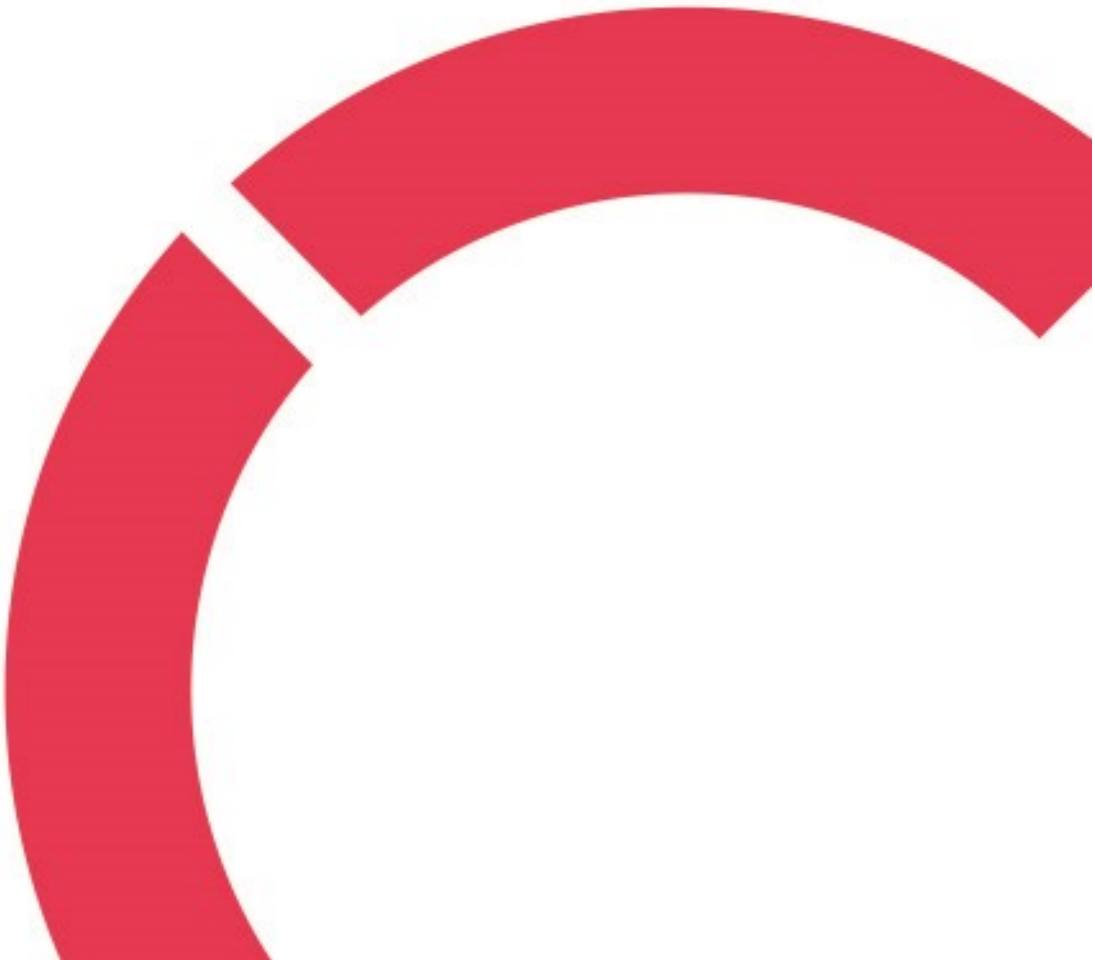
**Veronica Witick**

# SIIRTYMINEN TIETOKANTAPOHJAISISTA ARVIOINTIVÄLINEISTÄ XML-MUOTOISEEN RATKAISUUN

**Opinnäytetyö**
**CENTRIA-AMMATTIKORKEAKOULU**
**Tieto- ja viestintätekniikan koulutus**
**Toukokuu 2023**



**TIIVISTELMÄ OPINNÄYTETYÖSTÄ**

| **Centria-ammattikorkeakoulu** | **Aika**<br>Toukokuu 2023 | **Tekijä/tekijät**<br>Veronica Witick |
|---|---|---|
| **Koulutus**<br>Insinööri (AMK), tieto- ja viestintätekniikka | | ☒ AMK<br>☐ YAMK |
| **Työn nimi**<br>SIIRTYMINEN TIETOKANTAPOHJAISISTA ARVIOINTIVÄLINEISTÄ XML-MUOTOISEEN RATKAISUUN | | |
| **Työn ohjaaja**<br>Jari Isohanni | | **Sivumäärä**<br>34 |
| **Työelämäohjaaja**<br>Tom Cygnel | | |


Tämän opinnäytetyön toimeksiantajana oli ohjelmistoyritys Vitec Raisoft Oy. Yritys kehittää ja tarjoaa asiakkailleen arviointivälineitä eli instrumentteja, joiden avulla kerätään hoitotyössä käytettävää tietoa.

Vielä toistaiseksi instrumenttien data tallennetaan relaatiotietokannan tauluihin. Tässä työssä aloitettiin siirtymäprojekti, jonka tavoitteena on siirtyä käyttämään instrumenttien varastointiin tietokantataulujen sijasta XML-dokumentteja.

Työ koostui seuraavista osatehtävistä: instrumentti-XML:n rakenteen suunnittelu, XML-dokumentin eksportointiin tarvittavien Delphi-metodien lisääminen lähdekoodiin, XML-skeeman laatiminen sekä uuden instrumenttieditoriohjelman toteuttaminen.

Ensimmäinen osatehtävä oli suunnitella sellainen XML-rakenne, että kunkin yksittäisen instrumentin kaikki tiedot voidaan esittää yhtenä XML-dokumenttina. Relaatiotietokantojen ja XML-dokumenttien erilaisuudesta johtuen tämä ei ollut aivan suoraviivaista. Siinä missä data tietokannassa on järjestettynä tauluihin, XML-dokumentilla on hierarkkinen, puuta muistuttava rakenne.

Instrumenttidata piti pystyä eksportoimaan tietokantatauluista XML-dokumentiksi. Toisena osatehtävänä oli kirjoittaa lähdekoodiin uusia Delphi-metodeja, joiden avulla tämä eksportointi tehdään. Kolmantena osatehtävänä oli toteuttaa C#-kielellä editoriohjelma, joka avulla XML-formaatissa olevia instrumentteja pystyisi helposti tarkastelemaan ja muokkaamaan. Viimeisenä osatehtävänä oli laatia instrumentti-XML:ää kuvaava XML-skeema.

Instrumentti-XML:n suunnittelu sekä Delphi-metodien ja XML-skeeman kirjoittaminen saatiin valmiiksi. Instrumenttieditorin työstö jatkuu, mutta sekin saatiin toteutettua niin pitkälle, että sen toimintaa pääsi työn aikana laajasti testaamaan. Vaikka projekti on kesken, tähän mennessä saavutettujen välitavoitteiden pohjalta se todennäköisesti onnistutaan saattamaan päätökseen.



**Asiasanat**
C#, Delphi, Relaatiotietokanta, XML, XML-skeema

**ABSTRACT**

| | | |
|---|---|---|
| **Centria University of Applied Sciences** | **Date**<br>May 2023 | **Author**<br>Veronica Witick |
| **Degree programme**<br>Bachelor of Engineering, Information Technology | | |
| **Name of thesis**<br>CREATING AN XML-BASED SOLUTION FOR INSTRUMENT DATA PREVIOUSLY STORED IN A DATABASE | | |
| **Centria supervisor**<br>Jari Isohanni | | **Pages**<br>34 |
| **Instructor representing commissioning institution or company**<br>Tom Cygnel | | |


This work was commissioned by the software company Vitec Raisoft Oy. The company develops and provides assessment tools, also called instruments, which are used for collecting information that can be utilized in healthcare work.

For now, all instrument data is stored in a relational database. The current work comprises the first part of a project that aims at migrating the instruments from database tables into XML documents.

This work included the following subtasks: planning the structure of the instrument XML, writing the Delphi methods for exporting the data, creating an XML schema and making a new editor tool.

The first subtask was to plan instrument XML structure where one XML document contains all data of a single instrument. Due to the differences between relational databases and XML documents, this was not quite straightforward. The data in a database is arranged in tables, whereas XML documents have hierarchical, tree-like structure.

It should be possible to export the instrument data from database tables into an XML document. To this end, the second subtask was to write new Delphi methods to the backend source code. The third subtask was to use C# to create an editor tool that can be used for easy viewing and editing of the instrument XML. The last subtask was to write an XML schema that describes the instrument XML.

Planning of the instrument XML structure, as well as writing of the Delphi methods and the XML schema were finished. The work on the instrument editor tool is ongoing, but a large part of it has been implemented and tested by now. Even though the project is still in progress, it seems likely that it will be successfully finished according to the plans.



**Key words**
C#, Delphi, Relational database, XML, XML schema

## KÄSITTEIDEN MÄÄRITTELY

### Relaatiotietokanta

Relaatiotietokannassa informaatio on järjestettynä taulukoihin. Eri taulukoiden tietoja voidaan yhdistää toisiinsa niiden yhteisten taulukkoarvojen – perusavainten ja vierasavainten – avulla. (IBM.)

### XML

XML (Extensible Markup Language) on rakenteellisen informaation esittämiseen käytettävä tekstimuotoinen formaatti (XML Essentials).

### XML-skeema

XML-skeema on kieli, jonka avulla voidaan määritellä XML-dokumentin rakenne ja siihen kohdistuvat rajoitukset (Schema).

**KUVIOT**

**KUVAT**



**KOODIT**

# 1 JOHDANTO

Tämän opinnäytetyön toimeksiantajana on ohjelmistoyritys Vitec Raisoft Oy (jäljempänä Raisoft). Raisoftin tuotteiden tarkoituksena on kerätä tietoa, jota voidaan käyttää hoitotyöhön liittyvien päätöksien pohjana. Yritys kehittää ja tarjoaa asiakkailleen tiedonkeruuta varten arviointivälineitä eli instrumentteja, joista huomattava osa perustuu interRAI-järjestelmään. Näitä arviointivälineitä on laaja valikoima eri tarkoituksiin: ikäihmisten hoitotyöhön, kehitysvammaisten palveluihin ja mielenterveystyöhön. (Vitec Raisoft.)

Käytännössä instrumentit sisältävät arviointilomakkeen sekä muun muassa mittareita, joiden arvot lasketaan lomakevastausten perusteella. Lisäksi kokonaisuuteen kuuluu monesti instrumentti- tai asiakaskohtaisia raportti- ja hoitosuunnitelmapohjia.

Tällä hetkellä instrumenttien data tallennetaan relaatiotietokannan tauluihin. Jatkossa on kuitenkin tarkoituksena siirtyä käyttämään tähän tarkoitukseen XML-dokumentteja tietokantataulujen sijasta. Tässä työssä aloitettiin kyseinen siirtymäprojekti.

Työllä on kaksi keskeistä tavoitetta: jokaisen instrumentin tiedot tulisi voida esittää XML-tiedostona, ja tämän instrumentti-XML:n muokkaamista ja tarkastelua varten tulisi tehdä editoriohjelma. Editorin kieleksi sovittiin C#, ja projektin pääaskelet olivat alusta saakka selvillä, mutta muutoin sain toteutukseen suhteellisen vapaat kädet.

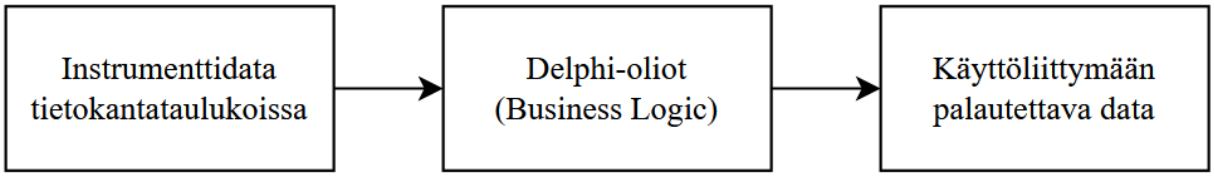
Kuvaan työn etenemisen pääasiallisesti osa-alueittain, en niinkään tiukan kronologisesti. Käytännössä työ etenikin siinä määrin poukkoillen, ettei kronologinen kuvaus välttämättä olisi kovin selkeä esitystapa. Tulin useaan kertaan hypelleeksi edestakaisin eri työvaiheiden välillä, sillä muutosten teko yhdessä paikassa sai toisinaan huomaamaan kehityskohteita toisaalla (ja välillä myös aiheutti niitä).

# 2 TÄSTÄ TYÖSTÄ YLEISESTI

Tässä luvussa kuvaan lyhyesti, miten suunniteltu instrumenttidatan siirto tietokantataulukoista XML-tiedostoihin vaikuttaa ohjelmiston toimintaan yleisellä tasolla. Lisäksi kuvailen relaatiotietokantojen ja XML-dokumenttien keskeisiä ominaisuuksia.

## 2.1 Lähtötilanne ja ohjelman suunniteltu toiminta

Nykyisellään ohjelma toimii kuviossa 1 esitetyllä tavalla. Tietokantatauluista haettavan instrumenttidatan perusteella muodostetaan erilaisia, lähdekoodin luokkiin perustuvia olioita. Näiden olioiden kautta saadaan edelleen käyttöliittymään palautettava data.




KUVIO 1. Ohjelmiston nykyinen toiminta

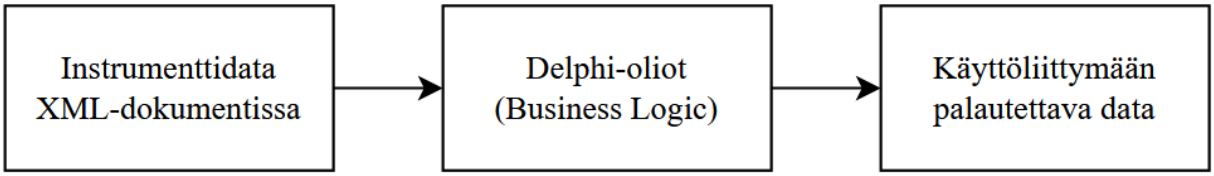
Tämä jo pitkään käytössä ollut ratkaisu on varsin toimiva, ja tietokannassa olevan instrumenttidatan muokkaukseen on olemassa yrityksen sisäisessä käytössä olevia editoriohjelmia. Nämä työvälineet mahdollistavat yleisimpien instrumenttimuutosten teon editorin käyttöliittymän kautta. Jotkin muutoksista joutuu kuitenkin tekemään suoraan tietokannassa.

Editorityövälineitä on useita erilaisia. Karkeasti ottaen jokaiselle instrumentin osa-alueelle on oma editorinsa: yksi lomakkeen rakennemuutoksille, toinen tekstimuutoksille, kolmas skriptien muokkaamiseen, neljäs loogisten hyppyjen käsittelyyn, viides käsikirjatekstien työstöön, ja niin edelleen.

Nykytilanteella on kuitenkin kääntöpuolensa. Kun muutokset tehdään tietokantaan – riippumatta siitä tehdäänkö ne työvälineitä hyödyntäen vai suoraan – peruutustoimintoa ei ole. Muutosten vaikutuksille ei myöskään ole selkeää rajaa, vaan ne voivat kohdistua koko tietokantaan. Tästä riskialttiudesta joh-

tuen muutostenteko-oikeus voidaan antaa vain suhteellisen harvoille. Lisäksi tietokantataulukoihin tehtyjä muutoksia on hankala jäljittää, ja olisi myös kätevämpää, jos editoriohjelmia olisi noin tusinan sijasta vain yksi yleiskäyttöinen.

Edellä mainitut syyt toimivat kimmokkeena tässä työssä aloitetulle projektille, jonka tavoitteena on kuviossa 2 kuvattu tilanne. Jatkossa instrumenttidata on siis tarkoitus säilyttää tietokannan sijasta XML-dokumentissa. Kutakin yksittäistä instrumenttia vastaisi yksi XML-dokumentti. Kaavion keskellä olevien olioiden ei ole tarkoitus juurikaan muuttua, joten toteutetun XML-rakenteen yhtenä vaatimuksena olikin, että sen tulisi sisältää kaikki kyseisten olioiden muodostamiseen vaadittu informaatio.




KUVIO 2. Ohjelmiston suunniteltu toiminta

### 2.2 Relaatiotietokannan ja XML-dokumentin vertailua

Relaatiotietokanta ja XML-dokumentti ovat tietorakenteina varsin erilaiset. Relaatiotietokannassa tiedot on järjestetty tauluihin, joiden tietoja on mahdollista yhdistellä toisiinsa hakujen avulla, kun taas XML-dokumentilla on elementeistä koostuva hierarkkinen, puumainen rakenne. Tästä johtuen relaatiotietokannassa säilytetyn instrumenttidatan järjestäminen XML-dokumentiksi ei ollut suoraviivaista, vaan rakenteen suunnittelu oli jo sinällään oma osatehtävänsä.

Seuraavissa alaluvuissa kuvaillaan relaatiotietokantojen ja XML-dokumenttien keskeisiä piirteitä.

#### 2.2.1 Relaatiotietokannat

Relaatiotietokannat muodostuvat tauluista. Tieto on järjestettynä taulun tietueiden (rivien) kenttiin (sarakkeisiin). Jokaisella rivillä on myös uniikki tunniste eli avain, joka voi olla joko perus- tai viiteavain. Näiden avainten avulla eri taulujen sisältämiä tietoja voidaan yhdistellä toisiinsa. (IBM; Oracle.)

Relaatiotietokantojen ylläpitoon käytetään erityisiä relaatiotietokannanhallintajärjestelmiä (Relational Database Management System, RDBMS). Ne ovat ohjelmia, joiden kautta käyttäjä voi tarkastella ja muokata tietokannan sisältämiä tietoja. (IBM; Oracle.)

### 2.2.2 XML

XML (Extensible Markup Language) on tekstimuotoinen formaatti, jota laajalti käytetään rakenteellisen informaation esittämiseen (XML Essentials). Sitä voidaan lukea ohjelmallisesti, mutta se on myös helposti ihmisen luettavissa (W3C 2008). Koodissa 1 on esimerkki XML-syntaksista.

```
<instrument>
        <longname>
                <text lang="fin">Kotihoito</text>
                <text lang="swe">Hemvård</text>
        </longname>
        <shortname/>
</instrument>
```

KOODI 1. Esimerkki XML-syntaksista

XML-dokumentti koostuu yhdestä tai useammasta elementistä. Nämä elementit voivat olla sisäkkäin. Dokumentissa on tasan yksi juurielementti, jonka sisällä kaikki muut elementit ovat. (W3C 2008.)

Mikäli elementillä on sisältöä, sen rajaajina ovat erilliset aloitus- ja lopetusmerkinnät eli tägit. Tyhjä elementti – eli elementti vailla sisältöä – voidaan merkitä yksittäisellä tägillä. (W3C 2008.) Koodissa 1 ”shortname” on tyhjä elementti.

Aloitusmerkinnöissä ja tyhjissä elementeissä voi olla attribuutteja. Näiden attribuuttien avulla voidaan esittää elementtiin liittyvää informaatiota nimi-arvo-pareina. (W3C 2008.)

# 3 INSTRUMENTTI-XML:N RAKENTEEN SUUNNITTELU

Ensimmäisenä työvaiheena oli instrumentti-XML:n rakenteen suunnittelu. Tavoitteena oli, että jokaista yksittäistä instrumenttia vastaisi yksi XML-dokumentti.

Kuten jo todettiin, relaatiotietokannassa instrumenttidata on järjestettynä hyvin eri tavalla kuin XML-dokumentissa. Tietokannassa yksittäisen instrumentin tiedot ovat hajallaan lukuisissa eri tauluissa, ja toisaalta yksittäisessä taulussa on useiden eri instrumenttien tietoja. Kaikilla riveillä on oltava avaimina toimivia tunnisteita, jotta tietoja pystytään hakemaan ja yhdistelemään. XML-dokumentin rakenne puolestaan on hierarkkinen. Tunnisteiden tarve on pienempi, sillä jo elementin sijainti dokumentissa kertoo paljon. Tosin esimerkiksi skriptien parametreissa tunnisteet ovat välttämättömiä, jotta tiedetään, mitä kysymystä tai skriptiä kukin parametri vastaa. Samasta syystä myös lomakekysymyksissä ja skripteissä on oltava tunnisteet.

Työtä helpotti suuresti se, että eräässä yrityksen sisäisessä käytössä olevista ohjelmista oli jo ennestään funktio, jolla pystyttiin eksportoimaan tietokantataulujen instrumenttidatasta tietynlainen XML, joka sisälsi lomakkeen ja skriptit. Sellaisenaan tätä funktiota tai sen tuottamaan XML-dokumenttia ei voinut hyödyntää, mutta ne toimivat mallina ja suuntaviittana tässä työssä suunnitellulle instrumentti-XML-rakenteelle.

## 3.1 Vaatimukset

Instrumentti-XML:n tulisi sisältää kaikki yksittäistä instrumenttia varten tarvittava informaatio. Lisäksi asetin sille joitain muitakin tavoitteita: turhaa toistoa tulisi välttää, ja dokumentin tulisi olla helposti navigoitavissa ja luettavissa myös ilman editorin apua.

Yksi keskeinen osatehtävä oli XML-dokumenttiin sisällytettävien tietojen kartoitus. Oli muun muassa selvitettävä, mitä attribuutteja tietyn kysymystyypin elementissä on oltava, jotta XML:ssä on mukana kaikki kyseisen kysymyksen määrittelemiseen tarvittava informaatio.

Edellisessä luvussa mainitusta mallina toimineesta XML-rakenteesta oli hyvä lähteä liikkeelle. Se kuitenkin sisälsi vain lomakkeen ja skriptit, vaikka instrumentteihin sisältyy paljon muitakin osioita. Lisäksi mukana olleista osioista puuttuivat muun muassa jotkin tärkeät attribuutit, eikä tähän XML:ään ollut mahdollista sisällyttää kuin yksi kieli kerrallaan.

Malli-XML-rakenteessa oli myös runsaasti toistoa: esimerkiksi skriptien parametrien alla oli kyseisen parametrin lähdekysymyksen käsikirjateksti, vaikka sitä ei parametrin yhteydessä tarvittu ja vaikka sama teksti löytyi myös alkuperäisen kysymyksen alta. Tällaisen redundantin informaation vaikutus dokumentin pituuteen oli huomattava. Erään instrumentin XML-tiedosto oli mallina toimineella funktiolla eksportoituna 13 185 riviä pitkä. Kun turhan toiston kirjoittamieni apumetodien avulla siisti pois, sama tiedosto oli enää 5973 rivin mittainen – toisin sanoen vain 45 % alkuperäisestä pituudestaan. (Tein tämän trimmauskokeilun melko varhaisessa vaiheessa projektia. Luovuin tästä lähestymistavasta melko pian, ja tein sen sijasta kokonaan erillisen funktion instrumentti-XML:n tuottamista varten.)

Dokumentin luettavuuden parantamiseksi koetin nimetä elementit ja attribuutit kuvaavasti, pääasiassa samalla tavoin kuin ne oli tietokannassakin nimetty. Koetin myös sijoittaa ja järjestää ne mahdollisimman loogisesti. Lisäksi kiinnitin huomiota siihen, että jos tietyn elementin sisällä oli useita erilaisia lapsielementtejä, niin ne tarvittaessa ryhmiteltiin omien apuelementtiensä alle. Ryhmittelyyn käytetyt ylimääräiset elementit eivät edustaneet mitään informaatiota, vaan niiden tehtävänä oli ainoastaan dokumentin käsittelyn helpottaminen. Turhan redundanttiuden vastakohtana tätä voisi kenties kutsua hyödylliseksi redundanttiudeksi.

Esimerkkinä tilanteesta, jossa ryhmittelyyn käytetyt elementit olivat tarpeen, voisi mainita vaikkapa skriptien parametrit. Yksittäistä skriptiä edustavan script-elementin sisällä on name-elementti ja vaihteleva määrä parameter-elementtejä (sekä lukuisia muita elementtejä, jotka eivät ole tämän esimerkin kannalta keskeisiä). Näistä name-elementti voi olla script-elementin alla suoraan, mutta parameter-elementit oli lukemisen ja ohjelmallisen käsittelyn helpottamiseksi järkevintä koota parameters-elementin sisälle (koodi 2). Näin parametrit pystyy ohjelmassa parameters-elementin paikantamisen jälkeen läpikäymään silmukkarakenteella ilman, että jokaisen kohdalla tarvitsisi tarkistaa, onko kyseessä parameter-elementti vai ei.

```
<script scriptname="sBMI" scriptid="1234567">
        <name>
                <text lang="fin">Painoindeksi</text>
```

```
                    <text lang="eng">Body Mass Index</text>
            </name>
            <parameters>
                    <parameter name="weight" id="234567"/>
                    <parameter name="height" id="345678"/>
            </parameters>
    </script>
```

KOODI 2. Esimerkki lapsielementtien ryhmittelyyn käytetystä elementistä

Koodissa 2 olevasta esimerkistä on jätetty pois suuri joukko script- ja parameter-elementtien attribuutteja sekä script-elementin lapsielementtejä.

### 3.2 Haasteita

Yksi tapaus, jossa ratkaisuvaihtoehtoja joutui punnitsemaan ja miettimään, oli loogisten hyppyjen sijoitus. Instrumentin arviointilomaketta täyttäessä tietyn vastauksen antaminen voi johtaa siihen, että lomakkeella hypätään seuraavan kysymyksen sijasta johonkin toiseen kohtaan. Joissain tapauksissa väliin jätettyihin kysymyksiin täytetään samalla vastaukset automaattisesti. Tällainen käyttäjän vastauksista riippuva lomakkeen käytös määritetään loogisten hyppyjen avulla.

Loogisten hyppyjen XML-rakenteelle ei ollut ennestään olemassa olevaa esikuvaa malliksi. Loogisille hypyille omistettu Dephi-yksikkö (unit) lähdekoodissa toki oli, ja sen luokkien ja metodien avulla pystyi helposti hakemaan tietokannasta tarvittavat tiedot.

Aluksi piti miettiä, mikä olisi looginen rakenne näille tiedoille – mukaan lukien se, kannattaisiko ne lisätä osaksi arviointipuuta vai tulisiko hypyillä mieluummin olla oma puunsa. Koska hypyt liittyvät lomakkeen kysymyksiin, niiden looginen sijoituspaikka XML-dokumentissa voisi olla kysymysten yhteydessä lomakkeella. Tätä ratkaisua minulle myös keskusteluissa ehdotettiin, ja sitä pariin otteeseen pohdin.

Päätin kuitenkin lopulta tehdä hypyille kokonaan oman, erillisen alipuunsa. Yhteen hyppyyn kun voi liittyä montakin kysymystä, joko hypyn ehtoina tai kohteina. Samoin yhteen kysymykseen voi liittyä

useampi eri hyppy. Jos hypyt olisivat hajallaan lomakkeella, niiden muokkaus ja poistaminen olisi hankalampaa. Hallinta on helpompaa ja vähemmän virhealtista, kun se voidaan tehdä keskitetysti yhdessä paikassa.

### 3.3 Käännössyntaksi

Monissa instrumenteissa on useampi kuin yksi kielivaihtoehto. Esimerkiksi Suomessa käytettävissä instrumenteissa on usein suomen lisäksi mahdollisuus valita tekstien kieleksi ruotsi.

Jotta instrumentin eri kielet saataisiin samaan XML-dokumenttiin mukaan, oli mietittävä tähän tarkoitukseen sopiva käännössyntaksi. Tähän liittyi pientä alkuhaparointia, eikä ensimmäinen hahmotelmani ollut kovinkaan onnistunut, etenkään näin jälkikäteen tarkastellessa. Sain tästä hahmotelmastani palautetta attribuuttien käyttöön liittyen. Palautteen mukaan käännösten laittaminen attribuuttiarvoihin voisi aiheuttaa ongelmia myöhemmin, sillä XML:n attribuutit eivät tue rivinvaihtoa. Lisäksi palautteessa todettiin, että kaikkien attribuuttien tulisi olla pienin kirjaimin. Palautteessa ehdotettiin koodissa 3 olevaa syntaksia.

```
<language text="fin">Käännösteksti</language>
```

KOODI 3. Palautteessa ehdotettu käännössyntaksi

Koodissa 4 puolestaan on esimerkki siitä, miltä XML palautteensaantihetkellä näytti. Nyt sen virheet pistävät silmään itsellänikin, joten jotain olen tullut oppineeksi.

```
<node type="question" infotype="TEXT" uniqueid="1234567">
        <texts FIN="Etunimi" SWE="Förnamn" />
        <manual>
                <FIN>Tämän kysymyksen käsikirjateksti
suomeksi</FIN>
                <SWE />
        </manual>
</node>
```

KOODI 4. Ensimmäinen, hylätty ehdotus käännössyntaksiksi

Koodissa 5 on uusi käännössyntaksi, jonka toteutin saadun palautteen pohjalta. Se on hieman erilainen, mutta pääpiirteissään samanlainen kuin palautteessa ehdotettu syntaksi.

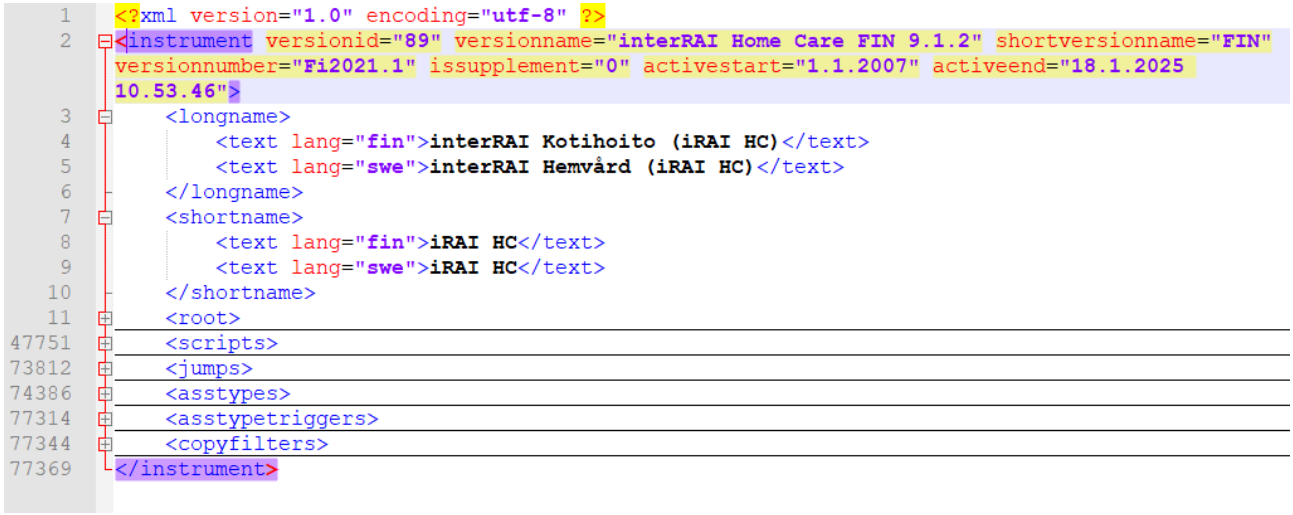
```
<node type="question" infotype="TEXT" uniqueid="1234567">
        <text lang="fin">Etunimi</text>
        <text lang="swe">Förnamn</text>
        <manual>
                <text lang="fin">Tämän kysymyksen käsi-
kirjateksti suomeksi</text>
                <text lang="swe" />
        </manual>
</node>
```

KOODI 5. Korjaamani käännössyntaksi

### 3.4 Lopullinen rakenne

Tämän tekstin kirjoittamishetkellä instrumentti-XML:n rakenne on alustavasti valmis. Joitain hienoisia muutoksia sinne tänne saattaa myöhemminkin tulla, mutta mitään suurempaa enää tuskin.

XML-dokumentin pituus vaihtelee instrumenteittain ja riippuu muun muassa arviointilomakkeen pituudesta sekä käsikirjatekstien ja skriptien määrästä. Kuvassa 1 on esitetty yhden melko suuren instrumentin XML. Siinä on nähtävissä muun muassa riviltä 73 812 alkava loogisten hyppyjen alipuu.




KUVA 1. Erään instrumentin XML-dokumentti

# 4 INSTRUMENTTI-XML:N TUOTTAVAT METODIT

Jotta halutunlainen instrumentti-XML saatiin eksportoitua, olemassa olevan sovelluksen lähdekoodiin oli kirjoitettava uusia metodeja. Olen tässä kirjoitelmassa erottanut näiden Delphi-metodien tekemisen omaksi työvaiheekseen, mutta käytännössä työstin niitä samanaikaisesti instrumentti-XML:n suunnittelun kanssa.

## 4.1 Delphin rooli tässä työssä

Olemassa oleva backend-lähdekoodi – mukaan lukien sovellus, jolla XML oli tarkoitus eksportoida – oli kirjoitettu Delphillä, joka oli itselleni uusi tuttavuus. Myöskään Delphi-ohjelmien kirjoittamiseen käytettyä RAD Studio -ohjelmointiympäristöä en ollut käyttänyt aiemmin. Aloitinkin tämän vaiheen uuden kielen opettelulla ja perehtymällä olemassa olevaan lähdekoodiin.

Delphi on korkean tason ohjelmointikieli, joka soveltuu olio-ohjelmointiin. Delphillä toteutetussa sovelluksessa on tyypillisesti yksi projektitiedosto (.dpr) ja useita yksikkötiedostoja (.pas). Valtaosa koodista on yleensä yksikkötiedostoissa. (Embarcadero 2022.)

Delphissä on kahdenlaisia rutiineja eli koodilohkoja, joita voidaan kutsua eri puolilta ohjelmaa: funktioita ja proseduureja. Niiden erona on, että funktio palauttaa arvon, kun taas proseduuri ei. (Embarcadero 2019.) Viittaan molempiin edellä mainittuihin tässä tekstissä termillä metodi.

## 4.2 Haasteita

Kaikeksi onneksi Delphin syntaksi osoittautui melko samantapaiseksi kuin muiden, aiemmin opiskelemieni ohjelmointikielien. Lisäksi uusien metodien kirjoittamiseen pystyi katsomaan mallia olemassa olevista metodeista. Lähdekoodissa oli ennestään lukuisia metodeja, joissa instrumenttiin liittyvää dataa haettiin tietokannasta ja joissa tuotettiin erilaisia XML-rakenteita.

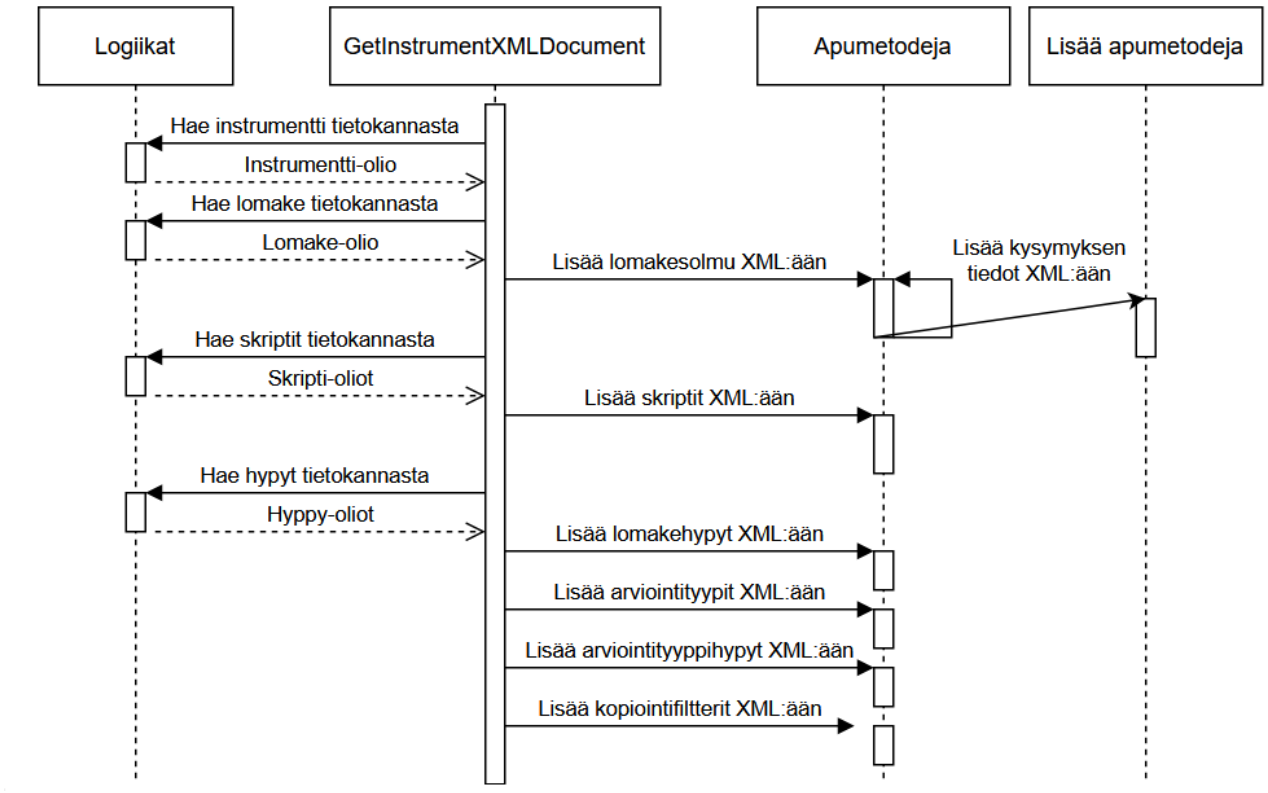
Pienenä haasteena etenkin työn alkuvaiheissa oli automaattisen roskienkeruun puute. Aiemmin olin kirjoittanut ohjelmakoodia enimmäkseen C#:lla ja Javalla, joten resurssien vapauttaminen ei ollut itselle luontevaa. En ollut varma, mitä resursseja pitäisi vapauttaa ja milloin – ja mitä ei. Toisinaan ohjelmointiympäristö herjasi muutosten jälkeen muistivuodosta, minkä jälkeen oli jäljitettävä syy. Ajan myötä itselleni alkoi muodostua parempi käsitys siitä, mitä olioita pitää muistin vapauttamiseksi tuhota.

Haasteellisuutta lisäsi osaltaan myös se, että relevantti lähdekoodi oli hajautettuna eri kansioihin ja tiedostoihin. Tämä hajautus selittynee sillä, että osa koodista oli useille eri sovelluksille yhteistä. Se sovellus, johon olin itse tekemässä muutoksia, oli vain yksi monista. Vähitellen kokonaisuudesta tuli kuitenkin helpompi hahmottaa. Tietyistä tiedostoista näki relevanttien luokkien (esimerkiksi loogisia hyppyjä edustavan luokan) sisällön, kun taas parista muusta keskeisestä tiedostosta löytyi hyviä esimerkkejä edellä mainittujen luokkien hyödyntämisestä.

### 4.3 Metodien yhteispeli

Kaiken kaikkiaan tulin lisänneeksi lähdekoodiin monta uutta metodia: yhden funktion ja lukuisia proseduureja, joita funktio kutsuu. Kuviossa 3 on erittäin pelkistetty kuvaus näiden instrumentti-XML:n rakentamiseen osallistuvien metodien toiminnasta. Moni pieni metodi – muun muassa tekstien lisäämiseen käytetyt proseduurit – on jätetty kuvauksesta pois. Keskellä oleva GetInstrumentXMLDocument on funktio, joka palauttaa valitun instrumentin XML-dokumentin. Sen vasemmalla puolen näkyvät logiikat ovat joukko luokkia ja niiden sisältämiä funktioita, jotka hakevat instrumentin tiettyyn osa-alueeseen liittyvän informaation tietokantatauluista ja palauttavat sitä edustavan olion. Oikealla puolestaan ovat proseduurit, jotka lisäävät rakenteilla olevaan XML-dokumenttiin tietyn osion instrumentista.

Logiikoita ei tässä työssä muutettu, mutta GetInstrumentXMLDocument ja muut kuviossa mukana olevat XML:n muodostukseen osallistuvat metodit olivat uusia lisäyksiä lähdekoodiin. Moni niistä pohjautui ennestään olemassa oleviin samantapaisiin metodeihin. Esimerkiksi lomakehypyt lisäävä AddJumpNodes oli kuitenkin vailla esikuvaa.




KUVIO 3. Sekvenssikaavio instrumentti-XML:n tuottavien metodien toiminnasta

# 5 INSTRUMENTTIEDITORI

Instrumentti-XML:n suunnittelun ja toteuttamisen ohella instrumenttieditorin teko oli tämän projektin keskeisin tehtävä. Nämäkään osatehtävät eivät käytännössä olleet täysin toisistaan erillisiä, sillä instrumentti-XML:n rakenne vaikutti siihen, miten editori hakee tietoa dokumentista, ja tämän ohjelmallisen käsittelyn teko mahdollisimman helpoksi puolestaan oli aspekti, joka oli otettava huomioon instrumentti-XML:n rakennetta suunnitellessa. Instrumenttieditoria työstäessäni tulin useaan otteeseen palanneeksi Delphi-metodien pariin, kun huomasin kehityskohteita XML-rakenteessa.

Tavoitteena oli, että editorilla olisi mahdollista tehdä XML-muotoiselle instrumentille ainakin kaikki se, mitä ennestään olemassa olevilla Delphi-pohjaisilla editoreilla pystytään tietokantaan tekemään.

## 5.1 Kieli ja ohjelmointiympäristö

Vanhoista editorityövälineistä poiketen instrumenttieditorin kieleksi sovittiin C#. Editori toteutettiin Windows Forms -sovelluksena Visual Studio -ohjelmointiympäristön avulla.

C# on moderni, erityisesti olio-ohjelmointiin soveltuva ohjelmointikieli. Sen ominaisuuksiin kuuluu muun muassa automaattinen roskienkeruu, jonka ansiosta saavuttamattomiin jääneiden olioiden viemä muistitila vapautetaan automaattisesti. (A tour of C# - Overview.) En valinnut tätä toteutuskieltä itse, mutta olisin todennäköisesti päätynyt samaan kieleen, jos olisin asiasta päättänyt. C#:n kanssa on helppo työskennellä, ja vastaan tuleviin kysymyksiin löytyy runsaasti dokumentaatiota ja vastauksia internetistä.

Aloitin editorin työstämisen perehtymällä siihen, miten C#:lla voi käsitellä XML-tiedostoja. Päädyin lopulta hyödyntämään tähän tarkoitukseen XmlDocument-luokkaa. Vaihtoehtona olisi ollut käyttää XmlReader- ja XmlWriter-luokkia, mutta XmlDocument vaikutti lukemani perusteella parhaalta ratkaisulta tässä tapauksessa. Se osoittautuikin varsin helppokäyttöiseksi ja käteväksi XML-dokumentin hallinnassa.

XmlDocument kuuluu C#-kielen System.Xml-nimiavaruuteen. Kuten nimestä voi päätellä, kyseinen luokka edustaa XML-dokumenttia, ja sen avulla voidaan muun muassa ladata XML-dokumentti muistiin sekä lukea ja muokata sitä. (XmlDocument Class (System.Xml).)

### 5.2 Rakenteen ja toiminnan suunnittelu

Tietokannassa olevan instrumenttidatan muokkaukseen on monta erillistä editoria. Tavoitteena oli, että uusi XML-dokumentille tarkoitettu instrumenttieditori mahdollistaisi vastaavanlaiset muokkaukset kuin kaikki vanhat editorityövälineet yhteensä sekä myös sellaiset muutokset, jotka aiemmin joutui tekemään suoraan tietokannassa. Lisäksi olisi suotavaa, että editoriin lisättäisiin muitakin toimintoja, joilla prosessia saataisiin entisestään sujuvoitettua.

Vanhoja editoreja ei suoraan voinut käyttää mallina, sillä ne oli kirjoitettu eri kielellä ja ne käsittelivät tietoja tietokannassa XML:n sijasta. Niistä sai kuitenkin vinkkejä sen suhteen, millaisilla komponenteilla eri toiminnot voisi graafisessa käyttöliittymässä toteuttaa (puurakenne, tekstikentät, pudotusvalikot, ponnahdusikkunat, ja niin edelleen). Olin itse käyttänyt vanhoja editoreja instrumenttimuutosten tekoon työssäni parin vuoden ajan ennen tämän projektin aloittamista, joten minulla oli tältä pohjalta myös joitain ideoita sen suhteen, mitä halusin uudessa editorissa toteuttaa toisin.

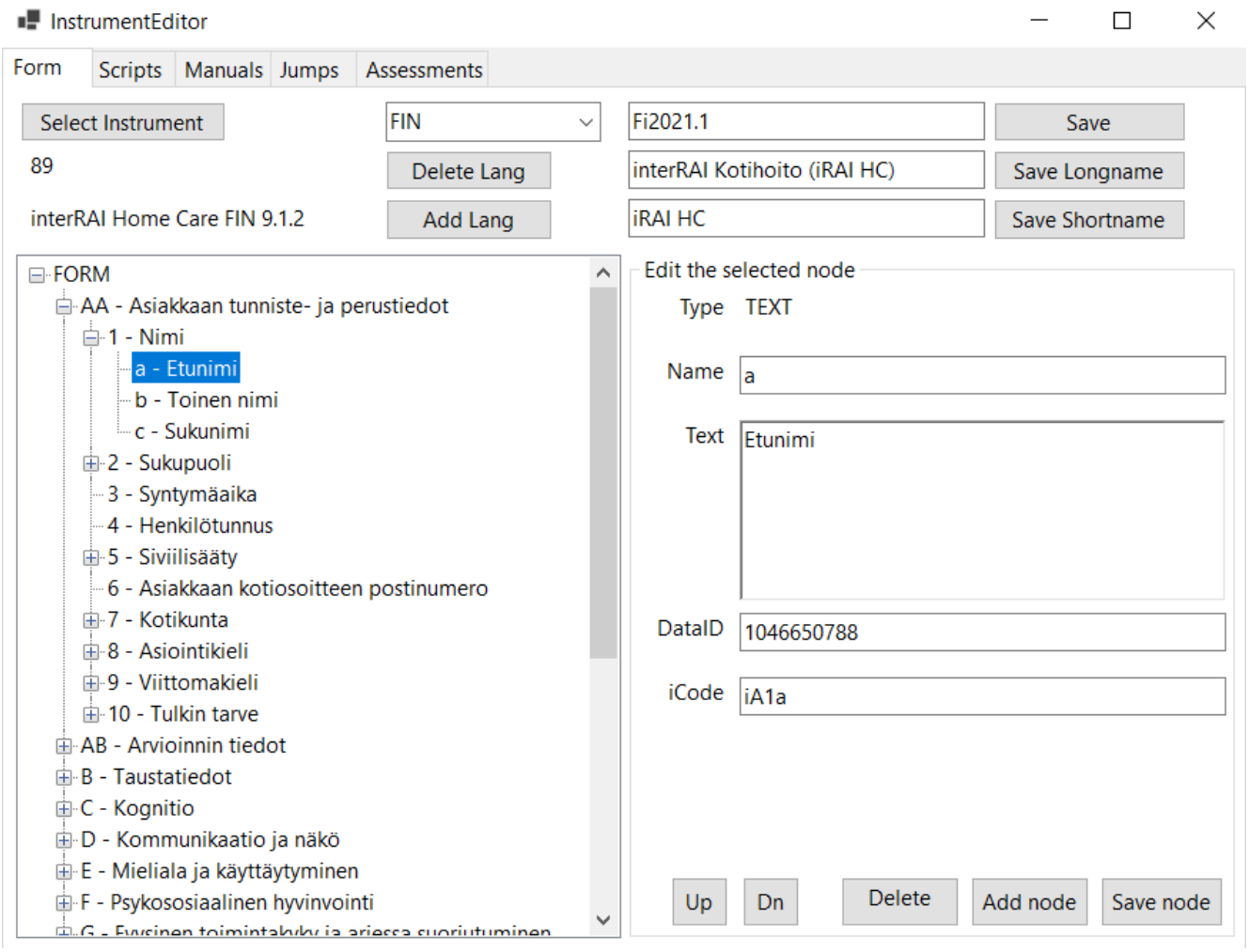
Pyrin instrumenttieditoria työstäessäni kirjoittamaan koodin niin, että metodit ovat tiedostoissa loogisessa järjestyksessä aihepiirin mukaan järjestettyinä, muuttujat ja luokat oli kuvaavasti nimetty ja koodin joukossa oli selittäviä kommentteja. Etenkin työn alkuvaiheessa tulin ehkä kommentoineeksi koodia liikaakin, mutta parempi varmaan niin päin (kuva 2).

```
1562
1563        /*********************************
1564         ****** Script-related things *****
1565         *********************************/
1566
1567
1568        /***** Getters *****/
1569
1570        // This method returns a list of all of the scripts in the selected instrument
            2 references
1571        public List<ScriptCreature> GetScripts()
1572        {
1573            // The script elements are inside a scripts element in the source XML
1574            XmlNode scriptContainer = xmlDoc.SelectSingleNode("//instrument/scripts");
1575
1576            // The "scripts" list is created here. The other option would be the constructor - not sure which one is better, but for now
                  this is here.
1577            scripts = new List<ScriptCreature>();
1578
1579            // Let's loop through the scripts (if any)
1580            if (scriptContainer.HasChildNodes)
1581            {
1582                for (int i = 0; i < scriptContainer.ChildNodes.Count; i++)
1583                {
1584                    XmlNode scriptNode = scriptContainer.ChildNodes[i];
1585
1586                    // Creating a ScriptCreature and adding it onto the list
1587                    scripts.Add(new ScriptCreature(scriptNode, selectedLanguage));
1588                }
1589            }
1590
```

KUVA 2. Ote AssessmentFormEditor-luokan tiheäkommenttisesta koodista

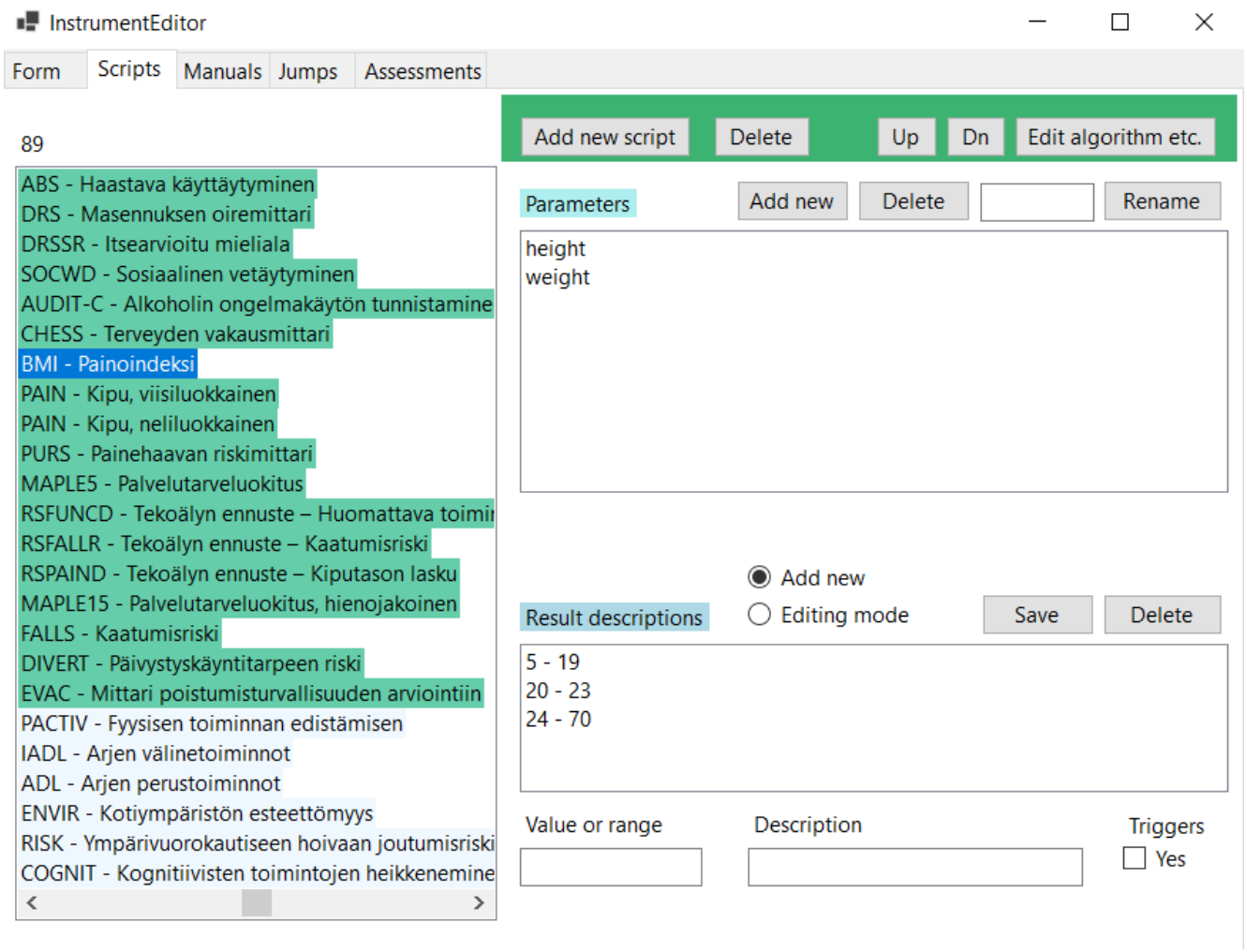
Pyrin myös tekemään graafisesta käyttöliittymästä mahdollisimman intuitiivisen ja helppokäyttöisen. Jaoin instrumenttimuokkauksiin liittyvät eri osa-alueet omille välilehdilleen. Karkeasti ottaen yksi instrumenttieditorin välilehti vastaa yhtä vanhaa editoria, mutta ihan yksi yhteen vastaavuudet eivät ole. Osa muokkaustoiminnoista on erillisissä ponnahdusikkunoissa.

Instrumenttieditorin ensimmäisellä välilehdellä (kuva 3) käyttäjä voi hakea muokattavan instrumentin XML-tiedoston ”Select Instrument” -painikkeesta avautuvan tiedostonvalintaikkunan kautta. Tällä välilehdellä on mahdollista muokata instrumenttiin liittyviä yleisiä tietoja sekä arviointilomakkeen sisältöä.



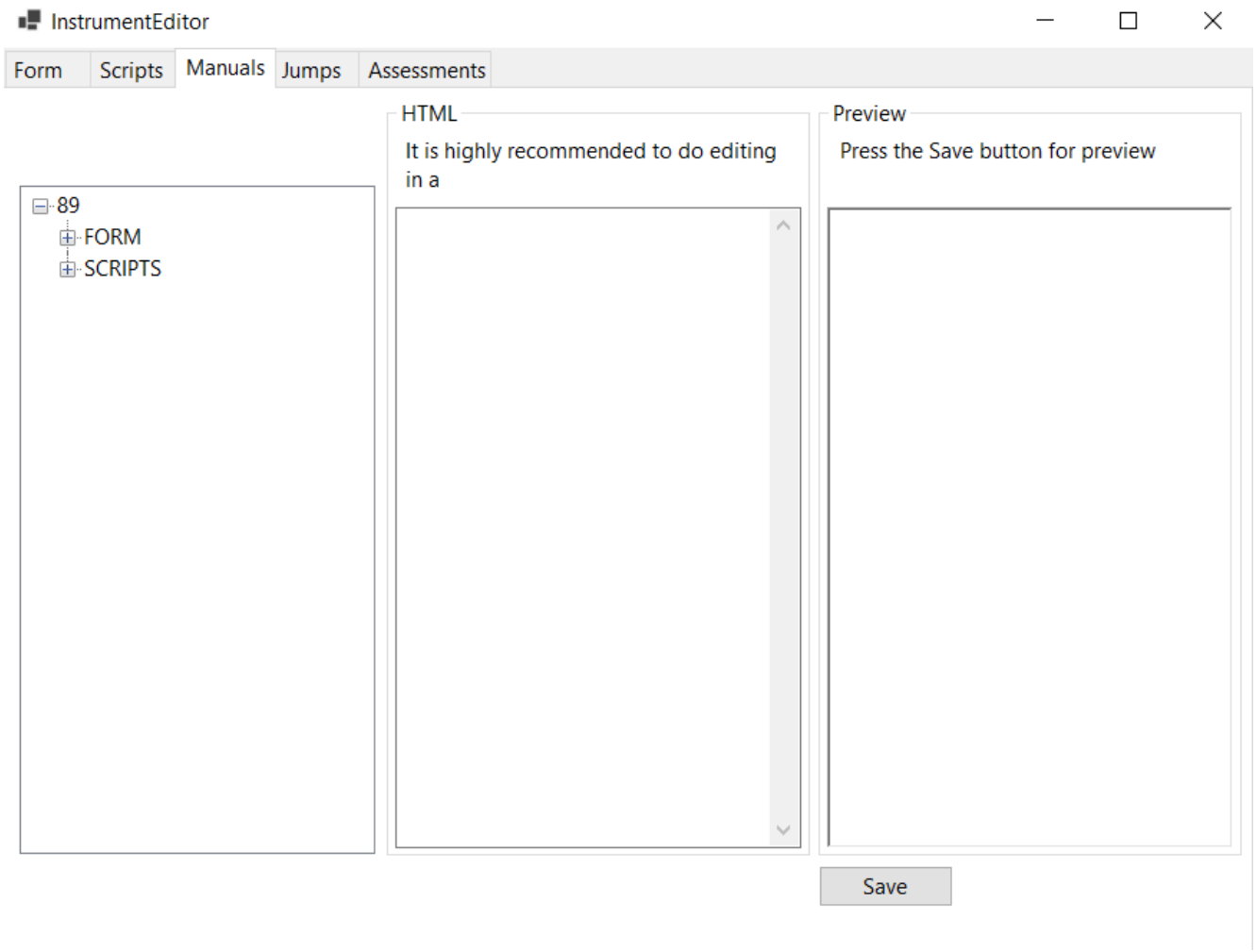

KUVA 3. Instrumenttieditorin lomakevälilehti

Instrumenttieditorin toisella välilehdellä (kuva 4) voidaan muokata instrumentin sisältämiä skriptejä. Painikkeesta "Edit algorithm etc." avautuu ponnahdusikkuna, jossa on muutama oma välilehtensä.




KUVA 4. Instrumenttieditorin skriptivälilehti

Lomakevälilehdellä (kuva 5) käyttäjä voi lisätä ja muokata lomakekysymyksiin ja skripteihin liittyviä käsikirjatekstejä.




KUVA 5. Instrumenttieditorin käsikirjavälilehti

Instrumenttieditorin luokat koetin kirjoittaa siten, että kullakin luokalla olisi selkeä vastuualue, joka on kuvattavissa yhdellä virkkeellä. Tämä osoittautui yllättävän haastavaksi. Käsittelen tätä tarkemmin luvussa 5.3.2.

Muistin säästämiseksi pyrin siihen, että näihin luokkiin pohjautuvia olioita luotaisiin vain tarvittaessa. Käytännössä editori toimii siten, että kun käyttäjä avaa tietyn välilehden, ohjelma hakee tätä välilehteä varten tarvittavat tiedot XmlDocument-oliosta ja luo tietoja vastaavat oliot. Esimerkiksi käyttäjän mennessä Script-välilehdelle ohjelma luo XML:n perusteella ScriptCreature-oliot, jotka se lisää AssessmentFormEditor-luokassa olevaan List-tyyppiseen muuttujaan.

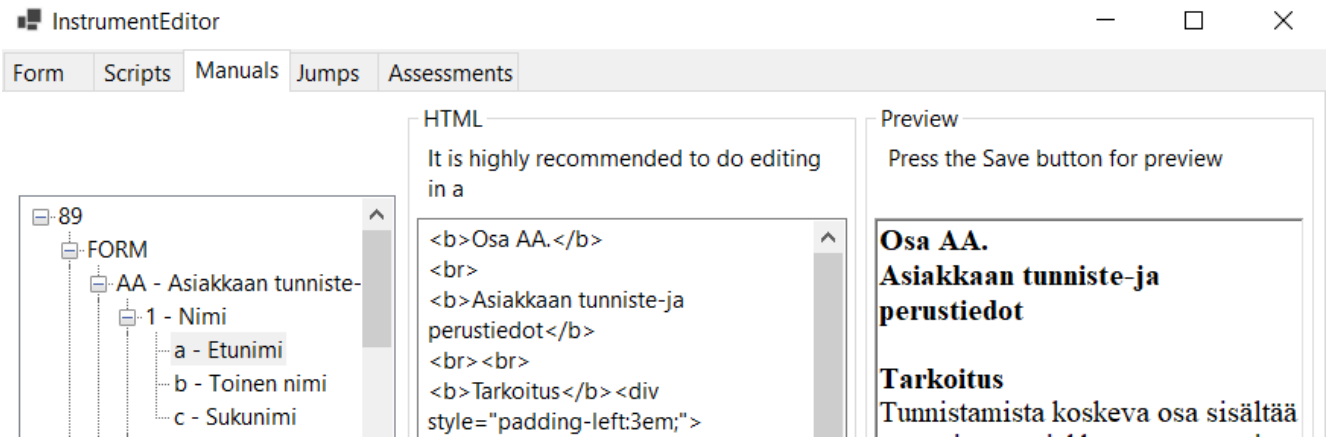
Instrumenttieditoriprojektin versionhallinta toteutettin TortoiseHg:n avulla. Sovelluksen tiedostoille perustettiin oma Mercurial-tietovarastonsa. Sen ansiosta varmuuskopiointi sujui helposti, ja tarvittaessa olisi ollut myös mahdollista palata aiempaan, toimivaan versioon, mikäli jokin tehty muutos olisi osoittautunut virheeksi.

## 5.3 Haasteita

Olio-ohjelmointi oli itselleni tuttua jo ennestään, mutta tästä työstä sai hyvää käytännön harjoitusta.

### 5.3.1 Käsikirjatekstien esikatselutoiminto

Editorin Manuals-välilehdellä (kuva 6) on HTML:llä kirjoitettujen käsikirjatekstien esikatselutoiminto. Tämän esikatselun toteutus ei ollut ihan niin suoraviivaista kuin olin odottanut. Visual Studio ei nimittäin antanut lisätä esikatselutoimintoon tarvittavaa WebBroser-komponenttia käyttöliittymään, vaikka sen olisi pitänyt olla mahdollista. Käytännössä WebBrowser ei ollut valittavissa Visual Studio työkalupakissa.




KUVA 6. Käsikirjavälilehden esikatselutoiminto

Sain kierrettyä ongelman käyttämällä esikatseluun RichTextBox-komponenttia, johon toin tekstin piilotetun (pelkästään koodiin kirjoitetun) WebBrowser-komponentin kautta. Tällä tavoin komponentin – tai lähinnä siihen liittyvän keskeisen toiminnon – sai onneksi lisättyä.

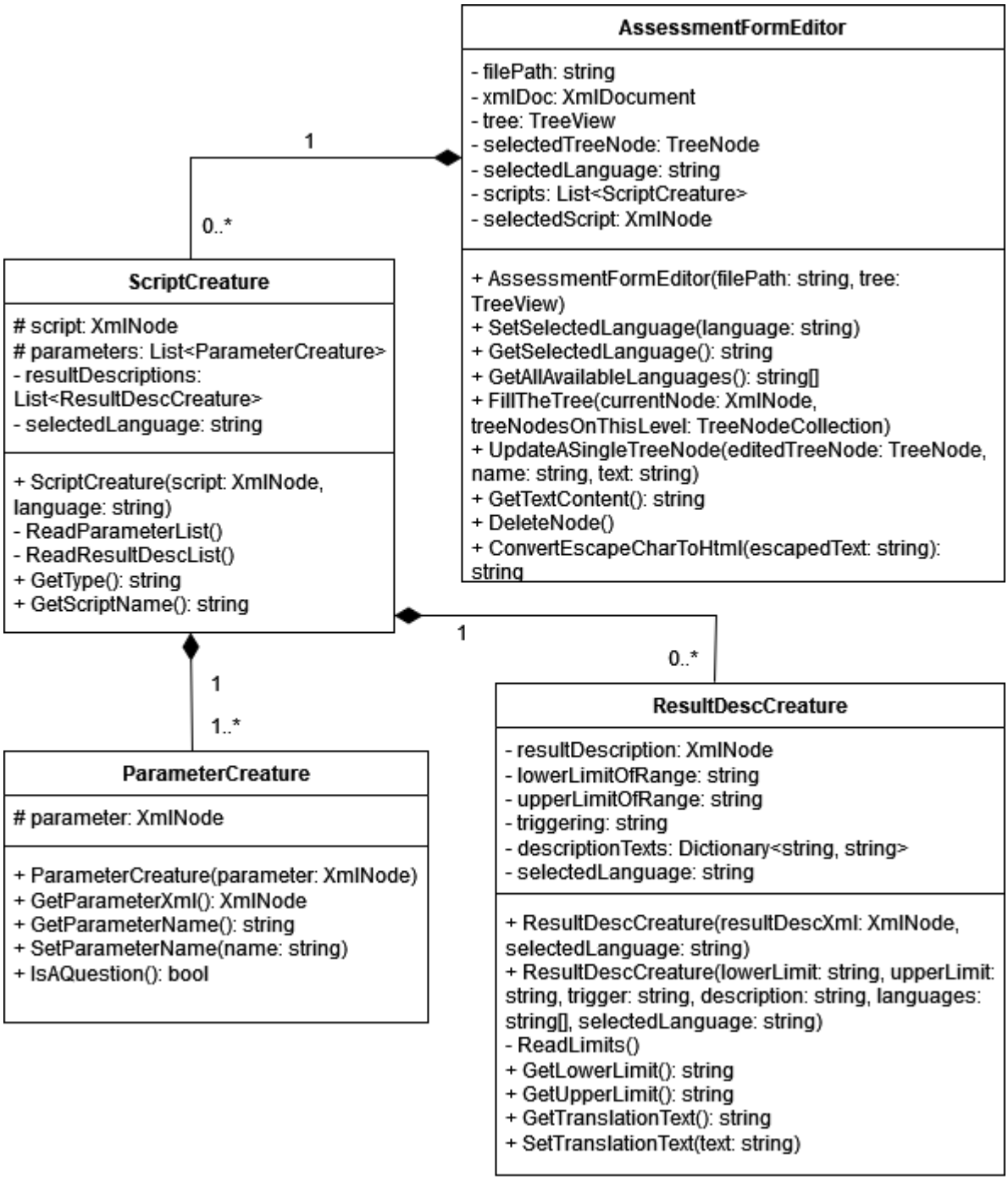
### 5.3.2 Luokkien välinen työnjako

Etenkin luokkien työnjaosta päättäminen osoittautui yllättävän vaikeaksi. Olin ajatellut hallitsevani aihealueen, mutta tilanne ei ollutkaan niin selkeä, kun käsillä oleva ohjelma oli huomattavasti laajempi kuin kursseilla toteutetut pienet harjoitusohjelmat.

Haasteita tuottivat esimerkiksi tilanteet, joissa pieni apuluokka tarvitsisi tietoa, joka oli vain isomman, hierarkiassa ylempänä olevan luokan käytettävissä. Tästä on esimerkkitapaus jäljempänä, JumpCreature-luokan kuvauksen yhteydessä.

## 5.4 Editorin toiminta

Instrumenttieditorissa on XML-dokumentin käsittelystä yleisesti vastaava iso luokka sekä joukko pienempiä luokkia, joista jokainen vastaa tiettyä osaa instrumentissa. Osa näistä luokista on esitetty kuviossa 4. Valtaosa metodeista sekä kaikki uusimmat luokat on tilanpuutteen vuoksi jätetty kaaviosta pois. Kuviosta puuttuvat myös sovelluksen ikkunoita edustavat Form-luokat, joissa niissäkin on paljon sovelluksen toiminnan kannalta keskeisiä metodeja.




KUVIO 4. Instrumenttieditorin luokkakaavio

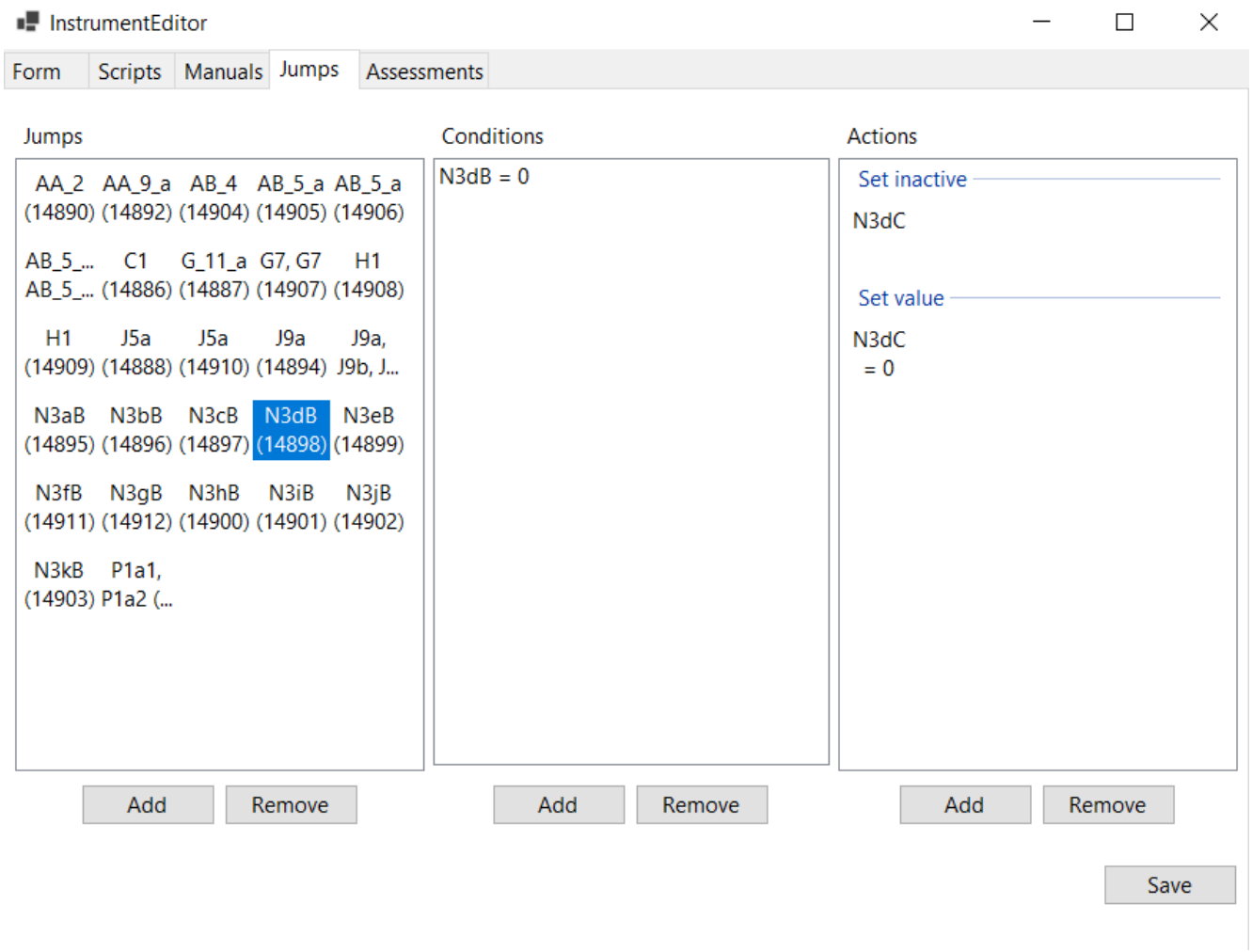
Esimerkiksi ScriptCreature-luokan olio vastaa yksittäistä instrumentin skriptiä (kuten BMI-mittaria). Skriptejä on instrumenteissa vaihteleva määrä, ja kukin niistä puolestaan sisältää vaihtelevan määrän parametreja (ParameterCreature) sekä mahdollisesti myös tuloskuvaustekstejä (ResultDescCreature).

Ihan kaikille instrumentin osasille – kuten lomakkeelle tai sen kysymyksille – ei omia luokkiaan tarvinnut tehdä, mutta muun muassa skriptien hallintaa niille omistettu oma luokka helpotti huomattavasti. Koodin kirjoittaminen ja kokonaisuuden hahmottaminen olisi ollut huomattavasti hankalampaa ilman näitä luokkia. Jatkossa luokat selkeyttävät myös koodin ylläpitoa.

XML-dokumentista vastaava iso luokka on nimeltään hieman hämäävästi AssessmentFormEditor. Sen vastuualue on kuitenkin paljon laajempi kuin pelkkä arviointilomakkeen muokkaus. Nimesin luokan työn varhaisessa vaiheessa, jolloin sen tehtäväkenttä oli vielä kapeampi. Karkeasti ottaen tämän luokan vastuulla on nykyisellään kaikki instrumentti-XML-tiedoston luku ja muokkaus sekä muiden luokkien hallinta (pois lukien ikkunoita edustavat Form-luokat). Luokan yhtenä jäsenmuuttujana on XmlDocument-olio.

### 5.5 Tarkastelussa JumpCreature-luokka

Yksi instrumenttieditorin uusimpia lisäyksiä on loogisille hypyille omistettu välilehti. Yksittäiseen hyppyyn (JumpCreature) liittyy yksi tai useampi ehto (JumpCondition), joiden on täytyttävä, jotta hyppy toteutetaan (JumpAction). Kuvasta 7 näkyy, miten hypyt esitetään editorin käyttöliittymässä.




KUVA 7. Instrumenttieditorin hyppyvälilehti

### 5.5.1 Nimien näyttäminen käyttöliittymässä

Lomakekysymysten ja skriptien nimet (esimerkiksi A1 tai BMI) ovat XML:ssä tallennettuina kysymys- ja skriptielementtien attribuutteihin, joten lomake- ja skriptivälilehdellä näiden nimien näyttäminen on varsin suoraviivaista. Hyppyjen kohdalla tilanne on kuitenkin hieman monimutkaisempi.

XML-dokumentin hyppypuussa on kyllä hyppyjen käsittelyn kannalta oleelliset tiedot, kuten hypyn kysymysten ja skriptien tunnisteet (esimerkiksi 123456). Siellä ei kuitenkaan ole tietoa näiden kysymysten ja skriptien nimistä. Tämä ei sellaisenaan ole kovin käyttäjäystävällistä, sillä ilman nimitietoja käyttäjä näkisi hyppyjä tarkastellessaan käyttöliittymässä kysymysten ja skriptien nimien sijasta pitkiä tunnisteita, jotka ole kovin kuvaavia.

Nimitietoja ei kannata tallentaa hyppyjen puuhun, sillä se olisi turhaa toistoa, ja silloin joutuisi myös erikseen huolehtimaan siitä, että hyppyjen nimitiedot pysyvät ajan tasalla. Nimet kun saattavat muuttua. On siis parasta hakea tuoreimmat nimitiedot alkuperäisestä lähteestä eli lomake- tai skriptipuusta.

Nimitietojen hakeminenkaan ei ole yksioikoista. Käyttöliittymässä näytettävä hyppydata tulee JumpCreature-olioista – ja sen kautta myös JumpCondition- ja JumpAction-olioista – ja nämä oliot puolestaan muodostetaan hyppy-XML:n perusteella. AssessmentFormEditor-luokalla taasen on tiedot kysymysten ja skriptien nimistä sekä metodit, jotka mahdollistavat nimien hakemisen lomake- ja skriptipuista tunnisteen perusteella. Ongelmana kuitenkin oli, ettei JumpCreature-luokasta käsin voi kutsua AssessmentFormEditor-luokan metodeita.

Ratkaisin nimiongelman lopulta siten, että JumpCreature-oliot luodaan kaksivaiheisesti. Kun käyttäjä avaa Jumps-välilehden, AssessmentFormEditor-luokan GetJumps-metodi luo JumpCreature-olion XmlDocument-oliossa olevan hyppydatan perusteella. Sen jälkeen se hakee hyppyyn tarvittavat nimidatan ja lähettää sen hyppyoliolle, joka puolestaan lähettää tiedot edelleen jäsenolioilleen (vaihteleva määrä JumpCondition- ja JumpAction-olioita) väliaikaista tallennusta ja käyttöliittymässä näyttöä varten. Seuraavissa koodikatkelmissa (koodi 6 ja koodi 7) on tästä pelkistetty kuvaus.

```
private XmlDocument xmlDoc;
private List<JumpCreature> jumpList;
```

KOODI 6. Osa AssessmentFormEditor-luokan jäsenmuuttujista

Koodissa 6 on kaksi tämän esimerkin kannalta keskeistä AssessmentFormEditor-luokan jäsenmuuttujaa. XmlDocument-olioita tarvitaan XML:n käsittelyyn, List<JumpCreature>-listaa puolestaan instrumentin kaikkien loogisten hyppyjen tallentamiseen. Koodissa 7 on esitetty hyppylistan täyttäminen ennen nimitietojen hakemista.

```
jumpList = new List<JumpCreature>();
XmlNode jumpContainer = xmlDoc.SelectSingleNode(”//instru-
ment/jumps”); // XmlDocument-olioon ladatussa XML-tiedos-
tossa jump-elementit ovat jumps-elementin sisällä

// Jokaista jump-elementtiä kohden luodaan JumpCreature-
olio, joka lisätään jumpList-listalle
for (int i = 0; jumpContainer.ChildNodes.Count; i++)
{
```

```
                XmlNode jumpXml = jumpContainer.ChildNodes[i];
                JumpCreature jumpC = new JumpCreature(jumpXml);
                jumpList.Add(jumpC);

                // Nimitietojen hakeminen tässä kohdin
        }
```

KOODI 7. Hyppyolioiden luonti AssessmentFormEditor-luokan GetJumps-metodissa

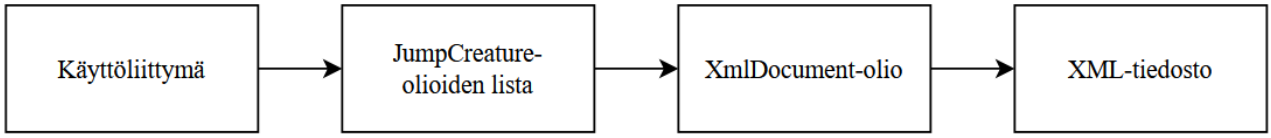
Kun JumpCreature-olio on lisätty jumpList -listalle, siinä on tiedot jäsenolioiden kysymysten ja mahdollisten skriptien tunnisteista, mutta ei vielä nimitietoja. Nimitietojen hakemista varten tunnisteet kerätään List<string>-tyyppisiin muuttujiin, minkä jälkeen niitä vastaavat kysymysten ja skriptien nimet haetaan XmlDocument-olion lomake- ja skriptipuista Dictionary<string, string>-muuttujaan. Dictionary-muuttujassa olevat nimitiedot lähetetään JumpCreature-oliolle, joka päivittää nimet niitä vastaavien JumpCondition- ja JumpAction-olioiden string-tyyppiseen jäsenmuuttujaan. Tämän jälkeen hypyt näytetään käyttöliittymässä. (Mainittakoon, että tämä ottaa aikansa – parisen sekuntia – ja Jumps-välilehti onkin instrumenttieditorin välilehdistä hitaimmin latautuva.)

Hyppyjen nimitiedot ovat JumpCreature-olioihin tallennettuna editorin käytön ajan.

### 5.5.2 Muutosten tallennus loogisiin hyppyihin

Omalta osaltaan päänvaivaa aiheutti kysymys siitä, missä vaiheessa ja millä tavoin muutokset kannattaa tallentaa eteenpäin, kun käyttäjä tekee muutoksia käyttöliittymässä. Ei olisi käytännöllistä, jos jokainen pieni muutos tallennettaisiin välittömästi XML-tiedostoon saakka – etenkään, kun tämä käytännössä käsittää koko lähdetiedoston ylikirjoittamisen.

Kuviossa 5 on esitetty tallennettavan informaation kulku loogisten hyppyjen tapauksessa. Kun käyttäjä lisää uusia hyppyjä tai muokkaa niitä joko Jumps-välilehdellä tai siihen liittyvissä ponnahdusikkunoissa, tieto tallentuu JumpCreature-olioihin. Kun käyttäjä painaa Jumps-välilehden Save-painiketta, tieto tallentuu JumpCreature-olioiden listalta XmlDocument-olioon. Kun käyttäjä sulkee editorin, ilmestyy valintaikkuna, jossa kysytään, haluaako käyttäjä tallentaa muutokset XML-tiedostoon. Vasta tässä vaiheessa muutokset siirtyvät lähdetiedostoon. Muutokset on siis vielä tässä vaiheessa mahdollista tarvittaessa perua.




KUVIO 5. Informaation kulku, kun muutoksia tallennetaan loogisiin hyppyihin

Tässä vaiheessa voisin sivuhuomautuksena mainita, ettei instrumenttieditorissa ainakaan toistaiseksi ole erillisiä peruutuspainikkeita. En pitänyt niitä kovin korkeana prioriteettina, sillä muutokset pystyy perumaan olemalla tallentamatta niitä editorin sulkemisvaiheessa. Tehtyjen muutosten menettämisen puolestaan pystyy välttämään tekemällä silloin tällöin välitallennuksia.

Kuvion 5 kaaviossa tätä ei ole kuvattuna, mutta tärkeää on myös käyttöliittymän päivitys muutosten jälkeen, jotta käyttäjä näkee tekemiensä muutosten vaikutukset. Loogisten hyppyjen tapauksessa muutoksia ei voi päivittää käyttöliittymään ennen tai samanaikaisesti kuin olioihin, sillä hypyt ovat aakkosjärjestyksessä, ja niihin tehdyt sisältömuutokset voivat vaikuttaa tähän järjestykseen. Yksittäisten hyppyjen muokkauksen yhteydessä JumpCreature-olioiden lista järjestetään Sort-metodin avulla. Vasta tämän jälkeen päivittynyt järjestys ja tehdyt muutokset näytetään käyttöliittymässä.

# 6 XML-SKEEMA

Kun instrumentti-XML-rakenne oli alustavasti suunniteltu ja toteutettu, se oli hyväksytettävä ja lyötävä lukkoon. Koska instrumentti-XML-tiedosto oli katselmoitavaksi pitkä, sain tehtäväkseni kirjoittaa sitä vastaavan XML-skeeman. Tätä skeemaa voitaisiin hyödyntää rakenteen katselmoinnin lisäksi XML-dokumentin validoinnissa ja sen varmistamisessa, että XML, backend ja mobiilisovellus ovat kaikki yhteensopivia (saman XML-skeeman mukaisia).

XML-skeeman avulla voidaan määritellä XML-dokumentin sisältöön liittyviä rajoituksia. Sitä voidaan käyttää myös dokumentin muodolliseen kuvailuun. Dokumentin tarkistusta skeeman avulla kutsutaan validoinniksi. Tässä työssä käytettiin skeeman kirjoittamiseen kieltä W3C XSD (XML Schema Definitions), mutta myös muita kielivaihtoehtoja skeemojen kirjoittamiseen on. (Schema.)

XML-skeema oli itselleni uusi tuttavuus, joten sen laatiminen vaati jonkin verran selvitystyötä ja opettelua.

## 6.1 Haasteita

Arviointilomakkeen rakenteeseen liittyy lukuisia sääntöjä – esimerkiksi se, mitä lapsisolmuja tietyillä solmuilla voi tai täytyy olla ja kuinka monta, sekä se, mitä attribuutteja näillä solmuilla on ja mitä arvoja ne voivat saada. Tällaisten sääntöjen kuvaaminen skeemassa oli melko suoraviivaista. Ongelmia tuotti kuitenkin se, että säännöistä huolimatta lomakkeen rakenne on varsin joustava. Tämän joustavuuden kuvaaminen oli paikoin haastavaa.

Yksi esimerkki haasteellisesta tapauksesta ovat lomakkeen description-solmut, jotka voivat sisältää sekä toisia description-solmuja että kysymyssolmuja millaisena yhdistelmänä tahansa. Instrumentti-XML:n rakenteen olin toteuttanut siten, että description-solmun ensimmäisenä lapsisolmuna ovat sen tekstit eri kielillä. Tämän jälkeen tulevat varsinaiset description- ja kysymys-lapsisolmut siinä järjestyksessä, jossa ne lomakkeella ovat. Skeeman ehtoja oli vaikea laatia siten, että solmun omien kuvaustekstien olisi oltava lapsisolmujen joukossa ensimmäisinä mutta että sen jälkeen tuleva sisältö olisi joustava. Jätin lopulta ensimmäisen vaatimuksen XML-skeemasta pois, vaikka käytännössä toteutankin XML:n vaatimuksen mukaisesti – tätä vain ei validoida.

Toinen haasteellinen tapaus oli constant-attribuutin käsittely. Yksittäisen kysymyksen näkökulmasta se on valinnainen attribuutti, joka voi joko olla tai puuttua. Tietynarvoinen constant-attribuutti voi esiintyä saman instrumentin lomakkeella vain kertaalleen. Eli attribuutin arvon on oltava uniikki, ja tämä tarkastelu on ulotettava koko lomakkeen laajuudelle. Voi olla, että onnistuin kirjoittamaan tämän ehdon skeemaan (kuvan 9 riveillä 1038 – 1041), mutta aika näyttää toimiiko tämä.

### 6.2 Lopullinen XML-skeema

Tekemääni XML-skeemaa ei vielä toistaiseksi kukaan ole ehtinyt katselmoimaan, mutta siitä huolimatta hukkaan vaiva ei suinkaan mennyt. Skeeman suurin hyöty itselleni tuli siitä, että pystyin käyttämään sitä spekseinä instrumenttieditoria työstäessäni. Tiedostossa on kätevästi yhteen paikkaan koottuna kaikki instrumentti-XML:n sisältö. Tiedostosta näkee nopeasti elementtien hierarkian, kunkin elementin attribuutit ja attribuuttien mahdolliset arvot.

Myös XML-skeeman kirjoitusprosessista – ei pelkästään sen lopputuloksesta – oli kiistatonta hyötyä. Se nimittäin pakotti käymään instrumentti-XML:n perinpohjaisesti läpi kohta kohdalta, yksi elementti ja attribuutti kerrallaan. Samalla tulin huomanneeksi useita kehityskohteita XML-dokumentin rakenteessa. Ainakin omalla tahollani tulin siis katselmoineeksi tuotetun XML-rakenteen XSD-tiedostoa laatiessani.

Kuvassa 8 on esitettynä kirjoittamani XML-skeeman alku. Valtaosa tiedostosta on elementtien ja attribuuttien määritelmiä. Tästäkin tiedostosta koetin väliotsikoiden ja järjestyksen avulla tehdä mahdollisimman helppolukuisen.

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">


<!-- *********** -->
<!-- DEFINITIONS -->
<!-- *********** -->


<!-- Translatable texts -->

<xs:complexType name="translationtexttype">
    <xs:simpleContent>
        <xs:extension base="xs:string">
            <xs:attribute name="lang" type="xs:string" use="required"/>
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>

<xs:group name="translationgroup">
    <xs:sequence>
        <xs:element name="text" type="translationtexttype" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
</xs:group>

<xs:complexType name="translationtype">
    <xs:sequence>
        <xs:group ref="translationgroup"/>
    </xs:sequence>
</xs:complexType>


<!-- *** INSTRUMENT *** -->


<!-- Attribute value restrictions (instrument) -->

<!-- Is the instrument supplement? -->
<xs:attribute name="issupplement">
    <xs:simpleType>
        <xs:restriction base="xs:integer">
            <xs:minInclusive value="0"/> <!-- false -->
            <xs:maxInclusive value="1"/> <!-- true -->
        </xs:restriction>
```

KUVA 8. XSD-tiedoston alku

Kuvassa 9 on skeeman loppuosa. Tiedoston viimeisillä riveillä on kuvattu instrumentti-XML:n runko. Kaiken tarkemman sisällön ja elementtien hierarkian näkee näitä rivejä edeltävistä määritelmistä.

```
1014  <!-- ************ -->
1015  <!-- DECLARATIONS -->
1016  <!-- ************ -->
1017
1018  <xs:element name="instrument">
1019      <xs:complexType>
1020          <xs:sequence>
1021              <xs:element name="longname" type="translationtype"/>
1022              <xs:element name="shortname" type="translationtype"/>
1023              <xs:element name="root" type="rootnodetype"/> <!-- Assessment form -->
1024              <xs:element name="scripts" type="scriptcontainertype"/> <!-- Scripts -->
1025              <xs:element name="jumps" type="jumpcontainertype"/> <!-- Logical jumps -->
1026              <xs:element name="asstypes" type="assessmentcontainertype"/> <!-- Assessment types
1027              <xs:element name="asstypetriggers" type="triggercontainertype"/> <!-- Assessment t
1028              <xs:element name="copyfilters" type="copyfiltercontainertype"/> <!-- Copy filters
1029          </xs:sequence>
1030          <xs:attribute name="versionid" type="xs:integer"/>
1031          <xs:attribute name="versionname" type="xs:string"/>
1032          <xs:attribute name="shortversionname" type="xs:string"/>
1033          <xs:attribute name="versionnumber" type="xs:string"/>
1034          <xs:attribute ref="issupplement"/>
1035          <xs:attribute name="activestart" type="xs:string"/> <!-- Could be "xs:date", if the fo
1036          <xs:attribute name="activeend" type="xs:string"/> <!-- Could be "xs:dateTime", if the
1037      </xs:complexType>
1038      <xs:unique name="unique_constant"> <!-- Constants should be unique within an instrument --
1039          <xs:selector xpath="node[@type='question']"/>
1040          <xs:field xpath="@constant"/>
1041      </xs:unique>
1042  </xs:element>
1043
1044  </xs:schema>
1045
```

KUVA 9. XSD-tiedoston loppu

# 7 ARVIOINTI JA POHDINTA

Olen tämän työn myötä saanut syvennettyä osaamistani etenkin olio-ohjelmoinnin suhteen. Kokonaisuuden hallinta ja luokkien yhteispelin suunnittelu oli välillä yllättävän haastavaa. Ohjelmalla on paljon eri tehtäviä, ja luokkien vastuualueiden raja on paikoin häilyvä. Tulin huomanneeksi, etteivät kaikki tehtäväjaot käytännössä vain toimineet, vaikka jako aluksi paperilla olisikin vaikuttanut järkevältä.

Työn myötä aloin myös ihan uudella tavalla arvostaa olioviittauksia. Laajasta, koko XML-dokumenttia edustavasta XmlDocument-oliosta pystyy hakemaan tietyn solmun ja tallentamaan sen erilliseen XmlNode-olioon. Kun tätä XmlNode-oliota muokkaa, muutos päätyy samalla XmlDocument-olioon, sillä XmlNode on edelleen osa sitä. Esimerkiksi arviointilomaketta edustavaa TreeView-puurakennetta piirtäessä jokaiseen TreeNode-solmuun voi Tag-ominasuudeksi tallentaa vastaavan XmlNode-olion. Kun käyttäjä valitsee tietyn TreeNode-solmun muokattavaksi, muokkauksen kohteena olevaa XML-elementtiä ei tarvitse erikseen käydä etsimässä.

Muutamaan otteeseen tulin huomanneeksi, että olin yrittänyt toteuttaa jotain asiaa turhan monimutkaisesti, vaikka suoraviivaisempikin tapa olisi ollut. Opetuksena tästä on, ettei pitäisi liian varhaisessa vaiheessa lyödä lukkoon, millä tavoin aikoo jonkin asian toteuttaa.

Pienimuotoisia haasteita tuotti myös omien rönsyilevien muistiinpanojen ja suunnitelmien hallinta. Tämän suhteen auttoi se, että noin työn puolivälin tienoilla järjestin keskeisimmät muistiinpanot siististi aihealueiden mukaan omiin tiedostoihinsa ja kansioihinsa.

# 8 YHTEENVETO JA JOHTOPÄÄTÖKSET

Työ kattoi alkuosan projektista, jonka tavoitteena on siirtää yrityksen arviointivälineet tietokantatauluista XML-dokumentteihin. Työn osatehtäviin kuuluivat instrumentti-XML-rakenteen suunnittelu, XML-dokumentin tuottamiseen käytettävien Delphi-metodien kirjoittaminen, XML-skeeman kirjoitus sekä instrumentti-XML:n muokkaukseen käytettävän instrumenttieditorin toteutus C#:lla.

Opinnäytetyön aikana siirtymäprojektin kaikkia osa-alueita on saatu työstettyä. Instrumentti-XML ja sen tuottavat Delphi-metodit ovat ainakin alustavasti valmiit, samoin XML-skeema. Editori on vielä keskeneräinen, mutta sitä on päässyt testaamaan, ja se etenee hyvää vauhtia, vaikka loppuosa ei tähän kirjoitelmaan ehdikään.

Projekti jatkuu opinnäytetyön jälkeen suunnitelmien mukaisesti. Instrumenttieditoriin on vielä lisättävä uusia välilehtiä ja toimintoja, jotta se kattaisi kaikki instrumentti-XML:n osat. Sen toiminta on myös testattava perusteellisesti. Lisäksi on kirjoitettava sellaiset uudet Delphi-metodit, jotka hakevat instrumentin tiedot tietokantataulukoiden sijasta XML-dokumentista (kuvio 2). Tämän jälkeen siirtymäprosessia päästään kunnolla testaamaan.

**LÄHTEET**

*A tour of C# - Overview*. Microsoft. 2023. Saatavissa: https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/. Viitattu 27.2.2023.

Embarcadero. 2022. *Language Overview - RAD Studio*. Saatavissa: https://docwiki.embarcadero.com/RADStudio/Alexandria/en/Language_Overview. Viitattu 27.2.2023.

Embarcadero. 2019. *Procedures and Functions (Delphi) - RAD Studio*. Saatavissa: https://docwiki.embarcadero.com/RADStudio/Alexandria/en/Procedures_and_Functions_(Delphi). Viitattu 1.3.2023.

IBM. *What is a relational database?* Saatavissa: https://www.ibm.com/topics/relational-databases. Viitattu 27.2.2023.

Oracle. *What Is a Relational Database*. Saatavissa: https://www.oracle.com/database/what-is-a-relational-database/. Viitattu 22.4.2023.

*Schema*. W3C. Saatavissa: https://www.w3.org/standards/xml/schema. Viitattu 17.2.2023.

Vitec Raisoft. *Terveyttä ja toimintakykyä*. Saatavissa: https://www.raisoft.com/fi/arviointivalineet/. Viitattu 27.2.2023.

W3C. 2008. *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. Saatavissa: https://www.w3.org/TR/xml/. Viitattu 22.4.2023.

*XML Essentials*. W3C. Saatavissa: https://www.w3.org/standards/xml/core. Viitattu 25.2.2023.

*XmlDocument Class (System.Xml)*. Microsoft. Saatavissa: https://learn.microsoft.com/en-us/dotnet/api/system.xml.xmldocument?view=net-7.0. Viitattu 31.3.2023.