Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

Automatizované publikování SQL dotazů ve formě webových služeb

Plzeň 2013 František Schneider

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 22. června 2013

František Schneider

Abstract

Web services are currently one of the most popular and useful technology that every IT professional should be aware of. The main purpose of this master thesis is describing everyting important about web services and related standards. Technologies of web services, SQL and auto-generated code are in this thesis connected and together create a tool that can publish SQL queries into the web service form.

Obsah

1 Úvod

Webové služby jsou v dnešní době jednou z nejpoužívanějších a nejdůležitějších technologií při integraci systémů. Oblasti použití webových služeb se neustále rozšiřují a představují velký potenciál nejen pro podniky a organizace, které se zabývají integrací systémů a podnikových procesů. Webové služby lze použít díky jejich robustnosti takřka na vše a vzhledem k trendu přesouvání aplikací z lokálního úložiště na Internet je to technologie, která bude používána ve stále větším měřítku.

Aktuální stav

Centrum informatizace a výpočetní techniky (CIV) se stará o provoz a rozvoj informačního systému studijní agendy na půdě Západočeské univerzity v Plzni. V současnosti rovněž poskytuje velké množství webových služeb běžících nad informačním systémem IS/STAG. Typicky se jedná např. o služby poskytující seznam studentů či zaměstnanců dané katedry, seznam diplomových prací nebo služby umožňující hromadný zápis známek pro nějaký předmět.

Cíl práce

Cílem této práce je seznámení se s webovými službami obecně a technologiemi, které jsou se službami spojeny. Dalším cílem je rozšíření současné aplikace „Webové služby nad IS/STAG“ o modul, který dokáže publikovat SQL dotazy jako webové služby. Tento modul by měl ulehčit práci programátorům i administrátorům IS/STAG a urychlit proces vytváření nových služeb. Výsledná realizovaná aplikace by měla být otestována nad provozní databází IS/STAG a připravena k reálnému provozu.

Stručný přehled

První a druhá kapitola slouží jako úvod do celé problematiky webových služeb a seznamuje čtenáře se základními pojmy, jako jsou XML, XSD a vzdálené volání procedur.

Ve třetí kapitole je už detailněji popsán význam webových služeb a protokoly, které jsou s webovými službami spojené.

První část čtvrté kapitoly popisuje aplikaci „Webové služby nad IS/STAG“ a dává do kontextu vztah SQL dotazu a webové služby. Druhá část kapitoly slouží jako zjednodušený návod pro ruční tvorbu webové služby, kterou lze spustit nad informačním systémem IS/STAG.

V páté kapitole jsou rozebrány nástroje a metody používané pro strojové generování zdrojových kódů. Rovněž jsou zde zhodnoceny jejich vlastnosti, klady a zápory.

Šestá kapitola popisuje implementaci webové aplikace, která dokáže SQL dotazy zpracovávat a poté publikovat jako webové služby.

Závěrečná kapitola obsahuje shrnutí diplomové práce a reflexi přínosu aplikace generující webové služby.

2 Úvod do XML

XML (eXtensible Markup Language) je dnes všeobecně známý pojem a v kontextu webových služeb představuje základní kámen pro standardy, na kterých jsou webové služby postaveny.

XML je rozšiřitelný značkovací jazyk a pochází z oblasti zaměřené na uchování a zpracování textových dokumentů[2]. Hlavní výhoda XML spočívá v jeho platformní nezávislosti. Význam i interpretace XML dokumentu bude všude stejná, at' už je dokument zpracováván na UNIX systému, či Windows.

2.1 XML dokument

XML dokument je textový soubor, pro který platí určitá pravidla. Nejdůležitější vlastností je požadavek na správnou strukturovanost (tzv. "well-formed").

Vlastnosti well-formed dokumentu:

- Dokument musí obsahovat pouze jeden kořenový (root) element.
- Jednotlivé značky mohou být zanořené, ale nesmí se křížit.
- Neprázdné elementy jsou definovány startovací a ukončovací značkou.
- Hodnoty atributů musí být uvedeny buď v jednoduchých, nebo složených uvozovkách. V rámci elementu musí být uvozovky stejného typu a nelze je kombinovat.

U značek se rozlišují velikosti písmen (je tzv. case sensitive), což znamená, že např. značka `<jmeno>` je odlišná od `<Jmeno>` i `<JMENO>`.

2.2 XML Schema Definition

XSD (XML Schema Definition) schéma je dokument, který popisuje formální strukturu XML dokumentu. XSD schéma je rovněž XML dokument. Výhoda

použití schémat spočívá v tom, že jejich interpretace je jednoznačná. Nejčastěji jsou XML schémata používána pro validaci, což je proces, při kterém se ověřuje, zda konkrétní XML dokument vyhovuje všem restrikcím definovaných ve schématu.

Listing 2.1: Příklad XML dokumentu s připojeným schématem.

```
<?xml version="1.0" encoding="UTF-8"?>
<uzivatel
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="uzivatel.xsd" >
    <jmeno>Jan</jmeno>
    <prijmeni>Sluzebnik</prijmeni>
    <role>Admin</role>
    <pocet>45</pocet>
 </uzivatel>
```

Listing 2.2: Příklad XSD schématu

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="uzivatel">
 <xs:complexType>
   <xs:sequence>
     <xs:element name="jmeno" type="xs:string"></xs:element>
     <xs:element name="prijmeni" type="xs:string"></xs:element>
     <xs:element name="role" type="xs:string"></xs:element>
     <xs:element name="pocet" type="xs:integer"></xs:element>
   </xs:sequence>
 </xs:complexType>
</xs:element>
</xs:schema>
```

3 Vzdálené volání procedur

Pro pochopení kontextu a významu webových služeb je nutné vysvětlit pojem **RPC – Remote Procedure Call**. Obecně se tímto pojmem označuje mechanismus, který umožňuje vzdálené volání procedur (metod) a funkcí. Jinak řečeno, je to technologie dovolující programu vykonat proceduru, která je uložena na jiném místě než samotný program. Typickým příkladem může být výpočet nějaké funkce na jiném počítači v síti.

Hlavní nevýhoda RPC spočívá v tom, že pro přenos a volání lze použít pouze základní primitivní datové typy. U komplexnějších typů je už nutné použít XML-RPC (viz sekce 3.1) nebo jeho nástupce protokol SOAP (viz sekce 4.2).

3.1 XML Remote Procedure Call

XML-RPC (XML Remote Procedure Call) je označení pro protokol pro vzdálené volání a přenos dat, která jsou ve formátu XML.

Základem XML-RPC je HTTP (HyperText Transfer Protocol) protokol, který umožňuje posílání dat pomocí metody POST na libovolný server. Ten na základě volání vykoná požadovanou akci (proceduru) a pomocí stejného mechanismu, jakým byla zpráva odeslána od klienta, pošle odpověď zpět. Server vrací buď konkrétní odpověď (zprávu), nebo informaci o chybě. Každá zpráva, pomocí které klient a server komunikují, se skládá z hlavičky a těla.

3.1.1 Hlavička požadavku

V hlavičce je na první řádce uvedena dotazovací metoda (v případě XML-RPC vždy POST), umístění XML-RPC serveru (může být třeba jen lomítko nebo prázdné), verze a druh protokolu. `User-Agent` identifikuje klienta, `Host` definuje adresu serveru v síti, `Content-Type` značí typ odesílaných dat – v tomto případě se musí jednat vždy o `text/xml`. `Content-Length` udává délku dokumentu, která musí být správně určena.

3.1.2 Tělo požadavku

Tělo požadavku je uvozeno elementem `<methodCall>`. Uvnitř tohoto elementu se nachází `<methodName>` obsahující název volané metody. Pokud volaná metoda vyžaduje nějaké parametry, je nutné tyto parametry uvést v elementu `<params>`. Jednotlivé parametry a jejich hodnoty jsou uvedeny jako obsah elementů `<param>` a `<value>`. Uvnitř elementu `<value>` lze volitelně uvést i datový typ. Ten se uvádí často pro lepší přehlednost, čitelnost a k rychlejšímu odhalování případných chyb.

Listing 3.1: Příklad vzdáleného volání pomocí XML-RPC

```
POST /CalcServer HTTP/1.0
User-Agent: myXMLRPCClient/1.0
Host: 192.168.1.2
Content-Type: text/xml
Content-Length: pocetZnaku

<?xml version="1.0"?>
       <methodCall>
         <methodName> SoucetClass.getSoucet </methodName>
              <params>
              <param>
                     <value> <int> 3 </int> </value>
              </param>
              <param>
                     <value> <float> 4.1 </float> </value>
              </param>
         </params>
       </methodCall>
</xml>
```

3.1.3 Hlavička odpovědi

V hlavičce odpovědi je na prvním řádku verze protokolu (HTTP/1.1) se stavovým kódem. Všechny XML-RPC odpovědi mají tvar `200 OK`[1] i v případě, že zpráva obsahuje chybovou hlášku. Řádek obsahující `Connection` s hodnotou `close` znamená, že klient musí pro další komunikaci znovu navázat nové spojení se serverem. `Content-Length` a `Content-Type` mají stejný význam jako v hlavičce požadavku. Řádek s položkou `Date` obsahuje informaci o tom, kdy byla odeslána odpověď.

3.1.4 Tělo odpovědi

Tělo XML-RPC odpovědi tvoří XML s kořenovým elementem `<methodResponse>`. Na rozdíl od těla požadavku zde musí být element `<params>` uveden vždy. Uvnitř musí taktéž existovat alespoň jeden element `<param>` obsahující výsledek volání.

Listing 3.2: Výsledek volání XML-RPC

```
HTTP/1.1 200 OK
Connection: close
Content-Length: pocetZnaku
Content-Type: text/xml
Date: Fri, 1 May 2013 16:31:06 GMT
Server: identifikace_serveru

<?xml version="1.0"?>
 <methodResponse>
   <params>
         <param>
               <value> <float> 7.1 </float> </value>
         </param>
    </params>
 </methodResponse>
</xml>
```

[1] Existuje výjimka v případě, že server je napsán např. v jazyce PHP, který vrací jinou hodnotu.

3.2 JSON Remote procedure Call

JSON (JavaScript Object Notation, Javascriptový objektový zápis) je odlehčený formát pro výměnu dat. Je dobře čitelný pro člověka, ale je vhodný i ke strojovému zpracování. Tento formát není závislý na žádném z programovacích jazyků, ale používá konvence, které jsou běžné např. v jazyce C, C++, Java nebo JavaScript. V poslední době nastal velký rozmach tohoto formátu a je považován za nástupce XML.

Hlavní výhoda oproti XML je jeho odlehčenost. XML formát je strukturovaný a hodně popisný. Při komunikaci, kdy je nutné hlídat a zajistit co nejmenší množství přenesených dat, je JSON vždy úspornější (např. data pro mobilní platformy).

JSON je tvořen dvěma základními strukturami:

- Kolekce struktur definovaných pomocí klíče a hodnoty. V různých programovacích jazycích je tato struktura realizována jako mapa (`HashMap`), záznam (`Struct`) nebo asociované pole.
- Uspořádaný seznam. V programovacích jazycích je realizován např. jako pole (`array`), vektor nebo seznam (`List`).

Pro použití JSON formátu byl vytvořen protokol **JSON-RPC**, který je podobný protokolu XML-RPC(viz 3.1), ale je mnohem jednodušší, a definuje pouze datové typy, názvy volaných metod a argumenty volání. Výhoda JSON-RPC spočívá i v tom, že umožňuje informovat volaný server, že klient nepožaduje zpracované výsledky ve stejném pořadí, v jakém je poslal (v případě vícenásobných požadavků), a rovněž může informovat server o tom, že nepožaduje odpověď žádnou.

Vzdálená metoda je opět volána prostřednictvím protokolu HTTP podobně jako v XML-RPC. Na rozdíl od XML-RPC je v hlavičce požadavku i odpovědi `Content-Type` hodnota nastavena na `application/json-rpc`.

3.2.1 Formát požadavku

Všechny přenášené zprávy směrem k serveru jsou serializované do podoby JSON formátu, ve kterém musí být uvedeny následující atributy:

- `method` – Textový řetězec s názvem metody.
- `params` – Pole objektů reprezentující argumenty volané metody.
- `id` – Identifikátor zprávy/klienta (může být cokoliv). Slouží ke spárování požadavku s odpovědí. Pokud je `id` nastaveno na `null`, pak to znamená, že klient neočekává žádnou odpověď.

3.2.2 Formát odpovědi

Server odpovídá na požadavky opět zprávami v JSON formátu obsahující tyto atributy:

- `result` – Obsahuje výsledek volání. V případě chyby musí být `null`.
- `error` – V případě výskytu chyby obsahuje číslo chyby, jinak `null`.
- `id` – Číslo požadavku, na který je posílána odpověď.

Následující příklad ukazuje výsledek volání vzdálené procedury ve formátu JSON.

Listing 3.3: Výsledek volání ve formátu JSON.

```
HTTP/1.1 200 OK
Connection: close
Content-Length: pocetZnaku
Content-Type: text/xml
Date: Fri, 1 May 2013 16:31:06 GMT
Server: identifikace_serveru

 {"result": "7.1", "error": null, "id":1}
```

4 Webové služby

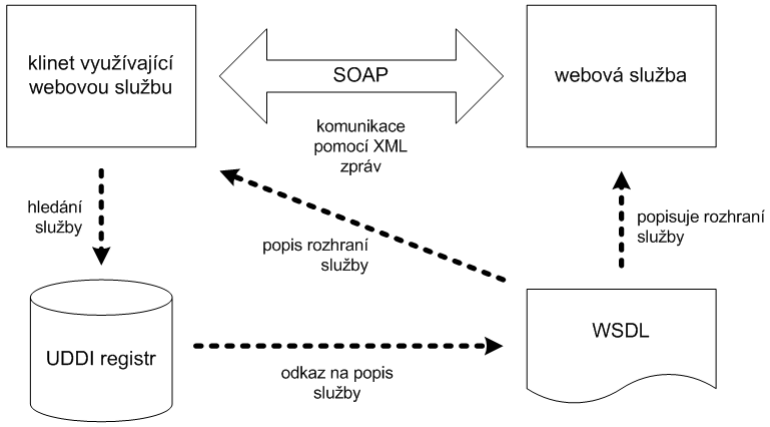
Existuje několik definicí termínu **Webová služba** (Web Service) a každá organizace či technický dokument jej definuje po svém. Je nutné rozlišovat pojem **Web Service** a **web service**. Obojí v překladu znamenají webová služba, ale pouze v prvním případě se jedná o technologii, kterou se snaží tato diplomová práce popsat. Pro potřeby této diplomové práce bude všude uváděn český pojem s malými písmeny, ale označující technologii **Web Service**.

Nejčastější definice říká, že webová služba je software poskytující nějakou funkcionalitu (službu) přístupnou skrze internet pomocí standardizovaného přenosu XML zpráv. Například webový klient vyvolá webovou službu posláním zprávy ve formátu XML a poté čeká na odpověď rovněž ve formátu XML.

Webové služby jsou soběstačné, modulární, distribuované a dynamické aplikace. Mohou být popsány, publikovány a vyvolávány skrze síť, aby produkovaly data, procesy a různé podpůrné prostředky. Webové služby pracují nad standardy, jako jsou TCP/IP, HTTP, JSON a XML.

4.1 Protokoly webových služeb

Jádro webových služeb tvoří tři základní technologie (někdy označováno jako komponenty): **SOAP** (Simple Object Access Protocol), **UDDI** (Universal Description, Discovery and Integration) a **WSDL** (Web Services Description Language).



Obrázek 4.1: Vazby mezi komponentami. Převzato z [9]

.

Základní vlastnosti webových služeb:

- **Jsou orientované na XML.**
  Použitím jazyka XML jako formátu pro výměnu dat jsou webové služby zbaveny závislosti na konkrétních výpočetních platformách i programovacích jazycích.

- **Jsou volně vázané.**
  To znamená, že je lze provázat s dalšími systémy a nadále jednoduše spravovat. Tato vlastnost je v dnešní době jednou z nejdůležitějších právě při integraci informačních systémů.

- **Nejsou jemně strukturované.**
  Pro využívání služeb na enterprise úrovni je volání služeb pomocí RPC příliš jednoduché a neefektivní, proto jsou webové služby poskytovány jako celek, nebo jen konkrétní webové metody.

- **Podporují synchronní i asynchronní volání.**
  Pro synchronní volání platí, že např. webový klient čeká na výsledek volání. U asynchronního klient zavolá danou webovou metodu (službu), ale nečeká na výsledek, a pokračuje dál v činnosti.

4.2 Simple Object Acces Protocol

SOAP (Simple Object Access Protocol) protokol specifikuje způsob výměny strukturovaných dat v implementaci webových služeb. Účelem SOAP protokolu je zasílání XML zpráv od odesílatele k příjemci. Navazuje na protokol XML-RPC.

Další vlastnosti SOAP protokolu:

- je to komunikační protokol.
- definuje formát odesílaných zpráv.
- je založen na XML (platformní a jazyková nezávislost).

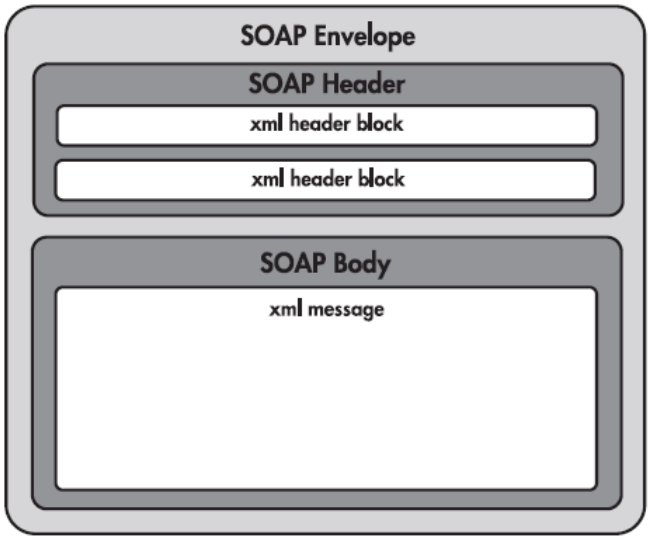
4.2.1 Základní elementy

SOAP zprávy jsou XML dokumenty, které obsahují následující emelenty:

- `<Envelope>`
- `<Header>`
- `<Body>`
- `<Fault>`

Elementy `<Envelope>` a `<Body>` jsou povinné. `<Header>` a `<Fault>` jsou dobrovolné a záleží na konkrétním příkladu, zda je vhodné je využít, či ne.

Element `<Envelope>` určuje začátek a konec zprávy, takže příjemce ví, kdy už přijal celou zprávu, a může ji začít zpracovávat. Platí pravidlo, že každý element `<Envelope>` smí obsahovat pouze jeden element `<Body>`. Element `<Body>` musí být přítomen vždy jako přímý potomek elementu `<Envelope>` a obsahuje tzv. `Application-specific data` – to jsou informace, které jsou zpracovávány webovou službou. Tento element již obsahuje název vzdálené metody a její argumenty. Po zpracování zprávy se do tohoto elementu rovněž zapíšou výsledky volání (stejně tak informace o případné chybě).



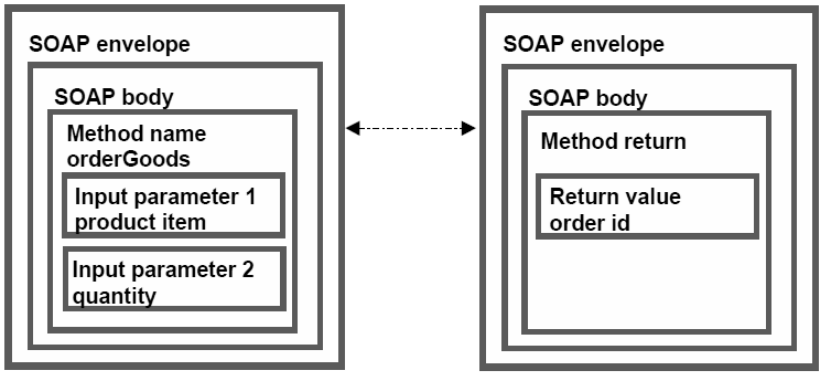
Obrázek 4.2: Hierarchická struktura elementů. Převzato z [10].

`Header` element obsahuje tzv. *hlavičkové informace*, které můžou specifikovat např. digitální podpis a další informace, které slouží ke zpracování obsahu zprávy.

SOAP poskytuje dva základní způsoby komunikace. První způsob vychází z RPC a je většinou synchronní – klient odešle zprávu a čeká na odpověď nebo na chybovou zprávu od serveru. Druhý typ je požadavek založený na dokumentu (SOAP document-style encoding), kde je celý XML dokument odeslán serveru uvnitř SOAP zprávy.

4.2.2 SOAP Remote Procedure Call

Webová služba založena na SOAP-RPC (Soap Remote Procedure Call) se jeví klientské aplikaci jako vzdálený objekt. Klient vyjádří svůj požadavek jako volaní vzdálené metody s příslušnými argumenty a server vrátí odpověď obsahující výsledek.



Obrázek 4.3: Příklad SOAP-RPC. Převzato z [10].

Příklad SOAP-RPC zprávy s elementem `<Envelope>` by mohl vypadat například takto:

Listing 4.1: Příklad SOAP-RPC zpávy.

```
<soap:envelope>
  <soap:body>
    <getSoucet>    <!-- nazev volane metody -->
      <cislo1>2.0</cislo1> <!-- prvni argument -->
      <cislo2>7</cislo2> <!-- druhy argument -->
    </getSoucet>
  </soap:body>
</soap:envelope>
```

Výsledek volání by vypadal takto:

Listing 4.2: Příklad SOAP-RPC odpovědi.

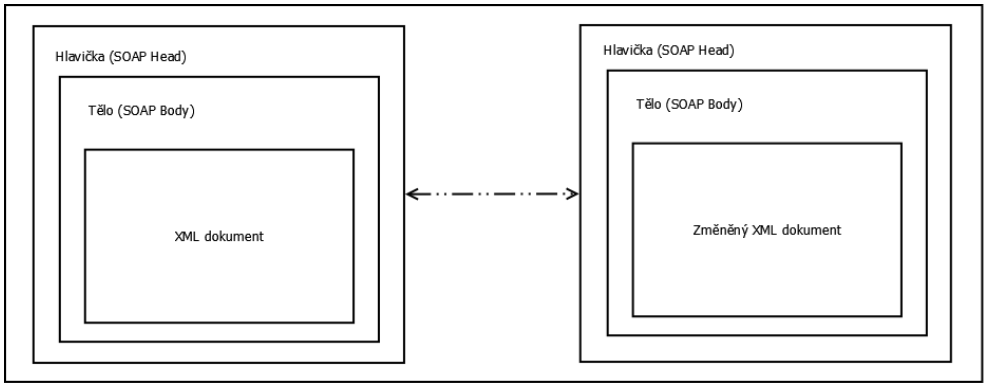
```
<soap:envelope>
  <soap:body>
    <getSoucetResponse>
        <vysledek>9.0</vysledek>
    </getSoucetResponse>
  </soap:body>
</soap:envelope>
```

Webové služby orientované na RPC jsou obvykle synchronní, což znamená, že klient odešle požadavek a čeká na výsledek, dokud požadavek není úplně zpracován.

4.2.3 XML dokument jako SOAP zpráva

Jak už bylo zmíněno výše, SOAP funguje na principu výměny zpráv. V tomto případě si klient se serverem navzájem posílají XML dokumenty (resp. fragmenty XML dokumentu) jako tělo SOAP zprávy namísto parametrů. XML dokument je označen (popsán) svými elementy, ale také XSD schématem. Hlavní výhoda XML dokumentu jako SOAP zprávy (document-style message) oproti SOAP-RPC spočívá právě ve schématu, které má klient i server k dispozici.

Změní-li se způsob komunikace, změní se schéma, ale klient bude i nadále schopen komunikovat se serverem. V případě změny metody volané pomocí SOAP-RPC může dojít k nefunkčnosti všech aplikací, které spoléhají na neměnnou podobu volané metody. Základní konvence říká, že podoba (signatura) SOAP-RPC zprávy i metody by měla zůstat neměnná. V situaci, kdy je použit XML dokument jako SOAP zpráva, toto pravidlo není tak striktní, protože v případě rozšíření či změny může být schéma aktualizováno, aniž by došlo k tomu, že volající (klientská) aplikace by byla nefunkční a nedostala žádný výsledek.



Obrázek 4.4: Výměna SOAP zprávy jako XML dokumentu.

Listing 4.3: Příklad zprávy obsahující XML dokument při odeslání.

```
<soap:envelope>
  <soap:body>
    <kalkulacka>
      <cislo1>2.0</cislo1>
      <cislo2>7</cislo2>
      <vysledek/>
    </kalkulacka>
  </soap:body>
</soap:envelope>
```

Výsledek volání by vypadal takto:

Listing 4.4: Příklad zprávy obsahující XML dokument při přijetí.

```
<soap:envelope>
  <soap:body>
     <kalkulacka>
      <cislo1>2.0</cislo1>
      <cislo2>7</cislo2>
      <vysledek>9</vysledek>
    </kalkulacka>
  </soap:body>
</soap:envelope>
```

4.2.4 Transport dat pomocí HTTP

Jako transportní mechanizmus pro přenos SOAP zpráv se používá HTTP protokol z důvodu velké podpory v různých aplikacích. Navíc je možné používat webové služby bez nutnosti zásahu do konfigurace aktivních síťových prvků, jako jsou firewally apod. Při použití technologií DCOM nebo CORBA je potřeba povolit komunikaci na portech, které používají příslušné přenosové protokoly.

SOAP požadavek lze posílat v těle HTTP požadavku za použití metody POST. To, že se jedná o SOAP požadavek, definuje hodnota `SOAPAction` v hlavičce HTTP požadavku. Tuto hlavičku používají jednak firewally k filtrování požadavků a jednak může obsahovat i URI[1] s identifikací služby, která

[1]URI – Uniform Resource Identifier. Textový řetězec sloužící k identifikaci zdroje.

se má volat. Pokud je obsah hlavičky prázdný, pak je spuštěna služba identifikována přímo adresou, na kterou směřuje požadavek. Při použití metody GET není požadavek definován jako SOAP zpráva. U metody POST tomu tak je, a to jak v případě požadavku, tak i odpovědi.

Listing 4.5: SOAP zpráva poslána přes HTTP

```
POST /kalkulacka HTTP/1.1
Content-Type: text/xml; charset=utf-8
Content-Length: delka
SOAPAction: "http://www.w3.org/2003/05/soap-envelope"
<soap:envelope>
  <soap:body>
    <getSoucet>
      <cislo1>2.0</cislo1>
      <cislo2>7</cislo2>
    </getSoucet>
  </soap:body>
</soap:envelope>

</soap:Envelope>
```

Listing 4.6: Přijatá SOAP zpráva pomocí HTTP

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: delka

<soap:envelope>
  <soap:body>
    <getSoucetResponse>
        <vysledek>9.0</vysledek>
    </getSoucetResponse>
  </soap:body>
</soap:envelope>
```

4.2.5 Výhody a nevýhody

I přes název *Simple* je protokol poměrně složitý na pochopení i učení. Další nevýhodou je silná závislost na HTTP protokolu a fakt, že protokol je bez-

stavový (všechny parametry se s každým voláním přenášejí znovu a znovu).

Hlavní výhody SOAPu spočívají zejména v jeho platformní nezávislosti, možnosti portace, dobré komunikaci s firewally, interoperabilitě a jeho otevřenosti. Velkou výhodou je rovněž fakt, že je velice rozšířený a všeobecně akceptovaný.

4.3 Web Service Description Language

WSDL (Web Service Description Language) je standard založený na XML a reprezentuje jazyk definující detaily kompletního rozhraní webové služby.

4.3.1 Charakteristika

WSDL dokument je XML soubor, který popisuje mechanismy a interakce s konkrétní webovou službou. WSDL je platformě nezávislý jazyk a je primárně určen k popisu služeb postavených na SOAP protokolu. WSDL slouží k tomu, aby bylo popsáno:

- **co** služba dělá (popis akcí, které služba poskytuje)
- **kde** je služba umístěna (detaily ohledně adresy, URL[2])
- **jak** lze službu vyvolat (detaily formátu dat, bezpečnostní protokoly zajišťující přístup k akcím)

Operace a zprávy jsou popsány abstraktně. WSDL rovněž definuje provázanost formátu zpráv a protokolu. Koncové body (tzv. endpoints) jsou definovány síťovým protokolem, ale WSDL dokáže popsat jakýkoliv koncový bod nezávisle na použitém protokolu či formátu zprávy. Rovněž definuje způsob nalezení koncového bodu pro danou službu třeba formou URL.

[2]URL – Unique Resource Locator. Jednoznačné určení umístění zdroje.

4.3.2 Struktura WSDL

WSDL je přímo založeno na XSD schématu a umožňuje seskupovat zprávy do operací a operace do rozhraní. Popis webové služby má na starosti WSDL dokument, který se skládá ze dvou částí:

1. **Definice rozhraní**
2. **Implementace služby**

V **definici rozhraní** (service-interface definition) jsou používány následující elementy:

- `<types>`
- `<operation>`
- `<message>`
- `<portType>`

Types

Element `<types>` obsahuje definice datových struktur. WSDL jazyk není výlučně vázán žádným typovým systémem, tudíž lze pro popis datových typů použít libovolný typový systém. Avšak XSD schémata se používají jako výchozí volba pro popis datových typů. Pokud jsou použity jednoduché datové typy, jako je `string` nebo `integer`, pak element `<types>` není vyžadován a ve WSDL dokumentu se nemusí vyskytovat. Častěji se používá pro popis komplexnějších datových struktur.

Listing 4.7: Příklad types elementu se složeným datovým typem

```
<types>
   <schema targetNamespace="http://example.com/veliciny.xsd"
          xmlns="http://www.w3.org/2000/10/XMLSchema">
    <element name="cas">
        <complexType>
          <all>
            <element name="hodnota" type="float"/>
            <element name="jednotka" type="string"/>
          </all>
        </complexType>
     </element>
   </schema>
  </types>
```

Message

Element `<message>` popisuje formát předávaných zpráv pomocí již předdefinovaných typů. Zpráva je složena z elementů, které se nazývají `<part>` a odpovídají parametru volané operace.

Listing 4.8: Příklad zprávy popisující vstupní a výstupní parametry

```
<wsdl:message name="getSoucetPozadavek">
       <wsdl:part name="cislo1" type="float"/>
       <wsdl:part name="cislo2" type="int"/>
</wsdl:message

<wsdl:message name="getSoucetVysledek">
       <wsdl:part name="vysledek" type="float"/>
</wsdl:message
```

PortType

Element `<portType>` definuje abstraktní typy a jejich operace, ale ne implementaci. Je to logické seskupení elementů `<operations>` a má za úkol popsat všechny operace, které webová služba podporuje.

Operation

Element `<operation>` má za úkol abstraktně popsat operace, které jsou službou podporovány. U operace se definuje, jaké má vstupy a výstupy. Vstup a výstup je popsán již existující zprávou (message). V SOAP RPC modelu odpovídá operace metodě[9].

Listing 4.9: Operation element

```
<wsdl:portType name="SoucetPortType">
    <wsdl:operation name="getSoucet">
       <wsdl:input message="tns:getSoucetPozadavek"/>
       <wsdl:output message="tns:getSoucetVysledek"/>
    </wsdl:operation>
 </wsdl:portType>
```

Jak už bylo výše zmíněno, význam WSDL spočívá v popsání webové služby abstraktně a definuje, jak může tvůrce (vývojář) danou službu konkrétně implementovat. Implementační část WSDL dokumentu používá následující elementy:

- `<binding>`
- `<port>`
- `<service>`

Tyto elementy říkají, **kde** je služba uložena, resp. který síťový protokol musí být použit pro odeslání zprávy. WSDL rovněž umožňuje stejně jako XSD import jiných WSDL dokumentů za pomocí elementu `<import>`.

Binding

Element `<binding>` poskytuje detailní informace o tom, jak se elementy `<portType>` převádějí na konkrétní protokol (SOAP, HTTP) a formát zpráv (RPC nebo jako XML dokument). `<portType>` definuje umístění webové služby a lze se na něj dívat jako na URL.

Port

Element `<port>` reprezentuje koncový bod služby definovaný jako kombinace síťové adresy a dříve definované vazby (binding)[9].

Service

Element `<service>` sdružuje kolekci elementů `<port>`. Každý element `<service>` je pojmenován a toto jméno musí být jedinečné napříč celým WSDL dokumentem.

4.4 Universal Description, Discovery and Integration

V předchozím textu bylo vysvětleno jak službu popsat, jak lze data přenášet a v jakém formátu. V této sekci je popsáno, kde webovou službu najít.

Pojem **UDDI** (Universal Description, Discovery and Integration) pokrývá několik významů, které se sebou často souvisí, ale nejčastěji se pod pojmem UDDI rozumí registr služeb. UDDI je XML orientovaný standard, který umožňuje registraci, vyhledávání a kategorizaci webových služeb. UDDI je rovněž platformě nezávislý a otevřený standard.

UDDI si lze představit jako zlaté stránky, ve kterých můžeme hledat firmy, čísla a jiné informace. To, zda bude nějaká webová služba zaregistrována do UDDI, záleží čistě na autorovi služby. Pokud však chceme, aby byla daná webová služba nějakým způsobem známá a věděli o ní i ostatní, je třeba využít UDDI registr a danou službu zaregistrovat.

Registr UDDI je členěn do tří základních kategorií:

- **White pages** – Obsahují informace o firmě, která služby poskytuje (název, adresa, kontakty apod.). Mají spíše obchodní charakter.
- **Yellow pages** – Členění firem dle oblasti působení (obchod, geografie, meteorologie, apod.) a použitých protokolů.

- **Green pages** – Obsahuje technické informace o webových službách.

4.5 Representational State Transfer

REST (Representational State Transfer) je označení pro architektonický styl založený na webovém standardu a protokolu HTTP. Je orientovaný více na přístup k datům a objektům.

Hlavní myšlenka služeb postavených na REST architektuře spočívá v tom, že není třeba používat komplexní a složité mechanismy pro komunikaci, jako je SOAP nebo RPC. V tomto případě je využit pouze HTTP protokol pro navázání spojení mezi dvěma komunikujícími stroji. RESTově založené aplikace používají HTTP požadavky (angl. `requests`) pro získání dat a dokážou jak data získávat, tak i vzdáleně vytvářet, upravovat či mazat.

Hlavním rysem RESTových služeb je, že komunikace mezi serverem a klientem je vždy bezstavová.

4.5.1 Základní elementy

REST identifikuje šest datových elementů:

- `Resource`
- `Resource Identifier`
- `Resource Metadata`
- `Representation`
- `Representation Metadata`
- `Control Data`

Resource

Klíčový element pro REST architekturu je zdroj (Resource). Jakákoliv informace, kterou lze pojmenovat, je zdroj: dokument, webová stránka nebo

výsledek vyhledávání. Zdroj je konceptuální objekt, který má svou identitu, stav a chování.

Resource Identifier

Identifikátor zdroje (Resource Identifier) označuje konkrétní zdroj, který lze pozorovat skrze jeho reprezentaci (Representation). Typicky se jedná o URL.

Representation

Reprezentace (Representation) může být např. webová stránka nebo dokument. Je to objekt, který lze získat **ze** zdroje, nikoliv zdroj samotný. Některé zdroje jsou statické a některé dynamické. U statických zdrojů platí, že výsledek bude vždy stejný; u dynamických se obsah mění s časem. V tomto případě to může být HTML dokument nebo obrázek.

Representation Metadata

Metadata reprezentace (Representation Metadata) popisuje tzv. **media type**. To jsou textové řetězce, které popisují formát obsahu: `application/xml`, `text/html` nebo `application/json`. Někdy jsou tyto řetězce označovány jako `MIME identifikátory`.

Resource Metadata

Obsahuje zdroje odkazů nebo různé proměnné.

Control data

Kontrolní data (Control data) obsahují např. informace o tom, kdy a jestli byl zdroj změněn.

4.5.2 Operace nad zdroji

Nad všemi zdroji lze provádět akce `POST`, `GET`, `PUT`, `DELETE`. Tyto akce jsou někdy přirovnávány k databázovým operacím nazývaných **CRUD** (Create, Read, Update, Delete).

- `POST` – Aktualizuje zdroj.
- `GET` – Vrátí kopii zdroje.
- `PUT` – Vytvoří zdroj.
- `DELETE` – Odstraní zdroj.

Operace `GET`, `PUT`, `DELETE` jsou tzv. „*idempotentní*“. To znamená, že identicky stejné požadavky vrátí vždy stejný výsledek bez ohledu na počet.

4.5.3 REST a W3C standard

REST je alternativou k mechanismům RPC a protokolům, jako je SOAP, WSDL apod. Je důležité mít na paměti, že REST je architektonický styl a **nejedná se o W3C standard**, tak jako tomu je u SOAPu. Neexistuje W3C doporučení pro ukázkový příklad REST aplikaci. I když to standard není, tak používá standardizované prvky jako HTTP, URL a XML.

Stejně jako webové služby jsou REST služby platformě nezávislé (nezáleží, jestli server se službou běží na UNIX serveru, nebo zda klient běží na Mac OS). REST nenabízí žádné vnitřně implementované bezpečnostní prvky, jako je šifrování, zpráva sezení (session), garanci kvality (QoS) apod. Všechny tyto prvky mohou být implementovány pomocí HTTP protokolu:

- Pro přihlášení jsou použity často tokeny ve formě jméno/heslo.
- Pro šifrování je použit HTTPS protokol (secure socketes layer).

4.5.4 Porovnání SOAP a REST

Při porovnání těchto dvou technologií je největší rozdíl v přístupu. REST požadavky jsou oproti SOAP požadavku mnohem jednodušší a srozumitelnější pro člověka. SOAP se zaměřuje na strojové zpracování. Následující příklad ukazuje jednoduchost REST požadavku:

Listing 4.10: Příklad REST požadavku

```
http://www.example.org/stock/StockPrice?StockName=IBM
```

Na příkladu je zajímavé to, že v URL adrese REST požadavku nikde není text `GetStockPrice` tak, jak by tomu bylo u typického SOAP požadavku. Toto je dáno běžnou konvencí dodržovanou v REST architektuře, ve které se častěji používají podstatná jména místo sloves. Jde však pouze o doporučenou konvenci.

5 Webové služby nad IS/STAG

Ve studijním informačním systému IS/STAG je uloženo mnoho užitečných informací, které jsou veřejně přístupné například přes nativního klienta, webové stránky či portál. Všechny tyto výstupy jsou ale orientovány na „uživatele-lidi“. Stále častěji se ale objevují požadavky na možnost získání informací z IS/STAG automatizovaně a ve strojem zpracované podobě. Obecně lze říci, že je třeba, aby IS/STAG měl další rozhraní, které bude sloužit k jeho integraci s dalšími počítačovými systémy[1].

„Webové služby nad IS/STAG“ je souhrnný název pro modul informačního systému IS/STAG, který umožňuje používat nasazené webové služby, prohlížet je a provádět další operace.

Pro ustálení terminologie je vhodné upřesnit a hlavně definovat pojmy „webová služba“ a „operace“ či „metoda“.

Definice 1 *Pod pojmem webová služba budeme rozumět* ***seskupení operací*** *sdružených pod danou webovou službou. Každá služba je jednoznačně identifikovatelná podle adresy (někdy označováno jako endpoint).*

Definice 2 *Pod pojmem operace či metoda budeme rozumět již konkrétní akci, která dokáže zobrazit užitečné informace z informačního systému IS/STAG na základě nějakého požadavku.*

5.1 Seznam služeb

Seznam všech webových služeb nad instalací IS/STAG na ZČU lze najít na webové adrese `https://stag-ws.zcu.cz/ws/help/list`. Stránku s touto adresou lze z obecného pohledu webových služeb považovat za jakýsi zjednodušený UDDI registr. Všechny služby jsou zde uvedeny jak v REST podobě, tak i SOAP. Tato sekce je velice důležitá, protože zde by se měly zobrazovat i uživatelem generované služby.

Typ	Adresa			Popis
REST	/ws/services/rest/absolventi	SOAP	/ws/services/soap/absolventi	Modul Absolventi
REST	/ws/services/rest/ciselniky	SOAP	/ws/services/soap/ciselniky	Číselníky STAGu
REST	/ws/services/rest/dSpaceImport	SOAP	/ws/services/soap/dSpaceImport	Zakazkove sluzby pro knihovnu ZCU, sl
REST	/ws/services/rest/dSpaceSimple	SOAP	/ws/services/soap/dSpaceSimple	lite verze
REST	/ws/services/rest/ects	SOAP	/ws/services/soap/ects	ECTS služby.
REST	/ws/services/rest/help	SOAP	/ws/services/soap/help	Různé pomocné a interní služby
REST	/ws/services/rest/kalendar	SOAP	/ws/services/soap/kalendar	Kalendář, harmonogram ak. roku, atd.
REST	/ws/services/rest/kvalifikacniprace	SOAP	/ws/services/soap/kvalifikacniprace	Kvalifikační práce (diplomky, bakalářky)

Obrázek 5.1: Výřez stránky se seznamem nasazených služeb.

5.1.1 Detail webové služby

Kliknutím na adresu webové služby se zobrazí webová stránka s detaily dané služby. Na této stránce je zobrazen název služby, typ, adresa a její WSDL schéma. Dále je zobrazen seznam operací, které jsou pod danou službou sdruženy.

Výřez webové stránky s detailem služby lze najít v příloze A.1.

5.1.2 Webové rozhraní nad službami

Pro jednodušší práci se službami existuje uživatelské webové rozhraní, které umožňuje uživatelsky příjemnější volání služeb. Hodnoty argumentů webových služeb mohou být rovněž „napovídány“, a urychlují tak práci se službami. Další z funkcí, které toto rozhraní nabízí, je i hromadné nahrání dat do databáze prostřednictvím dostupných „nahrávacích“ služeb. Typickým vstupem může být např. soubor typu XLS (Microsoft Excel) obsahující seznam studentů, kterým byl udělen zápočet.

5.2 Podporované formáty

Základní formát dat, ve kterém služby vrací výsledky je standardně XML, ale jsou podporovány i další formáty. Tyto formáty mají však význam pouze pro REST služby, kdy se do URL webové služby přidá speciální parametr **outputFormat**.

Seznam dostupných formátů:

- **XLS** – Formát programu Microsoft Excel pro ukládání tabulek.
- **CSV** – Jednoduchý textový formát pro ukládání tabulkových dat.
- **ICS** – Formát pro ukládání kalendářových dat (iCal).
- **JSON** – Odlehčený formát pro výměnu dat. Vhodný je pro strojové zpracování, ale je čitelný i pro člověka (viz 3.2).

5.3 Vztah služby a SQL dotazu

Většina služeb a jejich operace poskytují nějaké většinou užitečné informace z databáze IS/STAG. Tyto informace poskytují webové služby prostřednictvím **SQL dotazů** do databáze IS/STAG.

5.3.1 Jazyk SQL

SQL (Structured Query Language) je dotazovací jazyk, který se používá pro práci s daty v relačních databázích. Tento jazyk v sobě obsahuje příkazy, pomocí kterých lze vytvářet databáze (tabulky) či manipulovat s daty.

SQL patří do skupiny tzv. **deklarativních programovacích jazyků**, což znamená, že kód není napsán v samostatném programu, ale je vkládán do jiného programovacího jazyka. V případě webových služeb nad instalací systému IS/STAG se bude jednat o jazyk Java.

5.3.2 SQL dotazy

SQL dotaz je již konkrétní zápis příkazu v jazyce SQL. Jazyk SQL se skládá z několika částí, kde každá část slouží k něčemu jinému. Nejčastějším typem dotazu je **SELECT**.

- **SELECT** - získání dat z databáze
- **INSERT** - vložení dat do databáze
- **UPDATE** - změna dat v databázi

- **DELETE** - smazaní dat z databáze

5.4 Tvorba služeb nad instalací IS/STAG

V řeči programátora je webová služba (webová) aplikace napsaná v jazyce Java, která dokáže pomocí SQL dotazů zobrazit požadované výsledky z databáze. Webovou službu si lze v tomto případě představit jako pomyslnou „obálku“ nad SQL dotazy.

5.5 Technologie a nástroje

Před samotnou tvorbou služby je vhodné se seznámit s nástroji a frameworky, které je nutné použít, abychom mohli službu sestavit a poté i spustit.

5.5.1 Vývojové prostředí

Základním nástrojem je vývojové prostředí.

Pro vývoj webových a obecně Java EE aplikací je možné použít prostředí Eclipse s označením WTP (Web Tools Platform). Tato verze v sobě již obsahuje velké množství podpůrných nástrojů pro vývoj webových služeb a vývoj J2EE aplikací obecně. Prostředí Eclipse není však jediné, které lze použít pro vývoj služeb. Existují i další prostředí, jako je NetBeans či IntelliJ IDEA. Záleží na vkusu a zkušenostech každého vývojáře, avšak prostředí Eclipse osobně považuji za nejlepší, a tudíž jej doporučuji.

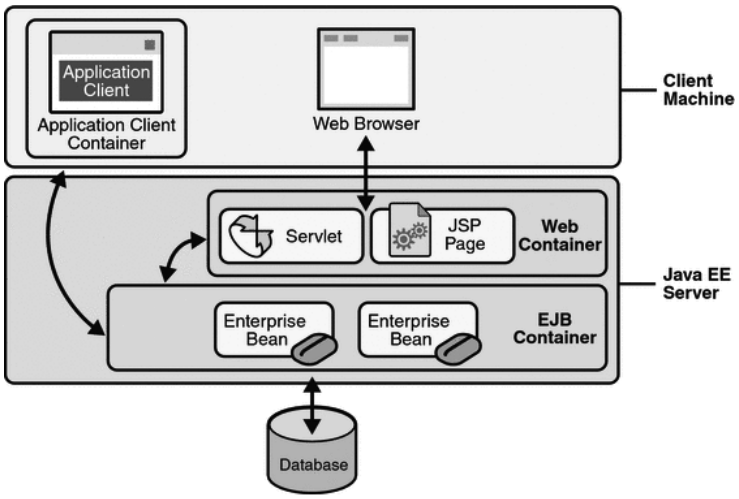
Podrobnější informace a samotné prostředí lze najít na [8].

5.5.2 Aplikační server

Aby bylo možné webovou službu spustit a poté i volat, je nutné mít k dispozici nějaké běhové prostředí. K tomuto účelu slouží aplikační server. Mezi nejznámější servery patří **Apache Tomcat**, GlassFish, JBoss a IBM WebSphere.

Aplikační server pracuje s **komponenty**, které reprezentují základní jednotky, ze kterých je složena výsledná aplikace. Existuje několik druhů komponent, z nichž nejdůležitější jsou **Webové komponenty** a **EJB komponenty (Enterprise JavaBeans)**.

Aby mohly být komponenty spuštěny, je třeba tzv. **kontejner**. Opět existují dva druhy kontejneru pro příslušné komponenty: **Webový kontejner** a **EJB kontejner**. Existují i další druhy kontejnerů, ale pouze tyto dva běží na aplikačním serveru. Architektura aplikačního serveru je zobrazena na obrázku 5.2.



Obrázek 5.2: Aplikační server. Převzato z [22].

Z pohledu této diplomové práce je důležitější EJB kontejner a EJB komponenty. EJB komponenty nejsou nic jiného než Java třídy, které tvoří logiku aplikace a dokáží manipulovat s daty. Všechny webové služby běžící nad STAGem mají jednotné uživatelské rozhraní, a tudíž není třeba se zabývat JSP (Java Server Pages) či HTML stránkami, které by představovaly *View* vrstvu. Podrobnější popis a význam lze dohledat v [22].

Aplikace „Webové služby nad IS/STAG“ a přidružené webové aplikace běží na aplikačním serveru **Apache Tomcat**.

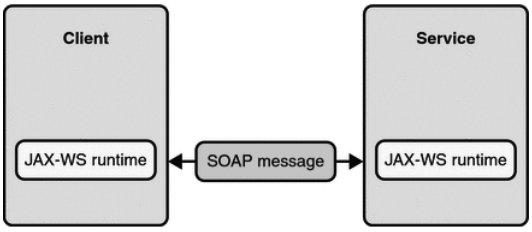
5.5.3 Apache CXF

Pro vývoj webových služeb nad STAGem je použit modifikovaný[1] framework Apache CXF, který poskytuje jednoduché modely a funkce usnadňující tvorbu služeb. Webové služby mohou být implementovány pomocí různých protokolů a standardů, jako jsou SOAP, XML, JSON a REST.

Apache CXF framework podporuje (a implementuje) několik standardů, z nichž nejdůležitější jsou **JAX-WS** a **JAX-RS**.

JAX-WS

JAX-WS (Java API for XML Web Service) je základní technologie určená k vývoji služeb založených na protokolu SOAP. Součástí JAX-WS je i technologie JAXB (Java Architecture for XML Binding) umožňující mapovat XML schémata na Java kód a SOAP zprávy. Hlavní výhoda JAXB spočívá v tom, že programátor nemusí vytvářet žádné XML parsery (programy zpracovávající XML dokument), které by zpracovávaly XML a SOAP zprávy.



Obrázek 5.3: Komunikace mezi klientem a službou JAX-WS. Převzato z [22].

JAX-RS

Java API for RESTful Web Service (JAX-RS) je specifikace pro podporu služeb založených na REST architektuře. JAX-RS specifikace nedefinuje žádné implementační detaily ohledně REST klienta, ale pomocí frameworku Apache CXF a jeho přidružených nástrojů lze vytvořit klienta, který dokáže komunikovat se službou založenou na JAX-RS.

[1]Framework byl upraven tak aby vyhovoval potřebám programátorům na CIVu

Java Architecture for XML Binding

JAXB (Java Architecture for XML Binding) je označení pro technologii umožňující převod XML dat na Java objekty a naopak. Aby bylo možné převádět Java objekty na XML a zpět, je nutné, aby příslušné Java třídy měly bezparametrický konstruktor a byly použity JAXB anotace. Název třídy je anotován anotací `@XmlRootElement`.

JAXB nemusí být použito pouze v kontextu webových služeb (JAX-WS), ale dá se použít v jakékoliv aplikaci, která potřebuje mapovat Java objekty na XML a naopak.

5.6 Příklad webové služby

Následující text ukazuje příklad implementace jednoduché webové služby (a její operaci), která dokáže pomocí SQL dotazu zobrazit data z databáze, kterou používá informační systém IS/STAG.

V databázi používající IS/STAG existuje tabulka se seznamy krajů (KRAJE) v České republice.

Pro tuto tabulku můžeme vytvořit SQL dotaz, který nám vrátí všechny řádky s kraji, u kterých je jejich kód větší než nějaké číslo. Budeme předpokládat, že nás zajímá pouze kód samotný a název kraje. SQL dotaz by měl pak následující tvar:

Listing 5.1: SQL dotaz.

```
select kraj_kod, nazev from kraje where kraj_kod>:kod
```

Poté, co je definován SQL dotaz, můžeme začít s tvorbou služby. Celý proces se dá rozdělit do čtyř fází:

1. Vytvoření tzv. **Service Endpoint Interface (SEI)**, což je Java rozhraní s definovanými metodami.
2. Vytvoření třídy, která implementuje rozhraní z kroku 1, a přidání anotací, které definují webovou službu a její operace.
3. Vytvoření třídy, která se bude starat o získávání dat z databáze.

4. Vytvoření XML souborů `ws-beans.xml`, `soap-services.xml`, `rest-services.xml` a `extension.xml`. V těchto souborech se definuje konfigurace aplikace[2].

V příloze B je zobrazena struktura (Eclipse) projektu s uspořádáním balíčků, tříd a XML souborů pro konfiguraci ukázkové služby. O samotný překlad a sestavení aplikace se ve většině případů postará samotné vývojové prostředí, ale sestavená aplikace musí být ve formě **JAR** (Java Archive) archívu. K sestavení lze použít nástroj `Ant`[27] nebo `Maven`[4].

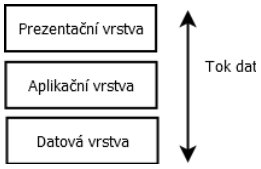
5.6.1 Architektura

Stejně jako každá jiná aplikace musí být tato ukázková aplikace postavena na nějaké architektuře. Pro většinu webových aplikací se používá tzv. MVC architektura, ale v tomto případě si vystačíme s obyčejnou **třívrstvou architekturou** složenou z datové, aplikační a prezentační vrstvy. Někdy jsou tyto dvě architektury vzájemně zaměňovány, ale je nutné mít na paměti, že MVC architektura není lineární a tvoří spíše pomyslný trojúhelník.

V případě naší ukázkové webové služby se musíme postarat pouze o implementaci **aplikační vrstvy** a přístupu k **datové vrstvě**. Prezentační vrstva sloužící k zobrazení dat uživateli je generována automaticky a my se o ni nemusíme starat.

V rámci Aplikační vrstvy musíme vytvořit vlastní logiku služby a v rámci přístupu k datové vrstvě musíme vytvořit tzv. **DAO – Data Access Object** objekty. Ty pracují s tzv. doménovým modelem, který tvoří síť objektů reprezentujících reálný svět. Ve světě Javy se tyto objekty nazývají „Java Beans“ a k jejich členským proměnným se přistupuje skrze veřejné *gettery* a *settery*. V případě této webové služby bude doménový model pouze instance třídy **KrajBean.java**.

[2]Používá se Spring konfigurace.



Obrázek 5.4: Třívrstvá architektura.

5.6.2 SEI rozhraní

Listing 5.2: Definice rozhraní

```
@WebService(targetNamespace = WSConstants.
   DEFAULT_SERVICE_NAMESPACE)
@StagComment("Ukazkova testovaci sluzba")
@DBDependency("STAGDataSource")
public interface IKrajService
{
    @Get
    @RequestWrapper(targetNamespace = WSConstants.
       DEFAULT_TYPES_NAMESPACE)
    @ResponseWrapper(targetNamespace = WSConstants.
       DEFAULT_TYPES_NAMESPACE)
    @StagMethod(authentication = 0, roles = "", expireTime =
       StagMethod.EXPIRE_TIME_ALWAYS)
    @WebResult(name = "KrajResult")
    @StagComment("Zobrazi kraje, ktere maji kod vetsi nez
       vstupni parametr.")
    public KrajListBean getKraje(@WebParam(name = "kod")
       @StagComment("Cislo kraje") int kod);
}
```

Z příkladu lze vypozorovat kód, který není úplně běžný. Jedná se o tzv. *anotace*[3], které přidělují rozhraní další vlastnosti v rámci frameworku CXF.

Anotace `@WebService` je součást knihovny z JAX-WS standardu. Ten poskytuje další anotace, které umožňují převést obecné Java třídy na webové služby.

[3]způsob zápisu kódu, jak přiřadit nějakému elementu (třídě, rozhraní, metodě, proměnné) nějakou další vlastnost nebo informaci

`@StagMethod` nahrazuje `@WebMethod` a identifikuje webovou metodu. Argument `autentication` říká, zda je nutné přihlášení uživatele při volání metody. `@WebResult` definuje název elementu, ve kterém bude uložen výsledek. `@StagComment` má význam pouze pro vizualizaci - uživateli napovídá, co má zadat jako parametr, nebo obecně popisuje význam služby (metody). Anotace, které začínají `@Stag` prefixem nejsou součástí CXF frameworku, ale pouze jeho rozšířením.

`@GET` anotace určuje způsob volání. Alternativně lze použít i anotaci `@POST`, která se používá u metod, které mají místo primitivních datových typů (int, String apod.) složitější typy.

5.6.3 Implementace rozhraní

Poté, co bylo vytvořeno rozhraní, je nutné jej implementovat.

Listing 5.3: Fragment kódu implementace rozhraní.

```
@WebService(targetNamespace = WSConstants.
   DEFAULT_SERVICE_NAMESPACE, endpointInterface = "cz.civ.stag
   .ws.service.IKrajService")
public class KrajServiceImpl implements IKrajService
{
        private IKrajManager manager;

    @StagMethod
    public KrajListBean getKraje(@WebParam(name = "kod")
       @StagComment("Cislo kraje") int kod)
    {
        return manager.getKraje(kod);
    }
}
```

Opět je zde použita anotace `@WebService`, která má navíc další parametry: `targetNamespace`, který definuje jmenné prostory elementů a `edpointService`, který obsahuje název výše uvedeného Java rozhraní. Tím se de facto prováže implementační třída s rozhraním z pohledu CXF frameworku. Třída `IKrajManager` představuje tenkou mezivrstvu mezi aplikační logikou a vrstvou poskytující doménový model.

5.6.4 Získání dat z databáze

K získání dat z databáze je použita třída **KrajDaoImpl** implementující JDBC rozhraní (konkrétně Spring JDBC).

Listing 5.4: Metoda získávající data z databáze pomocí JDBC.

```
public List<KrajBean> getKraje(int kod)
{
ArrayList<KrajBean> beans = new ArrayList<KrajBean>();
Map m = new HashMap<String, Object>();
    m.put("kod", kod);

List<Map<String, Object>> rows = getNamedParameterJdbcTemplate
    ().queryForList(SQL_getKraje, m);
   for (Map row : rows)
   {
     KrajBean bean = new KrajBean();
     String resultAsString = "";

     resultAsString = ""+row.get(KrajBean.PROPERTY_KRAJ_KOD);
         if(!resultAsString.equals("null"))
         {
                bean.setKraj_kod(""+resultAsString);
         }

         resultAsString = ""+row.get(KrajBean.PROPERTY_NAZEV);
         if(!resultAsString.equals("null"))
         {
                bean.setNazev(""+resultAsString);
          }

         beans.add(bean);
        }
     return beans;
}
```

5.6.5 Registrace služby

Registrace služby se provádí skrze XML soubor s názvem `ws-beans.xml`.

Listing 5.5: Registrace služby.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans>
    <bean id="krajeTest" class="cz.civ.stag.ws.service.
       KrajServiceImpl">
              <property name="manager" ref="manKraje" />
       </bean>
</beans>
```

Aby byla služba vidět v seznamu nasazených služeb, je nutné nadefinovat její **SOAP endpoint** a **REST endpoint**. Definice SOAP endpointu se provádí v XML souboru `soap-services.xml` a REST endpointu `rest-services.xml`.

Listing 5.6: Definice SOAP endpointu.

```
<?xml version="1.0" encoding="UTF-8"?>

<beans>
  <jaxws:endpoint id="soap.kraje"
                  implementor="#krajeTest"
                  address="/soap/krajeTest"
                  bindingUri="http://services.stag.zcu.cz/
                     binding/stag/soap"/>

</beans>
```

Listing 5.7: Definice REST endpointu.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans>
<jaxws:endpoint id="rest.kraje"
                  implementor="#krajeTest"
                  address="/rest/krajeTest"
                  bindingUri="http://services.stag.zcu.cz/
                     binding/stag/http"/>
</beans>
```

V obou těchto souborech je nejdůležitější atribut `impelmentor`, který obsahuje odkaz na zaregistrovanou službu definovanou v `ws-beans.xml`. Atribut **id** musí obsahovat vždy jednoznačný identifikátor, který se nesmí nikde jinde opakovat. Atribut `address` definuje adresu, pod kterou bude služba přístupná.

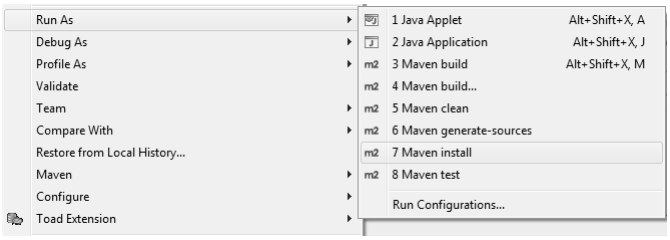
Poslední soubor, který je nutný pro správné nasazení služby je soubor `extension-xml`. Ten sdružuje všechny výše uvedené XML dokumenty do jednoho.

Listing 5.8: Extension.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans>
    <import resource="ws-beans.xml"/>
    <import resource="rest-services.xml"/>
    <import resource="soap-services.xml"/>
</beans>
```

5.6.6 Nasazení a spuštění služby

V této fázi by měl být vytvořen JAR archív s přeloženými Java třídy a adresářem META-INF. Následující obrázek ukazuje možnost sestavení služby do podoby JAR archívu prostřednictvím nástroje Maven spuštěného z Eclipse.



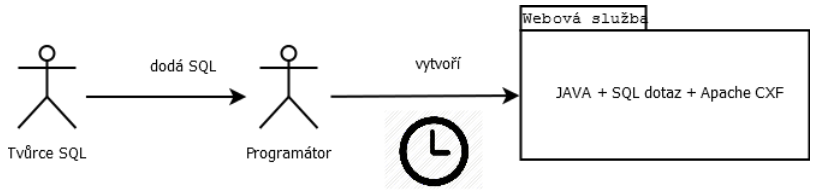
Obrázek 5.5: Vytvoření JAR archívu pomocí nástroje Maven.

Tento JAR archív je nutné zkopírovat do adresáře `webapps/ws/WEB-INF/lib/` nacházejícího se v instalaci aplikačního serveru Tomcat. Zadáním adresy služby `https://stag-demo.zcu.cz/ws/services/rest/kraje/getKraje?kod=80` do webového prohlížeče se zobrazí výsledek volání operace **getKraje(80)** (viz příloha C).

5.7 Cíl práce

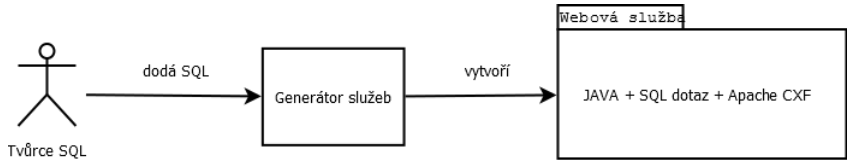
Typický administrátor STAGu je člověk, který je zběhlý ve správě databází, dokáže vytvářet komplexní a efektivní SQL dotazy, ale nemusí už nic vědět o tvorbě služeb pomocí Apache CXF.

Hlavním cílem této práce je poskytnout administrátorům systému IS/STAG nástroj, který by umožňoval automatizovanou tvorbu webových služeb bez ohledu na znalosti programovacího jazyka Java či frameworku CXF. Takový nástroj lze nazvat jako **generátor služeb**.



Obrázek 5.6: Současný proces tvorby služeb.

Úkony spojené s tvorbou Java tříd a XML souborů mohou být náročné jak časově, tak i technologicky, neboť programátor vytvářející služby musí ovládat příslušné technologie a rovněž musí mít i obecné programátorské dovednosti. Cílem je odstranit jak technologickou, tak i časovou bariéru.



Obrázek 5.7: Žádoucí proces tvorby služeb.

6 Strojové generování kódu

Strojové (automatizované) generování kódu je proces, při kterém jsou zdrojové soubory generovány programem. Při generování kódu je nutné nejdříve určit, co bude vstupem a co výstupem.

6.1 Motivace

V předchozí kapitole byl vysvětlen způsob tvorby webových služeb pomocí Java tříd, které používají framework Apache CXF. Cílem této kapitoly je analýza dostupných nástrojů, které by umožňovaly generovat Java třídy tak, aby mohl být použit i CXF framework.

Obrázek 6.1: Schéma generování kódu.

6.2 Výhody generovaného kódu

- **Produktivita**
  Předpokládáme-li, že znalosti generátoru a programátora budou rovnocenné, pak čas, který stráví programátor psaním kódu, bude vždy delší než čas, který potřebuje program.

- **Kvalita**
  Generovaný kód bude mít vždy stejnou kvalitu, ale v případě programátora je přítomen lidský faktor. Programátor může napsat pokaždé různě kvalitní kód, vezmeme-li v potaz, že je např. pod stresem, nemocný, nesoustředěný apod. Program, který má kód generovat, je napsán jen jednou a předpokládá se, že kód je vygenerován dle pravidel[1], která odpovídají čistému a kvalitnímu kódu. Cílem je nastavit pravidla tak, aby napsaný kód byl stejně tak dobrý, jako by byl napsán (dobře) ručně.

[1] Algoritmus, či vnitřní struktura programu řídící generování kódu.

- **Konzistence**
  Generátor kódu používá vždy konzistentní třídy, metody a argumenty. Dobře napsaný generátor by měl generovat vždy přeložitelný kód.

- **Abstrakce**
  Lze vytvořit generátor, který definuje pravidla pro generování kódu abstraktně. To znamená, že generátor je schopen generovat kód, který lze cílit na různé platformy i technologie, u kterých je možný předpoklad využití v budoucnosti, aniž by byl současný stav nějak dotčen. Typickým příkladem mohou být CASE nástroje uvedené v sekci 6.7.1.

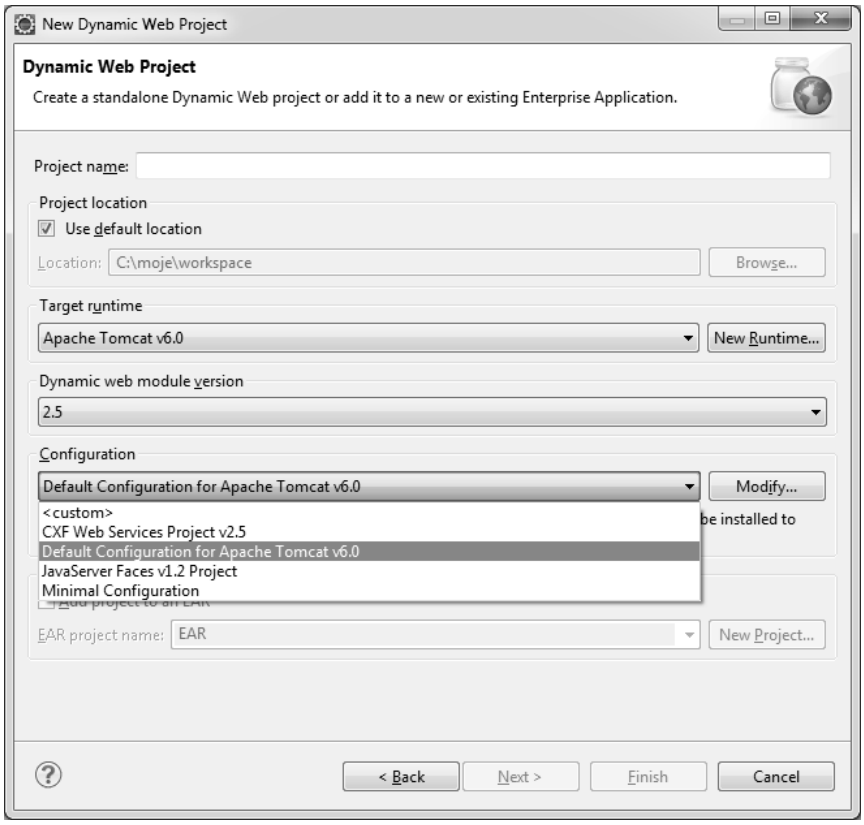
6.3 Vývojové prostředí jako generátor

V současnosti se každý programátor setkává s nějakým generátorem kódu, aniž by si to uvědomoval. Typický generátor je samotné vývojové prostředí (Eclipse, NetBeans, IntelliJ a další). Každé vyspělejší vývojové prostředí dokáže vytvořit třídu nebo poskytuje funkce, které programátorovi napovídají, co zrovna by mohl potřebovat (tzv. Template Proposals).

6.3.1 Průvodci

Dalším typickým generátorem kódu může být tzv. průvodce (Wizzard nebo Guide). Ten vede uživatele/programátora procesem generování po krocích a v každém kroku čeká na uživatelský vstup. Výstupem může být opět zdrojový kód v jazyce Java či jiné typy. To, co má většina generátorů společných, je klausule **Automated generated code, please don't modify**. Často se stává, že vygenerovaný kód není příliš čitelný nebo optimální. Z tohoto důvodu je zmíněná klausule většinou programátory ignorována, protože si potřebují kód upravit dle potřeby.

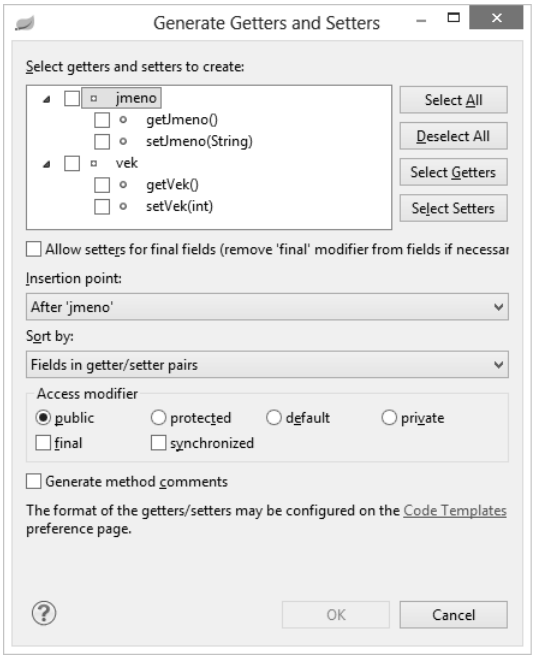
Nejčastěji se lze s průvodci setkat opět v nějakém vývojovém prostředí. Průvodce vede programátora (nebo libovolného uživatele) celým procesem generování, který může být rozplánován na několik kroků. V každém kroku pak generátor očekává nějaký uživatelský zásah v podobě vyplnění požadovaných polí či vybrání nějaké nabízené alternativy. Na obrázku 6.2 je zobrazen příklad průvodce a jednoho kroku z prostředí Eclipse.



Obrázek 6.2: Průvodce z prostředí Eclipse.

6.3.2 Generování metod

Další generátor, který je dnes součástí téměř každého vývojového prostředí, je generátor tzv. **getterů** a **setterů**. To jsou veřejné metody, které zajišťují přístup ke členským proměnným. Na obrázku 6.3 je zobrazen dialog z vývojového prostředí Eclipse, který umožňuje vygenerování metod.



Obrázek 6.3: Generátor metod pro členská data.

6.4 Základní metody

V základě lze použít tři přístupy pro generování Java kódu, kde generátor je:

1. **Řízený kódem** (Code-Driven)
2. **Řízený modelem**(Model-Driven: Custom)
3. **Řízený modelem-architekturou**(Model Driven Architecture)

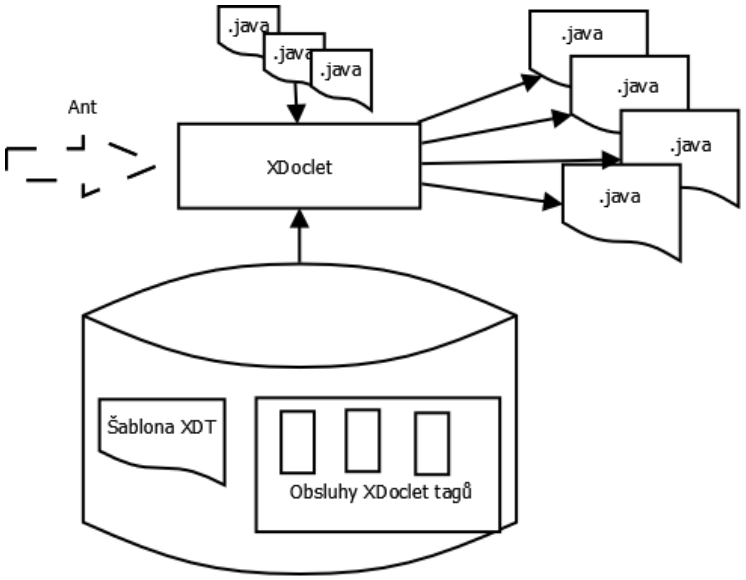
6.5 Generátor řízený kódem

6.5.1 XDoclet

Neznámějším zástupcem tohoto přístupu je nástroj **XDoclet**. Jedná se o otevřený (Open Source) projekt, který je určen pro generování Java kódu. Používá tzv. atributově orientované programování (Attribute-Oriented Programming), což znamená, že umožňuje vkládat do kódu speciální atributy, které rozšiřují význam třídy či metody. Tyto atributy (XDoclet tagy) se vkládají do JavaDoc[2] komentářů.

Hlavní funkce XDocletu, které generují kód jsou postavené na kombinaci použitých speciální atributů (XDoclet tagů) ve zdrojovém Java souboru a předdefinovaných šablon.

Pro spuštění generování kódu používá XDoclet nástroj **Ant**.



Obrázek 6.4: Generování zdrojových kódů pomocí XDoclet.

[2]Komentáře ke třídám a metodám, ze kterých lze vygenerovat Java dokumentaci.

Následující příklad ukazuje XDoclet šablonu, která dokáže vygenerovat třídu, její členská data a k těm příslušné **gettery** a **settery**.

Listing 6.1: XDoclet šablona. Kód převzat z [18].

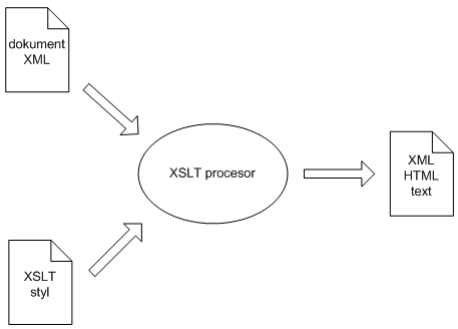
```
public class <XDtClass:className/>Form extends ActionForm {
<XDtMethod:forAllMethods>
<XDtMethod:ifIsGetter>
<XDtMethod:ifHasMethodTag tagName="dw.genStruts">
private <XDtMethod:methodType/> <XDtMethod:propertyName/>;
public <XDtMethod:methodType/> <XDtMethod:getterMethod/>()
{
        return <XDtMethod:propertyName/>;
}
public void <XDtMethod:setterMethod/>(<XDtMethod:methodType/>
   value)
{
     <XDtMethod:propertyName/> = value;
}
</XDtMethod:ifHasMethodTag>
</XDtMethod:ifIsGetter>
</XDtMethod:forAllMethods>
}
```

6.6 Generátor řízený modelem

Generátory, které jsou řízené modelem, mají společnou vlastnost v používání obecného modelu a šablon. Někdy se taky používá pojem šablonově řízené (Template-Based) generování kódu. Šablona je textový soubor, ve kterém jsou předdefinovány některé části, kterou jsou součástí výstupu, a pravidla aplikovaná na vstupní model.

6.6.1 XSLT

XSLT (eXtensible Stylesheet Language Transformation) je standard, který umožňuje převod XML dokumentů na jakýkoliv jiný formát. Nejčastěji je výstupní formát HTML nebo opět XML, ale záleží na způsobu použití šablony, která výstup definuje.



Obrázek 6.5: XSLT transformace. Převzato z [9].

Princip XSLT je hodně jednoduchý a spočívá pouze v tom, že XSLT procesor zpracuje vstupní XML dokument, aplikuje šablonu (styl) a výsledek zapíše do výstupního souboru. Šablona je XML dokument, který používá jazyk XSL (eXtensible Stylesheet Language). Pomocí tohoto jazyka jsou vytvářena pravidla pro transformaci jednotlivých prvků (elementů) ve vstupním XML souboru.

Hlavní výhoda XSLT je jeho rychlost a možnost aplikování více stylů na jeden XML dokument současně. Nevýhoda spočívá v jeho „nešikovnosti“, pokud je generován obyčejný text. Generování prostého textu je dosti komplikované, pokud existují požadavky na přesné umístění textu a konkrétních mezer (např. Java soubory). Proto je vhodnější pro zpracování dokumentů, u kterých je očekáván výstup v XML nebo HTML formátu.

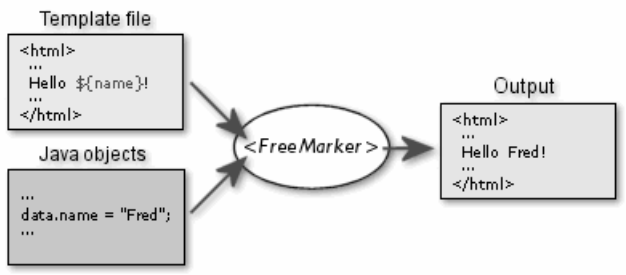
6.6.2 FreeMarker

FreeMarker je šablonově orientovaný nástroj založený na Javě. Tento nástroj byl vyvinut primárně pro potřeby aplikací, které jsou postaveny na tzv. návrhovém vzoru MVC (Model View Architecture). Z tohoto důvodu je často používán i ve webových aplikací používajících servlety[3]. Může být však použit pro generování jakéhokoliv výstupu.

FreeMarker je často označován jako framework pro webové aplikace, ale není tomu tak. Je to pouze vhodná komponenta, kterou webové aplikace mohou využívat – FreeMarker jako takový neposkytuje žádné možnosti práce

[3]Servlet – Modul napsaný v Javě umožňující zpracovat požadavek klienta a také odpovědět.

třeba s HTTP protokolem.



Obrázek 6.6: Nástroj FreeMarker. Převzato z [21].

FreeMarker je velice podobný nástroji **Velocity**, který poskytuje podobné funkce. Oba jsou to šablonově zaměřené nástroje, ale FreeMarker je robustnější a poskytuje větší možnosti konfigurace a více funkcí.

Hlavní funkce a vlastnosti:

- Umožňuje obalení vstupního modelu do univerzální podoby, se kterou umí pracovat.
- Podporuje lokalizaci a internacionalizaci (l10n a i18n).
- Snadno konfigurovatelný a rozšiřitelný.
- Umožňuje výstup směrovat do jakéhokoliv zapisovače (rozhraní Writer).
- Umožňuje načítat šablony ze souboru, databáze, webu nebo JAR archívu.
- Podporuje makra.
- Podpora pro JSP (Java Server Pages) značky.
- Obsahuje velké množství funkcí pro práci s textem.

Největší výhodou FreeMarkeru je jeho jednoduchost používání a čitelnost šablon. Šablony jsou ukládány do souborů s příponou **.FTL**. Jazyk, který se používá ke psaní šablon poskytuje všechny běžné konstrukce, jako je `if`, `else` nebo konstrukci pro tvorbu cyklů.

Následující příklad ukazuje způsob generování jednoduché Java třídy.

Listing 6.2: Vytvoření modelu

```
 Map<String, Object> model = new HashMap<String, Object>();

model.put("ClassName", "TestTrida");

List<String> clenData = new ArrayList<String>();
clenData.add("jmeno");
clenData.add("prijmeni");

model.put("clenData", clenData);
```

Listing 6.3: Šablona třídy

```
public class ${ClassName}
{
<#list clenData as data>
   private String ${data};
</#list>
}
```

Po ukončení procesu generování by výsledek mohl vypadat nějak takto:

Listing 6.4: Vygenerovaná Java třída.

```
public class TestTrida
{
private String jmeno;
private String prijmeni;
}
```

6.7 Modelem Řízená Architektura

Základní myšlenka je založena na tom, že vstupní model by měl být převeden na tzv. UML (Unified Modeling Language) model. UML je grafický jazyk umožňující specifikovat a vizualizovat skrze diagramy např. programové systémy, myšlenkové pochody či časové posloupnosti činností.

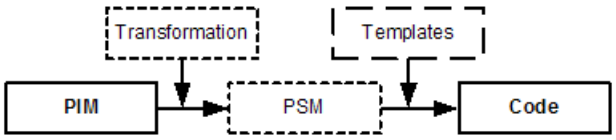
V této souvislosti se často používá pojem MDA (Model Driven Archi-

tecture), což je standard snažící se oddělit aplikační logiku od tzv. business logiky (např. definice tabulky s konkrétními sloupci) a technologické platformy.

MDA rozděluje architekturu na čtyři úrovně:

- **CIM** – Model nezávislý na počítačovém zpracování.
- **PIM** – Model nezávislý na platformě.
- **PSM** – Model závislý na konkrétní platformě .
- **Zdrojový kód**.

Proces generování zdrojového kódu začíná u PIM modelu, který se musí převést na UML model. Pomocí složité transformace se tento UML model převede do PSM modelu, na který lze už aplikovat šablonu s pravidly definující generovaný kód. Celkově je tento proces velice složitý a často jej lze použít pouze k vygenerování základních tříd patřících do datové vrstvy. Typicky se jedná např. o třídy, která mají pouze nějaká členská data, set metody a get metody. Podrobnější popis lze dohledat v [7].



Obrázek 6.7: Generátor kódu založený na MDA. Převzato z [20].

6.7.1 CASE nástroje

Nástrojům, které slouží k modelování architektury (většinou pomocí UML) se říká CASE (Computer Aided Software Engineering) nástroje. Ty dokážou vytvářet modely pomocí diagramů a k nim i případné dokumentace, protože člověk dokáže lépe chápat obrázek než složité technické pojmy. Často se používají i pro tzv. reversní inženýrství, které se zabývá zpětným vytvářením modelů ze zdrojového kódu. Nejznámější zástupci CASE nástrojů jsou Enterpise Architect, MagicDraw a Powerdesigner. Další nástroje a jejich vlastnosti lze dohledat v [7].

6.8 Shrnutí

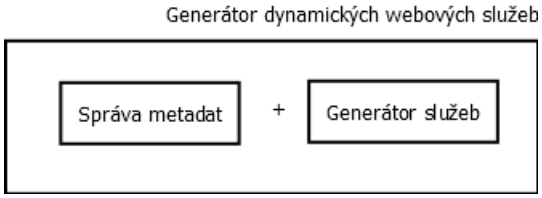
Jack Harrington ve své publikaci [3] zastává názor, že strojově generovaný kód však nelze použít vždy a za všech okolností. Použití generátoru se předpokládá u projektů s určitou stálostí architektury, rozhraní nebo kódu obecně. S každým velkým zásahem do projektu používajícího nějaký generátor roste pravděpodobnost výskytu chyb a nutnost úpravy generátoru. Je nutné mít na paměti, že generovaný kód je pouze podpůrný prostředek a nikdy nedokáže plně nahradit dobře napsaný kód přímo programátorem. Vždy bude existovat programový kód, který musí být napsán ručně a poté jej lze doplnit kódem vygenerovaným.

7 Dynamické webové služby

V předchozím textu (5.6) byl popsán proces ruční tvorby webových služeb za pomocí frameworku Apache CXF. V této kapitole je popsána architektura a implementace nové webové služby (dále jen generátor služeb), která dokáže tento proces zajistit sama bez zásahu programátora.

Aby byly nějakým způsobem odlišeny služby vytvořené ručně programátorem a služby, které vytvořil generátor služeb, budou se služby vytvořené generátorem nazývat **dynamické**.

Kompletní aplikace diplomové práce je rozdělena na dvě části: **Správa metadat** a **Generátor služeb**. Souhrně je tato aplikace nazvána jako **Generátor dynamických webových služeb**.



Obrázek 7.1: Projekt diplomové práce

7.1 Správa metadat

V rámci předmětu KIV/OPSWI byla vytvořena webová J2EE aplikace umožňující tvorbu tzv. **metadat**. Tato aplikace se nazývala opswi, ale v rámci diplomové práce byla přejmenována na **SQLServiceManager**.

7.1.1 Metadata jako popis webové služby

Účelem aplikace vytvořené v rámci předmětu KIV/OPSWI bylo generování XML dokumentů – **metadat**, která by popisovala webovou službu, její operace a hlavně jejich provázanost s SQL dotazy. Tato **metadata** měla sloužit jako vstup pro generátor webových služeb. V rámci diplomové práce byla tato aplikace rozšířena o některé nové funkce.

7.1.2 Předchozí stav

Původní verze aplikace poskytovala tyto základní funkce:

- Definice operací/metod pomocí SQL dotazů.
- Definice argumentů a jejich datových typů pro jednotlivé operace/metody.
- Nastavení přístupu k operaci/metodě pomocí uživatelských rolí.
- Nastavení komentářů (účel operace, popis vstupů apod.).
- Ukládání XML metadat na disk.
- Přihlášení/Odhlášení uživatelů.

Metadata popisující operace (metody) webové služby se ukládaly fyzicky přímo na disk a jakýkoliv přihlášený uživatel pak mohl metadata prostřednictvím aplikace znovu načíst a libovolně měnit. Navíc každý XML dokument s metadaty obsahoval pouze **jednu** operaci.

7.1.3 Současný stav

Stav, kdy se XML metadata ukládala na disk, byl nežádoucí, a proto bylo rozhodnuto o tom, že se data budou ukládat do databáze. Pro tyto účely vystačila jedna tabulka, ve které jsou uloženy informace o tvůrci webové služby společně s XML metadaty popisující službu. V této tabulce jsou rovněž uloženy informace jednoznačně identifikující službu (id), její název a stav.

Příklad XML metadat popisující službu je zobrazen v příloze D.

V rámci diplomové práce byla aplikace rozšířena o následující funkce:

- Definice dynamické webové služby (název, popis činnosti služby).
- Přiřazení operací/metod k příslušné službě.
- CRUD operace nad službami a metodami.
- Ukládání XML metadat do databáze.

- Nápověda.
- Komunikace s generátorem služeb.

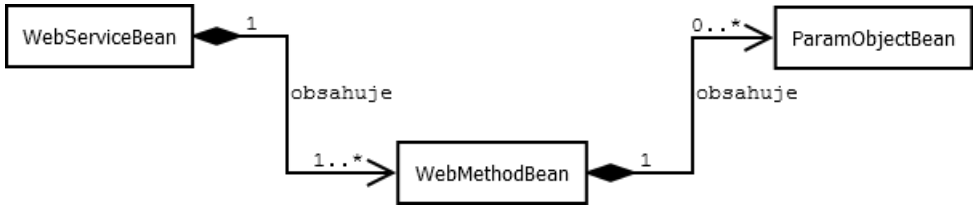
7.1.4 Architektura

Celá aplikace vytvářející metadata (SQLServiceManager) je postavena na Spring MVC architektuře.

Model

Model v této aplikaci představují Java Beans objekty (doménové objekty), které jsou instancemi tříd `WebServiceBean`, `WebMethodBean` a `ParamObjectBean`.

Operaci (metodu) webové služby reprezentuje objekt `WebMethodBean`. Každý objekt `WebServiceBean` musí obsahovat alespoň jednu operaci. Každá operace pak obsahuje 0 až N parametrů, které jsou získávány z SQL dotazu.



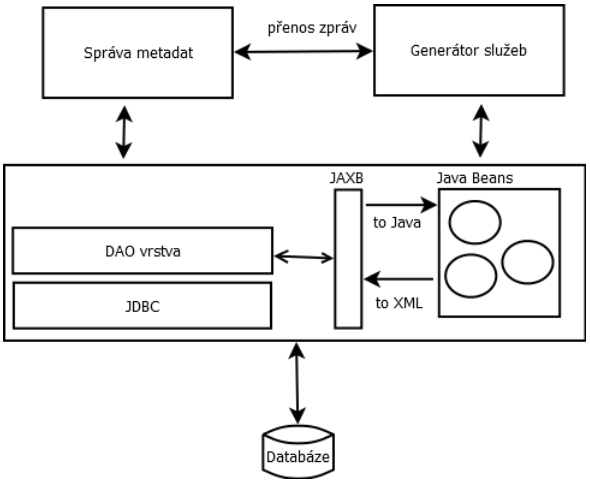
Obrázek 7.2: Vazba mezi datovými objekty.

Kompletní datový model je pomocí JAXB převáděn dle potřeby na XML metadata, která jsou ukládána do databáze. Při načítání z databáze jsou XML metadata opět pomocí JAXB převedena na datové objekty.

Controller

Aplikační logiku řeší množina Controllerů vytvořených už v oborovém projektu. V rámci diplomové práce došlo pouze k rozšíření o `IndexController`, který má za úkol předat příslušnému pohledu (JSP stránce) seznam objektů

typu `WebServiceBean`, které vlastní právě přihlášený uživatel. Aplikační logika byla rozšířena rovněž o `DeployController`, který řeší komunikaci s generátorem služeb.



Obrázek 7.3: Komunikace správy služeb a generátoru.

Komunikace probíhá prostřednictvím HTTP protokolu. `DeployController` volá generátor jako webovou službu a předá jeho operaci **createWebService** číslo, které identifikuje metadata webové služby, kterou chceme vygenerovat. Jako odpověď dostane od generátoru JSON zprávu s výsledkem generování. Obsah této zprávy je pak zobrazen uživateli, aby věděl, jak vše dopadlo.

Listing 7.1: Příklad JSON zpráv od generátoru.

```
{"status": "OK", "message": "C:/dip/generated/WSService_1.jar"}
{"status": "ERR", "message": "Chyba pri prekladu sluzby s~
   cislem 1."}
```

View

Prezentační (View) vrstva je řešena pomocí JSP stránek. Nejdůležitější stránky jsou: `Index.jsp`, `Testsql.jsp` a `Help.jsp`. `Index.jsp` se stará o zobrazení nasazených či rozpracovaných dynamických služeb.

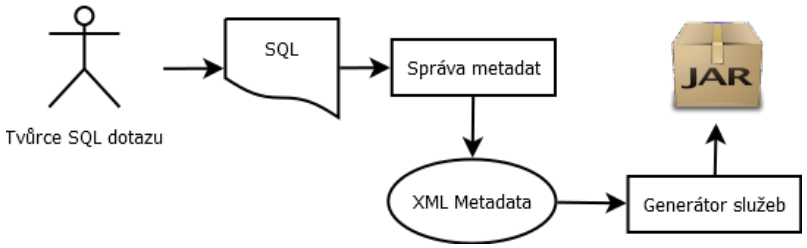
`Testsql.jsp` slouží uživateli k tomu, aby mohl vkládat SQL dotazy a ostatních potřebné parametry. Pomocí této stránky se vytváří objekty typu `WebMethodBean` a vkládají se pod objekt `WebServiceBean`.

`help.jsp` zobrazuje jednoduchou nápovědu pro obsluhu aplikace.

7.2 Generátor služeb

Před samotným vývojem služby byla provedená analýza dostupných nástrojů vhodných ke generování zdrojového kódu. Ze všech možností (viz 6.3) byl vybrán nástroj **FreeMarker**. V úvahu přicházel i nástroj Velocity, který má podobné vlastnosti, ale šablony napsané v syntaxi FreeMarkeru jsou čitelnější a přehlednější. Technologie XSLT byla rovněž zamítnuta z důvodu špatně čitelnosti šablon. Generování kódu z UML by bylo příliš složité a zbytečně komplikované.

Generátor služeb je implementován jako webová služba generující (uživatelské) dynamické webové služby z dostupných XML metadat. Cílem generátoru je vytvoření JAR archívu reprezentujícího nasaditelnou a spustitelnou webovou službu.



Obrázek 7.4: Proces tvorby JAR archívu.

7.2.1 Architektura generátoru

Architektura generátoru vychází ze základní třívrstvé architektury sestavené z prezntační, aplikační a datové vrstvy. Generátor nepotřebuje prezentační vrstvu a datová vrstva byla vyřešena v aplikaci spravující metadata (SQL-ServiceManager).

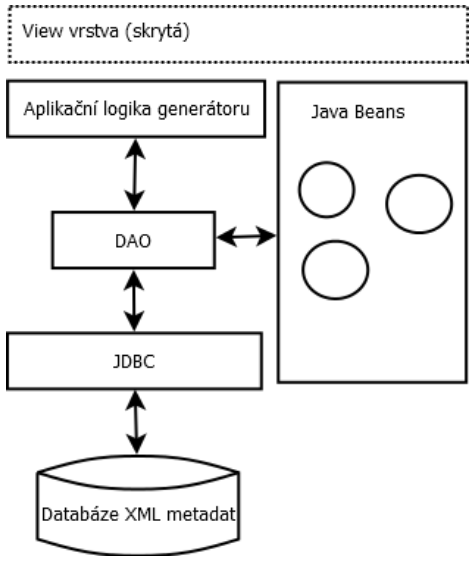
Generátor služeb poskytuje dvě základní operace (vytvoření a zrušení) pro práci s dynamickou službou:

Listing 7.2: Rozhraní služby generátoru a jeho operace.

```
public interface GeneratorService
{
@Get
@WebResult(name = "createWebService")
public ExportResultBean createWebService(@WebParam(name = "
    serviceId") BigDecimal serviceId);

@Get
@WebResult(name = "removeWebService")
public ExportResultBean removeWebService(@WebParam(name = "
    serviceId") BigDecimal serviceId);
}
```

Generátor služeb používá stejný datový model i přístup do databáze jako aplikace ke správě metadat (SQLServiceManager).



Obrázek 7.5: Architektura generátoru.

7.2.2 Controller

O vnitřní logiku generátoru se stará třída `WebServiceManagerImpl`, která spouští proces generování, a navíc dokáže nasazovat již existující dynamické služby (JAR archívy) v případě, že dojde k restartování aplikačního serveru. Po restartování aplikačního serveru nejsou dynamické služby zobrazeny automaticky jako ty klasické.

7.2.3 Proces generování služby

Proces generování služby je rozdělen do několika fází, kde každá fáze musí proběhnout bez chyby. Pokud dojde v jakékoliv fázi k chybě (výjimce), pak je celý proces znehodnocen a nelze vytvořit ani nasadit dynamickou službu.

Generátor si na základě čísla `serviceId` stáhne z databáze příslušná XML metadata a převede do podoby datových objektů pomocí JAXB. Tyto objekty pak použije jako zdroj dat (model) pro celý proces generování.

Inicializace

V této fázi se pomocí Ant skriptu připraví adresáře pro zdrojové kódy, FreeMarker šablony a sestavovací Ant skript build.xml. Tyto adresáře se vytvářejí v dočasném adresáři (temp) a po dokončení procesu se mažou.

Listing 7.3: Příprava adresářů.

```
String projectDir = ExtractFileUtils.createTempDirectory().
   getAbsolutePath() + ExtractFileUtils.FILE_SEPAR;

ExtractFileUtils.initDirStructures(projectDir);
```

Generování kódu

V této fázi se do připravených adresářů generují zdrojové kódy a XML soubory s konfigurací dynamické služby. Rovněž se vygeneruje samostatný soubor s SQL dotazem.

Nejprve se vygeneruje třída s rozhraním služby, pak její implementace, DAO rozhraní a rovněž jeho implementace. Nakonec se vygenerují XML soubory nutné pro registraci SOAP a REST endpointu (viz 5.6.5).

V příloze G je příklad vygenerovaného kódu a kódu, který byl napsán ručně.

Překlad kódu a sestavení služby

V této fázi se spouští pomocí nástroje **ProjectHelper** připravený skript **build.xml** (viz příloha F). V něm jsou definované cíle (targets), které se mají provést. Nejprve se vykoná cíl **compile**, který přeloží vygenerované Java soubory a z nich se poté sestaví výsledný JAR soubor. Tento výsledný soubor se poté překopíruje do adresáře, který je uveden konfiguračním souboru webové aplikace.

Následující kód ukazuje postup spouštějící cíle definované v **build.xml**. Nástroji ProjectHelper se musí říct, kde se nachází sestavovací skript `build.xml`, zdrojové Java soubory a podpůrné knihovny nutné pro překlad a sestavení JAR archívu.

Listing 7.4: Nastavení ProjectHelperu

```
Project projectToCompile = new Project();
projectToCompile.setUserProperty(PROPERTY_ANT_FILE, buildFile.
    getAbsolutePath());
projectToCompile.setUserProperty(PROPERTY_PROJECT_DIR,
    projectDir);
projectToCompile.setUserProperty(PROPERTY_INPUT_LIB_DIR,
    libPath);
projectToCompile.setUserProperty(PROPERTY_JAR_NAME, Constants.
    WS_SERVICE_PREFIX + serviceId + ".jar");

projectToCompile.init();
ProjectHelper anteHelper = ProjectHelper.getProjectHelper();
projectToCompile.addReference(PROPERTY_ANT_PROJECT_HELPER,
    anteHelper);
projectToCompile.addBuildListener(antLogger);
anteHelper.parse(projectToCompile, buildFile);
projectToCompile.executeTarget(projectToCompile.
    getDefaultTarget());
```

Nasazení a zrušení služby

Nasazení služby (JAR souboru) za běhu aplikačního serveru se provádí pomocí třídy `DynamicServiceManager` a jeho metody `publishEndpointFromJarFile`, která přijímá cestu k JAR souboru jako argument. Pokud se podaří službu úspěšně nasadit, pak jako návratovou hodnotu vrací `true`, jinak `false`.

Zrušení („odnasazení“) se provádí pomocí metody `removeEndpointFromJarFile`.

Třídu `DynamicServiceManager` dodal zadavatel práce, jelikož řešení tohoto problému nebylo obsahem této práce.

7.3 Testovací provoz

Aplikace SQLServiceManager společně s generátorem služeb byla v průběhu práce nasazena nad tzv. **demo** databází. Tato aplikace prošla testovacím provozem a na základě některých připomínek od vedoucího práce a ostatních zainteresovaných osob (administrátoři STAGu) byly odstraňovány chyby či přidávány požadované funkce.

Poté, co byla aplikace uznána jako vyhovující a bezproblémová, byla nasazena do reálného provozu. Kompletní otestovaná aplikace se nachází na adrese `https://stag-demo.zcu.cz/sqlmanager/`.

8 Závěr

8.1 Shrnutí práce

Seznámil jsem se základními standardy a zásadami, které se týkají webových služeb. Získané poznatky jsem shrnul v kapitolách 1 až 4. V těchto kapitolách jsem vysvětlil pojmy XML, XSD, SOAP, WSDL, REST, a jejich vztah k webovým službám.

V první polovině 5. kapitoly jsem stručně popsal aplikaci „Webové služby nad IS/STAG“ a dal do souvislosti SQL dotaz s webovou službou. Druhá polovina kapitoly popisuje návod pro vytvoření webové služby společně s potřebnými nástroji a frameworkem Apache CXF.

Prostudoval jsem si možnosti strojově generovaného kódu a hledal existující nástroje, které lze využít jako generátory zdrojového kódu. V kapitole 6 jsem zpracoval přehled těchto nástrojů a popsal základní pravidla pro generování.

Kapitola 7 zdůvodňuje výběr nástroje FreeMarker a popisuje implementaci výsledné aplikace. Na základě poznatků z kapitol 5 a 6 jsem navrhl a implementoval modul, který publikuje SQL dotazy jako dynamické webové služby přístupné na webovém rozhraní. Výslednou aplikaci jsem nasadil na provozní databázi IS/STAG, otestoval a připravil k reálnému provozu.

8.2 Přínosy

Hlavním přínosem pro CIV je kompletní aplikace samotná, jelikož dokáže ušetřit čas (i peníze), který by programátor strávil psaním neustále se opakujícím rutinním kódem. Po určitém školení by mohl aplikaci používat kdokoliv i se základní znalostí SQL dotazů.

Díky této práci jsem se naučil vytvářet webové služby a pochopil, jaké výhody i nevýhody může mít strojově generovaný kód. Rovněž jsem si prohloubil znalosti z oblasti J2EE.

Seznam použitých zkratek

XML Extensible Markup Language

XSD XML Schema Definition

WSDL Web Services Description Language

SOAP Simple Object Access Protocol

REST Representational State Transfer

UDDI Universal Description Discovery and Integration

SQL Structured Query Language

HTTP Hypertext Transfer Protocol

JAR Java ARchive

JSON JavaScript Object Notation

XSLT eXtensible Stylesheet Language Transformations

UML Unified Modeling Language

CASE Computer Aided Software Engineering

MVC Model View Controller

Literatura

[1] Tým CIV. *Webové služby nad IS/STAG* [online]. 2013 [cit. 2013-04-18]. Dostupné z: https://is-stag.zcu.cz/napoveda/web-services/ch10s02.html

[2] HEROUT, Pavel. *Java a XML*. 1. vyd. České Budějovice: Kopp, 2007, 313 s. ISBN 978-80-7232-307-4

[3] HERRINGTON, Jack. *Code generation in action*. 1. vyd. Greenwich, CT: Manning, c2003, xxvi, 342 p. ISBN 19-301-1097-9

[4] The Eclipse Foundation. *Maven Integration (m2e)* [online]. 2013 [cit. 2013-05-05]. Dostupné z: http://www.eclipse.org/m2e

[5] SCHNEIDER, František. *Webová aplikace určená k získání informací o SQL dotazech* [online]. 2013 [cit. 2013-06-15], 22s. Dostupné z: http://home.zcu.cz/~frschnei/OPSWI_Dokumentace.pdf

[6] BRITTENHAM, Peter. *Understanding WSDL in a UDDI registry* [online]. 2013 [cit. 2013-04-18]. Dostupné z: http://www.ibm.com/developerworks/webservices/library/ws-wsdl/

[7] KYSELA, J., T. PETŘÍK, V. KOLÁŘ, L. KUBĚNA, A. KOTCHEEV A I. CENIGOVÁ. *Nástroje pro vývoj aplikací a jejich vazba na CASE* [online]. 2007 [cit. 2013-06-15]. Dostupné z: http://panrepa.org/CASE/zima2007/ide_case_zima2007.pdf

[8] The Eclipse Foundation. *Eclipse Web Tools Platform Project* [online]. 2013 [cit. 2013-06-15]. Dostupné z: http://projects.eclipse.org/projects/webtools

[9] KOSEK, Jiří. *Využití webových služeb a protokolu SOAP při komunikaci* [online]. 2008 [cit. 2013-04-26]. Dostupné z: http://www.kosek.cz/diplomka/html/websluzby.html

[10] PAPAZOGLOU, Michael P. *Web services: principles and technology*. Harlow: Pearson Education, 2008, xxxii, 752 s. ISBN 978-0-321-15555-9

[11] RICHARDSON, Leonard a Sam RUBY. *RESTful web services: principles and technology*. 1st ed. Sebastopol: O'Reilly, 2007, xxiv, 419 s. ISBN 978-0-596-52926-0

[12] SUN, Bruce. *A multi-tier architecture for building RESTful Web services* [online]. 2009 [cit. 2013-04-26]. Dostupné z: `http://www.ibm.com/developerworks/web/library/wa-aj-multitier`

[13] FIELDING, Thomas Roy. *Architectural Styles and the Design of Network-based Software Architectures* [online]. 2000 [cit. 2013-04-26]. Dostupné z: `http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm`

[14] ELKSTEIN, Dr M. *Learn REST* [online]. 2008 [cit. 2013-04-26]. Dostupné z: `http://rest.elkstein.org/`

[15] Tým XDoclet. *XDoclet* [online]. 2005 [cit. 2013-05-01]. Dostupné z: `http://xdoclet.sourceforge.net/xdoclet/index.html`

[16] WALLS, Craig a Norman RICHARDS. *XDoclet in action: principles and technology*. 1st ed. Greenwich, CT: Manning, c2004, xxxii, 591 p. ISBN 19-323-9405-2

[17] MONNOX, Alan. *Applying Code Generation Techniques to the J2EE Development* [online]. 2005 [cit. 2013-05-01]. Dostupné z: `http://www.informit.com/articles/article.aspx?p=389718&seqNum=4`

[18] LI, Sing. *Reduce code bloat with XDoclet* [online]. 2004 [cit. 2013-05-01]. Dostupné z: `http://www.ibm.com/developerworks/java/library/j-xdoclet`

[19] NACCARATO, Giuseppe. *Template-Based Code Generation with Apache Velocity* [online]. 2004 [cit. 2013-05-02]. Dostupné z: `http://www.onjava.com/pub/a/onjava/2004/05/05/cg-vel1.html`

[20] HERRINGTON, Jack. *Code-Generation Techniques for Java* [online]. 2003 [cit. 2013-05-02]. Dostupné z: `http://www.onjava.com/pub/a/onjava/2003/09/03/generation.html`

[21] KESWANI, Deepak. *FreeMarker vs Velocity (FTL vs VM)* [online]. 2011 [cit. 2013-05-02]. Dostupné z: `http://deepak-keswani.blogspot.cz/2011/09/freemarker-vs-velocity-ftl-vs-vm.html`

[22] Tým Oracle. *The Java EE 5 Tutorial* [online]. 2010 [cit. 2013-06-06]. Dostupné z: `http://docs.oracle.com/javaee/5/tutorial/doc/javaeetutorial5.pdf`

[23] Tým CIV. *Webové služby nad IS/STAG* [online]. 2013 [cit. 2013-05-02]. Dostupné z: `https://stag-demo.zcu.cz/ws/help`

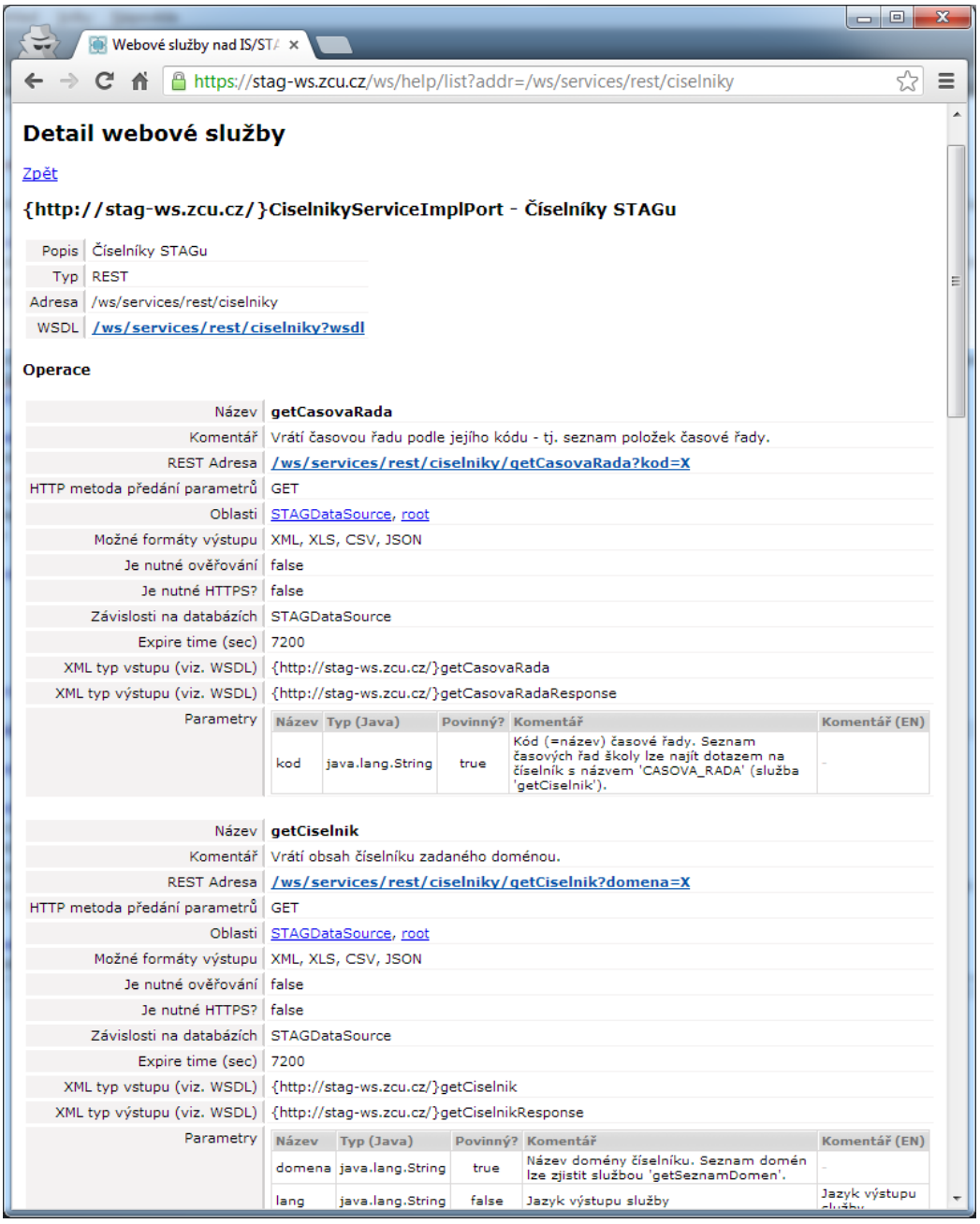
[24] SUDA, Brian. *SOAP Web Services* [online]. Edinburgh, 2003 [cit. 2013-05-02]. Diplomová práce. University of Edinburgh Dostupné z: `http://suda.co.uk/publications/MSc/brian.suda.thesis.pdf`

[25] WALLS, Craig a Ryan BREIDENBACH. *Spring in action: principles and technology*. 1st ed. Greenwich [Conn.]: Manning, 2005, xxviii, 444 p. ISBN 19-323-9435-4

[26] GUNZER, Harvig. *Introduction to Web Service* [online]. 2002 [cit. 2013-04-26]. Dostupné z: `http://www.daimi.au.dk/~thomasr/Wearable/intro_to_web_services_wp.pdf`

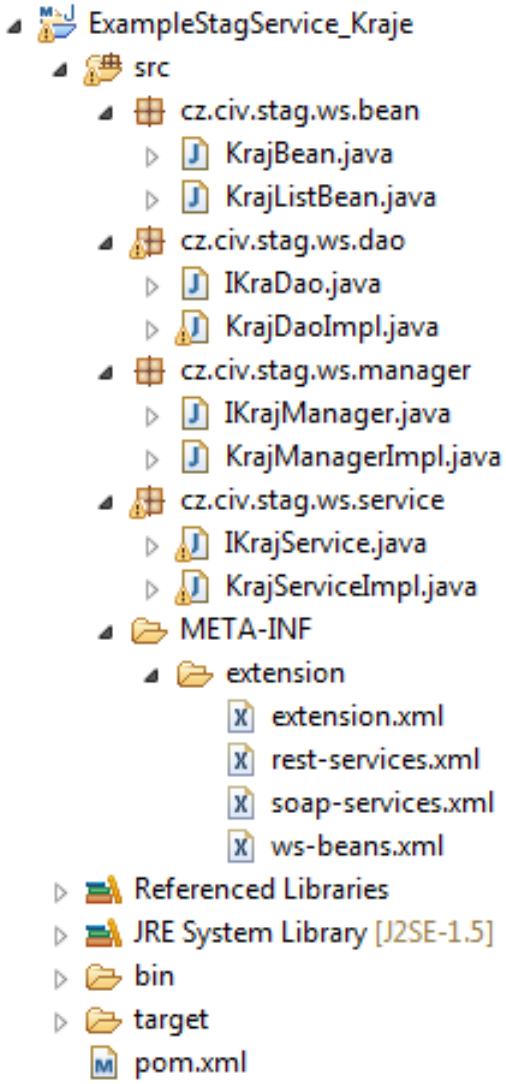
[27] LOUGHRAN, Steve, Erik HATCHER. *Ant in action: principles and technology*. 2nd ed. Greenwich, CT: Manning, c2007, xxxii, 566 p. ISBN 19-323-9480-X

A Příloha – Detail webové služby



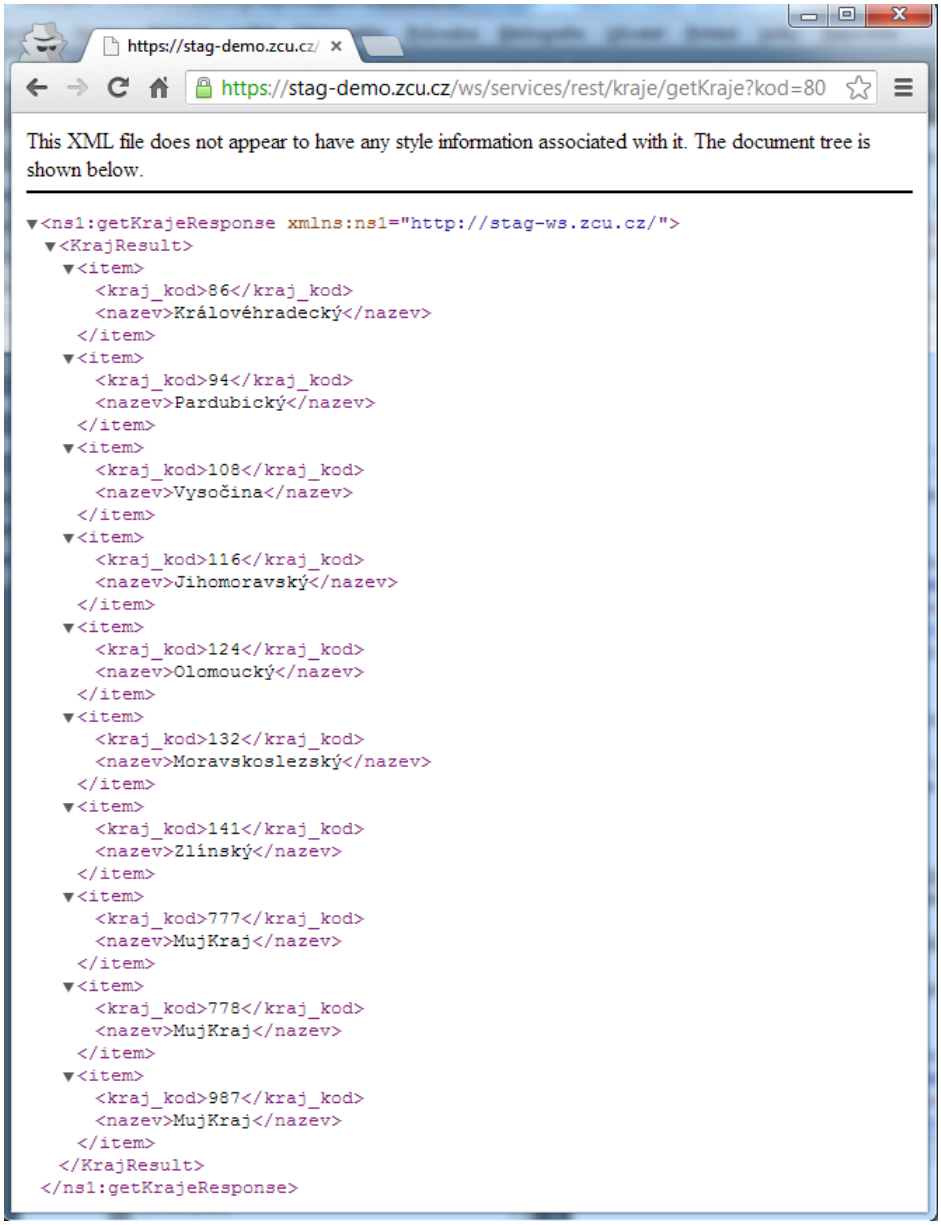
Obrázek A.1: Detail webové služby

B Příloha – Struktura ukázkové služby v Eclipse



Obrázek B.1: Ukázková služba v Eclipse.

C Příloha – Volání ukázkové webové služby



Obrázek C.1: Volaná ukázková služba ve webovém prohlížeči.

D Příloha – Struktura metadat

Listing D.1: Popis služby pomocí XML metadat.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<WebService>
	<description>Sluzba pro praci s~kraji</description>
	<id>1</id>
	<name>kraje</name>
	<status>A</status>
	<webMethods>
	 <method>
		<name>getKraje</name>
		<beanType>WSBean1_Method0</beanType>
		<columns>
			<column>KRAJ_KOD</column>
			<column>NAZEV</column>
		</columns>
		<description></description>
		<params />
		<roles />
		<sql>select kraj_kod, nazev from kraje</sql>
		<sqlType>select</sqlType>
		</method>
	</webMethods>
	<xml></xml>
</WebService>
```

E Příloha – Šablona pro rozhraní služby

Listing E.1: FreeMarker šablona pro rozhraní služby.

```
@WebService(targetNamespace = WSConstants.
   DEFAULT_SERVICE_NAMESPACE)
@StagComment("${webService.description}")
@DBDependency("STAGDataSource")
public interface ${className} {
<#list webService.webMethods as method>
       <@setGetType method=method />
       @RequestWrapper(targetNamespace = WSConstants.
          DEFAULT_TYPES_NAMESPACE)
       @ResponseWrapper(targetNamespace = WSConstants.
          DEFAULT_TYPES_NAMESPACE)
       @StagMethod(authentication = ${method.isRole()}, roles =
            "${method.roles}", expireTime = StagMethod.
          EXPIRE_TIME_ALWAYS)
       <#if method.sqlType == "select">
       @WebResult(name = "${method.beanType}Result")
       @StagComment("${method.description}")
       public <@setReturnType method=method/>${method.name}(
       <#list method.params as param>
              @WebParam(name = "${param.javaName}")
                 @StagComment("${param.description}") <
                 @setOptional param=param/> <@setLike param=
                 param/> <@setDomain param=param/> ${param.
                 javaName}<#if param_has_next>, </#if>
       </#list>
       );
       <#else>
       @WebResult(name = "upload_result")
       @StagComment("${method.description}")
       public <@setReturnType method=method/>${method.name}(
          @WebParam(name="stagUser") @StagUser String stagUser
          , ${method.beanType}ListBean listBean);
       </#if>
 </#list>}
```

F Příloha – Sestavovací Ant skript

Listing F.1: Sestavovací Ant skript dynamických služeb.

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="StagServiceBuilder" default="dist">
	<property name="src.dir" value="src" />
	<property name="build.dir" value="bin" />
	<property name="dist.dir" value="dist" />
	<property name="lib.dir" value="${input.lib.dir}" />
	<property name="meta.dir" value="${src.dir}/META-INF/
	   dynamic-extension" />
	<!-- ================ compiling the project
	   ==================== -->
	<target name="compile" depends="clean">
		<mkdir dir="${build.dir}" />

		<path id="compile.lib.dir">
			<fileset dir="${lib.dir}">
				<include name="**/*.jar" />
			</fileset>
		</path>
		<javac debug="true" compiler="extJavac" srcdir="
		   ${src.dir}" destdir="${build.dir}" encoding="
		   utf-8">
		     <compilerarg value="-Xlint" />
			<classpath refid="compile.lib.dir" />
		</javac>
		<copy todir="${meta.dir}">
		  <fileset dir=".">
		    <include name="**/*.xml"/>
			<exclude name="build.xml"/>
		  </fileset>
		</copy>
	</target>
	<!-- ================ creating distribution file
	   ==================== -->
	<target name="dist" depends="compile">
		<mkdir dir="${dist.dir}" />
		<!-- creating jar file -->
```

```
            <jar jarfile="${dist.dir}/${dist.jar}" compress=
                "true">
                    <fileset dir="${build.dir}">
                            <include name="**/*.*" />
                    </fileset>
                    <fileset dir="${src.dir}">
                            <include name="**/*.*" />
                    </fileset>
            </jar>
    </target>
    <!-- ================ the clean target
        ========================== -->
    <target name="clean">
            <delete dir="${dist.dir}" failonerror="false" />
            <delete dir="${build.dir}" failonerror="false" /
                >
    </target>

    <!-- ============== running the application
        ====================== -->
    <target name="run">
            <java classname="${mainclass}" fork="yes">
                    <classpath location="${dist.dir}/${dist.
                        jar}" />
            </java>
    </target>

    <path id="tests.classpath">
            <pathelement location="${build.dir}" />
            <fileset dir="${lib.dir}">
                    <include name="*.jar" />
            </fileset>
    </path>
</project>
```

G Porovnání generovaného a ručního kódu

```
public List<WSBean6_Method0>  getKraj(java.lang.Integer kod)
{
    ArrayList<WSBean6_Method0> beans = new ArrayList<WSBean6_Method0>();

     HashMap m = new HashMap();
        m.put("kod", kod);

        List<Map<String, Object>> rows = getNamedParameterJdbcTemplate().queryFor
        for (Map row : rows)
        {
            WSBean6_Method0 bean = new WSBean6_Method0();

            String resultAsString = "";

                resultAsString = ""+row.get(WSBean6_Method0.PROPERTY_Kraj_kod);
                if(!resultAsString.equals("null"))
                {
                    bean.setKraj_kod(""+resultAsString);
                }
                resultAsString = ""+row.get(WSBean6_Method0.PROPERTY_Nazev);
                if(!resultAsString.equals("null"))
                {
                    bean.setNazev(""+resultAsString);
                }

            beans.add(bean);
        }

    return beans;
}
```

```
public List<KrajBean>  getKraje(int kod)
{
    ArrayList<KrajBean> beans = new ArrayList<KrajBean>();

     Map m = new HashMap<String, Object>();
     m.put("kod", kod);

        List<Map<String, Object>> rows = getNamedParameterJdbcTemplate().queryFor
        for (Map row : rows)
        {
            KrajBean bean = new KrajBean();

            String resultAsString = "";

                resultAsString = ""+row.get(KrajBean.PROPERTY_KRAJ_KOD);
                if(!resultAsString.equals("null"))
                {
                    bean.setKraj_kod(""+resultAsString);
                }
                resultAsString = ""+row.get(KrajBean.PROPERTY_NAZEV);
                if(!resultAsString.equals("null"))
                {
                    bean.setNazev(""+resultAsString);
                }

            beans.add(bean);
        }

    return beans;
}
```

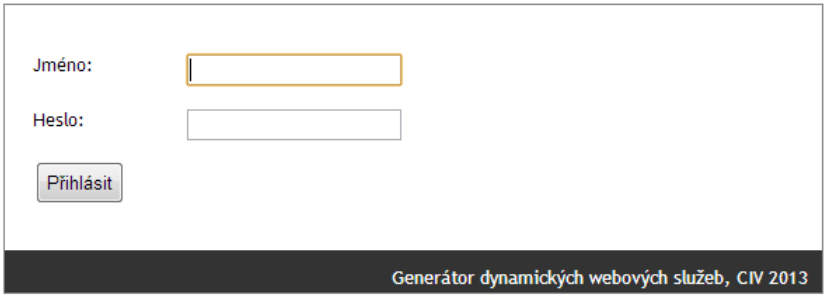
Obrázek G.1: Porovnání generovaného a ručně vytvořeného kódu

H Uživatelská příručka

Výsledná aplikace „Generátor dynamických webových služeb“ je zpřístupněna na adrese `https://stag-demo.zcu.cz/sqlmanager/`.

H.1 Přihlášení

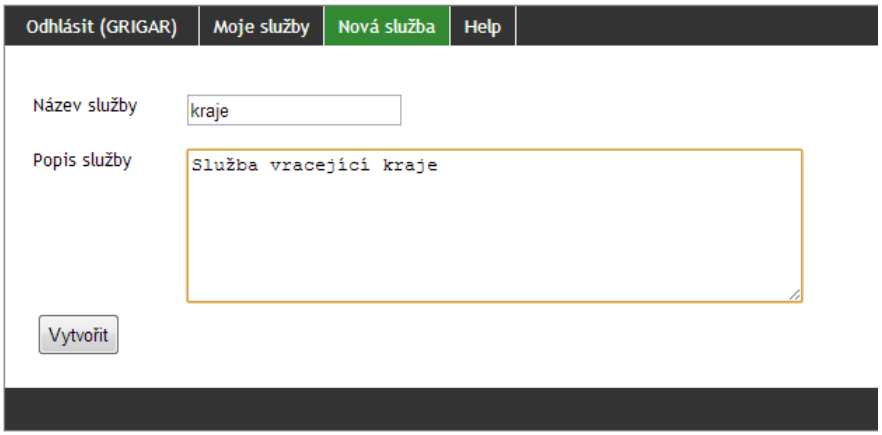
Pro přístup k celé aplikaci je nutné přihlášení pomocí uživatelského jména a hesla. Pro testovací účely lze použít uživatele *grigar* s univerzálním heslem **demo**.



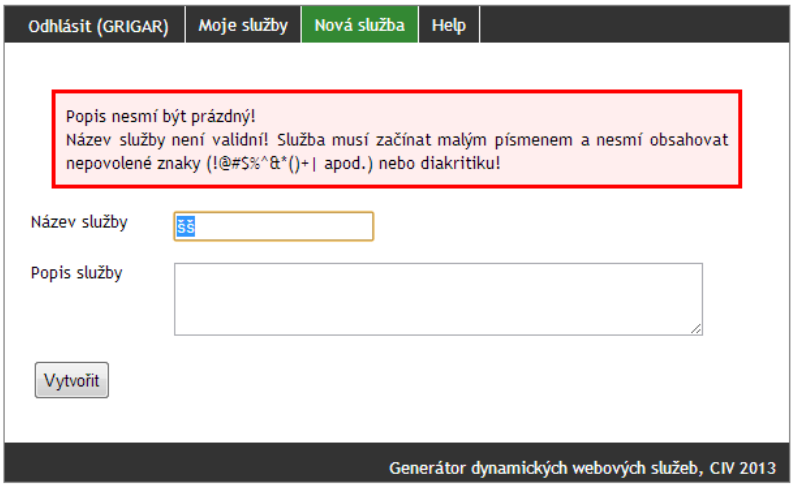
Obrázek H.1: Přihlašovací formulář.

H.2 Nová služba

Po kliknutí na položku **Nová služba** v horním menu se zobrazí formulář pro vytvoření nové dynamické služby. Do tohoto formuláře se zadává název a popis služby. Oba údaje jsou povinné. Pro název navíc platí, že musí být uveden malými písmeny a nesmí obsahovat nepovolené znaky (speciální znaky a diakritika). Pokud toto není dodrženo, pak je zobrazena chybová hláška.



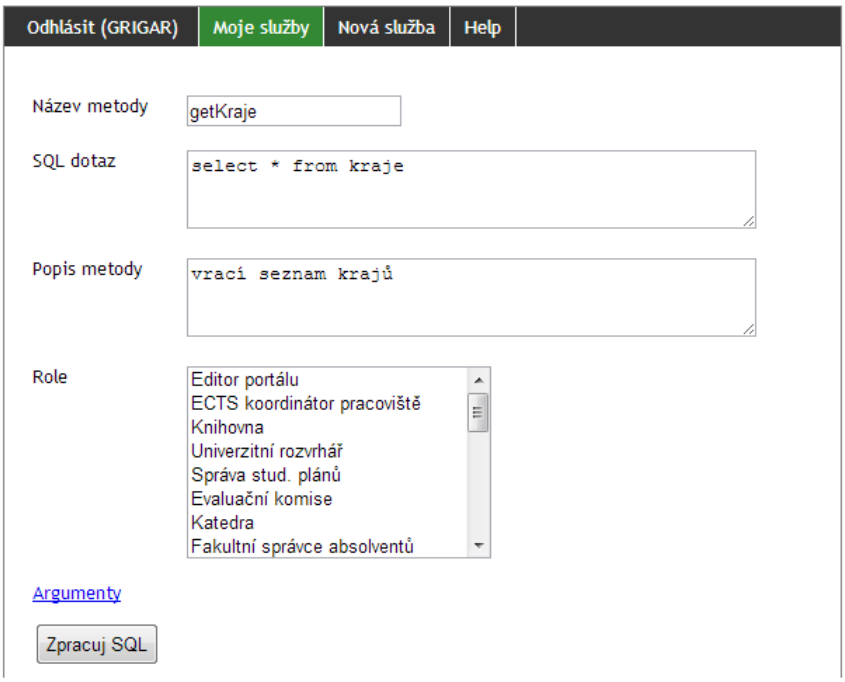
Obrázek H.2: Formulář pro vytvoření nové služby.



Obrázek H.3: Chybný název a nevyplněný popis služby.

H.3 Nová metoda

Poté, co je vytvořen název a popis služby, lze vytvořit popis operace/metody. Po kliknutí na odkaz **Nová metoda** se zobrazí formulář pro vkládání SQL dotazu a ostatních doplňujících informací. Ke každé operaci lze přiřadit 0 až N rolí uživatelů, kteří smí operaci/metodu volat.



Obrázek H.4: Formulář pro vytvoření nové metody.

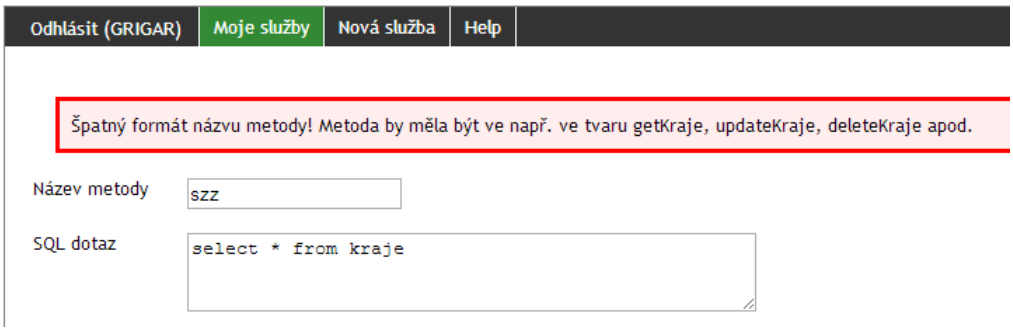
Stiskem tlačítka **Zpracuj SQL** dojde ke zpracování a vykonání SQL dotazu. Výsledkem může být žádost o vložení vstupních parametrů pro SQL dotaz (pokud nějaké obsahuje) nebo je zobrazena přímo tabulka zobrazující výsledek volání. Pokud je vše v pořádku a není nutná žádná další informace či chybně vyplněný údaj, pak se zobrazí v zeleném rámečku potvrzení o úspěšném vytvoření metody.

V případě špatného pojmenování metody nebo jiné chyby je zobrazena chybová hláška.

KRAJ_KOD	NUTS3	NAZEV	ZKRATKA	STAV	VZNIK_DNE	VZNIK_INFO	ZANIK_DNE	ZANIK_INFO
19	CZ010	Hlavní město Praha	Hl. m. Praha	1	2000-01-01 00:00:00.0			
27	CZ020	Středočeský	Středočeský	1	2000-01-01 00:00:00.0			
35	CZ031	Jihočeský	Jihočeský	1	2000-01-01 00:00:00.0			
43	CZ032	Plzeňský	Plzeňský	1	2000-01-01 00:00:00.0			
51	CZ041	Karlovarský	Karlovarský	1	2000-01-01 00:00:00.0			
60	CZ042	Ústecký	Ústecký	1	2000-01-01 00:00:00.0			
78	CZ051	Liberecký	Liberecký	1	2000-01-01 00:00:00.0			
86	CZ052	Královéhradecký	Královéhradecký	1	2000-01-01 00:00:00.0			
94	CZ053	Pardubický	Pardubický	1	2000-01-01 00:00:00.0			
108	CZ063	Vysočina	Vysočina	1	2000-01-01 00:00:00.0			

OK. Metoda byla uložena ke službě

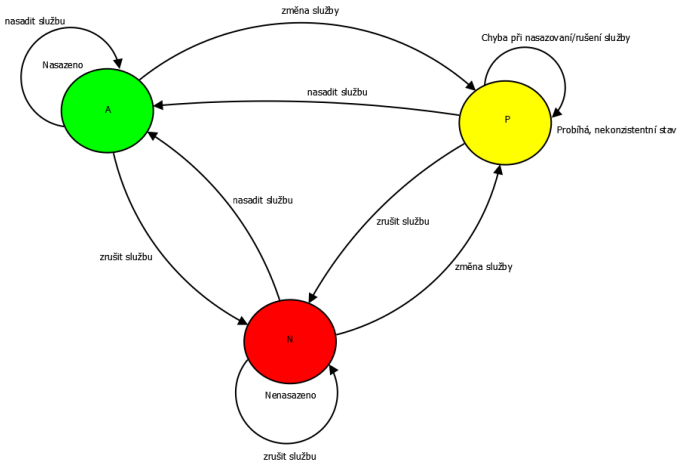
Obrázek H.5: Výsledek volání SQL dotazu.



Obrázek H.6: Špatně pojmenovaná metoda.

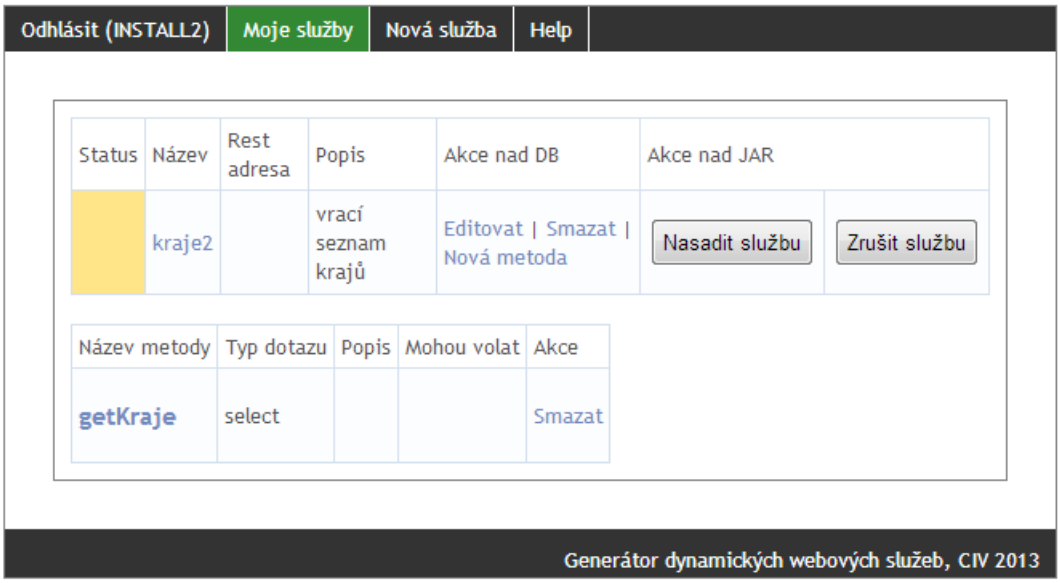
H.4 Stavy služby

Dynamické webové služby se mohou nacházet ve třech stavech: **Nasazená** (zeleně), **Zrušená** (červeně) a **Rozpracovaná** (oranžově).



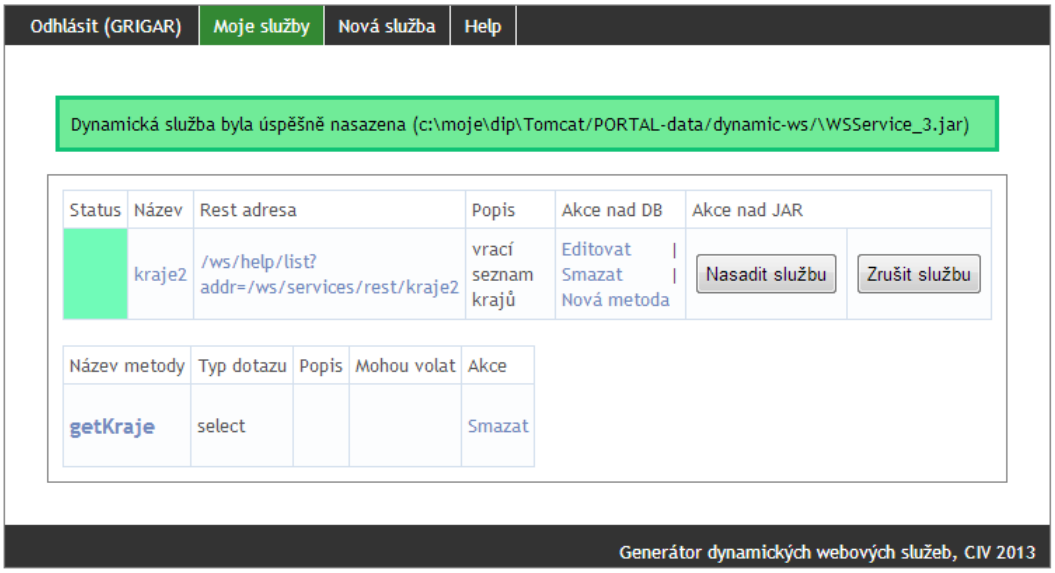
Obrázek H.7: Stavy dynamické webové služby.

Rozpracovaná služba definuje stav, kdy uživatel ještě nenasadil službu a je v rozpracovaném stavu. Do tohoto stavu se služba může dostat i v případě, že již byla nasazena, ale poté došlo k nějakým úpravám.



Obrázek H.8: Rozpracovaná služba

Nasazení služby se provede stiskem tlačítka **Nasadit službu.** Zrušení služby se provede stiskem tlačítka **Zrušit službu.**



Obrázek H.9: Nasazená služba

Obrázek H.10: Zrušená služba.

H.5 Nápověda

Vybráním položky **Help** v horním menu se zobrazí nápověda a rady pro práci s aplikací. Zde lze najít podrobnější popis všech funkcí aplikace a různá omezení, která pro některé funkce platí. Původní nápovědu lze rovněž najít v dokumentaci oborového projektu[5].

Obrázek H.11: Nápověda k aplikaci.

H.6 Chyba

Při nasazování či rušení dynamické služby může dojít k jevu, kdy je zobrazena chybová hláška typu **Chyba při nasazování služby...**. Toto je stav, který by prakticky neměl nastat, ale pokud se tak stane, je vhodné kontaktovat příslušného administrátora na CIVu a upozornit jej na tuto chybu. Většinou se jedná o omylem smazaný generátor služeb (DynamicServiceGenerator-1.0.jar).

Obrázek H.12: Chyba při nasazování služby.

I Obsah přiloženého média

Na přiloženém médiu se nacházejí tyto adresáře:

- **doc** – obsahuje tuto práci ve formátu pdf.
- **Tomcat** – obsahuje instalaci aplikačního serveru Tomcat s aplikací SQLServiceManager.
- **SQLServiceManager** – obsahuje zdrojové kódy aplikace pro správu metadat.
- **DynamicServerGenerator** – obsahuje zdrojové kódy generátoru webových služeb.
- **ExampleStagService_Kraje** – obsahuje zdrojové kódy ukázkové webové služby kraje.
- **Java** – obsahuje instalaci Javy.

Aplikační server Tomcat obsahuje pouze nasazenou aplikaci pro správu XML metadat SQLServiceManager bez generátoru služeb. Server se spouští souborem **startup.bat**, který se nachází v adresáři **Tomcat/bin**. V době spuštění serveru musí být volný port 8080 a adresář Java musí být na stejné úrovni jako adresář Tomcat.

Aplikace je po spuštění serveru zpřístupněna na adrese `http://localhost:8080/sqlmanager/`.