecture*). Znamená to, že je celá architektura založená na konceptu tzv. *zdroje (resource)*. Tímto pojmem označujeme u architektonického stylu REST jakýkoliv objekt uložený na síti, který

lze v rámci systému identifikovat. Zdrojem mohou být jakákoliv data, libovolný soubor informací.

Základní myšlenkou ROA je, že se nezabývá tolik způsobem, *jak* data měnit a zpracovávat. Zabývá se spíše jejich *strukturou, identifikací a reprezentací*. Samotné operace, které se nad daty provádí, jsou vždy standardní.

1.3 Základní principy

Následuje představení nejdůležitějších požadavků na architekturu webových služeb, které Fielding ve své práci popisuje. K některým z nich nabízím pro lepší představu a pochopení informace o tom, jakou mají obdobu u dnešního webu. Webové služby, které následující požadavky splňují, označujeme slovem *RESTful*, já se ve své práci držím českého termínu *RESTové*.

1.3.1 Komunikační vrstvy

Komunikace u RESTových webových služeb probíhá mezi klientem a serverem. Je velmi důležité, aby jak klient tak i server byly zaměnitelné. Jednotné komunikační rozhraní[1] je základním kamenem webových služeb. Fielding ve své práci také požaduje, aby bylo možné vkládat mezi klienta a server další vrstvy, mezilehlé servery a brány. Ty by ovšem měly být na sobě nezávislé a vzhledem k ostatním vrstvám neviditelné. Tyto vrstvy můžeme v dnešním webu vidět například jako:

- vrstvy, které ukládají průchozí data do mezipaměti (*cache*) a zmenšují tak odezvu cílového serveru;
- vrstvy, které řídí příchozí požadavky, podle jejich priority je propouští dále a vyrovnávají tak zátěž serveru (*load balancing*);
- vrstvy, které zajišťují dodatečné zabezpečení webových služeb.

1.3.2 Vyjednávání obsahu

Zdroje mívají různou reprezentaci, tou může být například textový dokument, webová stránka, XML dokument, apod. Zdroj se stejným

1. Fieldingem označováno jako *Uniform Interface*.

identifikátorem tedy může být podle požadavku klienta prezentován jiným způsobem. Fielding tento pojem označuje jako *vyjednávání obsahu (content negotiation)*.

U webových stránek se s rozdílnou reprezentací zdrojů příliš nesetkáváme. V drtivé většině případů má každý zdroj právě jednu reprezentaci. Praktickým příkladem zdroje s více reprezentacemi jsou například webové stránky určené pro mobilní prohlížeče. Jedná se o stejný zdroj, nicméně je prezentován jiným způsobem pro desktopové a mobilní prohlížeče.

1.3.3 Bezstavovost

Jedním z velmi omezujících požadavků na RESTové webové služby je bezstavovost. Každý klientský požadavek by měl být samostatný, nezávislý na ostatních. Klientské požadavky proto musí obsahovat všechny potřebné informace[2]. Stejně tak i odpověď serveru by měla pokaždé obsahovat kompletní data, pokud si uživatel explicitně nevyžádá jinou reprezentaci zdroje.

Tento požadavek se na první pohled může zdát jako striktní a zbytečný, protože se při více požadavcích posílají stejné informace vícekrát. Ze splnění bezstavové komunikace ale plynou pro webovou službu velmi velké výhody. Stává se tím velmi snadno škálovatelná. Aplikační servery lze jednoduše množit a zaměňovat, protože neuchovávají žádný stav. Veškeré informace o stavu jsou přenášeny v požadavku. Tento způsob přenášení stavu (*state transfer*) je jednou z hlavních myšlenek architektonického stylu REST.

1.3.4 Hypermédia pro reprezentaci stavů aplikace

Pojmem *hypermédia* označujeme média, která jsou propojena referencemi (tzv. *hyperlinky*) s jinými médii. Uživatel se tak může následováním hyperlinku dostávat k dalším informacím nebo vracet zpět. Typickým hypermédiem jsou dnešní webové stránky, na kterých můžeme sledovat, jak je informační síť hypermédií vysoce použitelná v praxi.

Fielding ve své práci požaduje, aby byla hypermédia použita pro reprezentaci stavů aplikace. Tento požadavek, často označovaný zkratkou

2. Fieldingem je tento způsob komunikace označován jako *client-stateless-server (CSS)*

HATEOAS (Hypermedia as the Engine of Application State), odlišuje architekturu REST od většiny ostatních architektur pro distribuované prostředí.

HATEOAS říká, že klientovi RESTové služby stačí jediná informace pro práci s aplikací, což je vstupní URI. Všechny další informace a operace by měly být dostupné skrze síť hypermédií. Nepotřebuje tedy znát přesné rozhraní služby, nepotřebuje mít předpřipravené URI všech potřebných zdrojů. K těmto informacím se dostane dynamicky. Síť hypermédií pak připomíná automat (*state machine*), ve kterém jsou jednotlivé stavy hypermédii a přechody mezi nimi hyperlinky.

1.3.5 Code-On-Demand

Jedním z mechanismů RESTových webových služeb je tzv. *kód na vyžádání (Code-On-Demand)*. Na rozdíl od ostatních zmíněných pojmů není závazný a záleží jen na konkrétní aplikaci, jestli jej bude implementovat.

Kód na vyžádání rozšiřuje funkcionalitu klienta. Server poskytuje klientovi možnost stáhnout si různé applety a skripty, které klient spouští sám. Díky tomu není potřeba, aby implementoval všechny potřebné funkce. Server může kód dalších funkcí dodat na vyžádání.

U dnešních webových stránek můžeme tento mechanismus vidět například jako klientský Javascript. Ten je vykonáván uvnitř prohlížeče a dodáván vzdáleným serverem.

1.3.6 Cache

Podpora pro vyrovnávací paměť (*cache*) je jedním z dalších omezení na systémy v architektonickém stylu REST. Z podpory cache plynou pro systém následující výhody:

- **odolnost vůči chybám** – systém, který má připravená data ve vyrovnávací paměti je může poskytnout klientovi i v případě poruchy;
- **menší zatížení serverů** – servery nemusí všechna data zpracovávat pokaždé, pokud je možnost mohou využít už předpřipravené výsledky;

- **rychlost komunikace** – načtení dat z vyrovnávací paměti je rychlejší, než zpracovávání dat při každém požadavku, proto se zkrátí odezva serveru;

- **snadnější škálovatelnost** – díky vyrovnávací paměti je možné obsluhovat mnohem více klientů, u kterých se často vyskytují stejné požadavky na server.

Na stranu druhou mohou mechanismy vyrovnávací paměti zvýšit nekonzistenci dat a aplikace tak může být více náchylná k chybám.

1.4 Srovnání REST a SOAP

Orientaci na zdroje RESTových služeb můžeme položit do kontrastu s principy protokolu *Simple Object Access Protocol (SOAP)*. SOAP je protokol určený pro výměnu zpráv po síti. Zprávy přenáší ve formátu XML s využitím protokolu HTTP jako komunikačního kanálu.

Na rozdíl od webových služeb REST, které používají pro manipulaci s daty pouze standardní sadu operací, je možné pomocí protokolu SOAP modelovat rozhraní poskytující libovolné operace. Protokol SOAP se nezabývá zveřejňováním zdrojů a jejich reprezentací, poskytuje prostředky pro definici vlastních operací, které s daty pracují. Je tak méně omezující a nabízí větší volnost. Hlavní rozdíly, které jsou mezi dvěma zmíněnými přístupy jsou, dle mého názoru, následující:

- RESTové služby jsou navržené především pro publikaci dat, zatímco služby využívající SOAP spíše pro zpracování dat.

- REST nabízí jednotné rozhraní, SOAP zase rozhraní založené na pevně definovaných kontraktech. Rozhraní RESTových služeb je méně strukturované, a je proto snáze zpracovatelné a lidsky čitelné. Na druhou stranu kvůli jednotnému rozhraní nemá tolik možností. Při využití protokolu SOAP je možné přesně hlídat formát a typy všech zpráv, REST je v tomto ohledu mnohem více benevolentní.

- RESTové služby jsou bezstavové, SOAP podporuje i složitější komunikaci – stavovou (*stateful*) a asynchronní.

- Webové služba používající protokol SOAP má zpravidla jeden přístupový bod. RESTová služba na druhou stranu zveřejňuje každý zdroj na jiné adrese.

Oba přístupy však mají v současném světě webových služeb své místo. Mezi webovými službami založenými na REST a SOAP je možné přecházet, existují i služby, které kombinují obojí.[3] [4] [5]

1.5 Vztah REST a HTTP

Mnoho z těchto požadavků webové služby splní už samotným využitím protokolu HTTP, jehož je Fielding spoluautorem. Mezi tyto požadavky patří například jednotné rozhraní, také identifikace zdrojů pomocí URI. Dále je to podpora pro další mezivrstvy mezi klientem a serverem. Všeobecně protokol HTTP poskytuje i prostředky pro vyjednávání obsahu nebo vyrovnávací paměť pomocí příslušných hlaviček. Nicméně ne všechny tyto požadavky jsou dodrženy jen samotným použitím protokolu HTTP. Například zmíněné vyjednávání obsahu nebo vyrovnávací paměť musí mít podporu i na aplikační vrstvě. Dále formulované požadavky, jako např. bezstavovost a všeobecná orientace na zdroje jsou již podstatným omezením i pro návrh samostatných aplikací.

Protokol HTTP bych tedy nepovažoval za implementaci REST principů, ale spíše za prostředek, který usnadňuje vývoj těchto aplikací, protože mnoho principů podporuje. Zřejmou výhodou je také, že jde o zažitý standard pro komunikaci v oblasti webu. Ve zkratce tedy část principů REST implementuje HTTP protokol, část musí dodržovat i samostatné aplikace.

1.6 Výhody RESTových služeb

Nyní následuje shrnutí hlavních výhod, které plynou ze zvolení architektury REST pro vývoj webových služeb.

Jednoduchý vývoj klientů. Jedním z požadavků na RESTové služby je jednotné rozhraní. Vzhledem k tomu, že RESTové služby jsou

3. `http://www.flickr.com/services/api/`
4. `https://www.x.com/developers/ebay/products`
5. `http://docs.amazonwebservices.com/AmazonS3/latest/API/`

v praxi založeny na protokolu HTTP, je vývoj klientů služby velmi jednoduchý. Většina platforem a jazyků nabízí nástroje pro práci s protokolem HTTP, proto je možné konzumovat RESTové služby téměř odkudkoliv.

Libovolný formát zpráv. Oproti protokolu SOAP, který pro veškerou komunikaci prosazuje formát XML, je v tomto ohledu REST na formátu nezávislý. Dnešní web je plný různých formátů, běžně používané jsou například XML, JSON,[6] HTML, vCard,[7] iCalendar,[8] dále formáty pro audio, video, obrázky... REST v jejich použití uživatele nijak neomezuje. Na rozdíl od protokolu SOAP není potřeba tyto formáty uzavírat do dalších obálek.

Nízká provázanost klienta a služby. Díky jednotnému rozhraní není klient se službou tolik provázán. Je možné například rozšiřovat RESTové rozhraní bez toho, aby byla narušena funkčnost současných klientů.

Jednoduchost. Požadavky i odpovědi jsou velmi jednoduché a lidsky čitelné.

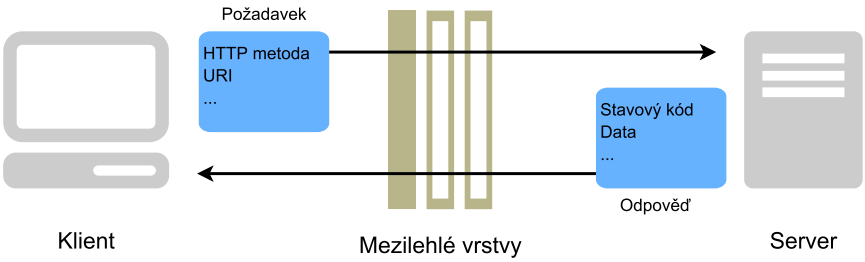
Orientace na zdroje. Složité systémy lze, podle mého názoru, mnohem snáze dekomponovat na jednotlivé zdroje než na velkou sadu metod. Některé procesy sice není jednoduché modelovat pomocí jednotného RESTového rozhraní, ale při použití správné struktury identifikátorů je systém orientovaný na zdroje lépe uchopitelný.

6. JSON = *JavaScript Object Notation*
7. Specifikace vCard (RFC 6350): `http://tools.ietf.org/html/rfc6350`
8. Specifikace iCalendar (RFC 5545): `http://tools.ietf.org/html/rfc5545`

Kapitola 2

Protokol HTTP

Hypertext Transfer Protocol, zkráceně *HTTP*, je síťový protokol pracující v aplikační vrstvě (dle ISO/OSI modelu) navržený pro použití v distribuovaném prostředí, jakým je například současný web. Je primárně určen pro výměnu informací ve formě hypermédií. Výměna informací probíhá mezi dvěma základními entitami – klientem a serverem. Ty komunikují pomocí dvou základních druhů zpráv, kterými jsou *požadavek klienta* a *odpověď serveru*. Někdy bývají také označovány i původními anglickými termíny *request* a *response*. [3] Zprávy jsou ve formě čistého textu obsahujícího kromě samotných dat některé dodatečné informace a mají jednoduchý, pevně definovaný formát.



Obrázek 2.1: Znázornění komunikace přes HTTP.

2.1 Zdroj

Protokol HTTP je schopný identifikovat zdroj, který se nachází kdekoliv na síti. V případě sítě WWW tedy nabízí možnosti, jak jednotným způsobem nalézt zdroj kdekoliv na světě. Pro tento identifikátor používá protokol HTTP označení *jednotný identifikátor zdroje*, zkráceně

URI (Uniform Resource Identifier). URI, které pro identifikaci zdroje používá jeho umístění na síti, se nazývá *URL (Uniform Resource Locator)*. Protože URL se pro identifikaci zdrojů na síti používá velmi často, bývají pojmy URI a URL zaměňovány. [5] Správné struktuře při návrhu URI věnuji v práci samostatnou sekci (4.1), protože jde o jednu z hlavních složek RESTového rozhraní.

2.2 Požadavek klienta

Klientský požadavek má následující strukturu:

```
(HTTP metoda) (URI) (verze HTTP)
(hlavičky)
(prázdný řádek)
(případná data)
```

- První řádek obsahuje HTTP metodu, identifikátor zdroje, se kterým požadavek pracuje (URI) a verzi HTTP (např. `HTTP/1.1`).
- Od druhého řádku začíná sekce obsahující HTTP hlavičky, tedy dodatečné údaje požadavku.
- Následuje obsah požadavku, odesílaná data, oddělený od hlaviček prázdným řádkem. Obsah se však nemusí vyskytovat vždy, záleží na HTTP metodě a konkrétním požadavku.

2.2.1 HTTP metody

HTTP metody označují druh manipulace se vzdáleným zdrojem nebo také typ operace, který požadavek má na vzdáleném serveru provést.

V současnosti používané verzi protokolu, HTTP/1.1, je definováno 8 různých metod. V následující tabulce uvádím význam a praktické použití několika nejpoužívanějších ve vztahu k architektuře REST. Pro větší podrobnosti k HTTP metodám odkazuji na RFC 2616.[1]

1. `http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html`

GET Metoda značí požadavek pro získání reprezentace daného zdroje. Tato metoda je pravděpodobně nejpoužívanější HTTP metodou, protože je používána pro čtení dat. Součástí **GET** požadavku nejsou žádná data, jakékoliv informace předané vzdálenému serveru se proto nacházejí buď v hlavičkách nebo URI.

POST Požadavek touto metodou odesílá přiložená data na server. Je určen pro vkládání nových dat, nicméně často používán jako univerzální metoda pro zápis dat. Požadavek **POST** je odesílán na URI kolekce, či jiného objektu, který data zpracuje. Může jít např. o přidání objektu do kolekce či připojení dat k existující entitě. Odpověď serveru by měla obsahovat URI vytvořeného objektu.

PUT Podobně jako **POST** požadavek vkládá data na server, ale má odlišný význam a použití. Požadavek **PUT** je směrován na cílovou URI vkládaného objektu. Výsledkem požadavku **PUT** je, že na server konkrétní URI uloží (vytvoří, či přepíše) přiložená data. Metoda **PUT** na rozdíl od metody **POST** patří mezi tzv. *idempotentní* (vysvětleno níže). Metoda **POST** je často používána pro *vytváření*, protože není známa URI vytvářeného objektu. Metoda **PUT** je potom používána pro *změny*, protože URI je již předem známá.

DELETE Požadavek na smazání zdroje.

OPTIONS Požadavek touto metodou zjišťuje dostupné komunikační možnosti pro konkrétní URI. Odpovědí je seznam HTTP metod, které daný zdroj podporuje.

HEAD Význam metody je téměř identický metodě **GET**. Rozdílem je, že odpověď serveru by neměla obsahovat žádná data, ale pouze hlavičky a stavové informace.

Podle svého chování jsou některé z HTTP metod dále označovány jako *bezpečné*. Bezpečné metody jsou ty, které nemění stav serveru. Je zavedenou konvencí, že metody **GET** a **HEAD**, nemění stav serveru, proto jsou zařazeny mezi bezpečné.

Některé z HTTP metod mají také vlastnost zvanou *idempotence*. Mezi idempotentní metody se řadí **GET**, **HEAD**, **PUT**, **DELETE** a **OPTIONS**. Jsou to metody, které nezpůsobují vedlejší efekty. Jejich společným

znakem je, že identický požadavek poslaný vícekrát za sebou má stejný efekt jako jeden požadavek.

U aplikací běžících ve webovém prohlížeči, např. u webových stránek, bývají většinou implementovány pouze metody `GET` a `POST`. Některé metody, např. `OPTIONS`, prohlížeč při obsluze webových stránek nepoužívá. Požadavky, které by dle specifikace HTTP měly používat metody `DELETE` či `PUT` jsou potom odesílány obecnější metodou `POST`. Je to způsobeno omezením webových prohlížečů, které dodatečné metody nepodporují. U RESTových webových služeb mají ale smysl i ostatní HTTP metody, protože právě HTTP metody jsou prostředkem pro jednotný způsob manipulace se zdroji, což je jeden ze základních požadavků na RESTové služby.

2.2.2 Hlavičky požadavku

HTTP hlavičky obsahují metadata a jiné dodatečné informace ke klientskému požadavku. Každá hlavička zabírá právě jeden řádek klientského požadavku a je ve formátu „`(Název hlavičky): (hodnota)`“.[2]

Příkladem hlaviček mohou být například:

- dodatečné údaje k datům přiloženým k požadavku (hlavičky `Content-Type` a `Content-Length`);
- informace, jaká data umí klient zpracovat (hlavičky `Accept`, `Accept-Charset`, `Accept-Encoding`, `Accept-Language`);
- informace o agentu uživatele (softwaru, který za uživatele odesílá požadavek), např. webovém prohlížeči (hlavička `User-Agent`);
- doménové jméno serveru, od kterého je relativně odvozena URI (hlavička `Host`);
- specifikace mechanismu pro cache (ukládání do vyrovnávací paměti) – hlavička `Cache-Control`;
- cookies, dodatečná data související s konkrétním sezením (session) – hlavičky `Cookies` a `Set-Cookies`.

2. V názvu hlavičky se nepoužívá pro oddělení slov mezer, slova bývají oddělena pomlčkou. Hodnotou je potom textový řetězec, který však nesmí přesahovat řádek vyhrazený pro každou hlavičku.

Kromě standardně používaných hlaviček je možné přidat i vlastní hlavičky použitelné pro specifické účely HTTP transakce. Pro vlastní hlavičky je zavedena konvence, která požaduje, aby jejich název měl prefix „`X-`“.

2.3 Odpověď serveru

Na klientský požadavek reaguje vzdálený server odpovědí (response). Tato odpověď má následující strukturu:

```
(verze HTTP) (stavový kód) (stavová hláška)
(hlavičky)
(prázdný řádek)
(případná data)
```

- První řádek obsahuje, podobně jako klientský požadavek, verzi HTTP. Za ní následují informace o stavu zpracovaného požadavku.
- Následující řádky jsou podobné jako u klientského požadavku, liší se jen druhy použitých HTTP hlaviček.

2.3.1 Stavové informace

Druh odpovědi serveru určuje *stavový kód (status code)*. Stavový kód je trojciferné číslo, které informuje o tom, jak dopadl klientský požadavek.

Společně se stavovým kódem je posílána i *stavová hláška (reason phrase)*, která obsahuje textový popis stavového kódu, jeho zdůvodnění.

Stavové kódy rozděluje HTTP podle významu do pěti kategorií:

Informační (1xx)
: Požadavek byl obdržen, ale ještě nebyl zpracován.

Úspěch (2xx)
: Požadavek byl úspěšně zpracován.

Přesměrování (3xx)
: Klient musí provést další akci, aby byl požadavek zpracován.

Chyba klienta (4xx)
Požadavek skončil chybou způsobenou klientem.

Chyba serveru (5xx)
Požadavek skončil chybou způsobenou serverem.

Celý výčet chybových kódu včetně jejich významu je k dispozici jako součást RFC 2616.[3]

3. `http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html`

Kapitola 3

REST na platformě .NET

3.1 Implementace ve WCF

Windows Communication Foundation (WCF) je aplikační rámec pro vývoj, nasazení a propojování servisně orientovaných aplikací (SOA) na platformě .NET. Je součástí .NET Frameworku od verze 3.0 a v současné době neustále ve vývoji. [6] WCF nabízí podporu pro velkou škálu komunikačních protokolů a vlastní vrstvu abstrakce nad použitými komunikačními technologiemi. Ve své práci se zabývám podporou pro RESTové webové služby v kontextu WCF. Nejdříve proto představím základní prvky programovacího modelu ve WCF.

3.1.1 Základní pojmy

Služba (service) je základním stavebním prvkem WCF. Jedná se o softwarový systém, který nabízí rozhraní na jednom či více *endpointů*, kde každý endpoint poskytuje alespoň jednu operaci.

Endpoint je konstrukt, skrze který jsou přijímány nebo odesílány zprávy.[1] Skládá se ze tří hlavních částí:

- **umístění (adresy)**, které definuje *kam* jsou zprávy posílány;
- **bindingu**, mechanismu použitého pro komunikaci, který popisuje *jak* mají být zprávy posílány;
- **kontraktu**, který definuje *co* služba umí zpracovávat, formát vstupních a výstupních zpráv.

1. Česky je možné termín přeložit jako *koncový bod*, nicméně v práci se držím zažitého anglického termínu.

Binding popisuje způsob, kterým komunikují endpointy služeb ve WCF.[2] Má dvě hlavní povinné složky:

- **výběr transportního protokolu** – definuje, který protokol je určen pro přenos zpráv (TCP, HTTP, MSMQ[3], vlastní, ...);
- **kódování zpráv** – definuje, jak jsou zprávy kódovány (binárně, textově, vlastním způsobem, ...).

Kromě zmíněných dvou může také popisovat, jak se mají chovat transakce a sezení, jak je služba zabezpečena, apod.

Kontrakt (contract) popisuje rozhraní služby zveřejněné endpointem. WCF rozlišuje 4 základní typy kontraktů:

- **Service contract** – Definuje, které operace může služba provést (součást endpointu).
- **Data contract** – Definuje datové typy, které jsou parametrem operací.
- **Fault contract** – Definuje, jak vypadají chyby, které mohou v aplikaci nastat.
- **Message contract** – Definuje, jakou strukturu má celá odesílaná/přijímaná zpráva.

Behavior umožňuje možné dále konfigurovat a rozšiřovat funkcionalitu základních prvků WCF.[4] Mezi základní typy behaviors patří:

- **Service behavior** – Popisuje chování služby, popisuje například způsoby, jakým se vytváří její instance (*instancing*), vztahy vláken a aplikací, a další;
- **Endpoint behavior** – Umožňuje inspekci a další zpracování vstupních a výstupních zpráv endpointu;
- **Operation behavior** – Ovlivňuje chování služby na úrovni operací, je tak možné kontrolovat např. serializaci vstupních a výstupních parametrů operace.

2. Česky možno přeložit jako *vazba*.
3. MSMQ = *Microsoft Message Queuing*.
4. Česky možno přeložit jako *chování*.

3.1.2 Hosting aplikací

Windows Communication Foundation podporuje hostování aplikací na webovém serveru IIS (Internet Information Services), ve službě Windows, procesu COM+, nebo vlastní hostovací aplikací (tzv. *self-hosting*). Pro hostování aplikace je potřeba všechny součásti propojit dohromady:

1. Je potřeba vytvořit tzv. *service host*, který řídí hostování vlastní implementace služby.

2. Service host je následně propojen s endpointy, které popisují vstupní a výstupní kanály služby. Je důležité, aby služba implementovala kontrakty všech endpointů.

3. Endpointy mají určené adresy a jsou jim přiřazeny bindingy, které definují komunikační protokol.

4. Jak endpointy, tak bindingy, tak služby mohou mít ještě svou dodatečnou konfiguraci.

5. Posledním krokem je spuštění service hostu tzv. *hostujícím procesem*. Hostující proces může poskytovat například webový server IIS nebo v případě self-hostingu vlastní aplikace. Tento krok se už může lišit podle typu hostingu.

   Pro IIS je potřeba vytvořit virtuální adresář, který je směřován k assembly aplikace a konfiguračním souborům.

3.1.3 Podpora pro REST

Již od verze 3.5 nabízí .NET Framework podporu pro RESTové služby, ve které představuje způsob, jak pomocí kontraktů služeb zveřejnit zdroje na určitých adresách. S těmito zdroji také umožňuje manipulovat s využitím celého potenciálu protokolu HTTP.

Základním kamenem RESTových služeb ve WCF je `WebHttpBinding`. Kromě tohoto bindingu nabízí WCF i další dva pro práci s protokolem HTTP – `BasicHttpBinding` a `WsHttpBinding`. Ty jsou však určeny pro použití s protokolem SOAP. Oproti těmto dvěma, `WebHttpBinding` nepoužívá protokol HTTP jen jako komunikační tunel pro odesílání zpráv. Používá jej na publikaci zdrojů na určitých URI.

3.1.4 Kontrakty RESTových služeb

Kontrakty služeb, jejichž endpointy používají `WebHttpBinding` potom umožňují mapovat práci se zdroji pomocí HTTP metod na operace kontraktu, viz následující ukázku (3.1).

```
[ServiceContract]
public interface INewsService
{
    [WebGet(UriTemplate = "/news")]
    ICollection<NewsItem> GetAllNews();

    [WebGet(UriTemplate = "/news?tag={tag}")]
    ICollection<NewsItem> GetNewsByTag(string tag);

    [WebInvoke(Method = "POST", UriTemplate = "/news")]
    NewsItem CreateNewsItem(NewsItem item);

    [WebInvoke(Method = "PUT", UriTemplate = "/news/{id}")]
    NewsItem UpdateNewsItem(string id, NewsItem item);

    [WebInvoke(Method = "DELETE", UriTemplate = "/news/{id}")]
    void DeleteNewsItem(string id);
}
```

Výpis kódu 3.1: Kontrakt jednoduché RESTové služby.

V ukázce lze vidět, že každá z operací je označena atributem `WebGet` nebo `WebInvoke`. Tyto atributy říkají, že operace kontraktu je ve skutečnosti jednou ze standardních operací nad zdrojem.

Atribut `WebGet` říká, že na dané adrese webová služba přijímá požadavek HTTP metodou **`GET`**. Atribut `WebInvoke` říká, že webová služba přijímá požadavky HTTP metodu **`POST`**. Atribut `WebInvoke` je také používán pro všechny ostatní HTTP metody (běžně **`DELETE`** a **`PUT`**), ty je potřeba nastavit do parametru `Method`. Tomu, které HTTP metody použít pro jaký účel se věnuji v kapitole o HTTP (viz podsekci 2.2.1).

U každého z atributů je ještě potřeba nastavit adresu, na které je operace se zdrojem zveřejněna. Pro snadnou definici adres zdrojů poskytuje WCF mechanismus šablon URI (`UriTemplate`).

Pomocí `UriTemplate` je možné definovat šablonu adresy, na které je zdroj zveřejněn. Jednotlivé části adresy se mohou mapovat na vstupní parametry operace.

3.1.5 Zpracování chyb

Nedílnou součástí jednotného rozhraní RESTových služeb jsou také chyby. Zatímco základním mechanismem pro práci s chybami v jazyce C# jsou výjimky, RESTové služby hlásí veškeré chyby pomocí *stavového kódu odpovědi* (popsáno v podsekci 2.3.1).

Ve Windows Communication Foundation verze 4 přibyl nový typ výjimek `WebFaultException`, který umožňuje společně s vyhozením výjimky vrátit i příslušný stavový kód. Chybová odpověď serveru ale nemusí obsahovat pouze stavový kód. Často je potřeba na klienta odeslat dodatečná data, například pro přesnou identifikaci problému. WCF proto s novým typem výjimek nabízí i možnost přibalit libovolný objekt do datové části odpovědi. Pro tyto účely slouží generická verze typu `WebFaultException`. Jednoduché zpracování chyby pomocí `WebFaultException` předvádím na následující ukázce (3.2).

```
NewsItem GetNewsItemById(string id)
{
    long newsItemId = long.Parse(id);
    try
    {
        return Db.NewsItems.First(x => x.Id == newsItemId);
    }
    catch (InvalidOperationException exception)
    {
        throw new WebFaultException<string>(
        "News item was not found.",
        HttpStatusCode.NotFound);
    }
}
```

Výpis kódu 3.2: Zpracování chyb pomocí `WebFaultException`.

3.1.6 Podpora pro vyjednávání obsahu

Důležitým znakem RESTových služeb je také již zmíněné *vyjednávání obsahu (content negotiation)* (viz podsekci 1.3.2). Ve zkratce lze říci, že vyjednávání obsahu je mechanismus, jak zveřejnit různé reprezentace téhož zdroje.

WCF nabízí u endpointu pro RESTové služby (`WebHttpEndpoint`) možnost specifikovat tzv. `ContentTypeMapper`. Tento objekt podle typu klientského požadavku vybere formát vstupní/výstupní zprávy. V současné verzi WCF jsou podporované formáty *JSON*, *XML*, *raw* (pro binární data).

3.2 Alternativy k WCF

V následující kapitole představím některé z alternativních technologií použitelných pro vývoj webových služeb v architektuře REST na platformě .NET.

3.2.1 ServiceStack

Projekt *ServiceStack*[5] je open-source alternativou Windows Communication Foundation. Ačkoliv kromě RESTu nabízí i přístup RPC,[6] není s ním tak úzce spjatý, jako WCF. Pro vývoj RESTových služeb je určen už od počátku.

Dle mého názoru se jedná o jednodušší řešení, než WCF, neboť struktura celé knihovny není tak hluboká a komplexní. Mezi jeho klady patří také vysoká rychlost[7] a podpora pro operační systém Linux (díky projektu *Mono*).[8]

3.2.2 OpenRASTA

Projekt *OpenRASTA*[9] je dalším alternativním open-source řešením určeným pro vývoj RESTových služeb. Na rozdíl od WCF nebo ServiceStack je orientovaný čistě na REST. Má také odlišný koncept, který je více

5. `http://www.servicestack.net/`
6. RPC = *remote procedure call* (vzdálené volání procedur).
7. Měření zveřejněno zde: `http://www.servicestack.net/benchmarks/`.
8. `http://www.mono-project.com/`
9. `https://github.com/openrasta/openrasta/wiki`

orientovaný na zdroje. Namísto na centrální rozhraní služby se více soustředí na samotné zdroje. S nimi pak manipulují tzv. *handlery*, formátování výstupu a čtení vstupu poté zajišťují *kodeky (codecs)*. Tento přístup je, podle mého názoru, pro RESTové služby vhodnější a je velkou výhodou tohoto projektu.

3.2.3 ASP.NET MVC

Jedním z možných způsobů implementace RESTových služeb na platformě .NET je také použití technologie *ASP.NET MVC*. Ačkoliv je technologie ASP.NET MVC navržena primárně pro obsluhu webových stránek, má dobrou podporu pro protokol HTTP a v některých případech může být vhodným řešením pro služby ve stylu REST. Oproti předchozím nabízí opět jiný programovací model založený na návrhovém vzoru *Model–view–controller*. Mezi nevýhody oproti WCF patří nižší možnosti konfigurace chování služby a omezené možnosti pro hosting služby. Výhodou je ale jednoduchost z pohledu programátora či velmi dobrá implementace routingu URL.[10]

10. *Routing* označuje směrování příchozích požadavků na jednotlivé akce podle URL.

Kapitola 4

Úskalí návrhu RESTových služeb

V následující kapitole se opět vzdaluji od platformy .NET a věnuji některým otázkám, problémům a doporučením týkajícím se obecně vývoje a návrhu RESTových služeb. V závěru této kapitoly také popisuji konzumaci RESTových služeb a přístup z odlišných prostředí.

4.1 Správná struktura URI

Návrh správného a použitelného rozhraní (API) RESTové webové služby je úzce spjatý se správnou strukturou a hierarchií URI. Proto v následující podkapitole popisuji některé z principů, kterých je dobré se při návrhu URI držet.

Obecnou strukturu URI definuje RFC 3986 následovně:[1] [7]

```
(schéma):(hierarchická část)[?(query string)][#(fragment)]
```

Schéma definuje z čeho je URI složeno a o jaký druh URI se jedná. V této části se budu věnovat pouze URI schématům běžně používaným pro RESTové webové služby, tedy schématům `http` a `https`. Mezi ostatní URI schémata pro představu patří například `geo` (pro identifikaci podle geografické lokace) nebo `data`, které v samotné URI umožňuje uchovávat datové bloky. Následující informace proto nemusí platit pro všechna schémata.

Hierarchická část adresy obsahuje informace pro identifikaci zdroje v hierarchické struktuře popsané daným schématem. Většinou začíná lomítky (`//`) za nimiž následuje hostname[2] serveru a port. Před samotným hostname ještě může být uveden uživatel a heslo pro přístup

1. Části v hranatých závorkách jsou nepovinné.
2. Pojmem *hostname* souhrně označuji doménové jméno nebo IP adresu serveru.

k danému serveru. Tento mechanismus ale není vhodné používat jako prostředek pro autentizaci v různých částech samotné webové služby, je určen pouze pro autentizaci vůči vzdálenému serveru, nikoliv vůči webové službě. Za informacemi o připojení k serveru poté následuje lokální cesta ke zdroji.

4.1.1 Cesta ke zdroji

Cesta ke zdroji je řetězec znaků, jehož jednotlivé části jsou odděleny lomítky. U RESTových služeb zastupuji názor, že by měly být lidsky čitelné, protože je lidé mohou často ručně zapisovat. Proto by se v nich nemělo vyskytovat mnoho speciálních znaků a měl by být kladen důraz na jednoduchost a srozumitelnost.

Cesta by měla znázorňovat hierarchii zdrojů tak, jak je skutečně modelována v aplikační logice webové služby. Jakékoliv další informace potřebné pro identifikaci zdroje lze zařadit na jiné místo v URI, či odesílat jako součást hlaviček požadavku. Většinu zdrojů publikovaných v RESTových webových službách rozděluji na dva základní typy:[3]

- **Kolekce zdrojů** – Obsahují seznam jednoduchých zdrojů. Je jej možné dále filtrovat nebo přistupovat v hierarchii dále ke konkrétnímu zdroji.
- **Jednoduché zdroje** – Jeden konkrétní objekt, například prvek kolekce. Nemusí ale jít o dále nedělitelný objekt. I jednoduché zdroje mohou mít strukturu a skládat se z více zdrojů, k nimž je možné v hierarchii dále přistupovat.

Správně strukturovaná cesta ke zdroji poté může vypadat například takto:

```
/branches/west/employees/231/address
```

Pod touto adresou lze například očekávat adresu zaměstnance s číslem 231 ze západní pobočky společnosti. Na příkladu je možné také názorně vidět hierarchie. S každou další částí URI následuje buď zanoření do některé z kolekcí, či zanoření do struktury zdroje.

3. Obecně je možné zdroje dělit na mnohem více typů. Zajímavé dělení zdrojů je popsáno např. v práci *Modeling RESTful applications* Sylvie Schreier. [8]

4.1.2 Query string a fragment

Za cestou ke zdroji ještě mohou následovat dvě nepovinné části, tj. *query string*[4] a *fragment*.

Query string se skládá z několika pojmenovaných parametrů, jímž mohou být přiřazeny různé hodnoty. Pokud adresa ukazuje na zdroj, jímž je algoritmus (například vyhledávání, výpočet...), je možné jej těmito hodnotami konfigurovat. U kolekcí zdrojů může být query string použit pro filtrování či řazení kolekce. Doporučuji se také držet konvence, že query string nemá být určen ke změně jakýchkoliv dat.

Fragment je jednoduchý řetězec, který ukazuje na určité místo v identifikovaném zdroji. Použitím fragmentu v URI je možné dát důraz na některý prvek z kolekce, či některou část objektu. U webových stránek je tato část používána pro rychlý přechod na konkrétní místo v dokumentu. S použitím u RESTových služeb jsem se zatím nesetkal, nicméně je možné, že i u nich by bylo možné nalézt využití.

Příkladem API, které strukturu URI vůbec nevyužívá je například RESTové API služeb *Flickr*.[5] Namísto identifikátorů zdrojů adresa obsahuje metody a parametry připomínající RPC. Tento způsob se příčí myšlence ROA (architektuře orientované na zdroje), a je z pohledu architektonického stylu REST špatně. Kromě jiného porušuje také princip jednotného rozhraní a protokol HTTP používá jen velmi omezeně.

4.2 Autentizace

Architektonický styl REST nevytváří velká omezení pro autentizaci webové služby. Jediným z možných omezení je zmiňovaná bezstavovost (viz podsekci 1.3.3). Bezstavovost říká, že všechny autentizační informace musí klient odesílat při každém požadavku na server.

U webových stránek je toto často obcházeno pomocí mechanismu *cookies*. Klientská strana se vůči serveru autentizuje svými autentizačními údaji pouze jednou. Poté obdrží autentizační token, pomocí kterého se autentizuje dále. Při dalších požadavcích na server tedy klientská strana již nepředává kompletní autentizační údaje, ale pouze token. Tím vzniká na serveru stav, který označuje, že je klientská strana autentizována,

4. Česky možno přeložit jako *dotaz* nebo případně *řetězec s dotazem*. S těmito termíny jsem se ale v praxi nesetkal.

5. `http://www.flickr.com/services/api/request.rest.html`

což je v rozporu s požadavkem na bezstavovost. Použití cookies je také některými považováno za anti-pattern[6] RESTových služeb [9].

Protokol HTTP podporuje dvě základní metody autentizace: [10]

- **Basic access authentication** (v překladu *jednoduché ověření přístupu*) – klientský požadavek jako součást hlaviček posílá své autentizační údaje zakódované pomocí base64. Toto schéma samo o sobě neposkytuje žádné zabezpečení proti odposlechu hesla po cestě. [11]

- **Digest access authentication** – pokročilejší mechanismus, který odstraňuje nutnost odesílat autentizační údaje s každým požadavkem v nešifrované podobě. Klientský požadavek obsahuje MD5 hash autentizačních údajů a řetězce, který obdržel od serveru v rámci požadavku na autentizaci. Jedná se tedy o dvoukrokovou autentizaci typu *výzva-odpověď*.

Pro RESTové služby je výhodnější používat co nejjednodušší autentizaci, právě kvůli zmiňované bezstavovosti. Pro dobrou bezpečnost je ale potřeba použít dostatečně zabezpečený kanál, zajištěný například protokolem HTTPS.

Trendem v oblasti autentizace webových služeb je i autentizace třetí stranou, například za použití autentizačních protokolů *OpenID*[7], či *OAuth*[8]. [12] Právě protokol OAuth bývá často autentizačním mechanismem RESTových služeb.[9] [10] Nelze jej však považovat za čistě bezstavový, neboť je stále vyžadováno na serveru ukládat stav.

Omezení RESTových služeb ovšem není nutné dodržovat vždy dogmaticky. Pokud systém vyžaduje použití autentizačního protokolu, který není bezstavový, je potřeba udělat kompromis.

---

6. Pojem *anti-patten* je pravý opak návrhového vzoru (pattern). Označuje postupy, které nejsou doporučované buď pro svojí neefektivitu nebo nesprávnost.
7. `http://openid.net/`
8. `http://oauth.net/`
9. `https://dev.twitter.com/docs/api`
10. `http://developers.gigya.com/020_Developer_Guide/85_REST/OAuth2`

Velmi zajímavé řešení autentizace nabízí REST API služeb Amazon S3.[11] Pro autentizaci požadavků používá vlastní schéma založené na algoritmu HMAC.[12]

4.3 Hypermédia

Požadavek na použití hypermédií pro reprezentaci stavů aplikace (HATEOAS, viz podsekci 1.3.4) bývá v mnoha RESTových službách opomíjen, ačkoliv je podle mnohých velmi důležitý. [13] [14]

Problémem v této oblasti je nedostatečná standardizace hypermédií. Při použití XML je možné se držet značek používaných jazykem HTML, tedy značky: [13]

```
<link rel="vztah" target="cíl" />
```

Problém ale nastává u dalších formátů, jako je například JSON. U tohoto formátu zatím neexistuje standardní způsob, jak hyperlinky zapisovat. Některé implementace se snaží kopírovat zmíněnou značku `link`, jiné značí hypermédia odlišně. Vzhledem k tomu, že jedním ze základních požadavků RESTové architektury je jednota rozhraní, je tato záležitost problém.

Jedním z řešení je odesílat veškeré hyperlinky jako součást hlavičky, jak definuje RFC 5988. [15] Tento způsob sice je definován ve standardu, ale není dobře podporován ani příliš využíván. Jeho nedostatkem je také, že všechny hyperlinky jsou umístěny na jednom místě v odpovědi serveru. Pokud odpověď obsahuje více zdrojů či vnořené zdroje, může být problém identifikovat, které hyperlinky patří ke kterému zdroji.

Dalším řešením je nově popsaný standard *HAL (Hypertext Application Language)*[14], který popisuje jednotný způsob práce s hypermédii ve formátech JSON i XML. Ačkoliv, dle mého názoru, řeší problém pragmatickým a vhodným způsobem, není zatím příliš využíván. V praxi jsem se s tímto formátem zatím nesetkal.

---

11. `http://docs.amazonwebservices.com/AmazonS3/latest/dev/RESTAuthentication.html`
12. HMAC = *Hash-based Message Authentication Code*.
13. Více informací zde: `http://www.w3schools.com/tags/tag_link.asp`.
14. Specifikace zde: `http://stateless.co/hal_specification.html`

4.3.1 Dynamicky objevitelné rozhraní vs. WSDL

Pro jednotnou definici rozhraní webových služeb byl navržen jazyk *Web Services Description Language*, neboli *WSDL*. Je to jazyk založený na XML, který detailně popisuje všechny operace, které webová služba poskytuje. Jazyk WSDL je vhodný hlavně pro popis rozhraní webových služeb používajících SOAP, protože nepoužívá jednotné rozhraní definované standardem.

Pro RESTové webové služby není WSDL tak důležitý a použitelný. Ačkoli WSDL ve verzi 2.0 dokáže popisovat RESTové webové služby [16], REST nabízí jiné mechanismy pro orientaci v rozhraní webové služby. Rozhraní služeb je jednotné a poháněné hypermédii (*hypermedia-driven*), proto klientovi stačí znalost komunikačního protokolu a následování hyperlinků pro zjištění všech možností služby. Díky HATEOAS tak u RESTových služeb odpadá potřeba pro popis rozhraní.

4.4 Konzumace služeb

Jednou z velkých výhod RESTových služeb je jejich snadné použití z různých prostředí. Základním prostředkem RESTových služeb je totiž protokol HTTP, jehož podpora bývá jednou ze běžných součástí standardních knihoven programovacích jazyků.

- Na platformě .NET, tedy například v jazycích C# nebo VB.NET, je možné pro konzumaci využít tříd `HttpWebRequest`[15] a `HttpWebResponse`.[16]

- Z prostředí jazyka PHP existuje možnost využít rozšíření pro přístup ke knihovně cURL.[17] Tato multiplatformní knihovna je velmi dobře použitelná i z příkazové řádky nebo z jiných jazyků, např. C.[18]

15. `http://msdn.microsoft.com/library/system.net.httpwebrequest.aspx`
16. `http://msdn.microsoft.com/library/system.net.httpwebresponse.aspx`
17. `http://php.net/manual/en/book.curl.php`
18. Webové stránky projektu cURL: `http://curl.haxx.se/`.

- Konzumace RESTových služeb ze skriptů v jazyce JavaScript probíhá skrze objekt `XmlHttpRequest`,[19] při použití populárního frameworku jQuery přes funkci `jQuery.ajax()`.[20]

4.4.1 Frameworky pro konzumaci

Pro účely snadnější konzumace RESTových služeb je navrženo i mnoho frameworků. V následující podsekci odkazuji na 2 frameworky, které mě zaujaly.

Restfulie je velmi jednoduchý a minimalistický framework jak pro konzumaci tak pro tvorbu RESTových služeb. Je navržen původně pro jazyk Ruby, ale v současnosti nabízí i verze v jazyce C#, JavaScript, Python a Java. Jeho velkou předností je, že se snaží co nejlépe implementovat požadavky RESTové architektury. Zabývá se například podporou vyjednávání obsahu a požadavkem HATEOAS. Zvláštností je, že jde o jeden z mála nástrojů, které podporují hyperlinky předávané v hlavičce (viz sekci 4.3). [21]

Guzzle je prezentován jako HTTP klient pro PHP. Nabízí alternativu k výše zmíněné knihovně cURL. Vzhledem k tomu, že je jazyk PHP na webu velmi rozšířený, je často potřeba konzumovat webové služby. Guzzle nabízí velmi dobrou podporu protokolu HTTP/1.1 a umožňuje zasílat prakticky libovolné požadavky odpovídající specifikaci HTTP. Co se týče požadavků architektury REST, má například dobrou podporu pro caching (ukládání do mezipaměti, viz podsekci 1.3.6). Oproti výše zmíněnému Restfulie jde o mnohem komplexnější řešení. [22]

4.4.2 Problém Same-Origin Policy

V dnešní době, kdy se velká část aplikací přesouvá do webového prohlížeče, se může stát, že klientem webové služby je i skript běžící ve webovém prohlížeči. V takovém případě programátor pravděpodobně narazí na problém spojený se *Same-Origin Policy*.[23] Toto omezení bylo

19. `http://www.w3schools.com/xml/xml_http.asp`
20. `http://api.jquery.com/jQuery.ajax/`
21. Webové stránky projektu Restfulie: `http://restfulie.caelum.com.br/`.
22. Webové stránky projektu Guzzle: `http://guzzlephp.org/`.
23. Česky možno přeložit jako *politika stejného původu*.

zavedeno z bezpečnostních důvodů pro zamezení odesílání citlivých dat z prohlížeče na jinou doménu, než je právě otevřená webová stránka. Bohužel tím také aplikacím běžícím v prohlížeči odebírá možnost komunikovat s jinou webovou stránkou či službou. Není tak možné konzumovat webovou službu nacházející se na odlišné doméně, než právě otevřená webová stránka. [17]

Vzhledem k tomu, že dnešní webové standardy jsou stále ve vývoji, bylo zavedena nová technologie nazvaná *Cross-origin resource sharing (CORS)*.[24] CORS definuje způsob, jakým webový server může povolit přístup k jeho zdrojům z webové stránky v jiné doméně. CORS operuje pomocí speciální sady hlaviček, které musí nutně podporovat jak klient, tak server. [18] V současnosti je technologie CORS podporována většinou moderních prohlížečů.[25]

Alternativou k technologii CORS pro starší prohlížeče je mechanismus zvaný *JSONP*.[26] Tento způsob obchází omezení Same-Origin Policy následovně:

- Požadavek mimo doménu je odeslán skrze tag `<script>`, který se, podobně jako např. obrázky, vymyká omezení Same-Origin Policy.

- Stažená data odpovědi jsou poté obalena JavaScriptovou funkcí, kterou je možné zavolat a vrátit požadovaná data odpovědi.

Nevýhodou tohoto řešení je, že programátor nemá téměř žádnou kontrolu nad tím, jak odchozí HTTP požadavek vypadá. Odpadá tak například možnost použití vlastních hlaviček, či jakékoliv jiné metody, než **`GET`**. Ačkoliv se může zdát, že tento mechanismus je „nečisté“ řešení, je JSONP v praxi velmi používán. Samotná implementace RESTových služeb ve WCF například obsahuje podporu pro JSONP.[27]

---

24. Česky možno přeložit jako *sdílení zdrojů napříč doménami*.
25. Aktuální podpora CORS mezi webovými prohlížeči: `http://caniuse.com/cors`
26. `http://www.json-p.org/`
27. `http://msdn.microsoft.com/en-us/library/ee834511.aspx`

4.5 Ladění RESTových služeb

Zjišťování, proč program nefunguje tak, jak má, je jedním z klíčových aspektů vývoje software. Při práci s RESTovými službami a jejich klienty je může nastat mnoho, na první pohled špatně odhalitelných, problémů.

Protože komunikace RESTových služeb s klienty je relativně dobře čitelná, často se pro ladění těchto služeb používá odchytávání a analýza odesílaných zpráv. Pro usnadnění ladění webových služeb existují specializované nástroje, které umožňují monitorovat a případně měnit veškerou komunikaci mezi službou a klientem. Jedním z těchto nástrojů je např. *Fiddler – Web Debugging Proxy*.[28]

Fiddler umožňuje logovat veškerý síťový provoz probíhající skrze protokol HTTP. Mezi velmi užitečné funkce tohoto nástroje patří například schopnost odchytávat i šifrovanou komunikaci skrze HTTPS protokol či možnost pozastavit komunikaci použitím breakpointů. Fiddler obsahuje také velmi užitečný nástroj pro kompozici vlastních HTTP požadavků. Je tak možné testovat webovou službu i v případě, kdy pro ni zatím neexistuje klient.

28. `http://www.fiddler2.com/fiddler2/`

Kapitola 5

Implementace vlastní webové služby

Součástí této bakalářské práce je i implementace webové služby, která splňuje požadavky architektury REST. Jejím účelem je předvést, jak RESTová služba může fungovat v praxi. Při studiu implementace RESTových služeb ve WCF může být užitečné nahlížet do existující a funkční implementace, proto jsou zdrojové kódy celé služby k dispozici jako součást přílohy práce.

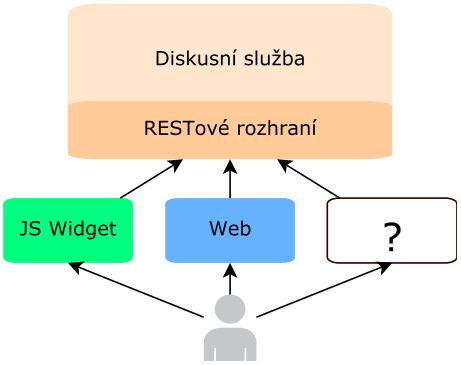
Aplikace potom může sloužit jako vzor, ukázka toho jak některé problémy vyřešit. Dále je také možné ji využít jako základ pro další aplikace, neboť její doménová logika je zcela oddělitelná od dalších součástí.

5.1 Základní funkčnost

Jako příklad webové služby jsem zvolil službu, která má na starost řízení diskuse. Prostřednictvím služby je možné vytvářet a komentovat diskusní témata. Službu je možné používat skrze JavaScriptový widget[1], který uživatel vloží na vlastní webovou stránku. K moderaci příspěvků a registraci uživatelů slouží přiložený web.

Hlavní výhodou služby ale je, že nabízí RESTové rozhraní, a proto je možné jednoduše doprogramovat další klienty. Veškeré operace, které provádí zmíněný web či JavaScriptový widget, je možné provést také přímo přes RESTové rozhraní služby. Samotný widget a web jsou dokonce pouze klienti služby, a předvádějí tak, jak je možné RESTové služby konzumovat z různých prostředí.

1. Malá jednoduchá aplikace vkládaná do webové stránky.

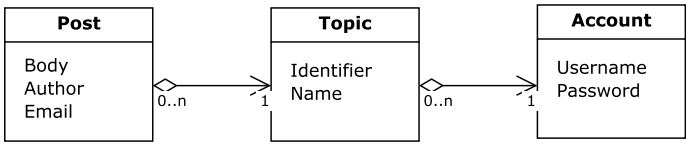


Obrázek 5.1: Znázornění vlastní služby a jejich klientů.

5.2 Doménová logika

Doménovou logiku služby jsem se snažil udržet co nejjednodušší. Důvodem je důraz hlavně na správnou implementaci RESTových požadavků, služba by poté byla také zbytečně komplexní a špatně uchopitelná pro někoho, kdo z ní bude potenciálně vycházet.

Službu tvoří tři základní entity: příspěvek, téma, účet. Jejich vztah a vlastnosti lze vidět na následujícím diagramu.



Obrázek 5.2: Entity-relationship diagram systému.

Část služby implementující doménovou logiku je rozdělena na tři moduly:[2]

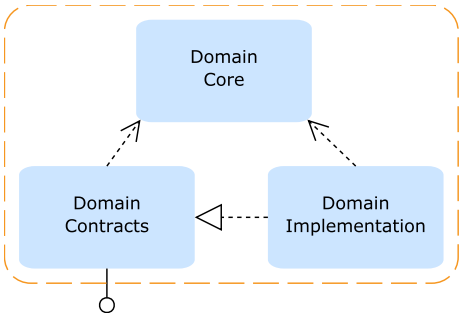
Domain.Core obsahuje bázové třídy, rozhraní pro další objekty a třídy pro komunikaci s databází. V této část se nachází základ pro entity, mapy entit, repositáře a validátory. Tato součást je znovupoužitelná v dalších službách.

Domain.Contracts obsahuje entity, které jsou základními jednotkami doménové logiky. Dále obsahuje rozhraní repositářů, které slouží

2. V rámci .NET platformy se jedná o jednotlivé assembly.

pro ukládání a manipulaci s entitami. Tato součást je nezávislá na konkrétním úložišti dat, slouží také jako hlavní rozhraní pro přístup k doménové logice.

Domain.Implementation obsahuje vlastní implementaci repositářů entit a validátorů. Jejím účelem je také mapovat entity do databázových tabulek.



Obrázek 5.3: Jednotlivé součásti doménové části

5.3 RESTové rozhraní

Nejdůležitější částí služby je implementace jejího RESTového rozhraní, proto se mu budu věnovat podrobněji.

RESTové rozhraní je rozděleno logicky na čtyři moduly:[3]

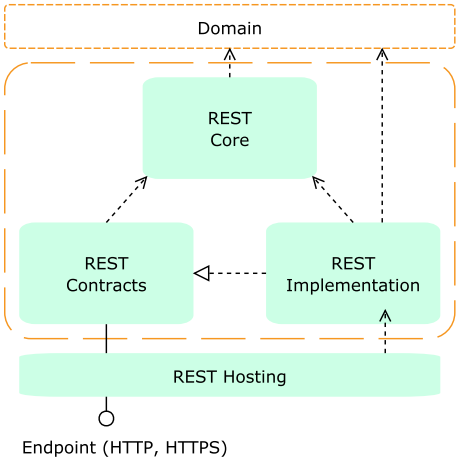
REST.Core obsahuje bázové třídy použité v dalších částech. Obsahuje společné výjimky, pomocné třídy a základ pro implementaci hypermédií. Patří sem implementace všech objektů, které jsou znovupoužitelné u jiných RESTových služeb a nezávislé na doménové logice.

REST.Contracts obsahuje popis rozhraní služby a popis datových typů, které služba používá. Z pohledu WCF obsahuje *service contract* a *data contracts*.

3. V rámci .NET platformy se jedná o jednotlivé assembly.

REST.Implementation obsahuje samotnou implementaci služby. Ta využívá modulu *Domain.Contract* pro obsluhu aplikační logiky. Kromě implementace samotného rozhraní služby popisuje například správu chyb celé služby.

REST.Hosting popisuje všechny koncové body služby a konfiguruje jejich chování. Má za úkol také propojit veškeré kontrakty s jejich implementacemi a řídit vytváření instancí služby.



Obrázek 5.4: Součásti RESTového rozhraní.

5.3.1 Druhy zdrojů

Při návrhu služby jsem vytvořil několik kategorií zdrojů, které se liší významem a svým použitím pro vstupní a výstupní datové typy v RESTovém rozhraní. Těmito jsou:

Obyčejný (common) zdroj – reprezentuje zdroj, který se vyskytuje na různých místech systému a nepopisuje přitom žádné entity domény. Implementacemi tohoto typu zdroje jsou například zdroje obsahující uživatelské zprávy nebo zdroje obsahující chybovou hlášku.

Kolekce zdrojů – reprezentuje seznam (kolekci) dalších zdrojů.

Primární reprezentace zdroje – reprezentuje jednu samostatnou entitu domény. Neobsahuje žádné další vnořené či navázané zdroje. Primární reprezentace je použita například jako výsledek operací `POST` nebo `PUT`, protože neobsahuje žádná zbytečná data navíc.

Rozšířená reprezentace zdroje – reprezentuje entitu domény včetně některých jejích vazeb. V praxi jde například o diskusní téma včetně příspěvků uvnitř. Rozšířená reprezentace je použita často jako výsledek operací `GET`.

Formulář – používá se při zakládání nebo úpravě zdrojů. Formulář je velmi podobný primárnímu zdroji s tím rozdílem, že má všechny údaje prázdné. Formulář se ale od primárního zdroje může lišit. Například primární reprezentace účtu neobsahuje heslo, zatímco formulář pro vytvoření/editaci účtu ano.

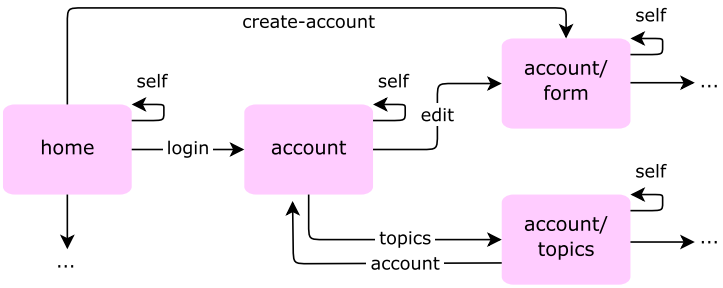
5.3.2 Implementace hypermédií

Při implementaci jsem se snažil dodržet požadavek HATEOAS (viz podsekci 1.3.4) a rozhraní služby je tak *objevitelné skrze hypermédia*. Mezi jednotlivými zdroji a operacemi lze přecházet pomocí hyperlinků. Vzhledem k tomu, že jde o požadavek RESTové architektury, který WCF neimplementuje, ukazuji v práci vlastní řešení. Ke každému zdroji je připojen údaj `links`, který obsahuje kolekci hyperlinků reprezentujících přechody k jiným částem služby. Jednotlivé odkazy jsou složené ze samotné adresy (údaj `href`) a vztahu ke konkrétnímu zdroji (údaj `rel`).

Na diagramu 5.5 lze vidět jednotlivé zdroje a přechody mezi nimi. Každá šipka značí hyperlink, její popisek je vztah hyperlinku ke konktétnímu zdroji.

5.3.3 Bezstavovost a vyjednávání obsahu

Diskusní služba je navržená jako čistě bezstavová. Také nepoužívá autentizační protokol vyžadující uložení stavu na serveru. Všechny požadavky jsou tedy na sobě nezávislé. Podporuje také více vláken a je tak teoreticky velmi dobře škálovatelná. Pro podporu více vláken bylo nutné zajistit správný přístup k databázi a věnovat větší pozornost obsluze sdílených objektů.



Obrázek 5.5: Část mapy hypermédií systému.

Služba nabízí i podporu více formátů zdrojů, což je možné označit jako triviální splnění RESTového požadavku pro vyjednávání obsahu. Podporované formáty služby jsou JSON a XML. Jejich výběr se řídí podle hlaviček `Accept` a `Content-Type`.

5.3.4 Pomocné třídy

V bakalářské práci jsem navrhl také několik netriviálních pomocných tříd, které usnadňují programování služby a odstraňují duplikaci kódu. Tyto třídy řeší například:

Obalení zpracováním chyb
: Každá operace je automaticky obalena zpracováním chyb, které je konfigurovatelné. Chyby, které v aplikaci vzniknou jsou odeslány klientovi v jednoduchém objektu obsahujícím údaje o chybě. V případě, že by toto implementováno nebylo, není klient schopný v případě chyby zjistit téměř žádné informace, například jaká chyba nastala.

Obalení zpracování požadavku transakcí
: Služba nabízí jednoduchý způsob, jak jednotlivé operace služby obalit databázovou transakcí. Zabrání to zbytečné duplikaci kódu pro obsluhu transakce a zajistí atomicitu změn.

Vynucení autentizace či zabezpečeného protokolu
: Některé operace vyžadují zabezpečený protokol (HTTPS) či přístup pouze pro ověřené uživatele. Tyto operace je možné jednoduše označit (pomocí atributu) a služba zajistí odmítnutí všech požadavků nezabezpečenou cestou či anonymními uživateli.

5.3.5 Podpora technologií

V této sekci popíši některé dodatečné technologie, které jsou ve službě implementovány či zprovozněny.

Podpora HTTP autentizace
Součástí práce je vlastní jednoduchá implementace podpory pro *Basic autentizaci* HTTP protokolu (viz sekci 4.2). Tento způsob autentizace je ve službě využíván pro veškeré operace vyžadující autorizovaný přístup.

Podpora pro Cross-origin resource sharing (CORS)
Služba podporuje technologii CORS pro podporu požadavků z webových prohlížečů skrz doménu. Tento způsob je použit i u JavaScriptového widgetu u prohlížečů, které CORS podporují.[4]

Podpora pro JSONP
Služba podporuje formát JSONP pro požadavky napříč doménou jako alternativu k technologii CORS pro starší prohlížeče. Tato funkčnost je implementována i na straně klienta – JavaScriptového widgetu. Podpora pro JSONP je součástí samotného WCF, služba tuto implementaci používá a rozšiřuje rozpoznání požadavků na JSONP pro větší kompatibilitu se současnými prohlížeči.

Podpora pro přepis metod
Někteří klienti mohou být omezení výběrem HTTP metod, proto služba nabízí způsob, jak toto vyřešit. Klient odešle požadavek metodou `GET` a v hlavičce či jako součást URI odešle skutečnou metodu, kterou chce použít. Poté je zajištěn výběr správné operace bez nutnosti rozšiřovat kontrakt služby.

5.4 Klienti

Pro praktickou demonstraci konzumace RESTové služby jsem připravil dva klienty služby: web a JavaScriptový widget. Tito ukazují přístup ze dvou naprosto odlišných prostředí.

4. Kód vychází z: `https://github.com/dhvik/Wcf-CORS-behavior`

5.4.1 Web

Ke službě jsem připojil jednoduchý web, který je napojen na službu skrze RESTové rozhraní. Ačkoliv by bylo možné web implementovat přímo jako součást služby, rozhodl jsem se jej naprogramovat jako odděleného klienta služby. Důvodem je ukázka způsobu, kterým používat RESTové rozhraní z prostředí jazyka C#. Web je naprogramován s použitím technologie ASP.NET MVC 3. Účel webového klienta je informovat o tom, jak vkládat diskusi na webové stránky a popsat podrobněji rozhraní služby. Dále umožňuje registraci uživatelských účtů a správu všech diskusí.

5.4.2 JavaScriptový widget

JavaScriptový widget je klient služby běžící ve webovém prohlížeči. Ukazuje, jak prakticky překonat některá omezení webových prohlížečů za pomocí CORS a JSONP (viz podsekci 4.4.2). Widget využívá pouze malou část služby, a to výpis tématu vč. příspěvků a přidání příspěvku k tématu. Nicméně lze na něm vidět, jak jednoduché je používat službu s použitím JavaScriptového frameworku *jQuery*.[5]

5.5 Požadavky pro nasazení služby

Služba je naprogramována pro platformu .NET Framework verze 4. Je připravena pro hosting na webovém serveru IIS verze 7.5. Služba využívá modulu *URL Rewrite*[6], ale tento modul není povinný a služba funguje i bez něj. Virtuální adresář služby v IIS musí mít zprovozněné vazby HTTP i HTTPS.

Služba také vyžaduje přístup k databázi, která je generována automaticky díky použití technologie *Entity Framework Code First*.[7] Při vývoji služby byl pro databázi použit server *Microsoft SQL Server 2008 R2*. Vzhledem k tomu, že se použitý Entity Framework chová i jako vrstva abstrakce nad databází (DAL)[8], je možné použít i jiné databázové servery.

---

5. Více informací zde: `http://jquery.com/`.
6. Více informací zde: `http://www.iis.net/download/urlrewrite`.
7. Více informací zde: `http://msdn.microsoft.com/en-us/data/ee712907`.
8. DAL = *Database abstraction layer*.

Kapitola 6

Příprava materiálů pro výuku

Součástí této práce jsou i materiály pro přednášku a cvičení, které mohou být použity pro výuku tématu vývoje RESTových služeb na platformě .NET.

K práci přikládám slidy pokrývající stručně zadané téma. V první části přednášky ukazuji na příkladech rozdíly oproti SOAP a představuji samotný architektonický styl. Dále už se zabývám základy RESTových služeb ve WCF. Slidy jsou doplněny o dokument s poznámkami pro přednášku.

Pro cvičení z předmětu je potom připravena samostatná úloha, ve které si studenti mohou osvojit některé z principů vývoje RESTových služeb v praxi. Součástí přílohy je zadání úlohy, její řešení a doprovodný text popisující podrobnosti úlohy.

Materiály jsou určeny pro studenty, kteří už mají s vývojem na platformě .NET zkušenosti. Předpokládám jejich základní znalost WCF, webových služeb a základy protokolu HTTP. Dodané studijní materiály bohužel neprobírají látku dostatečně do hloubky, nezabývají se například implementací všech omezení architektonického stylu REST, nebo samotnou konfigurací a přípravou pro hosting.

Jako studijní materiál pro studenty s hlubším zájmem o téma RESTových služeb je možné odkázat na tuto práci a přiloženou vzorovou aplikaci.

Závěr

Práce seznamuje čtenáře s architektonickým stylem REST a jeho možnostmi využití pro vývoj v oblasti webových služeb. Požadavky a omezení architektonického stylu jsou popsány teoreticky a následně uvedeny do praktických souvislostí. Věnuji se i protokolu HTTP jakožto nástroji RESTových služeb a srovnání principů RESTu s protokolem SOAP.

REST je představen v kontextu aktuálních technologií, v kontextu platformy .NET a jeho implementace WCF. V kapitole o WCF se podrobněji věnuji jednotlivým třídám a podrobnostem souvisejícím s vývojem pro .NET Framework. V závěru také odkazuji na několik alternativních řešení pro tuto platformu.

V samostatné kapitole se poté věnuji problémům a doporučením týkajících se obecně návrhu RESTových služeb. Dále popisuji, jak RESTové služby ladit a konzumovat. Odkazuji na několik knihoven určených pro konzumaci RESTových služeb pro různé platformy. Věnuji se také problémům při komunikaci RESTových služeb s aplikacemi běžícími ve webovém prohlížeči.

V praktické části poté představuji implementaci vlastní RESTové služby určenou pro řízení diskuse. Popisuji v ní architekturu a součásti projektu a dále vlastní řešení některých častých problémů. Součástí mé práce je také příprava studijních materiálů, které k práci přikládám.

V práci jsem se obecně zaměřoval hlavně na uvedení RESTu do praxe. Věnuji se několika důležitým a sporným otázkám, které mohou zajímat vývojáře a další zájemce o tento architektonický styl. Architektura REST je v mnoha ohledech nejasná a v informatické společnosti se okolo ní objevuje mnoho různých názorů a mýtů. V práci jsem se snažil pojmout co největší množství informací a nabídnout vlastní výklad architektury. REST API jsou díky své flexibilitě a jednoduchosti stále více používané a rozšířené. Podle mého názoru má tento styl budoucnost u nezávislých projektů i v komerční sféře.

Literatura

[1] World Wide Web Consortium. Web Services Architecture. [online; cit. 26. 2. 2012] <http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/>.

[2] Aaron Skonnard. A Developer's Introduction to Windows Communication Foundation 4. Technical report. [online; cit. 14. 11. 2011] <http://msdn.microsoft.com/en-us/library/ee354381.aspx>.

[3] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. RFC 2616, hypertext transfer protocol – HTTP/1.1. [online; cit. 24. 2. 2012] <http://www.rfc.net/rfc2616.html>.

[4] Roy T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

[5] M. Mealling, R. Denenberg, and W3C URI Interest Group. Report from the joint W3C/IETF URI planning interest group: Uniform resource identifiers (URIs), URLs, and uniform resource names (URNs), 2002. [online; cit. 7. 4. 2012] <http://tools.ietf.org/html/rfc3305>.

[6] Microsoft. What's New in Windows Communication Foundation 4.5. [online; cit. 7. 4. 2012] <http://msdn.microsoft.com/en-us/library/dd456789.aspx>.

[7] T. Berners-Lee, W3C/MIT, R. Fielding, Day Software, L. Masinter, and Adobe Systems. Report from the joint W3C/IETF URI planning interest group: Uniform resource identifiers (URIs), URLs, and uniform resource names (URNs), 2002. [online; cit. 7. 4. 2012] <http://tools.ietf.org/html/rfc3986>.

[8] S. Schreier. Modeling RESTful applications. In *Proceedings of the Second International Workshop on RESTful Design*, WS-REST '11, pages 15–21, New York, NY, USA, 2011. ACM.

[9] S. Tilkov. REST Anti-Patterns. *InfoQ*, (Jul 02), 2008. [online; cit. 16. 4. 2012] <http://www.infoq.com/articles/rest-anti-patterns>.

[10] Northwestern University, P. Hallam-Baker, Verisign, Inc., J. Hostetler, AbiSource, Inc., S. Lawrence, Agranat Systems, Inc., P. Leach, Microsoft Corporation, A. Luotonen, Netscape Communications Corporation, L. Stewart, and Open Market, Inc. HTTP Authentication: Basic and Digest Access Authentication, 1999. [online; cit. 16. 4. 2012] <http://tools.ietf.org/html/rfc2617>.

[11] J. Měcháček. Autentizace na webu. *Zpravodaj ÚVT MU*, (roč. X, č. 2), 1999. [online; cit. 16. 4. 2012] <http://www.ics.muni.cz/bulletin/articles/173.html>.

[12] A. Powell and D. Recordon. OpenID: Decentralised Single Sign-on for the Web. *Ariadne*, (Issue 51), 2007. [online; cit. 16. 4. 2012] <http://www.ariadne.ac.uk/issue51/powell-recordon/>.

[13] Roy T. Fielding. REST APIs must be hypertext-driven. *Untangled*, (October 20), 2008. [online; cit. 14. 11. 2011] <http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven>.

[14] Martin Fowler. Richardson Maturity Model: steps toward the glory of REST. 2010. [online; cit. 14. 11. 2011] <http://martinfowler.com/articles/richardsonMaturityModel.html>.

[15] M. Nottingham. Web linking, 2010. [online; cit. 13. 5. 2012] <http://tools.ietf.org/html/rfc5988>.

[16] L. Mandel. Describe REST web services with WSDL 2.0, 2008. [online; cit. 7. 4. 2012] <http://www.ibm.com/developerworks/webservices/library/ws-restwsdl/>.

[17] M. Zalewski and Google Inc. Browser security handbook, part 2. 2009. [online; cit. 17. 5. 2012] <http://code.google.com/p/browsersec/wiki/Part2>.

[18] World Wide Web Consortium. Cross-Origin Resource Sharing, W3C Working Draft 3. 2012. [online; cit. 17. 5. 2012] <http://www.w3.org/TR/2012/WD-cors-20120403/>.

Přílohy

Následuje seznam adresářů na přiloženém CD:

- `discussion-service` – obsahuje zdrojové kódy diskusní služby ve formě solution pro Microsoft Visual Studio 2010;
- `slidy` – obsahuje slidy pro přednášku k tomuto tématu jako součást výukových materiálů (ve formátech LaTeX a PDF);
- `slidy-text` – obsahuje poznámky k přednášce jako doprovod ke slidům (ve formátech LaTeX a PDF);
- `priklad-text` – obsahuje poznámky pro příklad ke cvičení jako součást výukových materiálů (ve formátech LaTeX a PDF);
- `priklad-zadani` – obsahuje zdrojové kódy zadání příkladu ke cvičení ve formě solution pro Microsoft Visual Studio 2010;
- `priklad-reseni` – obsahuje zdrojové kódy řešení příkladu ke cvičení ve formě solution pro Microsoft Visual Studio 2010;
- `text` – obsahuje tento text práce v elektronické podobě (ve formátech LaTeX a PDF) včetně všech obrázků a použitých bibliografických záznamů.