TESTIRANJE SPREJEMLJIVOSTI IZVEDBE SOAP SPLETNIH STORITEV SISTEMA SUMO

Robert Meolic[1], Janko Koležnik[1], Miroslav Beranič[2]
[1]EIMV Ljubljana, [2]Bintegra d.o.o. Maribor
robert.meolic@eimv.si, janko.koleznik@eimv.si, miroslav.beranic@bintegra.com

Povzetek

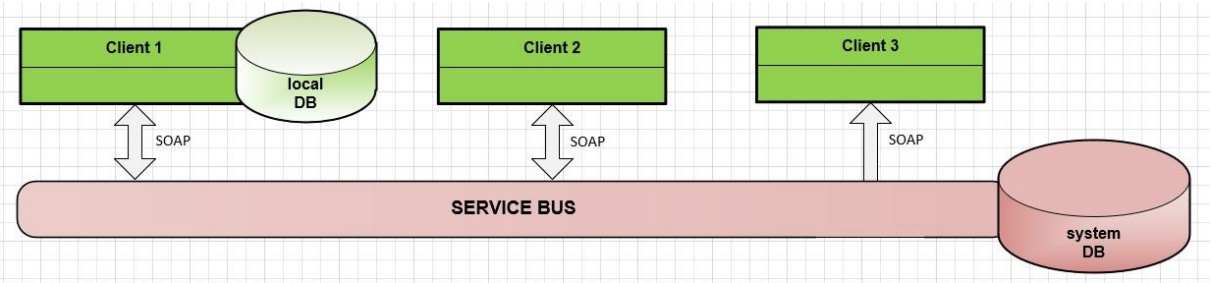
SUMO je Dynamic Thermal Rating (DTR) sistem, ki ga je razvil EIMV v sodelovanju z Inštitutom Jožef Stefan, Fakulteto za elektrotehniko Univerze v Ljubljani, podjetjem Bintegra in ELES-om, ki je hkrati tudi naročnik. Sistem SUMO je zgrajen kot storitveno usmerjena arhitektura. Njegova osrednja komponenta je integracijski vmesnik, ki ga predstavlja storitveno vodilo. Komunikacija med odjemalci in vodilom poteka preko SOAP spletnih storitev. V omrežje je SUMO vključen od leta 2013 in je trenutno v projektu nadgradnje učinkovitosti ter funkcionalnosti, hkrati pa se njegova uporaba širi izven Slovenije. S povečanjem kompleksnosti sistema se zelo povečuje zahtevnost testiranja na vseh ravneh v postopku razvoja in prevzema. V tem prispevku orišemo, kako so spletne storitve v sistemu SUMO izvedeni in prikažemo rešitve za testiranje sprejemljivosti izvedbe ter pripravo dokumentacije.

Ključne besede: SOA, Apache Karaf, Apache Camel, testiranje, FAT, SAT, SoapUI

1. UVOD

Elektroenergetski sistemi spadajo po vseh parametrih med velike sisteme in tako je tudi glede količine podatkov ter glede števila komponent sistema, ki delajo s podatki. Že dolgo časa ne gre več za lokalno omejene ter nepovezane naprave, danes se pričakuje, da bo podatek pridobljen na enem koncu omrežja porabljen na drugem koncu. Pri čemer si želimo, da so podprti in vključeni tako proizvodnja in distribucija kot tudi končni odjemalci.

Uveljavljen pristop pri izgradnji takšnih porazdeljenih sistemov je storitveno usmerjena arhitektura (angl. service-oriented architecture, SOA), kot je prikazana na sliki 1. Njen osrednji del je storitveno vodilo, ki ponuja enovit in univerzalen vmesnik. Za izmenjavo podatkov lahko izberemo različne tehnologije, npr. spletne storitve (SOAP, REST), komunikacijo s sporočili (MQTT) ali razne oblike neposredne komunikacije (OPC UA) [4].



Slika 1: Service-oriented architecture (SOA)

Arhitektura sistema SUMO [5,6,7] temelji na SOAP spletnih storitvah. Povpraševanja in odgovori so torej vdelani v datoteke XML, ki si jih preko protokola HTTP izmenjujejo odjemalci in storitveno vodilo. Na voljo je okoli 30 spletnih storitev (odvisno od verzije

sistema), ki imajo v povprečju od 5 do 10 metod. Celoten sistem je izveden z odprtokodnimi tehnologijami Apache Karaf, Apache CXF, Apache Camel, Apache ActiveMQ in PostgreSQL.

Problem, katerega rešitev opisujemo v članku je, kako učinkovito in elegantno opraviti testiranje sprejemljivosti sistema SUMO ter hkrati pripraviti dokumentacijo za uporabnike – programerje končnih aplikacij na oddaljenih odjemalcih, ki uporabljajo storitve SUMO.

2. TESTIRANJE PROGRAMSKE OPREME

Testiranje je ena od komponent programskega inženirstva v vseh razvojnih modelih. Cilji testiranja so vedno podobni, le da so v različni literaturi podani na različne načine (glej npr. številne razprave o standardu ISO/IEC/IEEE 29119 Software Testing). Po najbolj uveljavljeni definiciji je testiranje »izvajanje programske opreme z namenom iskanja napak« [2]. Testiranju sorodna izraza sta verifikacija in validacija. Verifikacija je dokazovanje, da sta specifikacija zasnove in izvedba programske opreme pravilni glede na dane zahteve (angl. Have we built the software right?), validacija pa je preverjanje, da je programska oprema uporabna na način, kot si je to ob naročilu predstavljal naročnik (angl. Have we built the right software?). Gre za tri različne metode, ker npr. iskanja napak (testiranje) gotovo ne gre enačiti z dokazovanjem odsotnosti napak (verifikacija). Navedeni izrazi se sicer včasih pomešajo, tudi zato, ker se metode med seboj dopolnjujejo in se lahko časovno ali celo vsebinsko prekrivajo (npr. rezultat verifikacije se uporabi pri testiranju).

Vse omenjene metode se nanašajo na testiranje v fazi razvoja programske opreme, ki pa ni predmet tega prispevka. Testiranje sprejemljivosti (angl. Acceptance Testing) je proces, ki se izvaja po končanem razvoju in preverja sprejemljivost izvedenega sistema s strani naročnika (angl. Validate that the product can work!). Po svoji vlogi je testiranje sprejemljivosti podobno drugim metodam validacije sistema, torej ima cilj vzpostaviti zaupanje naročnika v izveden sistem [8]. Ločimo FAT (angl. Factory Acceptance Test) in SAT (angl. Site Acceptance Test). Razlika med FAT in SAT je le v tem, da se FAT opravi v testnem okolju (vhodni podatki so simulirani), SAT pa v produkcijskem okolju (vhodni podatki so iz produkcije).

Bistveni del testiranja sprejemljivosti je preizkušanje delovanja programske opreme na ciljni strojni opremi. Poudarek je na preverjanju nefunkcionalnih zahtev, vključene funkcionalne zahteve pa se preverjajo po metodi črne skrinjice. Pripravljeni testi sprejemljivosti za sistem SUMO obsegajo:

- preverjanje namestitve programske opreme,
- preverjanje delovanja programske opreme po ponovnem zagonu strojne opreme,
- preverjanje mehanizmov za spremljanje delovanja (nadzor dnevnikov),
- preverjanje sklopljenosti komponent sistema in povezovanja z okolico,
- funkcionalno in obremenitveno preverjanje SOAP spletnih storitev.

V nadaljevanju prispevka se osredotočimo le na oris izvedbe SOAP spletnih storitev ter njihovo funkcionalno preverjanje. Poudarek je na avtomatizaciji testiranja in dokumentiranja, s katero smo dosegli naslednja dva cilja:

- avtomatično preverjanje delovanja spletnih storitev za predvidene primere uporabe,
- avtomatično dokumentiranje spletnih storitev.

3. IZVEDBA SOAP SPLETNIH STORITEV SISTEMA SUMO

3.1 Poslovna logika

Poslovna logika je implementirana z uporabo programskega jezika Java 8. Apache Karaf služi kot aplikacijski strežnik, kamor se nameščajo svežnji s programsko kodo. Svežnji so izvedeni v skladu s specifikacijo OSGi kar zagotavlja modularizacijo.

Modularizacija rešitve omogoča, da omejen nabor funkcionalnosti testiramo v predvidljivem okolju in simulaciji različnih testnih scenarijev.

3.2 SOAP spletni vmesniki

Spletni vmesniki so izvedeni z uporabo specifikacije SOAP/WSDL, za izmenjavo XML/XSD podatkovnih sporočil. Za implementacijsko okolje je uporabljen Apache CXF.

SOAP protokol omogoča opis metod in podatkovnih sporočil, za izmenjavo med strežnikom in odjemalcem. To omogoča predvidljivo delovanje tudi brez konkretne implementacije, kar omogoča lažje testiranje oz. pripravo testov pred končno implementacijo (Test-driven Development, TDD).

3.3 Izmenjava in povezava sporočil

Na strežniški strani za prevzem vhodnih sporočil (med drugimi tudi SOAP zahtev) skrbi ogrodje Apache Camel. Apache Camel je integracijsko ogrodje, ki vhodne zahteve preusmeri na JMS kanale, na katere so priklopljeni odjemalci, ki so izvajalci in naredijo predvideno logiko ter vrnejo odgovor. Ogrodje Apache Camel omogoča, da dinamično spreminjamo povezave med komponentami in unificirajo implementacijske komponente, ki so tako ne-odvisne o vhodnega protokola. Tako se npr. uporablja isti izvajalec zahtev za SOAP in MQTT zahteve, podprti so tudi drugi komunikacijski kanali. Povezani so z uporabo skupnih XSD shem, ki opisujejo podatkovne strukture na vhodu in izhodu. Za implementacijo JMS kanalov se uporablja Apache ActiveMQ.

Dinamično usmerjanje sporočil nam omogoča, da enostavneje in natančneje izvajamo testiranje ter tako omogočamo testabilnost izvorne kode.

3.4 Podatkovna baza

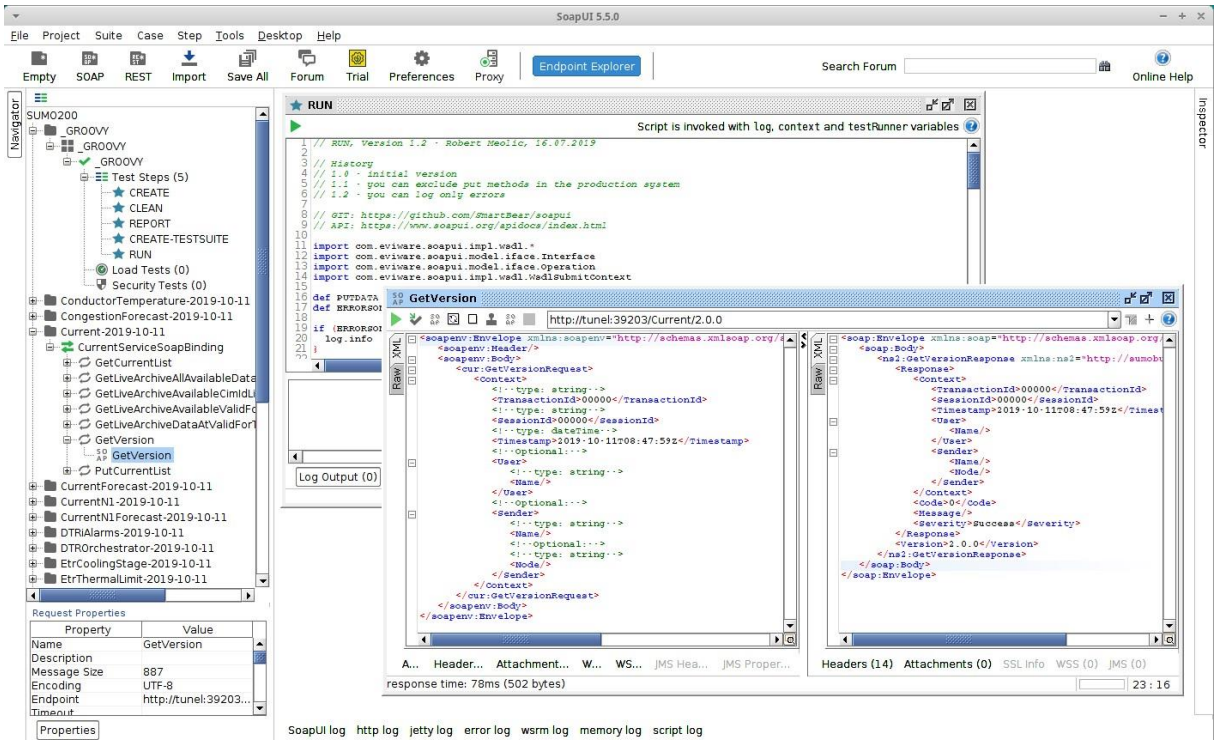
Za hrambo podatkov se uporablja PostgreSQL podatkovna baza. Fizično je baza razdeljena na realno-časovne in zgodovinske podatke, ki so logično predstavljeni v časovni enotnosti. Razdelitev je pomembna, ker se za različna časovna obdobja potrebuje različno granulacijo podatkov. Pri normalnem delovanju sistema se ustvari cca 3GB podatkov na dan.

4. TESTIRANJE SPREJEMLJIVOSTI IZVEDBE SOAP SPLETNIH STORITEV SISTEMA SUMO

4.1 Orodje SoapUI

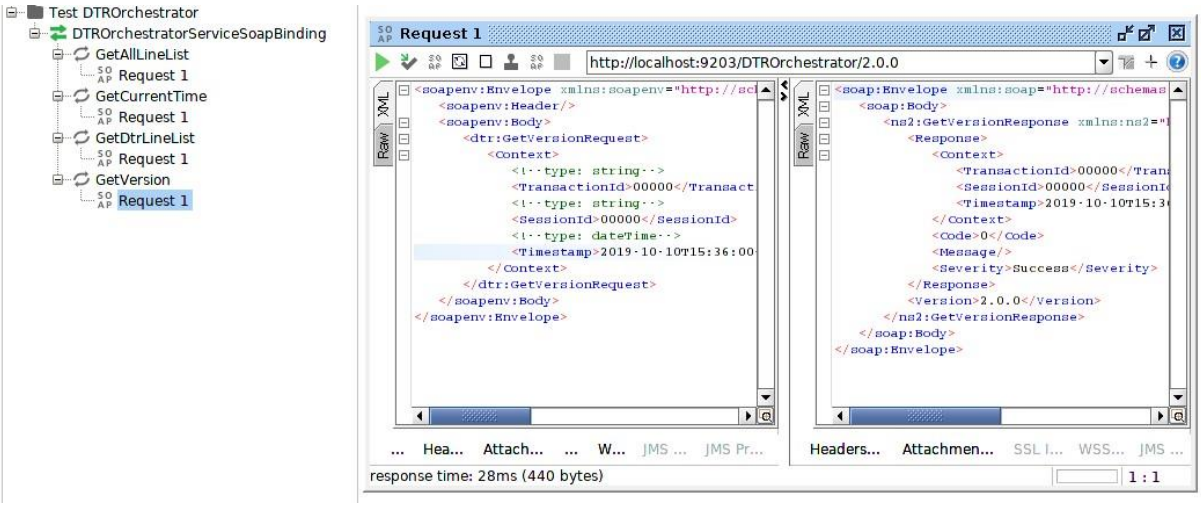
Testiranje SOAP spletnih storitev je bilo avtomatizirano s pomočjo orodja SoapUI, ki je rešitev podjetja SmartBear Software (slika 2). Domača stran tega orodja je

https://www.soapui.org/. Orodje je odprtokodno in vsa izvorna koda se deli v skladišču na naslovu https://github.com/SmartBear/soapui.



Slika 2: Uporabniški vmesnik orodja SoapUI

Osnovna funkcionalnost orodja SoapUI je preizkušanje odziva s pošiljanjem testnih SOAP poizvedovanj. V dialogu, ki ga ponudi orodje, podamo naslov, na katerem je dosegljiva specifikacija WSDL za izbrano storitev (npr. http://localhost:9203/DTROrchestrator/2.0.0?wsdl) in v delovni prostor (angl. Workspace) se doda nov projekt, v katerega so vključena testna poizvedovanja vseh metod izbrane storitve (slika 3). V tvorjenih poizvedovanjih so polja izpolnjena s privzetimi vrednostmi iz specifikacije WSDL oz. so prazna, če privzete vrednosti v specifikaciji niso podane. Ko izvedemo »Submit request«, orodje prikaže dobljen odgovor.



Slika 3: Primer SOAP poizvedbe, ki jo tvori orodje SoapUI ter prikaz odgovora

Orodje SoapUI omogoča, da posamezna poizvedovanja združujemo v testne nabore (angl. test suite). Vsak testni nabor je sestavljen iz enega ali več testnih primerov (angl. Test Case). Testni primeri imajo po enega ali več korakov, ki so najbolj pogosto kar enostavna SOAP poizvedovanja.

4.2 Ogrodje rešitve

Zgoraj opisana funkcionalnost orodja SoapUI je osnova za pripravo testov sprejemljivosti. Da bi bila rešitev koristna za končnega uporabnika, je potrebno izpolniti naslednje zahteve:

- avtomatično se morajo testirati vse metode vseh spletnih storitev iz danega spiska,
- avtomatično se morajo v poizvedovanja vpisati smiselne vrednosti polj, pri čemer se lahko zahteva, da se uporabi vrednost, ki je del odgovora na prejšnjo zahtevo,
- avtomatično se mora preveriti, da imajo vsi odgovori pravilno vsebino,
- vse zahteve in odgovori se morajo dokumentirati,
- generirani test sprejemljivosti mora biti shranjen in ga mora biti možno izvesti oz. ponoviti brez uporabe grafičnega vmesnika.

Za avtomatizacijo testov orodje SoapUI ponuja skriptni jezik Groovy (http://groovy-lang.org/). To je izpeljanka jezika Java in se ga lahko prevaja ali pa tolmači. Rešitev je sestavljena iz več ločenih skript, ki vsaka opravi svoj del naloge:

- Skripta CREATE: Pregleda podan spisek spletnih storitev in za vsakega tvori projekt v trenutnem delovnem prostoru. Za vsako metodo tvori en primer poizvedovanja. Skripta tudi nastavi polja vseh poizvedovanj z željenimi oz. smiselnimi vrednostmi.
- Skripta CREATE-TESTSUITE: V vsakem projektu, ki ga je tvorila skripta CREATE, tvori testni nabor, v katerega vključi en testni primer sestavljen iz vseh obstoječih poizvedovanj Skripta uporabi vrednosti polj, ki jih je pripravila skripta CREATE oz. v zahtevanih primerih prenese rezultate iz predhodnih poizvedovanj v naslednja poizvedovanja. Vsa poizvedovanja in pogoji glede ustreznosti odgovorov se shranijo na način, da jih je možno izvesti brez grafičnega vmesnika.
- Skripta RUN: Izvede vsa poizvedovanja, ki jih je pripravila skripta CREATE. Za vsako preveri status odgovora in izpiše napake. Orodje SoapUI si vse odgovore shranjuje.
- Skripta REPORT: Vsa izvedena poizvedovanja in shranjene odgovore zapiše v dokument tako, da ga lahko nato neposredno vključimo v sistem za obdelavo besedil LaTeX ter avtomatično tvorimo dokument PDF.

V nadaljevanju komentiramo najbolj zanimive dele posameznih skript, generiranje dokumenta PDF in avtomatično izvajanje pripravljenih testov. Pri pripravi skript nam je bila v veliko pomoč izvorna koda orodja SoapUI.

4.3 Skripta CREATE

Osnovna naloga skripte CREATE (slika 4) je, da glede na datoteko WSDL, ki se nahaja na podanem naslovu URL (spremenljivka *pathToWSDL* v kodi na sliki 4) tvori projekte za vse spletne storitve in primere vseh poizvedovanj. Kljub temu, da ta naloga izgleda kompleksna, je rešitev v jeziku Groovy, ki se izvaja v okolju orodja SoapUI, zelo pregledna in jedrnata (slika 4). V prikazani kodi s klici funkcije *replaceAll* nadomestimo privzete vrednosti, ki jih definira

datoteka WSDL, z vrednostmi, ki jih želimo uporabiti med testiranjem. Prikazani sta samo dve zamenjavi, v naši končni rešitvi je takih zamenjav več kot 30.

```
def project = workspace.createProject(projectName, new File(fullProjectPath))
WsdlInterfaceFactory.importWsdl(project,pathToWSDL,false)
for (Interface inface : project.getInterfaceList()) {
   for (Operation op : inface.getAllOperations()) {
      def requestContent = op.createRequest(true)
      requestContent = requestContent.replaceAll("<Timestamp>.*</Timestamp>","<Timestamp>$DATETIME</Timestamp>")
      requestContent = requestContent.replaceAll("<CimId>.*</CimId>","<CimId>$CIMID</CimId>")
      ...
      newRequest = op.addNewRequest(op.getName())
      newRequest.setRequestContent(requestContent)
   }
}
```

Slika 4: Jedro skripte CREATE

4.4 Skripta CREATE-TESTSUITE

Skripta CREATE-TESTSUITE (slika 5) predpostavlja, da je skripta CREATE tvorila projekt za vsako spletno storitev in en primer poizvedovanja za vsako metodo. Tvoriti testne primere sestavljene iz že tvorjenih poizvedovanj, bi zato ob odlični podpori orodja SoapUI bila trivialna naloga, če ne bi bilo dodatnih zahtev. Za zgodovinske podatke v bazi ima sistem SUMO metode, ki vrnejo časovne žige shranjenih podatkov ter metode, ki za podan časovni žig vrnejo podatek, če ta obstaja. Pri slednjih metodah je torej potrebno v poizvedovanju navesti enega od časovnih žigov, ki so rezultat prve metode. Na sliki 5 je prikazano tudi postavljanje trditev, ki se bodo preverjali med testiranjem, npr. polje »Severity« v odgovoru mora biti »Success«.

```
for (WsdlProject project: workspace.getProjectList()) {
   for (Interface inface : project.getInterfaceList()) {
      def ts = project.addNewTestSuite(inface.getName())
      def tc = ts.addNewTestCase(inface.getName())
      for (Operation op : inface.getAllOperations()) {
         if (op.getName().equals("GetLiveArchiveDataAtTimeStampForLine")) {
            tc.addTestStep(PropertyTransfersStepFactory.TRANSFER_TYPE,"TransferTimeStamp")
            def testStep = tc.getTestStepByName("TransferTimeStamp");
            testStep.addTransfer("TransferTimeStamp");
            def transfer = testStep.getTransferByName("TransferTimeStamp");
            transfer.setSourceStepName("GetLiveArchiveAvailableDataTimeStampListForLine");
            transfer.setSourcePropertyName("Response");
            transfer.setSourcePathLanguage(com.eviware.soapui.impl.wsdl.teststeps.PathLanguage["XPATH"]);
            transfer.setSourcePath("""//TimeStamp""");
            transfer.setTargetStepName("GetLiveArchiveDataAtTimeStampForLine");
            transfer.setTargetPropertyName("Request");
            transfer.setTargetPathLanguage(com.eviware.soapui.impl.wsdl.teststeps.PathLanguage["XPATH"]);
            transfer.setTargetPath("""//TimeStamp""");
         }
         def request = op.getRequestAt(0)
         def stepConfig = WsdlTestRequestStepFactory.createConfig((WsdlRequest)request,op.getName())
         def step = tc.addTestStep(stepConfig)
         assert = step.addAssertion("Contains")
         assert.setName("Success")
         assert.setToken("<Severity>Success</Severity>")
      }
   }
}
```

Slika 5: Jedro skripte CREATE-TESTSUITE

4.5 Skripta RUN

Skripta RUN (slika 6) izvede vsa pripravljena poizvedovanja. Izvajanje je v posebni skripti le zato, da je rešitev modularna, kar se v praksi dobro obnese.

```
for (WsdlProject project: workspace.getProjectList()) {
    for (Interface inface : project.getInterfaceList()) {
        for (Operation op : inface.getAllOperations()) {
            def request = op.getRequestAt(0)
            def requestContext = new WsdlSubmitContext(request)
            def submit = request.submit(requestContext,false)
            def response = submit.getResponse()
            if (response != null) {
                def responseMessage = response.getContentAsString()
                request.setResponse(response,requestContext)
            }
        }
    }
}
```

Slika 6: Jedro skripte RUN

4.6 Skripta REPORT

Skripta REPORT (slika 7) uporabi rezultate, ki so v prvi vrsti namenjeni testiranju, ter iz njih tvori priročnik, v katerem so dokumentirana vsa poizvedovanja in odgovori za testirane spletne storitve. Uporabljen je sistem za obdelavo besedil LaTeX, s katerim lahko tvorimo zelo kvalitetne dokumente PDF. Poseben izziv predstavljajo poizvedovanja, ki vrnejo veliko količino podatkov (npr. več sto »tower«-jev), v dokumentaciji pa želimo prikazati le enega. Take situacije lahko rešimo s funkcijami za obdelavo stringov in uporabo regularnih izrazov kot je to prikazano v podani kodi.

```
for (WsdlProject project: workspace.getProjectList()) {
    for (Interface inface : project.getInterfaceList()) {
        def ifName = inface.getName()
        new File("$texdir/$ifName"+".tex").withWriter('utf-8') { writer -> writer.writeLine \\chapter{$ifName} }
        for (Operation op : inface.getAllOperations()) {
            def opName = op.getName()
            def request = op.getRequestAt(0)
            def requestContent = request.getRequestContent()
            def responseContent = request.getResponseContentAsXml()
            new File("$xmldir/$ifName-$opName-request").withWriter('utf-8') { writer -> writer.writeLine requestContent }
            new File("$texdir/$ifName"+".tex").withWriterAppend('utf-8') {
                writer -> writer.writeLine "\\section{$opName}\nREQUEST:\\lstinputlisting{../$xmldir/$ifName-$opName-request}"
            }
            if (responseContent != null) {
                responseContent = responseContent.replaceFirst("(?ms)</Tower>\\s*<Tower>.*</Tower>","</Tower>")
                new File("$xmldir/$ifName-$opName-response").withWriter('utf-8') { writer -> writer.writeLine responseContent }
                new File("$texdir/$ifName"+".tex").withWriterAppend('utf-8') {
                    writer -> writer.writeLine "RESPONSE:\\lstinputlisting{../$xmldir/$ifName-$opName-response}"
                }
            }
        }
    }
}
```

Slika 7: Jedro skripte REPORT

4.7 Generiranje dokumenta PDF in avtomatično izvajanje pripravljenih testov

Skripta REPORT za vsako spletno storitev pripravi tekstovne datoteke za vsa poizvedovanja in odgovore (*$xmldir/$ifName-$opName-request* in *$xmldir/$ifName-$opName-response*) ter datoteko, ki vse te podatke poveže skupaj (*$texdir/$ifName*). K temu dodamo še ogrodje dokumenta (ukazi za formatiranje dokumenta ter besedilo naslovne strani) in iz vsega skupaj s sistemom za obdelavo besedila LaTeX generiramo dokument PDF. LaTeX je uveljavljeno prosto orodje za oblikovanje profesionalnih tehničnih besedil, za katerega je na voljo veliko priročnikov in spletnih virov, za kratek uvod glej na primer [3].

Skripta CREATE-TESTSUITE za vsako spletno storitev pripravi testni primer, ki se shrani v datoteko xml. Te teste lahko nato kadarkoli avtomatično (v terminalu, brez zagona GUI) izvedemo z orodjem *testrunner*, ki je del SoapUI, na naslednji način:

```
testrunner.sh -r -I ./DTROrchestrator.xml
```

5. ZAKLJUČEK

Članek opisuje testiranje sprejemljivosti in dokumentiranje programske opreme, kakor je bilo izvedeno v uspešnem projektu. Podali smo precej tehničnih podrobnosti, s čimer smo ponazorili, kako sta fazi preverjanja (tudi validacija) in dokumentiranja programske opreme v številnih značilnostih podobni fazama načrtovanja ter izvedbe. Ravno tako zahtevata izvirnost pri izvedbi, algoritmično razmišljanje ter odlično tehnično znanje. V splošnem je mogoče za pripravo avtomatiziranega testiranja in dokumentiranja, kar sta splošno sprejeta cilja, uporabiti še mnogo naprednejše metode od teh, ki smo jih uporabili mi (glej npr. [1]). Ne glede na izbran razvojni model velja, da za uspešno izvedbo projektov vlogi testiranja in dokumentiranja ne smemo podcenjevati (ali celo ignorirati) in da se morajo te naloge delegirati ustrezno izobraženim ter izkušenim članom ekipe.

Sistem SUMO je s svojo arhitekturo SOA velik in kompleksen sistem, hkrati pa ima v produkciji značilnosti varnostno-kritičnega sistema (napačno delovanje oz. nedelovanje lahko povzroči materialno škodo). Naročnik in razvijalci sistema so pravilno prepoznali te značilnosti ter primerno izvedli vse faze razvojnega cikla, tudi testiranje in dokumentiranje. S takim pristopom smo tik pred uspešnim zaključkom prve večje nadgradnje sistema in tudi uspešne širitve uporabe sistema v tujino.

6. VIRI

[1] S. Karakatič, T. Schweighofer: A novel approach to generating test cases with genetic programming, Zbornik, 10th International Conference KMO, Maribor, 2015. LNBIP, zv. 224, str. 260-271.

[2] G. J. Myers, C. Sandler, T. Badgett: The Art of Software Testing, John Wiley & Sons, 3rd edition, 2011.

[3] J. Slak: Praktičen uvod v LaTeX, 2018. http://e6.ijs.si/~jslak/files/prakticen_uvod_v_latex.pdf

[4] M. Papazoglou: Web Services and SOA: Principles and Technology, Pearson 2nd edition, 2012

[5] A. Souvent, J. Kosmač, M. Pantoš, R. Vončina, M. Maksić: SUMO - a system for real-time assessment and short-term forecast of operational limits in the Slovenian transmission network, Zbornik, First South East European Regional CIGRÉ Conference, Portorož, 2016.

[6] Š. Vidrih, J. Kosmač, T. Tomšič: Dynamic Thermal Rating System in Slovenian Transmission Power System, Zbornik, 17th IEEE International Conference EUROCON, Ohrid, 2017.

[7] Š. Vidrih, A. Matko, J. Kosmač, T. Tomšič, A. Donko: Operational Experiences with the Dynamic Thermal Rating System, Zbornik, 2nd South East European Regional CIGRÉ Conference, Kyiv, 2018.

[8] What is Acceptance testing. Online: http://tryqa.com/what-is-acceptance-testing/