TALLINNA TEHNIKAÜLIKOOL
TALLINN UNIVERSITY OF TECHNOLOGY

Infotehnoloogia teaduskond
Arvutiteaduse instituut
Võrgutarkvara õppetool

ITV40LT

Janno Põldma 061808IAPB

# *F#* TÜÜBIPAKKUJA ANDMEVAHETUSKIHILE *X-TEE*

Bakalaureusetöö

Juhendaja: Jaagup Irve

Tehnikateaduste magister

Tallinn 2015

# Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

. . . . . .  . . . . . .

(kuupäev)  (allkiri)

# Annotatsioon

Käesoleva töö peamiseks eesmärgiks on luua *F#* programmeerimiskeele võimalusi kasutades tüübipakkuja teek, mis teeb võimalikuks andmevahetuskihi *X-tee* teenuste ja andmetüüpide sujuvama kasutuselevõtu *.NET* tarkvaraplatvormil.

Töös antakse ülevaade andmevahetuskihist *X-tee* ning kirjeldatakse teenuste kasutuselevõtmise tööprotsessi. Samuti tutvustatakse *F#* programmeerimiskeelt ja tüübipakkujate üldist ülesehitust.

Töö praktilises osas analüüsitakse, milliseid *.NET* tarkvaraplatvormi võimalusi kasutada tüübipakkuja realiseerimisel, põhjendatakse kasutatavate komponentide valikut ning kirjeldatakse loodava lahenduse ülesehitust. Lisaks tuuakse välja mõned valitud näited tüübipakkuja praktilisest kasutamisest reaalsete *X-tee* teenusepakkujatega.

Töö tulemusena valmib esialgse funktsionaalsusega tüübipakkuja, mis võimaldab lahenduse perspektiivikust demonstreerida valitud *X-tee* teenusepakkujate andmetüüpide tuvastamise ja teenuseliideste genereerimisega.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 56 leheküljel, 3 peatükki, 6 joonist ja 10 koodinäidist.

# Abstract

Main goal of this thesis is to create *F#* type provider for *.NET* framework, that allows seamless integration with data types and services provided by data exchange layer *X-Road* (*X-tee* in Estonian).

This thesis gives brief overview of data exchange layer *X-Road* and describes common process that is used to implement service interfaces. Also, main features of *F#* programming languages are introduced, particulary type provider mechanism which is the core of this thesis.

Practical part of the thesis analyses *.NET* framework features that are available to implement required solution, and reasons about software components that are chosen. The design and inner workings of the type provider will also be described. The work will present some practical examples of the type provider prototype in use with real *X-Road* service provider.

As the result of this work a type provider with initial functionality is built, which allows to demonstrate perspective of the solution that helps developers with detecting types in integrated development environments and provides functional service interfaces in compiled applications.

The thesis is written in Estonian and contains 56 pages of text, 3 chapters, 6 figures and 10 code examples.

# Sisukord

**Jooniste nimekiri** **7**

**Koodinäidiste nimekiri** **8**

**Lühendite ja mõistete sõnastik** **9**

**Sissejuhatus** **10**

**1 Tehnoloogiline platvorm** **11**

1.1 Andmevahetuskiht *X-tee* . . . 11

1.1.1 Teenuste kirjeldamine . . . 12

1.1.2 *X-tee* protokollide versioonid . . . 14

1.1.3 *WSDL* ja *SOAP* standardite sidumise stiil . . . 15

1.1.4 Manuseid sisaldavad sõnumid . . . 16

1.1.5 Teenuste kasutamine . . . 17

1.1.6 Üldine arendaja tööprotsess *X-tee* teenuse kasutuselevõtmisel . . . 17

1.2 *F#* programmeerimiskeel . . . 18

1.2.1 Arendusvahendid . . . 20

1.3 *F#* keele tüübipakkujad . . . 21

1.3.1 Kaduvate andmetüüpidega tüübipakkujad . . . 22

1.3.2 Genereeritud andmetüüpidega tüübipakkujad . . . 23

1.3.3 Tüübipakkuja ülesehitus . . . 24

**2 *X-tee* tüübipakkuja realiseerimine** **26**

2.1 Eesmärgid ja nendest tingitud piirangud . . . 26

2.2 Tüübipakkuja liigi valik . . . 27

2.3 Genereeritava koodi planeerimine . . . 28

2.3.1 Pakutavate andmetüüpide paigutus . . . 28

2.3.2 Tegevused teenuse väljakutsumisel . . . 29

2.3.3 Sõnumites edastatavate objektide serialiseerimine . . . 30

2.3.4 *MIME* manustega päringute võimaldamine . . . 32

2.4 Tüübipakkuja arendamine . . . 34

2.4.1 *WSDL* dokumendi töötlemine . . . 34

2.4.2 Koodigenereerimise tehnoloogia valik . . . 36

2.4.3 Andmetüüpide genereerimine . . . 38

2.4.4 Prototüübis realiseeritud teenusekirjelduse elemendid . . . 39

2.4.5 Teenuseid väljakutsuvate meetodite ülesehitus . . . 41

2.4.6 Tüübipakkuja liidese realiseerimine . . . . . 42

**3 Lahenduse praktiline kasutamine 44**

3.1 Tüübipakkuja kasutamine arendusvahendis . . . . . 44

3.2 Tüübipakkuja testimine valitud teenusepakkujatega . . . . . 45

3.2.1 *X-tee* protokoll 2.0 juhendi näidisteenus . . . . . 45

3.2.2 *X-tee* protokoll 3.1 juhendi näidisteenus . . . . . 46

3.2.3 *E-toimik* . . . . . 47

3.3 Tulemused . . . . . 50

3.3.1 Edasiarendusvõimalused . . . . . 51

**Kokkuvõte 53**

**Summary 54**

**Kasutatud kirjandus 55**

# Jooniste nimekiri

1 *X-tee* põhimõtteline skeem . . . . . 12
2 *X-tee* teenusekirjeldustes kasutatavad standardid . . . . . 13
3 Kaduvate tüüpidega tüübipakkuja tööpõhimõte . . . . . 23
4 Genereeritud tüüpidega tüübipakkuja tööpõhimõte . . . . . 24
5 Genereeritud andmetüüpide ülesehitus . . . . . 29
6 Tüübipakkuja kasutamise näidissession *Visual Studio* projektis . . . . . 44

# Koodinäidiste nimekiri


1 Kaduvate tüüpidega tüübipakkuja kasutamine . . . . . 22
2 Kaduvate tüüpide kompileerimisel tekkiv kood . . . . . 23
3 Teenuse väljakutsel teostatavad tegevused . . . . . 29
4 *MIME* manuseid toetav andmetüüp . . . . . 33
5 Binaarse sisuga andmetüübilt sisu eemaldamine *XML* sõnumis . . . . . 33
6 *XML Schema simpleType* elemendi lugemine . . . . . 35
7 *X-tee* teenuse realisatsioon . . . . . 41
8 *E-toimiku* liidese loomine . . . . . 47
9 Isiku lisamine ja vaatamine *E-toimikus* . . . . . 47
10 Faili lisamine ja vaatamine *E-toimikus* . . . . . 49

# Lühendite ja mõistete sõnastik

**MIME** *Multi-Purpose Internet Mail Extensions*

Interneti e-kirjade protokolli laiendus, mis võimaldab protkolli vahendusel edastada erinevat tüüpi andmefaile, nt. pildi-, heli-, video- ja rakendusfaile.

**RPC** *Remote Procedure Call*

Protsessidevaheline suhtlusviis, mis võimaldab programmil käivitada teises asukohas paiknevat meetodit.

**SOAP** *Simple Object Access Protocoll*

Arvutivõrkudes kasutatav protokoll, mille alusel veebiteenused omavahel struktuurseid andmeid vahetavad.

**WSDL** *Web Services Description Language*

*XML*-põhine rakendusliidese kirjeldamise formaat, mis spetsifitseerib veebiteenuse poolt pakutud teenuseid.

**XML** *Extensible Markup Language*

Andmeedastusformaat, mis pakub kindlatele reeglitele tuginedes vormingut, mis on samaaegselt inim- ja masinloetav.

**XML-RPC** *Extensible Markup Language Remote Procedure Call*

*RPC* protokollile ehitatud suhtlusviis, mis kasutab teise asukoha meetodite väljakutsumiseks *XML* vormingut.

**XML Schema**

Reeglite hulk, mis võimaldab kokku panna *XML* sõnumi definitsiooni, et kehtestada piiranguid kasutatavate elementide, tüüpide ja atribuutide kohta.

# Sissejuhatus

*X-tee* on peamine andmevahetuskiht, mis seob omavahel erinevaid riigi infosüsteeme ja korraldab nendevahelist andmevahetust. *X-tee* teenusepakkujatega liidestumisel tuleb reeglina läbi teha etapp, kus teenusepakkuja spetsifikatsiooni alusel luuakse klientsüsteemis andmevahetuseks vajalikud andmetüübid ning neid kasutavad teenuseliidesed. Selle sammu läbimiseks on mitmeid lähemisvõimalusi, kuid arendajale on üks mugavamaid variante kasutada oma arendusplatvormile loodud koodigeneraatorit, mis spetsifikatsiooni alusel vajalikud klassid loob. Antud töö peamiseks eesmärgiks on luua lahendus, mis eemaldab tööprotsessist käsitsi koodigenereerimise sammu ning muudab selle klientrakenduse kompileerimise automaatseks osaks.

Eesmärgi täitmiseks on käesolev bakalaureusetöö jagatud järgmistesse etappidesse:

- Andmevahetuskihi *X-tee* tutvustamine ja teenusepakkuja spetsifikatsiooni formaadist, mille alusel vajalikud andmetüübid ja teenuseliidesed luuakse, ülevaate andmine.

- Ülevaade *F#* programmeerimiskeelest, mis pakub oma võimaluste hulgas ka antud lahenduse poolt kasutatavat kompileerimisprotsessi laiendamise võimalust. Samuti tutvustatakse programmeerimiskeelega seotud taristut.

- Analüüsitakse võimalikke tarkvaraarendusplatvormi komponente, mida lahenduse ülesehitamisel kasutada, kirjeldatakse loodava rakendusteegi sisemist ülesehitust ning põhjendatakse valitud tehnoloogiate valikut.

Väljatoodud etappide tulemuste põhjal pannakse kokku rakendusteegi prototüüp, mida tutvustatakse töö viimases osas. Seal esitatakse praktilised näited lahenduse kasutamise kohta, selgitatakse välja probleemsed kohad ja pakutakse välja võimalusi nende parandamiseks ning arutletakse edasiste rakenduse täiendamisvõimaluste üle.

Loodud lahenduse lähtekood on avalikult saadaval *GitHub*-is aadressil:
`https://github.com/janno-p/XRoadProvider`.

Kompileeritud kujul on lahendus saadaval *NuGet*-i paketina aadressil:
`http://www.nuget.org/packages/XRoadProvider`.

# 1 Tehnoloogiline platvorm

Käesolevas peatükis antakse ülevaade andmevahetuskihist *X-tee*, millega liidestumist üritatakse antud töö tulemusena arendaja jaoks lihtsamaks ja mugavamaks muuta. Samuti kirjeldatakse tehnoloogilist platvormi, millel loodav tehniline lahendus põhineb. Lisaks tutvustatakse peamiseid arendusvahendeid, mis uut funktsionaalsust ära oskavad kasutada ja selle tulemusena kiiremat arendusprotsessi võimaldavad.

## 1.1 Andmevahetuskiht *X-tee*

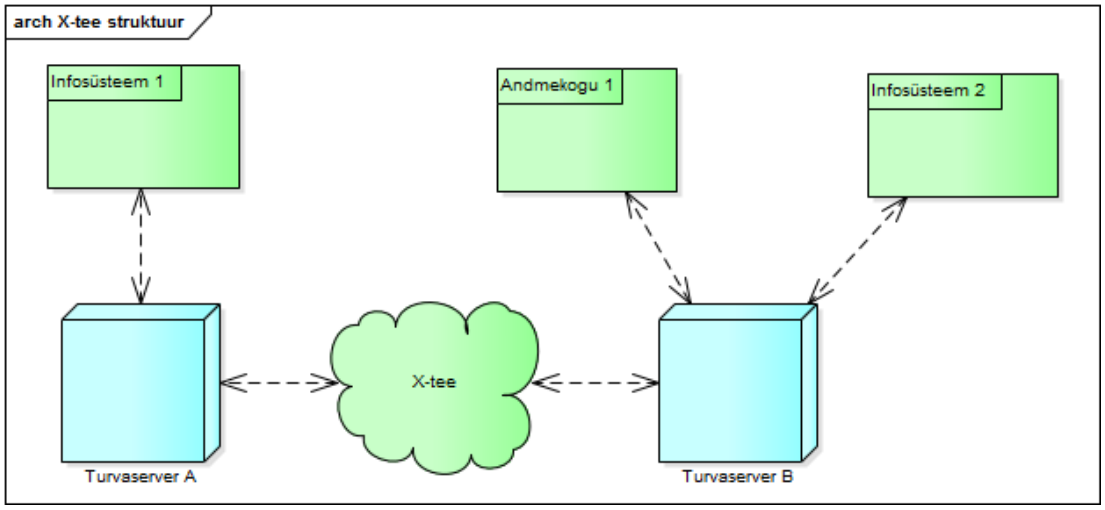
Andmevahetuskiht *X-tee* on riigi infosüsteemide tehniline ja organisatsiooniline keskkond, mis võimaldab korraldada turvalist internetipõhist andmevahetust riigi infosüsteemide vahel. *X-tee* võimaldab asutustel/inimestel turvaliselt andmeid vahetada, samuti korraldada isikute juurdepääsu riigi andmekogudes säilitatavatele ja töödeldavatele andmetele. [1]

*X-tee* pakub mitmekülgseid turvalahendusi: autentimine, mitmetasemeline autoriseerimine, kõrgetasemeline logide töötlemise süsteem ning krüpteeritud ja ajatempliga varustatud andmeliiklus. [1]

*X-tee* andmevahetuskihi põhimõtteline ülesehitus seisneb selles, et erinevad infosüsteemid omavahel sõnumeid otse ei vaheta. Selle asemel on igal infosüsteemil oma turvaserver, mis vahendab kõiki antud infosüsteemi sõnumeid vastavalt andmevahetusprotokollile. Adressaadist lähtuvalt oskab turvaserver teateid suunata teistele sama turvaserveri juures registreeritud andmekogudele või suhelda väliste turvaserveritega, kes oma andmekogusid teenindavad. *X-tee* põhimõtteline skeem on esitatud joonisel 1 (lk. 12).

Igale andmekogule on *X-tee* andmevahetuskihi kasutamiseks defineeritud oma unikaalne teenusepakkuja tunnus - *producer* nimetus. Antud nimetust kasutab *X-tee* infrastruktuur infosüsteemi poole pöördumisel tema teenusekirjelduste küsimisel või teenuste kasutamisel. Vanemates *X-tee* versioonides on reeglina üks nimetus seotud kindla asutuse ja infosüsteemiga, kuid uuemates on lisatud võimalus alamandmekogude loomiseks. [2]

Enda andmekogude ja teiste turvaserverite teenindamiseks defineerib konkreetne turvaserver mitmeid metateenuseid, mis teevad võimalikuks erinevate infosüsteemide




Joonis 1: *X-tee* põhimõtteline skeem

ja nende poolt pakutavate teenuste avastamise. Näiteks teenus `xrd.listProducers` on realiseeritud asutuse turvaserveris ning see väljastab olemasolevate andmekogude loetelu. Teenus on mõeldud kasutamiseks infosüsteemist andmekogude nimede automaatseks kättesaamiseks. Nimekirjas olevate andmekogude käest on hiljem võimalik metapäringuga `allowedMethods` omakorda teada saada andmekogu poolt pakutavaid lubatud teenuseid. [2]

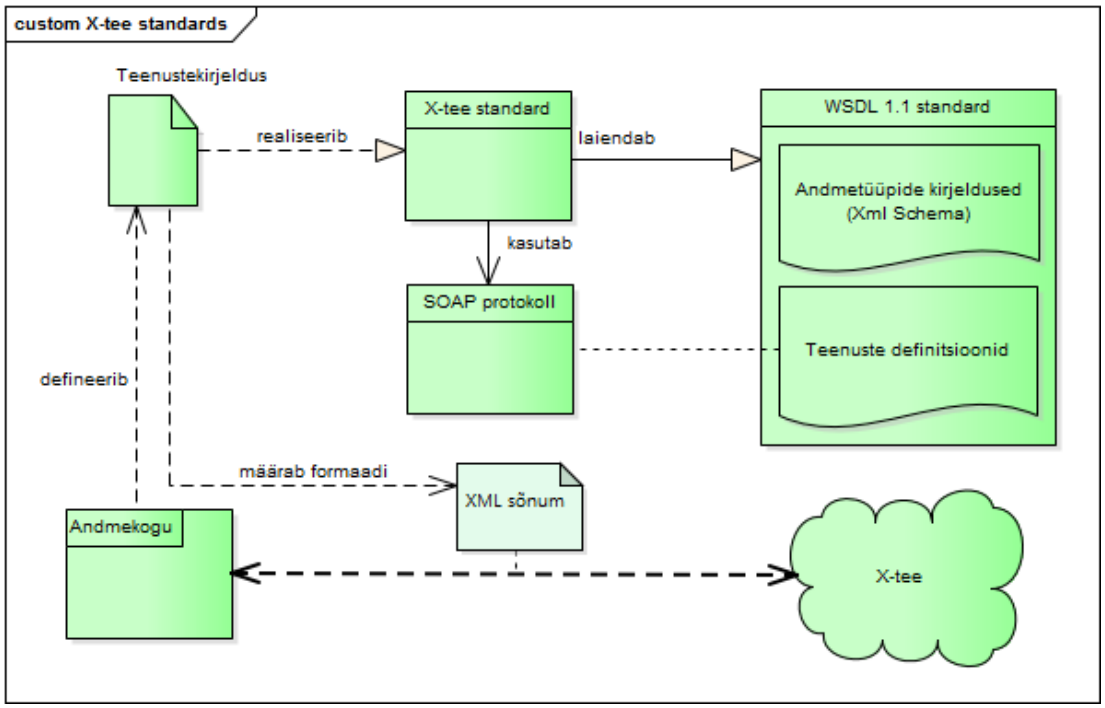
Kuna käesoleva tööga realiseeritav lahendus põhineb andmevahetuskihi *X-tee* turvaserverite poolt jagatavate andmekogude teenusekirjelduste kasutamisel andmetüüpide ja teenusekirjelduste loomiseks, siis järgmisena kirjeldatakse seda osa *X-tee* funktsionaalsusest detailsemalt.

### 1.1.1 Teenuste kirjeldamine

Teenuste kasutamine andmevahetuskihi *X-tee* vahendusel tähendab kindla eelnevalt kokkulepitud formaadiga sõnumite vahetamist infosüsteemide ja/või turvaserverite vahel. Konkreetsemalt on tegu *XML* formaadis dokumentidega, mis alluvad *SOAP*-standardi [3] sõnumiedastusprotokollile.

Selleks, et sõnumid oleks kõikidele osapooltele üheselt mõistetaval kujul, peavad vastavad teenused olema kirjeldatud kindlas formaadis, mis määraks ära piirangud, millega sõnumite koostamisel ja vastuvõtmisel arvestada tuleb. Andmevahetuskiht *X-tee* eeldab teenusekirjelduste koostamisel *WSDL* 1.1 [4] formaadi kasutamist. Iga andmekogu peab oma kõikide teenuste kirjeldused esitama ühises *WSDL* dokumendis, mida *X-tee* turvaserver vahendab teistele osapooltele liidestumise otstarbeks.

Lisaks teenustele kirjeldatakse *WSDL* dokumendis ka kõik andmetüübid, millega teenused opereerivad. Andmetüüpide definitsioonid esitatakse *WSDL* dokumendis ilmutatud kujul *XML Schema* [5] formaadis. Soovi korral võib teenusekirjelduse looja kasutada andmetüüpe ka teistes asukohtades defineeritud failidest, kuid sellisel juhul peab lähtedokument neile standardile vastavalt viitama.




Joonis 2: *X-tee* teenusekirjeldustes kasutatavad standardid

Andmevahetuskiht *X-tee* täiendab *WSDL* standardit omapoolsete elementide lisamisega teenusekirjeldustesse ja nendel põhinevatesse sõnumitesse. Antud töö seisukohast olulisemad elemendid on järgmised [2] (siin on välja toodud *X-tee* protokoll 3.1 väljad, mis on vahetult ülekantavad vanemates versioonides kasutatavatele elementidele):

- `/definitions/service/port/xrd:address` - väärtus puudub, kuid kasutusel on atribuut `producer`, mille väärtuseks on andmekogu nimi.
- `/definitions/service/port/xrd:title` - andmekogu pealkiri (kasutajale näitamiseks).
- `/definitions/binding/operation/xrd:version` - teenuse versioon.
- *X-tee* spetsiifilised *SOAP*-päise (*Header*) elemendid: `consumer`, `producer`, `userId`, `id`, `service`, `issue`, jt. - kannavad täiendavat teavet kasutatava teenuse, konkreetse päringu ja reaalse kasutaja kohta.

*X-tee* andmekogu teenusekirjelduse seos erinevate standardite ja protokollidega on visuaalselt kujutatud joonisel 2 (lk. 13).

### 1.1.2 *X-tee* protokollide versioonid

Seoses uute tehnoloogiate kasutuselevõtuga ja täiendava funktsionaalsuse loomisega tehakse muudatusi andmevahetuskihi *X-tee* protokollis. Versioonimuudatused on põhjustanud ka muudatusi andmekogude teenusekirjeldustes ning sellest tulenevalt ka edastatavate sõnumite struktuurides.

Käesoleva töö kirjutamise hetkeks on protokollide erinevused peamiselt sõnumiedastuseks kasutatava standardi ja *WSDL* ja *SOAP* standardite sidumise stiili (*binding style*) osas. Andmevahetuskihi *X-tee* protokolli versioonid on järgmised [6]:

- **Versioon 1.0** (ei ole kasutuses)
  Baseerub *XML-RPC* standardile.

- **Versioon 2.0** (kasutusel, kuid pole soovitatav)
  Baseerub *SOAP* standardile, kasutab *RPC/Encoded* stiili. Kasutab *X-tee* nimeruumi `http://x-tee.riik.ee/xsd/xtee.xsd`. Andmekogu nimeruum on kujul `http://producers.name.xtee.riik.ee/producer/name`.

- **Versioon 3.0** (kasutusel, kuid pole soovitatav)
  Baseerub *SOAP* standardile, kasutab *Document/Literal wrapped* stiili. Kasutab *X-tee* nimeruumi `http://x-rd.net/xsd/xroad.xsd`. Andmekogu nimeruum on kujul `http://name.ee.x-rd.net/producer/`.

- **Versioon 3.1** (kasutusel, asendab versiooni 3.0)
  Baseerub *SOAP* standardile, kasutab *Document/Literal wrapped* stiili. Kasutab *X-tee* nimeruumi `http://x-road.ee/xsd/x-road.xsd`. Andmekogu nimeruum on kujul `http://name.x-road.ee/producer/`.

- **Versioon 4.0** (pole veel kasutusel)
  Baseerub *SOAP* standardile, kasutab *Document/Literal wrapped* stiili. Teenuste defineerimise nimeruumi ei normeerita *X-tee* protokolli tasandil. [7]

Antud töös jäävad käsitlusest välja versioonid, mis ei ole enam või veel kasutuses. Samuti jäetakse kõrvale versioon 3.0, kuna funktsionaalsuses erineb see 3.1 versioonist minimaalselt. Selline versioonide valik tähendab, et loodav lahendus peab toetama

kahte erinevat protokolli versiooni, mille peamine sisuline erinevus seisneb kasutatavate *WSDL* ja *SOAP* standardite sidumise stiilis.

### 1.1.3 *WSDL* ja *SOAP* standardite sidumise stiil

*SOAP* standardi sidumise stiil defineerib, millist sõnumiedastusprotokolli kuju kirjeldatav teenus kasutab. Kasutada on kaks esitusviisi: *Document* (sõnumid, mis sisaldavad dokumente) ja *RPC* (sõnumid, mis sisaldavad parameetreid ja tagastatavaid väärtuseid). Kasutatav väärtus mõjutab peamiselt seda, kuidas pannakse kokku *SOAP* sõnumi keha (*Body*) element: [4]

- *RPC* stiili korral on iga *WSDL* dokumendis defineeritud sõnumi osa (*message part*) vastavalt teenuse parameeter või tagastatav väärtus. Kõik nimetatud osad lisatakse sõnumis ühise *SOAP* keha sees paikneva elemendi alamelementideks, mille nimi tuletatakse konkreetse teenuse nimest ja tema definitsiooni juures välja toodud nimeruumi väärtusest.

- *Document* stiil ei defineeri eraldi ülemelementi. Selle asemel edastatakse kõik sõnumi osad vahetult *SOAP* keha elemendi alamelementidena.

Stiilile lisaks tuleb teenuse kirjelduses anda teada, kas sõnumi osad edastatakse teatud kodeeringus. Kodeeringu kasutamine defineeritakse teenuse *SOAP* keha elemendi kirjelduses, kasutades atribuuti `use`. Nimetatud väärtus mõjutab edastatava sõnumi esitust järgnevalt: [4]

- `encoded` väärtus tähistab, et sõnumi osade edastamisel oodatakse kooderingu kasutamist. Nõutud kodeeringu tuvastab *SOAP* keha kirjelduse atribuudi `encodingStyle` väärtus ning *X-tee* teenuste puhul on kasutusel *SOAP* kodeering (*SOAP Encoding*).

- `literal` väärtus tähistab, et sõnumi osade edastamisel ei kasutata spetsiaalset kodeeringut, vaid esitusviis vastab üks-ühele *WSDL* dokumendis antud *XML Schema* elementide ja tüüpide kirjeldustele.

Stiili nimetus ja kodeeringu kasutamise info annavad kokku üldkasutatava tunnuse sõnumiedastuse formaadile. *X-tee* protokolli versioon 2.0 kasutab *RPC/Encoded* vormingut. Versioonis 3.1 on kasutusel *Document/Literal wrapped*. Järelliides *wrapped* tähistab kokkuleppelist kitsendust *Document/Literal* vormingule, kus teenusekirjelduses

esitatakse täpselt 1 sõnumi osa, mis sarnaselt *RPC* stiilile koondab sõnumi sisu ainsa ühise elemendi alla.

### 1.1.4 Manuseid sisaldavad sõnumid

*SOAP* protokolli kasutavad X-tee teenuse päringud võivad sisaldada ka manuseid. Manustega sõnumi edastamiseks kasutatakse *MIME Multipart/Related* edastusviisi. Kui tavalisel kujul koosneb päringu sisu ainult edastatavast *XML* sõnumist, siis manusega sõnumi puhul jaguneb päringu sisu mitmeks erinevaks osaks, millest üks on *XML* sõnum ja ülejäänud osad tähistavad eraldi manuseid, mis võivad edastada erinevat sisu.

Sellised päringud peavad olema vastavuses *MIME* spetsifikatsiooniga [8] ning täiendavalt rakendatakse neile *X-tee* protokolli poolt järgmiseid piiranguid [2]:

- *MIME* konteineri välja *Content-Type* parameetri type väärtuseks on `text/xml` või `application/xop+xml`.
- *MIME* konteineri välja *Content-Type* parameetri `boundary` väärtuseks on *MIME* kodeeritud teate elementide eraldaja.
- *MIME* konteineri esimene osa on alati *SOAP* ümbrik (*Envelope*), mis sisaldab päist ja keha.
- *SOAP* ümbriku osa kodeeringu (*Content-Transfer-Encoding*) parameetri väärtuseks on `8bit`.
- *SOAP* ümbrik sisaldab viiteid kõikidele manustele, kasutades selleks manuse päise välja *Content-Id* väärtust.
- Teenus esitatakse *MIME* konteinerina, kui teenuse kirjelduses on kirjeldatud *MIME* sõnumi sidumise viis (*binding*).
- Kui teenus esitatakse *MIME* konteinerina, siis manustena saadetakse parajasti kõik teenuse sisendis olevad skalaarsed elemendid, mille tüüp on `xsd:base64Binary` või `xsd:hexBinary`.
- Kui teenuse sisend sisaldab *MIME* manust, siis väljundis olevas teenuse osas esitatakse edastatud sisendparameetri väärtusena vastuvõetud manuse *SHA-512* räsi.

### 1.1.5 Teenuste kasutamine

Teenuste kasutamine käib *X-tee* andmevahetuskihil läbi turvaserveri. Selleks saadetakse sinna *WSDL* kirjelduse tingimustele vastav kokkupandud sõnum. Turvaserver eraldab *X-tee* sõnumi *SOAP* päisest vajaliku info teenust pakkuva andmekogu kohta ning edastab sõnumi (vajadusel teise turvaserveri vahendusel) õigele vastuvõtjale. Päringu vastu võtnud infosüsteem teenindab sõnumi ning tagastab turvaserverile eduka töö tulemusena *WSDL*-is määratud päringu vastuse või veateate. Turvaserver edastab saadud vastuse algsele teenuse väljakutsujale.

Klientsüsteemi jaoks on turvaserver vahendaja rollis, kes teab millistele serveritele saadetud päringud edastada. Sellest tulenevalt ei pea arendamise hetkel testimise eesmärgil turvaserverit kasutama, sest samade päringutega on võimalik infosüsteemi serveriga ka vahetult suhelda.

### 1.1.6 Üldine arendaja tööprotsess *X-tee* teenuse kasutuselevõtmisel

Lihtsustatud vormis sisaldab uue *X-tee* teenuse kasutuselevõtt administreerivat ja tehnilist osa. Administreeriv osa tähendab, et uuele teenusele tuleb *X-tee* kaudu saada vajalikud juurdepääsud, ehk lubada vastavale asutusele konkreetse teenuse kasutamine. Tehniline osa kujutab endast teenusekirjelduse alusel sobiva liidese loomist ning selle integreerimist enda arendatavasse süsteemi. Liidese loomisel on erinevaid variante:

- **Liidese käsitsi loomine**
  Arendaja loob ise vajalikud klassid või kirjutab käsitsi kogu koodi, mis paneb kokku teenuse kasutamise jaoks vajaliku päringu. Peamiseks probleemides on see, et mahukate päringute kokkupanemine võtab kaua aega ning on tõenäoline teha inimlikke vigu enne, kui töötava rakenduseni jõutakse. Samuti on taolises olukorras võimalik, et kood koostatakse näidispäringu eeskujul, mis ei pruugi katta ega demonstreerida kõiki kasutatava teenuse võimalusi.

- **Süsteemi poolt pakutavate töövahendite kasutamine**
  Tarkvaraplatvormid pakuvad sageli ise tööriistu, mis suudavad *WSDL* kirjelduste alusel teenuse liidese genereerida. *.NET* platvormil saab kasutada *Visual Studio*-ga kaasa tulevaid rakendusi `xsd.exe`, `wsdl.exe` ja `svcutil.exe`. Probleemiks sellise lahenduse kasutamisel on see, et nimetatud tööriistad ei oska automaatselt arvestada X-tee eripäradega ning nende töölesaamiseks on vaja programme seadistada ja täiendada. Lisaks on konkreetsed programmid seotud

arendusvahendiga *Visual Studio*, mis piirab kasutamist ainult *Windows* platvormi kasuks.

- **Kolmanda osapoole poolt loodud tööriistade kasutamine**
  Käesoleva töö kirjutamise hetkeks loodud rakendusi, mis on mõeldud spetsiaalselt *X-tee* teenuste kasutamise lihtsustamiseks. Aadressil `https://www.ria.ee/x-tee-tarkvara/` on välja toodud 2 lahendust, mis pakuvad võimalusi *X-tee* teenuste kasutuselevõtmiseks:

  1. *.NET* platvormil on kasutatav ***X-tee.NET*** (`http://xtee.codeplex.com`), mis on võimeline etteantud *WSDL*-ist koodi genereerima, et seda hiljem olemasolevatesse projektidesse lisada. Töö kirjutamise hetkel töötas rakendus ainult *Windows*-i keskkonnas, *Linux*-i keskkonnas *Mono* platvormil kasutamine ebaõnnestus.

  2. *Java* platvormile mõeldud ***J-road*** (`https://github.com/nortal/j-road`), mis on analoogne toode *X-tee.NET*-iga, sisaldades ka eelgenereeritult suurt osa Eesti e-teenustest. Uute teenuste jaoks on võimalik kasutada generaatorit.

  Generaatorite kasutamise teeb tülikaks see, et tuleb läbida mitmeid samme, enne kui liides töövalmis on. Esmalt tuleb koodi genereerimiseks kasutada arendusvahendi välist eraldiseisvat programmi. Seejärel on vaja genereeritud kood käsitsi integreerida olemasolevasse rakendusse, lahendades selle käigus võimalikud konfliktid (nt. kood, mida on eelmisest korrast kohandatud teatud vajadustest lähtuvalt).

  Lisaks on generaatoreid ebaotstarbekas kasutada teenuse ühekordseks katsetamiseks, nt. teenuse käitumise proovimisel või demonstratsiooni tegemise eesmärgil. See tegevus eeldab ikkagi teatud ettevalmistamist genereerimise ja keskkonna ülesseadistamise näol.

## 1.2 *F#* programmeerimiskeel

*F#* on algselt *Microsofti* ja *Microsoft Research* poolt *.NET* arendusplatvormile loodud programmeerimiskeel. Esimestes versioonides oli keele arendusprotsess suletud, kuid versioonis 2.0 muudeti projekt avalikuks. Peale avalikustamist on *F#* programmeerimiskeel avatud lähtekoodiga, mille arendamist juhivad *F# Software Foundation* (`http://fsharp.org/`), *Microsoft* ja avatud lähtekoodi panustavad

arendajad. Lähtekoodi avamine muutis võimalikuks ka *F#* programmeerimiskeele kättesaadavakstegemise *Mono* tarkvaraplatvormi baasil teistel mitte-*Windows* operatsioonisüsteemidel. [9]

*F#* põlvneb funktsionaalsest programmeerimiskeelest *ML* ning omab täiendavalt mõjutusi *OCaml*, *C#*, *Python*, *Haskell*, *Scala* ja *Erlang* keeltest [9]. Viimane stabiilne versioon *F#* keelest on 3.1.2; järgmine suurem versiooniuuendus on 4.0, mis antakse välja koos *Visual Studio 2015* versiooniga (avaldamine planeeritud 2015. aasta teises pooles).

*F#* on üldise otstarbega programmeerimiskeel, mida kasutatakse mitmetes valdkondades: ärilettevõtete tarkvaraarendus (*Enterprise Programming*), andmeteadus (*Data Science*), veebiarendus (*Web Development*), mobiilsete rakenduste ja mängude arendus (*Mobile Apps and Games*), masinõpe (*Machine Learning*), pilvesüsteemide arendus (*Cloud Programming*), majandusarvutus (*Financial Computing*), matemaatika ja statistika ning andmepöördus (*Data Access*) [10]. [11]

*F#* programmeerimiskeel ühendab funktsionaalsele programmeerimisstiilile iseloomulikku sisutiheda, väljendusrikka ja kombineeritava stiili *.NET* platvormi virtuaalmasina, teekide, koostalitusvõime ja objektide mudeliga. [12]

*F#* programmeerimiskeele omadused:

- Range tüübisüsteem, mis välistab suure osa tüüpide valesti kasutamisest tingitud vigadest.

- Minimalistlik süntaks ja automaatne tüübituvastus tähendavad, et kood on enamasti kordades lühem analoogsest koodist objekt-orienteeritud lähenemisega *C#* keeles. Süntaksit võib kohati võrrelda pseudokoodiga ning tänu automaatsele tüübituvastussüsteemile on kood võrreldav dünaamiliste andmetüüpidega programmeerimiskeeltega (skriptimiskeeltega).

- Rõhutatult funktsionaalsele stiilile suunatud lähenemine teeb koodi ülevaatlikumaks, täpsemaks ning vähem altimaks juhuslike vigade tekkimisele.

- Mitmete programmeerimisparadigmade (imperatiivne, objekt-orienteeritud, funktsionaalne) toetus võimaldab valida lahenduse just konkreetse probleemi olemusest lähtuvalt.

- *F#* on *.NET* platvormil kompileeritav keel, kuid võimaldab ka skriptide kirjutamist, mis tähendab, et kogu *F#* keele funktsionaalsust on võimalik läbi proovida

käsureainterpretaatoriga. Sellest tulenevalt saab arendatavat koodi läbi proovida ilma kompileerimata ning skriptitud käsklused on võimalik hiljem ümber vormistada automaattestideks.

### 1.2.1 Arendusvahendid

*F#* on toetatud mitmete integreeritud programmeerimiskeskkondade [10] poolt:

- ***Visual Studio***
  Ajaloost tulenevalt on *Visual Studio* üks esimesi *F#* keele tuge pakkuv arendusvahend. Alates *Visual Studio 2010* versioonist on *F#* keel programmiga kohe kaasas olevate arendusvahendite hulgas.

  *F# Software Foundation* poolt on loodud *Visual Studio*-le pistikprogramm *Visual F# Power Tools*, mis laiendab arendustegevust täiendavate visuaalsete abivahenditega, refaktoriseerimise- ja koodigenereerimise võimalustega.

  *Visual Studio* on töö kirjutamise hetkel saadaval ainult *Windows* operatsioonisüsteemile.

- ***MonoDevelop***
  Avatud lähtekoodiga programmeerimiskeskkond *Mono* platvormile, mis töötab erinevates operatsioonisüsteemides.

  *F# Software Foundation* poolt on loodud avatud lähtekoodiga arendusvahenditele, sh. *MonoDevelop* jaoks pistikprogramm *fsharpbindings*, mis võimaldab *F#* keele toetust nendes programmides.

- ***Xamarin Studio***
  *MonoDevelop*-i baasilt loodud arenduskeskkond, mis võimaldab *.NET/Mono* programmeerimiskeeli kasutada mobiilsetel platvormidel. Toetab *F#* keelt alates 3.0 versioonist.

Kuna *F#* keel pakub süsteemset tuge kompilaatori kasutamiseks ja keelekonstruktsioonide töötlemiseks, on see soodustanud programmeerimiskeele toetuse lisandumist mitmete koodiredaktorite juurde:

- ***Tsunami***
  Tekstiredaktor, mis töötab *Windows* operatsioonisüsteemis. Pakub korralikku *F#*

toetust koodi fragmentide, funktsioonide, meetodite ja atribuutide pakkumisega ning toetab ka *F#* spetsiifiliste funktsionaalsuse kasutamist otse redaktoris.

- ***Atom***
  Toetab *F#* keelt läbi *FSharp.Atom* nimelise laienduse. Võimaldab põhilisi kaasaegseid arendustöid soodustavaid lahendusi: koodipakkumine (*code completion*), visuaalne eristamine, dokumentatsiooni kuvamine hüpikakendes. Lisaks on suuteline avama ja kasutama *F#* projekte.

  *Atom* koodiredaktor ise on kasutatav erinevates operatsioonisüsteemides.

## 1.3 *F#* keele tüübipakkujad

Tüübipakkuja (*type provider*) mehhanism on programmeerimisvahend, mis lisandus *F#* keelele versioonis 3.0. Tüübipakkuja on kompileerimise ajal kasutatav komponent, mis etteantud staatiliste parameetrite põhjal identifitseerib välise inforuumi ja sellele juurdepääsu viisi. Tüübipakkuja võimaldab *F#* kompilaatorile ja tööriistadele kahte peamist asja [13]:

- **pakutava komponendi signatuur**
  Käitub programmeerimisliidesena valitud inforuumile, mis koostatakse nõudmise peale vastavalt *F#* kompilaatori vajadustele. *F#* keele jaoks sisaldab komponendi signatuur pakutavaid koodi nimeruume, tüüpe, meetodeid, klassimuutujaid, atribuute ja konstante, mis koos moodustavad *.NET* tarkvaraplatvormile omase objekt-orienteeritud kirjelduse valitud inforuumile.

- **pakutava komponendi realisatsioon**
  Antud osa võib tuleneda reaalsest *.NET* teegist, mis realiseerib komponendi signatuuri - genereeritud tüüpide mudel (*generative model for the provided types*). Alternatiivina võib realisatsioon koosneda ka kustutusfunktsioonidest (*erasure functions*), mis kujutisena teisendavad pakutavaid tüüpe ja meetodeid reaalseteks tüüpideks ja avaldisteks, nn. kaduvate tüüpide mudel (*erasure model for the provided types*).

Lihtsustatud kujul võimaldavad *F#* keele tüübipakkujad ehitada *F#* kompilaatori jaoks pistikprogramme, mis genereerivad välise informatsiooni alusel uute andmetüüpide definitsioone ning seovad need vastava realisatsiooniga. Tüübipakkuja kaudu võib lugeda lokaalse andmebaasi struktuuri ning selle põhjal genereerida andmebaasitabelitele vastavad andmetüübid. Samuti võib tüübipakkuja analüüsida

teiste programmeerimiskeelte koodi, nt. *JavaScript* või *MATLAB* ning genereerida andmetüüpe vastava koodiga töötamiseks. [14]

Realisatsiooni aluseks valitud mudelist tulenevalt saab tüübipakkujad jagada kahte kategooriasse:

- kaduvate andmetüüpidega (*erased types*) tüübipakkujad
- genereeritud andmetüüpidega (*generated types*) tüübipakkujad

### 1.3.1 Kaduvate andmetüüpidega tüübipakkujad

Kaduvate andmetüübidega tüübipakkuja teeb iseäralikuks see, et arendajale arendusvahendis pakutavad andmetüübid ja väljad on ajutise iseloomuga. Pakutavad andmetüübid eksisteerivad ainult kompilaatorile pakutavas komponendi signatuuris, mis tähendab, et arendusvahend oskab neid välja näidata tavaliste tüüpidena. Kompileerimisel teisendatakse need andmetüübid tüübipakkujas realiseeritud kustutusfunktsioonidega teisteks, üldisemateks tüüpideks, ning loodavasse teeki algselt pakutud andmetüüpidele viidet ei jää.

Joonisel 3 (lk. 23) on kujutatud teoreetilist kaduvate tüüpidega tüübipakkujat, mis pakub signatuuri andmetüübile `Isik`, millega on seotud kaks meetodit `AnnaNimi()` ja `SätiNimi`. Arendusvahendis saab `Isik` tüüpi kasutada tavalise *.NET* klassina, mida eelpool nimetatud arendusvahendid oskavad tõlgendada ja teha koodipakkumisi nendele meetoditele. Võimalik tüübipakkujat kasutav kood on esitatud koodinäidises 1.

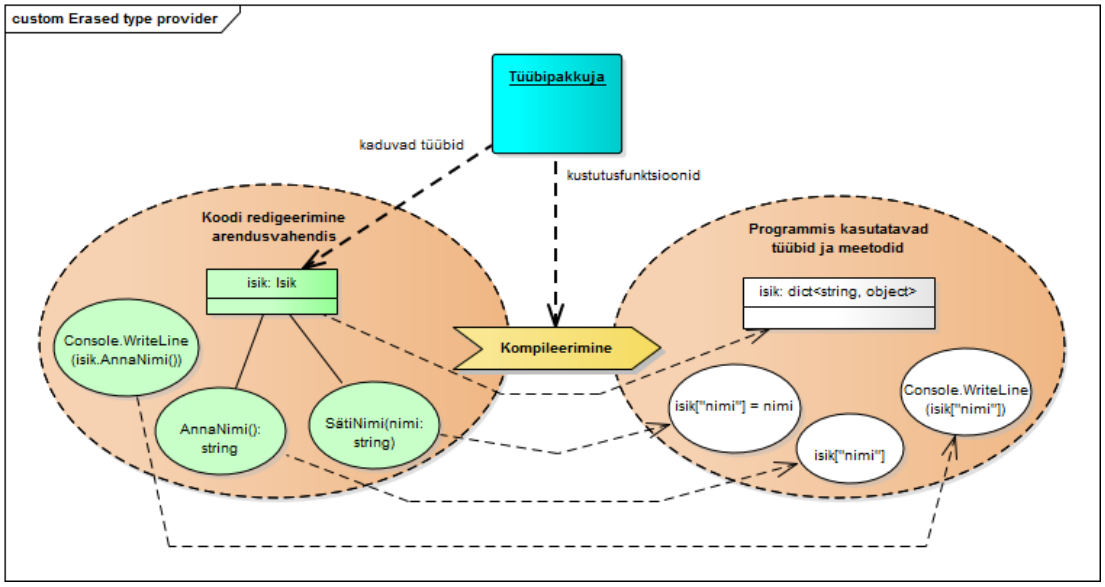
```
type TP = MyTypeProvider<"arg1", "arg2">

let isik = new TP.Isik()
isik.SätiNimi("Ants")

Console.WriteLine(isik.AnnaNimi())
```

Koodinäidis 1: Kaduvate tüüpidega tüübipakkuja kasutamine

Koodi kompileerimisel rakendab *F#* kompilaator antud tüüpidele tüübipakkuja poolt defineeritud kustutusfunktsioone. Oletame, et antud näite puhul on reaalne tüüp, mida signatuuri realiseerimisel kasutatakse, `System.Collections.Generic` nimeruumis defineeritud `Dictionary`, millega seatakse vastavusse andmetüübi poolt defineeritavad väljad ja nende väärtused. Sellise ülesehitusega genereeriks kompilaator




Joonis 3: Kaduvate tüüpidega tüübipakkuja tööpõhimõte

programmi, mis oleks ilma tüübipakkujat kasutamata võrdväärne koodinäidises 2 kujutatud koodiga.

```
let isik = new Dictionary<string,obj>()
isik.["nimi"] <- box "Ants"

Console.WriteLine(isik.["nimi"])
```

Koodinäidis 2: Kaduvate tüüpide kompileerimisel tekkiv kood

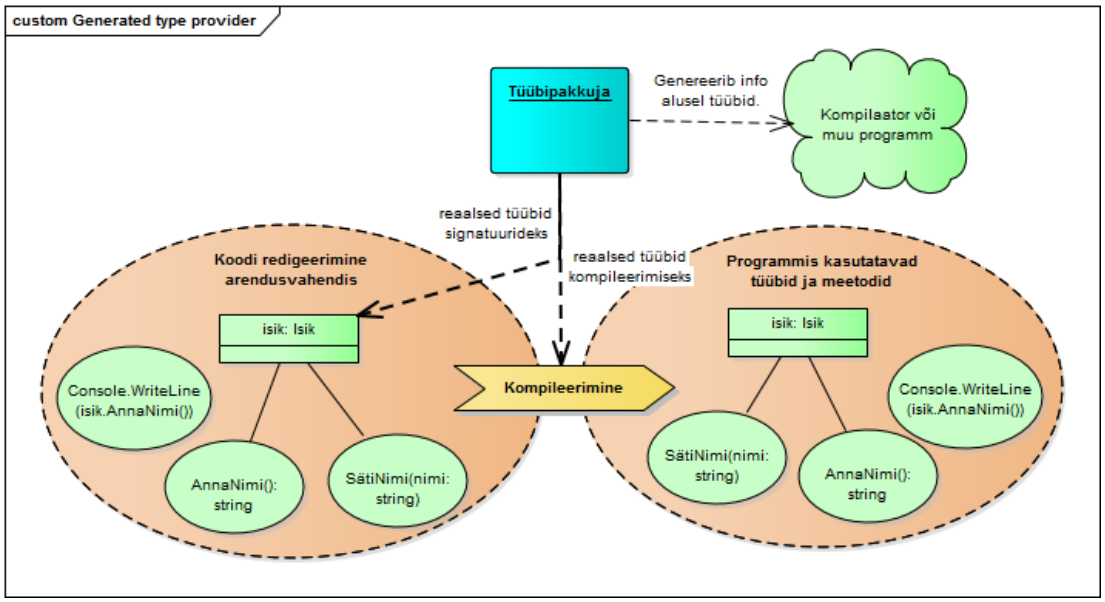
Kuna kaduvate andmetüüpide korral satuvad lõpprakendusse üldisemad tüübid, kui need, mille vastu arendati, siis on selliste tüübipakkujate puhul raskendatud objekt-orienteeritud programmeerimise praktikatest tuntud konstruktsioonide esitamine. Näiteks kaovad kompileerimisel pärinevusseosed kaduvate tüüpide vahel. Samuti ei ole kaduvate tüüpide kasutamisel loodud teegid taaskasutatavad teistes *.NET* platvormi programmeerimiskeeltes, kuna reaalselt kasutatavaid lisaväärtusega andmetüüpe ei teki.

### 1.3.2 Genereeritud andmetüüpidega tüübipakkujad

Genereeritud andmetüüpidega tüübipakkujad tekitavad inforuumist pärit info alusel reaalsed tüübid, mis lõpptulemusena kompileeritakse ka tüübipakkujat kasutava rakenduse baitkoodi. Reeglina põhineb antud tüübipakkuja korral komponendi signatuur ja realisatsioon täpselt samal funktsionaalsusel: andmete alusel ehitatakse ajutine teek, mis hoiab vajalikke tüüpe ja meetodeid, mida on vaja inforuumiga

suhtlemisel. Kompileerimisel kopeeritakse ajutisest teegist kõik genereeritud objektid sihtrakendusse, mille tulemusena (erinevalt kaduvatest andmetüüpidest) on kõik kasutatud tüübid reaalselt ehitatud rakenduses olemas.

Andmetüüpide ja meetodite loomisel võib genereeritud tüübipakkuja välja kutsuda mõnda kolmanda osapoole programmi, mis vajaliku koodi kokku paneb ning tüübipakkuja saab kasutada selle väljundit signatuuride komponendis ja kompileerimisel. Genereeritud tüüpidega tüübipakkuja tööpõhimõte on kujutatud joonisel 4.




Joonis 4: Genereeritud tüüpidega tüübipakkuja tööpõhimõte

Kuna genereeritud andmetüüpide puhul nende info ei kao, siis võimaldab see kasutada levinud programmeerimispraktikaid, nt. klasside pärilust. Samas on nad mahukamad, sest lõpprakendusse lisatakse kõik genereeritud tüübid, ka need, mida rakendus reaalselt ei kasuta. Kuna tüübipakkuja poolt tekitatud genereeritud andmetüübid on reaalselt eksisteerivad *.NET* klassid, siis on genereeritud andmetüüpe sisaldavad teegid kasutatavad ka teistes arendusplatvormi keeltes, nt. *C#*-s ehitatavates rakendustes.

### 1.3.3 Tüübipakkuja ülesehitus

Tüübipakkuja realiseerimine kõige madalamal tasemel kujutab endast eraldi teegi loomist, mis defineerib tüübipakkuja funktsionaalsust realiseeriva klassi. Selleks, et *F#* kompilaator suudaks edukalt tüübipakkuja funktsionaalsust kasutada, peab tüübipakkuja klass olema tähistatud `Microsoft.FSharp.Core.CompilerServices` nimeruumis defineeritud `TypeProviderAttribute` atribuudiga. Lisaks peab

tüübipakkuja klass realiseerima funktsionaalsuse samas nimeruumise defineeritud `ITypeProvider` liidesele. Nimetatud liides kirjeldab vajalikud meetodid, mida kompilaator kasutab genereeritavate tüüpide kohta info pärimiseks.

Kuna tüübipakkujate rakendusliides on keerukas ja ei paku enda funktsionaalsuse realiseerimise jaoks abistavat raamistikku, siis on *F# Software Foundation* poolt loodud tüübipakkujate arendamise lihtsustamiseks lähteprojekt (*F# Type Provider Start Pack*) (`https://github.com/fsprojects/FSharp.TypeProviders.StarterPack`).
Nimetatud projekt sisaldab klasse ja meetodeid, mis pakuvad enamlevinud konstruktsioone ja funktsionaalsust, mida tüübipakkuja arendamisel vaja võib minna. Suur rõhk projektis on kaduvate tüüpide toetamisel, kuid teatud osa on kasutatav ka genereeritud tüüpide jaoks.

Vaikimisi ei oska F# kompilaator suvalisest teegist tüübipakkuja funktsionaalsust otsida. Vastav teek tuleb tähistada eelmainitud nimeruumis defineeritud teegi-taseme atribuudiga `TypeProviderAssemblyAttribute`. Selle olemasolu korral oskab kompilaator arvestada, et tegu on teegiga, mis sisaldab tüübipakkujate kirjeldusi.

# 2 *X-tee* tüübipakkuja realiseerimine

## 2.1 Eesmärgid ja nendest tingitud piirangud

Antud töö praktiliseks eesmärgiks on tõestada andmevahetuskihi *X-tee* tüübipakkuja tehnilist teostatavust, et konkreetsete näidetega oleks võimalik demonstreerida lahenduse otstarbekust ja potentsiaali. Töö tulemusena ei valmi täisfunktsionaalne rakendusteek, vaid prototüüp, mis on suuteline toime tulema valitud teenusepakkujate WSDL kirjeldustega. Konkreetsemad eesmärgid loodavale lahendusele on järgmised:

- Luua töötav tüübipakkuja prototüüp, mis on võimeline lihtsamatele *WSDL* dokumentidele andmetüüpe genereerima ning operatsioonikirjelduste alusel looma liidesed teenuste kasutamiseks.
- Tüübipakkuja peab olema võimeline töötama valitud *WSDL*-idega.
- Tüübipakkuja ei pea täielikus mahus realiseerima *WSDL* ja *XML Schema* spetsifikatsioone, vaid ainult seda osa, mis on tarvilik valitud *WSDL*-ide töötlemiseks.
- Tüübipakkuja peab suutma genereerida liidesed *X-tee* protokolli versioonidele 2.0 ja 3.1.
- Realiseeritakse võimalus *MIME/Multipart* sõnumite vahetamiseks.

Pikemas perspektiivis (käesoleva töö edasiarendustena) peaks realisatsioon vastama järgmistele tingimustele:

- Tüübipakkuja peab vastama *X-tee* arendusjuhendis kirjeldatud tingimustele. [2]
- Tüübipakkuja peab olema suuteline töötlema levinumaid *XML Schema* [5] elemente ja looma nende põhjal üheseltmõistetavaid ja liiasusteta andmetüüpe.
- Tüübipakkuja peab olema lihtsalt kasutatav teistes *.NET* platvormi keeltes (nt. *C#*) kirjutatud projektides.
- Minimeerida lõpprakenduse sõltuvust teistest teekidest.

- Võimalikult lühike reaktsiooniaeg tüübipakkuja funktsionaalsuse kasutamisel arendamise käigus.

- Lubada läbi seadistamise kohandada genereeruvaid andmetüüpe arendaja eelistuste alusel.

- Võimaldada päringute logimist arendaja poolt etteantud viisil.

## 2.2 Tüübipakkuja liigi valik

Sobiva tüübipakkuja liigi valimiseks on vajalik kaduvate ja genereeritud tüüpidega tüübipakkujate omavaheline võrdlus. Kaduvate tüüpidega tüübipakkuja eelised võrreldes genereeritud tüüpidega on järgmised:

- Kaduvate tüüpidega tüübipakkuja puhul on võimalik kasutada vajadusepõhist realisatsiooni (laisk laadimine e. *lazy loading*). Tüübipakkuja ei pea kohe tekitama kõiki *WSDL* dokumendis kirjeldatud andmetüüpe, vaid saab neid juurde laadida jooksvalt, kui arendaja neid kasutada soovib. Genereeritud tüüpidega tüübipakkuja puhul tuleb tekitada korraga andmetüübid kõikidele *WSDL* dokumendis kirjeldatud olemitele, kuna tüübipakkuja opereerib reaalsete tüüpidega, mille tekitamisel ei ole võimalik ette teada, milliseid tüüpe arendaja kasutama plaanib hakata.

- Eelmisest punktist lähtuvalt omab kaduvate tüüpidega tüübipakkuja kiiruse eelist väga suurte *WSDL* dokumentide puhul, kuna vajadusepõhine lähenemine ei nõua kogu lähteinfo korraga töötlemist.

Peamised takistused kaduvate tüüpidega tüübipakkuja puhul on järgmised:

- Kaduvate tüüpidega tüübipakkuja kasutamine teiste *.NET* platvormi keeltes on keerukam, kuna tekkinud tüübid on *F#* keele spetsiifilised ning neid ei saa otse kasutada. Tüübipakkuja funktsionaalsuse kasutamine on võimalik ainult kaudselt ning selleks tuleks *F#* projektis ehitada vaheliides, mis sobituks *.NET* platvormi üldiste tüüpidega.

- Kaduvate tüüpidega tüübipakkuja muudab keerukamaks pärinevusseoste haldamise. Pärinevusseosed on teada ainult koodi kirjutamise ajal ning kompileeritud koodis ei ole loomulikul viisil võimalik baastüüpide korral konkreetseid tüüpe tuvastada. Võimalik on lisada meetodid, mis tüübiteisendusi

teevad, kuid see läheks lahku tavalistest keelekonstruktsioonidest tüüpide tuvastamisel.

Väljatoodud argumentidest lähtudes ja neid eesmärkidega võrreldes on antud töö kontekstis eelistatud genereeritud tüüpidega tüübipakkuja realiseerimine. Vajadusepõhist lähemist on võimalik mööndustega rakendada ka genereeritud tüüpide jaoks läbi tüübipakkuja parameetrite, kus kasutaja saab ette anda konkreetsed teenused, mida *WSDL* dokumendist välja lugeda.

Selleks, et genereeritavad andmetüübid oleks kasutatavad võimalikult suurel arvul *.NET* platvormi programmeerimiskeeltes, genereeritakse andmetüübid sisemiselt *C#* keeles, kasutades üldiselt levinud ja toetatud koodikonstruktsioone.

## 2.3 Genereeritava koodi planeerimine

### 2.3.1 Pakutavate andmetüüpide paigutus

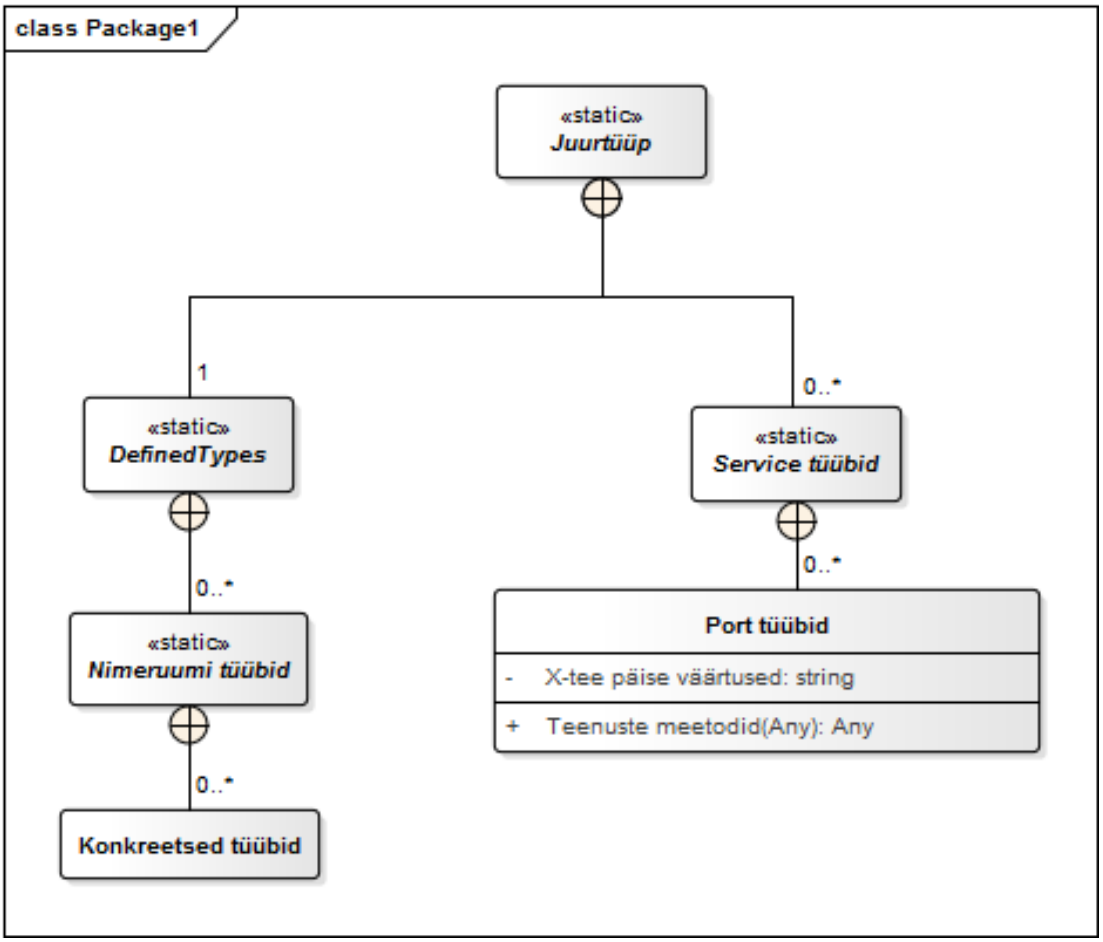
Enne tüübipakkuja arendamisega alustamist tuleb paika panna genereeritava koodi eeldatav struktuur.

Tüübipakkujate omapärast lähtudes on juba eelnevalt piiratud, et kogu genereeritud kood hakkab olema ühes juurtüübis, mis vastab konkreetsete parameetrite väärtusega tüübipakkujale. Selle juurtüübi sisemine ülesehitus on aga vabalt valitav.

*WSDL* jaguneb põhimõtteliselt teenuste ja nende poolt kasutatavate andmetüüpide defineerimiseks. Sama põhimõte peegeldub ka tüübipakkuja poolt genereeritud koodi struktuuris. Andmetüüpide hoidmiseks genereeritaks eraldi tüüp `DefinedTypes`, mille sees on defineerivate *XML* nimeruumide kaupa jaotatud andmetüübid.

Juurtüübi alla lisatakse ka eraldi andmetüüp kõikide *WSDL*-is kirjeldatud `service` elementide jaoks, kasutades defineeritud nime. Selle andmetüübi alamtüüpideks on kõik `port` elementide alusel loodavad andmetüübid, mis omakorda saavad kõikide operatsioonide jaoks vastavad meetodid. `Port` tüübid sisaldavad lisaks välju kõigi *X-tee* standardpäises kasutatavate elementide jaoks.

Genereeritava andmetüüpide struktuuri põhimõtteline skeem on esitatud joonisel 5 (lk. 29).




Joonis 5: Genereeritud andmetüüpide ülesehitus

### 2.3.2 Tegevused teenuse väljakutsumisel

Teine põhikomponent genereeritava koodi juures on teenuste väljakutsumise loogika. Üldine põhimõte on kirjeldatav pseudokoodina koodinäidises 3.

```
ühenduse avamine ja kanali loomine teenusepakkujaga.
kui tegu on multipart päringuga:
    MIME konteineri päise kirjutamine XML sõnumi jaoks.
XML sõnumi kirjutamine kanalisse:
    SOAP päise kirjutamine koos X-tee päise väärtustega.
    SOAP keha kirjutamine:
        iga päringu parameetri serialiseerimine.
kui tegu on multipart päringuga:
    iga lisatud manuse lisamine MIME konteinerina.
vastusvoost päringu tulemuse lugemine.
kui tegu on multipart vastusega:
    loe kõik lisatud MIME manused rakenduse mällu.
vastuse XML sõnumi lugemine:
    SOAP keha lugemine:
        iga vastuse parameetri deserialiseerimine objektiks.
```

```
ühenduse sulgemine.
vastuseks saadud objektide tagastamine.
```

Koodinäidis 3: Teenuse väljakutsel teostatavad tegevused

Koodinäidises 3 väljatoodud funktsionaalsuse saab jagada kaheks: üldine osa, mis tuleb läbi teha iga teenuse väljakutse puhul ühtemoodi, ning varieeruv osa, mis sõltub konkreetsest teenusest. Teenusespetsiifiline osa on antud juhul *SOAP* päise ja keha kirjutamine ning samuti *SOAP* kehast vastuse lugemine. Ülejäänud funktsionaalsus hakkab paiknema teenuste ühises baasklassis, mis omakorda teenusespetsiifilist koodi õiges kohas välja kutsub.

### 2.3.3 Sõnumites edastatavate objektide serialiseerimine

Sõnumites edastatavate objektide serialiseerimiseks (ja deserialiseerimiseks) on välja pakkuda mitmeid variante. Antud töös võeti vaatluse alla järgmised alternatiivid:

- **Serialiseerimise realiseerimine rakendusteegis**
  Rakendusteegis serialiseerimise realiseerimine tähendaks seda, et loodav lahendus tegeleb objektide *XML* formaati teisendamisega iseseisvalt. Variandi eeliseks on, et see annab suurema paindlikkuse tüüpide loomisel, kuna loogikat saab vastavalt rakenduse vajadusele ümber teha. Probleemiks on suurem töömaht funktsionaalsuse loomisel.

- **Süsteemne komponent *XmlSerializer***
  Antud komponendi näol on tegu *.NET* platvormi poolt pakutava lahendusega andmetüüpide teisendamiseks *XML* formaati. *XmlSerializer*-i funktsionaalsus on kasutusel mitmes *.NET* tarkvaraplatvormi enda rakenduses, mis *WSDL*-i alusel teenuseliideseid genereerib ning sellega seoses on komponenti võimalik kasutada kõigi enamlevinud *XML* formaadi konstruktsioonide ehitamisel. Vaikimisi on *XmlSerializer* võimeline serialiseerima globaalse (*public*) juurdepääsuga tüüpe ning nende sama juurdepääsuga väljasid. Täiendavalt on serialiseerimisprotsessi võimalik juhtida vastavate atribuutide kasutamisega andmetüüpide defineerimise juures.

  Variandi eeliseks on, et selle kasutamine aitab kokku hoida aega prototüübi arendamisel ning andmetüüpide teisendamise saab esialgu kõrvale jätta. Probleemiks on, et komponent ei võimalda piisavalt paindlikku juurdepääsu serialiseerimisprotsessile, mis tähendab, et andmetüüpide struktuurile on komponendi poolt ette seatud teatud piirangud.

- **Süsteemne komponent *DataContractSerializer***
  Komponent sarnaneb *XmlSerializer*-ile, aga tegemist on uuema lahendusega. Toetatud on erinevad formaadid (nt. *XML*, *JSON*), serialiseerimisprotsess on kiirem ning nõuded andmetüüpide ülesehitusele on vähem piiravad.

  Variandi eelised on samad, mis *XmlSerializer*-i puhul, kuid suureks probleemiks on see, et kõik *XML* formaadi võimalused ei ole kasutatavad (nt. ei toetata elemendi atribuutide serialiseerimist).

- **Kolmanda osapoole komponendid**
  Antud variant jääb kõrvale, kuna läheb vastuollu seatud eesmärgiga, millega soovitakse minimeerida loodava lahenduse sõltuvust teistest teekidest.

Prototüübi ehitamiseks otsustati kasutada *XmlSerializer* komponenti. Valiku tegemisel sai määravaks arenduse eeldatav ajakulu ning *XML* formaadi laiapõhjaline toetus.

Objektide serialiseerimiseks on kasutada `System.Xml.Serialization` nimeruumis defineeritud klass `XmlSerializer`. Konkreetne `XmlSerializer` objekt on seotud kindla andmetüübiga, mis määratakse loomisel konstruktori argumendina. Täiendavate argumentidena on võimalik lisada detaile serialiseerimisprotsessi muutmiseks, kui klassi kirjeldus ei ole konkreetses olukorras sobiv.

Serialiseerimisprotsessi juhtimiseks on `System.Xml.Serialization` nimeruumis defineeritud mitmeid klassi ja välja taseme atribuute, mille kasutamine andmetüüpide defineerimisel võimaldab anda `XmlSerializer`-ile juhiseid, kuidas konkreetne andmetüüp *XML* formaadis esitada. Järgnevalt tuuakse välja antud töös olulisemad kasutatud atribuudid:

- `XmlRootAttribute` - kasutatakse *XML*-is juurelemendi nimetuse juhtimiseks. Vaikimisi on kasutusel andmetüübi enda nimi, kuid antud atribuudi kaudu saab eraldi määrata elemendi nime ja nimeruumi.

- `XmlTypeAttribute` - kasutatakse andmetüübi *XML*-tüübi määramiseks. Andmetüüpi kasutatakse juurelemendi nimena (kui eelnevalt nimetatud atribuut puudub) või olukorras, kus abstraktse tüübi kasutamine eeldab konkreetse tüübi välja toomist.

- `XmlElementAttribute` - kasutatakse väljade defineerimisel ning selle kasutamine sunnib välja esitama antud objekti alamelemendina. Võimaldab määrata alternatiivset nimetust (vaikimisi on kasutusel klassidefinitsioonis

esitatud välja nimi).

- `XmlAttributeAttribute` - kasutatakse väljade defineerimisel ning selle kasutamine sunnib välja esitama antud objekti alguselemendi atribuudina. Samamoodi on lubatud atribuudi nimetuse määramine.

- `XmlArrayAttribute` - kasutatakse massiiv-tüüpi väljade serialiseerimise juhtimiseks. Antud atribuut lubab defineerida massiivis antud väärtustele ühise elemendi, mille alla iga väärtus alamelemendina lisatakse.

- `XmlArrayItemAttribute` - kasutatakse massiivina esitatava välja ühe väärtuse alamelemendi omaduste kirjeldamiseks.

- `XmlIgnoreAttribute` - võimaldab teatud väljasid serialiseerimisest välja jätta. Antud atribuudi kasutamine tähendab, et konkreetset välja ei lisata loodud *XML* sõnumisse.

- `XmlTextAttribute` - vaikimisi esitatakse väljad alamelementidena, kuid antud atribuudi kasutamisel kirjutatakse välja sisu konkreetse objekti algus- ja lõpuelementide vahel tekstina.

### 2.3.4 *MIME* manustega päringute võimaldamine

*MIME* manustega päringute kuju erineb tavalisest päringust, mis koosneb ainult *XML* sõnumist. Manustega päringu puhul tuleb *XML* sõnum koos manustega lisada kindlas formaadis *MIME* konteineritesse, mis kokku moodustavad *MIME/Multipart* sõnumi. Kuna *MIME* manus võib olla erineva sisuga, sh. binaarfailid ja tekstifailid, siis on mõistlik nende esitlemiseks koodis kasutada võimalikult universaalset andmetüüpi. Antud töö jaoks on selleks tüübiks valitud `System.IO` nimeruumis defineeritud `Stream` klassi, mis võimaldab manusteks kasutada erinevaid sisendeid lokaalsete failide, mäluobjektide ja isegi võrguresursside näol.

*X-tee* spetsifikatsiooni järgi peab igale manusele olema viidatud mõne *XML* sõnumi *SOAP* ümbriku sisu elemendi poolt. Seega peab serialiseerimise protsess oskama arvestada manuste olemasoluga. Kasutatav süsteemne komponent *XmlSerializer* ei oska vaikimisi *MIME* manuseid *XML* elementidega siduda. Funktsionaalsuse võimaldamiseks genereerib tüübipakkuja manuseid sisaldavate väljade jaoks spetsiaalse andmetüübi `BinaryContent`, mille definitsioon on välja toodud koodinäidises 4 (lk. 33).

```
public class BinaryContent
{
    public BinaryContent()
    { }

    public BinaryContent(IDictionary<string, Stream> attachments, string id)
    {
        Id = string.Format("cid:{0}", id);
        Content = attachments[id];
    }

    [XmlIgnore()]
    public Stream Content { get; set; }

    [XmlAttribute("href", Form=XmlSchemaForm.Unqualified)]
    public string Id { get; set; }

    [XmlText()]
    public byte[] Value { get { ... } set { ... } }
}
```

Koodinäidis 4: *MIME* manuseid toetav andmetüüp

Koodinäidises on näha, et loodud tüübil 3 andmevälja: `Content`, kus hoitakse kasutaja poolt väärtustatud manuse sisu; `Id`, mis sisaldab manuse unikaalset tunnust `Content-Id` ning `Value`, mis tagastab `Content` välja sisu baidimassiivina. `Content` väli on *XmlSerializer*-i eest peidetud, kuna seda andmetüüpi ei oska komponent kasutada. Väli `Id` serialiseeritakse *XML* atribuudina nimega `href`. Välja `Value` väärtuse oskab serialiseerija teisendada `base64` kodeeringus tekstiks, mis vaikimisi edastakse elemendi sisuna.

Antud kuju ei sobi manuste edastamiseks, kuna andmed tuleks edastada eraldi *MIME* konteineris, mitte elemendi sisuna, kuid see esitus on vajalik, et sama välja oleks võimalik kasutada teenustes, mis ei eelda *MIME* päringute kasutamist.

*MIME* päringuga teenuste jaoks genereerib tüübipakkuja vastava teenuse koodi selliselt, et enne andmete serialiseerimist kohandatakse konkreetse andmetüübi käitumist. Koodinäidis 5 demonstreerib, kuidas see toimib:

```
XmlAttributeOverrides bodyOverrides = new XmlAttributeOverrides();
XmlAttributes bodyAttributes = new XmlAttributes();
bodyAttributes.XmlIgnore = true;
```

```
bodyOverrides.Add(typeof(BinaryContent), "Value", bodyAttributes);
```

Koodinäidis 5: Binaarse sisuga andmetüübilt sisu eemaldamine *XML* sõnumis

Antud kood lisab `Value` väljale `XmlIgnore` atribuudi, mis tähendab, et serialiseerija seda välja ei arvesta. Kohanduse tulemusena vastab serialiseeritud *XML* sõnum *MIME* sõnumi kujule.

*MIME* manused edastatakse teenusele lisaparameeter `attachments` kujul, mis on defineeritud `IDictionary<string,Stream>` tüübina ning seob vastavusse *MIME* manuse `Content-Id` ja manuse sisu. Iga *MIME* manus tuleb edastada nimetatud parameetri elemendina ning see peab omama vastava `Id` väärtusega `BinaryContent` tüüpi väärtust *XML* sõnumiks serialiseeritavate parameetrite või nende alamobjektide hulgas.

## 2.4 Tüübipakkuja arendamine

Loodava tüübipakkuja funktsionaalsuse saab jagada kolmeks peamiseks mooduliks:

- *WSDL* dokumendi lahtimõtestamine ja info vahetüüpidesse lugemine. Selle võib omakorda jagada kahe erineva, kuid väga seotud tehnoloogia kasutamiseks:
  - *WSDL* spetsifiilise info töötlemine (genereeritavad teenused);
  - *XML Schema* spetsiifilise info töötlemine (genereeritavad andmetüübid);
- Vahetüüpidesse salvestatud info alusel koodi genereerimine ja ajutise teegi kompileerimine.
- Tüübipakkuja liides, mis eelnevaid mooduleid välja kutsub ja juhib.

### 2.4.1 *WSDL* dokumendi töötlemine

*WSDL* dokumendi töötlemine on vajalik genereeritavate andmetüüpide ja teenuse meetodite kirjelduste väljaselgitamiseks. Kuna üheks eesmärgiks on lõpplahenduse minimaalne sõltuvus välistest komponentidest, siis on antud alamülesande lahendamiseks kaks varianti: kasutada *.NET* platvormi poolt pakutavat lahendust või realiseerida *WSDL* dokumendi lugemine käsitsi.

Esimene valik langes olemasoleva *.NET* funktsionaalsuse kasuks. *WSDL* dokumentide lugemiseks pakub *.NET* platvorm `ServiceDescription` andmetüüpi `System.Web.Services.Description` nimeruumist. Nimetatud klass oskab etteantud failist andmed sisse lugeda, järgides *WSDL* spetsifikatsiooni, ning loetud info salvestatakse antud tüübi vastavatel andmeväljadel.

Prototüübi väljatöötamise käigus selgus, et lahendus ei sobi, kuna see ei ole võimeline *WSDL* dokumendist välja lugema kõiki *X-tee* poolt kasutatavaid definitsioone. Konkreetne probleem tekib *MIME/Multipart* sõnumi kirjeldustega, kus alamelemendid kaotavad vajaliku info andmetüüpide kokkuviimiseks mujal dokumendis. Tulenevalt sellest, et probleem esineb varieeruval kujul *.NET* ja *Mono* platvormidel, siis ei ole mõistlik seda komponenti loodavas lahenduses kasutada ning *WSDL* dokumendi töötlemine tuleb täiendavalt juurde ehitada.

Lõplikult lahendati *WSDL* dokumendi lugemine kasutades *.NET* platvormi poolt pakutavat `XDocument` klassi `System.Xml.Linq` nimeruumist. *WSDL* dokumendi elemente läbi lugedes saadakse vahetüüpidesse teenusepakkuja poolt kirjeldatud teenuste ja andmetüüpide definitsioonid. `XDocument` klass toetab hästi funktsionaalset arendusstiili ning üldine põhimõte *XML* elementide lugemisel on järgmine:

- Iga huvipakkuva elemendi jaoks on loodud eraldi funktsioon, mis seda elementi töötleb. Elemendi definitsiooni alusel toimub selle töötlemine (definitsioonid on *WSDL* jaoks võetud *WSDL* spetsifikatsioonist [4] ja *XML Schema* jaoks nimeruumi kirjelduse failist [5] või *W3Schools* lehe *XML Schema* elementide indeksist [15]).

- Alamelementide lugemisel hoiatakse meeles hetkeolekut, et tuvastada lihtsamad vead definitsioonides või anda märku realiseerimata harudest. Kasutades mustrituvastamist (*pattern matching*), on võimalik elemente ükshaaval läbi käia ning uuendada konkreetsele elemendile vastava vahetüübi andmestikku.

- Alamelementide töötlemisel kutsutakse rekursiivselt välja konkreetsetele elementidele vastavaid töötlemisfunktsioone.

Näiteks *XML Schema* `simpleType` elemendi töötlemiseks kasutatav funktsioon on välja toodud koodinäidises 6:

```
/// Extracts `simpleType` element specification from schema definition.
let rec private parseSimpleType (node: XElement): SimpleTypeSpec =
    let content =
        node.Elements()
        |> Seq.fold (fun (state, spec) node ->
```

```
        match node, state with
        | Xsd "annotation", Begin ->
            Annotation, spec
        | Xsd "restriction", (Begin | Annotation) ->
            Content, Some(Restriction(parseSimpleTypeRestriction node))
        | Xsd "union", (Begin | Annotation) ->
            Content, Some(Union(parseUnion node))
        | Xsd "list", (Begin | Annotation) ->
            Content, node |> notImplementedIn "simpleType"
        | _ -> node |> notExpectedIn "simpleType"
        ) (Begin, None)
    |> snd
match content with
| Some content -> content
| _ -> failwith "Element simpleType is expected to contain either restriction
    , list or union element."
```

Koodinäidis 6: *XML Schema simpleType* elemendi lugemine

Konkreetse näite puhul vaadatakse ükshaaval järjest läbi antud elemendi `simpleType` alamelemendid. Lugemise algolekuks määratakse `Begin` ja loetud väärtuseks `None`. Igale elemendile rakendatakse mustrituvastust ning võrreldakse konkreetse elemendi nime ja selle elemendile lubatud olekuid. Kui ükski muster ei sobi, on tegu tundmatu elemendiga, mida ei tohiks kasutada, või on elementi kasutatud vale koha peal. Igale mustrile on ette nähtud eraldi alamülesanne: `annotation` elementi ignoreeritakse, ainult uuendatakse olekut; `restriction` ja `union` elementidele kutsutakse välja vastavaid lugemisfunktsioone; `list` element on antud hetkel realiseerimata ning kui tuvastatakse selle kasutamine konkreetses *WSDL* dokumendis, antakse vastav veateade.

Rakendades väljatoodud põhimõtet tervele *WSDL* dokumendile, on antud etapi tulemusena kogu vajalik info (sh. *X-tee* spetsiifilised laiendused) loetud vahetüüpidesse, mis on võimalik edasi anda koodigenereerimise moodulisse.

### 2.4.2 Koodigenereerimise tehnoloogia valik

*.NET* tarkvaraplatvormil on mitmeid erinevaid võimalusi koodi genereerimiseks. Järgnevalt tuuakse välja alternatiivid, mille kasutamist kaaluti antud töö tegemisel:

- **Koodi esitamine tekstilisel kujul**
  Tekstilisel kujul olev kood tähendab seda, et kood pannakse rakenduses kokku tekstitöötluse tulemusena, hoides erinevaid koodilõike teksti tüüpi muutujates

ning kombineerides neid vastavalt vajadusele.

Tekstilisel kujul olev kood eeldab kogu vaatluse all oleva koodilõigu tervikliku pildi nägemist. See tähendab, et koodi genereerimise hetkel peab kogu info olemas olema. Ilma täiendava infrastruktuuri loomiseta ei ole võimalik vastavalt vajadusele koodiosasid jooksvalt täiendada.

Antud töö raames tähendaks see seda, et andmetüüpide genereerimiseks tuleks ehitada täiendav andmekiht, mis hoiaks klasside ja meetodite terviklikku kuju vahetult enne koodi genereerimist.

- ***CodeDom***
  *CodeDom* tähistab *.NET* tarkvaraplatvormi `System.CodeDom` nimeruumis defineeritud klasside kasutamist. Selles nimeruumis defineeritakse võimalikud koodikonstruktsioonid ja tükid, mille abil saab programselt koodi kokku panna.

  *CodeDom* võimaluse kasutamine on arendamise osas mahukas, kuna kasutatavad klassid on pikkade nimedega ning keerulisemad lausekonstruktsioonid muutuvad koodis suureks ja hoomamatuks. Sellest tulenevalt kannatab koodi arusaadavus.

- **Tüübipakkujate lähtepakett**
  Lähtepakett pakub funktsionaalsust, mis sarnaselt *CodeDom*-ile võimaldab väiksematest tükkidest koodi komponeerida. Erinevalt *CodeDom*-ist on leitud tasakaal tükelduse ja arusaadavuse osas ning koodi kokkupanemine on ülevaatlikum.

  Probleemiks tüübipakkujate lähtepaketi kasutamisel on see, et genereeritud andmetüüpidega tüübipakkujate loomine on halvemini toetatud ja esialgsel katsetamisel tuli kulutada aega komponendi enda parandamisele.

- ***TypeBuilder***
  *TypeBuilder* on *.NET* platvormi poolt pakutav võimalus dünaamiliselt programmi töötamise ajal uusi tüüpe, meetodeid jms. ehitada ning kasutusele võtta. Komponent koosneb `System.Reflection.Emit` nimeruumis defineeritud klassidest, mille abil koodi ehitamine käib.

  Antud võimalus (mis on ka tüübipakkujate lähtepaketi poolt sisemiselt kasutatav) jääb hätta serialiseerimiseks vajalike atribuutide genereerimisega. Probleemiks on ringsõltuvus, mis tekib tüübi metainfo alusel väärtustatavate atribuutidega. Genereerimisel on andmetüüp ise veel ehitamata, kuid seda ei ole võimalik

ehitada, sest ta on sõltuvuses teisest andmetüübist, millele atribuuti määrata üritatakse.

Prototüübi kiire loomise huvides osutus kõige otstarbekamaks kasutada *CodeDom* ja tekstilisel kujul esitatava koodi kombinatsiooni. *CodeDom* on kasutusel selle koodiosa loomisel, mis on sõltuvuses tüübipakkuja argumentidest või funktsionaalsusest, mida ei ole võimalik ühe korraga luua. Tekstiliselt esitatakse need koodilõigud, mis ei sõltu muust loogikast, nt. abifunktsioonid serialiseerimisel ja sõnumitöötlusel.

*CodeDom* puuduseid on võimalik minimeerida, luues sobivaid abifunktsioone või tüübialiaseid, mis võtavad lühemalt kokku enamlevinud konstruktsioone või keerukamaid koodiosasid.

### 2.4.3 Andmetüüpide genereerimine

Andmetüüpide genereerimisel tuleb arvestada *WSDL* ja *XML Schema* tehnoloogiate eripäradega. Genereerimise seisukohast peab toime tulema asjaoluga, et andmetüübid saavad olla defineeritud globaalselt (neile saab viidata nimega) või lokaalselt (andmetüübi kirjeldus antakse elemendi enda juures). Lisaks on võimalik tüüpidele uusi nimesid anda, nende struktuuri laiendada või kitsendada ning kasutada erinevaid kombineerimise võimalusi. *WSDL* ja *XML Schema* lubavad ka teiste nimeruumide kirjeldustele viidata ning nende andmetüüpe kasutada.

Kuna *XML Schema* ei näe ette globaalsete tüüpide defineerimise järjekorda, siis ei ole võimalik andmetüüpe genereerida järgemööda ükshaaval. Tüüpide omavahelised seosed võivad olla rekursiivsed, millest tekib vajadus andmetüüpide vahemälus hoidmiseks ning sealjuures peab olema võimalus andmetüüpe jooksvalt täiendada (st. vahemälus olevad definitsioonid ei pea olema täielikud).

Antud lahenduses luuakse selle jaoks esmalt kõikide globaalsete tüüpide jaoks võtme järgi indekseeritavad loendid, kus võtmeks on tüübi nimeruumist ja nimetusest koosnev unikaalne tunnus ning väärtuseks vastav *WSDL* dokumendist loetud vahetüüp. Hilisem tüüpide genereerimine saab lõpliku koodi genereerimisel kasutada nimetatud loendeid vahetüüpide andmete omavaheliseks sidumiseks.

Analoogseid loendeid on samuti vaja genereeritud andmetüüpide jaoks, et jooksvalt oleks võimalik pidada arvestust juba loodud tüüpide kohta. Need luuakse samuti koos globaalsete tüüpide loenditega, kuid on algolekus tühjad. Antud loendeid täiendatakse kui koodi genereerimisel tekib konkreetse tüübi järele vajadus.

Lokaalselt defineeritud andmetüüpide puhul tuleb arvestada sellega, et konkreetne tüüp on oluline ainult kindlas kontekstis. Selliste tüüpide puhul on kasutusel lähenemine, kus genereeritud andmetüüp lisatakse alamtüübiks andmetüübile, mis defineerib välja, kus alamtüüp kasutusel on. Kuna lokaalsetel andmetüüpidel ei ole defineeritud nime, siis loob generaator uue tüübi, kasutades vastava elemendi nimetust, ning lisab sellele järelliite `Type`.

Genereeritavad andmetüübid ise suuremat rakendusloogikat ei sisalda. Põhirõhk on suunatud definitsioonis kirjeldatud elementide alusel sobivate väljade loomisele ja seotud andmetüüpide tekitamisele.

### 2.4.4 Prototüübis realiseeritud teenusekirjelduse elemendid

Töö käigus loodav prototüüp ei realiseeri *WSDL* ja *XML Schema* spetsifikatsiooni täies mahus. Järgnevalt tuuakse välja peamised elemendid, mida loodav tüübipakkuja kasutada oskab.

**_WSDL_ spetsifikatsiooni elemendid**

- `service` - loetakse kõik `port` alamelementide kirjeldused. Tüübipakkuja saab hakkama ka rohkem kui ühe `service` elemendi olemasoluga.

- `port` - tuvastatakse vastav `binding` element. Lisaks loetakse *SOAP* ja *X-tee* protokolli nimeruumides defineeritud `address` elemente ning nende poolt edasikantavat infot serveri asukoha ja andmekogu nime kohta.

- `binding` - tuvastatakse vastav `portType` element. Alamelementidest on kasutusel *SOAP* nimeruumi `binding`, millest loetakse kasutatava sidumise stiili info. Kontrollitakse, et `transport` väärtus vastaks *HTTP* protokollile.

- `binding`/`operation` - elementidest on kasutuses *X-tee* teenuse versiooni tähistav element. Samuti valideeritakse, et *SOAP* `operation` elemendis kasutatav stiil oleks vastavuses `binding` elemendi omaga. Alamelementidest töödeldakse `input` ja `output` elemendid, millest eraldatakse *SOAP* ümbrikus kasutatavate sõnumiosade (päis ja keha) info ning kas teenus kasutab *MIME* manuseid.

- `portType` - töödeldakse paralleelselt `binding` elemendiga vastavate osade kokkuviimiseks. Tuvastatakse päringus kasutatavate sõnumite definitsioonid ning nende alamosad.

- `message` - eraldatakse info kas päringu sisendi või väljundi kohta. Alamelemendid `part` defineerivad konkreetsed *XML Schema* elemendid või tüübid, mida päringus on nõutud kasutada.

- `types` - eraldatakse info kõikide *WSDL* dokumendis esitatud *XML Schema* definitsioonide kohta.

***XML Schema* spetsifikatsiooni elemendid**

- `schema` - globaalse taseme elementidest on toetatud järgmiste elementide kasutamine: `annotation` (sisu ignoreeritakse); `include` ja `import` - mujal defineeritud skeemide kaasamine antud definitsioonidele; `complexType`, `element`, `simpleType`, `attribute`, `attributeGroup` - *XML* dokumendi põhielementide, atribuutide ja kasutatavate andmetüüpide definitsioonid;

- `attributeGroup` - toetatud on ainult `attribute` definitsioonid;

- `simpleType` - toetatud on `restriction` definitsioonid;

- `simpleType restriction` - toetatud on `enumeration`, `minLength` ja `pattern` alamelementide kasutamine;

- `element` - alamelementidest on toetus `simpleType` ja `complexType` kasutamisele. Arvestatakse atribuudiga defineeritud tüüpidega;

- `attribute` - toetatud andmetüübile viitamine ja lokaalselt alamelemendina defineeritud tüüp. Eraldi käsitlemine *SOAP* kodeeringus massiivide jaoks;

- `complexType` - toetatud on `simpleContent`, `complexContent`, `choice`, `sequence`, `all`, `attribute` nimelised alamelemendid. Samuti toetatakse abstraktsete tüüpide definitsioone;

- `simpleContent` ja `complexContent` - toetatud on alamelement `extension`;

- `all` - kõiki võimalikke alamelemente on lubatud kasutada;

- `sequence` - toetatud on alamelement `element`;

- `choice` - toetatud on alamelemendid `element` ja `sequence`;

- `extension` - toetatud on alamelemendid `sequence` ja `attribute`. Lisaks loetakse

atribuudist info laiendatava või piiratava tüübi kohta;

Täiendavalt on piiratud võimalusi elementide kordsuse `minOccurs` ja `maxOccurs` kasutamisel. Elemendid, mille `maxOccurs` on väärtusega 1, saab defineerida lihtväljadena, kuid suuremate väärtuste korral tuleb kasutada massiive väärtuste hoidmiseks. Hetkel on toetatud ainult lihtsamate massiivide toetus, kus massiivi moodustab täpselt 1 alamelement.

### 2.4.5 Teenuseid väljakutsuvate meetodite ülesehitus

Peamine funktsionaalsuse loogika asub `port` elemendile vastava genereeritud andmetüübi meetodites. Et vähendada korduva koodi tekkimist on teenuse väljakutse viidud ühte kõrgema taseme meetodisse `MakeServiceCall`. Antud meetod tegeleb loogikaga, mis on ühine kõikide teenuste väljakutsete juures: ühenduse loomine, päringu koostamine, vastuse lugemine ja töötlemine ning ühenduse lõpetamine. Konkreetse teenuse spetsiifiline loogika antakse meetodisse parameetritena, milleks on anonüümsed funktsioonid vastava teenuse päise ja sõnumi keha serialiseerimiseks ning samuti vastusest sisu väljalugemiseks.

Kuna *WSDL* standard lubab ühele `service` elemendile defineerida mitmeid `port` elemente, mis omakorda tähendab mitme erineva klassi genereerimist, siis on üldine kood viidud abstraktsesse baasklassi, millest iga konkreetne klass pärineb.

Iga teenuse väljakutset hõlmava meetodi sisu koosneb põhiliselt eelpool nimetatud anonüümsete funktsioonide defineerimisest. Peale anonüümsete funktsioonide defineerimist kasutatakse neid baasklassi meetodi `MakeServiceCall` väljakutsel ning tagastatakse saadud tulemus. Ühe võimaliku meetodi definitsioon on välja toodud koodinäidises 7.

```
public xroad.testSystemResponse testSystem(xroad.testSystem body) {
    // Defineerib päise elemendid, mis on teenusekirjelduse järgi nõutud.
    var requiredHeaders = new string[] { ... };
    // Defineeritakse funktsioon päise kirjutamiseks.
    Action<XmlWriter> writeHeader = (writer) => {
        base.WriteHeader(writer, "testSystem", requiredHeaders);
    };
    // Defineeritakse funktsioon, mis tegeleb päringu sisu kirjutamisega.
    Action<XmlWriter> writeBody = (writer) => {
        writer.WriteAttributeString("xmlns", "svc", null, "http://aktorstest.x-
            road.ee/producer");
        ...
```

```
            bodySerializer.Serialize(writer, body);
        };
        // Defineeritakse funktsioon, mis tegeleb vastuse sisu lugemisega.
        Func<XmlReader, IDictionary<string, Stream>, xroad.testSystemResponse>
            readBody = (reader, responseAttachments) => {
            ...
            xroad.testSystemResponse v0 = null;
            while (MoveToElement(reader, null, null, 3))
            {
                ...
                if (reader.LocalName == "body") {
                    v0 = ((xroad.testSystemResponse)(bodySerializer.Deserialize(reader)
                        ));
                }
            }
            return v0;
        };
        return base.MakeServiceCall<xroad.testSystemResponse>(null, writeHeader,
            writeBody, readBody);
}
```

Koodinäidis 7: *X-tee* teenuse realisatsioon

### 2.4.6 Tüübipakkuja liidese realiseerimine

Viimase sammuna toimiva tüübipakkuja loomisel on vajalik eelnevalt loodud funktsionaalsuse väljakutsumise võimaldamine. Selleks, et kompilaator ja arendusvahendid oskaks tüübipakkujaga suhelda, tuleb luua vastav klass, mis läbi kindlate liideste rakendamise võimaldab ligipääsu loodud funktsionaalsusele.

Loodav klass peab realiseerima järgmised liidesklassid:

- `Microsoft.FSharp.Core.CompilerServices.IProvidedNamespaces`
- `Microsoft.FSharp.Core.CompilerServices.ITypeProvider`

Esimene liidesklass `IProvidedNamespaces` on vajalik selleks, et kirjeldada ära koodinimeruum, kus loodav tüübipakkuja paiknema hakkab. Olulisemad realiseeritavad meetodid on järgmised:

- `GetTypes()` - tagastab loodava tüübipakkuja enda klassidefinitsiooni;
- `NamespaceName` - annab soovitud nimeruumi nime loodavale tüübipakkujale;

Teine liidesklass `ITypeProvider` tegeleb konkreetsete tüübipakkujate funktsionaalsuste võimaldamisega. Antud töö kontekstis on olulisem tähendus järgmistel meetoditel:

- `GetStaticParameters(...)`
  Antud meetod defineerib tüübipakkuja enda parameetrid, mille kaudu saab arendajale anda lisavõimalusi genereeritava koodi käitumise defineerimiseks. Käesolevas töös on kõige olulisem parameeter *WSDL* dokumendi faili nimi või võrguasukoha aadress;

- `ApplyStaticArguments(...)`
  Selles meetodis toimub konkreetse tüübipakkuja koodi genereerimine. Etteantud parameetrite alusel kutsutakse välja kood, mis laeb etteantud asukohast *WSDL* dokumendi ning genereerib selle alusel uued andmetüübid. Viimase sammuna tagastatakse genereeritud koodi juurtüüp;

Eraldi tuleb arvestada sellega, kuidas arendusvahendid kasutavad tüübipakkujat. Aktiivse koodimuutmise tulemusena võib arendusvahend sama tüübipakkuja poole pöörduda mitmeid kordi. Kuna *WSDL* dokumendid on sageli suured ja nende töötlemine võib võtta märgatava aja, siis on vajalik optimeerida nende kasutamist. Antud juhul saab kasutada definitsioonide vahemälus hoidmist (eeldusega, et sama failinime või võrguasukoha taga asuv *WSDL* dokument ühe arendussessiooni käigus ei muutu).

Kui siiski *WSDL* dokumendis muudatusi tehakse, säilib arendusvahendi taaskäivitamiseni esimese töötlemise seis. See on aktsepteeritav, kuna antud tüübipakkuja ei ole mõeldud *WSDL* dokumendi disainimisvahendiks või validaatoriks, vaid tegu on arendusvahendiga, mis peaks hõlbustama varasemalt koostatud teenusekirjelduste lugemist.

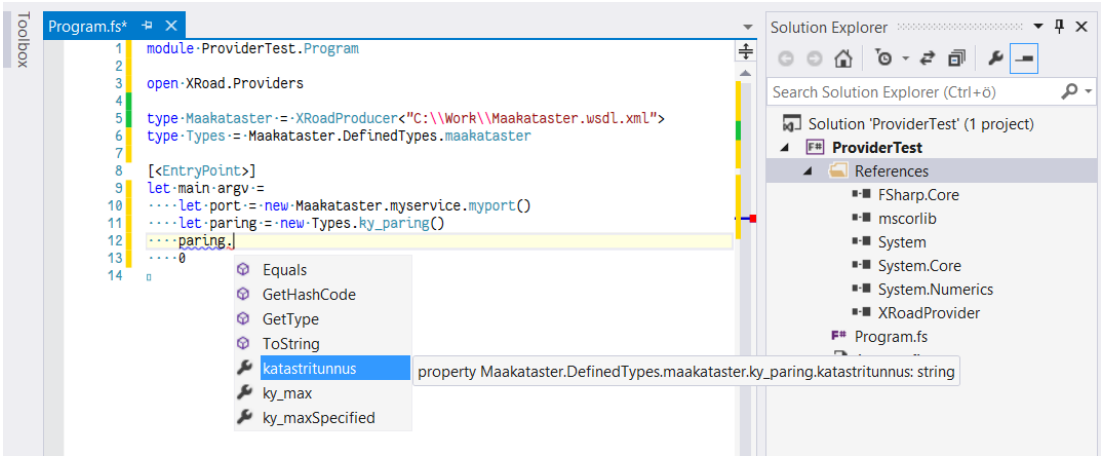
Kirjeldatud funktsionaalsuse realiseerimise tulemusena oskavad arendusvahendid ja *F#* kompilaator luua andmetüübi `XRoadProducer`, mis esindab tüübipakkuja abstraktset tüüpi. Sellel on defineeritud staatiline parameeter, millega määratakse ära teenusekirjelduste asukoht. Parameetri rakendamise tulemusena luuakse konkreetne tüüpipakkuja, mis sisaldab etteantud teenusekirjelduse poolt nõutud funktsionaalsust.

# 3 Lahenduse praktiline kasutamine

## 3.1 Tüübipakkuja kasutamine arendusvahendis

Valmis kujul on väliselt tüübipakkuja näol tegu tüüpilise .NET teegiga. Peamiseks komponendiks on fail nimega `XRoadProvider.dll`, mis on kasutatav *.NET* või *Mono* platvormi *F#* programmeerimiskeele projektides. Selleks, et kasutada tüübipakkuja funktsionaalsust *Visual Studio* projektis, tuleb projekti lisada viide tüübipakkuja teegile. Peale seda on võimalik tüübipakkuja abil saada juurdepääs valitud *WSDL* dokumentidele ja neid realiseerivatele teenustele.

Valitud *WSDL* dokumendi kasutamiseks tuleb defineerida antud projektis uus tüüp, mis luuakse tüübipakkuja poolt. Tüübipakkujale tuleb parameetriks ette anda *WSDL* dokumendi faili asukoht või võrguaadress. Tüübipakkuja näidissessioon on kujutatud joonisel 6.




Joonis 6: Tüübipakkuja kasutamise näidissession *Visual Studio* projektis

*F#* programmeerimiskeel pakub võimalust koodi jooksutada ka interaktiivses käsureaaknas või skriptina. See tähendab, et soovi korral on võimalus koodi läbi proovida ilma vahepealse kompileerimiseta. Üldjoontes on tüübipakkuja analoogselt kasutatav ka skriptimiskeskkonnas, erinevused on tehnilist laadi, nt. teegi laadimisel tuleb skripti alguses kasutada koodirida:
`#r @"XRoadProvider.dll".`

Kogu kirjeldatud *Visual Studio* funktsionaalsus on sarnaste töövõtetega korratav ka *MonoDevelop* arenduskeskkonnas ja teistes *F#* keelt toetavates koodiredaktorites.

## 3.2 Tüübipakkuja testimine valitud teenusepakkujatega

Järgnevalt viidi läbi mõned tüübipakkuja testimissessioonid valitud *X-tee* teenusepakkujatega. Testimissessioonid viidi läbi *F#* keele skriptimiskeskkonnas. Testimiseks valiti teenusekirjeldused *X-tee* arendajatele mõeldud juhenditest. Praktiliseks teenuste proovimiseks valiti *E-toimiku* infosüsteem, kuna töö autoril on konkreetse süsteemi testimiseks kõige parem juurdepääs. Nimetatud teenusekirjelduste toetamine loeti prototüübi üheks eelduseks.

### 3.2.1 *X-tee* protokoll 2.0 juhendi näidisteenus

Antud näidisteenusel puudub võimalus teenuse reaalseks väljakutsumiseks, kuna vastav teenusepakkuja ei ole *arendus-X-tee* keskkonnas kättesaadav. Samas on tegu piisavalt väikese *WSDL* dokumendiga, et proovida, kuidas tüübipakkuja *RPC/Encoded* stiilis esitatud tüübi- ja teenusekirjelduste laadimisega toime tuleb.

Juhendist võetud teenusekirjelduse lugemise peale annab tüübipakkuja vea

```
error FS3033: The type provider 'XRoad.Providers.Impl+XRoadProducerProvider'
    reported an error:
Message mimeResponse does not contain part p1
```

Lähemal uurimisel on selge, et probleemiks on viga `uploadMime` teenuse kirjelduses, kus `binding` plokis kirjeldatud teenuse väljund viitab sõnumiosale nimega `p1`:

```
<!-- /definitions/binding/operation[@name='uploadMime']/output/multipartRelated
    -->
<mime:part>
  <mime:content part="p1" type="application/binary" />
</mime:part>
```

Samas, vastav sõnumiosa puudub:

```
<!-- /definitions -->
<message name="mimeResponse">
  <part name="paring" type="tns:mime_paring" />
  <part name="keha" type="tns:mime_vastus" />
  <part name="p2" type="base64Binary" />
</message>
```

Teenuse kirjeldust vaadates võib eeldada, et tegelikult on soovitud viidata sõnumiosale `p2`. Peale parandust suudab tüübipakkuja edukalt teenuste kirjeldused laadida ning arendusvahend oskab pakkuda 3 erinevat teenust ning nendega seotud andmetüübid.

Lihtsa testserveri abil uuritud väljaminevad sõnumid vastavad visuaalselt võimalikule veebiteenuse sisendile, kuid kuna vastav teenus puudub, siis ei ole neid võimalik praktiliselt testida.

### 3.2.2 *X-tee* protokoll 3.1 juhendi näidisteenus

Sarnaselt protokoll 2.0 teenusega, puudub võimalus 3.1 versiooni näidisdokumendi praktiliseks testimiseks. Võimalik on genereeritud andmetüüpide, teenuste ja sõnumite visuaalne hindamine.

Juhendist võetud teenusekirjelduse lugemise peale annab tüübipakkuja vea

```
error FS3033: The type provider 'XRoad.Providers.Impl+XRoadProducerProvider'
    reported an error:
Missing global element definition ({http://x-road.ee/xsd/x-road.xsd}
    unitValidResponse)
```

Tegemist on tüübipakkuja poolt antud veateatega, mis väidab, et kuskil dokumendis on viidatud elemendile, mida reaalselt defineeritud ei ole. Probleemsele elemendile on viidatud järgmises sõnumiosa definitsioonis:

```
<!-- /definitions -->
<message name="unitValidResponse">
  <part name="body" element="xrd:unitValidResponse"/>
</message>
```

Uurides viidatud nimeruumi definitsiooni faili, selgub, et sellist elementi ei olegi defineeritud. Sama nimega on defineeritud kompleks-tüüp.

```
<!-- /schema -->
<complexType name="unitValidResponse">
```

Teenusekirjelduse parandamiseks tuleb lisada elementide definitsioonid teenusepakkuja enda skeemi.

```
<!-- /definitions/types/schema -->
<element name="unitValidResponse" type="xrd:unitValidResponse" />
```

Algses kasutuskohas tuleb viidata lisatud elemendile:

```
<message name="unitValidResponse">
  <part name="body" element="tns:unitValidResponse"/>
</message>
```

Analoogsed parandused on vaja teha lisaks ka elementidele `unitValid`, `unitRepresentResponse` ja `unitRepresent`.

Sarnaselt 2.0 protokolli kasutavale teenusele, vastavad väljaminevad sõnumid visuaalselt võimalikule veebiteenuse sisendile, kuid kuna vastav teenus puudub, siis ei ole neid võimalik praktiliselt testida.

### 3.2.3 *E-toimik*

Kättesaadavuse tõttu pakub infosüsteem *E-toimik* kõige paremat ülevaadet lahenduse praktilisuse kohta. Tegemist on suure infosüsteemiga, mille *WSDL* dokument on mahukas, kuid lai teenuste valik annab võimaluse spetsiifiliste funktsionaalsuste järeleproovimiseks.

#### Andmetüüpide genereerimine

Käesoleva töö tulemusena käib andmetüüpide loomine arendusvahendi ja kompilaatori vahendusel automaatselt. Kogu liidese loomiseks vajalik kood on esitatud koodinäidises 8, kus põhitöö toimub viimasel real.

```
open XRoad.Providers

type Etoimik = XRoadProducer<"http://xx.x.xxx.xxx/etoimik.asmx?wsdl">
```

Koodinäidis 8: *E-toimiku* liidese loomine

Kuna teenusekirjeldus on mahukas, võtab liidese genereerimine orienteeruvalt 10 sekundit. Selle aja jooksul vastloodud tüübile koodipakkumine ei tööta, kuid liidese loomine on ühekordne tegevus, mis edasist tööd ei sega.

#### Teenuse kasutamine

Konkreetse teenuse näiteks valitakse üks andmete muutmise teenus ja teine, millega neid muudetud andmeid uuesti küsitakse. Koodinäidis 9 demonstreerib isiku andmete lisamist ja nende pärimist:

```
let service = new Etoimik.EtoimikService.EtoimikServicePort()
service.Ametniknimi <- "test.kasutaja"
service.ProducerUri <- "http://xx.x.xxx.xxx/etoimik.asmx"

let arg1 = new Etoimik.DefinedTypes.etoimik.IsikuLisamineMuutmineParing()
```

```
let isik = new Etoimik.DefinedTypes.etoimik.FyysilineIsik()
isik.Eesnimi <- "Eesnimi"
isik.Nimi <- "Nimi"
isik.Kood <- "30101010007"
arg1.isik <- isik
arg1.kasutaja <- isik
let v1, _ = service.IsikuLisamineMuutmine(arg1)

let arg2 = new Etoimik.DefinedTypes.etoimik.IsikuVaataminePäring()
arg2.objektID <- v1.ObjektID
arg2.kasutaja <- isik
let v2, _ = service.IsikuVaatamine(arg2)

printfn "Isikute arv vastuses: %d" v2.Length
printfn "Isiku tüüp on: %s" (v2.[0].GetType().Name)
let fi = v2.[0] :?> Etoimik.DefinedTypes.etoimik.FyysilineIsik
printfn "Isik: %s %s (%s)" fi.Eesnimi fi.Nimi fi.Kood
```

Koodinäidis 9: Isiku lisamine ja vaatamine *E-toimikus*

Koodinäidises 9 luuakse esimese asjana teenuse meetodeid sisaldav andmetüüp. Kuna *E-toimik* eeldab *X-tee* päise elemendi `ametniknimi` kasutamist, tuleb see enne teenuse kasutamist väärtustada. Samuti tuleb anda serveri asukoht, kuna teenusekirjelduses antud väärtus ei ole korrektne.

Teine plokk tegeleb isiku lisamiseks vajalike andmete kokkupanemisega ja teeb teenuse väljakutse. Tagastatud väärtus vastab lisatud isiku andmetele, sisaldades ka isiku uut unikaalset `ObjektID`-d. Kasutades seda väärtust, tehakse kolmandas plokis isiku vaatamise päring.

Viimases plokis kirjutatakse vaatamise päringu tulemus standardväljundisse ning tulemus on järgmine:

```
Isikute arv vastuses: 1
Isiku tüüp on: FyysilineIsik
Isik: Eesnimi Nimi (30101010007)
```

Tulemus on ootuspärane ning samal ajal demonstreerib liidese võimekust lihtsamate andmetüüpide serialiseerimisel ja deserialiseerimisel. Teise päringu vastus sisaldab massiivi, mille töötlemine on olnud vigadeta.

#### *MIME* manustega teenuse kasutamine

Ühe eesmärgina on tõstatatud *MIME* manuste kasutamise võimalus genereeritud

liidesega. Kuna *E-toimik* defineerib *MIME* manustega teenuseid, siis on võimalik antud funktsionaalsus läbi proovida. Koodinäidis 10 demonstreerib manuste edastamist ja vastuvõtmist.

```
open System.Collections.Generic
open System.IO

let sisu = new MemoryStream([| 0uy; 1uy; 2uy; 3uy |])
let att3 = Dictionary<string,Stream>()
att3.Add("faili-sisu", sisu)

let arg3 = Etoimik.DefinedTypes.etoimik.FailiLisamineMuutmineParing()
arg3.fail <- Etoimik.DefinedTypes.etoimik.Fail()
arg3.fail.Nimetus <- "Sodifail.bin"
arg3.fail.Sisu <- Etoimik.BinaryContent(att3, "faili-sisu")
arg3.kasutaja <- isik
let v3, _ = service.FailiLisamineMuutmine(arg3, att3)

let arg4 = Etoimik.DefinedTypes.etoimik.FailiVaatamineParing()
arg4.ObjektID <- v3.ObjektID
arg4.kasutaja <- isik
let v4, _, att4 = service.FailiVaatamine(arg4)

printfn "Manuste arv vastuses: %d" att4.Count
let m = att4 |> Seq.map (fun x -> ("cid:" + x.Key, x.Value)) |> Map.ofSeq
printfn "Failiga seotud manus on olemas: %b" (m.ContainsKey(v4.Sisu.Id))
printfn "Manuse sisu: %A" ((m.[v4.Sisu.Id] :?> MemoryStream).ToArray())
```

Koodinäidis 10: Faili lisamine ja vaatamine *E-toimikus*

Esimese asjana defineeritakse manuse objekt, mis sisaldab 4-baidist sisu. Faili lisamise päringu koostamisel seostatakse faili sisu manuste loendiga ja määratakse manuse tunnus. Fail edastatakse infosüsteemi *E-toimik* `FailiLisamineMuutmine` päringu väljakutsega.

Seejärel üritatakse sama faili infosüsteemist lugeda. Selleks kasutatakse päringus failile omistatud unikaalset tunnust `ObjektID` `FailiVaatamine`. Viimase sammuna kuvatakse standardväljundisse info loetud faili kohta, mis on järgmine:

```
Manuste arv vastuses: 1
Failiga seotud manus on olemas: true
Manuse sisu: [|0uy; 1uy; 2uy; 3uy|]
```

Tagastatud manuse sisu vastab algselt edastatule, seega võib järeldada, et manuste teisaldamine *MIME* konteinerites on töökorras.

Teatud ebamugavus on sisse jäänud seoses andmetüübi ja manuse kokkuviimisega. Nimelt deserialiseerimisel kasutatavad tunnused ei klapi. Sõnumi sees on tunnusele lisatud eesliide `cid:`, mida manuste nimekirjas ei kasutata. Sellest tulenevalt tuleb hetkel manuste kokkuviimisel tunnuseid käsitsi parandada.

## 3.3 Tulemused

Praktiliste testide tulemuse saab kokku võtta sellega, et lahendusele püstitatud eesmärgid said täidetud. On loodud tehniline lahendus, mis, kasutades *F#* tüübipakkujate mehhanismi, oskab arendusvahendis automaatselt genereerida signatuure etteantud teenusekirjeldustele ning kompileerimisel luua praktiliselt kasutatava funktsionaalsusega rakendusi. Tüübipakkuja tuleb toime arendust toetanud *X-tee* juhendi näidisteenuste *WSDL* spetsifikatsioonidega ning ühe praktilise kasutuskohana toetab infosüsteemiga *E-toimik* liidestumist.

Loodud lahendus suudab edukalt töödelda *X-tee* protkolli versioonide 2.0 ja 3.1 juhendite näidisteenusekirjeldust ning arvestada nende eripäradega päringute kokkupanemisel ja vastuvõtmisel. Praktiliste katsetuste läbiviimiseks ei ole protokolli versiooni 3.1 jaoks hetkel sobivat süsteemi.

Tüübipakkuja abil on võimalik edastada päringuid *MIME* manuseid kasutades, kui teenusekirjeldus seda ette näeb, ning samuti on võimaldatud *MIME* manustega vastuste vastuvõtmine.

Praktiliste näidete alusel on võimalik järeldada, et antud lahendus sobib hästi testide kirjutamiseks, nt. väljatoodud koodinäidised on võimalik minimaalse vaevaga vormistada vastava infosüsteemi automaattestideks. Samuti pakub käsureaaknas arendamine mugavat võimalust teenusega tutvumiseks ning käitumise proovimiseks: kirjutatud koodi saab koheselt käivitada ja vastavalt vajadusele muuta või kohandada. Sama kasutusviis sobib ka teenuse tutvustamiseks, kuna on võimalik interaktiivselt näidata, milliseid parameetreid teenused kasutavad, kuidas erineva sisendi peale käituvad ning mis on selle tulemus.

Kuna *F#* programmeerimiskeel on kompileeritud keel, siis lisaks käsureavõimalustele saab antud lahendust kasutada ka reaalsetes rakendusprogrammides *X-tee* teenuseliideste realiseerimisel. Selle paremaks toetamiseks peab genereeritud kood olema kiire, efektiivne ja võimalikult väikese jalajäljega, mis tuleks saavutada prototüübi edasiarenduste käigus.

### 3.3.1 Edasiarendusvõimalused

- Testimisest tuli välja, et *MIME* manuste vastuvõtmisel on jäänud sisse keerukus manuste seostamisel päringu vastuse objektidega. Tüübipakkuja poolt loodud kood võiks tulevikus arvestada selle eripäraga ning andmete kokkuviimist lihtsutada.

- Laetud tüübidefinitsioonide vahemälus hoidmist saab täiendavalt optimeerida. Hetkel piirdub vahemälu kasutamine ühe teenusekirjelduse kontekstiga, kuid tegelikult on samad definitsioonid üldiselt korduvkasutatavad. Optimeerimine võimaldaks aega kokku hoida olukorras, kus on kasutusel mitu tüübipakkuja objekti samaaegselt või objekti konfiguratsiooni muudetakse.

- Võrguressursi kättesaadavusega võib olla probleeme, nt. arendamise käigus oli olukordi, kus `www.w3.org` server ei vastanud. Sellega toimetulemiseks tuleb lisada seadistamisvõimalus, mis lubab määrata lokaalset kataloogi, kust kättesaamatuid definitsioone otsida.

- Kuna üks teenusekirjeldus on sageli seotud mitme teise definitsioonifailiga, siis saab täiendavat kiirusevõitu hankida failide paralleelsest töötlemisest ja asünkroonsest laadimisest.

- `XmlSerializer`-i kasutamist saab optimeerida. Hetkel luuakse uus objekt serialiseerimise hetkel, mis tähendab, et sisemiselt peab süsteem iga kord kulutama aega serialiseerimiseks vajalike detailide arvutamiseks. Serialiseerimiseks kasutatavate objektide taaskasutamine aitaks vältida seda liigset arvutamist ning muudaks teenuse liideste töö kiiremaks.

- `XmlSerializer`-i kasutamine seab piirangud kasutatavate klasside struktuurile ning ei ole piisavalt paindlik, et võimaldada teatud etappides alternatiivset lähenemist. Nimetatud komponendist loobumine ja tüübipakkuja jaoks kohandatud seriliseerimise koodi kirjutamine annaks suurema vabaduse genereeritava koodi planeerimisel. Töö kirjutamise hetkel teada olevad nõrgad kohad, mida uus lahendus parandada lubaks:

  - Eelmises punktis nimetatud `XmlSerializer`-i loomise ajakulu ei vaja eraldi lahendamist.

  - Võimaldaks paremini siduda *MIME* manuseid ja genereeritud objekte, kuna manuse sisu ja vastava objekti kokkuviimise saab teha liidese poolt.

  - `XmlSerializer`-i toetus `choice` elementidele on mitterahuldav, kuna kaotab ära andmetüüpide struktuuri; uue lahendusega saab kasutajale pakkuda paremaid tüüpe.

- Lahendusel puuduvad hetkel korralikud automaattestid. Nende loomine aitaks tagada rakenduse töökindlust ka suuremate muudatuste tegemisel.

# Kokkuvõte

Käesoleva töö peamiseks eesmärgiks oli luua lahendus, mis, kasutades *F#* programmeerimiskeele tüübipakkuja mehhanismi, võimaldaks kiiremat ja mugavamat tööprotsessi andmevahetuskihil *X-tee* infosüsteemidega liidestumisel ning nende teenuste kasutusele võtmisel. Töö tulemusena pidi valmima prototüüp, mille abil saaks konkreetse infosüsteemi näitel lahendust demonstreerida.

Töö esimeses osas tutvustati andmevahetuskihi *X-tee* põhimõisteid ning toodi välja probleemid, millega uute infosüsteemidega liidestumisel tuleb kokku puutuda. Lisaks kirjeldati *X-tee* platvormil kasutatavat teenusekirjelduse formaati, mille alusel rakendusliideseid ehitatakse. Samuti anti ülevaade *F#* programmeerimiskeelest ning tüübipakkuja mehhanismist, mis oli antud töö aluseks.

Töö tulemusena valmis tüübipakkuja prototüüp, mis võimaldab arendusvahendis *X-tee* teenusekirjelduste põhjal automaatselt vajalikud signatuurid luua, mis omakorda lihtsustab liideste loomist. Lisaks luuakse kompileerimisel teenuseliideste klassid ja meetodid, mis võimaldavad vajalikke päringuid teha lihtsalt ja korrektses formaadis. Lisaks *XML* päringutele lubab prototüüp kasutada ka keerukamaid päringuid, millega liigutada infosüsteemide vahel manustena erinevate failide sisu.

Praktiliste katsetuste tegemisega jõuti tulemuseni, et loodud lahendusel on potentsiaali liidestumisprotsessi lihtsamaks muutmisel. Samuti leiti, et tüübipakkuja on sobilik erinevates kasutusvaldkondades: testimisel, teenuste kasutamise näitlikult selgitamisel, uute teenustega tutvumisel ning ka reaalsetes lõpprakendustes kasutamisel. Praktiliste katsetuste tulemusena pakuti välja hulk edasiarendamisvõimalusi, mille realiseerimisel saab prototüübi põhjal luua kasutusmugava, täpse ja kiire rakendusteegi *X-tee* teenustega liidestumiseks.

# Summary

The main goal for this thesis was to create a solution, that allows faster and convenier working process when interfacing with various information systems on data exchange layer *X-Road*. The solution was bound to use *F#* programming language type provider mechanism for the task. As the result of the work a prototype was expected to be built which would allow to demonstrate the solution with particular information system.

First part of the thesis introduces data exchange layer *X-Road* and explains issues that need to be taken care of when building an interface for information system. It also presents service description format which is used by *X-Road* and used to build application interfaces. Finally, overview about *F#* programming language and type provider mechanism is given.

The result of this work is a type provider prototype, which allows development environment to display signatures of types and methods to the developer which simplifies building process. Using the type provider, the compiler can build required classes and methods for service interfaces, which implement functionality that is required to execute correct service request against information systems. In addition to plain *XML* request, the prototype allows usage of more complex request, which transfer file contents as attachments between information system and client application.

As the result of practical example sessions it was stated that implemented solution has potential to make process of interfacing with information systems easier. Also the type provider is suitable in various practises like testing, explaing service use through practical examples and trying out new services. The compiler can use type provider mechanism to build working service layer for end-user applications. Based on the results of practical example sessions, possible improvement and extension ideas were presented which would allow this prototype to evolve into easily used, strict and fast component to integrate *X-Road* services.

# Kasutatud kirjandus


[1] Riigi Infosüsteemi Amet. Andmevahetuskiht X-tee. [WWW] https://www.ria.ee/x-tee/ (19.02.2015)

[2] Riigi Infosüsteemi Amet. Nõuded infosüsteemidele ja adapterserveritele. [WWW] http://x-road.ee/docs/est/nouded_infosusteemidele_ja_adapterserveritele.pdf (17.05.2015)

[3] Box D., Ehnebuske D., Kakivaya G. Simple Object Access Protocol (SOAP) 1.1. [WWW] http://www.w3.org/TR/2000/NOTE-SOAP-20000508/ (16.05.2015)

[4] Christensen E., Curbera F., Meredith G. Web Services Description Language (WSDL) 1.1. [WWW] http://www.w3.org/TR/wsdl (16.05.2015)

[5] The World Wide Web Consortium. XML Schema. [WWW] http://www.w3.org/2001/XMLSchema (16.03.2015)

[6] Riigi Infosüsteemi Amet. X-tee sõnumi protokolli versioonid. [WWW] https://www.ria.ee/public/x_tee/X-tee_sonumi_protokolli_versioonid.pdf (16.05.2015)

[7] Riigi Infosüsteemi Amet. X-tee sõnumi protokolli muudatused versioonilt 3.1 versioonile 4.0 üleminekul. [WWW] https://www.ria.ee/public/x_tee/X-tee_sonumi_protokolli_muudatused.pdf (16.05.2015)

[8] Barton J. J., Thatte S., Nielsen H. F. SOAP Messages with Attachments. [WWW] http://www.w3.org/TR/SOAP-attachments (16.05.2015)

[9] Wikipedia, the free encyclopedia. F Sharp (programming language). [WWW] http://en.wikipedia.org/wiki/F_Sharp_%28programming_language%29 (17.05.2015)

[10] Vallaste H. e-teatmik. [WWW] http://vallaste.ee/index.htm (17.05.2015)

[11] F# Software Foundation. F# Software Foundation. [WWW] http://fsharp.org/ (17.05.2015)

[12] Petricek T., Skeet J. *Real-World Functional Programming: With examples in F# and C#*. Greenwich, CT: Manning, 2010.

[13] Syme D., Battocchi K., Takeda K. Strongly-Typed Language Support for Internet-Scale Information Sources. [WWW] `http://research.microsoft.com/pubs/173076/information-rich-themes-v4.pdf` (17.05.2015)

[14] Petricek T., Trelford P. *F# Deep Dives*. Shelter Island, NY: Manning, 2015.

[15] w3schools.com. XML Schema Reference. [WWW] `http://www.w3schools.com/schema/schema_elements_ref.asp` (16.04.2015)