UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

Optimalizácia SOAP klienta pre Python

Bakalárska práca

2015
Tamás Bögi

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

Optimalizácia SOAP klienta pre Python

Bakalárska práca

Študijný program:	Informatika
Študijný odbor:	2508 Informatika
Školiace pracovisko:	Katedra Informatiky
Školiteľ:	RNDr. Tomáš Kulich, PhD

Bratislava 2015
Tamás Bögi

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Tamás Bögi
Študijný program: informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Optimalizácia SOAP klienta pre Python
Optimization of SOAP client for Python

Cieľ:
- Naštudovať projekt PySimpleSOAP
- Odstrániť niekoľko chýb súvisiacich s typovými konverziami
- Zrýchliť kód (predpokladá sa, že súčasná implementácia funguje asymptoticky neoptimálne)
- Pokúsiť sa vylepšiť kompatibilitu s Python3
- Urobiť rôzne ďalšie vylepšenia, ak sa nájdu ďalšie problémy (zlý design, chyba, chýbajúca dokumentácia)

Vedúci: RNDr. Tomáš Kulich, PhD.
Katedra: FMFI.KI - Katedra informatiky
Vedúci katedry: doc. RNDr. Daniel Olejár, PhD.

Dátum zadania: 30.10.2015

Dátum schválenia: 30.10.2015 doc. RNDr. Daniel Olejár, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Čestne prehlasujem, že som túto bakalársku prácu vypracoval samostatne s použitím citovaných zdrojov.

.....

Poďakovanie

Chcel by som poďakovať v prvom rade môjmu vedúcemu RNDr. Tomášovi Kulichovi, PhD. za návrh užitočnej témy, za rady poskytnuté pri riešení danej problematiky a za zavedenie do kolektívu, do akého som chcel vždy patriť.

Abstrakt

Webové služby sú v dnešnej dobe podobne rozšírené ako samotné webové stránky. Používajú sa na poskytovanie funkcionality napísanej v ľubovolnom programovacom jazyku a bežiacej na rôznych systémoch pre vzdialené aplikácie, v iných jazykoch a na iných systémoch. V tejto práci sme naštudovali pojmy, formáty a protokoly súvisiace s prevádzkovaním webových služieb. Vylepšili sme knižnicu PySimpleSoap, ktorú môžeme následne používať aj v komerčných produktoch, a v neposlednom rade sme získali užitočné skúsenosti, ktoré určite využijeme v našich nasledujúcich projektoch.

Kľúčové slová: Webové služby, WSDL, SOAP

Abstract

Nowadays web services are almost as widespread as the web sites we all know. They provide functionality written in arbitary programming languages running on any operating systems for other remote applications written in other languages on different systems. In this thesis we studied concepts, formats and protocols related to operating web services. We improved the PySimpleSoap library, which is now good enough to be used in commercial products, and at last but not least we acquired useful experience that will come in handy for our next projects.

Key words: Web services, WSDL, SOAP

Obsah

Úvod

„Šťastný je ten, kto nič neočakáva, pretože on nikdy nebude sklamaný“

Alexander Pope

Prečo začína bakalárska práca študenta informatiky na tému Optimalizácia SOAP klienta pre Python citátom o šťastí od anglického básnika 18. storočia? Lebo nejako začať predsa treba, a veríme tomu že daná myšlienka bude čitateľovi užitočná, a hlavne preto, lebo ako sa neskôr ukáže, ušetrilo by to hojné množstvo práce aj autorovi.

A teraz už k téme. Našou úlohou bude vylepšiť knižnicu PySimpleSoap, ktorá má slúžiť na prácu s webovými službami v jazyku Python. Na to aby sme to mohli spraviť, potrebujeme si vyjasniť zopár pojmov. V prvej krátkej kapitole budeme rozprávať o samotných webových službách. Povieme si čo sú zač, aké majú výhody a nevýhody, prečo by sme ich chceli alebo naopak nechceli využívať.

V informatike nič nie je len tak, počítače síce dokážu už rozpoznať objekty na obrázku, dokonca tvoria súvislú anglickú vetu o tom čo „vidia“, ovládajú našu domácnosť a dopravné prostriedky, uľahčujú, a v istých prípadoch sťažujú naše životy, ale nevedia, aspoň zatiaľ, rozmýšľať. Na to, aby rozumeli našim požiadavkám, ale aj jeden druhému slúžia prísne protokoly, a nie je to inak ani v prípade webových služieb. Preto druhú výrazne dlhšiu kapitolu venujeme významným protokolom.

Tretiu kapitolu zaberú implementačné detaily vybranej knižnice. Pozrieme si bližšie ako autori riešili niektoré generálne, či pre jazyk špecifické problematiky nastávajúce pri komunikácii s webovou službou, a prečo tie riešenia nefungujú tak, ako by mali. Podľa možnosti ich následne opravíme alebo vylepšíme.

V závere zhodnotíme výsledok našej práce, a porozmýšľame o budúcnosti knižnice, a o alternatívnych možnostiach.

A človek predsa málokedy robí niečo len tak bez cieľa a motivácie. Našou motiváciou okrem dokončenia štúdia je fakt, že spoločnosť pre ktorú pracujeme, a aj veľa vývojárov na svete by potrebovali knižnicu, ktorá nie len existuje, ale aj funguje.

Kapitola 1.

Webové služby

1.1. Definícia webových služieb

Internet sa stal neoddeliteľnou súčasťou nášho každodenného života. Navštevujeme webové stránky z rozličných dôvodov, slúžia nám ako forma zábavy, nekonečný zdroj informácií alebo ako pracovný nástroj. Vďaka nim ľahko zistíme aktuálne počasie na druhom konci sveta, konverzný kurz pre ľubovolné meny či kúpime lístok na vlak. Webové stránky sú však navrhnuté tak, aby sme ich vedeli snadno používať my, ľudia, a nie sú vhodné pre aplikácie. To ale vôbec nevadí, lebo práve na to sú webové služby. Sú to aplikácie poskytujúce svoje dáta a služby iným aplikáciam cez sieť, pomocou štandardizovaných formátov a protokolov. Základ webových služieb tvoria WSDL a SOAP, oba založené na XML. Prvý je jazyk na definíciu rozhrania a možnosti prístupu k danej službe, druhý je protokol, ktorý definuje ako má komunikácia prebiehať.

1.2. Výhody a nevýhody

Použitie štandardov ako XML, robí webové služby univerzálnym a nezávislým na architektúre, programovacích paradigmách a operačných systémoch. Funkcionalita poskytovaná objektovo orientovaným Java programom bežiacim na systéme Windows môže byť bez problémov využitá procedurálnou aplikáciou písanou v jazyku C++ na Linuxe.

Ďalšou výhodou webových služieb je podobná ako výhoda knižníc či modulov programovacieho jazyka. Často nemá zmysel spraviť niečo, čo už pred nami spravil niekto iný, pravdepodobne lepšie ako by sme to urobili sami. Tak isto môžeme pomocou krátkych programov spojiť funkcionalitu viacerých rôznych služieb a vytvoriť tak väčšiu, komplikovanejšiu aplikáciu. To samozrejme neznamená, že je dobrý nápad využívať

webové služby na sčítanie dvoch čísel a podobné úlohy, ktoré budeme používať ako príklady v neskorších kapitolách.

Ako to už býva, nič nie je dokonalé a bez nevýhod. Na prístup k webovým službám potrebujeme sieťové pripojenie čo nemusíme mať vždy zaručené. Môže sa nám tiež stať, že poskytovateľ služby zanikne, nestáva sa to síce často, ale nemožno to vylúčiť, zatiaľ čo program, ktorý by sme napísali sami, garantovane nezmizne zo dňa na deň.

Všestrannosť a nezávislosť na jazykoch ale neznamená, že webové služby sú v každom z nich rovnako dobre podporované. Kým v C# vieme rýchlo a ľahko vyrobiť server alebo klienta webovej služby, jeden z najpopulárnejších jazykov, Python nemá ani jednu spoľahlivú knižnicu na prácu s nimi.

Kapitola 2.

Protokoly a formáty

2.1. Protokol HTTP

Hypertext Transfer Protocol je protokolom siedmej, aplikačnej vrstvy modelu OSI, slúžiacej na výmenu a prenos hypertextových dokumentov medzi serverom a klientom, typicky implementovaný pomocou protokolu TCP, štandardne používajúc port 80. Je založený na požiadavkách a odpovediach. Klient, ktorý je vo väčšine prípadov webový prehliadač, pošle na server svoju požiadavku, na čo mu server odpovie. Formát týchto správ sa zmenil od veľmi jednoduchého až po robustnejší počas vývoja protokolu.

2.1.1. Prehľad vývoja

HTTP / 0.9

- prvá dokumentovaná verzia z roku 1991 navrhnutá počítačovým výskumníkom Tim Berners-Lee.
- metódy: GET
- v dnešnej dobe sa už nepoužíva, ale niektoré web servery, ako napríklad Apache ho ešte stále podporujú
- spojenie medzi serverom a klientom je ukončené po prenose dokumentu

Požiadavka**:**

- jediný riadok ASCII znakov
- začína slovom GET po čom nasleduje cesta k súboru
- neobsahuje žiadne iné informácie
- e.g. GET /index

Odpoveď**:**

- prúd znakov rovnakého kódovania, reprezentujúci vypýtaný HTML dokument

HTTP / 1.0

- robustnejší ako jeho predchodca, implementovaný rôznymi spôsobmi, neformálne publikovaný v roku 1996
- metódy: GET, POST, HEAD
- už neslúži len na prenos HTML súborov
- spojenie medzi serverom a klientom je ukončené po prenose dokumentu

Požiadavka**:**

- obsahuje verziu protokolu
- voliteľné hlavičky v ASCII
- voliteľné telo požiadavky

Odpoveď**:**

- hlavička obsahujúce stav odpovede, plus ďalšie voliteľné v ASCII
- telo odpovede v ľubovoľnom formáte e.g. HTML, obrázok...

HTTP / 1.1

- prvýkrát publikovaný v roku 1997, upravený v 1999
- metódy: GET, POST, HEAD, OPTIONS, PUT, DELETE, TRACE, CONNECT
- nové možnosti ako kešovanie a trvalé spojenia, nové hlavičky
- požiadavka a odpoveď majú rovnaký formát ako pri HTTP / 1.0
- používa sa aj dnes

HTTP / 2

- publikovaný v roku 2015
- binárny protokol
- podporuje kompresiu hlavičiek, multiplexovanie, inteligentné posielanie súborov, ktorý by klient vypýtal neskôr e.g. css
- podporovaný každým významným prehliadačom

2.1.2. Metódy

Sú to druhy žiadostí, ktoré vyjadrujú požadovanú akciu, ktorú chceme vykonať s označeným zdrojom. Metódy, ktoré nespôsobujú žiadnu zmenu a slúžia len na získavanie informácii sa nazývajú bezpečné.

GET	Najčastejšie používané. Požiadavka na vypýtanie zdroja, súboru.
POST	Posiela dáta na server, spravidla údaje z vyplneného formulára
HEAD	Funguje ako GET s rozdielom, že odpoveď neobsahuje telo, len hlavičky
OPTIONS	Vyžiada podporované metódy pre daný zdroj
PUT	Žiadosť na ukladanie na server. Existujúci súbor sa prepíše
DELETE	Zmaže zdroj
TRACE	Klient dostane kópiu svojej požiadavky, vhodné na zistenie zmien spravené prechodnými servermi
CONNECT	Používané spolu s proxy serverom na SSL tunelovanie

2.1.3. Stavové kódy

Trojciferné číslo v desiatkovej sústave, ktoré je súčasťou hlavičky odpovede servera, oznamujúc výsledok spracovania požiadavky. Prvá cifra určuje typ odpovede a zvyšné dve to upresnia. Ohľadom na počet stavových kódov a na to že z pohľadu tejto práce sú nepodstatné, uvádzame len význam prvých cifier spolu s niekoľkými príkladmi.

1xx Odpoveď je informačného charakteru. e.g. server obdržal hlavičku požiadavky a čaká na telo

2xx Požiadavka bola úspešne spracovaná a telo odpovede obsahuje potrebné informácie

3xx Klient musí vykonať ďalšie akcie. e.g. uvedený zdroj nie je jednoznačný, bolo presunuté alebo klient má používať proxy

4xx Chyba na strane klienta. e.g. použitá metóda nie je podporovaná, klient nemá dostatočné prístupové práva, požiadavka je syntakticky nesprávna

5xx Chyba na strane servera. e.g. nepodporovaná verzia protokolu, server je preťažený alebo nemá dostatok miesta na uloženie poslaného súboru

2.1.4. Príklad konverzácie medzi prehliadačom a serverom

Spustili sme jednoduchý server podávajúci lokálne súbory z pracovného adresára príkazom `python -m http.server 8000`, následne sme sa prehliadačom pripojili na adresu `localhost:8000`. Správy sme odchytili pomocou programu wireshark.

Žiadosť klienta:

```
GET /test HTTP/1.1
Host: localhost:8000
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:46.0)
Gecko/20100101 Firefox/46.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

Ako vidíme prvý riadok našej žiadosti je názov metódy, cesta k súboru a verzia protokolu. Nasledujú hlavičky ako adresa hosťa, typ použitého klienta, očakávaný typ, jazyk a kódovanie obsahu, požiadavka na zachovanie spojenia.

Odpoveď servera

```
HTTP/1.0 200 OK
Server: SimpleHTTP/0.6 Python/3.4.3
Date: Mon, 02 May 2016 22:09:21 GMT
Content-type: text/html
Content-Length: 90
Last-Modified: Mon, 02 May 2016 20:20:42 GMT

<!DOCTYPE html>
<html>
<head>
  <title>Hello</title>
</head>
<body>
  Text
</body>
</html>
```

Odpoveď pozostáva z verzie protokolu a stavového kódu, v našom prípade je to 200 OK, keďže súbor, ktorý sme vypýtali existuje a môže byť doručený. Pozostáva z hlavičiek a z patričného obsahu, čo je krátky html súbor. Hlavičky obsahujú informáciu o type servera, čase odoslania odpovede, typ obsahu, dĺžku a čas jeho poslednej úpravy.

2.2. WSDL

Ako už vieme, webové služby obsluhujú ostatné aplikácie. Na to, aby sa daná aplikácia vedela pripojiť k službe a využívať poskytovanú funkcionalitu, server a klient sa musia dohodnúť na spoločnom formáte komunikácie, ktorému obidvaja rozumejú. Server teda potrebuje spôsob publikovania svojho rozhrania. Riešenie tejto úlohy má na starosti práve WSDL – Web Service Description Language, jazyk založený na XML.

Pri tvorbe webových služieb založených na WSDL existujú dva protichodné prístupy. Môžeme napísať WSDL ručne, a nechať si vygenerovať kostru programu, do ktorej doplníme funkcionalitu, alebo napíšeme program a generujeme WSDL. Keďže potrebujeme funkčnú webovú službu na neskoršie testovanie knižnice, a písanie aplikácií je výrazne ľahšie a rýchlejšie vďaka vývojovým prostrediam, ako ručná tvorba WSDL súboru, zvolíme si druhú alternatívu, a ťažšiu časť práce ponecháme prostrediu MonoDevelop.

2.2.1. Formát WSDL súboru

WSDL definuje webové služby pomocou šiestich XML elementov, z ktorých tri sú abstraktné a tri konkrétne. Toto rozdelenie umožňuje znovupoužitie abstraktnej časti, kedže neobsahuje žiadne konkrétne detaily. Pre lepšie pochopenie použijeme časti vygenerovaného WSDL pre webovú službu s jedinou funkciou, sčítanie dvoch čísel ako príklad.

Abstraktné časti:

- **Types** definícia dátových typov pomocou typového systému XSD (XML Schema Definition)
- **Message** reprezentácia správy pomocou typov
- **PortType** sada operácií pozostávajúci zo vstupných a výstupných správ

Konkrétne časti:

- **Binding** definuje prenosný protokol pre daný portType
- **Port** určí prístupový bod pre binding pridaním adresy
- **Service** zoznam portov tvoriacich službu

2.2.2. Časť Types

```
<types>
 <xs:schema elementFormDefault="qualified"
  targetNamespace="http://tempuri.org/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="sum">
   <xs:complexType>
    <xs:sequence>
     <xs:element minOccurs="1" maxOccurs="1" name="a" type="xs:int" />
     <xs:element minOccurs="1" maxOccurs="1" name="b" type="xs:int" />
    </xs:sequence>
   </xs:complexType>
  </xs:element>

  <xs:element name="sumResponse">
   <xs:complexType>
    <xs:sequence>
     <xs:element minOccurs="1" maxOccurs="1" name="sumResult"
      type="xs:int" />
    </xs:sequence>
   </xs:complexType>
  </xs:element>

 </xs:schema>
</types>
```

V príklade dobre vidíme, že formát je ľudsky čitateľný, ale veľmi rozvláčny, čo je hlavným dôvodom prečo ho ľudia považujú za komplikovaný. V skutočnosti je to ale pomerne jednoduché.

Máme tu element *sum*, ktorý je komplexného typu, to znamená, že sa skladá z iných elementov. Keďže v našom prípade reprezentuje zbierku vstupných parametrov pre funkciu súčet dáva to zmysel. Obsahuje sekvenciu elementov menami *a* a *b,* ktoré sú celé čísla. Sekvencia, ako element a complex type, je ďalším tagom z XSD. Naznačuje, že všetky jeho potomkovia musia byť v danom poradí. V tomto prípade teda jedna až jedna (čiže presne jeden) element s menom *a* nasledovaný presne jedným elementom *b.*

SumResponse podobne definuje element pre výstup funkcie. V princípe by to mohlo byť zapísané aj ako `<xs:element name="sumResponse" type="xs:int">` ale to sa používa len v prípade jedného štýlu viazania, viď nižšie.

Môžeme tu naraziť aj na tagy restriction – obmedzenia, enumeration – zoznamy možných hodnôt, union – n-tice a choice – výber medzi rôznymi elementmi.

2.2.3 Časť Message a štýl viazania

Message je abstraktná časť WSDL dokumentu reprezentujúci jednu správu v komunikácii. Skladá sa z jednej alebo viacerých logických častí, ktoré sú buď elementy alebo typy, podľa toho aký štýl WSDL viazania (binding) používame.

Poznáme štyri až šesť štýlov podľa toho ktoré y nich považujeme za plnohodnotné a ktoré nie. Viazanie je typu document – server posieľa dokument, alebo RPC (Remote Procedure Call) – operácia je vzdialené volanie zdielanej funkcie. Ďalej viazanie môže – encoded, ale nemusí – literal, byť zakódované. Pre štýl dokument je ešte posledná voľba, či chceme veci zabaliť – wrapped.

Organizácia Web Services Interoperability je združenie založené na propagáciu kompatibility medzi viacerými špecifikáciami webových služieb, a napriek tomu, že nedefinuje štandardy, len smernice, väčšina poskytovateľov ich dodržiava. Zakódovanie viazaní nie je kompatibilné s WS-I, preto tie možnosti nás nebudú zaujímať. Ostávajú nám kombinácie:

1. Document/literal

```
<types>
 <xs:schema elementFormDefault="qualified"
  targetNamespace="http://tempuri.org/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="a" type="xs:int" />
  <xs:element name="b" type="xs:int" />
  <xs:element name="sumResult" type="xs:int" />
  <xs:element name="int" type="xs:int" />
 </xs:schema>
</types>
<message name="sumSoapIn">
 <part name="a" element="s0:a" />
 <part name="b" element="s0:b" />
</message>
```

2. RPC/literal

```
<message name="sumSoapIn">
 <part name="a" element="s0:a" />
 <part name="b" element="s0:b" />
</message>
```

3. Document/literal wrapped

```
<message name="sumSoapIn">
 <part name="parameters" element="s0:sum" />
</message>
```

V nasledujúcich odsekoch sa budeme zmieňovať o protokole SOAP, ktorý sme ešte nerozoberali, avšak prosíme čitateľa aby akceptoval tieto ustanovenia za pravdivé aj bez príkladov, keďže WSDL a SOAP sú natoľko blízke pojmy, že je ich ťažké rozdeliť. Zároveň tak rozsiahle, že nechceme medzi nimi skákať, a venujeme im vlastné časti kapitoly.

Každý zo spomínaných prístupov má svoje silné a slabé stránky. Prvý, Document/literal generuje SOAP správy, ktoré sú validné XML dokumenty, ale má dve veľmi vážne nedostatky. Nie je to vždy v súlade so smernicami WS-I, ktoré vyžadujú, aby telo správy obsahovalo jeden element. V príklade by obsahovalo dva, čísla *a* a *b*. Väčší problém je, že pýtame len dokument, a prislúchajúca SOAP správa nebude obsahovať názov „volanej“ funkcie. Výber správnej funkcie na základe parametrov je náročné a v istých prípadoch nemožné.

Jedinou nevýhodou druhého riešenia je fakt, že správa bude obsahovať prvky, ktoré nie sú súčasťou definície schémy, preto nebude správa validná.

Na šťastie je tu tretia možnosť, kombinujúca výhody prvých dvoch. Máme element s názvom funkcie, do ktorého zabalíme parametre, tým sa vyhneme problémom množstva elementov v tele správy a prenášame aj potrebnú informáciu na zavolanie správnej funkcie. XML ale nedovoľuje viaceré elementy s rovnakým menom, s čím strácame preťaženie funkcií. Tak isto sa WSDL súbor stáva menej čitateľným. Tieto nevýhody sú ale zanedbateľné v porovnaní s výhodami, ktoré robia z Document/literal wrapped fakticky primárny výber.

2.2.4. Časť PortType a typy komunikácie

PortType, posledná z trojice abstraktných častí dokumentu sú operácie pozostávajúce z predchádzajúcich správ. WSDL podporuje štyri typy operácií podľa toho kto pošle komu správu a v akom poradí:

- **One – way** Jednosmerná komunikácia, klient pošle správu na server
- **Request – response** Obojsmerná, klient pošle správu, následne obdrží odpoveď
- **Solicit – response** Obojsmerná, s tým rozdielom, že prvú správu pošle server
- **Notification** Jednosmerná, server oznámi niečo klientovi

Bez ohľadu na to koľkáta je v poradí, správu od klienta označuje XSD tag input, a správu od server tag output. Pri obojsmerných operáciach je možné definovať aj nepovinný fault – chybu, ktorá informuje o nesprávnych argumentoch alebo iných problémoch ktoré znemožnili správne vykonanie príkazu. Príklad z našej jednoduchej WSDL:

```
<portType name="ServiceSoap">
 <operation name="sum">
  <input message="s0:sumSoapIn" />
  <output message="s0:sumSoapOut" />
 </operation>
</portType>
```

2.2.5. Časť Binding

```
<binding name="ServiceSoap" type="s0:ServiceSoap">
 <soap:binding transport="http://schemas.xmlsoap.org/soap/http" />
 <operation name="sum">
  <soap:operation soapAction="http://tempuri.org/sum" style="document"/>
  <input>
   <soap:body use="literal" />
  </input>
  <output>
   <soap:body use="literal" />
  </output>
 </operation>
</binding>
```

Prvá z konkrétnych častí, binding tj. viazanie spája PortType, v uvedenom príklade portType ServiceSoap viažeme na transportný protokol HTTP. (sice sme neukazovali, lebo našim cieľom je dopracovať k protokolu SOAP, ale vo vygenerovanej WSDL máme

správy, portTypy a viazanie aj pre komunikáciu cez HttpGet, HttpPost a aj pre soap verziu 1.2). Ako vidíme v tomto bode máme možnosť vybrať si medzi štýlmi viazania.

2.2.6. Časti Services a Ports

Zvyšné dve konkrétne časti preberieme spoločne, keďže jeden je potomok druhého. Porty určia na akú adresu musíme poslať svoje požiadavky, a umožnia tým prístup k poskytovaným službám. Podľa špecifikácie musia obsahovať práve jednu adresu, a žiadne iné informácie o viazaní.

Services udáva názov celej služby a zgrupuje všetky poskytnuté prístupové body.

```
<service name="Service">
 <port name="ServiceSoap" binding="s0:ServiceSoap">
  <soap:address location="http://127.0.0.1:8080/Service.asmx" />
 </port>
 <port name="ServiceSoap12" binding="s0:ServiceSoap12">
  <soap12:address location="http://127.0.0.1:8080/Service.asmx" />
 </port>
</service>
```

2.2.7. Verzia 1.1 a verzia 2.0

V predošlej časti sme opisovali verziu 1.1 napriek tomu, že už od roku 2007, čo je skoro celá dekáda máme verziu 2.0. Dôvodom je, že aj keď sa zopár vecí zmenilo, z nášho hľadiska nie sú príliš dôležité. Služba s ktorou pracujeme, tak isto ako mnoho iných služieb používa starú verziu. Okrem toho veľa zo zmien sú len syntaktické. Časť Message zanikla, stala sa súčasťou operácií, ktoré nepodporujú preťaženie (s čím stráca RPC výhodu). PortTypes bolo premenované na interfaces, rozhrania, a ports na endpoints, koncové body, ktoré lepšie vystihujú ich funkciu. Zmenil sa aj názov celého jazyka z Web Service Definition Language na Web Service Description Language.

2.3. SOAP

V predošlej časti sme zistili ako máme efektívne, a pre iné aplikácie zrozumiteľne popísať naše webové služby. Predtým ako ich začneme využívať potrebujeme už len spôsob ako dostať potrebné informácie z jedného miesta na druhé. To nám vyrieši protokol SOAP.

Spočiatku skratka pre Simple Object Access Protocol, dnes už len štvorpísmenkový názov. Bol navrhnutý pánmi Dave Winer, Don Box, Bob Atkinson, and Mohsen Al-Ghosein ako protokol na prístup k zdielaným objektom v roku 1998. Časom sa stal de facto štandardom pre komunikáciu pomocou XML správ v decentralizovanom distribouvanom prostredí. Súčasťou špecifikácie sú model spracovania správ, formát komunikačného rámca, popis viazaní s prenosnými protokolmi a model rozšíriteľnosti.

Pri spracovaní správ rozdeľujeme účastníkov podľa ich úlohy na odosielateľa, príjemcu a sprostredkovateľa, ktorá je odosielateľom aj príjemcom zároveň.

SOAP rieši len základnú problematiku toho, že chceme dostať správu z jedného bodu do druhej. To ako budú tieto správy smerované či zabezpečené už prenechá rozšíreniam.

Čo sa týka viazania, nie je závislý na žiadnom prenosnom protokole. Môže používať SMTP, TCP, UDP, JMS aj iné. Napriek širokému výberu implementácie preferujú HTTP. Dôvodom tohto silného sklonu k jednej z mnoho alternatív je fakt, že HTTP je skoro všade podporovaný a zvyčajne neblokovaný firewallmi.

Na to, aby sme vedeli debugovať problémy nastávajúce počas komunikácie, už nám stačí len pozrieť ako by mala správa vypadať.

2.3.1. Komunikačný rámec

Komunikačný rámec je najdôležitejšou časťou špecifikácie. Je to elektronická obdoba poštovej obálky, preto aj názov hlavného elementu, envelope. Obálka môže obsahovať voliteľné hlavičky v elemente header. V elemente body nájdeme telo správy, alebo v prípade chyby element fault. Fault musí obsahovať položky faultcode a faultstring, ktoré určujú typ chyby vo formáte, ktorý je čitateľný pre počítač respektíve pre človeka. Hlavičky nie sú štandardizované ako v prípade HTTP a iných protokolov, ale sú všeobecné obaly ako telo. Majú však štandardizovaný atribút mustUnderstand, čo naznačuje, že príjemca správy má podporovať danú hlavičku, keď chce spracovať správu. Uvádzame príklad komunikácie medzi našou testovacou službou a klientom knižnice PySimpleSoap použitím štýlu Document/literal wrapped:

Požiadavka klienta:

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soap:Header/>
 <soap:Body>
  <Sum xmlns="http://tempuri.org/">
   <a>1</a>
   <b>2</b>
  </Sum>
 </soap:Body>
</soap:Envelope>
```

Odpoveď servera:

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
 <soap:Body>
  <SumResponse xmlns="http://tempuri.org/">
   <SumResult>3</SumResult>
  </SumResponse>
 </soap:Body>
</soap:Envelope>
```

V našom jednoduchom príklade nevidíme žiadne hlavičky, vidíme však štruktúru správ. Ako sme to už popísali, požiadavka a odpoveď majú rovnaký formát. V tele požiadavky je element Sum reprezentujúci vstupné parametre pre funkciu s totožným názvom. Odpoveď je dôkladne zabalený súčet.

2.3.2. SOAP vs REST

Pri študovaní webových služieb sme často narazili na pojem Representational State Transfer, skrátene REST, a na otázku či nie je to lepšie riešenie ako SOAP. Prvá vec čo musíme pochopiť je, že sa nedajú úplne porovnávať, SOAP je totiž protokol a REST architektúra. Napriek tomu to každý porovnáva, urobíme tak aj my.

To ako vyzerajú a cestujú SOAP správy už vieme, sú to XML dokumenty a na posun používajú široké spektrum protokolov. REST nemá taký veľký výber čo sa týka cestovania. Je založený na HTTP a používa jeho metódy GET, PUT, POST a DELETE. Má však voľnejší výber formátov na prenos informácie. V podstate by mal zvládnuť všetky formáty, ktoré sa dajú preniesť protokolom HTTP, ale väčšinou používa JSON. Kde je XML jedným extrémom, obšírne správy obsahujúce veľké množstvo dodatočných informácií, JSON predstavuje druhý. Neobsahuje žiadne dodatočné informácie. K službám REST nepatrí ani dlhý dokument WSDL, za ktorými by väčšina z nás nesmútila. Chýbajú mu ale aj rozšírenia. Ktorá voľba je teda lepšia? Záleží na tom čo potrebujeme.

Keď chceme čím jednoduchší formát, rýchlejšie parsovanie, menej „zbytočných“ informácií, kešovateľné volania pravdepodobne si vyberieme REST. Ako to ukazujú štatistiky, je populárnejšie ako SOAP, a je to často lepšia voľba.

Keď nám naopak nevadí robustnosť, ale potrebujeme prísne definované formáty, spoľahlivú komunikáciu (zabudovaná logika pre zlyhanie – faulty), bezpečnosť či atomické transakcie (tie nám ľahko zabezpečia rozšírenia), tak SOAP je správna voľba.

V našom prípade sme možnosť výberu nedostali. Máloktoré z webových služieb poskytujú rozhranie SOAP aj REST zároveň, a keď sa trochu zamyslíme, zistíme, že ani náš poskytovateľ nemohol volne vyberať. Predáva totiž lístky na hromadnú dopravu, a pri práci s peniazmi sú ACID transakcie a spoľahlivosť predsa dôležitejšie ako šetrenie s prenesenými bytmi.

Kapitola 3.

PySimpleSoap

3.1. Ciele knižnice

PySimpleSoap je voľne šíriteľná knižnica jazyka Python na prácu s SOAP webovými službami. Vznikla ako spoločná snaha niekoľkých ľudí, pôvodne ako klient pre Argentina's IRS. Spočiatku malá knižnica, s necelých 200 riadkami zdrojového kódu, dnes má už zhruba 4000.

Vysnívané ciele autorov boli:

- Jednoduchosť
- Flexibilita, podpora viacerých SOAP dialektov
- Vyhnúť sa dynamickým triedam
- Podpora viacerých HTTP knižníc z dôvodu bezpečnosti
- Podpora jazyka Python2 a Python3 súčasne
- Rýchlosť

A čo sa im z toho podarilo? Program takýchto rozmerov, s mnoho spoluautormi a nepríliš prísnymi pravidlami prispievania, sa stáva veľmi rýchlo neprehľadným „špagetovým“ kódom. Hlavný autor knižnice nemá dostatok času na spravovanie, preto je možné, že sme dostali plné práva prispievania na GitHube, a môžeme samovoľne modifikovať zdrojový bez toho aby niekto na to dozeral, či skontroloval naše rozhodnutia. Napriek tomuto je knižnica stále pomerne prehľadná, a spĺňa, teda aspoň čiastočne, všetky svoje ciele.

3.2. Implementácia klienta a opravovanie chýb

PySimpleSoap má aj serverovú implementáciu, ale my sa zamierame hlavne na klienta. Zaujímajú nás teda súbory: simplexml, transport, helpers a client.

3.2.1. SimpleXml

Obsahuje jedinú triedu SimpleXMLElement, ktorej úloha, ako ľahko uhádneme, je spracovanie XML správ. Tieto časti programu pracujú s našimi žiadosťami ako posledné, a s odpoveďami servera ako prvé. Riešia prevod medzi pythonickými štruktúrami ako list alebo slovník, a XML stromom. Jedna z problémov čo sme si všimli bolo nesprávne parsovanie zoznamov. Kým sme ako odpoveď od servera dostali jedno číslo, alebo komplexný objekt obsahujúci viacero položiek, napríklad človeka s menom a priezviskom, bolo všetko poriadku. Skúsili sme poslať zoznam čísel, a dostali sme jediné číslo.

Prvá naša iniciatíva bola samozrejme preskúmať poslanú žiadosť a získanú odpoveď. Na prvý pohľad bolo všetko v poriadku. V žiadosti bol názov správnej funkcie, a odpoveď obsahovala naozaj zoznam čísel. Náš posudok znel, XML správa je taká, aká má byť, výsledná pythonová štruktúra je však iná, problém musí teda nastať v kroku parsovania. Omyl. Jednu malú skutočnosť sme neuvedomili. WSDL a SOAP napriek tomu, že sú čitateľné, neboli vymyslené na to, aby sme ich čítali, čo sme rýchlo zistili sami.

V tomto štádiu práce server na testovanie typických prípadov prenosu informácie sme prevádzkovali pomocou Javy, presnejšie s prostredím Eclipse. Existuje veľa možností na tvorbu jednoduchých webových služieb, a dve jazyky s najlepšou podporou sú práve Java a C#. Keďže C# patrí medzi jazyky .NET, ktorý primárne beží na Windowse a my pracujeme s Linuxom vybrali sme Javu.

S čím sme nepočítali bol fakt, že väčšina návodov na tvorbu webovej služby SOAP má už svoje roky vďaka popularite REST, a v dokumentácii je ľahké niečo prehliadnuť. Napokon sme zistili, že sme nepovedali všetky potrebné informácie na vytvorenie správneho WSDL.

Berúc do úvahy, že sme už mali aj iné ťažkosti podobného charakteru pri rozbehaní

našich testov, zavolala sa správna funkcia, ale všetky parametre boli nedefinované následkom nesprávne nastavených prefixov a menových priestorov v žiadosti, sme sa rozhodli vyskúšať inú možnosť, C#. Mono je voľne dostupná implementácia .NET prostredia, ktorá na rozdiel od originálu je multi platformová. Použili sme teda Mono, a vývojové prostredie MonoDevelop. Konečne sme mali v ruke nástroj ktorý keď spustíme, napíšeme si v funkciu akú chceme, podľa potreby prepneme štýl viazania, vydáme príkaz na spustenie servera a zrazu všetko funguje.

Nenašli sme teda nič, čo by bolo treba vyčítať tejto triede, a nespravili sme tu žiadne zmeny.

3.2.2. Transport

Súbor v ktorom sú triedy na používanie rozličných transportných knižníc. Na výber máme hneď tri možnosti, knižnice urllib2, httplib2 a pycurl. Jediná podmienka, aby sme ich mohli použiť je, aby sme ich mali nainštalované. Na prácu so serverom kvôli ktorej sme v prvom rade potrebovali SOAP knižnicu, potrebujeme zabezpečenie, keďže ide o službu, pri ktorej sa manipuluje aj s peniazmi, a cookies pre účeli prihlasovania uživateľov, z poskytnutých knižníc nám vyhovoval urllib2.

Z času na čas pri práci na projekte, na ktorom pracujú aj iný vývojári sa oplatí aktualizovať našu kópiu. Stiahli sme teda najnovšieho mastra z GitHubu veriac tomu, že ako v každom slušnom projekte, sa dostanú do hlavnej vetvy len pretestované zmeny, ktoré funkcionalitu pridávajú alebo opravujú, ale v žiadnom prípade nepokazia. Verili sme tomu, ale nemali sme, a samozrejme na túto aktualizáciu veľkoryso zabudli. Pracovali sme na na inej chybe, pri ktorej nám stačila vlastná malá webová služba bez zabezpečenia a cookies.

Po oprave spomenutej chyby sme sa snažili pripojiť na našu hlavnú cieľovú webovú službu, lenže nešlo to. Vítala nás divná chybová hláška o tom, že jedna z funkcií v súbore Transport dostala neočakávaný argument, „context“. Bolo to prekvapenie, lebo doteraz všetko fungovalo, a ani v tom súbore, ani nikde inde sme nezmenili nič čo i len trochu súviselo s prenosom.

Pri snahe odhaliť dôvod tohto náhleho zlyhania, sme pozreli aj záznamy o aktualizovaní operačného systému, lebo zhodou okolností tesne pred zistením chyby sme

nainštalovali systémom navrhnuté vylepšenia. Nebýva zvykom, ale z času na čas zdanlivo malá a nevinná zmena dokáže spôsobiť veľké problémy, dokonca nefunkčnosť niektorých častí systému. Presvedčení, že príčina chyby môže byť zmena v niektorej zo základných systémových knižníc ktoré Python v pozadí používa, sme prezreli záznamy. A schválne, bolo tam zopár podozrivých položiek. Zmeny sme vrátili a spustili klienta. Výsledok rovnaká chyba.

Konečne sme si spomenuli na osudnú aktualizáciu projektu. Neveriac sme pozerali na posledné zmeny v hlavnej vetve. Robili sa na nej malé zlepšenia práve v časti slúžiacej na prenos správ, ktoré vyzerali neškodne. Pridávali sa bezpečnostné kontexty.

Čo si autor zmeny neuvedomil je, že parameter kontext bol pridaný do knižnice urllib2 pri verzii Pythona 2.7.9 respektíve 3.2.0. Pridali sme teda podmienku na zahrnutie kontextu len pre dostatočne nové verzie. Teraz sme už boli viac skeptickí. Chceli sme si overiť, či daná zmena funguje dobre aj v novších verziách. Neskúsenému požívateľovi Linuxu sa môže zdať aktualizácia systémového Pythona ako dobrý nápad. Podarilo sa nám dostať operačný systém do stavu, v ktorom isté časti odmietli fungovať. Pri snahe napraviť túto banálnu chybu sa nám podarilo dostať do stavu, v ktorom záloha dát a preinštalovanie systému bolo ľahšie, rýchlejšie a spoľahlivejšie ako ďalšie opravy.

Poučení z našich predošlých chýb sme rozbehali Python vo virtuálnych prostrediach, a pre istotu sme začali testovať knižnicu s viacerými verziami. Tu sme narazili na druhú chybu, ktorú spravil autor pri pridávaní kontextu. Na prístup k objektom pomocou URL používa urllib2 objekty triedy OpenerDirector. Tieto objekty je možné vybudovať zo zoznamu manipulačných objektov, ktoré riešia úlohy ako používanie cookie alebo certifikátu. Taktiež poskytuje funkciu urlopen, ktorá berie parameter adresu URL, a niekoľko voliteľných, mimo iného kontext. Počas prvého volania vybuduje z týchto parametrov globálny OpenerDirector, ktorý následne používa. Autor zmeny najprv zobral funkciu urlopen, potom to podľa potreby vymenil na funkciu open nového OpenerDirectora, ktorý vybudoval pre spracovanie cookie. Chyba spočíva v tom, že v poslednom kroku zavolá funkciu urlopen alebo naopak open rovanko, s parametrom kontext. Lenže urlopen taký parameter má, lebo je to praktické volanie vybudovania nového objektu OpenerDirector, ale open už nemá, lebo je to metóda spomínaného objektu, ktorá potrebnú informáciu nesie v sebe. Našim riešením bolo v každom prípade budovať vlastný objekt rovno so správnymi schopnosťami.

3.2.3. Helpers

Ako názov napovedá ide o súbor, v ktorom sú pomocné funkcie a triedy. Zaujímavé pre nás budú hlavne triedy Alias a Struct, a funkcia postprocess_element.

Prvý nedostatok, na ktorý sme narazili bola neschopnosť knižnice pracovať s desatinnými číslami. Celé čísla zvládala bezchybne, no desatinné skončili chybovou hláškou. Koreňom problému je, že Python je dynamicky typovaný jazyk, pričom webové služby majú silnú predstavu o tom aké typy argumentov ich funkcie očakávajú. Okrem toho Python pozná štyri typy čísel:

- **int** a **long** Reprezentácia celých čísel medzi ktorými Python automaticky prepína podľa potreby. Jedna z príjemných javov jazyka je práve schopnosť pracovať s ľubovolne veľkými číslami. C++ či Java pre každého známe jazyky dokážu vypočítať a ukladať číslo 5!, nevedia si bez pomoci poradiť s 50!. Python to zvláda. Pri celých číslach treba pri najhoršom riešiť nedostatok pamäti, ale pretečenie nikdy.
- **float** Reprezentuje desatinné čísla, nie je rozdelené na float a double ako sme si na to mohli zvyknúť, má však dĺžku ako double máva
- **complex** Predstavuje komplexné čísla, nestretneme sa s nimi príliš často

XSD definuje omnoho viac typov celých čísel, a keďže sú to súčasti štandardu môžeme očakávať ich výskyt v WSDL súboroch:

- **byte**
- **decimal**
- **int**
- **integer**
- **long**
- **short**

Na to aby sme s nimi vedeli pracovať a správne ich parsovať z textovej podoby, musíme každý typ zmapovať na Pythonovské typy. Analogicky to platí aj pre desatinné čísla, pre rôzne reprezentovanie textov a podobne.

Ostávajú teda otázky, prečo fungujú operácie s celými číslami a s textom, ale nie s desatinnými, a prečo si to doteraz nikto nevšimol. Máme tu zase nešťastnú zhodu okolností. Najčastejšie typy, ktoré používame sú int, float, a string. Tieto typy máme aj v Pythone, zafungujú teda aj priamočiaro. Problém nastane až v chvíli, keď chceme napríklad zavolať operáciu s vstupom alebo výstupom typu double. Vtedy už Python si sám neporadí. Program sa zrúti pri kontrole správnosti parametrov. Už vieme prečo sa to prejavuje len v istých prípadoch, ale stále nám nie je jasné ako taká základná chyba mohla takto dlho unikať.

Dôvodov je hneď niekoľko. Keď sa programátor snaží riešiť nejakú úlohu, veľakrát narazí na časť problému, ktorý už pred ním niekto iný riešil a tým pádom nemá zmysel sa snažiť spraviť to sám. Siahne vtedy za nástrojom, knižnicou či modulom aby si pomohol. V lepšom prípade to znamená úspešné zvládnutie časti a pokračovanie ďalej v práci. Stáva sa ale, že daný nástroj nespĺňa očakávania. Vývojár má na výber z dvoch možností. Snaží si zmeniť, upraviť nástroj pre svoje účely, čo ako sme to sami zažili, po chvíľke trápenia omrzí, alebo si skúsi nájsť inú alternatívu. A to je presne smer, ktorý si vyberie väčšina, a treba dodať, že právom. Veď vôbec nie je rentabilné, aby sa každý nástroj menil, v tom prípade by už ich mohol sám naprogramovať, presne podľa svojej potreby, namiesto toho, aby strácal čas snahou pochopiť nie práve najprehľadnejšiu tvorbu niekoho iného.

Druhým dôvodom môže byť, že nikdy nenarazil na funkciu, ktorá pracuje s typom double, alebo s inými problematickými typmi, ktoré sa vyskytujú ešte zriedkavejšie. Nie je bezprikladný jav ani posielanie každého parametra ako text. Zruinuje to trochu zmysel protokolu, ale nie je žiadnym spôsobom zakázaný. Pokojne môžeme zadefinovať funkciu na spočítanie čísel aj s parametrami a výstupom typu string. Nedáva to veľký zmysel, ale nič nám v tom nebráni. Server dostane čísla ako text, spracuje ich, vypočíta sumu, výsledok prevedie do textového formátu a pošle klientovi. Je po tom na klientovi, aby čísla parsoval pre svoje účely. Pri hľadaní služieb na testovanie knižnice, pred tým ako sme službu začali sprevádzkovať sami, sme podobný príklad naozaj videli.

Poslednú otázku, čo spôsobuje nesprávnu funkcionalitu, napriek tomu že máme objekty typu Alias, ktoré boli určené na riešenie tejto problematiky, zodpovieme neskôr, úzko súvisí totiž s druhým závažným nedostatkom.

Hlavnou príčinou našej snahy vylepšiť knižnicu je jej rýchlosť, alebo práve naopak pomalosť. Domnievali sme sa, že riešenie niektorých vecí musí byť asymptoticky neoptimálne. Tento pocit v nás posilnili aj vedomosti získané pri študovaní WSDL a SOAP. Podľa toho, čo sme sa dozvedeli, spracovanie popisu služby, aj tvorba či spracovanie správy by malo byť možné v čase O(n). Nie je predsa prijateľné, aby parsovanie služieb trvalo desiatky sekúnd ani v prípade dokumentu s rádovo tisíckami riadkov. Medzi časom písania zadania práce a časom kedy sme sa dostali k študovaniu a oprave programu iní kolaboranti uskutočnili zmeny. Niektoré z nich mali za následok zrýchlenie tvorby a spracovanie správ, lebo naše merania ukazovali dostatočne malé časy pri týchto činnostiach. Zaoberali sme sa teda s problémom pomalého načítavania definície webovej služby.

Zaujímavý jav, ktorý sme pozorovali bola chybová hláška o dosiahnutí maximálnej hĺbky rekurzie pri spracovaní WSDL. Vďaka kešovaniu druhý pokus vždy uspel. Pri debugovaní sme zistili, že objekty triedy Struct, ktoré sú v podstate upravené slovníky a majú reprezentovať jednotlivé komplexné elementy použité v definícii služby, sa pri narazení na prvok toho istého typu vyrobia znovu. Nemusí to nutne znamenať značné spomalenie alebo chyby, a len preto, aby sme znížili konštantu v časovej zložitosti nemá ani zmysel zmeniť toto správanie. Takáto úprava by znamenala väčšie zmeny, ktoré sme chceli spraviť len ak je to nutné.

Hláška o maximálnej hĺbke rekurzie, by mala slúžiť ako dostatočne dobrá smernica na odhalenie miesta chyby. Chceli sme nájsť v najlepšom prípade presný element, pri ktorom chyba nastane. Zmiatli nás však rovnaké chyby pri debugových výpisoch. Môžu, a v prípade komplexných webových služieb aj budú existovať elementy, ktoré vytvárajú cyklickú definíciu. Textová reprezentácia triedy Struct bola napísaná rekurzívne. Obsahovala svoje meno, a všetky potomky. Dokonca sme videli niečo, čo dodnes nechápeme prečo to niekto považoval za prijateľné riešenie. V momente, keď sme dosiahli maximálnu hĺbku rekurzie a dostali sme chybovú hlášku počas tvorby textovej reprezentácie elementu, zobrali sme hlášku a pridali k práve vyrábanému textu. Výsledok bol ohromujúci. Nekonečne dlhý string plný chybových hlásení v konzole. Nájsť v tom chaose relevantnú informáciu bolo nemožné.

Na naše šťastie neobsahuje knižnica príliš veľa rekurzívnych funkcií, a fakt, že sme danú chybu dostali aj bez jediného konzolového výpisu, nás doviedol na správne miesto. Funkcia postprocess_element. Všetko vyzerá logicky, dva parametre, knižnica elementov čakajúcich na spracovanie, a zoznam už spracovaných elementov, aby sme nespravili tú istú prácu viackrát. Funkcia sa dá popísať ľahko. Zober si všetky elementy, ktoré musíš spracovať, a po jednom ich prejdi. Keď je daný element typu Struct, rekurzívne spracuj všetky jeho súčasti, analogicky vykonaj pre zoznam. V prípade, že narazíš na element, ktorý si už videl, ukonči vetvu rekurzie. A tu sa nám vypomstí, že v predchádzajúcich krokoch vytvárame nové Structy.

Trochu sa tu zastavíme a pozrieme si, ako sa porovnávajú primitívne typy a objekty. Python na porovnávanie výrazov používa `==` pre kontrolu hodnotnej rovnosti a kľúčové slovo `is` ako porovnanie hodnotnej a zároveň typovej rovnosti. Platia teda súvislosti:

```
int(5) == int(5)  je True
int(5) == long(5) je True
int(5) is int(5)  je True
int(5) is long(5) je False
```

Pri porovnávaní slovníkov a zoznamov je to trochu komplikovanejšie, ale stále intuitívne. Sú to objekty, ktoré majú svoje pamäťové adresy. V obidvoch prípadoch platí že `==` ich porovnáva po zložkách a `is` podľa adresy. Príklady:

```
a = [1,2,3]              a = {'x':1, 'y':2, 'z':3}
b = [1,2,3]              b = {'x':1, 'y':2, 'z':3}
c = [3,2,1]              c = {'x':3, 'y':2, 'z':1}
a == b je True           a == b je True
a is b je False          a is b je False
a == c je False          a == c je False
```

Jediné čo ostáva sú triedy definované používateľmi. V tomto prípade základné správanie `==` je rovnaké ako `is`. Porovnávame teda pamäťové miesta:

```
class T:
  def __init__(self, value):
    self.value = value
  def __repr__(self):
    return str(self.value)

T(1) == T(1) je False
T(1) is T(1) je False
```

Môžeme to ale ovplyvniť. V python2 stačí zadefinovať metódu `__cmp__` s dvoma argumentami. Prvý, ako sme na to zvyknutí v Pythone, je self, a druhý je other. Dostaneme teda seba a niekoho iného aby sme sa porovnali ako len chceme. Výstupom má byť číslo. Záporné znamená že sme menší ako ten druhý, nula signalizuje rovnosť a kladné číslo je znakom toho, že sme väčší. Jediná metóda vyrieši všetky porovnávacie operácie.

V Python3 `__cmp__` sa nepoužíva. Namiesto neho bolo zavedené šesť nových metód. `__eq__, __ne__, __lt__, __gt__, __le__, __ge__`. Sú to operácie s boolovským výstupom, v poradí: rovnosť, nerovnosť, ostro menší, ostro väčší, menší alebo rovný, väčší alebo rovný. Treba dodať že tieto metódy fungujú aj v Python2 a v prípade, že sú zadefinované majú prednosť oproti univerzálnej `__cmp__`.

Po pochopení skutočnosti, ako boli porovnávania a príslušnosť v zozname u objektov typu Alias a Struct myslené, sme došli k názoru, že problémy v obidvoch prípadoch sa dajú vyriešiť dvoma prístupmi. Buď refaktorujeme program, a snažíme zaručiť, aby každý Struct a Alias sa vyrábalo presne raz, alebo dodefinujeme vhodne zvolené metódy na porovnávanie. Prvá možnosť by pravdepodobne mala za následky kopu nových chýb. Druhá možnosť s veľkou pravdepodobnosťou nič nepokazí, elegantne vyrieši naše najzávažnejšie problémy a je dostatočne priamočiara. Samozrejme sme sa rozhodli za druhé riešenie. Potrebovali sme ešte definovať hashovacie funkcie, aby spomínané objekty mohli byť použité ako kľúče v slovníkoch, a mali sme po problémoch.

3.2.4. Client

V súbore Client nájdeme triedu pre vytváranie klienta, ktorá spája všetky predošlé časti. Konštruktor triedy berie početné množstvo parametrov. Môžeme zadefinovať mimo iného menový priestor služby, verziu SOAP protokolu, bezpečnostný certifikát, súbor s ssh kľúčmi, meno a heslo pre prihlasovanie do služby, či chceme udržiavať spojenie a hlavne vieme povedať cestu k WSDL dokumentu, ktorý opisuje webový službu. Väčšinu týchto parametrov nastavovať nebudeme. V prípade jednoduchých služieb všetky potrebné informácie dostaneme z WSDL. Pre úspešnú komunikáciu s našou firemnou službou nám stačilo určiť URL popisu a nastaviť sessions na hodnotu True.

Spracovanie dokumentu je zdĺhavý ale priamočiary proces. Najprv skúsime načítať kešovanú už spracovanú verziu, v prípade zlyhania alebo keď máme zakázané kešovanie prečítame si WSDL z poskytnutej adresy a v dvoch krokoch z neho spravíme zoznam služieb. Väčšinu práce prvého kroku, prevedenie súboru na XML strom, vykonáva trieda SimpleXMLElement. Druhý krok je tvorba zoznamu služieb. Informáciu spracúvame v poradí ako vyskytujú v dokumente, dávame pozor na to, či neodkazujeme na elementy, správy, operácie či porty ktoré neexistujú.

Zaujímavosťou je, že kým väčšina SOAP knižníc vytvára dynamické triedy pre každý element, PySimpleSOAP má všetko uložené ako jednoduché Pythonovské konštrukcie. Nevyrába ani dynamicky nové metódy, ale preťaží metódu __getattr__ aby zavolal jednu spoločnú metódu s názvom požadovanej funkcie a s poskytnutými parametrami. Štruktúru týchto parametrov potom rekurzívne porovná s popisom funkcie. Všimneme si, že kým popis funkcie môže byť technicky nekonečný vďaka cyklom, skutočné parametre, ktoré podľa nich validujeme by mali byť vždy konečné. SOAP, ako ani iné protokoly na komunikáciu medzi rôznymi aplikáciami, je principiálne nevhodný na posielanie referencií. To znamená, že keď chceme poslať v správe spájaný zoznam klasického vyhotovenia tj. každý element má referenciu na nasledujúci, chybu nemáme hľadať v knižnici, ale v našom návrhu.

Napriek tomu, že to má byť hlavnou triedou celého klienta, a hľadali sme tu príčinu viacerých chýb, zdá sa, že je tu všetko v poriadku. Spravili sme tu len kozmetické úpravy ako oprava preklepu, a pridali metódu na reprezentovanie objektu, inšpirovanú reprezentáciou klienta z knižnice SUDS.

Záver

V našej práci sme snažili vylepšiť jednu z mnoha SOAP knižníc jazyka Python. Opravili sme niekoľko chýb, zlepšili výkon knižnice a spravili aj jemnú kozmetickú úpravu. Prácu považujeme za úspešne zvládnutú, a odchádzame od projektu s novými vedomosťami z viacerých hľadísk.

Oboznámili sme sa s webovými službami, ich výhodami, nevýhodami a možnosťami ich využitia. Naštudovali sme si s nimi spojené protokoly a formáty ako XML, WSDL a SOAP a čiastočne aj architektúru REST. Pri študovaní funkcionality sme nahliadli aj do implementácie iných knižníc, ako príklad môžeme uviesť Suds. To či prístup s dynamickými triedami alebo s jednoduchými Pythonovskými štruktúrami je lepší, si netrúfame rozhodnúť. Zdá sa nám skôr ako otázka vkusu a nie ako merateľný rozdiel vo výkone výsledného projektu.

Dostali sme lekciu o tom, že aj závažné problémy môžu byť spôsobené triviálnymi nedostatkami, preto je lepšie radšej všetko kontrolovať, aj keď sa to zdá byť zbytočné a banálne, lebo môže nám to uľahčiť život a urýchliť prácu. Taktiež sme zistili, že hlavná vetva projektu na GitHube nie je nutne zárukou korektnosti, a zmena Pythonovej verzie, na ktorú sa spolieha operačný systém nie je síce natoľko devastujúca, ako klasický zlomyseľný vtip informatikov: `sudo rm -rf /`, ale nie je to ani dobrý nápad.

Ako návrh do budúcnosti, si myslíme že by bolo dobré napísať chýbajúcu dokumentáciu k projektu, čo sme žiaľ nestihli. Pre nadšencov SOAP a každému kto potrebuje pracovať s webovými službami v jazyku Python silne odporúčame pozrieť knižnicu Zeep, ktorú sme si všimli až v poslednej chvíli. Je to nový, rýchlo rozvíjajúci sa projekt ktorý vyzerá sľubne.

Použitá literatúra a iné zdroje

[1] R. Fielding, J. Reschke: Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content, 2014 [Online]

http://tools.ietf.org/html/rfc7231

[2] Ilya Grigorik: High Performance Browser Networking, 2013 [Online]

http://chimera.labs.oreilly.com/books/1230000000545/ch09.html

[3] William Bowers: Big list of http static server one-liners, 2013 [Online]

https://gist.github.com/willurd/5720255

[4] Erik Christensen, Francisco Curbera, Greg Meredith, Sanjiva Weerawarana:

Web Services Description Language (WSDL) 1.1, 2001 [Online]

https://www.w3.org/TR/wsdl

[5] Roberto Chinnici, Jean-Jacques Moreau, Arthur Ryman, Sanjiva Weerawarana: Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language, 2007 [Online]

https://www.w3.org/TR/wsdl20/

[6] Russell Butek: Which style of WSDL should I use?, 2005 [Online]

https://www.ibm.com/developerworks/library/ws-whichwsdl/

[7] Eric Newcomer: Understanding Web Services, 2002 [Online]

http://buhoz.net/public/books/web/web_services/Understanding.Web.Services.XML.WSDL.SOAP.and.UDDI.pdf

[8] Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, Henrik Frystyk Nielsen, Anish Karmarkar, Yves Lafon: SOAP Version 1.2 Part 1: Messaging Framework (Second Edition), 2007 [Online]

https://www.w3.org/TR/soap12-part1/

[9] Claire Hunsaker: REST VS SOAP: When Is REST Better?, 2015 [Online]
https://stormpath.com/blog/rest-vs-soap

[10] Mariano Reingart: Python Simple SOAP Library, 2016 [Online]
https://github.com/pysimplesoap/pysimplesoap

[11] w3schools: XSD Numeric Data Types, 2016 [Online]
http://www.w3schools.com/xml/schema_dtypes_numeric.asp

[12] Python documentation: Names for built-in types [Online]
https://docs.python.org/2/library/types.html

[13] Python documentation: Data model [Online]
https://docs.python.org/2/reference/datamodel.html

[14] Michael van Tellingen: Zeep: Python SOAP client, 2016[Online]
http://docs.python-zeep.org/en/latest/