



Osaamista  
ja oivallusta  
tulevaisuuden  
tekemiseen

Kalle Mustonen

# WsdI2OpenApi: SOAP-rajapinta- kuvauksen muuntaminen OpenAPI- määrittelyksi

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

5.5.2021

Tekijä Otsikko  Sivumäärä Aika	Kalle Mustonen Wsd2OpenApi: SOAP-rajapintakuvauksen muuntaminen OpenAPI-määrittelyksi  35 sivua 5.5.2021
Tutkinto	insinööri (AMK)
Tutkinto-ohjelma	Tieto- ja viestintätekniikka
Ammatillinen pääaine	Ohjelmistotuotanto
Ohjaajat	Lehtori Vesa Ollikainen Ohjelmistoasiantuntija Petri Kiuttu
<p>Insinööritoiminnan tavoitteena oli kehittää työkalu, joka automaattisesti muuntaa WSDL- ja XSD -skeematiedostoilla kuvattua SOAP-rajapinnan OpenAPI 3 -määrittelyn mukaiseksi REST-rajapintakuvaukseksi. Tarkoituksena oli, että työkalu otettaisiin osaksi isompaa projektia, missä toimeksiantaja halusi olemassa olevan SOAP-rajapinnan rinnalle REST-rajapinnan. Työkalun tehtävänä olisi muuntaa nämä olemassa olevat SOAP-rajapintakuvaukset REST-rajapintakuvauksiksi, joiden pohjalta REST-rajapinnan luominen olisi triviaalia. Syntynyttä REST-rajapintakuvausta käytettäisiin myös palvelun testaamisessa sekä lähetettäisiin kolmansille osapuolille tarpeen mukaan. Insinööritoiminnan toimeksiantajana oli suomalainen finanssialan ratkaisuja kehittävä ja palveluita tuottava yritys.</p> <p>Insinööritoiminnan tarkoituksena oli myös perehtyä näihin edellä mainittuihin rajapintakuvausformaatteihin ja määrittelyihin sekä lyhyesti kertoa SOAP- ja REST-teknologioista ja niiden eroavaisuuksista.</p> <p>Insinööritoiminnan lopputuloksena syntyi Wsd2OpenApi-niminen Maven-liitännäinen, joka pysyy automaattisesti muuntamaan WSDL- ja XSD-tiedostoilla kuvattua SOAP-rajapintakuvauksen OpenAPI 3 REST -rajapintakuvaukseksi.</p>	
Avainsanat	WSDL, XSD, SOAP, REST, OpenAPI, Java, Maven

Author Title	Kalle Mustonen Wsd2OpenApi: Transforming SOAP Interface Description into OpenAPI Specification
Number of Pages Date	35 pages 5 May 2021
Degree	Bachelor of Engineering
Degree Programme	Information and Communication Technology
Professional Major	Software Engineering
Instructors	Vesa Ollikainen, Senior Lecturer Petri Kiuttu, Software Specialist
<p>The goal of the study was to develop a tool that could automatically transform a SOAP web service described with WSDL and XSD files into a REST API description according to the OpenAPI 3 specifications. The intention was for the tool to be incorporated into a larger project where the commissioner wanted a REST API alongside their existing SOAP web service. The purpose of the tool would be to convert these existing SOAP web service interface descriptions into REST API descriptions, rendering the creation of the REST API from these descriptions trivial. The REST interface description would also be used to test the service, and it would be sent to third parties as needed. The commissioner is a Finnish company that develops solutions and provides services for the financial sector.</p> <p>The thesis also introduces the above-mentioned interface description formats and definitions, and briefly describes SOAP and REST technologies, and their differences.</p> <p>As a result of the study, a Maven plugin named Wsd2OpenApi was developed, which can automatically convert a SOAP web service interface description described with WSDL and XSD files into an OpenAPI 3 REST interface description.</p>	
Keywords	WSDL, XSD, SOAP, REST, OpenAPI, Java, Maven

## Sisällys

### Lyhenteet

1	Johdanto	1
2	Tavoitteet ja toimintaympäristö	2
2.1	Toimeksiantajan historia	2
2.2	Toimeksiantajan palveluväylä	2
2.2.1	Nykytilanne	2
2.2.2	Tulevaisuus	4
3	SOAP-rajapintakuvaus	5
3.1	SOAP lyhyesti	5
3.2	WSDL	7
3.3	WSDL-elementit	7
3.3.1	Definitions	10
3.3.2	Types	10
3.3.3	Message	11
3.3.4	PortType	12
3.3.5	Binding	12
3.3.6	Service ja port	13
3.4	XSD	13
3.5	Keskeisimmät XSD-elementit	15
3.5.1	ComplexType ja simpleType	15
3.5.2	Sequence ja choice	16
4	REST-rajapintakuvaus OpenAPI 3.0.1	16
4.1	REST lyhyesti	16
4.2	REST API	17
4.3	OpenAPI	19
4.4	OpenAPI 3.0.1 -komponentit	21
4.4.1	Metadata ja info	21

4.4.2	Servers	21
4.4.3	Paths	21
4.4.4	Components ja schemas	23
5	Rajapintakuvauksien erot	23
5.1	SOAP vs. REST	23
5.2	WSDL vs. OpenAPI	24
5.3	Muunnostyökalut WSDL:stä OpenAPI-kuvaukseksi	25
6	Wsd2OpenApi-työkalu	26
6.1	Työkalun tärkeimmät riippuvuudet	26
6.2	Työkalun käyttö	27
6.3	Työkalun toiminta	28
7	Ratkaisun tulokset ja arviointi	32
8	Yhteenveto	32
	Lähteet	34

## Lyhenteet

API	Application Programming Interface. Ohjelmointirajapinta, jonka avulla eri ohjelmat voivat kommunikoida keskenään.
HTTP	Hyper Text Transfer Protocol. Viestintäprotokolla, jonka avulla tietoa voidaan siirtää internetin välityksellä.
JSON	Java Script Object Notation. Tiedostomuoto, jolla voidaan kuvata tietorakenteita. Käytetään yleensä tiedonvälitykseen.
MOJO	Maven plain Old Java Object. Jokainen ajettava Maven-päämäärä (goal) on MOJO. Maven-liitännäinen sisältää yhden tai useamman MOJO:n.
OAS	Open API Specification. Rajapintamäärittely, jota noudattamalla voidaan kuvata REST-palvelun rajapinta.
POM	Project Object Model. XML-tiedosto, missä on kuvattuna kaikki Maven-projektin käyttämät riippuvuudet ja muuta projektiin liittyvää metadataa.
REST	Representational State Transfer. Arkkitehtoninen tyyli, jota noudattamalla voidaan luoda verkkopalvelurajapintoja.
SMTP	Simple Mail Transfer Protocol. Protokolla sähköpostien lähettämistä varten.
SOAP	Tunnettiin ennen lyhenteenä sanoista Simple Object Access Protocol, mutta nykyään se tunnetaan pelkkänä nimenä. SOAP on XML-pohjainen protokolla, jonka avulla verkkopalvelut voivat kommunikoida.
URI	Uniform Resource Identifier. Merkkijono, jonka avulla kerrotaan tietyn tiedon tai resurssin sijainti.

WSDL	Web Services Description Language. XML-merkintäkieleen pohjautuva formaatti, millä kuvataan SOAP-rajapintoja.
XML	Extensible Markup Language. Merkintäkieli, millä voidaan kuvata tietorakenteita.
XSD	XML Schema Definition. XML-merkintäkieleen pohjautuva formaatti, millä määritellään se, kuinka XML-elementit tulee WSDL- tai XML-dokumentissa kuvata.
YAML	YAML Ain't Markup Language. Merkintäkieli, millä voidaan kuvata tietorakenteita. Käytetään yleensä ohjelmistojen konfiguroimiseen.

## 1 Johdanto

Insinöörityössä tutustutaan SOAP- ja REST-rajapintakuvauksiin, niiden eroavaisuuksiin ja siihen, miten SOAP-rajapintakuvaus voidaan automaattisesti muuntaa REST-rajapintakuvaukseksi. SOAP on viestintäprotokolla, jossa yleensä kommunikoidaan verkon ylitse viesteillä XML-formaatissa. REST on arkkitehtoninen tyyli verkkorajapintojen suunnitteluun, mutta yleensä REST-rajapinnoista puhuttaessa tarkoitetaan rajapintaa, joka lähettää ja vastaanottaa viestejä JSON-formaatissa. SOAP ja REST ovat vaihtoehtoisia tapoja toteuttaa tietojärjestelmien välinen kommunikointi ja rajapinta.

Työn toimeksiantajana toiminut Samlink-niminen yritys haluaa olemassa olevan SOAP-rajapintakuvauksensa rinnalle OpenAPI 3.0.1 -määrittelyn mukaisen REST-rajapintakuvauksen, jotta heidän palveluväylänsä voi tarjota sekä SOAP- että REST-rajapinnan. REST-rajapintakuvausta käytettäisiin myös palvelun testaamisessa sekä lähetettäisiin kolmansille osapuolille tarpeen mukaan.

Luvussa 2 kerrotaan tarkemmin työn toimeksiantajan historiasta sekä kuvataan heidän nykytilanteensa, ongelma ja tämän ongelman ratkaiseva toimeksianto tarkemmin. Luvussa 3 kuvaillaan lyhyesti SOAP-rajapinta ja sen ominaisuudet sekä paneudutaan tarkemmin WSDL- ja XSD-formaatteihin, joilla SOAP-rajapinnat kuvataan. Luvussa 4 taas kerrotaan lyhyesti REST-rajapinnan ominaisuudet ja paneudutaan tarkemmin siihen, miten REST-rajapinta kuvataan OpenAPI 3.0.1 -rajapintamäärittelyn mukaisesti. Luvussa 5 vertaillaan näitä kahta eri rajapintakuvausta keskenään ja kerrotaan niiden eroista.

Luvussa 6 esitellään insinöörityössä kehitetty työkalu, jolla voidaan automaattisesti muuntaa WSDL- ja XSD-skeematiedostoilla kuvattu SOAP-rajapinta OpenAPI 3.0.1 -määrittelyn mukaiseksi REST-rajapinnaksi. Luvussa 7 arvioidaan, kuinka hyvin tämä työkalu ratkaisee toimeksiantajan ongelman. Luvussa 8 tehdään yhteenveto koko insinöörityöstä.

Tämä insinöörityö on suunnattu niille, jotka ovat pohtineet sitä, miten SOAP-rajapintakuvaus voidaan muuntaa REST-rajapintakuvaukseksi. SOAP-rajapintoja pidetään huomattavasti raskaampina ja vaikeampina käyttää kuin REST-rajapintoja. Sen takia muutos



yleensä halutaan tehdä. Insinööritoiminta tarjoaa myös yleistä tietoa SOAP- ja REST-rajapintakuvauksista sekä niiden eroista.

## 2 Tavoitteet ja toimintaympäristö

### 2.1 Toimeksiantajan historia

Insinööritoiminnan toimeksiantajana on Samlink, joka on suomalainen finanssialan ratkaisuja kehittävä ja palveluita tuottava yritys. [1.] Liikevaihto vuonna 2019 oli 107,4 miljoonaa euroa, ja henkilöstöä samana vuonna oli 364. Samlinkin pääkonttori sijaitsee Espoon Leppävaarassa ja toinen konttori Jyväskylässä. Samlink on perustettu vuonna 1994. [2.]

Samlinkista tuli virallisesti osa Cognizantia 1.4.2019. Cognizant on kansainvälinen asiantuntijapalveluita tarjoava yritys. Samlinkin Cognizant hankki sen takia, että se halusi vahvistaa pankkialan osaamistaan entisestään sekä vahvistaa asemaansa Pohjoismaissa. [3.]

### 2.2 Toimeksiantajan palveluväylä

Insinööritoiminnan tavoitteena on ratkaista toimeksiantajan palveluväylän rajapintakuvauksiin liittyvä ongelma, joka kuvataan tarkemmin aliluvussa 2.2.2. Toimeksiantajalla on käytössään heidän kehittämänsä palveluväylätuote, joka toimii rajapintana Samlinkin sisäisen tai kolmannelta osapuolelta tulevan liikenteen ja tietokannan välissä. Palveluväylä koostuu lukuisista erilaisista mikropalveluista, joista jokainen on keskittynyt tiettyyn toimialaan tai toimintoon. Palveluväylän mikropalvelut tarjoavat käyttäjilleen SOAP-rajapinnan, joka on kuvattu WSDL- ja XSD-skeemoilla. SOAP on XML-pohjainen viestintäprotokolla, joka kuvataan WSDL- ja XSD-skeemoilla, jotka on kirjoitettu XML-merkkauksielellä.

#### 2.2.1 Nykytilanne

Palveluväylän mikropalveluiden SOAP-rajapinnat on toteutettu ns. "sopimus ensin" (Contract First) -periaatteella, mikä tarkoittaa sitä, että palvelurajapintaa laadittaessa

WSDL- ja XSD-skeemat kirjoitetaan ensin käsin, ja niistä sitten generoidaan tarvittavat luokat, joita ohjelma käyttää. [4.] Vaihtoehtoinen tapa olisi luoda luokat ensin, joista sitten generoitaisiin WSDL- ja XSD -skeemat. Tämän strategian nimi on ”sopimus viimeisenä” (Contract Last). Kuvassa 1 esitetään esimerkki siitä, miten SOAP-rajapinnan operaatio voidaan kuvata WSDL-skeemassa.

```
<wsdl:portType name="SampleDataPortType">
  <wsdl:operation name="getSampleData">
    <wsdl:input message="tns:getSampleDataRequestMessage" />
    <wsdl:output message="tns:getSampleDataResponseMessage" />
  </wsdl:operation>
</wsdl:portType>
```

Kuva 1. Esimerkki SOAP-operaation kuvauksesta WSDL-skeemana.

Kuvassa on rajapinta nimeltään SampleDataPortType, joka sisältää yhden getSampleData-nimisen operaation. Operaatio ottaa vastaan getSampleDataRequestMessage-nimisen elementin ja palauttaa getSampleDataResponseMessage-nimisen elementin. Nämä pyyntö- ja vastauselementit sisältävät sitä informaatiota mitä rajapinnan välillä kulkee. Kuvan getSampleDataRequestMessage, getSampleDataResponseMessage ja muut tarvittavat elementit, kuten binding ja service, on määritelty tarkemmin kuvan ulkopuolella. Näistä elementeistä kerrotaan tarkemmin luvussa kolme.

WSDL- ja XSD-skeemoista generoidaan tarvittavat luokat kehityksessä käytössä olevalle ohjelmointikielelle, kuten Java. Työkaluna voidaan käyttää esimerkiksi Apachen cxf-codegen-plugin-nimistä Maven-liitännäistä [5], joka generoi tarvittavat Java-luokat annetusta WSDL-skeemasta. Kuvan 1 operaatiokuvauksesta generoituisi kuvan 2 näköinen Java-rajapinta.

```
16 @WebService(targetNamespace = "http://mydomain.com/services/SampleData", name = "SampleDataPortType")
17 @XmlSeeAlso({ObjectFactory.class})
18 @SOAPBinding(parameterStyle = SOAPBinding.ParameterStyle.BARE)
19 public interface SampleDataPortType {
20
21     @WebMethod(action = "http://mydomain.com/services/SampleData/getSampleData")
22     @WebResult(name = "GetSampleDataResponse", targetNamespace = "http://mydomain.com/services/SampleData", partName = "response")
23     public GetSampleDataResponse getSampleData(
24         @WebParam(partName = "request", name = "GetSampleDataRequest", targetNamespace = "http://mydomain.com/services/SampleData")
25         GetSampleDataRequest request
26     );
27 }
28
```

Kuva 2. WSDL-skeemasta cxf-codegen-pluginilla generoitu Java-rajapinta.

Liitännäinen generoi rajapintaluokan lisäksi myös muut tarvittavat luokat, kuten Data-Type. SampleDataPortType-rajapinnan toiminnallisuus toteutettaisiin sitten erikseen sen toteuttavassa luokassa.

### 2.2.2 Tulevaisuus

Tulevaisuudessa toimeksiantaja haluaa, että palveluväylän mikropalvelut tarjoavat REST-rajapinnan SOAP-rajapinnan rinnalle, koska kaikki asiakassovellukset eivät halua tai pysty tekemään SOAP-kutsua esimerkiksi sen monimutkaisuuden takia. Tarkoituksena ei kuitenkaan ole muuntaa jo olemassa olevia rajapintoja noudattamaan kaikkia REST:in periaatteita vaan riittää, että nykyinen rajapinta pystyy viestimään JSON-formaatilla niin, että mikä tahansa REST API -asiakassovellus pystyy sitä kutsumaan.

Palveluväylän SOAP-rajapintoja kuvaavat WSDL- ja XSD-skeemat ovat käsin luotuja. Palveluväylä koostuu lukuisista erilaisista mikropalveluista, joten jokaisesta WSDL- ja XSD-skeemoista halutaan luoda OpenAPI 3.0.1 -määrittelyjä vastaavat REST-rajapintakuvaukset. Kuva 3 on esimerkki siitä, miltä kuvan 1 WSDL:llä kuvattu SOAP-operaatio voisi näyttää OpenAPI 3.0.1 -määrittelyn mukaisena REST-rajapintakuvauksena YAML-formaatissa.

```
27   /data/{dataId}:
28   get:
29     summary: Get sample data by id
30     operationId: getSampleData
31     parameters:
32       - name: dataId
33         in: path
34         required: true
35         schema:
36           type: string
37     responses:
38       "200":
39         description: Successful operation
40         content:
41           application/json:
42             schema:
43               $ref: "#/components/schemas/DataType"
```

Kuva 3. Esimerkki REST-operaation kuvauksesta OpenAPI 3.0.1 -määrittelyn mukaisena.

Kuvassa on määritelty operaatio nimeltään `getSampleData`, joka löytyy polun `/data/{dataId}` takaa. Operaatio ottaa vastaan yhden merkkijonotyyppisen `dataId`-nimisen parametrin, joka sijoitetaan osaksi operaatiota kutsuvaa polkua. Operaatio palauttaa `Data-  
Type`-nimisen skeemaolion, joka on tarkemmin määritelty kuvan ulkopuolella.

Näiden uusien REST-rajapintakuvauksien luominen käsin olemassa olevista skeemoista olisi valtaisa työ, joten toimeksiantaja haluaa automatisoida tämän prosessin jollain työkalulla. Insinööriyössä kehitetty `WsdI2OpenApi`-niminen Maven-liitännäinen yrittää ratkaista tämän ongelman automatisoimalla rajapintakuvausmuunnoksen. Työkalun luomia REST-rajapintakuvauksia hyödynnettäisiin myös palvelun integraatiotestaamisessa sekä niitä tarjottaisiin kolmansille tai muille palvelua kutsuville osapuolille, jotka mieluummin haluavat kutsua REST-rajapintaa SOAP-rajapinnan sijaan.

### 3 SOAP-rajapintakuvaus

SOAP-rajapintakuvaus ja sen ymmärtäminen on keskeinen osa insinööriyön tuotoksena tehtyä `WsdI2OpenApi`-nimistä Maven-liitännäistä. Työkalu ottaa sisääntulona SOAP-rajapintakuvauksen WSDL-tiedostona ja muuntaa sen OpenAPI 3.0.1 REST-rajapintakuvaukseksi. Prosessin ymmärtämiseksi tutustutaan aluksi SOAP-rajapintojen toimintaan.

#### 3.1 SOAP lyhyesti

SOAP on XML-pohjainen viestintäprotokolla, jonka avulla kaksi tahoa voi kommunikoida keskenään verkon yli [6]. Viestinnän osapuolet ovat yleensä jokin asiakassovellus ja backend-sovellus. Asiakassovellus lähettää viestin backend-sovellukselle, joka viestin vastaanotettuaan lukee sen ja lähettää vastauksen takaisin kutsujalle. Backend-sovellusta ajetaan yleensä erillisellä taustapalvelimella.

SOAP on alustasta ja ohjelmointikielestä riippumaton protokolla, mikä tarkoittaa sitä, että esimerkin asiakassovellus voi olla esimerkiksi macOS-käyttöjärjestelmällä ajettava JavaScript-sovellus, joka kutsuu Linux-palvelimella ajettavaa Java-kielistä backend-sovellusta. Kuvassa 4 on esimerkki SOAP-pyynnöstä (rivit 1-21) ja vastauksesta (rivit 23-44).

```

1  --- REQUEST ---
2
3  --- HEADERS ---
4
5  POST http://localhost:8088/mockSampleDataBinding HTTP/1.1
6  Accept-Encoding: gzip,deflate
7  Content-Type: text/xml; charset=UTF-8
8  SOAPAction: "http://mydomain.com/services/SampleData/getSampleData"
9  Content-Length: 187
10 Host: localhost:8088
11 Connection: Keep-Alive
12 User-Agent: Apache-HttpClient/4.5.5 (Java/12.0.1)
13
14 --- BODY ---
15
16 <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
17   <soapenv:Header/>
18   <soapenv:Body>
19     <dataId>1234</dataId>
20   </soapenv:Body>
21 </soapenv:Envelope>
22
23 --- RESPONSE ---
24
25 --- HEADERS ---
26
27 HTTP/1.1 200 OK
28 Content-Type: text/xml; charset=utf-8
29 Content-Encoding: gzip
30 Content-Length: 195
31 Server: Jetty(6.1.26)
32
33 --- BODY ---
34
35 <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
36   xmlns:sam="http://mydomain.com/services/SampleDataTypes">
37   <soapenv:Header/>
38   <soapenv:Body>
39     <sam:DataType>
40       <dataId>1234</dataId>
41       <dataValue>Some sample data.</dataValue>
42     </sam:DataType>
43   </soapenv:Body>
44 </soapenv:Envelope>
45

```

Kuva 4. Esimerkki SOAP-pyynnöstä ja vastauksesta.

Pyynnön ja vastauksen ylätunnisteet (HEADERS) sisältävät niihin liittyvää metadataa, kuten käytetty HTTP-metodi (POST, rivi 5) ja viestin sisällön formaatti (text/xml, rivi 7). Yleensä SOAP-viestit on määritelty käyttämään ainoastaan HTTP POST -metodia, joten

viestien pyynnössä on yleensä runko. Viestin runko (BODY) sisältää sitten sitä dataa, joka on jotenkin olennaista rajapinnan toiminnan kanssa. SOAP-viestin juurielementtinä on Envelope-elementti (rivi 16). Envelope-elementti sisältää Header- ja Body-elementit. Header-elementti voi sisältää autentikointiin liittyvää informaatiota tai muuta metadataa. Esimerkissä Header-elementti on kuitenkin tyhjä. Body-elementti sisältää sitä informaatiota, mitä palvelun operaatio ottaa vastaan tai palauttaa. Kyseinen esimerkkioperaatio palauttaa haetun DataType-rakenteen (rivi 39) annetun dataId:n (rivi 19) perusteella.

SOAP-rajapintaoperaatiot, sekä viestien rakenne että niiden sisältöön liittyvät rajoitukset kuvataan WSDL- ja XSD-tiedostoissa.

### 3.2 WSDL

WSDL (Web Services Description Language) on XML-formaattiin perustuva merkkauksielistandardi, jolla voidaan kuvata SOAP-verkkopalvelun rajapinta eli miten sitä kutsutaan ja millaisia operaatioita se sisältää. Myös REST-rajapinta on mahdollista kuvata WSDL:n 2.0-versiolla. Ensimmäinen versio 1.0 kehitettiin vuonna 2000 Microsoftin ja IBM:n yhteistyönä. WSDL:stä on olemassa neljä eri versiota: 1.0, 1.1, 1.2 ja 2.0. [7.] Tässä keskityn kuitenkin versioon 1.1, sillä se on toimeksiantajan palveluväylän käyttämä versio.

### 3.3 WSDL-elementit

WSDL-dokumentti koostuu erilaisista XML-elementeistä. [8.] XML-elementti kuvataan alku- ja lopputunnisteen avulla, ja elementin sisältö on niiden välissä seuraavan esimerkin mukaisesti.

```
1)
<myElement>Elementin sisältö</myElement>

2)
<myElement/>

3)
<parentElement>
  <childElement>Lapsielementin sisältö</childElement>
</parentElement>
```

Esimerkkikoodi 1. Kolme erilaista esimerkkiä XML-elementistä.

Yleensä XML-elementti, jolla on tekstisisältöä tai muuta sisältöä, kuvataan alku- ja elementin sulkevan lopputunnisteen avulla (Esimerkkikoodi 1 esimerkki 1). Elementti, jolla ei ole varsinaista sisältöä voidaan kuvata yhdellä itsensä sulkevalla tunnisteella (Esimerkkikoodi 1 esimerkki 2). Elementin ensimmäisen, eli aloitustunnisteen sisälle voidaan lisätä attribuutteja [9].

```
<myElement myAttributeKey="attribuutinArvo" />
```

Esimerkkikoodi 2. Attribuutti lisättynä itsensä sulkevan elementin sisälle.

Attribuutti on avain-arvo-pari, jolla voidaan määritellä elementin jokin ominaisuus, esimerkiksi id-tunniste tai nimi.

WSDL-dokumentti koostuu seuraavista elementeistä. Luettelo pohjautuu W3C:n WSDL-määrittelyyn [8].

- <definitions> – Toimii WSDL-dokumentin juurielementtinä, jonka sisällä kaikki muut elementit sijaitsevat sekä määrittelee palvelun nimen ja julistaa tarvittavat nimiavaruudet. Nimiavaruus on joukko nimeltään uniikkeja XML-elementtejä. Nimiavaruuksien avulla erotetaan kaksi samannimistä elementtiä toisistaan.
- <message> – Määrittelee rajapinnan operaation pyynnön tai vastauksen sisällön.
- <operation> – Kuvailee rajapinnan metodin ja sisältää metodiin kuuluvat message-elementit.
- <portType> - Sisältää joukon operation-elementtejä.
- <binding> - Elementin avulla määritellään verkkopalvelun käytössä oleva viestiformaatti ja protokolla.
- <port> - Elementin tarkoituksena on yhdistää binding-elementti palvelun verkko-osoitteeseen.
- <service> - Sisältää joukon port-elementtejä.
- <types> - Elementin sisälle määritellään käytössä olevat XSD-skeemaoliot.
- <documentation> - Sisältää elementtiä kuvailevaa dokumentaatiota. Tämän elementin voi sijoittaa minkä tahansa toisen elementin sisälle.
- <import> - Tämän elementin avulla voidaan ottaa haluttuun WSDL-dokumenttiin mukaan muita ulkoisia WSDL- tai XSD-skeemoja.

Kuvassa 5 on yksinkertainen esimerkki WSDL-dokumentista, joka hyödyntää kaikkia yllä mainittuja elementtejä.

```

1  <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2  <wsdl:definitions targetNamespace="http://mydomain.com/services/SampleData"
3    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
4    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
5    xmlns:tns="http://mydomain.com/services/SampleData"
6    name="SampleData">
7
8    <wsdl:types>
9      <xsd:schema>
10       <xsd:import id="SampleData.xsd"
11         schemaLocation="../XSD/SampleData.xsd"
12         namespace="http://mydomain.com/services/SampleData"/>
13     </xsd:schema>
14   </wsdl:types>
15
16   <wsdl:message name="getSampleDataRequestMessage">
17     <wsdl:part name="request" element="tns:GetSampleDataRequest" />
18   </wsdl:message>
19
20   <wsdl:message name="getSampleDataResponseMessage">
21     <wsdl:part name="response" element="tns:GetSampleDataResponse" />
22   </wsdl:message>
23
24   <wsdl:portType name="SampleDataPortType">
25     <wsdl:operation name="getSampleData">
26       <wsdl:input message="tns:getSampleDataRequestMessage" />
27       <wsdl:output message="tns:getSampleDataResponseMessage" />
28     </wsdl:operation>
29   </wsdl:portType>
30
31   <wsdl:binding name="SampleDataBinding" type="tns:SampleDataPortType">
32     <soap:binding style="document"
33       transport="http://schemas.xmlsoap.org/soap/http" />
34     <wsdl:operation name="getSampleData">
35       <soap:operation soapAction="http://mydomain.com/services/SampleData/getSampleData" />
36       <wsdl:input>
37         <soap:body use="literal" />
38       </wsdl:input>
39       <wsdl:output>
40         <soap:body use="literal" />
41       </wsdl:output>
42     </wsdl:operation>
43   </wsdl:binding>
44
45   <wsdl:service name="SampleDataService">
46     <wsdl:documentation>SampleDataService for a WSDL example</wsdl:documentation>
47     <wsdl:port name="SampleDataPort" binding="tns:SampleDataBinding">
48       <soap:address
49         location="http://localhost:9090/services/MyService" />
50     </wsdl:port>
51   </wsdl:service>
52 </wsdl:definitions>
53

```

Kuva 5. Yksinkertainen WSDL-esimerkki.



Kuvassa dokumentoitu palvelu olisi nimeltään SampleDataService, jonka rajapinnassa olisi yksi operaatio nimeltään getSampleData. Kyseinen operaatio ottaisi vastaan GetSampleDataRequest-elementin ja palauttaisi GetSampleDataResponse-elementin. Nämä request- ja response -elementit ovat tarkemmin kuvailtuna ulkoisessa XSD-dokumentissa, johon viitataan rivillä 11.

### 3.3.1 Definitions

Kuvan 5 rivillä 2 sijaitsee kyseisen palvelukuvauksen definitions-elementti. Elementti määrittelee palvelun nimeksi "SampleData" name-attribuutin avulla. Elementti määrittelee myös käytössä olevat uniikit nimiavaruudet. Nimiavaruudet ovat jotain toisia WSDL- tai XSD-dokumentteja, missä käytössä olevat elementit on määritelty. Esimerkiksi rivillä 17 esitelty GetSampleDataRequest-elementti tulee tns-nimisestä nimiavaruudesta. Tns-nimiavaruus on määritelty rivillä 5.

Kohdenimiavaruus määritellään targetNamespace-attribuutilla, ja sen avulla WSDL-dokumentti voi viitata itse itseensä tai jokin toinen WSDL-dokumentti voisi viitata tähän dokumenttiin.

### 3.3.2 Types

Kuvan 5 rivillä 8 sijaitsee types-elementti. Elementin sisällä olevan import-elementin avulla viitataan ulkoiseen XSD-dokumenttiin nimeltään SampleData.xsd. XSD-dokumentin sijainti on määritelty schemaLocation-attribuutin avulla. SampleData-skeema otetaan mukaan kyseiseen palvelukuvaukseen, ja sille määritellään oma nimiavaruus namespace-attribuutin avulla. Määriteltyjen tyyppien ei tarvitse kuitenkaan olla erillisessä XSD-tiedostossa, vaan ne voidaan myös esitellä WSDL-dokumentissa types-elementin sisällä kuvan 6 tavoin xsd:schema-elementin avulla.

```

7   <wsdl:types>
8     <xsd:schema targetNamespace="http://mydomain.com/services/SampleData"
9       xmlns="http://www.w3.org/2001/XMLSchema" xmlns:tns="http://mydomain.com/services/SampleData"
10      elementFormDefault="qualified" attributeFormDefault="unqualified"
11      version="1.0">
12
13     <xsd:element name="GetSampleDataRequest">
14       <xsd:complexType>
15         <xsd:sequence>
16           <xsd:element name="dataId" type="tns:DataIdType" />
17         </xsd:sequence>
18       </xsd:complexType>
19     </xsd:element>
20
21     <xsd:element name="GetSampleDataResponse">
22       <xsd:complexType>
23         <xsd:sequence>
24           <xsd:element name="dataId" type="tns:DataIdType" />
25           <xsd:element name="dataValue" type="xsd:string" />
26           <xsd:choice>
27             <xsd:element name="ChoiceA" type="xsd:string" />
28             <xsd:element name="ChoiceB" type="xsd:string" />
29           </xsd:choice>
30         </xsd:sequence>
31       </xsd:complexType>
32     </xsd:element>
33
34     <xsd:simpleType name="DataIdType">
35       <xsd:annotation>
36         <xsd:documentation xml:lang="en">Data id key</xsd:documentation>
37       </xsd:annotation>
38       <xsd:restriction base="xsd:token">
39         <xsd:maxLength value="10" />
40       </xsd:restriction>
41     </xsd:simpleType>
42
43   </xsd:schema>
44 </wsdl:types>

```

Kuva 6. Schema-elementti types-elementin sisällä.

Yleisin tapa on kuitenkin määritellä schema-elementti ja sen sisältö erillisessä XSD-tiedostossa, joka sitten tuodaan WSDL-dokumenttiin import-elementin avulla.

### 3.3.3 Message

Kuvan 5 riveillä 16 ja 20 sijaitsevat esimerkkikuvauksen message-elementit. Niillä kuvataan sitä dataa, mitä palvelun rajapinnat tuottavat tai ottavat vastaan [10]. Message-elementin sisällä voi olla nolla tai useampi part-elementti. Jokainen part-elementti on yksi parametri operaation metodissa. Part-elementin type-attribuutti viittaa aina johonkin types-elementin sisällä kuvattuun tyyppiin. Esimerkissä getSampleDataRequestMessage

message-elementin part-elementissä viitataan tns-nimiavaruudessa määriteltyyn GetSampleDataRequest-elementtiin. Kyseinen elementti on tarkemmin määritelty SampleData-skeemassa.

### 3.3.4 PortType

Kuvan 5 rivillä 24 sijaitsee esimerkkikuvauksen portType-elementti. PortType-elementti sisältää yhden tai useamman operation-elementin, ja sen tarkoituksena on yhdistää nämä operaatiot message-elementteihin input- ja output-elementtien avulla. Tämä on kaikkein yleisin tapa SOAP-palveluita kuvatessa [11]. Esimerkkikuvauksessa operaation nimi olisi getSampleData, ja se ottaa pyynnössä vastaan GetSampleDataRequest-elementin ja palauttaa GetSampleDataResponse-elementin. Kuva 7 esittää, miltä kyseinen rajapinta näyttäisi Java-koodissa.

```

16 @WebService(targetNamespace = "http://mydomain.com/services/SampleData", name = "SampleDataPortType")
17 @XmlSeeAlso({ObjectFactory.class})
18 @SOAPBinding(parameterStyle = SOAPBinding.ParameterStyle.BARE)
19 public interface SampleDataPortType {
20
21     @WebMethod(action = "http://mydomain.com/services/SampleData/getSampleData")
22     @WebResult(name = "GetSampleDataResponse", targetNamespace = "http://mydomain.com/services/SampleData", partName = "response")
23     public GetSampleDataResponse getSampleData(
24         @WebParam(partName = "request", name = "GetSampleDataRequest", targetNamespace = "http://mydomain.com/services/SampleData")
25         GetSampleDataRequest request
26     );
27 }
28

```

Kuva 7. Esimerkkikuvauksesta generoitu Java-kielen rajapinta.

Kuvan 7 Java-koodi on generoitu kuvan 5 esimerkkikuvauksesta Apache cxf:n cxf-codegen-plugin Maven-liitännäisellä. Kuvasta voidaan havaita, että rajapinnan nimeksi tulee portType-elementin nimi ja metodiksi operation-elementin nimi.

### 3.3.5 Binding

Kuvan 5 rivillä 31 sijaitsee esimerkkikuvauksen binding-elementti. Binding-elementin tarkoituksena on määritellä se, minkä protokollan ylitse portType-elementissä määritellyt operaatiot kuljettavat dataa [12]. PortTypen transport-attribuutti määrää sen minkä protokollan yli viestit kulkevat. Sama portType voidaan määritellä käyttämään useampaa eri protokollaa. Esimerkkikuvauksessa rivillä 32 käytetään soap:binding-elementtiä, jonka transport-attribuutiksi on valittu SOAP HTTP -protokolla. Vaihtoehtoisesti voitaisiin käyttää esimerkiksi sähköpostiviestien lähettämiseen tarkoitettua SMTP-protokollaa.

### 3.3.6 Service ja port

Kuvan 5 rivillä 45 sijaitsee esimerkkikuvauksen service-elementti. Service-elementin sisällä määritellään palvelun käytössä olevat port-elementit. Port-elementti sidotaan tiettyyn binding-elementtiin binding-attribuutin avulla [13]. Kuvassa 5 rivillä 47 port-elementti nimeltään SampleDataPort on sidottu rivillä 31 määriteltyyn SampleDataBinding-nimiseen binding-elementtiin. Port-elementin tarkoituksena on liittää tietty osoite tiettyyn binding-elementtiin. Port-elementin osoite määritellään soap:address-elementin avulla [14].

## 3.4 XSD

XSD on XML-skeemamäärittely, jolla kuvataan XML-dokumentin rakennetta. XSD:n avulla voidaan määrittellä, minkälaisia oikeinkirjoitus- ja muita sääntöjä jonkin tietyn XML-asiakirjan tulee noudattaa [15]. XSD-dokumentteja käytetään yleensä WSDL-dokumenttien rinnalla ja niissä kuvataan kaikki WSDL-dokumenteissa käytössä olevat tyypit. XSD-dokumentit koostuvat monista erilaisista elementeistä. XSD-elementtejä on olemassa paljon erilaisia, ja ne voidaan jakaa kahdeksaan eri kategoriaan [16].

- Ylätason elementit (Top Level Elements) – Elementit, jotka voidaan esitellä ylimmällä mahdollisella tasolla, eli schema-elementin suorana lapsena. Näitä elementtejä ovat mm. element, simpleType ja complexType.
- Partikkelit (Particles) – Elementit, joille voidaan määrittellä minOccurs- ja maxOccurs-attribuutit. Kyseisillä attribuuteilla voidaan säädellä niiden sallittuja esiintymiskertoja. Partikkeleiksi luetaan mm. sequence- ja choice-elementit, sekä element-elementti kuuluu myös tähän joukkoon.
- Useita XML-dokumentteja ja nimiavaruuksia (Multiple XML Documents and Namespaces) – Tähän kategoriaan kuuluvien elementtien avulla voidaan tuoda uusia skeemaelementtejä mukaan toisista nimiavaruuksista tai uudelleen määrittellä saman nimiavaruuden elementtejä. Näitä elementtejä ovat import-, include- ja redefine-elementit.
- Identiteettirajoitteet (Identity Constraints) – Elementin tunnistamiseen liittyvät elementit.
- Attribuutit (Attributes) – Mahdollistaa erilaisten attribuuttien määrittämisen skeemassa.
- Nimetyt Skeemaoliot (Named Schema Objects) – Elementit, joille voidaan määrittellä nimi. Nimettyjä olioita ovat mm. attribute-, complexType- ja simpleType-elementit.
- Complex Type -määitykset (Complex Type Definitions) – Elementit, jotka luovat ComplexType-määityksiä.

- Simple Type -määrittelyt (Simple Type Definitions) – Elementit, jotka luovat SimpleType-määrittelyksiä.

Kuvassa 8 on kuvan 5 WSDL-kuvaukseen tuotu XSD-skeema.

```

1  <xsd:schema elementFormDefault="unqualified" version="1.0"
2    targetNamespace="http://mydomain.com/services/SampleData"
3    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
4    xmlns:tns="http://mydomain.com/services/SampleData">
5
6    <xsd:element name="GetSampleDataRequest">
7      <xsd:complexType>
8        <xsd:sequence>
9          <xsd:element name="dataId" type="tns:DataIdType" />
10         </xsd:sequence>
11       </xsd:complexType>
12     </xsd:element>
13
14     <xsd:element name="GetSampleDataResponse">
15       <xsd:complexType>
16         <xsd:sequence>
17           <xsd:element name="dataId" type="tns:DataIdType" />
18           <xsd:element name="dataValue" type="xsd:string" />
19           <xsd:choice>
20             <xsd:element name="ChoiceA" type="xsd:string" />
21             <xsd:element name="ChoiceB" type="xsd:string" />
22           </xsd:choice>
23         </xsd:sequence>
24       </xsd:complexType>
25     </xsd:element>
26
27     <xsd:simpleType name="DataIdType">
28       <xsd:annotation>
29         <xsd:documentation xml:lang="en">Data id key</xsd:documentation>
30       </xsd:annotation>
31       <xsd:restriction base="xsd:token">
32         <xsd:maxLength value="10" />
33       </xsd:restriction>
34     </xsd:simpleType>
35
36   </xsd:schema>
37
38

```

Kuva 8. Yksinkertainen XSD-esimerkki.

Kuvassa on kuvailtu GetSampleDataRequest-, GetSampleDataResponse- ja DataIdType-nimiset elementit. GetSampleDataRequest-elementillä on lapsenaan yksi dataId-elementti, joka on tyypiltään DataIdType. DataIdType-elementti on tyypiltään token,

mutta sen maksimipituus on rajoitettu kymmeneen merkkiin. `GetSampleDataRequest`-elementillä on lapsenaan `dataId`- ja `dataValue`-elementit. `DataId`-elementti viittaa `DataIdType`-elementtiin ja `dataValue`-elementti on tyypiltään merkkijono.

### 3.5 Keskeisimmät XSD-elementit

Erilaisia XSD-elementtejä on todella paljon, ja niiden kaikkien läpikäyminen tässä insinööriyössä ei olisi lukijalle eikä kirjoittajalle mukavaa, joten olen päättänyt kertoa vain insinööriyölle keskeisimmät XSD-elementit. Insinööriyön tuotoksena saadun `WsdI2OpenApi`-nimisen Maven-liitännäisen kannalta keskeisimmät XSD-elementit ovat `complexType`, `simpleType`, `sequence` ja `choice`. Niistä `ComplexType` ja `simpleType` ovat yleisimpiä elementtejä XSD-skeemoissa. `Sequence` ja `choice` -elementtien muuntaminen OpenAPI 3 -kuvaukseen ei ole niin yksinkertaista kuin esimerkiksi `maxLength`-elementin.

#### 3.5.1 `ComplexType` ja `simpleType`

Kuvan 8 rivillä 15 on XSD-esimerkin `complexType`-elementti. `ComplexType` on elementti, joka yleensä sisältää toisia elementtejä ja attribuutteja [17.]. Esimerkissä `complexType` sisältää lapsenaan `sequence`-elementin, jonka sisällä on taas lueteltu muita elementtejä.

Kuvan 8 rivillä 27 on XSD-esimerkin `simpleType`-elementti. `SimpleType` on elementti, jolle voidaan määritellä nimi ja muita attribuuttielementtejä. Sen tarkoituksena on yleensä viitata johonkin olemassa olevaan yksinkertaiseen tyyppiin, kuten `string`-tyypin merkkijonoon, ja asettaa sille rajoituksia `restriction`-elementin avulla. `SimpleType` ei kuitenkaan voi sisältää esimerkiksi `sequence`-elementtiä. [18.]

Esimerkissä `DataIdType`-elementille on kirjoitettu dokumentaatiota `annotation`- ja `documentation`-elementtien avulla. `Restriction`-elementin `base`-attribuutiksi on määritelty token, ja maksimipituus on rajattu kymmeneen merkkiin `maxLength`-elementin avulla.

### 3.5.2 Sequence ja choice

Kuvan 8 rivillä 16 on XSD-esimerkin sequence-elementti. Sequence-elementti on elementti, jolla voi olla mielivaltainen määrä lapsielementtejä. Yksittäisen lapsielementin olemassaolon pakollisuuden voi määritellä minOccurs-attribuutilla [19]. XSD-esimerkin sequence-elementti sisältää kolme lapsielementtiä, ja ne ovat dataId, dataValue ja choice-elementti. MinOccurs-attribuutin vakioarvo on 1, joten esimerkin kaikki lapsielementit ovat pakollisia.

Kuvan 8 rivillä 19 on XSD-esimerkin choice-elementti. Choice-elementti on elementti, joka sallii, että ainoastaan yksi sen lapsielementeistä on olemassa samanaikaisesti [20]. XSD-esimerkin choice-elementti sisältää kaksi lapsielementtiä nimeltään ChoiceA ja ChoiceB, joten jommankumman näistä on pakko löytyä DataType-elementistä.

## 4 REST-rajapintakuvaus OpenAPI 3.0.1

OpenAPI on yksi käytetyimmistä REST-rajapintakuvauksista, ja sen ymmärtäminen on tärkeä osa insinööriyössä kehitetyn Wsdl2OpenAPI-nimistä Maven-liitännäistä. Liitännäinen muuntaa sisään tulevan WSDL-tiedostolla kuvatun SOAP-rajapintakuvauksen OpenAPI 3.0.1 REST-rajapintakuvaukseksi. OpenAPI 3.0.1 on uusin OpenAPI-määrittelyn versio. REST- ja REST API -käsitteet käydään myös työssä lyhyesti läpi, vaikka ne eivät ole keskeisessä osassa.

### 4.1 REST lyhyesti

REST tulee sanoista Representational State Transfer, ja se on kehitetty vuonna 2000. REST tarkoittaa arkkitehtonista tyyliä, jonka avulla voidaan tarjota standardeja verkon välillä kommunikoivien tietokoneiden välillä, jotta ne voisivat kommunikoida keskenään helpommin. [21.]

REST:in pääperiaatteet:

- Asiakas-palvelin – On olemassa erillinen palvelin, ja erillinen asiakasohjelma, joka sitten kutsuu palvelinta.

- Tilaton (Stateless) – Jokainen asiakaspyyntö tulee olla ymmärrettävissä pelkästään pyynnön sisällöstä, eikä se voi hyödyntää mitään palvelimelle tallennettua tietoa. Asiakassovellus on vastuussa istunnon tilasta.
- Välimuistissa (Cacheable) – Välimuistirajoitukset edellyttävät, että pyynnön vastauksessa on kerrottava, voidaanko sitä säilöä välimuistissa vai ei. Jos vastaus voidaan tallentaa välimuistiin, niin asiakassovellus voi hyödyntää vastauksen tietoa tulevilla pyynnöillä.
- Yhtenäinen rajapinta (Uniform interface) – HTTP-metodeja GET, POST, PUT ja DELETE on käytettävä. URI:a käytetään aina resurssina, ja pyyntö palauttaa aina HTTP-vastauksen, joka sisältää statuksen ja bodyn.
- Kerroksinen järjestelmä (Layered system) – Järjestelmä, missä kukin komponentti ei näe vuorovaikutuksen kanssa olevaa komponenttia pidemmälle.

## 4.2 REST API

Yleensä kun puhutaan REST:stä, niin sillä tarkoitetaan REST API:a, eli REST-rajapintaa. REST API on yleensä palvelinohjelma, joka ottaa vastaan JSON-formatoitua tietoa HTTP-pyyntönä asiakassovellukselta ja palauttaa vastauksen tälle pyynnölle JSON-formaatissa. Kuva 9 on esimerkki REST API:n JSON-pyyntöstä ja vastauksesta.



```
1  --- REQUEST ---
2
3  --- HEADERS ---
4
5  GET http://localhost:8088/data/1234 HTTP/1.1
6  Accept-Encoding: gzip,deflate
7  Content-Length: 0
8  Host: localhost:8088
9  Connection: Keep-Alive
10 User-Agent: Apache-HttpClient/4.5.2 (Java/15)
11
12 --- BODY ---
13
14
15 --- RESPONSE ---
16
17 --- HEADERS ---
18
19 HTTP/1.1 200 OK
20 Date: Fri, 26 Mar 2021 10:35:52 GMT
21 Content-Type: application/json
22
23 --- BODY ---
24
25 {
26   "dataId" : "1234",
27   "dataValue" : "Some sample data."
28 }
29
```

Kuva 9. Esimerkki REST API:n pyynnöstä ja vastauksesta.

Pyynnön ja vastauksen ylätunnisteet (HEADERS) sisältävät niiden viesteihin liittyvää metadataa, kuten käytetty HTTP-metodi (GET, rivi 6) tai vastauksen rungon formaatti (application/json, rivi 21). GET-metodi ei voi sisältää pyynnössään runkoa, joten sen takia pyynnön runko on tyhjä. Pyynnön parametri dataId 1234 on annettu GET-metodin polussa. Jos operaatio vaatisi POST-metodin käyttöä, niin dataId lähetettäisiin pyynnön rungossa kuvan 10 tavoin.

```

30
31 POST http://localhost:8088/data HTTP/1.1
32 Accept-Encoding: gzip,deflate
33 Content-Type: application/json
34 Content-Length: 23
35 Host: localhost:8088
36 Connection: Keep-Alive
37 User-Agent: Apache-HttpClient/4.5.2 (Java/15)
38
39 {
40   "dataId": "1234",
41 }
42

```

Kuva 10. DataId annettu pyynnön rungossa, kun käytössä on POST-metodi.

REST-rajapintojen operaatiot ovat erilaisten polkujen takana. Kuvan 10 operaatio on polun /data takana. Samassa polussa voi myös sijaita toimintoja muille HTTP-metodeille kuin POST. REST-rajapinnassa yleensä POST-metodi on varattu operaatioille, jotka luovat järjestelmään jotain uutta, eikä kuvan 10 mukaisille kyselyoperaatioille. Sen takia kuvan 9 GET-metodiesimerkki soveltaa paremmin REST:in periaatteita.

#### 4.3 OpenAPI

OpenAPI määrittely (OpenAPI specification) tunnettiin ennen nimellä Swagger-määrittely (Swagger Specification), joka on avoimen lähdekoodin formaatti, jonka avulla voidaan kuvailla REST-rajapintoja. Rajapintamäärittelyt kirjoitetaan YAML- tai JSON-tiedostoina, jotka ovat helposti luettavasti. Tämän kaltaisen rajapintamäärittelyn isoin hyöty on siinä, että se voidaan jakaa kolmannelle osapuolelle (taholle, joka haluaa käyttää rajapintaa), ja heidän on helppo ottaa rajapinta käyttöönsä. [22.]

Kuvassa 11 on esimerkki OpenAPI 3.0.1 -määrittelyn mukaisesta dokumentista YAML-tiedostomuotona.

```

1  openapi: "3.0.1"
2  info:
3    version: 1.0.0
4    title: SampleDataService
5    description: Simple data service with one operation
6  servers:
7    - url: http://mydomain.dev.com/v1
8      description: Development server
9    - url: http://mydomain.service.com/v1
10     description: Production server
11  paths:
12    /data/{dataId}:
13      get:
14        summary: Get sample data by id
15        operationId: getSampleData
16        parameters:
17          - name: dataId
18            in: path
19            required: true
20            schema:
21              type: string
22        responses:
23          "200":
24            description: Successful operation
25            content:
26              application/json:
27                schema:
28                  $ref: "#/components/schemas/DataType"
29            default:
30              description: Something went wrong.
31  components:
32    schemas:
33      DataType:
34        required:
35          - dataId
36          - dataValue
37        properties:
38          dataId:
39            type: string
40          dataValue:
41            type: string
42

```

Kuva 11. Yksinkertainen OpenAPI-rajapintakuvausesimerkki YAML-formaatissa.

Kuvan 11 rajapintakuvaus näyttää sen, miltä kuvan 5 WSDL:llä kuvattu SOAP-rajapinta voisi näyttää REST-maailmaan käännettynä. Suurena erona on kuitenkin se, että yleensä SOAP-rajapinta tukee vain POST-metodia, ja kuvassa 10 on käytetty GET-metodia.

## 4.4 OpenAPI 3.0.1 -komponentit

OpenAPI-määrittely koostuu monista erilaisista osakomponenteista [23]. Tutustutaan seuraavaksi näistä osakomponenteista keskeisempiin.

### 4.4.1 Metadata ja info

OpenAPI-dokumentti alkaa aina openapi-komponentilla, jossa määritellään käytössä oleva OAS-versio, eli rajapintamäärittelyn versio. Kuvan 11 rivillä 2 on esimerkkikuvauksen Info-komponentti. Info-komponentti sisältää lisää metadataa kuten palvelun version, nimen ja kuvauksen. Info-komponentti voi sisältää myös muutakin metadataa, kuten yhteyshenkilö tai lisenssitietoja.

### 4.4.2 Servers

Kuvan 11 rivillä 6 on esimerkkikuvauksen servers-komponentti. Servers-komponentti on lista, missä määritellään rajapinnan pohjaosoitteet, mitkä toimivat kutsuttavan operaation pohjaosoitteena. Siinä voidaan erikseen määritellä kehitys- ja tuotantoympäristön sekä muiden mahdollisten ympäristöjen palvelintiedot. Esimerkkikuvauksessa on määritelty kehitys- ja tuotantopalvelimet.

### 4.4.3 Paths

Kuvan 11 rivillä 11 on esimerkkikuvauksen paths-komponentti. Paths-komponentti on joukko polkuja, missä määritellään rajapinnan yksittäiset polut ja niiden polkujen takana olevat HTTP-metodit ja -operaatiot. Servers-osiossa määritelty osoite yhdistettynä paths-osiossa määriteltyyn polkuun on palvelun rajapinnan operaation osoite.

Kuvan 11 rivillä 16 määritellään parameters-komponentti. Se sisältää muun muassa parametrin nimen, tyylin ja sijainnin. Sijainniksi on määritelty path, mikä tarkoittaa sitä, että parametri dataId syötetään osana pyynnön polkua. Esimerkkikuvauksen mukaan GET HTTP -metodilla varustettu pyyntö osoitteeseen `http://mydomain.ser-`

vice.com/v1/data/1234, kutsuisi palvelun tuotantoympäristön getSampleData-operaatiota parametrilla 1234. Kuvassa 12 näytetään, miten paths-osio tulisi kirjoittaa, jos rajapintaa haluttaisiin kutsua POST-metodilla kuvan 10 tavoin.

```
5  paths:
6    /data:
7      post:
8        summary: Get sample data by id via POST
9        operationId: getSampleDataViaPost
10       requestBody:
11         content:
12           application/json:
13             schema:
14               type: object
15               properties:
16                 dataId:
17                   type: string
18       responses:
19         "200":
20           description: Successful operation
21           content:
22             application/json:
23               schema:
24                 $ref: "#/components/schemas/DataType"
25
```

Kuva 12. POST-metodia hyödyntävä operaatiokuvaus.

Kuvan 12 rivillä 10 sijaitsee requestBody-komponentti, minkä avulla määritellään pyynnön runko, formaatti ja sisältö. Esimerkissä sisältö on määritelty JSON-formaatiksi, ja pyynnön runko sisältää yhden olion, jolla on yksi string-tyyppinen arvo nimeltään dataId.

Kuvan 11 rivillä 22 sijaitsee getSampleData-operaation responses-osio. Responses-osiossa määritellään operaation palauttavat vastaukset. Vastaukset on luokiteltu HTTP-tilakoodien mukaan. Koodin 200 alle on määritelty operaation palauttama vastaus pyynnön onnistuessa. Muiden koodien, kuten koodien 404 tai 500 avulla, voidaan määritellä tunnettuja rajapinnan palauttamia virhevastauksia. Tilakoodin sijaan voidaan käyttää myös avainta default, jonka alle voidaan määritellä kaikki vastaukset, joille ei olla erikseen määritelty tilakoodia (kuva 11 rivi 29).

#### 4.4.4 Components ja schemas

Kuvan 11 rivillä 31 sijaitsee esimerkkikuvauksen components-osio. Components-osio sisältää schemas-komponentin, missä määritellään rajapintakuvauksen skeemaoliot. Skeemaolion avulla voidaan määritellä operaation sisään ja ulostulevaa dataa. Skeemaoliot otetaan käyttöön luomalla niihin viittaus toisesta skeemaoliosta \$ref-komponentin avulla.

Kuvan 11 rivillä 33 määritellään DataType-niminen skeemaolio, joka sisältää kaksi pakollista string-tyyppistä kenttää dataId ja dataValue. DataType-skeemaolio merkataan mahdolliseksi paluuarvoksi getSampleData-operaatiolle viittaamalla siihen \$ref-komponentilla (kuva 11 rivi 28).

## 5 Rajapintakuvauksien erot

Rajapintakuvauksien erot ovat oleellisia silloin, kun halutaan ymmärtää, miten muunnos toisesta toiseen kannattaa tehdä. Tämä on hyvin relevanttia varsinkin silloin, kun ollaan ohjelmoimassa tätä varten työkalua tai jos halutaan ymmärtää, mitä tämä työkalu tekee ja miten se toimii.

SOAP ja REST ovat hyvin erilaisia, kuten myös niiden rajapintakuvaukset WSDL ja OpenAPI, vaikka molempien toteutuksella voidaan päästä samaan lopputulokseen eli verkkopalveluun, joka tarjoaa halutun rajapinnan.

### 5.1 SOAP vs. REST

REST:iä pidetään helppokäyttöisempänä, koska sen avulla voidaan viestitellä JSON-formaatissa. SOAP pakottaa käyttämään XML-formaattia, ja POST-metodia HTTP-protokollaa käytettäessä. REST tukee useampia tietformaatteja, muun muassa JSON-formaattia XML-formaatin lisäksi, ja se tukee eri HTTP-metodeja kuten GET, POST, PUT ja DELETE. Sen sijaan SOAP käyttää verkon yli tapahtuvaan tiedonsiirtoon pelkästään POST-metodia HTTP-protokollan kanssa.

XML-formaatin pakottaminen tekee SOAP:sta vähemmän tehokkaamman, koska sitä on raskaampi jäsentää kuin JSON:ia. SOAP:n raskaammat viestit vievät myös enemmän kaistaa kuin kevyemmät REST-pyynnöt. REST:iä pidetään myös joustavampana, vaikka tosin REST pakottaa käyttämään HTTP-protokollaa, mitä SOAP ei tee. SOAP:ssa on myös sisäänrakennettu virheidenkäsittely, joka kertoo kutsujalle, mikä pyynnössä meni pieleen [24.]

REST kattaa yli 70 % kaikista avoimista rajapinnoista. Suurin syy tälle on se, että REST:in ja JSON:in kanssa on helpompi toimia varsinkin silloin, kun kyseessä on julkinen rajapinta, sillä asiakassovelluksen ja palvelimen välille ei välttämättä tarvita erillistä WSDL:n kaltaista sopimusta. Yksinkertaisuus on suurin syy, miksi isot yritykset kuten Amazon ja Google ovat siirtämässä heidän rajapintojaan SOAP:sta REST:iin. Toki yritykset joutuvat vielä jonkin aikaa pitämään SOAP-rajapintojaan yllä ns. legacy-asiakkaiden, eli pitkään asiakkaina olleiden tahojen, takia. [25.]

## 5.2 WSDL vs. OpenAPI

WSDL ja OpenAPI -dokumenttien tarkoituksena on toimia sopimuksena palvelun rajapinnalle, jossa on kuvailtu kaikki palvelun tarjoamat operaatiot. Dokumentin tulee olla sellaisessa formaatissa, joka on sekä ihmisen että koneen luettavissa. WSDL-kuvausta ei kuitenkaan voida suoraan muuntaa OpenAPI-kuvaukseksi sellaisenaan, koska kuvausformaatit ovat erilaisia ja OpenAPI ei tue kaikki samoja ominaisuuksia ja/tai elementtejä kuin WSDL.

WSDL on kirjoitettava XML-formaatin dokumenttina, kun taas OpenAPI voidaan kirjoittaa joko JSON- tai YAML-dokumenttina. Tämän takia WSDL-dokumentit ovat yleensä pidempiä kuin OpenAPI-dokumentit. Eron voi huomata vertaamalla kuvan 5 WSDL-dokumenttia kuvan 11 YAML-dokumenttiin.

XSD:n tarjoamat elementit kuten choice ja sequence on korvattavissa OpenAPI:n oneOf- ja allof-komponenteilla (kuva 13).



Kuva 13. Esimerkki siitä, miten XSD choice- ja sequence-elementit voidaan kirjoittaa OpenAPI 3 -kuvauksessa.

Monet elementit, kuten aiemmin mainittu maxLength, on helppoa kuvata OpenAPI 3 -dokumentissa osana skeemaoliota.

### 5.3 Muunnostyökalut WSDL:stä OpenAPI-kuvaukseksi

Ennen insinööri työn aloittamista tein taustatutkimusta siitä, onko jokin toinen taho jo kehittänyt työkalun, mikä osaisi automaattisesti muuntaa annetun SOAP-rajapintakuvausten OpenAPI 3 -kuvaukseksi. Aluksi tutustuin OpenAPI:n tarjoamaan työkalulistaan openapi.tools [26], enkä löytänyt mitään avoimen lähdekoodin työkalua, mutta yksi maksullinen palvelu löytyi, nimeltään API Transformer.

Apimaticin kehittämä API Transformer tarjoaa maksullisen rajapinnan, mihin kutsuja voi lähettää WSDL- ja XSD -tiedostonsa, ja vastauksena saadaan annettu rajapintakuvaus muunnettuna OpenAPI 3 -tiedostoksi halutussa formaatissa (JSON ja/tai YAML). Apimatic tarjoaa myös tarvittavan kutsujasovelluksen lähdekoodin haluamallaan ohjelmointikielellä, joten tämä kutsujasovellus olisi helposti muokattavissa ja liitettävissä Samlinkin prosessiin. [27.]

Toimeksiantaja ei kuitenkaan halunnut ottaa kyseistä työkalua käyttöön, koska WSDL- ja XSD-tiedostojen lähettäminen ulkoiselle taholle ei kuulostanut hyvältä idealta. Muita syitä olivat palvelun käyttöön kuluvat kulut ja se, että jos kehittäisimme oman työkalun talon sisällä, voisimme paremmin muokata sitä toimeksiantajan tarpeisiin.



## 6 Wsdl2OpenApi-työkalu

Insinöörityön tuloksena ohjelmoitiin Wsdl2OpenApi-työkalu. Kyseessä on Maven-liitännäinen, joka automaattisesti muuntaa WSDL- ja XSD-skeematiedostoilla kuvatun SOAP-rajapinnan OpenAPI 3.0.1 -rajapintamäärittelyn mukaiseksi REST-rajapintakuvaukseksi, joka tallennetaan sekä JSON- että YAML-formaatin tiedostoiksi. Työkalu on ohjelmoitu Java-ohjelmointikielellä noudattaen olio-ohjelmoinnin peruseräitä.

### 6.1 Työkalun tärkeimmät riippuvuudet

Projektin riippuvuuksienhallintana on käytetty Mavenia, ja projektin riippuvuudet ovat määriteltynä Mavenin vaatimassa POM-tiedostossa. Kuva 14 on ote projektin POM-tiedostosta, ja sen tarkoituksena on tuoda lukijalle ilmi, miltä POM-tiedoston sisältö näyttää ja miten siellä on riippuvuudet määritetty.

```
41     <dependencies>
42     ...
43     <dependency>
44         <groupId>org.apache.maven</groupId>
45         <artifactId>maven-plugin-api</artifactId>
46         <version>3.6.3</version>
47         <scope>provided</scope>
48     </dependency>
49     --
```

Kuva 14. Ote projektin POM-tiedostosta.

Riippuvuudet määritellään dependency-tagien sisälle, jotka täytyy sijoittaa dependencies-tagien sisälle. Dependencies-tagien sisälle voidaan määritellä mielivaltainen määrä dependency-tageja. Lisättävästä riippuvuudesta on kerrottava groupId, artifactId ja version. Scope-elementin lisääminen on valinnaista, ja se määrittelee sen, millaisessa tilanteessa riippuvuus halutaan ottaa projektiin mukaan. Projektin maven-plugin-api-riippuvuuden scope-elementin arvoksi on määritetty provided, mikä tarkoittaa sitä, että riippuvuutta hyödynnetään koonti- ja testausvaiheissa, mutta riippuvuutta ei viedä tuotantoversioon. Olettamuksena on se, että jokin toinen taho tarjoaa saman riippuvuuden ajon aikana. [28.]

WsdI2OpenApi-työkalu hyödyntää useita avoimen lähdekoodin lisenssillä varustettuja kirjastoja, joista oheisessa luettelussa on kuvailta näistä tärkeimmät. Kunkin kohdan alussa kaksoispisteellä on eroteltu groupId ja artifactId. Käytin myös kaikista riippuvuuksista uusinta saatavilla olevaa versiota.

- org.apache.maven:maven-plugin-api – Tarvittava riippuvuus maven-liitännäisen luomiseen.
- org.apache.maven:maven-plugin-annotations – Kyseinen riippuvuus tuo mukanaan Java-annotaatioita, jotka helpottavat Maven-liitännäisen kehityksessä.
- org.membrane.soa:membrane-service-proxy – Riippuvuus tuo mukanaan WSDLParser-nimisen luokan, jonka avulla voidaan jäsentää WSDL- ja XSD-tiedostoja.
- org.openapi4j:open-api-parser – Riippuvuus tuo mukanaan OpenApi3Parser-nimisen luokan, jolla voidaan lukea OpenAPI 3 -mukaisia YAML ja JSON -tiedostoja. Työkalu validoi luomansa YAML- ja JSON-tiedostot OpenApi3Parserilla.
- com.fasterxml.jackson.core:jackson-databind – Riippuvuus tuo mukanaan mapper-luokkia, joiden avulla voidaan muuntaa oliopuu JSON-formaatiksi ja JSON-formaatti YAML-formaatiksi.
- junit:junit – Yleinen Java-kirjasto yksikkötestien tekemiseen.
- org.projectlombok:lombok – Riippuvuus tuo mukanaan Java-annotaatioita, jotka vähentävät boilerplate-koodin, eli aina samanlaisena kirjoitettavan pohjakoodin määrää. Koodarin ei tarvitse enää kirjoittaa toistuvia setter- tai getter-metodeja, vaan ne hoituvat käyttämällä lombok:in @Getter, @Setter tai @Data -annotaatioita. Mielestäni jokaisen Java-projektin tulisi käyttää lombok-kirjastoa.

## 6.2 Työkalun käyttö

Työkalu voidaan ajaa Mavenin mvn-komennolla, mutta tarkoituksena on, että työkalu liitetään Samlinkin väylien palveluiden Client-projekteihin, missä kunkin palvelun WSDL-rajapintakuvaus sijaitsee. Client-projektien yhteiseen parent POM -tiedostoon voitaisiin lisätä WsdI2OpenApi-maven-plugin kuvan 15 tavoin.

```

10  <build>
11    <plugins>
12      <plugin>
13        <groupId>fi.samlink.maven.plugins</groupId>
14        <artifactId>Wsd12OpenApi-maven-plugin</artifactId>
15        <version>1.0</version>
16        <executions>
17          <execution>
18            <id>Wsd12OpenApi-exe</id>
19            <phase>package</phase>
20            <configuration>
21              <wsdlPath>target\WSDL\SampleDataService.wsdl</wsdlPath>
22            </configuration>
23            <goals>
24              <goal>Wsd12OpenApi</goal>
25            </goals>
26          </execution>
27        </executions>
28      </plugin>
29    </plugins>
30  </build>
31

```

Kuva 15. Esimerkki siitä, miten Wsd12OpenApi voitaisiin ottaa käyttöön client-projektin POM-tiedostossa.

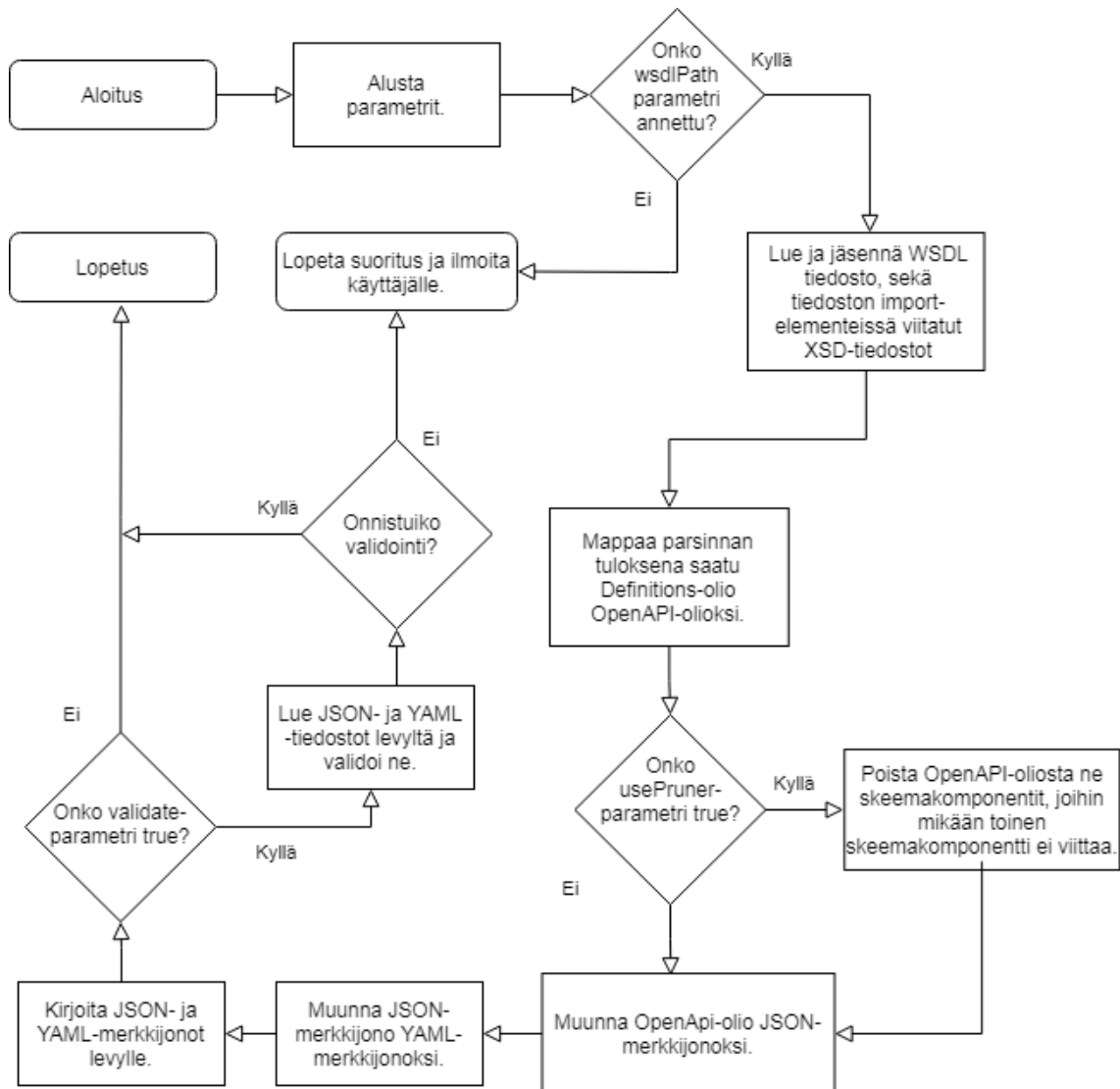
Työkalun on tarkoitus käynnistyä automaattisesti client-projektin rakennusvaiheessa (build) ja generoida projektin WSDL- ja XSD-tiedostoista YAML- ja JSON-formaatin OpenAPI 3 -tiedostot.

Työkalu vaatii käynnistyessään vähintään yhden parametrin nimeltään wsdlPath, joka on luettavan WSDL-tiedoston polku. Myös luotujen tiedostojen polku voidaan erikseen määritellä. Oletuksena se on projektin juuri. Myös kaikki OpenAPI 3 -infokomponentin arvot (title, contact, version, jne.) voidaan muuttaa antamalla niille halutut arvot. Oletusarvot näille on Samlinkin omat. OpenApi3Parserilla tehtävä YAML- ja JSON-tiedostojen validointi on oletusarvoisesti päällä, mutta tämän voi halutessaan ottaa pois päältä, jos ei esimerkiksi välitetä siitä tai luotetaan, että työkalu generoi aina oikeanlaisen OpenAPI 3 -tiedoston.

### 6.3 Työkalun toiminta

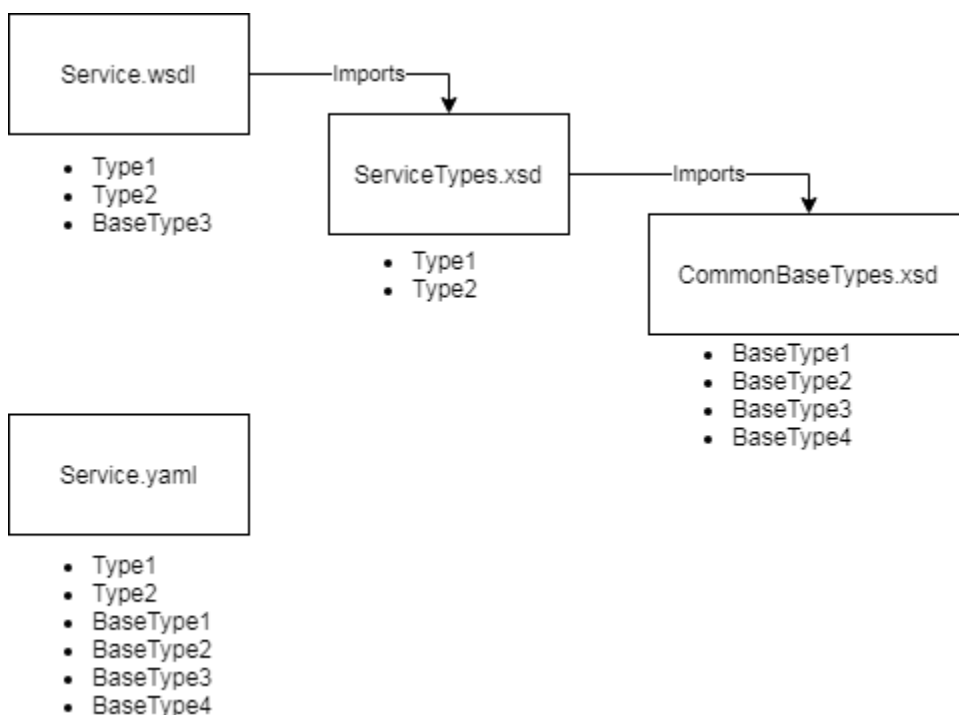
Käynnistyessään työkalu tarkistaa, että wsdlPath-parametri on annettu, ja jos sitä ei ole, niin työkalu lopettaa suorituksensa. Työkalu ilmoittaa tästä käyttäjälleen virheilmoituksen

tekstillä: "wsdlPath empty. Please provide wsdlPath parameter." Työkalun tarkemman toiminnan voi havaita ohjelman vuokaaviosta (kuva 16).



Kuva 16. Wsd2OpenApi-työkalun toiminnan vuokaavio.

Ratkaisussa on kuitenkin se ongelma, että WSDL-tiedostoa luettaessa tuodaan mukaan kaikki sen XSD-tiedostot, jolloin nämä XSD-tiedostot tuovat mukaan myös kaikki heidän XSD-tiedostonsa. Tästä seuraa se, että lopulliseen OpenAPI 3 -kuvaukseen saattaa tulla tyyppejä, joita palvelun kuvaus ei tarvitse. Kuva 17 havainnollistaa tätä ongelmaa.



Kuva 17. Service.wSDL-tiedostossa käytössä olevat tyypit vs. Service.yaml-tiedostoon päätyvät tyypit.

Esimerkkikuvan Service.wSDL-kuvaus tarvitsee XML-tyypit Type1, Type2 ja BaseType3. Näistä Type1 ja Type2 on määritelty palvelun omassa XSD-tiedostossa ServiceTypes.xsd. BaseType3 on määritelty kaikille palveluille yhteisessä XSD-tiedostossa CommonBaseTypes.xsd. Koska työkalu muuntaa kaikki lukemansa WSDL- ja XSD-tiedostot OpenAPI 3 -olioksi, niin se ei tiedä, mitä tyyppejä palvelu oikeasti tarvitsee, jolloin OpenAPI 3 YAML -tiedostoon tulee turhaan palvelulle tarpeettomat BaseType1, BaseType2 ja BaseType4.

Työkalun usePruner-parametrilla voidaan hallita sitä, halutaanko pruner (karsija)-luokkaa käyttää. Oletuksena pruner on true. Pruner karsii OpenAPI 3 -oliosta pois kaikki sellaiset skeematyypit, joihin mikään toinen skeema ei viittaa, jolloin lopulliseen OpenAPI 3 -rajapintakuvaukseen ei päätyisi esimerkin BaseType1, BaseType2 ja BaseType4.

Kuvassa 18 on esimerkki siitä, miltä työkalun tekemä muutos näyttää.

## WSDL

```

1 <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2 <wdd:definitions targetNamespace="http://mydomain.com/services/SampleData"
3 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
4 xmlns:wdd="http://schemas.xmlsoap.org/wsdl/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
5 xmlns:tns="http://mydomain.com/services/SampleData"
6 name="SampleData">
7
8   <wdd:types>
9     <xsd:schema
10       id="SampleData.xsd"
11       schemaLocation="../XSD/SampleData.xsd"
12       namespace="http://mydomain.com/services/SampleData"/>
13   </xsd:schema>
14 </wdd:types>
15
16 <wdd:message name="getSampleDataRequestMessage">
17   <wdd:part name="request" element="tns:getSampleDataRequest" />
18 </wdd:message>
19
20 <wdd:message name="getSampleDataResponseMessage">
21   <wdd:part name="response" element="tns:getSampleDataResponse" />
22 </wdd:message>
23
24 <wdd:portType name="SampleDataPortType">
25   <wdd:operation name="getSampleData">
26     <wdd:input message="tns:getSampleDataRequestMessage" />
27     <wdd:output message="tns:getSampleDataResponseMessage" />
28   </wdd:operation>
29 </wdd:portType>
30
31 <wdd:binding name="SampleDataBinding" type="tns:SampleDataPortType">
32   <soap:binding style="document"
33     transport="http://schemas.xmlsoap.org/soap/http" />
34   <wdd:operation name="getSampleData">
35     <soap:operation soapAction="http://mydomain.com/services/SampleData/getSampleData" />
36     <wdd:input>
37       <soap:body use="literal" />
38     </wdd:input>
39     <wdd:output>
40       <soap:body use="literal" />
41     </wdd:output>
42   </wdd:operation>
43 </wdd:binding>
44
45 <wdd:service name="SampleDataService">
46   <wdd:documentation:SampleDataService for a WSDL example:/wdd:documentation>
47   <wdd:port name="SampleDataPort" binding="tns:SampleDataBinding">
48     <soap:address
49       location="http://localhost:9090/services/MyService" />
50   </wdd:port>
51 </wdd:service>
52 </wdd:definitions>

```

## YAML

```

1 ---
2 openapi: "3.0.1"
3 info:
4   title: "SampleData"
5   contact:
6     name: "My SampleData AB"
7     url: "http://sampledata.fi"
8     email: "info@sampledata.fi"
9   version: "1.0"
10 tags:
11   - name: "SampleData"
12     description: "The SampleData API"
13 paths:
14   /getSampleData:
15     post:
16       tags:
17         - "SampleData"
18       operationId: "getSampleData"
19       requestBody:
20         content:
21           application/json:
22             schema:
23               $ref: "#/components/schemas/getSampleDataRequest"
24             required: true
25       responses:
26         default:
27           description: "default response"
28           content:
29             application/json:
30               schema:
31                 $ref: "#/components/schemas/getSampleDataResponse"
32 components:
33   schemas:
34     DataIdType:
35       title: "DataIdType"
36       maxLength: 18
37       type: "string"
38     getSampleDataRequest:
39       required:
40         - "dataid"
41       allOf:
42         - required:
43             - "datavalue"
44           properties:
45             dataid:
46               title: "dataid"
47               $ref: "#/components/schemas/DataIdType"
48             datavalue:
49               title: "datavalue"
50               type: "string"
51       oneOf:
52         - required:
53             - "choicea"
54           properties:
55             choicea:
56               title: "Choicea"
57               type: "string"
58         - required:
59             - "choiceb"
60           properties:
61             choiceb:
62               title: "Choiceb"
63               type: "string"
64     getSampleDataResponse:
65       required:
66         - "dataid"
67       allOf:
68         - properties:
69             dataid:
70               title: "dataid"
71               $ref: "#/components/schemas/DataIdType"
72

```

## XSD

```

1 <xsd:schema elementFormDefault="unqualified" version="1.0"
2   targetNamespace="http://mydomain.com/services/SampleData"
3   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
4   xmlns:tns="http://mydomain.com/services/SampleData">
5
6   <xsd:element name="getSampleDataRequest">
7     <xsd:complexType>
8       <xsd:sequence>
9         <xsd:element name="dataid" type="tns:DataIdType" />
10       </xsd:sequence>
11     </xsd:complexType>
12   </xsd:element>
13
14   <xsd:element name="getSampleDataResponse">
15     <xsd:complexType>
16       <xsd:sequence>
17         <xsd:element name="dataid" type="tns:DataIdType" />
18         <xsd:element name="datavalue" type="xsd:string" />
19         <xsd:choice>
20           <xsd:element name="choicea" type="xsd:string" />
21           <xsd:element name="choiceb" type="xsd:string" />
22         </xsd:choice>
23       </xsd:sequence>
24     </xsd:complexType>
25   </xsd:element>
26
27   <xsd:simpleType name="DataIdType">
28     <xsd:annotation>
29       <xsd:documentation xml:lang="en">Data id key</xsd:documentation>
30     </xsd:annotation>
31     <xsd:restriction base="xsd:string">
32       <xsd:maxLength value="18" />
33     </xsd:restriction>
34   </xsd:simpleType>
35
36 </xsd:schema>
37

```

Kuva 18. WSDL- ja XSD-tiedostoista muodostuu YAML-tiedosto.

Kuvan vasemmalla puolella on SOAP-rajapintakuvauksen WSDL- ja XSD-tiedostot. Kuvan oikealla puolella on SOAP-rajapintakuvausta vastaava OpenAPI 3.0.1 -määrittelyn mukainen REST-rajapintakuvaus YAML-tiedostona.

## 7 Ratkaisun tulokset ja arviointi

Wsdli2OpenApi-niminen Maven-liitännäinen toimii moitteettomasti käytössä olleen testiaineiston perusteella. Testausaineistona toimi Samlinkin omat WSDL- ja XSD-skeemat, jotka ovat kirjoitettu WSDL-versiolla 1.1. Voi olla mahdollista, että tulevaisuudessa tulee vastaan sellaisia skeemarakenteita, mitä työkalun kehitysvaiheessa ei olla osattu huomioida, jolloin työkalu ei osaa lukea niitä oikealla tavalla. Jos tulevaisuudessa on tarvetta tarjota tukea muille WSDL-versioille, niin se vaatisi hieman työkalun lisäkehitystä. WSDL 2.0 on rakenteeltaan hyvin erilainen kuin WSDL 1.x -version dokumentit, joten tämän version tukeminen vaatisi todella paljon lisäkehitystä.

Valitettavasti projekti missä työkalu oli tarkoitus ottaa käyttöön, on tällä hetkellä keskeytyneenä, ja kyseisen projektin jatkuminen on epävarmaa. Onneksi tulevaisuudessa hämmöttää toinen projekti, missä työkalulle tulee olemaan tarvetta, joten turhaa työtä ei kuitenkaan ole tehty.

Jatkokehitystä työkalulle kuitenkin tarvitaan, koska se jäi hieman kesken edellä mainitun projektin keskeytymisen takia. Työkalu ei tällä hetkellä luo kuin yhden default responses -komponentin OpenAPI 3 -kuvauksen responses-osioon. Tarkoituksena olisi, että jatkossa onnistunut pyyntö olisi "200"-response, ja muut vikakoodit omilla paikoillaan. Myös XSD-tyyppien dokumentointi ei siirry OpenAPI 3 -kuvaukseen tällä hetkellä.

## 8 Yhteenveto

Insinööriyössä perehdyttiin SOAP- ja REST -rajapintakuvauksiin ja luotiin työkalu, joka osaa tehdä muunnoksen SOAP:n WSDL- ja XSD-pohjaisesta rajapintakuvauksesta REST:n OpenAPI 3 -kuvaukseksi.

Työn alussa esiteltiin tavoitteet ja toimintaympäristö sekä kerrottiin lyhyesti toimeksiantajan historiasta. Tämän jälkeen kerrottiin toimeksiantajan palveluväylän rajapinnan nykytilanteesta ja tulevaisuuden suunnitelmista. Toimeksiantajan tulevaisuuden suunnitelmana oli saada REST-rajapintakuvaukset olemassa olevien SOAP-rajapintakuvauksien rinnalle. Toimeksiantaja halusi työkalun, joka osaisi muuntaa SOAP-rajapintakuvauksen REST-rajapintakuvaukseksi.

Työn keskivaiheella esiteltiin rajapintamuunnostyökalun kannalta olennaiset rajapintateknologiat SOAP ja REST sekä niiden rajapintakuvaustavat ja rajapintojen eroavaisuudet. SOAP-rajapinta kuvataan WSDL- ja XSD-dokumenteilla. REST-rajapinta kuvataan OpenAPI 3 -määrittelyn mukaisena JSON- tai YAML-tiedostona. Molemmat rajapintakuvausformaatit käytiin työssä läpi selkeiden esimerkkikuvausten avulla. REST-teknologia osoittautui helpommaksi ja kevyemmäksi vaihtoehdoksi kuin SOAP, sillä se asettaa toteuttajalleen vähemmän rajoituksia.

Työn lopussa esiteltiin rajapintakuvausmuunnoksen tekevä työkalu nimeltään Wsdl2OpenApi. Wsdl2OpenApi on Maven-liitännäinen, joka muuntaa sille annetun WSDL- ja XSD-tiedostoilla kuvatun SOAP-rajapintakuvauksen JSON- ja YAML-tiedostoilla kuvatuksi OpenAPI 3.0.1 -määrittelyn mukaiseksi REST-rajapintakuvaukseksi. Työkalun käyttö ja toiminta kuvattiin esimerkkien avulla. Työkalun kehitys jäi vielä hie- man kesken, mutta jatkokehitystä tehdään sitten, kun sille on tarvetta. Työkalulle tehtiin koodikatselmointi, ja työkalun koodin laatuun oltiin Samlinkilla tyytyväisiä.

Isoimmat insinööriyöhön liittyvät haasteet olivat Wsdl2OpenApi-työkalun suunnittelu- ja kehitysvaiheessa. Minun oli vaikea valita, millä kirjastolla haluaisin tehdä WSDL- ja XSD-dokumenttien jäsentämisen. Vaihtoehtoja oli paljon, mutta yksikään niistä ei heti vaikuttanut toistaan paremmalta. Kokeilujen jälkeen päädyin lopulta käyttämään membrane-service-proxy-kirjaston mukana tulevaa WSDLParser-nimistä luokkaa, sillä se oli todella kevyt ja helppokäyttöinen.

Työkalua tehdessä pohdin sitä, että kuinka moni on vastaavaa työkalua joutunut joskus tekemään. Sitten oivalsin sen, että tästä olisi mahdollista jatkokehittää avoimenlähdekoodin versio, jonka voisi laittaa yleiseen jakoon kaikkien iloksi. Avoimenlähdekoodin versiota vastaavasta työkalusta ei ole tällä hetkellä saatavilla.



## Lähteet

- 1 Samlinkin kotisivut. Verkkoaineisto. <https://www.samlink.fi/> Luettu 1.2.2021.
- 2 Samlinkin yritystiedot. Verkkoaineisto. Finder. <https://www.finder.fi/Sovellukset+ja+ohjelmistot/Oy+Samlink+Ab/Espoo/yhteystiedot/201912> Luettu 1.2.2021.
- 3 Samlinkista osa Cognizanttia. Verkkoaineisto. <https://www.samlink.fi/blog/samlinkista-osa-cognizantia/> Luettu 1.2.2021.
- 4 Defining Contract first webservises with wsdl generation from java. Verkkoaineisto. <http://cxf.apache.org/docs/defining-contract-first-webservises-with-wsdl-generation-from-java.html> Luettu 13.2.2021.
- 5 Maven cxf-codegen-plugin. Verkkoaineisto. <http://cxf.apache.org/docs/maven-cxf-codegen-plugin-wsdl-to-java.html> Luettu 2.4.2021.
- 6 XML Soap. Verkkoaineisto. [https://www.w3schools.com/xml/xml\\_soap.asp](https://www.w3schools.com/xml/xml_soap.asp) Luettu 19.2.2021.
- 7 WSDL - Introduction. Verkkoaineisto. [https://www.tutorialspoint.com/wsdl/wsdl\\_introduction.htm](https://www.tutorialspoint.com/wsdl/wsdl_introduction.htm) Luettu 19.2.2021.
- 8 WSDL - Elements. Verkkoaineisto. [https://www.tutorialspoint.com/wsdl/wsdl\\_elements.htm](https://www.tutorialspoint.com/wsdl/wsdl_elements.htm) Luettu 22.2.2021.
- 9 XML - Elements. Verkkoaineisto. [https://www.tutorialspoint.com/xml/xml\\_elements.htm](https://www.tutorialspoint.com/xml/xml_elements.htm) Luettu 22.2.2021.
- 10 WSDL - <message> Element. Verkkoaineisto. [https://www.tutorialspoint.com/wsdl/wsdl\\_message.htm](https://www.tutorialspoint.com/wsdl/wsdl_message.htm) Luettu 7.3.2021.
- 11 WSDL - <portType> Element. Verkkoaineisto. [https://www.tutorialspoint.com/wsdl/wsdl\\_port\\_type.htm](https://www.tutorialspoint.com/wsdl/wsdl_port_type.htm) Luettu 12.3.2021.
- 12 WSDL - <binding> Element. Verkkoaineisto. [https://www.tutorialspoint.com/wsdl/wsdl\\_binding.htm](https://www.tutorialspoint.com/wsdl/wsdl_binding.htm) Luettu 12.3.2021.
- 13 WSDL - <service> Element. Verkkoaineisto. [https://www.tutorialspoint.com/wsdl/wsdl\\_service.htm](https://www.tutorialspoint.com/wsdl/wsdl_service.htm) Luettu 12.3.2021.

- 14 WSDL - <port> Element. Verkkoaineisto. [https://www.tutorialspoint.com/wsdl/wsdl\\_ports.htm](https://www.tutorialspoint.com/wsdl/wsdl_ports.htm) Luettu 12.3.2021.
- 15 XSD - Overview. Verkkoaineisto. [https://www.tutorialspoint.com/xsd/xsd\\_overview.htm](https://www.tutorialspoint.com/xsd/xsd_overview.htm) Luettu 12.3.2021.
- 16 XML Schema Elements. Verkkoaineisto. [https://docs.microsoft.com/en-us/previous-versions/dotnet/netframework-4.0/ms256142\(v=vs.100\)](https://docs.microsoft.com/en-us/previous-versions/dotnet/netframework-4.0/ms256142(v=vs.100)) Luettu 19.3.2021.
- 17 Complex Type Definitions. Verkkoaineisto. [https://docs.microsoft.com/en-us/previous-versions/windows/desktop/ms764719\(v=vs.85\)](https://docs.microsoft.com/en-us/previous-versions/windows/desktop/ms764719(v=vs.85)) Luettu 31.3.2021.
- 18 Simple Type Definitions. Verkkoaineisto. [https://docs.microsoft.com/en-us/previous-versions/windows/desktop/ms764742\(v=vs.85\)](https://docs.microsoft.com/en-us/previous-versions/windows/desktop/ms764742(v=vs.85)) Luettu 31.3.2021.
- 19 XML Schema sequence Element. Verkkoaineisto. [https://www.w3schools.com/xml/el\\_sequence.asp](https://www.w3schools.com/xml/el_sequence.asp) Luettu 31.3.2021.
- 20 XML Schema choice Element. Verkkoaineisto. [https://www.w3schools.com/xml/el\\_choice.asp](https://www.w3schools.com/xml/el_choice.asp) Luettu 31.3.2021.
- 21 What is REST? Verkkoaineisto. <https://www.codecademy.com/articles/what-is-rest> Luettu 22.3.2021.
- 22 What is OpenAPI? Verkkoaineisto. <https://rapidapi.com/blog/api-glossary/openapi/> Luettu 26.3.2021.
- 23 RESTful APIs: Tutorial of OpenAPI Specification. Verkkoaineisto. <https://medium.com/@amirm.lavasani/restful-apis-tutorial-of-openapi-specification-eeada0e3901d> Luettu 28.3.2021.
- 24 SOAP vs REST. What's the Difference? Verkkoaineisto. <https://smartbear.com/blog/soap-vs-rest-whats-the-difference/> Luettu 29.3.2021.
- 25 REST VS SOAP: When Is REST Better? Verkkoaineisto. <https://stormpath.com/blog/rest-vs-soap> Luettu 3.4.2021.
- 26 OpenAPI.Tools. Verkkoaineisto. <https://openapi.tools/> Luettu 4.4.2021.
- 27 API Transformer. Verkkoaineisto. <https://www.apimatic.io/transformer> Luettu 4.4.2021.
- 28 Maven Dependency Scopes. Verkkoaineisto. <https://howtodoinjava.com/maven/maven-dependency-scopes/#provided> Luettu 30.3.2021.