Tampereen ammattikorkeakoulu

Asiakasintegraation ulkoistamisen tuki

Rajapinnan suunnittelu ja käyttöönotto

Ville Salmén

OPINNÄYTETYÖ
Huhtikuu 2019

Tieto- ja viestintätekniikan koulutus
Ohjelmistotekniikka

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tieto- ja viestintätekniikan koulutus
Ohjelmistotekniikka

SALMÉN VILLE:
Asiakasintegraation ulkoistamisen tuki
Rajapinnan suunnittelu ja käyttöönotto

Opinnäytetyö 63 sivua, joista liitteitä 14 sivua
Huhtikuu 2019

Paikka- ja tilaajatieto ovat hätäpuhelun tehokkaan käsittelyn kannalta tärkeitä soittajan tietoja, joiden automaattista keräämistä varten hätäkeskustietojärjestelmän on toteutettava integraatioita eri palveluihin. Työssä suunniteltiin Insta Response™ –tuoteperheelle entistä laajempi integraatioarkkitehtuuri, jonka myötä pystyttäisiin paremmin ulkoistamaan näiden integraatioiden toteutus jollekin muulle taholle.

Työn tutkiva vaihe koostui selvitystyöstä maailmalla käytössä olevista standardeista liittyen paikka- tai tilaajatieto –tyyppiseen tiedonsiirtoon. Tutkittavina lähteinä oli esimerkiksi Euroopan Hätänumeroyhdistyksen (EENA) dokumentteja liittyen hätäkeskustoiminnan eri osa-alueisiin.

Taustatutkimuksen perusteella työssä toteutettiin paikka- ja tilaajatiedon REST- ja gRPC-rajapintojen määritelmät, OpenAPI:lla ja Protocol Buffereilla kuvattuna. Rajapintojen sisällön suunnittelussa huomioitiin kansainvälisesti tunnettujen standardien käyttö, jotta voitaisiin parantaa rajapintojen yhteensopivuutta eri asiakasympäristöissä.

Suunniteltuja rajapintoja on tarkoitus käyttää Responsesta, ja niiden toteutuksesta huolehtii asiakkaan toimintatavat tunteva integraatiokumppani. Rajapintojen määrittely kuvauskielillä mahdollisti toteutusten generoinnin, käyttäen tarkoitukseen soveltuvia Maven-laajennuksia. Generoinnin myötä voidaan taata esimerkiksi toteutusten yhdenmukaisuutta, kun palvelinpäästä huolehtii jokin muu taho.

Tässä opinnäytetyöraportissa käydään rajapintojen määrittelytiedostojen lisäksi läpi Client-toteutusten generointiin käytettyjä Maven-laajennusten konfiguraatioita. Myös rajapintojen käyttöä generoitujen Java-toteutusten avulla esiteltiin yksinkertaistettujen esimerkkien kautta.

Asiasanat: insta, response, rest, grpc, rajapinta, openapi, protobuf, generointi

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in ICT Engineering
Software engineering

SALMÉN VILLE:
Outsourcing support for customer integration
Interface design and implementation

Bachelor's thesis 63 pages, appendices 14 pages
April 2019

Efficient handling of an emergency call depends on the information about the caller's identity and location being readily available. In order to handle the automatic information gathering process, the CAD-system provider must implement integrations to different ANI and ALI service providers. The subject of this thesis was to design a better integration architecture to the Insta Response™ product family, in order to ease the outsourcing of the integration-work.

The investigating part of the thesis consists of researching already available standards, related to similar location and identity information exchanges. For example, the European Emergency Number Association (EENA) has multiple relevant documents referenced in this thesis.

Based on the research, definitions for ALI and ANI interfaces were made, with support for both REST and gRPC. OpenAPI and Protocol Buffers were used to describe these APIs. In order to improve compatibility in different customer environments, generally known standards were used in the interface definitions.

The designed APIs are meant to be used from Response, and the server-side implementation is done by an integration partner. Using interface description languages for the definitions enabled the use of code generation with related Maven plugins. Code generation ensures consistency between implementations, which is an important factor, when an external party is taking care of the server-side implementations.

In addition to the interface definitions, this report also covers the Maven plugin configurations, which were used to generate the client implementations. Also the use of the interface was demonstrated with simple examples, using the generated Java-client implementations.

Key words: insta, response, rest, grpc, interface, protobuf, generation

SISÄLLYS

LYHENTEET JA TERMIT

Response	Insta Response™. Hätä- ja hälytyskeskustuoteperhe
ERICA	Emergency Response Integrated Common Authorities. Suomen valtakunnallisen hätäkeskustietojärjestelmän kutsumanimi.
client	Asiakassovellus. Jotain palvelua käyttävä ohjelma.
server	Palvelinsovellus. Jotakin palvelua esim. verkon yli tarjoava ohjelma.
OAS	OpenAPI Specification. REST-rajapintojen kuvaamiseen käytetty formaatti.
REST	REpresentational State Transfer. Järjestelmien välinen kommunikaatioarkkitehtuurityyli.
YAML	YAML Ain't Markup Language. Lukijaystävällinen tiedon esitysmuoto
JSON	JavaScript Object Notiation. Lukijaystävällinen tietorakenteiden kuvauskieli
XML	Extensible Markup Language. Tietorakenteiden kuvauskieli.
gRPC	gRPC Remote Procedure Calls. Avoimen lähdekoodin RPC-kirjasto ja protokolla
protobuf	Protocol Buffers. Googlen kehittämä serialisoitavien tietorakenteiden kuvausformaatti.
protoc	Protocol Buffer Compiler. Generoinnissa käytetty kääntäjä.
enumerointi	Tyyppien luettelu kokonaislukuvakioina joille on annettu toinen nimi.
stream	Datan tai esim. olioiden siirtoa tai käsittelyä ikään kuin ”virtana”, jota voi manipuloida erilaisilla operaatioilla.
ISO 3166	Standardi maiden nimien ja alueiden esitykseen
ISO 8601	Standardi päivämäärän ja ajan esitykseen
HTTP	Internetin tiedonsiirtoprotokolla. Vanha versio on 1.1 ja uusi 2.0.
POST	Luomiseen ja tiedon päivitykseen tarkoitettu HTTP-metodi.

GET	Tiedon hakemiseen tarkoitettu HTTP-metodi.
lokalisointiavain	Yksilöivä avain, jolla voidaan hakea sitä vastaava käännetty käyttöliittymäteksti.

1 JOHDANTO

Termi ”matkapuhelin” on tunnettu nykyisessä muodossaan vasta noin 20 vuotta (History of Mobile Phones 2019). Puheluliikenteen siirtyminen perinteisiltä lankapuhelimilta GSM, 3G ja 4G verkkoihin on muuttanut käsitystä siitä, että kuka puhuu, ja erityisesti mistä kyseinen puhelu tulee. Erityisen kiinnostuneita soittajan tunnistamisesta ja paikantamisesta ovat aina olleet hätäkeskuspäivystäjät, koska kyseisillä tiedoilla on tärkeä merkitys esimerkiksi häiriöpuheluiden tunnistamisessa, sekä lähimmän yksikön hälyttämisessä tehtävälle.

Nykypäivänä kun 70% hätäpuheluista soitetaan mobiililaitteista, niin erityisesti paikannukseen käytetyt menetelmät ovat kokeneet radikaaleimman muutoksen lankapuhelinten valtakauteen verrattuna (Mobile Identity 2018). Esimerkiksi mobiililaitteen paikannukseen on olemassa useita tekniikoita, joten hätäpuhelun tehokasta käsittelyä varten olisi tärkeää siirtää eri paikannuslähteiltä saatuja tuloksia automaattisesti päivystäjän nähtäville.

Tämän opinnäytetyön aihe keskittyy juuri edellä mainitun paikannuslähde-hätäkeskuspäivystäjä –tyyppisellä yhteysvälillä tapahtuvaan tiedonsiirtoon ja sen kehittämiseen. Työssä käsitellään päivystäjää kiinnostavaa paikka- ja tilaajatietoa tarjoavien palveluiden integroimista osaksi hätäkeskustietojärjestelmää tavalla, jossa tietojärjestelmän toimittajan ei itse tarvitse ottaa kantaa käytettävissä oleviin tietolähteisiin, vaan integraation voi toteuttaa jokin ulkoinen taho, käyttäen tässä työssä määriteltyä rajapintaa.

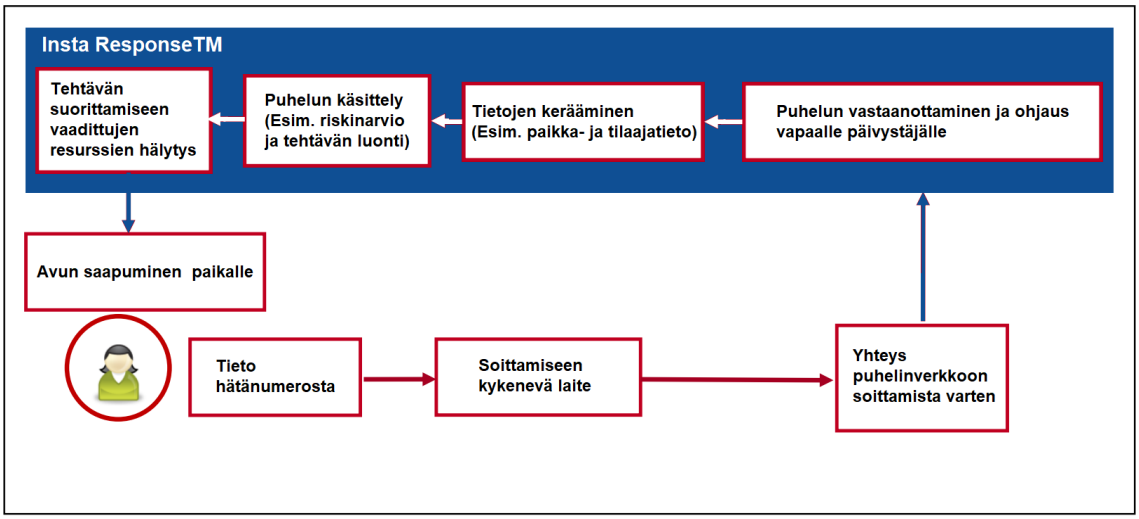
2 PAIKKA- JA TILAAJATIETO SEKÄ INSTA RESPONSE

Toteutettu työ kohdistui hätäkeskustoiminnan tukemiseen tarkoitettuun Insta Response™-tuoteperheeseen. Oleellinen osa hätäkeskustoimintaa on hätäpuheluiden vastaanottaminen, joten puheluiden käsittelyn helpottamiseksi Response-tuoteperhe sisältää mahdollisuuden integroida soittajan tietoja tarjoavia palveluita. Työn tavoitteena oli kehittää entistä laajempi ja konfiguroitavampi integraatioarkkitehtuuri, joka helpottaa uusien palveluiden liittämistä osaksi Responsea.

Seuraavissa alikappaleissa esitellään Response-tuoteperhe, käsitellään hätäpuhelun soittajasta saatavilla olevaa paikka- ja tilaajatietoa, sekä kartoitetaan perusteluita integraatioarkkitehtuurin kehittämisen tarpeelle.

2.1 Insta Response™ hätäkeskustoiminnan tukena

Hätätilanteen syntymisestä avun saapumiseen liittyy useita vaiheita, joista ensimmäinen on jo yksinkertainen tieto hätänumeron olemassaolosta. Hätäpuheluun vastaamisen ja avun lähettämisen välillä tapahtuvat vaiheet voivat vaihdella maakohtaisesti. Kuvassa 1 on havainnollistettu esimerkiksi Suomen hätäkeskustoiminnassa sovellettua palveluketjua, sekä Insta Response™ –tuoteperheen roolia osana Suomen uutta ERICA-hätäkeskustietojärjestelmää. (112 Service Chain Description 2011).



KUVA 1. Response osana 112-palveluketjua (112 Service Chain Description 2011, muokattu)

Insta Response^{TM}-tuoteperhe on suomalaisen perheyritys Insta Group Oy:n kehittämä hätä- ja hälytyskeskusratkaisu. Response-tuoteperheen ominaisuuksia on esimerkiksi yhden virtuaalisen tilannehuoneen muodostaminen useammasta hätäkeskuksesta, mikä mahdollistaa ruuhkan jakamisen keskusten välillä. Kriittisen toimintaympäristön saatavuusvaatimuksien täyttäminen ja kaikkien toimialojen edustajien tarpeisiin vastaaminen ovat myös Responsen kulmakiviä. (Insta DefSec Oy 2019.)

Response-tuoteperhe koostuu kolmesta osasta; operatiiviseen käyttöön tarkoitetusta Insta Response Clientistä, hallinnollisen tiedon käsittelyn työkalusta Insta Response Portalista ja Insta Response Field -kenttäjärjestelmästä. Kaikki tuoteperheen sovellukset toimivat yhdessä yhteisen ohjelmistoalustan kautta. (Insta DefSec Oy 2019.)

2.2 Paikka- ja tilaajatieto

Hätäpuhelun soittajan tunnistaminen ja sijainnin paikannus ovat oleellinen osa hätäpuhelun käsittelyä. Esimerkiksi varmistamalla soittajan identiteetti voidaan suojautua hätänumeron väärinkäytöltä, sekä määrittää tarkempia vaatimuksia mahdolliselle tehtävälle (Mobile Identity 2018, 4).

Tarvittaessa päivystäjä voi myös kysellä paikka- ja tilaajatietoa suullisesti, ja syöttää tiedot manuaalisesti järjestelmään, mutta automaattinen tiedonkeruu nopeuttaa huomattavasti puhelun käsittelyä, sekä oletettavasti parantaa esimerkiksi paikkatiedon tarkkuutta, riippuen käytetyistä paikannusmenetelmistä. Automaattisen paikka- ja tilaajatiedon välittämiseen voi kuitenkin liittyä haasteita esimerkiksi yksityisyydensuojan johdosta.

Paikka- ja tilaajatiedolla tarkoitetaan tämän työn kontekstissa puhelinnumeroa vastaavan puhelimen sijaintia ja puhelinnumeron omistavan henkilön tietoja. Paikkatiedosta käytetään tässä työssä ja Responsen näkökulmasta myös englanninkielistä termiä Automatic Location Identification (ALI). Tilaajatiedon vastaava termi taas on Automatic Number Identification (ANI).

2.2.1 Tilaajatieto (ANI)

Perinteisesti maailmalla ANI tunnetaan teknologiana, joka näyttää puhelun vastaanottajalle soittajan numeron. ANI:n syntymän juuret ovat Yhdysvalloissa noin 1940-luvulla, ja se kehitettiin alun perin operaattoreiden kaukopuheluiden laskutusta varten. Kaukopuheluissa puhelua joudutaan kuljettamaan useamman keskuksen kautta, joten automaattisen soittajan tietojen selvityksen myötä pystyttiin laskuttamaan oikeaa asiakasta ilman ylimääräistä manuaalista selvitystyötä vaativia vaiheita. (Calling line identification circuit 1940.)

Hätäkeskustoiminnan näkökulmasta tarkasteltuna ei kuitenkaan olla kiinnostuneita teleoperaattoreiden käyttämästä teknologiasta soittajan tunnistukseen, vaan halutaan enemmänkin tietää, mitä kaikkea muuta tietoa soittajasta voidaan saada puhelinnumeron perusteella; kuten nimi ja osoite. Eli Response-ympäristössä termillä ANI tarkoitetaan puhelinnumeron omistavan henkilön tietojen hakua jostakin Responseen integroidusta palvelusta, käyttäen tässä työssä suunniteltua rajapintaa.

2.2.2 Paikkatieto (ALI)

Paikkatieto on Responsen näkökulmasta pelkkä koordinaattipiste, joka edustaa paikannetun puhelimen sijaintia. ALI-termin määritelmästä on kuitenkin eriäviä näkemyksiä esimerkiksi Pohjois-Amerikan Hätänumeroyhdistyksellä (NENA). NENA:n määrittelemä ALI-kysely sisältää käytännössä kentät myös tilaajatiedolle. (Standard ALI Query Best Practices 2011).

Yhdysvaltojen tapa käsitellä paikka- ja tilaajatietoa saman ALI-termin alla selittyy osittain edellisessä kappaleessa käsitellystä ANI:n tuomasta historian taakasta. Eli kun ANI rajataan pelkkänä numeron tunnistuksen menetelmänä, niin ALI:lla tarkoitetaan yleisesti kaikkea muuta numeroon ja sen omistajaan liittyvää tietoa. (Definition of Automatic Location Identification 2019).

Responsessa ANI ja ALI ovat kuitenkin rajattu selkeisiin vastuualueisiin. Paikkatiedolla todellakin tarkoitetaan puhdasta paikannustulosta. Mobiililaitteen paikannukseen taas on olemassa useita tekniikoita, mutta tässä työssä ei oteta kantaa paikannusmenetelmiin, vaan määritellään enemmänkin periaatteita useiden paikannuslähteiden tulosten välittämiseen rajapinnan yli.

2.3 Integraation ulkoistamisen tarve

Jotta Responsen ydinlogiikka saadaan täysin käyttöön eri maissa sijaitsevissa asiakasympäristöissä, tarvitaan toisistaan poikkeaville paikka- ja tilaajatiedon tuotantomenetelmille erilliset integraatiototeutukset, koska tässä työssä suunnitellun rajapinnan kaltaisen tiedon välitykselle ei ole olemassa yleisiä standardeja.

Asiakasympäristöjen integraatioiden tukena käytetään yhteistä integraatiokumppania, joka tuntee asiakkaan toimintatavat. Kaikkien mahdollisten integraatioiden toteuttaminen Response-kehitystiimin toimesta ei ole kuitenkaan ideaalista projektin skaalautuvuuden kannalta. Toteutustyön ulkoistaminen integraatiokumppanille mahdollistaisi tuotekehityksen resurssien ohjaamisen muualle, eikä aikaa kuluisi useiden asiakkaiden toimintatapoihin perehtymiseen, varsinkin kun olemassa on jo toteutukseen pätevä yhteistyökumppani.

Response-tuoteperheelle tarvitaan siis nykyistä laajempi ja konfiguroitavampi integraatioarkkitehtuuri tilaaja- ja paikkatiedoille, koska jokaisen kohdemaan tilaaja- ja paikkatietopalveluiden integraatiot tuovat laajennustarpeita esimerkiksi erilaisten osoitteistojen, vaihtelevien tietolähteiden, jne. myötä. Eri integraatioiden toteuttaminen olemassa olevaan rajapintaan olisi hankalaa, johtuen sidoksista Responsen tietomalliin, mikä hankaloittaa työn ulkoistamista esimerkiksi versiopäivitysten aiheuttamien mahdollisten epäyhteensopivuuksien myötä.

3 RAJAPINNAN SUUNNITTELU

Hätäkeskustoimintaan riittävien vaatimusten määrittäminen on ensisijaisen tärkeää, jotta rajapintoja voidaan käyttää mahdollisimman monipuolisesti. Rajapintojen sisällön suunnittelussa oli huomioitava kansainvälinen käyttöympäristö, sekä Responsen omat vaatimukset tilaaja- ja paikkatiedolle.

Tämän luvun alkuun käydään läpi molempia rajapintoja koskevat yleiset suunnitteluperiaatteet, minkä jälkeen esitellään paikka- ja tilaajatiedon rajapintojen suunnittelun vaiheet. Molemmista rajapinnoista esitellään suunnittelun aikana tutkittuja tietolähteitä, sekä koostetaan yhteenveto toteutettavasta rakenteesta.

3.1 Rajapinnan yleiset suunnitteluperiaatteet

Suunnittelun tavoitteena oli tutkia eri toimintamalleja ja tietorakenteita, joiden käytäntöjä soveltamalla voitiin koostaa mahdollisimman hyvin kaikkia tahoja palvelevia rajapintojen määritelmiä. Suunnittelun apuna käytettiin myös olemassa olevia standardeja esimerkiksi koordinaattipisteen, maatiedon ja ajan esitykseen.

Vaikka rajapintojen haluttaisiin palvelevan kaikkia mahdollisia asiakasintegraatioita, oli kuitenkin vältettävä tarpeetonta ylisuunnittelua, joka ei palvellut tuotteen etua nyt tai tulevaisuudessa. On parempi jättää asioita puuttumaan, kuin tehdä liikaa, koska uusien kenttien lisääminen on aina helpompaa kuin vanhojen poistaminen tuotantokäytössä olevasta rajapinnasta.

Myös virheenkäsittely oli huomioitava suunnittelussa. Paikka- ja tilaajatiedon kyselyn epäonnistuminen ei haittaa hätäpuheluiden käsittelyä, mutta tiedolla on silti tärkeä ajansäästöllinen osuus koko prosessissa. Siksi rajapinnan toimintaan liittyvien vikatilanteiden selvittäminen olisi tehtävä mahdollisimman suoraviivaiseksi. Vaikka rajapinnan toteutuksesta ja ylläpidosta huolehtii ulkopuolinen taho, niin Responsen näkökulmasta ollaan silti kiinnostuneita seuraamaan poikkeustilanteita, jotta voidaan suorittaa vaadittavat toimenpiteet.

3.2 Tilaajatieto (ANI)

Tilaajatiedon rakenne on parhaimmillaan hyvinkin yksinkertainen. Vähimmäisvaatimuksina tilaajatiedolle ovat puhelinnumeron omistavan henkilön nimi ja osoite, sekä mahdollisesti liittymän tyyppi; eli onko kyseessä matka- vai lankapuhelin.

Nykypäivänä lankapuhelimella voidaan puhekielessä tarkoittaa myös IP-verkon yli toimivaa kiinteää puhelinta. Vanhaa ja uutta teknologiaa yhdistää kuitenkin sidonnaisuus tiettyyn paikkaan, sekä tyypillisten matkapuhelinten paikannusmenetelmien toimimattomuus. Lankapuheluiden erottelu mobiililaitteista on siis hyödyllistä hätäkeskustoiminnan kannalta, koska esimerkiksi tieto hätäpuhelun saapuvan vanhoja kuparilinjoja pitkin, yhdistää soittajan välittömästi johonkin tiettyyn osoitteeseen.

Vähimmäisvaatimuksien lisäksi tilaajatietoon voitaisiin liittää paljon muutakin tietoa, mitä esimerkiksi Euroopan ja Pohjois-Amerikan Hätänumeroyhdistykset ovat ideoineet omiin dokumentteihinsa. Hyvien ideoiden lisäksi on kuitenkin muistettava arvioida niiden hyöty Response-tuoteperheelle.

3.2.1 EENA Mobile Identity

Euroopan Hätänumeroyhdistys EENA:n laatima Mobile Identity –dokumentti käsittelee nykypäivän ja tulevaisuuden visioita mobiililaitteiden tunnistautumisista hätäpuheluiden yhteydessä. Digitalisaation ja mobiililaitteiden yleistymisen myötä pystyttäisiin keräämään ja siirtämään entistä enemmän tietoa soittajasta hätäkeskuspäivystäjälle. Nykypäivän tilaajatiedon tilanne on nimittäin hieman haastava mobiililaitteiden syrjäytettyä kiinteät lankapuhelimet, joiden omistajien tiedot ovat aina saatavilla. (Mobile Identity 2018).

EENA:n ehdottama mobiili-identiteetti sisältäisi nimen ja osoitteen lisäksi tietoja esimerkiksi soittajan syntymäpäivästä, äidinkielestä ja oleellisista terveyteen liittyvistä asioista. Kaikki edellä mainitut tiedot olisivat ymmärrettävästi hyödyllisiä

hätäpuhelun käsittelyssä, mutta niitä varten tarvittaisiin olemassa oleva infrastruktuuri kohdemaan tilaajatiedon palveluntarjoajalta. EENA:n dokumentti sisältää vain ratkaisuehdotuksia mobiili-identiteetin toteutukseen, mutta ei yhtään viitettä tuotantokäytössä olevaan järjestelmään.

Koska tiedossa ei ole vielä yhtäkään EENA:n Mobile Identity:n tyyppistä tietoa tarjoavaa tahoa, on loogista jättää ylimääräiset kentät pois rajapinnan määrittelystä. Tuotekehityksen ja tulevaisuuden hätäkeskustoiminnan kannalta on kuitenkin tärkeää tiedostaa mahdollisesti tulevaisuudessa saatavilla olevan tiedon määrä ja sen merkitys.

3.2.2 Tilaajatiedon yhteenveto

Tilaajatiedon rajapinnan määrittelyssä päädyttiin lopulta mahdollisimman yksinkertaiseen ratkaisuun, vaikka mahdollisuuksia hyödylliselle tietosisällölle olisi paljonkin. Ongelmaksi muodostuu tilaajatiedon luonne hieman epäluotettavana tietolähteenä, johtuen esimerkiksi teknisen toteutuksen puutteesta, mobiililaitteiden liittymien tyypeistä, tai jopa teleoperaattoreiden suostumattomuudesta jakaa tietoja hätäkeskusten kanssa (Mobile Identity 2018, 8).

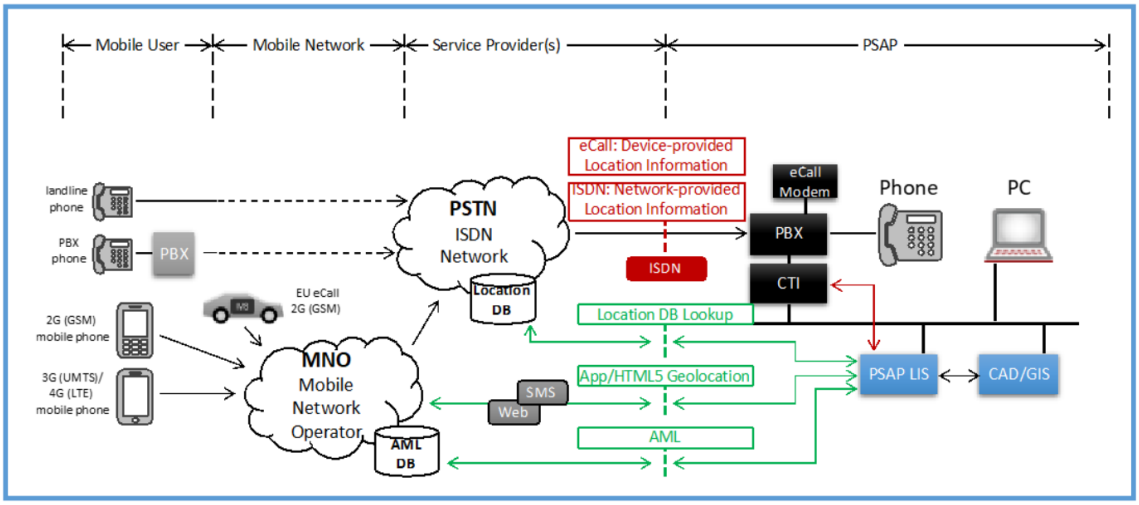
Taulukossa 1 on esitelty tilaajatiedon rajapinnan kentät.

TAULUKKO 1. Tilaajatiedon kentät

Kenttä	**Kuvaus**
Nimi	Tilaajan nimi
Osoite	Sisältää ainakin seuraavat tiedot: • Katu • Kaupunki, postinumero tai muu vastaava tarkentava tieto • Alue ISO 3166-2:n mukaan • Maa ISO 3166-1:n mukaan
Liittymän tyyppi	Matka- tai lankapuhelin

3.3 Paikkatieto (ALI)

Vaikka paikkatieto yksinkertaisimmillaan on vain koordinaattipiste, niin yhdenkin pisteen määrittelyyn voi sisältyä useita vaiheita. Useiden paikannusmenetelmien myötä voidaan saada hyvinkin erityyppisiä tuloksia, joiden mahdolliset erot oli huomioitava rajapinnan suunnittelussa. Kuvassa 2 on EENA:n hahmotelma nykypäivän paikannusmenetelmistä, joita voidaan esimerkiksi hallinnoida erillisen paikkatietopalvelun (Location Information Service) välityksellä. Vastaavanlainen paikkatietopalvelun tuki toteutettiin myös tässä työssä.



KUVA 2. Paikkatietopalvelu (LIS) paikannuksen apuna (The Role of GIS in NG112 2018, 21)

Koska paikannuslähteitä voi olla useita, haluttiin mahdollistaa useiden tulosten siirtäminen rajapinnan yli listana. Tällöin integraatioiden toteuttamisen yhteydessä voidaan tehdä valinta siitä, että halutaanko kaikki paikkatietolähteet yhdistää yhden rajapinnan toteutuksen taakse listarakenteeseen, vai paljastetaanko jokainen Responselle asti omana rajapinnan toteutuksenaan.

Myös epäonnistuneet paikannustulokset haluttiin välitettävän rajapinnan vastauksissa, sillä sekin itsessään on merkityksellistä tietoa. Tietoa virhetilanteista voidaan hyödyntää vianselvityksessä integraatiokumppanin kanssa, sekä tarvittaessa paikannuskyselyitä voidaan ohjata muualle vikaantuneen lähteen sijaan.

Vähimmäisvaatimuksena yhdelle paikannustulokselle on koordinaattipiste, aikaleima ja tarkkuus. Lisäksi haluttaisiin myös lisätietoa käytetystä paikannusmenetelmästä, jota voidaan hyödyntää esimerkiksi järjestelmänvalvonnassa tai tulosten yksilöinnissä.

3.3.1 EENA Advanced Mobile Location (AML)

Advanced Mobile Location (AML) on Euroopan Hätänumeroyhdistyksen hallinnoima, hätäpuhelun yhteydessä automaattisen paikkatiedon välittämiseen tarkoitettu spesifikaatio. AML:n määrittelee laitevalmistajien ja hätäpuheluita vastaanottavien tahojen välille yhteiset standardit tiedonsiirtoon, mahdollisimman yksinkertaisella ja kustannustehokkaalla tavalla. (AML Specifications & Requirements 2016).

Toteutettu AML on laitetason ohjelmisto, joka aktivoituu automaattisesti hätäpuhelun yhteydessä. Ohjelmisto suorittaa laitteen paikannuksen käyttäen saatavilla olevia paikannusmenetelmiä, ja välittää tiedon SMS-viestillä eteenpäin käyttäjän huomaamatta mitään. Vaikka vastuu toteutuksesta on lähtökohtaisesti laitevalmistajalla, Google ja Apple ovat myös tehneet omat käyttöjärjestelmätason toteutukset AML-spesifikaation pohjalta. (AML Specifications & Requirements 2016).

AML ei liity suoraan Responseen, mutta rajapinnan toteuttava paikkatietopalvelu joutuu mahdollisesti käsittelemään myös AML-viestejä. Tästä syystä AML-spesifikaatio toimii hyvänä viitteenä rajapinnan suunnittelussa. Ainakin WGS84-standardin mukainen koordinaattipisteen esitys ja tarkkuuden kuvaaminen alueen säteenä metreissä ovat kiinnostavia tietoja.

3.3.2 OMA SpecWorks Mobile Location Protocol

OMA SpecWorks on standardien kehitysorganisaatio, jonka laatima Mobile Location Protocol (MLP) määrittelee sovellustason protokollan mobiililaitteiden paikannukseen (Mobile Location Protocol 2018). MLP-määrittelydokumentti sisältää

noin 150 erilaista paikkatiedon kuvaamiseen liittyvää elementtiä. Dokumentista nostettiin muutama kiinnostava tieto osaksi rajapinnan suunnitelmaa.

MLP määrittelee useita mahdollisesti tulevaisuudessa kiinnostavia tietoja; kuten korkeuteen, nopeuteen ja suuntaan liittyviä asioita. Näille kentille ei kuitenkaan ole nykyhetkellä mitään selvää käyttötarkoitusta, joten ne jätettiin suunnitelman ulkopuolelle.

Paikkatiedon rajapinnan suunnitelmaan MLP-dokumentista valittiin ainoastaan muutama paikkatietopalvelinkohtainen tuloskoodi. Koodit koettiin tarpeellisiksi useita paikannustuloksia mahdollisesti sisältävän rajapinnan listarakenteen johdosta. Valittuja koodeja esitellään tarkemmin seuraavassa virheenkäsittelyluvussa.

3.3.3 Paikkatiedon virhetilanteiden käsittely

Yksittäisen paikannustuloksen onnistumisen kuvaamiseen käytettiin enumeraatioita, jotka edustivat ennalta määriteltyjä mahdollisia poikkeustilanteita. Listan suunnittelun viitteenä käytettiin Open Mobile Alliancen Mobile Location Protocol -spesifikaation kuvaamia tilakoodeja. Taulukossa 2 on esitelty valitut tuloskoodit.

TAULUKKO 2. Rajapinnan vastauksen tuloskoodit (Mobile Location Protocol 2018, 141)

Teksti	**Kuvaus**
OK	Kaikki Ok
SYSTEM_FAILURE	Palvelinpään virhe
UNSPECIFIED_ERROR	Määrittelemätön virhe
UNAUTHORIZED_APPLICATION	Autentikointivirhe
UNKNOWN_SUBSCRIBER	Puhelinnumeroa ei löydy
ABSENT_SUBSCRIBER	Puhelinnumero löytyy, mutta ei ole juuri nyt paikannettavissa
POSITION_METHOD_FAILURE	Paikannusmenetelmän virhe

TIMEOUT	Paikannuksen aikakatkaisu

3.3.4 Paikkatiedon yhteenveto

Paikkatiedon esittämiseen ja tuloksen tarkkuuteen voitaisiin käyttää todella monimutkaisiakin menetelmiä erilaisten geometristen kuvioiden muodossa. Ainoaksi geometriaksi rajapinnan suunnittelussa valittiin yksinkertainen ympyrä; eli keskipistettä kuvaavat koordinaatit, sekä sädettä kuvaava tarkkuus. Lisäksi rajapintaa laajennettiin muutamalla lisätiedolla liittyen paikannuslähteeseen ja paikannuksen onnistumisen kuvailuun. Myös tuloksen ajantasaisuuden kuvaamisen tueksi otettiin mukaan kenttä aikaleimalle.

Aikaleiman esitysformaatiksi valittiin ISO 8601 –standardin osoittama esitystapa, sen yleisesti laajan käytön vuoksi. Tässä työssä tutkituissa laitteistoläheisemmissä dokumenteissa käytettiin ISO 8601 –standardia tiiviimpää ja vaikeammin luettavaa muotoa; eli ”vvvvkkppttmmss”, mutta kyseinen kompakti formaatti on tarpeeton tämän työn rajapinnan kontekstissa.

Taulukossa 3 on esitelty paikkatiedon rajapinnan kentät. Taulukon kentät kuvaavat yhtä paikannustulosta, joita palautetaan listana rajapinnan yli. Rajapinnan toteuttavalle taholle annetaan siis mahdollisuus tarjota Responseen tietoa kaikista yritetyistä paikannuslähteistä.

TAULUKKO 3. Paikkatiedon kentät

Kenttä	**Kuvaus**
Aikaleima	Kuvaa paikannustuloksen tuoreutta ISO 8601 –formaatissa millisekuntien tarkkuudella ja Zulu-aikavyöhykkeellä (UTC+0). Esim. ” 2019-08-17T06:15:04.321Z”

Korkeusaste (latitude)	Koordinaattipisteen (WGS84) korkeusasteen esitys desimaalilukuna. Merkintä viidellä desimaalilla vastaa 1,1 metrin tarkkuutta
Leveysaste (longitude)	Koordinaattipisteen (WGS84) leveysasteen esitys desimaalilukuna.
Tarkkuus	Paikannustuloksen tarkkuus esitettynä ympyrän säteenä metreissä
Paikannuslähteen kuvaus	Tekstikenttä paikannuslähteen kuvaamiselle
Paikannustuloksen lisätieto	Sisältää kaksi kenttää: • Taulukon 2 kuvaama tuloskoodi • Vapaa tekstikenttä virhetilanteen lisätiedolle

4 RAJAPINNAN TOTEUTUS

Rajapintojen suunnitelmien realisointi käyttökelpoiseksi toteutukseksi koostui kahdesta vaiheesta, joita tässä luvussa käsitellään tarkemmin. Ensimmäiseksi oli luotava rajapintojen tarkat määritelmät käyttäen jotain kuvausformaattia. Määritelmien perusteella pystyttiin taas generoimaan valmiit toteutukset rajapinnan käyttöä varten.

4.1 Teknologiat

Suunnitelmien mukaiset tilaaja- ja paikkatiedon rajapinnat määriteltiin käyttäen valittuja kuvauskieliä. Määrittelytiedostoista pystyttiin generoimaan Java-kielen client-toteutukset Maven-laajennusten avulla. Määrittelytiedostot myös pakattiin omalla Maven-laajennuksellaan toimituskelpoiseen muotoon.

Valitut rajapintojen kuvaamiseen tarkoitetut tai soveltuvat kielet olivat OpenAPI Specification (OAS) ja Protocol buffers (protobuf). Molemmista tilaaja- ja paikkatiedon rajapinnoista luotiin määritelmät sekä OAS:illa kuvailtuna että protobuffilla. Lopputuloksen työssä siis syntyi yhteensä neljä erilaista rajapinnan määritelmää.

Käyttämällä molempia teknologioita kaksinkertaistettiin ylläpidettävien määrittelytiedostojen määrä. Saman rajapinnan ylläpitäminen kahdella eri tekniikalla, pitäen molempien määrittelyt mahdollisimman samankaltaisina, voi aiheuttaa hieman ylimääräistä työtä johtuen kielien välisistä eroista. Esimerkiksi protobuf ei enää uusimmassa versiossa tue kenttien määrittelyä pakollisiksi, mikä taas eroaa OAS:in käytännöistä.

Tukemalla molempia teknologioita mahdollistetaan rajapinnan toteutus perinteisesti REST:illä, tai vaihtoehtoisesti modernimmalla stream-pohjaisella gRPC:llä. Streamien myötä gRPC mahdollistaisi monipuolisempia käyttötapoja rajapinnalle, esimerkiksi jatkuvasti päivittyvän paikkatiedon myötä. gRPC:n toiminta HTTP/2:n yli saattaa kuitenkin esimerkiksi olla yhteensopimatonta vanhemman sukupolven suljettujen verkkojen palomuurien kanssa, joten perinteiselle

HTTP/1.1:n yli toimivalle REST:ille on todennäköisesti tarvetta myös tulevaisuudessakin.

4.1.1 OpenAPI ja REST

OpenAPI Specification (OAS) on REST-rajapintojen kuvausformaatti. OAS:in aikaisempi kutsumanimi oli Swagger Specification, kunnes uusin 3.0 versio julkaistiin. OAS-rajapintojen kehittämiseen on olemassa avoimen lähdekoodin Swagger-työkaluja; kuten selainpohjainen editori, interaktiivista dokumentaatiota ja koodigeneraattoreita. (What Is OpenAPI? 2019.)

OpenAPI-kuvauksia voidaan kirjoittaa joko YAML tai JSON-muodossa. OAS mahdollistaa rajapinnan kuvaamisen lisäksi määrittelyn erilaisille metatiedoille ja autentikaatiometodeille. Kuvassa 3 on kuvauksen perusrakenne YAML:illa kirjoitettuna. (Basic Stucture 2019.)

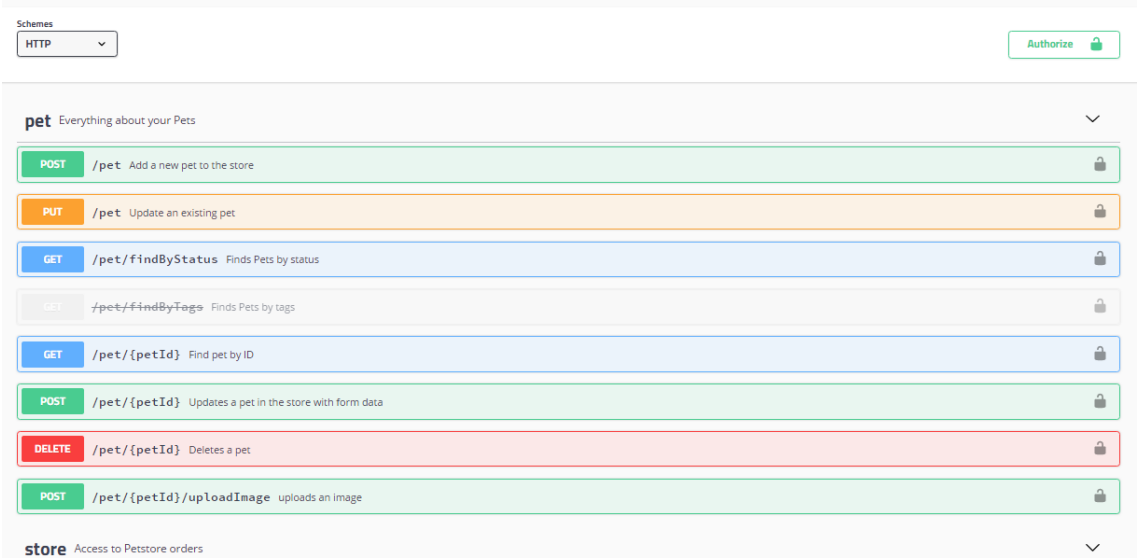
```yaml
openapi: 3.0.0
info:
  title: Sample API
  description: Optional multiline or single-line description in [CommonMark](http://co
  version: 0.1.9

servers:
  - url: http://api.example.com/v1
    description: Optional server description, e.g. Main (production) server
  - url: http://staging-api.example.com
    description: Optional server description, e.g. Internal staging server for testing

paths:
  /users:
    get:
      summary: Returns a list of users.
      description: Optional extended description in CommonMark or HTML.
      responses:
        '200':    # status code
          description: A JSON array of user names
          content:
            application/json:
              schema:
                type: array
                items:
                  type: string
```

KUVA 3. OpenAPI-kuvauksen perusrakenne (Basic Structure 2019)

OAS:in mukaisista YAML- tai JSON-tiedostoista voidaan generoida interaktiivista dokumentaatiota sekä palvelin- että client-pään REST-toteutuksia, mikä tekee OpenAPI:n kaltaisten abstraktien kuvausten käyttämisestä kannattavaa. Kuvassa 4 on esitelty Swagger UI:lla luotu interaktiivinen web-selaimessa toimiva dokumentaatiosivu.



KUVA 4. Generoitu Swagger dokumentaatio (Swagger UI 2019)

4.1.2 Protocol buffers ja gRPC

Protocol bufferit ovat XML-kielen tavoin tapa kuvata serialisoitavia tietorakenteita, mutta Googlen mukaan yksinkertaisempia ja huomattavasti nopeampia XML:ään verrattuna (Protocol Buffers Developer Guide 2018). Kuvassa 5 on esitelty tietorakenteiden kuvaamista Protocol buffereiden avulla. Kuvassa käytettyjä required ja optional-kenttiä ei enää tueta Protocol buffereiden uusimmassa 3.0 versiossa.

```
message Person {
  required string name = 1;
  required int32 id = 2;
  optional string email = 3;

  enum PhoneType {
    MOBILE = 0;
    HOME = 1;
    WORK = 2;
  }

  message PhoneNumber {
    required string number = 1;
    optional PhoneType type = 2 [default = HOME];
  }

  repeated PhoneNumber phone = 4;
}
```

KUVA 5. Protocol bufferin perusrakenne (Protocol Buffers Developer Guide 2018)

Protocol buffers itsessään ei ole mikään rajapintojen kuvauskieli OpenAPI:n lailla. Protocol buffereita on tarkoitus käyttää halutun tietorakenteen kuvaamiseen, jonka jälkeen määritelmästä voidaan generoida monipuolisesti toteutuksia esimerkiksi datan siirtoon ja tallennukseen (Protocol Buffers Developer Guide 2018). Protocol buffereita voidaan kuitenkin käyttää OpenAPI-tyyppisesti kuvaamaan alunperin Googlen kehittämän gRPC:n rajapintoja.

gRPC:tä voisi alhaisella tasolla kuvailla protokollana API-kutsujen välittämiseen HTTP/2:n yli (Introduction to gRPC 2017). Kyseessä on kuitenkin suurempi avoimeen lähdekoodiin perustuva ekosysteemi, johon kuuluu tehokkaita työkaluja sovellusten ja palveluiden välisen kommunikaation kehittämiseen. Esimerkiksi gRPC:n tuki kaksisuuntaiseen stream-pohjaiseen viestintään mahdollistaa REST-rajapintoja monipuolisemman käytön.

gRPC käyttää oletuksena Protocol buffereita rajapinnan kuvaamiseen kuvan 6 osoittamalla tavalla, sekä rajapinnassa liikkuvien tietorakenteiden määrittämiseen esimerkiksi aikaisemman kuvan 5 esittämässä muodossa. gRPC tukee

myös muita formaatteja, kuten JSON:ia serilalisoitavien rakenteiden kuvaamiseen. OpenAPI:n lailla myös Protocol buffereilla määritellystä gRPC rajapinnasta voidaan generoida sovellus- ja palvelinpään toteutuksia useille eri kielille. (What is gRPC? 2019.)

```
// The greeter service definition.
service Greeter {
  // Sends a greeting
  rpc SayHello (HelloRequest) returns (HelloReply) {}
}
```

KUVA 6. Rajapinnan määrittely Protocol buffereilla (What is gRPC? 2019)

4.1.3 Reactor-gRPC

Työssä generoitujen REST- ja gRPC-clienttien käyttö käärittiin Project Reactorin luokkien sisään, jotta rajapintoja käyttävälle Response-laajennukselle ei turhaan luotaisi suoria riippuvuuksia generoituihin toteutuksiin. Project Reactor on Java 8 implementaatio Reactive Streams -spesifikaatiosta, joka taas on standardi asynkronisten streamien prosessointiin (Intro to Reactor Core 2019).

Reactor tarjoaa kaksi tietotyyppiä Mono ja Flux, joiden kautta voidaan välittää esimerkiksi tässä työssä toteutettujen rajapintojen vastauksia asynkronisesti Responseen. Reactorin tarjoamat luokat mahdollistavat Javan stream-tyyppisten funktionaalisten operaatioiden käytön, sekä esimerkiksi useiden Monojen yhdistämisen yhdeksi kuunneltavaksi Fluxiksi.

4.1.4 Java ja Maven

Rajapintojen client-toteutukset generoitiin Java-kielelle Maven-laajennusten avulla. Apache Maven on pääosin Java-projekteille tarkoitettu konfiguraatioiden hallintatyökalu. Esimerkiksi projektin riippuvuudet voidaan määritellä XML-muo-

dossa pom.xml-nimiseen tiedostoon, jota lukemalla Maven pystyy automaattisesti käännösvaiheessa lataamaan riippuvuudet julkisesta Maven Repositorystä tai jostain erikseen konfiguroidusta osoitteesta (What is Maven? 2019).

Riippuvuuksien versioiden ja projektin konfiguraatioiden hallinnan lisäksi Mavenin rakennusprosessin eri vaiheisiin voi liittää suoritettavaksi erillisiä laajennuksia. Kuvassa 7 on koodiesimerkki laajennuksen liittämisestä osaksi projektin rakennusprosessia. Tämän työn rajapintojen toteutuksien generointeihin käytettiin vastaavanlaisia laajennuksia.

```
<project>
  ...
  <build>
    <plugins>
      <plugin>
        <artifactId>maven-myquery-plugin</artifactId>
        <version>1.0</version>
        <configuration>
          <url>http://www.foobar.com/query</url>
          <timeout>10</timeout>
          <options>
            <option>one</option>
            <option>two</option>
            <option>three</option>
          </options>
        </configuration>
      </plugin>
    </plugins>
  </build>
  ...
</project>
```

KUVA 7. Esimerkki Maven-laajennuksesta (Guide to Configuring Plug-ins 2019)

4.2 Rajapintojen määrittelyt

Seuraavissa aliluvuissa on esitelty avainkohtia työssä määritellyistä rajapinnoista OpenAPI:lla ja Protocol buffereilla kuvattuina. Molemmista paikka- ja tilaajatiedon

rajapinnoista tehtiin määritelmät OpeanAPI:lla ja protobuffilla, mutta toiston vähentämiseksi tässä luvussa esitellään vain tilaajatiedon OpenAPI-määrittely, sekä paikkatiedon rakennetta näytetään vain protobuf-muodossa. Kaikki työssä toteutetut määritelmät ovat kuitenkin kokonaisuudessaan luettavissa liitteistä 1-4.

4.2.1 Tilaajatieto, OpenAPI ja REST

Tilaajatiedon REST-rajapinta sisältää yhden POST-metodin (kuva 8), jonka kautta voidaan hakea tilaajatietoja tarjotulle puhelinnumerolle. Luonteeltaan identifyNumber-metodi on enemmänkin GET-tyylinen, mutta GET-kutsulla tunnistettava puhelinnumero jouduttaisiin liittää parametrina osaksi kyselyä, mikä taas paljastaisi http-käyttölokiin GDPR:n mukaisia henkilökohtaisia tietoja.

```
paths:
  /ani/identifyNumber:
    post:
      operationId: getNumberIdentity
      summary: Identify number
      description: Method for requesting identity information for the given number.
      tags:
        - ANI
      requestBody:
        description: A request containing parsed phone number with country code
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/NumberIdentityRequest'
      responses:
        '200':
          description: Number identified succesfully
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/IdentityResponse'
```

KUVA 8. Tilaajatiedon identifyNumber-metodi

API-kutsun palvelupyynnön runko (requestBody) sisältää yksinkertaisen kuvassa 9 esitetyn NumberIdentityRequest:in, joka sisältää tunnistettavan puhelinnumeron. Onnistuneen vastauksen mukana taas palautetaan kuvan 10 tyyppinen IdentityResponse.

```yaml
NumberIdentityRequest:
  type: object
  properties:
    phoneNumber:
      type: string
      description: >
        A string representation of the phone number to locate.
        Containing a country code and NO whitespace.
      example: '+4981234567890'
```

KUVA 9. Tilaajatiedon kyselyn palvelupyynnön runko

Kuvan 10 IdentityResponse sisältää juurikin suunitelman mukaisen rakenteen. Puhelintyypin kuvaamiseen käytettiin luonnollisesti enumeraatiota, mikä helpottaa generoituvan toteutuksen käsittelyä.

```yaml
IdentityResponse:
  type: object
  properties:
    name:
      type: string
      description: Full name of the subscription owner.
      example: John Smith
    address:
      description: Billing or installation address.
      $ref: '#/components/schemas/Address'
    phoneType:
      type: string
      description: >
        Describes the type of the phone.

        MOBILE means a device's location may be seperate from the billing or installation address.
        A mobile device in most cases has to be located by other means (e.g. ALI).

        A FIXED phone is tied to its installation address, e.g. landlines.
        No seperate locationing should be required for these types of devices,
        nor may not even be possible.
      enum: [MOBILE, FIXED]
```

KUVA 10. Tilaajatiedon vastaus

Kuvassa 11 on esitelty IdentityResponsen sisältämä Address-skeema osoitetiedoille. Osoitteen kuvaamiseen käytetään suunnitelman mukaisesti ISO 3166 – standardin maa- ja aluekoodeja.

```
Address:
  type: object
  properties:
    country:
      type: string
      description: ISO 3166-1 alpha-2 country code.
      example: US
    region:
      type: string
      description: Region should be primarily interpreted as ISO 3166-2
        region code and secondarily as a regular text.
      example: US-LA
    locality:
      type: string
      description: >
        Geographic area that makes the interpretation of street address
        information combined with country and region unambigiuos.
        City, municipality or similar.
      example: New Orleans
    streetAddress:
      type: string
      description: Street address.
      example: 701 Bourbon St
```

KUVA 11. Osoitetiedon malli

4.2.2 Paikkatieto, protobuf ja gRPC

Paikkatiedon gRPC-rajapinta koostuu yhdestä LocateNumber-metodista (kuva 12). Rajapintaan ei toistaiseksi vielä lisätty stream-toiminnallisuutta hyödyntäviä metodeja esimerkiksi päivittyvää paikkatietoa varten. Vastaavanlaiset laajennukset rajapintaan voidaan tarvittaessa toteuttaa tulevaisuudessa.

```
/**
 * This is an API for making ALI-requests from Insta Response.
 */
service ALI {
  /**
   * Returns a list of location results for the given number.
   */
  rpc LocateNumber (NumberLocationRequest) returns (LocationResponse);
}
```

KUVA 12. Paikkatiedon gRPC-rajapinta

Kuvan 13 LocationResponse sisältää suunnitelman mukaisen listan paikannustuloksia. Repeated-avainsana kuvaa tyypin toistuvan 0-n kertaa yhdessä LocationResponsen instanssissa. Toistuvista repeated-kentistä generoinnin tuloksena syntyy taulukkomaisia tyyppejä.

Lisätiedoksi vastaukseen lisättiin parentSource-kenttä, yksittäisten API-kutsujen kohteiden yksilöimiseen ensisijaisesti lokitusta varten. ParentSource-kentässä ohjeistettiin kuitenkin käyttämään selkeää koodimaista rakennetta, koska vapaan tekstikentän sisällöstä ei oletuksena pitäisi voida päätellä mitään, eikä sitä voida näyttää sellaisenaankaan käyttöliittymällä. Asettamalla kevyitä rajoituksia sisällön ulkoasulle ei menetetä luettavuutta, mutta helpotetaan käsittelyä koodissa, jotta sisältöä voidaan tarvittaessa käyttää esimerkiksi lokalisointiavaimena tai jonkin käyttöliittymäikonin tunnisteena.

```
message LocationResponse {
  /** A list of location results. */
  repeated Location results = 1;
  /**
   * Represents an single endpoint called from Response.
   * Enumeration-like format with UPPERCASE letters and underscores should be used.
   * E.g. "NATIONAL_ALI_INTEGRATION"
   */
  string parentSource = 2;
}
```

KUVA 13. gRPC-rajapinnan vastaus

Yksittäistä paikannustulosta kuvaava Location on määritelty kuvassa 14. Location:in source-kentässä noudatetaan samaa periaatetta kuin kuvan 13 LocationResponsen parentSource:ssa, muuten tietosisältö on täysin suunnitelman mukaista.

```
message Location {
  /**
   * ISO 8601 date-time in millisecond precision.
   * Date-time must be given in Zulu-timezone,
   * and zone identifier 'Z' must be included in the value.
   * Defines how recent this location information is. E.g. "2019-08-17T06:15:04.321Z"
   */
  string timestamp = 1;
  /** Decimal representation of latitude, based on WGS84 standard. E.g. "29.95876" */
  double latitude = 2;
  /** Decimal representation of longitude, based on WGS84 standard. E.g. "-90.06557" */
  double longitude = 3;
  /** Describes the accuracy of the location result as a circle's radius in meters. E.g. "2" */
  int32 radius = 4;
  /**
   * Describes the source for this location result.
   * Enumeration-like format with UPPERCASE letters and underscores should be used.
   * E.g. "NETWORK_TRIANGULATION"
   */
  string source = 5;
  /** Holds information about the success of this single location attempt. */
  ResultInfo resultInfo = 6;
  }
```

KUVA 14. Yksittäisen paikannustuloksen rakenne

Kuvan 15 ResultInfo:n rooli on enemmänkin olla tukena virhetilanteiden selvittämisessä, minkä takia errorMessage-kenttä ei saa missään tapauksessa sisältää henkilökohtaisia tietoja. ResultCode-enumeraatio taas koostuu kokonaisuudessaan taulukon 2 vastauksen tuloskoodeista. Liite 4 sisältää ResultCode-enumeraation täydellisen toteutuksen.

```
message ResultInfo {
  /**
   * A string field for additional information about the failed location result
   * for debugging purposes. MUST NOT contain any personal identifying information!
   */
  string errorMessage = 1;
  ResultCode resultCode = 2;

  /** OMA SpecWorks Mobile Location Protocol -inspired result codes */
  enum ResultCode {
    /** No error occurred while processing the request. */
    OK = 0;
    ...
```

KUVA 15. Paikannustuloksen lisätieto onnistumisesta

4.3 Toteutuksien generointi

Koodin generoinnissa on kyse toteutuksien tai toteutusrunkojen luomisesta määrittelytiedoston pohjalta omalla kääntäjällään. Generoinnin käyttämistä voidaan perustella neljällä periaatteella: tuottavuus, yksinkertaisuus, siirrettävyys ja yhdenmukaisuus. (Gabriele Tomassetti 2018.)

Kehityksessä säästetään ylimääräisiä työvaiheita, kun yhdellä generaattoritoteutuksella voidaan generoida aina uusi versio rajapinnasta. Yksinkertaisuutta ja siirrettävyyttä saadaan taas siitä, kun generoitava toteutus on määritelty abstraktina kuvauksena. Pelkkä kuvaus on täysin alustariippumaton, joten generoinnin voi toteuttaa haluamalleen kielelle. Generoitu koodi on taas yhdenmukaista, koska ihmisen näppäilyvirheiltä vältytään, ja generointi seuraa samaa kaavaa jokaisella ajokerralla.

Generoinnin myötä luodaan kuitenkin haasteita ylläpidon ja monimutkaisuuden kannalta. Käytettäessä generaattoria tullaan riippuvaiseksi sen luomista toteutuksista, joten generaattorin ylläpidosta on pidettävä huolta. Samalla myös generoitu koodi on huomattavasti monimutkaisempaa luettavaa, verrattuna käsin kirjoitettuun.

4.3.1 OpenAPI

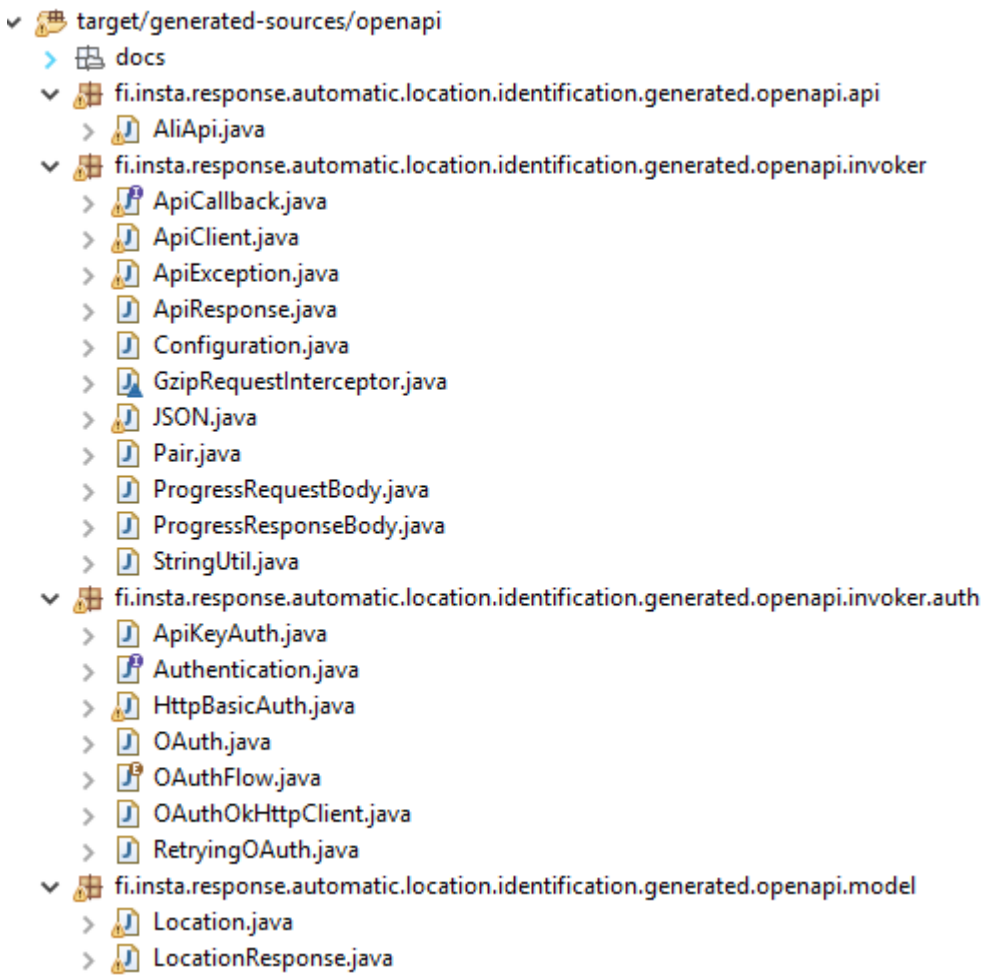
REST-API:n toteutuksien generointiin käytettiin openapi-generator-maven-plugin –Maven-laajennusta. Laajennus sisältää useita generaattoritoteutuksia client- ja server-toteutuksien luontia varten. Esimerkiksi Java-clientin generointiin käytetään generatorName-parametrin arvoa ”java”. Springiä käyttävä Java-server taas voidaan luoda arvolla ”spring”. (OpenAPITools 2019.)

Kuvassa 16 on esitelty minimikonfiguraatio laajennuksen ajamista varten. Työssä toteutettu konfiguraatio paikkatiedon client-toteutuksen generointia varten on taas esitelty liitteessä 5. Generoitujen toteutusten vaatimia riippuvuusmäärittelyitä ei ole esitelty kuvassa 16 ja liitteessä 5.

```
<plugin>
    <groupId>org.openapitools</groupId>
    <artifactId>openapi-generator-maven-plugin</artifactId>
    <version>3.3.4</version>
    <executions>
        <execution>
            <goals>
                <goal>generate</goal>
            </goals>
            <configuration>
                <inputSpec>${project.basedir}/src/main/resources/api.yaml</inputSpec>
                <generatorName>java</generatorName>
                <configOptions>
                   <sourceFolder>src/gen/java/main</sourceFolder>
                </configOptions>
            </configuration>
        </execution>
    </executions>
</plugin>
```

KUVA 16. Esimerkki Java-clientin generoimisesta (OpenAPITools 2019)

Generoinnin tuloksena projektin target-kansioon luodaan useita luokkia rajapinnan käyttöä varten. Kuvan 17 kuvakaappauksessa on esimerkiksi listaus paikkatiedon rajapinnan generoinnin tuloksista. Aikaisemmin mainittu generoidun koodin huono luettavuus on huomattavissa jo luotujen tiedostojen määrästä. Rajapinnan yksinkertaiseen käyttöön tarvittaisiin esimerkiksi vain AliApi- ja ApiClient-luokat. Myös model-paketin alle luodut Location- ja LocationResponse-luokat ovat tärkeitä rajapinnassa siirrettävän tietosisällön määrittämiseen.



KUVA 17. Paikkatiedon rajapinnan generoitu client-toteutus

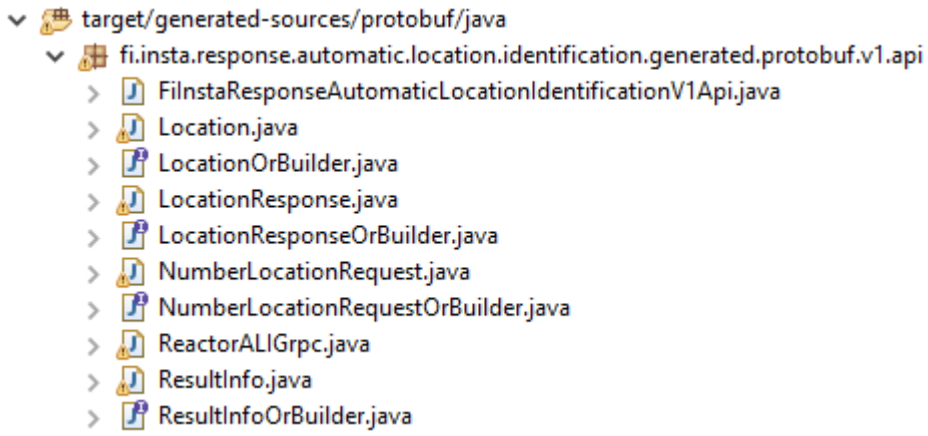
4.3.2 gRPC

gRPC-rajapintojen generointiin käytettiin protobuf-maven-plugin –Maven-laajennusta. Laajennuksen tarkoituksena on integroida Protocol buffereiden kääntäjä (protoc) osaksi Maven-projektin elinkaarta (Xolstice 2019). Laajennus tuottaa automaattisesti jokaisesta löytämästään proto-tiedostosta oman Java-toteutuksensa. Generoinnin tuloksena syntyy client- ja server-päiden luokat kerralla. Kuvassa 18 on yksinkertainen esimerkki laajennuksen suorituksesta. Itse työssä toteutettu laajennuksen konfiguraatio on nähtävissä liittessä 6.

```
<plugin>
    <groupId>org.xolstice.maven.plugins</groupId>
    <artifactId>protobuf-maven-plugin</artifactId>
    <version>0.6.1</version>
    <configuration>
    <protocExecutable>/usr/local/bin/protoc</protocExecutable>
    </configuration>
    <executions>
    <execution>
        <goals>
        <goal>compile</goal>
        <goal>test-compile</goal>
        </goals>
    </execution>
    </executions>
</plugin>
```

KUVA 18. Esimerkki gRPC-rajapinnan generoinnista (Xolstice 2019)

Laajennuksen suorittaminen on suoraviivaista, sekä generoituvien luokkien käsittely on huomattavasti selkeämpää OpenAPI:iin verrattuna. Kuvassa 19 on nähtävillä paikkatiedon rajapinnan generoidut luokat ja rajapinnat.



KUVA 19. Generoidut gRPC-luokat

4.3.3 Reactor-gRPC

Kuvassa 19 näkyvä ReactorALIGrpc-tiedosto on saanut alkunsa reactor-grpc –laajennuksesta, joka liitettiin osaksi aikaisempaa gRPC-rajapinnan generointilaajennusta. Laajennus liitettiin osaksi protobuf-mave-pluginin ajoa kuvan 20 osoittamalla tavalla. Tarkemman kuvan laajennuksen liitoskohdasta saa liitteestä 6.

```
<!-- For Reactor gRPC -->
<configuration>
    <protocPlugins>
        <protocPlugin>
            <id>reactor-grpc</id>
            <groupId>com.salesforce.servicelibs</groupId>
            <artifactId>reactor-grpc</artifactId>
            <version>${reactive.grpc.version}</version>
            <mainClass>com.salesforce.reactorgrpc.ReactorGrpcGenerator</mainClass>
        </protocPlugin>
    </protocPlugins>
</configuration>
```

KUVA 20. reactor-grpc –laajennuksen ajo osana protobuf-maven-pluginia

Reactor-gRPC protoc-laajennus luo Reactor tyyppiset gRPC-sidokset, jotta rajapintaa voidaan käyttää suoraan Reactor-kirjaston Mono- ja Flux-luokkien kautta. Eli laajennus muuttaa esimerkiksi yksinkertaisen rajapinnan metodin rakennetta kuvan 21 osoittamalla tavalla.

```
// Default configuration
public void sayHello(HelloRequest request, StreamObserver<HelloResponse> responseObserver) {
    ...
}

// With reactor-grpc plugin
public Mono<HelloResponse> sayHello(Mono<HelloRequest> request) {
    ...
}
```

KUVA 21. Esimerkki reactor-grpc –laajennuksen vaikutuksesta

4.4 Määrittelytiedostojen paketointi

Mavenin paketointivaiheeseen lisättiin laajennus, joka pakkaa kaikki OpenAPI ja protobuf määrittelytiedostot kahteen zip-pakettiin integraatiokumppanille toimitusta varten. Kuvassa 22 näytetyn laajennuksen suorituksen tuloksena syntyy kaksi pakettia; ani.zip ja ali.zip.

```
<plugin>
<!-- Release service protos as zip artifact-->
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-assembly-plugin</artifactId>
    <configuration>
        <appendAssemblyId>true</appendAssemblyId>
    </configuration>
    <executions>
        <execution>
            <id>create-ani-zip</id>
            <phase>package</phase>
            <goals>
                <goal>single</goal>
            </goals>
            <configuration>
                <descriptors>
                    <descriptor>src/main/assembly/ani-zip.xml</descriptor>
                </descriptors>
            </configuration>
        </execution>
        <execution>
            <id>create-ali-zip</id>
            ...
        </execution>
    </executions>
</plugin>
```

KUVA 22. Maven-assembly-pluginin käyttö tiedostojen paketoinnissa

Laajennus tarvitsee kuitenkin tarkemman toimintaohjeen paketointia varten, mikä on määritetty descriptor-avainsanalla. Kuvassa 23 on esitelty paketoinnin konfiguraatiotiedosto paikkatiedon rajapinnalle.

```
<assembly xmlns="http://maven.apache.org/plugins/maven-assembly-plugin/assembly/1.1.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/plugins/maven-assembly-plugin/assembly/1.1.0
        http://maven.apache.org/xsd/assembly-1.1.0.xsd">
    <id>ali</id>
    <includeBaseDirectory>false</includeBaseDirectory>
    <formats>
        <format>zip</format>
    </formats>
    <fileSets>
        <fileSet>
            <directory>src/main/ali</directory>
            <outputDirectory>/</outputDirectory>
            <includes>
                <include>**</include>
            </includes>
        </fileSet>
    </fileSets>
</assembly>
```

KUVA 23. Paikkatiedon rajapinnan paketointikonfiguraatio

5 RAJAPINNAN KÄYTTÖ

Tässä luvussa käydään läpi yleisellä tasolla generoitujen Java-kielen client-toteutusten käyttöä. Ensimmäiseksi käydään läpi REST- ja gRPC-rajapintojen käyttö erikseen, jonka jälkeen havainnollistetaan Reactor-kirjaston tuomia mahdollisuuksia eri tietolähteiden niputtamiseen. Käytetyt esimerkit eivät ole tuotantokelpoisia toteutuksia, esimerkiksi autentikoinnin puutteen vuoksi.

5.1 REST-rajapinnan alustus ja käyttö

REST-rajapinnan käyttö koostuu yksinkertaisimmillaan kahdesta osasta; http-yhteyttä hallinnoivan ApiClientin konfiguroinnista, sekä määritellyn rajapinnan toteutukset sisältävän luokan käytöstä ApiClientin hallinnoiman yhteyden yli.

5.1.1 REST-clientin alustus

Openapi-generator-maven-pluginin java-generaattori käyttää oletus HTTP-client toteutuksena Android- ja Java-sovelluksille tarkoitettua OkHttp-kirjastoa. OkHttp:n käyttö on esimerkiksi paikkatiedon rajapinnan generoinnissa käärittyä ApiClient-nimisen luokan sisälle. Kuvassa 24 on esitelty minimalistinen esimerkki ApiClientin konfiguraatiosta.

```
String host = "127.0.0.1";
String port = "8888";
ApiClient client = new ApiClient();

client.setBasePath(String.format("http://%s:%s", host, port));
client.setConnectTimeout(5000);
client.setReadTimeout(5000);
client.setWriteTimeout(5000);
```

KUVA 24. HTTP-clientin konfigurointi

Konfiguroitua ApiClient-luokkaa käyttämällä generoinnissa syntynyt AliApi-luokka pystyy suorittamaan rajapinnan määrittelemiä toimintoja oikeassa osoitteessa. Kuvassa 24 alustettu ApiClient on annettava parametrina AliApi:n rakentajassa.

5.1.2 REST-clientin käyttö

HTTP-yhteyden konfiguraation sisältävän ApiClient-luokan alustaminen on ainoa vaadittu manuaalinen toimenpide rajapinnan käytössä. ApiClientin rakentajassa saanut AliApi-luokan instanssi on täysin valmis suorittamaan API-kutsuja, mitä on havainnollistettu kuvassa 25.

```
ApiClient client = new ApiClient();

// Configure the ApiClient
...

AliApi aliService = new AliApi(client);

// Synchronous API call
LocationResponse response = aliService.locateNumber("+3581234567890");

// Async
aliService.locateNumberAsync("+3581234567890", new ApiCallback<LocationResponse> {

    @Override
    void onSuccess(LocationResponse result, int statusCode, Map<String, List<String>> responseHeaders) {
        // Handle result here
    }

    // Also onFailure, onUploadProgress and onDownloadProgress methods must be implemented
    ...
});
```

KUVA 25. REST-rajapinnan käyttö

5.2 gRPC-rajapinnan alustus ja käyttö

Generoidun gRPC rajapinnan alustus ja käyttö sisältää vastaavat vaiheet REST:iin verrattuna, eli aluksi konfiguroidaan yhteysväylä, jonka yli rajapinnan metodeita suoritetaan erillisellä client-stubilla.

5.2.1 gRPC-clietin luonti ja alustus

Rajapinnan metodeiden kutsumista varten on luotava yhteys määriteltyyn yhteyspisteeseen. Kuvassa 26 on esimerkki kanavan alustamisesta, ja client-stubin luonnista. gRPC:n ManagedChannel-luokka tarjoaa metodeita yhteyskanavan elinkaaren hallintaan. ReactorALIStub taas on generaation tuloksena syntynyt luokka, joka tarjoaa clientin rajapinnan käyttöä varten. Client-stubille on määriteltävä käytettävä yhteyskanava.

```
String host = "127.0.0.1";
String port = "8888";

ManagedChannel channel = ManagedChannelBuilder.forAddress(host, port).usePlaintext().build();

ReactorALIStub clientStub = ReactorALIGrpc.newReactorStub(channel);
```

KUVA 26. gRPC-kanavan luonti ja alustus

5.2.2 gRPC-rajapinnan käyttö

Aikaisemmin käsitellyn Reactor-laajennuksen vaikutukset (kuva 21) on huomioitava rajapinnan käytössä. Rajapinnan metodin paluuarvon muuttuessa void-tyyppistä Reactorin Mono- tai Flux-luokkaan, niin samalla muuttuu tulosten käsittelykin koodissa. Kuvassa 27 on yksinkertainen esimerkki paikkatiedon rajapinnan käytöstä.

```
ReactorALIStub clientStub;

// Configure channel and assign the stub
...

NumberLocationRequest request = NumberLocationRequest.newBuilder()
                                                     .setPhoneNumber("+3581234567890")
                                                     .build();

clientStub.locateNumber(request).subscribe(new Subsriber<LocationResponse> {

    @Override
    public void onNext(LocationResponse result) {
        // Handle result here
    }

    @Override
    public void onSubscribe(Subscription sub) {
        /**
        Signal the Publisher that this Subscription is demanding n-amount of events.
        NO events will be sent until demand is signaled via this method.
        */
        sub.request(Long.MAX_VALUE);
    }

    // Also onComplete and onError methods must be implemented.
    ...
});
```

KUVA 27. gRPC-rajapinnan käyttö

Vaikka rajapinnan locateNumber-metodi palauttaa Mono-tyypin, mikä tarkoittaa nollaa tai yhtä vastausta rajapinnalta, niin samaa Subscriberia käytettäisiin myös mahdollisesti useita vastauksia palauttavan Fluxin kanssa. Fluxin tapauksessa voitaisiin jo kuvitella käyttötapausta esimerkiksi aikaisemmin mainitun päivittyvän paikkatiedon rajapinnassa.

5.3 Vastausten yhdistäminen Reactorilla

Suunnittelun yhtenä tavoitteena oli mahdollistaa useampi REST ja gRPC yhteyspisteen niputtaminen yhden Responsesta kutsuttavan metodin taakse. Reactorin map- ja merge-metodit tarjosivat tehokkaita työkaluja vastausten yhdistämiseen. Kuvassa 28 on havainnollistettu REST- ja gRPC-kutsujen vastausten yhdistämistä yhdeksi kuunneltavaksi Fluxiksi.

```
public ALIResult locationResponseToALIResult(LocationResponse resultToConvert) {
    // Example of a helper method that maps generated REST and gRPC LocationResponse-types
    // to a single commonly used type (ALIResult).
    // I.e. this is the point where generated code will no longer be used.
    ...
}
String numberToLocate = "+3581234567890";


// ---REST---
// Configured and assigned elsewhere
AliApi aliService;

Mono<ALIResult> restCall = Mono.fromCallable(() -> {
    // Executed after subscription
    LocationResponse restResponse = aliService.locateNumber(numberToLocate);
    return locationResponseToALIResult(restResponse);
});


// ---GRPC---
// Configured and assigned elsewhere
ReactorALIStub clientStub;
NumberLocationRequest request;

Mono<ALIResult> grpcCall = clientStub.locateNumber(request).map(grpcResponse -> {
    return locationResponseToALIResult(grpcResponse);
});


// ---MERGE---
Flux<ALIResult> combinedCall = grpcCall.mergeWith(restCall);

// Use as usual. Subscribe triggers the actual API calls.
combinedCall.subscribe();
```

KUVA 28. REST- ja gRPC kutsujen yhdistäminen

Kuvan 28 esimerkissä mergellä luodun Fluxin subscibe synnyttäisi kaksi onNext-tapahtumassa välitettyä paikannustulosta; yksi vastaus REST-rajapinnalta ja toinen gRPC:ltä. Mergen voi suorittaa myös helposti useammalle Monolle tai Fluxille, käyttämällä merge(Iterable) -metodia, jolle voi syöttää listan kaikista yhdistettävistä kohteista.

6 YHTEENVETO JA POHDINTA

Toteutettu työ oli suhteellisen monipuolinen kokonaisuus, johon kuului toimialatietämyksen kerryttämistä, rajapintojen suunnittelua, toteutusta ja käyttöä. Rajapinnan integroiminen osaksi Responsea rajattiin tämän työn ulkopuolelle, koska siihen liittynyt arkkitehtuurin suunnittelu olisi itsessään voinut melkein olla jo oma opinnäytetyönsä. Rajapintatoteutusten käytön esittelyssä siis tyydyttiin vain yksinkertaisiin esimerkkeihin. Sisältöä työhön löytyi jo riittävästi esimerkiksi paikka ja tilaajatietoon perehtymisen myötä.

Työn toteutuksen aikana kerättiin useita eri näkökulmia paikka- ja tilaajatiedon roolista hätäkeskustoiminnassa. Rajapintamääritelmiin johtaneen tiedon lisäksi saatiin ymmärrystä ALI- ja ANI-termien historiasta, minkä myötä voidaan esimerkiksi välttyä väärinymmärryksiltä tulevaisuuden keskusteluissa eri maiden organisaatioiden kanssa.

Esimerkiksi tilaajatiedon tulevaisuudennäkymistä saatiin myös kiinnostavaa pohdittavaa osaksi tulevaisuuden tuotekehitystä. EENA:n Mobile Identity dokumentin monipuoliset, mutta toistaiseksi visioinnin tasolle jääneet ehdotukset antoivat inspiroivan tunteen niistä mahdollisuuksista, mihin tulevaisuuden hätäkeskustoiminnassa voidaan päästä. Ennen mitään käytännön toteutuksia on kuitenkin ratkottava selviä haasteita infrastruktuurin ja tietosuojan suhteen, mikä tulee vaatimaan panostusta järjestelmien toimittajilta, päättäjiltä, sekä hätäkeskustoimintaa pyörittäviltä tahoilta.

Määrittelyissä kuitenkin päädyttiin hyvien rajapintojen suunnitteluperiaatteiden mukaisesti mahdollisimman yksinkertaisiin ratkaisuihin, mikä ilmeni hyvinkin selkeästi usein toistuvista toteamuksista siitä, että jokin ominaisuus olisi kiva olla, mutta sitä ei ole järkevää toteuttaa juuri nyt. Rajapintojen määrittely jollakin kuvauskielellä ja toteutuksien generointi kuitenkin mahdollistaa vaivattoman rajapinnan laajennuksen tulevaisuudessa.

Työ eteni loppuvaiheeseen asti rajapintojen määrittelyiden ja toteutusten generoinnin osalta, mutta rajapinnan käyttöä Responsesta ei raportin kirjoitushetkellä

oltu vielä viimeistelty. Vaikka tämän raportin näkökulmasta rajapintojen käyttö ei ollut se pääaihe, niin se on kuitenkin tärkeä täytettävä vaatimus, jotta rajapintojen määritelmistä saataisiin mitään arvoa aikaiseksi. Rajapintojen käyttöön liittyvät puutteet ovat luonteeltaan enimmäkseen vain raakaa työtä vaativia vaiheita, joten työssä päästiin hyvinkin lähelle työn alkuperäistä tavoitetta, johon kuului myös rajapintojen käytön integroiminen osaksi Responsea.

Tärkeä osuus työn lopputuloksena saatujen rajapintojen määritelmien hiomisessa oli katselmoinneissa saadut kommentit, joiden myötä saatiin karsittua epäselvyyksiä ja luotua tarkempia määritelmiä esimerkiksi vapaiden tekstikenttien sisällöille, jotta kenttien arvoista voitaisiin esimerkiksi saada hyötyä jossakin muuallakin kuin virheenetsinnässä. Samaa ulkoisten kommenttien korvaamattomuutta voi myös korostaa tämän opinnäytetyöraportin sisällön eheyttämisessä. Suuri kiitos siis kaikille itse työn tekemisessä ja raportin kommentoinnissa apuna olleille henkilöille, koska yksin ei olisi mitenkään päässyt samaan lopputulokseen.

Kiitokset kuuluu myös Insta Group:ille kannustavasta suhtautumisesta työn ja opiskelun yhdistämiseen, sekä kaikille ihmisille toimistolla ja sen ulkopuolella, jotka ovat tehneet tästä kirjoitusurakasta odotettua kevyemmän.

LÄHTEET

uSwitch. 2019. History of Mobile Phones. Luettu 20.4.2019. https://www.uswitch.com/mobiles/guides/history-of-mobile-phones/

EENA. 2011. 112 Service Chain Description. Luettu 8.4.2019. https://eena.org/wp-content/uploads/2018/11/112-Service-Chain-Description.pdf

Insta DefSec Oy. 2019. Hätä- ja hälytyskeskusratkaisut. Luettu 15.3.2019. https://www.instadefsec.fi/ratkaisut/hata-ja-halytyskeskusratkaisut.html

Insta DefSec Oy. 2019. Insta ResponseTM -tuoteperhe. Luettu 15.3.2019. https://www.instadefsec.fi/ratkaisut/hata-ja-halytyskeskusratkaisut/insta-response-e2-84-a2-tuoteperhe

Franklin A Korn. 1940. Calling line identification circuit. Luettu 20.3.2019. https://patents.google.com/patent/US2265844A/en?oq=US34370240A

NENA. 2011. Standard ALI Query Best Practices. Luettu 20.3.2019. https://www.nena.org/page/XML_Schemas

Law insider. 2019. Definition of Automatic Location Identification. Luettu 20.3.2019. https://www.lawinsider.com/dictionary/automatic-location-identification

EENA. 2018. Mobile Identity. Luettu 25.3.2019. https://eena.org/wp-content/uploads/2018/11/Mobile-Identity-Platform-for-the-Emergency-Services.pdf

Open Mobile Alliance. 2018. Mobile Location Protocol. Luettu 25.3.2019. https://www.openmobilealliance.org/release/MLP/OMA-LIF-MLP-V3_1-20040316-C.pdf

EENA. 2018. The Role of Geographic Information Systems in Next Generation 112. Luettu 27.3.2019. https://eena.org/wp-content/uploads/2018/11/Role-of-GIS-in-NG112.pdf

EENA. 2016. Advanced Mobile Location (AML) Specifications & Requirements. Luettu 27.3.2019. https://eena.org/wp-content/uploads/2018/11/Advanced-Mobile-Location-AML-Specifications-requirements.pdf

What Is OpenAPI? 2019. Luettu 13.3.2019. https://swagger.io/docs/specification/about/

Basic Structure. 2019. Luettu 13.3.2019. https://swagger.io/docs/specification/basic-structure/

Swagger UI. 2019. Luettu 13.3.2019. https://swagger.io/tools/swagger-ui/

Google. 2019. Protocol Buffers Developer Guide. Luettu 10.4.2019. https://developers.google.com/protocol-buffers/docs/overview

Container Solutions. 2017. Introduction to gRPC. Luettu 11.4.2019. https://container-solutions.com/introduction-to-grpc/

GRPC. 2019. What is gRPC? Luettu 11.4.2019. https://grpc.io/docs/guides/

Baeldung. 2019. Intro to Reactor Core. Luettu 11.4.2019. https://www.baeldung.com/reactor-core

Apache Maven Project. 2019. What is Maven? Luettu 11.4.2019. https://maven.apache.org/what-is-maven.html

Gabriele Tomassetti. 2018. A Guide to Code Generation. Luettu 13.3.2019. https://tomassetti.me/code-generation/

OpenAPITools. 2019. openapi-generator-maven-plugin. Luettu 15.4.2019. https://github.com/OpenAPITools/openapi-generator/tree/master/modules/openapi-generator-maven-plugin

Xolstice. 2019. Maven Protocol Buffers Plugin. Luettu 15.4.2019. https://github.com/xolstice/protobuf-maven-plugin

LIITTEET

Liite 1. Tilaajatiedon rajapinnan OpenAPI määritelmä 1 (3)

```
openapi: 3.0.0

info:
  title: Automatic Number Identification (ANI) API
  description: >
    This is an API for making ANI-requests from Insta Response.
  contact:
    name: Insta DefSec
    url: http://www.instaresponse.fi
    email: ids_info@insta.fi
  version: 1.0.0

security:
  - basicAuth: []
  - bearerAuth: []

paths:
  /ani/identifyNumber:
    post:
      operationId: getNumberIdentity
      summary: Identify number
      description: Method for requesting identity information for the
given number.
      tags:
        - ANI
      requestBody:
        description: A request containing parsed phone number with coun-
try code
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/NumberIdentityRequest'
      responses:
        '200':
          description: Number identified succesfully
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/IdentityResponse'
        '400':
          description: Malformed number identification message
        '401':
          description: Authentication failed
        '403':
```

```
description: No permission to number identification
```

```
        '404':
          description: No subscription found for given number
        '408':
          description: Number identification request timed out
        '500':
          description: Internal server error
        '504':
          description: Identification response timed out

  /health:
    get:
      operationId: health
      summary: Get the status of a service
      description:  Sometimes a service instance can be incapable of han-
dling requests yet still be running. For example, it might have ran out
of database connections. Dependant services can use this endpoint to in-
quire the status of the service periodically.
      tags:
        - Health
      security: []
      responses:
        '204':
          description: Service is health and capable of handling service
calls.
        'default':
          description: Service is unhealthy. Service calls to other end-
points should be suspended.

components:
  securitySchemes:
    basicAuth:
      type: http
      scheme: basic
    bearerAuth:
      type: http
      scheme: bearer
      bearerFormat: shared secret
  schemas:
    NumberIdentityRequest:
      type: object
      properties:
        phoneNumber:
          type: string
          description: A string representation of the phone number to lo-
cate. Containing a country code and NO whitespace.
          example: '+4981234567890'
    IdentityResponse:
      type: object
      properties:
```

```
        name:
          type: string
          description: Full name of the subscription owner.
          example: John Smith
        address:
          description: Billing or installation address.
          $ref: '#/components/schemas/Address'
        phoneType:
          type: string
          description: >
            Describes the type of the phone.
            MOBILE means a device's location may be seperate from the
billing or installation address. A mobile device in most cases has to be
located by other means (e.g. ALI).
            A FIXED phone is tied to its installation address, e.g. land-
lines. No seperate locationing should be required for these types of de-
vices, nor may not even be possible.
          enum: [MOBILE, FIXED]
    Address:
      type: object
      properties:
        country:
          type: string
          description: ISO 3166-1 alpha-2 country code.
          example: US
        region:
          type: string
          description: Region should be primarily interpreted as ISO
3166-2
            region code and secondarily as a regular text.
          example: US-LA
        locality:
          type: string
          description: Geographic area that makes the interpretation of
street address information combined with country and region unambigiuos.
City, municipality or similar.
          example: New Orleans
        streetAddress:
          type: string
          description: Street address.
          example: 701 Bourbon St
```

Liite 2. Tilaajatiedon rajapinnan protobuf määritelmä 1 (2)

```
syntax = "proto3";

option java_multiple_files = true;

package fi.insta.response.automatic.number.identifier.generated.proto-
buf.v1.api;

/**
 * This is an API for making ANI-requests from Insta Response.
 */
service ANI {

  /**
   * Method for requesting identity information for the given number.
   */
  rpc IdentifyNumber (NumberIdentityRequest) returns (IdentityResponse);

}

/** A request containing parsed phone number with country code */
message NumberIdentityRequest {
  /** A string representation of the phone number to identify, containing
a country code and NO whitespace. E.g. "+4981234567890" */
  string phoneNumber = 1;
}

/** Contains the identifying information related to the given number  */
message IdentityResponse {
  /** Full name of the subscription owner. E.g. "John Smith" */
  string name = 1;
  /** Billing or installation address. */
  Address address = 2;
  /** Type of the phone. */
  PhoneType phoneType = 3;

  /** Describes the type of the phone. */
 enum PhoneType {
   /** Device's location may be seperate from the billing or installation
address. A mobile device in most cases has to be located by other means
(e.g. ALI). */
   MOBILE = 0;
   /** Phone is tied to its installation address, e.g. landlines. No
seperate locationing should be required for these types of devices, nor
may not even be possible. */
   FIXED = 1;
 }
}
```

```
message Address {
  /** ISO 3166-1 alpha-2 country code, e.g. "US". */
  string country = 1;
  /** Region should be primarily interpreted as ISO 3166-2, e.g. "US-LA".
*/
  string region = 2;
  /** Geographic area that makes the interpretation of street address in-
formation combined with country and region unambigiuos. City, municipal-
ity or similar. E.g. "New Orleans". */
  string locality = 3;
  /** Street address, e.g. "701 Bourbon St". */
  string streetAddress = 4;
}
```

Liite 3. Paikkatiedon rajapinnan OpenAPI määritelmä 1 (4)

```yaml
openapi: 3.0.0

info:
  title: Automatic Location Identification (ALI) API
  description: >
    This is an API for making ALI-requests from Insta Response.
  contact:
    name: Insta DefSec
    url: http://www.instaresponse.fi
    email: ids_info@insta.fi
  version: 1.0.0

security:
  - basicAuth: []
  - bearerAuth: []

paths:
  /ali/locateNumber:
    post:
      operationId: getNumberLocation
      summary: Locate number
      description: Returns a list of location results for the given num-
ber.
      tags:
          - ALI
      requestBody:
        description: A request containing parsed phone number with coun-
try code
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/LocationRequest'
      responses:
        '200':
          description: A list of location results.
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/LocationResponse'
        '400':
          description: Malformed number identification message
        '401':
          description: Authentication failed
        '403':
          description: No permission to number location
        '408':
```

```
          description: Location request timed out
        '500':
          description: Internal server error
        '504':
          description: Location response timed out

  /health:
    get:
      operationId: health
      summary: Get the status of a service
      description:  Sometimes a service instance can be incapable of han-
dling requests yet still be running. For example, it might have ran out
of database connections. Dependant services can use this endpoint to in-
quire the status of the service periodically.
      tags:
        - Health
      security: []
      responses:
        '204':
          description: Service is health and capable of handling service
calls.
        'default':
          description: Service is unhealthy. Service calls to other end-
points should be suspended.

components:
  securitySchemes:
    basicAuth:
      type: http
      scheme: basic
    bearerAuth:
      type: http
      scheme: bearer
      bearerFormat: shared secret

  schemas:
    LocationRequest:
      type: object
      properties:
        phoneNumber:
          type: string
          description: A string representation of the phone number to lo-
cate. Containing a country code and NO whitespace.
          example: '+4981234567890'

    LocationResponse:
      type: object
      properties:
        parentSource:
```

```
          type: string
          description: Represents an single endpoint called from Re-
sponse. Enumeration-like format with UPPERCASE letters and underscores
should be used.
          example: NATIONAL_ALI_INTEGRATION
        results:
          type: array
          items:
            $ref: '#/components/schemas/Location'

    Location:
      type: object
      properties:
        timestamp:
          type: string
          format: date-time
          description: ISO 8601 date-time in millisecond precision. Date-
time must be given in Zulu-timezone, and zone identifier 'Z' must be in-
cluded in the value. Defines how recent this location information is.
          example: '2019-08-17T06:15:04.321Z'
        latitude:
          type: number
          format: double
          description: Decimal representation of latitude, based on WGS84
standard.
          example: 29.95876
        longitude:
          type: number
          format: double
          description: Decimal representation of longitude, based on
WGS84 standard.
          example: -90.06557
        radius:
          type: integer
          format: int32
          description: Describes the accuracy of the location result as a
circle's radius in meters.
          example: 2
        source:
          type: string
          description: Describes the source for this location result.
Enumeration-like format with UPPERCASE letters and underscores should be
used.
          example: NETWORK_TRIANGULATION
        resultInfo:
          $ref: '#/components/schemas/ResultInfo'

    ResultInfo:
      type: object
```

```
      properties:
        errorMessage:
          type: string
          description: A string field for additional information about
the failed location result for debugging purposes. MUST NOT contain any
personal identifying information!
        resultCode:
          type: string
          description: >
            OMA SpecWorks Mobile Location Protocol -inspired result
codes:
              * `OK` - No error occurred while processing the request.

              * `SYSTEM_FAILURE` - The request can not be handled because
of a general problem in the location server.

              * `UNSPECIFIED_ERROR` - An unspecified error used in case
none of the other errors apply.

              * `UNAUTHORIZED_APPLICATION` - The requesting location-
based application is not allowed to access the location server or a wrong
password has been supplied.

              * `UNKNOWN_SUBSCRIBER` - Unknown subscriber. The user is
unknown, i.e. no such subscription exists.

              * `ABSENT_SUBSCRIBER` - Absent subscriber. The user is cur-
rently not reachable.

              * `POSITION_METHOD_FAILURE` - Position method failure. The
location service fails to obtain the user’s position. The exact cause may
be indicated in the errorMessage-field.

              * `TIMEOUT` - Timer expiry for the requested location
method
          enum:
            - OK
            - SYSTEM_FAILURE
            - UNSPECIFIED_ERROR
            - UNAUTHORIZED_APPLICATION
            - UNKNOWN_SUBSCRIBER
            - ABSENT_SUBSCRIBER
            - POSITION_METHOD_FAILURE
            - TIMEOUT
```

Liite 4. Paikkatiedon rajapinnan protobuf määritelmä 1 (3)

```
syntax = "proto3";

option java_multiple_files = true;

package fi.insta.response.automatic.location.identification.gener-
ated.protobuf.v1.api;

/**
 * This is an API for making ALI-requests from Insta Response.
 */
service ALI {

  /**
   * Returns a list of location results for the given number.
   */
  rpc LocateNumber (NumberLocationRequest) returns (LocationResponse);

}

/* Contains a parsed phone number with country code */
message NumberLocationRequest {
  /** A string representation of the phone number to locate, containing a
country code and NO whitespace. E.g. "+4981234567890" */
  string phoneNumber = 1;
}

message LocationResponse {
  /** A list of location results. */
  repeated Location results = 1;
  /** Represents an single endpoint called from Response. Enumeration-
like format with UPPERCASE letters and underscores should be used. E.g.
"NATIONAL_ALI_INTEGRATION" */
  string parentSource = 2;
}

message Location {
  /** ISO 8601 date-time in millisecond precision. Date-time must be
given in Zulu-timezone, and zone identifier 'Z' must be included in the
value. Defines how recent this location information is. E.g. "2019-08-
17T06:15:04.321Z" */
  string timestamp = 1;
  /** Decimal representation of latitude, based on WGS84 standard. E.g.
"29.95876" */
  double latitude = 2;
  /** Decimal representation of longitude, based on WGS84 standard. E.g.
"-90.06557" */
  double longitude = 3;
```

```
  /** Describes the accuracy of the location result as a circle's radius
in meters. E.g. "2" */
  int32 radius = 4;
  /** Describes the source for this location result. Enumeration-like
format with UPPERCASE letters and underscores should be used. E.g. "NET-
WORK_TRIANGULATION" */
  string source = 5;
  /** Holds information about the success of this single location at-
tempt. */
  ResultInfo resultInfo = 6;
  }

  message ResultInfo {
    /** A string field for additional information about the failed loca-
tion result for debugging purposes. MUST NOT contain any personal identi-
fying information! */
    string errorMessage = 1;
    ResultCode resultCode = 2;

    /** OMA SpecWorks Mobile Location Protocol -inspired result codes */
    enum ResultCode {
      /** No error occurred while processing the request. */
      OK = 0;

      /** The request can not be handled because of a general problem in
the location server. */
      SYSTEM_FAILURE = 1;

      /** An unspecified error used in case none of the other errors ap-
ply. */
      UNSPECIFIED_ERROR = 2;

      /** The requesting location-based application is not allowed to ac-
cess the location server or a wrong password has been supplied. */
      UNAUTHORIZED_APPLICATION = 3;

      /** Unknown subscriber. The user is unknown, i.e. no such subscrip-
tion exists. */
      UNKNOWN_SUBSCRIBER = 4;

      /** Absent subscriber. The user is currently not reachable. */
      ABSENT_SUBSCRIBER = 5;

      /** Position method failure. The location service fails to obtain
the user’s position. The exact cause may be indicated in the er-
rorMessage-field. */
      POSITION_METHOD_FAILURE = 6;
```

```
    /** Timer expiry for the requested location method. */
    TIMEOUT = 7;
  }
}
```

Liite 5. Paikkatiedon REST-rajapinnan generointi

```
<plugin>
    <!-- Generate classes for REST-API for internal use -->
    <groupId>org.openapitools</groupId>
    <artifactId>openapi-generator-maven-plugin</artifactId>
    <version>3.3.4</version>
    <executions>
        <execution>
            <goals>
                <goal>generate</goal>
            </goals>
            <configuration>
                <inputSpec>${project.basedir}/src/main/ali/openapi/fi.in-
sta.response.automatic.location.identification.api.yaml</inputSpec>
                <generatorName>java</generatorName>
                <generateApiTests>false</generateApiTests>
                <apiPackage>fi.insta.response.automatic.location.identi-
fication.generated.openapi.api</apiPackage>
                <modelPackage>fi.insta.response.automatic.location.iden-
tification.generated.openapi.model</modelPackage>
                <invokerPackage>fi.insta.response.automatic.loca-
tion.identification.generated.openapi.invoker</invokerPackage>
                <configHelp>true</configHelp>
                <configOptions>
                    <sourceFolder>/</sourceFolder>
                    <dateLibrary>java8</dateLibrary>
                    <java8>true</java8>
                </configOptions>
            </configuration>
        </execution>
    </executions>
</plugin>
```

Liite 6. Paikka- ja tilaajatiedon gRPC –rajapinnan generointi

```
<plugin>
    <!-- Generate classes from protos for internal use -->
    <groupId>org.xolstice.maven.plugins</groupId>
    <artifactId>protobuf-maven-plugin</artifactId>
    <version>0.6.1</version>
    <configuration>
        <protocArtifact>com.google.protobuf:protoc:${protoc.ver-
sion}:exe:${os.detected.classifier}</protocArtifact>
        <pluginId>grpc-java</pluginId>
        <pluginArtifact>io.grpc:protoc-gen-grpc-java:${grpc.ver-
sion}:exe:${os.detected.classifier}</pluginArtifact>
        <protoSourceRoot>src/main</protoSourceRoot>
        <checkStaleness>true</checkStaleness>
    </configuration>
    <executions>
        <execution>
            <goals>
                <goal>compile</goal>
                <goal>compile-custom</goal>
            </goals>
            <!-- For Reactor gRPC -->
            <configuration>
                <protocPlugins>
                    <protocPlugin>
                        <id>reactor-grpc</id>
                        <groupId>com.salesforce.servicelibs</groupId>
                        <artifactId>reactor-grpc</artifactId>
                        <version>${reactive.grpc.version}</version>
                        <mainClass>com.salesforce.reactorgrpc.Reac-
torGrpcGenerator</mainClass>
                    </protocPlugin>
                </protocPlugins>
            </configuration>
        </execution>
    </executions>
</plugin>
```