Mikko Viitaila

MUSICXML:STÄ MUSIIKIKSI

Tietotekniikan pro gradu -tutkielma

Ohjelmistotekniikan linja

17.11.2003

Jyväskylän yliopisto

Tietotekniikan laitos

Tekijä: Mikko Viitaila

Yhteystiedot: Sähköposti `mikko.viitaila@cc.jyu.fi` ja puh. 050 505 6220

Työn nimi: MusicXML:stä musiikiksi

Title in English: From MusicXML to Music

Työ: Pro gradu -tutkielma

Sivumäärä: 93 + 8

Linja: Ohjelmistotekniikka

Teettäjä: Jyväskylän yliopisto, tietotekniikan laitos

Avainsanat: MusicXML, rakenteinen, kuvauskieli, XML, SAX, MIDI, elektroniset soittimet, äänisynteesi, Java Sound API

Keywords: MusicXML, structured, description language, XML, SAX, MIDI, electronic instruments, sound synthesis, Java Sound API

Tiivistelmä: Tutkielmassa tutkitaan MusicXML-määrittelykielen soveltuvuutta yleiskäyttöiseksi musiikin kuvauskieleksi. Erityisesti paneudutaan sen soveltuvuuteen lähdekieleksi auditiivisen esityksen muodostamisessa. Tutkimuksen tueksi kehitetään sovellus, jolla pyritään muuntamaan reaaliaikaisesti MusicXML-dokumentin sisältämä musiikki-informaatio soivaksi esitykseksi. Lisäksi arvioidaan kehitettävän sovelluksen kehitystyössä käytettyjen työkalujen soveltuvuutta auditiivisen esityksen tuottamiseen.

Abstract: This thesis examines the suitability of MusicXML as a general purpose description language for musical information. The main concern is on its capabilities on serving as a source language for generating auditive representation. To fully examine this, a prototype of a software that produces audio output from MusicXML documents in real-time was developed. In addition, the tools used in the development process and their suitability for producing the auditive representation are also studied.

Termiluettelo

DOM	*Document Object Model*, W3C:n suositus rajapinnaksi, joka mahdollistaa XML-dokumenttien käsittelyn puumaisessa tietorakenteessa.
DTD	*Document Type Definition*, XML-dokumenttien muotokuvaus. DTD-dokumenteilla määritellään XML-dokumenttien looginen rakenne.
HTML	*HyperText Markup Language*, rakenteinen WWW-sivujen kuvauskieli. Sillä määritellään WWW-sivujen rakenne, sisältö ja ulkoasu.
HTTP	*Hypertext Transport Protocol*, hypertekstidokumenttien siirtoon Internetissä tarkoitettu protokolla.
HumDrum	Musiikki-informaation kuvaamisessa käytetty yleiskäyttöinen kuvauskieli.
ISO	*International Organization for Standardization*, kansainvälinen vuonna 1947 perustettu standardointijärjestö.
J2SE	*Java 2 Standard Edition*, Java 2 -ohjelmointikielen perusversio.
JAXP	*Java API for XML Parsing*, eri XML-jäsentimien käytön mahdollistava yhtenäinen rajapinta.
Kern	HumDrum:iin kuuluva länsimaisessa taidemusiikissa käytetyn viivastonotaation kuvauskieli.

MIDI	*Musical Instrument Digital Interface*, vuonna 1983 julkaistu kansainvälinen liitäntästandardi, jota käytetään elektronisten soitinten liittämisessä toisiinsa.
MuseData	Länsimaisessa taidemusiikissa käytetyn viivastonotaation kuvauksessa käytetty musiikin kuvauskieli.
MusicXML	XML 1.0:n mukaisesti suunniteltu rakenteinen musiikki-informaation kuvauskieli.
NIFF	*Notation Interchange File Format*, nuotinkirjoitusohjelmistojen väliseen tiedonsiirtoon tarkoitettu musiikin kuvauskieli.
SAX	*Simple API for XML*, avoimeen lähdekoodiin perustuva rajapintamääritys, joka mahdollistaa tapahtumapohjaisen XML-dokumenttien käsittelyn.
SGML	*Standard Generalized Markup Language*, vuonna 1986 julkaistu asiakirjastandardi. Merkintätapa, jolla kuvataan tekstien rakennetta standardiesitysmuotona.
SMDL	*Standard Music Description Language*, SGML-standardiin perustuva musiikin kuvauskieli.
URI	*Uniform Resource Identifier*, yksikäsitteinen tietoverkossa sijaitsevan resurssin tunniste.
URL	*Uniform Resource Locator*, resurssien sijainnin määrittelyyn tietoverkossa, esimerkiksi Internetissä, tarkoitettu standardi.
W3C	*World Wide Web Consortium*, WWW:n ja siihen liittyvien standardien kehittämisorganisaatio.

Xerces	Apache-projektin kehittämä SAX ja DOM -rajapinnat toteuttava XML-jäsennin.
XHTML	*Extensible HyperText Markup Language*, XML 1.0:n mukaisesti suunniteltu versio HTML 4.0 -sivunkuvauskielestä.
XML	*Extensible Markup Language*, W3C:n kehittämä rakenteinen dokumenttien kuvauskieli.
XSD	*XML Schema Definition*, XML-dokumenttien rakenteen määrittelyyn käytetty kieli.
XSL	*Extensible Stylesheet Language*, XML-tyylien kuvauksessa käytettävä kieli.
XSLT	*Extensible Stylesheet Language Transformations*, XML-tyylien muunnoksiin erikoistunut W3C-suositus.

Sisältö

Kuvat

Taulukot

Esimerkit

1 Johdanto

Musiikin tarkka kuvaaminen on ollut vaikeaa kautta historian. Musiikki-informaation monimutkaisuus ja tulkinnanvaraisuus aiheuttavat hankaluuksia sen kuvaamisessa myös tietokoneita ja -järjestelmiä varten. Lukuun ottamatta länsimaisessa kulttuurissa yleisesti käytettyä graafista nuottikirjoitusta eli viivastonotaatiota mitään yhteistä standardia musiikin kuvaamiselle ei nykypäivään mennessä ole vielä olemassa. Musiikkialan ammattilaisten keskuudessa yhteiselle standardille on suuri tarve, ja tällaiseksi yleiskäyttöiseksi musiikin kuvauskieleksi on pyrkimässä uusi XML-standardien mukaisesti määritelty MusicXML-määrittelykieli.

1.1 Tutkielman tavoite

Tutkielman tavoite on tutkia MusicXML-määrittelykielen soveltuvuutta yleiskäyttöiseksi musiikin kuvauskieleksi. Erityisesti paneudutaan tutkimaan sen soveltuvuutta lähdekieleksi auditiivisen esityksen muodostamisessa. Tutkimuksen tueksi tutkielmassa kehitetään sovellus, joka pyrkii muodostamaan reaaliaikaisesti soivan esityksen käyttäen MusicXML-määrittelykieltä lähdekielenä. Lisäksi tavoitteena on tutkia sovelluksen kehitystyössä käytettävien tekniikoiden soveltuvuutta käyttötarkoitukseensa.

1.2 Tutkielman rakenne

Tutkielma jakautuu yhdeksään lukuun, joista ensimmäisessä on tutkielman johdanto. Varsinainen tutkielma alkaa musiikki-informaation luonteen ja ominaisuuksien tarkastelulla luvussa 2. Samalla kuvataan musiikin kuvaamisessa yleisesti käytettyä graafista viivastonotaatiota ja sen keskeisiä elementtejä. Lisäksi pohditaan musiikin tarkassa kuvaamisessa kohdattuja vaikeuksia ja lopuksi kerrotaan lyhyesti olemassa olevista musiikin kuvauskielistä. Kolmannessa luvussa kuvataan reaaliaikaisen musiikki-informaation ja elektronisten soitinten yhteenliittämisessä käytettävää MIDI-standardia tarkastellen sen käyttötarkoituksia ja toimintaa, sen hyviä ja huonoja puolia sekä sen toimintaa ja syntaksia. Luvussa 4 kuvataan XML 1.0 -standardia sekä tutkielmassa käsiteltävää MusicXML-määrittelykieltä. MusicXML-määrittelykielen osalta paneudutaan sen historiaan, käyttötarkoituksiin ja

kuvataan sen syntaksia ja tyyppimäärityksiä. Luvussa 5 esitellään kehitettävän sovelluksen kehitystyössä käytettäviä työkaluja ja tekniikoita sekä niiden toimintaa. Kuudennessa luvussa esitetään kehitettävän sovelluksen vaatimusmäärittely, ja luvussa 7 sovelluksen arkkitehtuuri ja toiminta. Luvussa 8 pohditaan ja arvioidaan tehdyn työn tuloksia sekä käytettyjen työkalujen ja tekniikoiden soveltuvuutta kehitetyn sovelluksen kannalta. Viimeisessä yhdeksännessä luvussa on tutkielman yhteenveto.

2 Musiikki informaationa

Luvussa pohditaan musiikin olemusta ja luonnetta informaationa. Luvussa 2.1 kuvataan musiikin eri olomuotoja ja luvussa 2.2 sen kuvaamisessa käytettyä *viivastonotaatiota*. Luvussa 2.3 pohditaan musiikin kuvaamisessa kohdattuja vaikeuksia ja luvussa 2.4 rakenteisten dokumenttien mahdollisuuksia musiikki-informaation kuvaamiseen. Lopuksi luvussa 2.5 kerrotaan olemassa olevista musiikin kuvauskielistä. Musiikin osalta luvussa keskitytään perinteiseen länsimaiseen taide- ja populaarimusiikkiin ja sen kuvaamisessa käytettyyn viivastonotaatioon.

2.1 Musiikin kolme olomuotoa

Musiikki on terminä erittäin abstrakti käsite. Ensisijaisesti sen voidaan ajatella olevan korvin kuultavaa ääntä. Ääntäkin voi olla monenlaista, mutta musiikilla on tiettyjä ominaisuuksia, jotka erottavat sen muista äänen lajeista. Musiikki on organisoitua, yleensä etukäteen harkittua, yhtä aikaa soivien äänten muodostamaa äänimassaa, josta muodostuu korvin aistittava kokonaisuus. Se koostuu ajallisesti etenevästä *rytmistä*, peräkkäin esiintyvien äänten muodostamista *melodioista* ja yhtä aikaa soivien äänten muodostamasta *harmoniasta*. Näiden eri elementtien sulautuessa ja organisoituessa yhteen muodostuu yhtenäinen kokonaisuus, jota kutsutaan musiikiksi. [Oksala, 1971, sivu 11]

Korvin kuultava elävä musiikki syntyy sitä soittavien muusikoiden soittimista. Ennen korvin kuultavaa esitystä musiikin on kuitenkin täytynyt olla olemassa jossain muodossa sen luojan, säveltäjän, ja edelleen sitä soittavien soittajien ajatuksissa. Musiikin säveltäjän on täytynyt jollain keinolla siirtää ajatuksensa ja mielikuvansa sävellyksensä musiikillisesta sisällöstä soittajille. Tähän tarkoitukseen länsimaisessa kulttuurissa käytetään nuottikirjoitusta, jonka voidaan ajatella olevan säveltäjän ja soittajien välinen yhteinen kieli. Nuottikirjoitus on siis eräänlainen musiikillinen koodi, jota kaikkien sitä käyttävien osapuolten on ymmärrettävä.

Musiikin voidaan ajatella esiintyvän ainakin seuraavissa kolmessa olomuodossa [Selfridge-Field, 1997, sivu 7]:

1. Musiikin säveltäjän mielikuva sävellyksestään eli *musiikillinen sisältö*
2. Jollakin kaikkien osapuolten ymmärtämällä tavalla *koodattu esitys*
3. Korvin kuultava musiikki eli *auditiivinen esitys*.

Muusikon tuottamien fyysisten liikkeiden ja eleiden voidaan ajatella olevan neljäs olomuoto tähän listaan, mutta sillä ei varsinaisesti ole mitään tekemistä itse kuultavan musiikin kanssa.

Tämä tutkielma keskittyy näistä kolmesta olomuodosta kahteen jälkimmäiseen, koodattuun ja auditiiviseen esitykseen. Musiikki voidaan tallentaa näissä molemmissa olomuodoissa. Auditiivinen esitys voidaan tallentaa äänitallenteena, joka on kuunneltavan musiikin ensisijainen tallennusmuoto. Siinä soittajien soittama musiikki tallennetaan esimerkiksi nauhalle tai CD-levylle, jolta se voidaan kuunnella korvin kuultavana, auditiivisena esityksenä kerta toisensa jälkeen. [Tiensuu, 1991, sivu 264]

Äänitallenne ei kuitenkaan ole soveltuvin tallennusmuoto uuden auditiivisen esityksen tuottamisen ohjeeksi. Koodattu esitys onkin soveltuvampi juuri tähän tarkoitukseen. Sen tarkoitus on ensisijaisesti musiikki-informaation tallentaminen kaikkien sitä käyttävien osapuolten ymmärtämässä muodossa, josta heidän on mahdollista tuottaa uusi soitettu, auditiivinen esitys [Oksala, 1971, sivu 27]. Koodatun esityksen tarkoitus on siis tallentaa aikasidonnainen ja katoava auditiivinen esitys pysyvässä, visuaalisessa muodossa, josta se voidaan uudelleen muuttaa soivaksi auditiiviseksi esitykseksi. [Tiensuu, 1991, sivu 264]

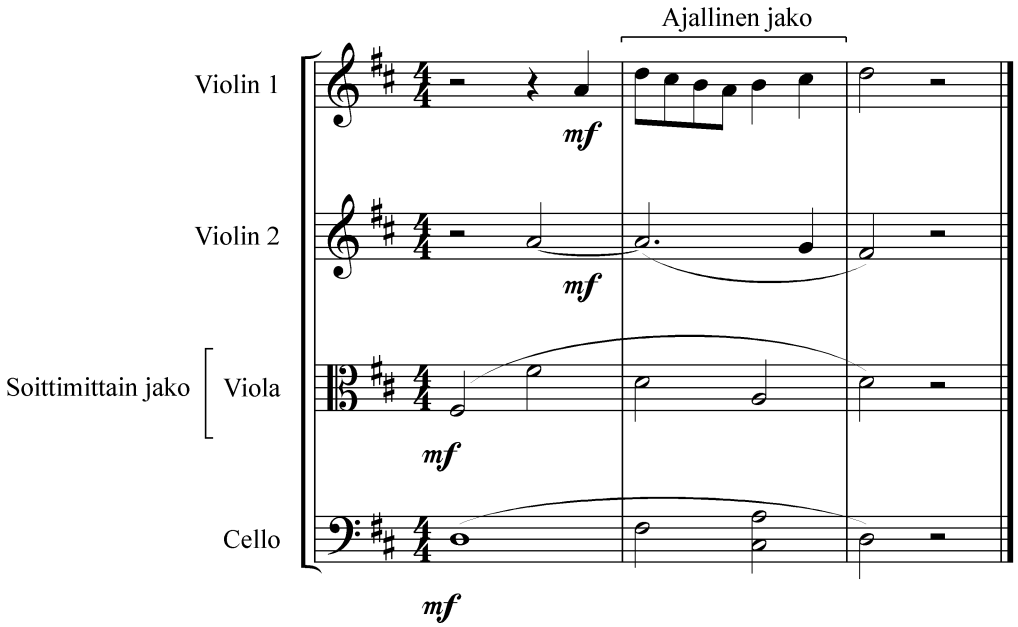
Yleisin koodatun esityksen tallennusmuoto on nuottikirjoitus, joka on säilynyt lähes muuttumattomana yli 300 vuotta. Nuottikirjoituksen menestyksestä kertoo myös se, että moni kuvaustapa on yrittänyt korvata sen siinä onnistumatta [Byrd, 2001, sivu 241]. Esimerkiksi vuonna 2001 Yhdysvaltojen kongressin kirjastossa (*Library of Congress*) oli yli 6 miljoonaa nuottikirjoituksen avulla koodattua musiikkikappaletta [Byrd, 2001, sivu 239]. Nuottikirjoituksen yleispiirteitä ja elementtejä käsitellään seuraavassa luvussa.

2.2 Nuottikirjoituksen elementit

Nuottikirjoituksen tarkoitus on toimia musiikillisena koodina, jolla osoitetaan erilaisia musiikillisia tapahtumia. Tällaisia tapahtumia voivat olla esimerkiksi soitettavan sävelen

sävelkorkeus, kesto, voimakkuus ja sävy. Perinteisessä länsimaisessa nuottikirjoituksessa näitä tapahtumia on kuvattu *viivastonotaatiolla*, joka koostuu erilaisista elementeistä, kuten *nuottiviivastoista*, *tahdeista*, *nuoteista*, *tauoista*, *nuottiavaimista*, *lyriikoista* ja *esitysmerkinnöistä* (kuva 1). [Oksala, 1971, sivu 27; Tiensuu, 1991, sivu 264]

Kuvassa 1 on esimerkki erään musiikkikappaleen kolmesta tahdista. Kappale on sävelletty jousikvartetille, johon kuuluu neljä soitinta; kaksi viulua, alttoviulu ja sello.



Kuva 1. Musiikkinotaation eri elementit.

2.2.1 Nuottiviivastot, intervallit ja oktaavialat

Viivastonotaatiossa kukin soitin esitetään omalla nuottiviivastollaan, joka koostuu viidestä vaakasuorasta viivasta. Nuottiviivaston alussa määritellään kappaleen *sävellaji*, *tahtilaji* ja muut mahdolliset koko kappaleeseen vaikuttavat merkinnät.

Länsimainen taidemusiikki perustuu seitsemään erikorkuiseen säveleen, joita kutsutaan *juurisäveliksi*. Juurisävelten nimet ovat alhaalta ylöspäin *c*, *d*, *e*, *f*, *g*, *a* ja *h*. Yhtä siirtymää sävelestä toiseen kutsutaan *sävelaskeleeksi* ja etäisyyttä kahden sävelen välillä *intervalliksi*. Intervalleja ovat pienimmästä suurimpaan *priimi*, *sekunti*, *terssi*, *kvartti*, *kvintti*, *seksti*,

septimi, *oktaavi*, *nooni, desiimi*, *undesiimi*, *duodesiimi*, *tredesiimi* ja *kvartdesiimi* [Apajalahti, 1990, sivu 84]. Etäisyyttä kahdella samannimisellä juurisävelellä on siis kahdeksan sävelaskelta eli oktaavi. Termiä oktaavi käytetään myös erottelemaan erikorkuiset samannimiset sävelet toisistaan. Oktaavit ovat numeroitu siten, että kunkin oktaavin ensimmäinen sävel on c. Esimerkiksi pianon keski-c on nimeltään *yksiviivainen c*, seuraava c *kaksiviivainen c* ja niin edelleen. Alaspäin liikuttaessa vuorossa ovat *pieni*, *suuri*, *kontra-* ja *subkontraoktaavi*. Oktaavien nimistä puhuttaessa käytetään myös termiä *oktaaviala* [Oksala, 1971, sivut 21-25].

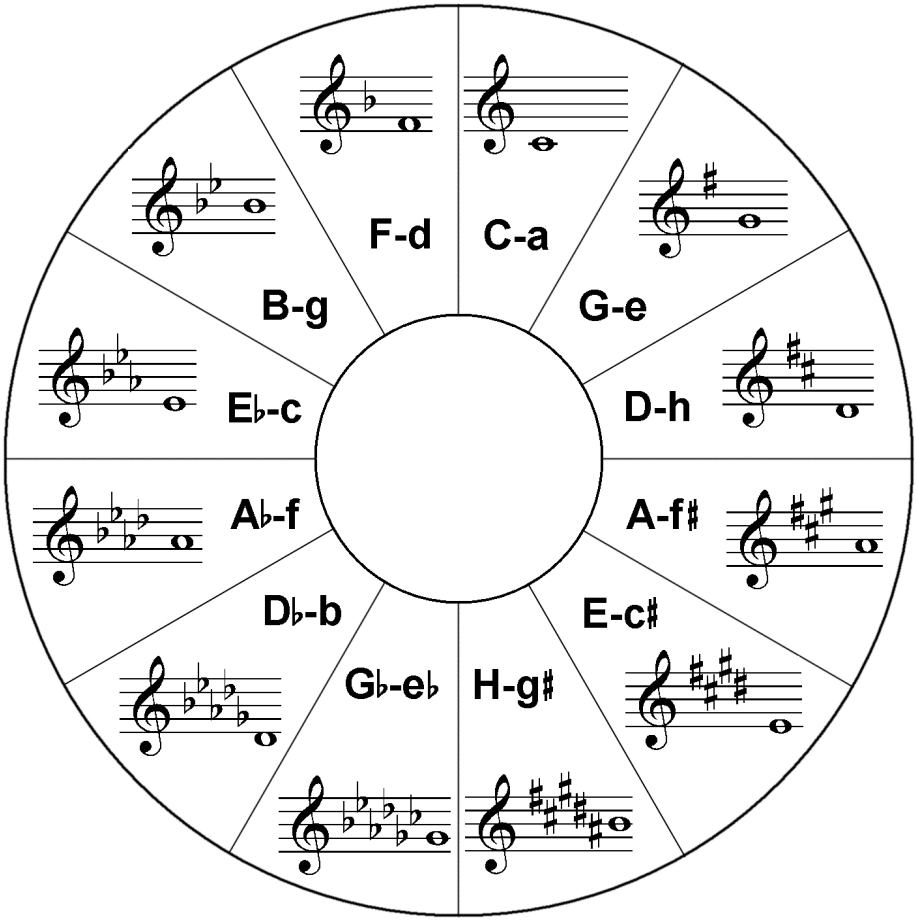
2.2.2 Nuottiavaimet ja sävellajit

Jokaisen nuottiviivaston alussa on *nuottiavain*, josta käytetään myös termiä *klaavi*. Nuottiavaimen avulla määritellään nuottiviivastolla sijaitsevien nuottien soiva sävelkorkeus [Oksala, 1971, sivu 31]. Esimerkiksi kuvan 1 kappaleen viulujen nuottiavain on nimeltään *G-nuottiavain*, joka osoittaa yksiviivaisen oktaavin g-sävelen paikan nuottiviivastolla. Vastaavasti sellon nuottiavain on nimeltään *F-nuottiavain* ja se osoittaa pienen oktaavin f-sävelen paikan nuottiviivastolla. Nuottiavain määrää sävelen sijainnin nuottiviivastolla sille viivalle, jota sen keskusta osoittaa [Oksala, 1971, sivu 32]. Esimerkissä vielä huomiotta jäänyt alttoviulun nuottiavain on nimeltään *C-nuottiavain*.

Lisäksi on olemassa instrumentteja, jotka soivat eri sävelkorkeudella kuin mitä niiden soitettavaksi nuottiviivastolla on kirjoitettu. Tällaisia soittimia kutsutaan *transponoiviksi* soittimiksi, joita ovat esimerkiksi klarinetti, saksofoni ja trumpetti. [Takala, 1992, sivut 164-165]

Viivastonotaatiossa sävellaji ilmoitetaan kappaleen alussa tarvittavalla määrällä *korotus-* tai *alennusmerkkejä*. Korotus- ja alennusmerkeillä muutetaan sävelen korkeutta puoli sävelaskelta ylös- tai alaspäin. Nuottiviivaston alkuun sijoitetuista korotus- ja alennusmerkeistä käytetään nimitystä *etumerkintä* ja ne ovat voimassa koko nuottiviivaston ajan ellei toisin ilmoiteta. Vaikka etumerkintä merkitään vain yhden oktaavin säveliin, se korottaa tai alentaa osoittamansa sävelet jokaisessa oktaavialassa. [Oksala, 1971, sivu 55]

Länsimaisessa taidemusiikissa sävellajit kulkevat viiden sävelaskeleen eli kvintin välein ylös- tai alaspäin siten, että jokaisella kerralla lisätään yksi korotus- tai alennusmerkki. Tästä sävellajien kierrosta käytetään termiä *kvinttiympyrä*. Sävellajeja on sekä *duureja* että *molleja* siten, että samalla etumerkinnällä varustettuja sävellajeja kutsutaan *rinnakkaissävellajeiksi* [Oksala, 1971, sivu 64]. Kuvassa 2 on havainnollistettu sävellajien kiertoa kvinttiympyrässä. Sävellajien nimissä duuria merkitään suurella kirjaimella ja mollia pienellä.



Kuva 2. Sävellajien nimet ja etumerkinnät kvinttiympyrässä.

2.2.3 Tahtilajit ja sävelten kestot

Viivastonotaatiossa kukin nuottiviivasto on jaettu *tahtilajin* mukaisesti ajallisesti yhtä pitkiin osiin (lukuun ottamatta tahtilajin vaihdoksia), joita kutsutaan tahdeiksi. Kappaleen

tahtilaji ilmoitetaan ensimmäisen tahdin alussa, joka kuvan 1 esimerkkikappaleessa on neljä neljäsosaa. Tämä merkintä tarkoittaa sitä, että kappaleen peruspulssi on neljäsosanuotin mittainen ja jokainen tahti koostuu neljästä neljäsosanuotin pituisesta pulssista. [Oksala, 1971, sivut 101-102]

Sävel merkitään nuottiviivastolle nuotilla, jonka vertikaalinen sijainti nuottiviivastolla kertoo sävelen sävelkorkeuden [Oksala, 1971, sivu 71]. Nuotti voi sijaita nuottiviivaston viivan päällä, viivojen välissä tai viivojen loppuessa ylä- tai ala-apuviivojen päällä tai välissä [Oksala, 1971, sivut 28-29]. Sävel voi olla myös korotettu tai alennettu joko etumerkinnän osoittaman sävellajin mukaisesti tai tilapäisellä etumerkillä merkittynä. Tilapäistä korotusta tai alennusta kutsutaan *kromaattiseksi* korotukseksi tai alennukseksi ja se on voimassa kyseisen tahdin loppuun [Oksala, 1971, sivu 57].

Monella soittimella voidaan soittaa myös monta eri säveltä samanaikaisesti ja tällaisia soittimia kutsutaan *polyfonisiksi* soittimiksi. Esimerkiksi kuvan 1 esimerkkikappaleen toisessa tahdissa sello soittaa samanaikaisesti sekä sävelen *cis* eli korotetun *c*:een että sävelen *a*.

Nuotin ulkoasu kertoo sävelen soivan keston, josta käytetään termiä *aika-arvo*. Nuottien aika-arvot puolittuvat pienempiin aika-arvoihin siirryttäessä. Nuottien nimet ovat suurimmasta aika-arvosta alkaen *kokonuotti*, *puolinuotti*, *neljäsosanuotti*, *kahdeksasosanuotti*, *kuudestoistaosanuotti* ja niin edelleen. Pienin yleisesti käytössä oleva aika-arvo on *sadaskahdeskymmeneskahdeksasosanuotti*. Lisäksi käytössä on harvinaisempi *kaksoiskokonuotti*, josta käytetään myös termiä *brevis* [Oksala, 1971, sivut 68-70]. Pisteellä nuotin oikealla puolella merkitään sen ajallisen keston lisäämistä puolella. Tahtiviivojen yli kestävien sävelien merkitsemisessä nuotti kirjoitetaan tarvittavan pituisena uudelleen ja nuotit yhdistetään toisiinsa *sitomiskaarella*. Sitomiskaarta käytetään myös helpottamaan rytmityksen hahmottamista tai poikkeuksellisten aika-arvojen merkitsemiseen [Oksala, 1971, sivut 82-84]. Kuvassa 3 on havainnollistettu saman kestoisen sävelen erilaisia merkitsemistapoja.

Kuva 3. Saman sävelen soivan keston erilaisia merkitsemistapoja.

Tauoilla on luonnollisesti vain kesto, joka määräytyy taukomerkin ulkoasun perusteella. Taukojen kestot vastaavat nuottien aika-arvoja [Oksala, 1971, sivu 79]. Yleensä tauot ja nuotit pyritään merkitsemään siten, että koko tahdin kesto täyttyy. Kuvan 1 esimerkkikappaleen ensimmäisen viulun sävelien ja taukojen kestot ovat vasemmalta oikealle *puolitauko*, *neljäsosatauko*, *neljäsosanuotti*, neljä *kahdeksasosanuottia*, kaksi *neljäsosanuottia*, *puolinuotti* ja puolitauko. Vastaavat sävelkorkeudet ovat *a*, *d*, *cis*, eli korotettu *c*, *h*, *a*, *h*, *cis* ja *d*.

Kuvassa 1 esiintyvät muut merkinnät ovat esitysmerkintöjä, joita ovat esitysvoimakkuuden määritys (*mf* = *mezzoforte* eli *melko voimakkaasti*) ja melodialinjojen hahmottamisen helpottamiseksi käytettävät kaaret.

2.3 Musiikki-informaation kuvaaminen

Luvussa pohditaan millaista musiikki on informaationa sekä ongelmia musiikin täsmällisessä formaalissa kuvaamisessa.

2.3.1 Visuaalinen vs. auditiivinen informaatio

Perinteisessä viivastonotaatiossa eli visuaalisessa esityksessä on paljon informaatiota, joka ei vaikuta mitenkään auditiiviseen esitykseen eli soivaan kuulokuvaan [Bellini, 2001, sivu 84]. Vastaavasti auditiivisessa esityksessä on paljon asioita, joiden merkitseminen visuaaliseen esitykseen on hankalaa, ellei kokonaan mahdotonta. Esimerkiksi nuotinkirjoitusohjelmistojen tarvitsemat tiedot musiikin visuaalisessa esityksessä esiintyvistä nuottien varsista ja ryhmittelyistä palkkien avulla eivät mitenkään vaikuta saman kappaleen auditiiviseen esitykseen, sillä soivan musiikin käsittelyyn keskittyville ohjelmistoille pelkkä tieto sävelen kestosta on täysin riittävä [Selfridge-Field, 1997, sivut 8 ja 14].

Monet auditiiviseen esitykseen vaikuttavat tekijät ovat kulkeneet suullisena perintönä vuosisatojen ajan. Esimerkiksi tietyn aikakauden sävellykselle saattaa olla olemassa tiettyjä esitystapoja, joiden merkitsemiseksi nuottikuvaan ei ole olemassa mitään keinoja [Selfridge-Field, 1997, sivut 12 ja 15]. Lisäksi monet säveltäjät ovat elinaikanaan itse johtaneet sävellyksiensä harjoituksia ja esityksiä ja täten päässeet suullisesti antamaan soittajille neuvoja ja ohjeita kappaleen tulkinnasta [Tiensuu, 1991, sivu 267].

2.3.2 Absoluuttinen vs. suhteellinen informaatio

Eräs vaikeus musiikki-informaation kuvaamisessa on erottelu absoluuttisen ja suhteellisen informaation välillä sekä näiden välisen tärkeysjärjestyksen määrittäminen. Suhteellisella informaatiolla tarkoitetaan sellaista tiedon esitystapaa, joka on riippuvainen siitä kontekstista jossa se esitetään. Absoluuttinen informaatio taas on sellaista informaatiota, jonka sisältö tai merkitys ei muutu vaikka se irrotettaisiin kontekstistaan. Musiikki-informaatiossa on kuitenkin paljon sekä suhteellista että absoluuttista informaatiota. Tietokoneiden näkökulmasta ajatellen absoluuttinen informaatio on helpommin ja suoraviivaisemmin käsiteltävissä kuin suhteellinen informaatio. Lisäksi osa musiikki-informaation tulkinnasta on riippuvainen kontekstistaan. [Selfridge-Field, 1997, sivu 11; Tiensuu, 1991, sivu 266]

Esimerkiksi kappaleen *tempo* eli esitysnopeus voidaan ilmaista nuottikuvassa sanallisesti, esimerkiksi *Allegro* eli *vilkkaasti* tai *Largo* eli *raskaasti*, tai numeerisen tarkasti, esimerkiksi 120 neljäsosanuottia minuutissa. Todellinen tempo on kuitenkin aina viimekädessä musiikin esittäjän päätettävissä. Samoin on asian laita sointivoimakkuuksien tulkinnan suhteen. Esimerkiksi suhteellinen termi *forte* eli *voimakkaasti* tarkoittaa absoluuttisena voimakkuutena täysin eri arvoa kitaran ja tuuban sointivoimakkuutena [Tiensuu, 1991, sivu 266].

Jo yksittäisen nuotinkin soivan sävelkorkeuden määrittely on riippuvainen kontekstistaan. Sävelkorkeus voidaan määritellä jo kappaleen sävellajissa kappaleen alussa tai paikallisesti kyseisen nuotin yhteydessä. Lisäksi nuotin soiva kestokin voidaan määritellä usealla eri tavalla. Nuottikuvassa esiintyvä neljäsosanuotti on voitu määritellä soitettavaksi nopeana ja lyhyenä (*staccato*), jolloin sen soiva kesto kuulokuvassa onkin paljon lyhyempi kuin mitä

nuotin aika-arvo antaa ymmärtää. Auditiivisessa esityksessä olisi siis paremminkin kyse lyhyistä nuoteista ja pitkistä tauoista niiden välillä. [Selfridge-Field, 1997, sivu 10]

2.4 Musiikin rakenteisuus ja sisäiset hierarkiat

Luvussa kuvataan musiikki-informaation sisäisiä hierarkioita sekä erilaisia lähestymistapoja sen kuvaamiseen rakenteisina dokumentteina.

2.4.1 Musiikki-informaation sisäiset hierarkiat

Musiikki on sisäiseltä rakenteeltaan vahvasti hierarkkinen. Täten sen kuvaaminen rakenteisina dokumentteina tuntuu luontevalta ja ilmeisen suoraviivaiselta. Tällaisia hierarkkisia rakenteita musiikissa voivat olla esimerkiksi:

- Nuottiviivastot sivulla
- Tahdit nuottiviivastolla
- Soinnut tahdissa
- Nuotit soinnussa.

Kuten yllä olevasta listasta käy ilmi, musiikin sisäiset rakenteet sopivat varsin hyvin kuvattavaksi rakenteisten dokumenttien tarjoamin keinoin [Castan, 2001a, sivu 100]. Poikkeus vahvistaa kuitenkin säännön ja näin on myös musiikissa. Esimerkiksi melodialinjan merkitsemisessä käytetty kaari voi ylittää tahtiviivan tai rivinvaihto voi esiintyä kesken tahdin. Tällaisten musiikki-informaation sisäisen hierarkian rikkovien poikkeusten tuomat ongelmat voidaan kuitenkin ratkaista rakenteisten dokumenttien tarjoamin keinoin merkkaamalla ne esimerkiksi elementtien ominaisuuksiksi. [Castan, 2001a, sivu 101]

2.4.2 Musiikki-informaation kaksi ulottuvuutta

Musiikki on informaationa kaksiulotteista. Kuvassa 1 havainnollistettiin tätä kaksiulotteisuutta nuottiesimerkin avulla. Esimerkin neljälle instrumentille sävelletty kappale voidaan jakaa osiin horisontaalisesti soittaja tai instrumentti kerrallaan tai vertikaalisesti tahti kerrallaan. Horisontaalisesti jaettuna kukin instrumentti koostuu useasta tahdista ja vertikaalisesti jaettuna kukin tahti koostuu useasta instrumentista. [Selfridge-Field, 1997, sivu 12]

Erilaiset musiikkiohjelmistot käsittelevät musiikki-informaatiota eri näkökulmista, joista toinen lähestymistapa voi olla toiselle ohjelmistolle parempi kuin toinen. Tähän päivään mennessä useimmat musiikki-informaation tallentamiseen tarkoitetut kuvauskielet ovat valinneet näistä ulottuvuuksista toisen ja siten heikentäneet omaa soveltuvuuttaan musiikki-informaation käsittelemiseen eri näkökulmista. Olemassa olevia musiikin kuvauskieliä esitellään tarkemmin seuraavassa luvussa.

Edellisissä kappaleissa esitellyt asiat ja ongelmat tulee ottaa huomioon suunniteltaessa yleiskäyttöistä musiikki-informaation kuvaamiseen tarkoitettua kuvauskieltä. Tässä tutkielmassa tutkittu MusicXML-määrittelykieli on hyvä esimerkki yrityksestä ratkaista nämä ongelmat.

2.5 Musiikin kuvauskieliä

Luvussa kuvataan lyhyesti tässä tutkielmassa tutkittavan MusicXML-määrittelykielen suunnittelun lähtökohtina olleita kahta akateemista musiikin kuvauskieltä. Luvussa 2.5.1 kuvataan *HumDrum*:a ja luvussa 2.5.2 *MuseData*:a. Muista olemassa olevista harvinaisemmista musiikin kuvauskielistä mainittakoon *SMDL* (*Standard Music Description Language*, ISO 10743) [Newcomb, 1991], *NIFF* (*Notation Interchange File Format*) [Grande, 1997] ja *abc* [Walshaw, 2003], joita ei tässä tutkielmassa käsitellä.

2.5.1 HumDrum

HumDrum:n tarkoitus on toimia musiikki-informaation tutkimiseen ja analysointiin tarkoitettuna yleiskäyttöisenä musiikin kuvauskielenä. Se on alun perin kehitetty juuri musiikin tutkijoiden käyttöön ja on siten laajalti tutkijoiden käytössä ympäri akateemisen maailman. HumDrum on helposti laajennettavissa ja sillä on teoreettisesti mahdollista kuvata mitä tahansa sarjallisessa muodossa esitettävissä olevaa informaatiota kuten esimerkiksi tanssiaskelia, tunnetiloja tai vaikkapa muusikon liikkeitä esityksen aikana. [Huron, 1997, sivu 375]

HumDrum:iin kuuluva *Kern* on erityinen länsimaisessa viivastonotaatiossa esiintyvien elementtien kuvaustapa. HumDrum-kuvauskieliset tiedostot ovat tekstitiedostoja, joissa

kaikki yhtäaikaiset musiikilliset tapahtumat esitetään samalla rivillä ja ajallisesti eteenpäin liikuttaessa siirrytään seuraavalle riville [Huron, 1997, sivu 377]. HumDrum on valinnut musiikin kahdesta ulottuvuudesta toisen, ajallisen jaon [Anthony, 2001, sivu 164].

HumDrum Toolkit on HumDrum-kuvauskielisten tiedostojen analysointiin ja editointiin tarkoitettu työkaluohjelmapaketti, joka sisältää noin 70 erilaista apuohjelmaa HumDrum-tiedostojen käsittelyyn. Suurin osa HumDrum:a käyttävistä ohjelmistoista on komentoriviohjelmia ja kuvauskielellä on kirjoitettu arviolta 9000 musiikkikappaletta. [Huron, 1997, sivut 398-399]

HumDrum:n käyttöä yleiskäyttöisenä musiikin kuvauskielenä on vaikeuttanut sen alkuperäinen akateeminen käyttötarkoitus sekä sitä tukevien ohjelmistojen vähäinen määrä. Lisäksi kielen syntaksi on vaikealukuista ja täten erittäin virhealtista ihmisen muokattavaksi.

2.5.2 MuseData

Kuten HumDrum myös MuseData on tarkoitettu musiikillisten käsitteiden kuvaamiseen, keskittyen kuitenkin enemmän musiikkikappaleiden loogisen rakenteen esittämiseen. Sen ensisijainen tarkoitus on toimia eräänlaisena ohjelmistoriippumattomana lähdekielenä, josta konvertoimalla muihin formaatteihin saadaan eri tarkoituksiin ja ohjelmistoille sopivampia tiedostoformaatteja [Hewlett, 1997a, sivut 402-403]. MuseData:n ensisijaisia käyttötarkoituksia ovatkin tiedon tallennus ja säilöntä esimerkiksi tietokannoissa, eikä sitä ole tarkoitettu minkään tietyn erityissovelluksen käyttöön [Hewlett, 1997a, sivu 445].

MuseData-tiedostot ovat tekstitiedostoja, joissa musiikilliset käsitteet esitetään kukin omalla rivillään aikajärjestyksessä. Lisäksi kukin instrumentti esitetään omassa erillisessä tiedostossaan [Hewlett, 1997a, sivu 403]. MuseData on siis tehnyt valintansa musiikki-informaation kahden ulottuvuuden välillä soittimittain tai nuottiviivastoittain jaettuna, toisin kuin HumDrum [Anthony, 2001, sivu 164].

Kuten HumDrum:n myös MuseData:n syntaksi on vaikealukuista ja altista inhimillisille virheille. Lisäksi sen käyttötarkoitus ainoastaan lähdekieleksi muille formaateille ja infor-

maation tallentamiselle ovat vaikeuttaneet sen yleistymistä yleiskäyttöisenä musiikin kuvauskielenä.

3 MIDI

Luvussa kuvataan elektronisten soitinten väliseen tiedonsiirtoon tarkoitettua MIDI-standardia (*Musical Instrument Digital Interface*). Luvussa 3.1 kerrotaan MIDI:n historiasta ja luvussa 3.2 käyttötarkoituksista ja -ympäristöistä. Luvussa 3.3 pohditaan MIDI:n hyviä ja huonoja puolia. Lopuksi luvussa 3.4 kuvataan MIDI:n toimintaa ja syntaksia.

3.1 Historia

1980-luvun alussa elektronisten soitinten ja syntetisaattoreiden käytön lisääntyessä muusikoilla ilmeni tarvetta niiden yhteenliittämiselle. Studiotekniikan kehittyessä useine raitoineen ja lisääntyvine mahdollisuuksineen studiossa tuotettuja esityksiä alkoi olla mahdotonta esittää konserttitilanteessa. Muusikot halusivat mahdollisuuden ohjata useampaa soitinta tai syntetisaattoria yhdeltä koskettimistolta saadakseen aikaiseksi studiotuotosta vastaavan auditiivisen lopputuloksen. Niinpä vuoden 1982 NAMM-messuilla (*National Association of Music Merchants*) esitettiin ajatus yhteisen liitäntästandardin kehittämisestä elektronisten soitinten väliseen tiedonsiirtoon. [Romanowski, 1990, sivut 13-16; Hirvi, 1995, sivut 12-13]

Messujen jälkeen suurimmat laitevalmistajat, kuten Roland, Yamaha, Korg ja Kawai, aloittivat yhteisen liitännän suunnittelutyön. Puolentoista vuoden suunnittelutyön jälkeen elokuussa 1983 esiteltiin MIDI 1.0 -spesifikaatio. Määritystä julkistettaessa korostettiin sen olevan ainoastaan suositus, jota valmistajat voivat joko noudattaa tai olla noudattamatta. Valmistajat ottivat kuitenkin määrityksen nopeasti käyttöön ja nykyisin MIDI on elektronisten soitinten liitäntästandardi ja käytännössä ainoa sellainen. [Romanowski, 1990, sivut 13-16]

3.2 Käyttötarkoitukset ja -ympäristöt

MIDI merkitsee elektronisten soitinten digitaalista tiedonsiirtoväylää. Toisin sanoen se on erilaisten digitaalisten soitinten ja laitteiden yhteinen kieli, jolla ne voivat kommunikoida toistensa kanssa. Kommunikoinnilla tarkoitetaan sitä, että laitteet lähettävät ja vastaanottavat keskenään MIDI-viestejä ja toimivat näiden viestien mukaisesti [Tuominen, 1989, sivu

13]. MIDI:n avulla ei siis siirretä ääntä tai musiikkia, vaan ainoastaan tietoa erilaisista musiikillisista tapahtumista [Doan, 1994, sivu 10; Hirvi, 1995, sivu 14-15]. Seuraavissa kappaleissa esitellään MIDI:n erilaisia käyttötarkoituksia, -ympäristöjä ja sovelluksia.

Kahden tai useamman elektronisen soittimen yhteenliittäminen: Eräs MIDI:n alkuperäisistä käyttötarkoituksista oli mahdollistaa eri laitteiden ohjaaminen toisella laitteella. MIDI:n avulla esimerkiksi kaksi tai useampia syntetisaattoreita voidaan liittää yhteen. Soittajan soittaessa yhtä näistä laitteista myös muut laitteet tuottavat soitetun äänen. Näin mahdollistetaan useiden erilaisten äänimaailmojen tuottaminen ilman, että kaikkien soitinten pitäisi olla soittajan ulottuvilla. [Romanowski, 1990, sivu 10]

Samplerit, rumpukoneet ja sekvensserit: Muusikoiden käyttöön on kehitetty lukuisia elektronisia soittimia ja apuvälineitä. *Samplereilla* voidaan äänittää ääninäytteitä, joita voidaan edelleen käyttää ”instrumentteina” eli sointiääninä. *Rumpukoneita* käytetään synteettisinä rytmisoittimina. Eri äänilähteitä voidaan yhdistää ja synkronoida *sekvenssereillä*, jotka nykyisin on pääosin korvattu tietokoneissa toimivilla sekvensseriohjelmistoilla. Näillä kaikilla laitteilla on yksi yhteinen tekijä: MIDI. MIDI:n avulla laitteet voidaan kytkeä samaan järjestelmään ja synkronoida toimimaan saumattomasti yhdessä. Muusikko voi ohjata laitteita yhdestä ja samasta paikasta ja näin saada aikaan toimivan kokonaisuuden. [Tuominen, 1989, sivut 15-17; Penfold, 1990, sivut 102-114]

MIDI ja tietokoneet: Kun MIDI:ä 1980-luvun alkupuolella suunniteltiin, tietokoneet olivat vielä liian kalliita ja harvinaisia henkilökohtaisiksi työvälineiksi. Tietokoneita ei myöskään silloin juuri käytetty musiikin tekemiseen. Siksi MIDI:n yleistymistä tietokoneissa 1990-luvulle tultaessa ei osattu aavistaa. Äänikorttien myötä musiikin tekeminen on kuitenkin mahdollista tietokoneella, ja nykyisin äänikortit on varustettu MIDI-liitännällä ja MIDI-syntetisaattorilla. Näin ollen ennen vain ammattilaisten käyttöön tarkoitetut työkalut ovat jokaisen saatavilla. Tämä kehitys on edesauttanut MIDI:n saavuttamaa suosiota huomattavasti. [Doan, 1994, sivu 10; Tuominen, 1989, sivu 13]

1990-luvulla tietokone on ottanut tehtäväkseen monia tehtäviä, joita ennen hoitivat erilliset MIDI-laitteet. Nykyaikaiset tietokoneiden käyttöliittymät ovat parempia näiden sovellusten käyttämiseen kuin yksittäiset muutaman rivin näytöllä varustetut laitteet. Tietokoneella

voidaan esimerkiksi käyttää sekvensseri-ohjelmistoa ja ohjata samalla muita ulkoisia MIDI-laitteita yhdestä ja samasta joustavasta käyttöliittymästä. [Hirvi, 1995, sivu 39-40]

Standard MIDI File 1.0: MIDI-tietoa voidaan myös tallentaa tietokoneella MIDI-tiedostoihin, jotka voidaan avata ja ottaa uudelleen käsittelyyn myöhemmin. Tästä formaatista käytetään nimeä *Standard MIDI File 1.0* [Romanowski, 1990, sivu 97]. MIDI-tiedostot sisältävät MIDI-laitteiden käyttöön tarkoitettua aikaleimattua musiikillista informaatiota. Ne siis edustavat tallennettuja MIDI-viestejä lisättynä aikasidonnaisella informaatiolla [Hewlett, 1997b, sivu 43]. Tämä tiedostomuoto nousi standardin asemaan ja on ollut tähän päivään mennessä ainoa tapa siirtää tietoa eri musiikkiohjelmistojen välillä [Good, 2001a].

General MIDI Instrument Specification: MIDI-syntetisaattoreissa jokaisella eri sointiäänellä, eli instrumentilla, on oma numeronsa. Jatkossa sanalla instrumentti tarkoitetaan juuri MIDI-laitteen käyttämää sointiääntä ja erillisestä, oikeasta instrumentista, käytetään termiä MIDI-laite. Alkuperäinen MIDI 1.0 -spesifikaatio jätti paljon asioita määrittelemättä, sillä tarkoitus oli tehdä standardista mahdollisimman avoin. Tämä johti valmistajakohtaisiin ratkaisuihin, jotka vaikeuttivat osaltaan MIDI-laitteiden yhteensopivuutta [Romanowski, 1990, sivu 17]. Laitevalmistajat saivat esimerkiksi itse päättää eri instrumenttien numeroinnin omissa laitteissaan. Käyttäjän piti siis tietää, mikä haluamansa instrumentin numero kussakin laitteessa on [Hirvi, 1995, sivut 91-92]

Vuonna 1991 julkaistiin *General MIDI Instrument Specification*, joka määrittelee kaikille standardia noudattaville MIDI-laitteille 128 yhteistä instrumenttia [Hewlett, 1997b, sivu 44]. Kaikkien standardia noudattavien MIDI-laitteiden instrumenttien numeroinnit vastaavat siis toisiaan ja esimerkiksi pianon instrumenttinumero on kaikissa laitteissa sama. Käyttäjän ei siis enää tarvitse erikseen selvittää pianon instrumenttinumeroa vastaanottavasta MIDI-laitteesta käyttääkseen laitetta. [Hewlett, 1997b, sivut 68-69]

General MIDI Instrument Specification -standardi on osoittautunut kuitenkin riittämättömäksi, sillä erilaisia soittimia on olemassa huomattavasti enemmän kuin 128. Lisäksi standardin määrittelyssä on annettu omia numeroita saman soittimen eri sointiväreille ja siten turhaan monimutkaistettu sen käyttöä esimerkiksi nuotinkirjoitusohjelmistoissa. [Hewlett, 1997b, sivut 68-69]

General MIDI Instrument Specification -standardista puhuttaessa voidaan puhua lyhyemmin *General MIDI* -standardista, tai vielä lyhyemmin käyttää lyhennettä *GM*. General MIDI Instrument Specification -standardista on julkaistu versio 2.0, josta käytetään lyhennettä *GM2*. Se korjaa ensimmäisen version puutteita, mutta sitä tukevia MIDI-laitteita ei ole vielä paljon markkinoilla. [MIDI Manufacturers Association, 2003]

3.3 Hyvät ja huonot puolet

MIDI:llä on hyvät ja huonot puolensa. Alkuperäiseen tarkoitukseensa, musiikillisten tapahtumien välittämiseen ja auditiivisen esityksen tuottamiseen, MIDI on erinomainen esimerkki riittävän avoimesta ja laajennettavissa olevasta kuvauskielestä, joka on kestänyt vuosikymmenten kehityksen tuomat paineet. Se on edelleen ainoa tapa saada elektroniset soittimet keskustelemaan keskenään ja sen käytön räjähdysmäinen kasvu tietokoneissa on osoittanut, että MIDI oli onnistunut standardi jo syntyessään.

MIDI:llä on kuitenkin omat puutteensa. MIDI:n suunnittelussa asetetut rajoitukset ovat herättäneet voimakasta kritiikkiä sen käyttäjien keskuudessa. Tällaisia rajoituksia ovat esimerkiksi 16:n kanavan ja 128:n instrumentin riittämättömyys ja sarjamuotoisen liikenteen tuoma hitaus tiedonsiirtoon. [Romanowski, 1990, sivu 112]

MIDI suunniteltiin ainoastaan elektronisten instrumenttien väliseen tiedonsiirtoon ja välittämään tietoa musiikillisista tapahtumista. Täten sen avulla ei voida välittää tietoja esimerkiksi kappaleen nimestä tai säveltäjästä. Sävelkorkeuden ilmaiseminen on MIDI:ssä toteutettu absoluuttisina arvoina, jolloin tieto esimerkiksi nuottiavaimesta on jätetty turhana informaationa pois. Perinteinen länsimainen viivastonotaatio sisältää paljon sellaista tietoa, jota ei voida MIDI:n keinoin mitenkään ilmaista. Esimerkiksi nuotinkirjoitusohjelmistoissa saattaa olla jopa 300 graafista elementtiä, kun taas MIDI pystyy kuvaamaan vain noin 40 [Bellini, 2001, sivu 84]. Näin esimerkiksi MIDI-tiedostojen käyttö tiedonsiirtoformaattina eri musiikkisovellusten välillä hukkaa paljon tärkeää informaatiota, jota esimerkiksi nuotinkirjoitusohjelmistot tarvitsisivat muodostaakseen oikeanlaisen visuaalisen esityksen kappaleesta. [Good, 2001a]

Tässä tutkielmassa toteutettavan sovelluksen näkökulmasta MIDI on erinomainen työväline, sillä sen avulla MusicXML-tiedoston sisältämä musiikillinen informaatio on muunnettavissa soivaan muotoon, musiikiksi, johon tarkoitukseen MIDI alunperin kehitettiin.

3.4 Toiminta ja syntaksi

MIDI-laitteet kommunikoivat keskenään lähettämällä toisilleen *MIDI-viestejä*. MIDI-tiedossa ei siirretä informaatiota ajasta, vaan sen voidaan ajatella olevan erilaisten musiikillisten tapahtumien virtaa. Vastaanottava MIDI-laite käsittelee saamansa MIDI-viestin välittömästi sen saatuaan ja jää odottamaan seuraavaa viestiä. Musiikkikappale on staattisena objektina, kuten nuottina tai äänitallenteena, olemassa pysyvänä kokonaisuutena. MIDI:n tehtävä on tarjota kaksisuuntainen liikenne tällaisten staattisten objektien ja reaaliaikaisten musiikillisten tapahtumien välille. [Hewlett, 1997b, sivu 42]

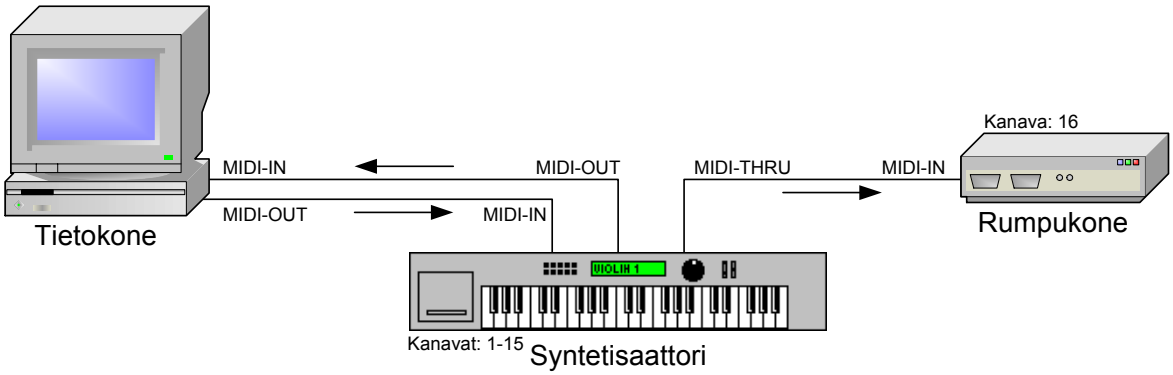
MIDI-laitteilla on kaksi roolia, lähettäjä ja vastaanottaja [Hewlett, 1997b, sivut 42-43]. MIDI-laite voi hoitaa myös näitä molempia rooleja ja esimerkiksi MIDI-koskettimistot toimivat usein kumpanakin. Kun soittaja painaa jotakin koskettimiston kosketinta, kosketTimisto muuntaa painetun koskettimen sävelkorkeuden ja painovoimakkuuden MIDI-viestiksi ja lähettää sen jollekin toiselle MIDI-laitteelle. Vastaavasti se voi toimia vastaanottajana toimien toiselta MIDI-laitteelta vastaanottamiensa MIDI-viestien mukaisesti.

MIDI:n tiedonsiirto on asynkronista, joka tarkoittaa sitä, että siirrettävässä liikenteessä ei ole tahdistuskoodia mukana. MIDI-liikenne on sarjamuotoista ja tieto kulkee kaapelissa vain yhteen suuntaan. Tarvitaan siis kaksi kaapelia, jotta kaksi yhteenliitettävää MIDI-laitetta voisivat keskustella kumpaankin suuntaan. [Romanowski, 1990, sivut 23 ja 33]

MIDI-laitteet kytketään toisiinsa *MIDI-kaapeleilla* niistä löytyvien *MIDI-liittimien* avulla, joista käytetään myös nimitystä *MIDI-portti*. MIDI-portteja on kolmea tyyppiä: *MIDI-IN* eli *sisääntulo*, *MIDI-OUT* eli *ulostulo* ja *MIDI-THRU* eli *läpivienti*. MIDI-laite kopioi MIDI-IN-porttiin saapuvan MIDI-liikenteen MIDI-THRU-porttiin ja näin läpivientiportin avulla voidaan ketjuttaa useampia laitteita saman järjestelmän piiriin. [Doan, 1994, sivut 10-11; Tuominen, 1989, sivu 18]

Tietokoneiden tarjoamat MIDI-palvelut noudattavat samaa logiikkaa kuin ulkoisetkin MIDI-laitteet. Esimerkiksi käyttöjärjestelmä näyttää äänikortilla olevan syntetisaattorin sovellusohjelmille standardinmukaisena MIDI-OUT-porttina ja sovellusohjelmistot voivat lähettää MIDI-viestejä tähän MIDI-porttiin samoin kuin lähettäisivät niitä ulkoiselle MIDI-laitteelle. [Sun Microsystems Inc., 2001]

Kuvassa 4 on havainnollistettu MIDI-kytkentöjä esimerkillä MIDI-ympäristöstä, joka koostuu tietokoneesta, syntetisaattorista ja rumpukoneesta. Kuvan MIDI-laitteiden ketjutus toimii siten, että koskettimiston MIDI-THRU-porttiin kopioituu kaikki tietokoneen lähettämä MIDI-tieto ja näin myös rumpukone voi vastaanottaa saman MIDI-liikenteen. Kuvassa näkyviin kanavanumeroihin paneudutaan tarkemmin seuraavassa luvussa.



Kuva 4. Tietokone, syntetisaattori ja rumpukone MIDI-ympäristössä.

3.4.1 Kanavat ja instrumentit

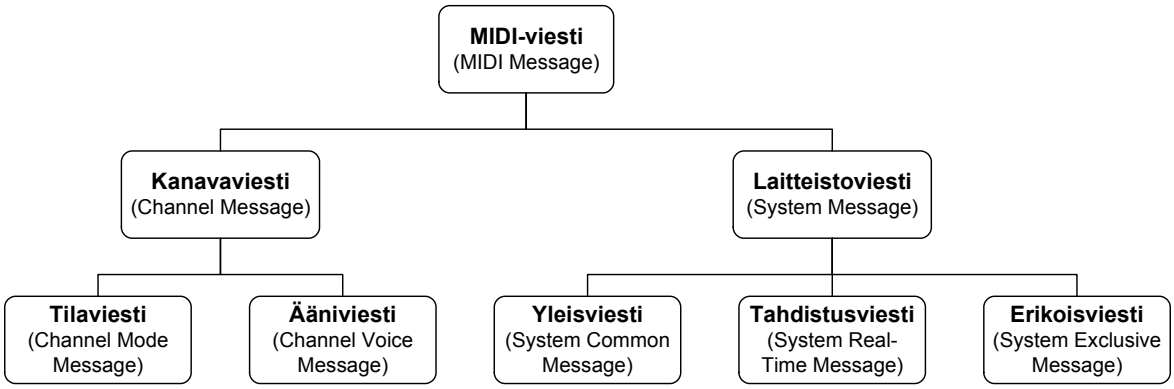
MIDI-kanavat ovat MIDI:n kulmakivi. Niitä on 16 ja niiden voidaan ajatella olevan MIDI-viestien osoitteita. Juuri MIDI-kanavien avulla esimerkiksi kuvassa 4 esitetty MIDI-ympäristö voidaan toteuttaa järkevästi. Jokainen MIDI-laite voidaan asettaa lähettämään ja vastaanottamaan MIDI-viestejä tietyllä kanavalla, jolloin kukin laite kykenee erottamaan MIDI-liikenteestä juuri sille laitteelle tarkoitetut viestit. MIDI-viestintä onnistuu siis vain, kun sekä lähettävä että vastaanottava MIDI-laite käyttää samaa MIDI-kanavaa. Nykyaikaisia MIDI-laitteita voidaan asettaa toimimaan myös useammalla kuin yhdellä kanavalla samanaikaisesti; tällaisia laitteita kutsutaan *multitimbraalisiksi* soittimiksi [Tuominen, 1989, sivut 25-26]. Esimerkiksi kuvan 4 järjestelmän syntetisaattori on asetettu toimimaan

kanavilla 1-15 ja rumpukone ainoastaan kanavalla 16. Näin kuvan tietokone osaa lähettäessään MIDI-viestejä osoittaa ne kanavanumeroiden avulla juuri oikeille laitteille.

Luvussa 3.2 mainitut syntetisaattorien *instrumenttinumerot* ovat MIDI:n toinen kulmakivi. Kullekin syntetisaattorin kanavalle voidaan asettaa käytettävä instrumentti ja näin syntetisaattori toistaa kaikki sille kanavalle lähetetyt MIDI-viestit käyttämällä tätä määriteltyä instrumenttia [Romanowski, 1990, sivut 53-54]. Jos esimerkiksi kuvan 4 syntetisaattorin kanavalle 1 asetetaan instrumentiksi sello ja kanavalle 2 pasuuna, niin tietokoneen lähettämät viestit kanavalle 1 toistetaan sellon äänellä ja kanavalle 2 lähetetyt viestit pasuunan äänellä.

3.4.2 MIDI-viestit

MIDI-viestin voidaan ajatella olevan käsky toiselle laitteelle suorittaa jokin toiminto. Tällainen toiminto voi olla vaikkapa nuotin soiton aloittaminen tai instrumentinvaihto [Tuominen, 1989, sivu 21]. MIDI-viestit on luokiteltu MIDI-standardissa kahteen pääluokkaan: *kanavaviesteihin (Channel Message)* ja *laitteistoviesteihin (System Message)*, jotka edelleen jakautuvat muihin alaluokkiin [Romanowski, 1990, sivu 37]. Kuvassa 5 on havainnollistettu MIDI-standardin mukaista MIDI-viestien luokittelua. MIDI-viestejä esitellään seuraavissa kappaleissa tässä tutkielmassa toteutetun sovelluksen kannalta oleellisin osin.



Kuva 5. MIDI-viestien hierarkia.

Kanavaviestit: Kanavaviestit ovat nimensä mukaisesti kanavakohtaisia viestejä. Niillä välitetään tietoa tietylle kanavalle asetetulle MIDI-laitteelle [Romanowski, 1990, sivu 37]. Kanavaviestit jakautuvat edelleen *tilaviesteihin* (*Channel Mode Message)* ja *ääniviesteihin* (*Channel Voice Message*) [Tuominen, 1989, sivu 49].

Tilaviesteihin kuuluvat viestit, joilla määritellään miten vastaanottavan MIDI-laitteen tulee käsitellä vastaanottamansa ääniviestit. Tilaviesteihin kuuluu lisäksi ”apuviestejä”, kuten esimerkiksi *kaikki äänet pois* -viesti (*All Notes Off Message*), jolla käsketään kohdelaitetta sammuttamaan kaikki sillä hetkellä päällä olevat äänet. Tätä viestiä kutsutaan myös nimellä *paniikkiviesti*, sillä sitä tarvitaan usein virhetilanteissa. [Romanowski, 1990, sivu 59; Tuominen, 1989, sivu 55]

Ääniviestit ovat MIDI-standardin keskeisimmät ja käytetyimmät viestit. Niiden avulla välitetään kaikki välittömimmin korvin kuultavaan äänen muodostamiseen tarvittavat asiat [Tuominen, 1989, sivu 49]. Tärkeimpiin ääniviesteihin lukeutuvat seuraavat MIDI-viestit:

1. *Ääni päälle -viesti* (*Note On Message*)
2. *Ääni pois -viesti* (*Note Off Message*)
3. *Instrumentinvaihtoviesti* (*Program Change Message*)

Ääni päälle -viestillä käsketään vastaanottavaa laitetta aloittamaan sävelen soitto ja välitetään tieto soitettavan sävelen korkeudesta ja voimakkuudesta. Tämän viestin saatuaan vastaanottava laite siis soittaa viestin määrittelemän sävelen määritellyllä sävelkorkeudella ja voimakkuudella [Hirvi, 1995, sivu 76]. Sävelen korkeus ja voimakkuus määritellään MIDI:ssä numeerisina arvoina väliltä 0-127. Sävelkorkeus määritellään siten, että yksiviivaisen oktaavin c-sävel saa arvon 60, cis eli korotettu c arvon 61 ja niin edelleen. Sävelten voimakkuudet määritellään vastaavalla tavalla siten, että suurin eli voimakkain arvo on 127 ja käytännössä äänetön eli hiljaisin arvo on 0 [Hirvi, 1995, sivu 23].

Ääni pois -viestillä on ääni päälle -viestiin verrattuna käänteinen vaikutus. Nimensä mukaisesti viestin saatuaan kohdelaite lopettaa viestin määrittelemän sävelen soiton. Jokaista ääni päälle -viestiä kohden pitää lähettää myös ääni pois -viesti. Vaihtoehtoinen ja itse asiassa yleisempi tapa on lähettää ääni pois -viestin asemesta ääni päälle -viesti, jonka

määrittelemä sävelkorkeus on sama kuin sitä vastaavan ääni päälle -viestin, mutta voimakkuus on nolla. [Romanowski, 1990, sivu 44]

Instrumentinvaihtoviestillä välitetään käsky kohdelaitteelle vaihtaa kanavan käyttämää sointiääntä eli instrumenttia. Tämän viestin saatuaan vastaanottava laite vaihtaa käyttämänsä sointiäänen esimerkiksi pianosta tuubaan [Hirvi, 1995, sivu 79]. Instrumentinvaihtoviestillä vaihdetaan vain kohdelaitteen käyttämää instrumenttia, eikä sillä voida mitenkään vaikuttaa esimerkiksi valitun instrumentin sointiväriin tai käyttötapaan. Jos sointiväriin halutaan vaikuttaa vaikkapa erilaisilla mahdollisilla efekteillä, ne on lähetettävä valmistajakohtaisilla *erikoisviesteillä* [Romanowski, 1990, sivu 54].

Ääniviesteihin lukeutuvat lisäksi *kanavakohtainen jälkipaino -viesti* (*Channel Pressure Message*), *kosketinkohtainen jälkipaino -viesti* (*Polyphonic Aftertouch Message*), *ohjainmuutosviesti* (*Control Change Message*) ja *taivutusviesti* (*Pitch Wheel Change Message*). [Romanowski, 1990, sivu 39]

Laitteistoviestit: Laitteistoviestit ovat nimensä mukaisesti tarkoitettu kaikille järjestelmään kytketyille vastaanottaville MIDI-laitteille. Jokaisen järjestelmään kytketyn laitteen on otettava vastaan nämä viestit riippumatta siitä, mitä kanavaa tai kanavia ne on asetettu käyttämään [Romanowski, 1990, sivu 39]. Laitteistoviestit luokitellaan edelleen *yleisviesteihin* (*System Common Message*), *tahdistusviesteihin* (*System Real-Time Message*) ja *erikoisviesteihin* (*System Exclusive Message*) [Tuominen, 1989, sivu 50].

Yleisviestejä käytetään esimerkiksi koko järjestelmää koskevan virityspyynnön välittämiseen tai ennalta tallennetun kappaleen aloitus- ja lopetuspyyntöjen välittämiseen [Romanowski, 1990, sivut 69-70]. Tahdistusviesteihin kuuluvat esimerkiksi *MIDI-kello* sekä muut järjestelmän tahdistukseen liittyvät viestit. Järjestelmän asetusten palautus alkuarvoihin tapahtuu *System Reset* -tahdistusviestillä [Romanowski, 1990, sivu 74]. Erikoisviestejä ovat *valmistajakohtaiset* viestit sekä harvinaiset *universaalit* viestit. Valmistajakohtaiset viestit on varattu laitevalmistajien omille erikoisviesteille mahdollistaen laajennusten tekemisen MIDI-standardiin [Hirvi, 1995, sivu 87].

4 MusicXML

Luvussa kuvataan musiikki-informaation kuvaamiseen tarkoitettua rakenteista MusicXML-määrittelykieltä. Luvussa 4.1 kerrotaan MusicXML:stä yleisesti, sen historiasta ja käyttötarkoituksista. XML:ää ja sen syntaksia esitellään luvussa 4.2. Luvussa 4.3 tarkastellaan MusicXML:n syntaksia esimerkkien avulla ja luvuissa 4.4-4.5 kuvataan XML:n ja MusicXML:n tyyppimäärityksiä. MusicXML:n syntaksia ja tyyppimäärityksiä käsitellään tässä tutkielmassa toteutettavan sovelluksen kannalta oleellisin osin.

4.1 Mikä on MusicXML?

MusicXML on musiikki-informaation visuaalisessa esityksessä käytetyn viivastonotaation kuvaamiseen tarkoitettu rakenteinen määrittelykieli. Sen tarkoitus on mahdollistaa sekä populaarimusiikin että taidemusiikin kuvaaminen 1600-luvulta lähtien. [Good, 2001b, sivu 114]

MusicXML pyrkii mahdollistamaan kaiken musiikissa olevan informaation kuvaamisen rakenteiseen muotoon MusicXML-dokumentteihin pääpainon ollessa kuitenkin viivastonotaation ominaisuuksien esittämisessä. Tulevaisuudessa MusicXML tulee todennäköisesti pyrkimään Internet-maailman musiikkialan standardiksi, jota tullaan käyttämään erilaisten musiikkisovellusten ”yhteisenä kielenä”. MusicXML on määritelty XML 1.0:n (*Extensible Markup Language 1.0*) mukaisesti [Good, 2001a].

4.1.1 Historia ja eri versiot

Musiikin kuvaamiseen on kehitetty monia erilaisia kuvauskieliä kautta tietotekniikan historian. Kirjassa *Beyond MIDI: A Handbook of Musical Codes* on esitetty yli 20 erilaista musiikin kuvauskieltä ja kymmeniä muita kuvauskieliä on olemassa. Yksikään näistä kuvauskielistä ei ole onnistunut nousemaan standardin asemaan (lukuun ottamatta MIDI:ä elektronisten soitinten välisessä tiedonsiirrossa), joten yhteiselle standardille on todellinen tarve [Castan, 2001a, sivu 96]. Myös tietoverkkojen kautta tapahtuva musiikin levitys ja myynti ovat kärsineet standardin puutteesta [Good, 2001a].

MusicXML pyrkii ratkaisemaan nämä ongelmat tarjoamalla musiikin tuottajille ja kuluttajille avoimen standardin painetun musiikin kuvaamiseen rakenteiseen muotoon. MusicXML-dokumentit ovat tekstitiedostoja ja siten sellaisinaan soveliaita Internetin välityksellä tapahtuvaan tiedonsiirtoon. XML-pohjaisuudestaan johtuen MusicXML tarjoaa myös sovelluskehittäjille hyvät lähtökohdat uusien sovellusten kehittämiseen. [Good, 2001a]

MusicXML on yhdysvaltalaisen *Recordare LLC* -nimisen yrityksen kehittämä määritys, joka tätä kirjoitettaessa (18.9.2003) kantaa versionumeroa 0.7b. Sen ensimmäinen tuotantoversio (0.6) julkistettiin 14.3.2002, jonka jälkeen se on ollut julkisessa beeta-testauksessa.

MusicXML-määritystä suunnitellessa Recordare LLC noudatti kahta strategista päälinjaa [Good, 2001a]:

1. Lähtökohtana MusicXML-määritykselle pidettiin kahta olemassa olevaa ja hyväksi osoittautunutta akateemista musiikin tallennusformaattia: *MuseData*:a ja *HumDrum*:a (katso luku 2.5).
2. MusicXML-määritys kehitettiin käyttäen inkrementaalista ohjelmistokehitysprosessimallia. Määritystä kehitettiin yhdessä sitä käyttävien sovellusten kehitystyön kanssa, jolloin saatiin samalla palautetta sen soveltuvuudesta käytäntöön.

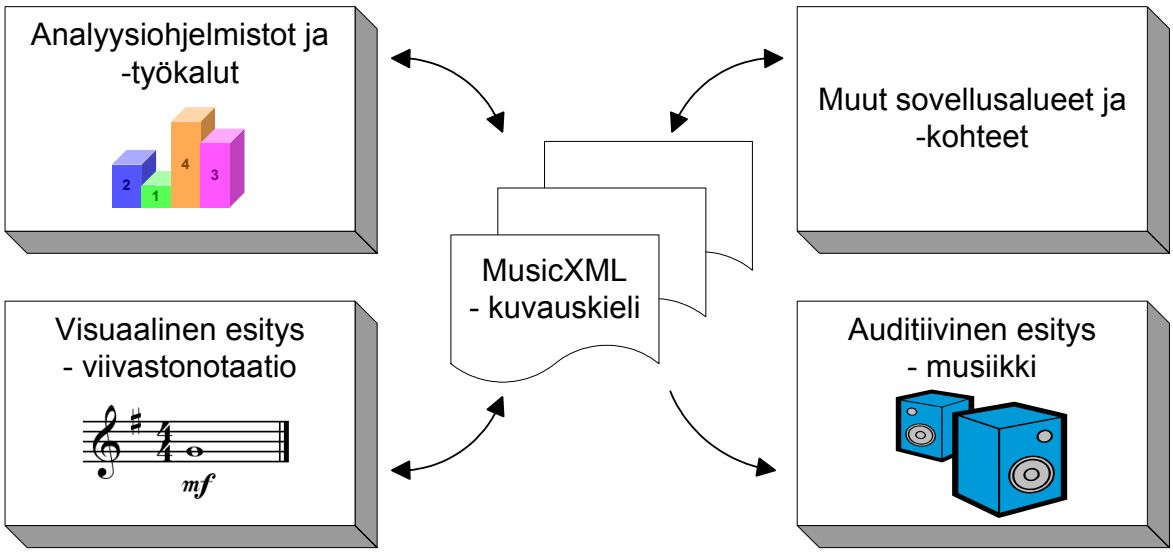
Inkrementaalisen prosessimallin mukaisessa ohjelmistokehityksessä sovellukselle (tai tässä tapauksessa määritykselle) laaditaan vaatimusmäärittely ja analyysi, ja tämän jälkeen kehitetään osa eli inkrementti kerrallaan valmiiksi [Sommerville, 1995, sivu 497]. Määrityksen kehitystyössä Recordare LLC toteutti prototyypin sovelluksesta, joka teki kahdensuuntaisen muunnoksen MuseData-formaattiin, luki NIFF-tiedostoja sekä kirjoitti standardeja MIDI-tiedostoja (*Standard MIDI File 1.0*) [Good, 2001a].

MusicXML-määritystä on tarkennettu ja kehitetty iteratiivisesti yhdessä sovelluskehittäjien kesken ympäri maailman. Yrityksen tulevaisuuden suunnitelmiin kuuluu standardoida MusicXML-määritys yleiseksi standardiksi todennäköisesti OASIS:n (*Organization for the Advancement of Structured Information Standards*) kautta. [Good, 2001a]

MusicXML-määrityksen lisenssi on saatavilla WWW-muodossa osoitteesta `http://www.musicxml.org/dtds/license.html` ja määrityksen käyttäminen on täysin maksutonta. MusicXML:n DTD-dokumentit (*Document Type Definition*) eli *tyyppimääritysdokumentit* ovat saatavilla WWW-muodossa osoitteesta `http://www.musicxml.org/dtds/index.html`. XML-dokumenttien ja MusicXML:n tyyppimäärityksiä käsitellään tarkemmin luvuissa 4.4 ja 4.5.

4.1.2 Käyttötilanteet, -tarkoitukset ja hyödyt

Luvussa kuvataan MusicXML-määrittelykielen käyttötilanteita, -tarkoituksia ja sen käytöstä saatavia hyötyjä. Kuvassa 6 on havainnollistettu MusicXML-määrittelykielen käyttöä erilaisten käyttötilanteiden ja -tarkoitusten yhteisenä tekijänä.



Kuva 6. MusicXML erilaisten käyttötilanteiden ja -tarkoitusten yhteisenä tekijänä.

Yhteinen kieli eri musiikkiohjelmistojen välille: Tähän päivään mennessä jokainen eri musiikkiohjelmisto on käyttänyt omaa tallennusformaattiansa, joten tiedonsiirto eri ohjelmistojen välillä on ollut erittäin hankalaa tai lähes mahdotonta. 1980-luvulla kehitetty MIDI (katso luku 3) onnistui ratkaisemaan vastaavan ongelman mahdollistaen syntetisaattorien ja muiden elektronisten soitinten välisen tiedonsiirron. MusicXML yrittää tehdä sähköisille nuottijulkaisuille ja musiikkiohjelmistoille saman, minkä MIDI teki elektronisille instrumenteille. [Good, 2001a]

Nykyisellään jokaisella eri musiikkiohjelmistolla on omat ilmaiset tai maksulliset tiedostojen katselu- ja soitto-ohjelmat, ja vain niiden avulla esimerkiksi Internetistä ladattu musiikkitiedosto on voitu soittaa tai tulostaa [Anthony, 2001, sivut 150-152]. Ohjelmistojen tallennusformaatit ovat lisäksi olleet pääsääntöisesti binäärisiä ja useimmat ohjelmistot ovat pitäneet formaattinsa salaisina jättäen sovelluskehittäjät vaille mahdollisuuksia kehittää uusia ohjelmistoja tiedostojen käsittelyyn. Tämä on aiheuttanut sen, että kutakin tiedostoa on voitu käsitellä vain sillä ohjelmalla jolla se on luotu. Musiikin kuluttajille tällainen menettely ei ole onnistunut tarjoamaan mitään lisäarvoa verrattuna paperiversion hankkimiseen samasta nuotista. [Good, 2001a]

MusicXML on saavuttanut hyviä tuloksia edellä mainituissa asioissa jo beetatestausvaiheessa. Esimerkiksi eräs nuottien skannausohjelmisto, *Visiv SharpEye Music Reader* (`http://www.visiv.co.uk/`), osaa tallentaa tiedostonsa MusicXML-muodossa, jotka edelleen voidaan avata jollain toisella ohjelmistolla. Tämäntyyppinen ohjelmistojen yhteistoiminta on ennen MusicXML-määritystä ollut mahdotonta. [Good, 2001a]

Vapaa ja avoin standardi sovelluskehittäjille: Ennen MusicXML:n julkaisemista eri tallennusformaatit, mukaan lukien akateemiset formaatit kuten MuseData ja HumDrum, sitoivat ohjelmistokehittäjät käyttämään jotakin tiettyä ohjelmointikieltä ja -ympäristöä. Esimerkiksi *Coda Music Technologies* -yrityksen markkinajohtajan asemassa olevan nuotinkirjoitusohjelmiston *Finale*:n *plug-in*:it, eli kolmansien osapuolten tekemät lisätoiminnot, oli pakko ohjelmoida C- tai C++-kielellä. HumDrum-formaatti toimi vain Unix-ympäristössä ja MuseData-työkalut toimivat harvinaisessa TenX-käyttöjärjestelmässä. Täten jos sovelluskehittäjä halusi tehdä sovelluksensa jollekin tietylle ohjelmistolle, hänen oli sitouduttava käyttämään tiettyjä työvälineitä. [Good, 2001a]

Eräs XML-tekniikoiden eduista on se, että niille on olemassa todella suuri määrä erilaisia työkaluja eri ympäristöihin [Harold, 1999, sivut 30-33]. Työkalujen saatavuus vapauttaa ohjelmistosuunnittelijat valitsemaan itselleen omiin käyttötarkoituksiinsa sopivimmat ohjelmointityökalut ja -ympäristöt [Good, 2001a; McLaughlin, 2000, sivut 12 ja 22-23]. Musiikkiohjelmistojen kehityksessä päästään siis vihdoin siirtymään toteutuskeskeisyydestä tarkoituskeskeisyyteen.

Lisäksi MusicXML-määrittelykieli on määritelty XML 1.0 -standardin mukaisesti. Jos XML-tekniikat ovat sovelluskehittäjälle ennestään tuttuja, MusicXML:ää käyttävien ohjelmistojen kehittämisen ei pitäisi tuottaa vaikeuksia.

Käyttö tietoverkoissa: Eri ohjelmistojen kirjo on vaikeuttanut myös painetun musiikin julkaisemista ja myymistä sähköisessä muodossa Internetin välityksellä. Tähän asti painetun musiikin myynti ja käyttö tietoverkoissa on estynyt eri ohjelmistojen omien, patentoitujen tallennusformaattien käytön vuoksi [Good, 2001b, sivu 120]. Lisäksi musiikkikappaleiden nuotteja myydään esimerkiksi kuvatiedostoina, joille on pelkän paperille tulostamisen lisäksi vaikea keksiä mitään muuta käyttöä [Bellini, 2001, sivu 84]. MusicXML-standardin käyttöönotto musiikin tallennusmuotona tuo kuluttajille, sisällöntuottajille ja ohjelmistokehittäjille laajan kirjon valmiita työvälineitä käytettäväksi.

XML on myös laitealustariippumaton, joten sama XML-dokumentti näyttää samalta ja sitä voidaan käsitellä samalla tavalla missä tahansa järjestelmässä [Good, 2001a]. Tässäkin tutkielmassa toteutettavan MusicXML-soittimen yksi tavoitteista on laitealustariippumattomuus ja MusicXML:n käyttö tietoverkosta ladattavana dokumenttina.

Tekijänoikeudet ja tietoturvan puute ovat olleet MP3:n (*MPEG Audio Layer 3*) ja MIDI:n heikkouksia [Anthony, 2001, sivu 143]. XML:n metadatan ja digitaalisten allekirjoitusten käyttö MusicXML:ssä voivat olla ratkaisu näihin ongelmiin musiikin julkaisemisessa ja myynnissä tietoverkoissa. [Good, 2001a]

Kuvitteellinen käyttöskenaario voisi olla seuraavanlainen: käyttäjä selailee Internetissä toimivan musiikkikaupan nuottikatalogia ja napsauttaa linkkiä, joka käynnistää MusicXML-soittimen, kuuntelee hieman kappaleen alkua ja päättää ostaa kyseisen nuotin. Maksamisen jälkeen käyttäjä saa ladata omalle koneelleen kyseisen MusicXML-dokumentin ja voi sitten edelleen tulostaa, kuunnella tai käsitellä sitä haluamallaan sovelluksella. Kaikki tämä kuulostaa tutulta, sillä olemmehan tottuneet tällaisiin palveluihin esimerkiksi tavallisten tekstidokumenttien käsittelyssä.

Tiedon hakeminen MusicXML-dokumenteista ja niiden analysointi: Musiikki informaationa on monimutkaista ja siten myös sen analysointi on ei-triviaalia [Byrd, 2001, sivu

239]. Myös tiedonhaku musiikkidokumenteista ja musiikkidatan analysointi on ollut vaikeaa monien eri tallennusformaattien vuoksi. MusicXML tarjoaa standardoidun tietorakenteen ja XML-tekniikat valmiit työkalut näihin tarkoituksiin [Good, 2002].

Informaation etsiminen musiikkidatasta on kompleksinen operaatio. Kyselyt ovat usein sumeita (*sumea logiikka*) ja tiedon sisäiset relaatiot erittäin monimutkaisia. Yksi tulevaisuuden käyttötarkoituksista voisi olla vaikkapa hyräilemällä annetun melodian etsiminen suuresta joukosta musiikkidokumentteja. Hakujen tekemiselle XML-dokumenteista on jo olemassa tekniikoita ja työkaluja, jotka soveltuvat sellaisinaan myös MusicXML:n käyttöön. Toisaalta musiikkidatan haku- ja analysointialgoritmit vaativat vielä paljon tutkimus- ja kehitystyötä, johon sovelluskehittäjät MusicXML:n myötä saavat keskittyä. [Good, 2001a]

4.2 XML ja sen syntaksi

Luvussa kuvataan yleisesti XML 1.0 -määritystä ja sen syntaksia.

4.2.1 XML yleisesti

XML on W3C:n (*World Wide Web Consortium*) virallinen suositus yleiskäyttöiseksi tekniikaksi tekstimuotoisten rakenteisten dokumenttien merkkaamiselle ja sen kantaisä on vuonna 1986 standardoitu SGML (*Standard Generalized Markup Language*) [Nykänen, 2001, sivu 10]. XML:n voidaan ajatella olevan yksinkertaistettu ja yleistetty SGML:n osajoukko [Castan, 2001a, sivu 96]. XML 1.0:n suunnittelu aloitettiin vuonna 1996 ja ensimmäinen *XML 1.0* -versio valmistui vuonna 1998. Määritystä korjattiin vuonna 2000 valmistuneessa *XML 1.0 (Second Edition)* -versiossa [Nykänen, 2001, sivu12].

SGML:n, ja siten myös XML:n, perusajatuksena on tiedon esittäminen rakenteisena, hierarkkisena, puumaisena rakenteena. Verrattuna vaikkapa Internetin WWW-sivujen kuvauskielenä käytettyyn HTML:ään (*HyperText Markup Language*), XML ei ota kantaa tiedon esittämiseen tai muotoiluun liittyviin asioihin, vaan XML-dokumentit sisältävät vain tietoa ennalta määritellyssä hierarkkisessa järjestyksessä. [Castan, 2001a, sivu 96]

Eräs XML:n vahvuuksista on se, että XML-dokumentit ovat tekstitiedostoja ja tieto tallennetaan niihin selkokielisenä [Harold, 1999, sivu 31]. Tällä tavoin käyttäjä saa käsityksen dokumentin sisällöstä jo ainoastaan lukemalla XML-dokumenttia. Tällaista tiedon esitystapaa kutsutaan *itsekuvailevaksi* tai *itsedokumentoivaksi* esitystavaksi [Holzner, 2001, sivu 28].

4.2.2 XML:n syntaksi

XML-dokumentit koostuvat *elementeistä* ja *attribuuteista*, jotka esiintyvät dokumenteissa hierarkkisena rakenteena. Elementit sisältävät tietoa ja ne voivat sisältää myös toisia elementtejä. Elementeillä voi olla myös attribuutteja, joilla voidaan merkata elementeille lisämääreitä. Toisin kuin HTML:ssä tai SGML:ssä XML-dokumentin on oltava *hyvin muodostettu* eli elementtien on esiinnyttävä dokumentissa hierarkkisen järjestyksen mukaisesti aidosti sisäkkäin [Castan, 2001a, sivu 96]. Hyvin muodostetulla XML-dokumentilla on myös oltava *juurielementti*, joka sisältää kaikki muut dokumentin elementit [Holzner, 2001, sivu 80].

Elementille ja attribuutille voidaan määritellä myös *nimiavaruus* (*namespace*), jolla varmistetaan elementin tai attribuutin nimeämisen yksikäsitteisyys. Nimiavaruuksien avulla voidaan esimerkiksi yhdistää kaksi tai useampia XML-sanastoja ilman vaaraa elementtien tai attribuuttien nimien päällekkäisyydestä [Roy, 2001, sivu 39; Nykänen, 2001, sivut 182-183]. MusicXML-määrittelykieli ei käytä nimiavaruuksia, joten niitä ei käsitellä tässä yhteydessä tämän enempää.

Jokainen XML-dokumentti koostuu kahdesta osasta: *esittelyosasta* (*prolog*) ja *esiintymäosasta* (*instance*). Esittelyosa voi sisältää:

- XML-julistuksen (*XML Declaration*)
- Dokumentin tyyppijulistuksen (*Document Type Declaration*)
- Kommentteja ja/tai prosessointiohjeita.

Esiintymäosa koostuu vähintään yhdestä elementistä, dokumentin juurielementistä, jonka sisään koko loppudokumentti sijoittuu. [Nykänen, 2001, sivu 105]

Esiintymäosan elementit merkataan XML-dokumenttiin alku- (*start tag*) ja lopputunnisteiden (*end tag*) avulla. Elementillä pitää aina olla sekä alku- että lopputunniste, esimerkiksi `<henkilö>...</henkilö>`. Elementtiä kutsutaan *lapsielementiksi*, jos se sisältyy johonkin toiseen elementtiin, ja *vanhempielementiksi*, jos sillä on lapsielementtejä [Nykänen, 2001, sivu 19]. Elementti voi olla myös tyhjä, jolloin se voidaan merkata dokumenttiin käyttäen lyhennysmerkintää `<henkilö/>` [Nykänen, 2001, sivu 112]. *Kommentit* merkataan XML-dokumenttiin `<!--` ja `-->` -tunnisteiden väliin ja ne jätetään yleensä dokumenttia käsittelevässä sovelluksessa huomiotta [Nykänen, 2001, sivu 108].

Elementteihin voidaan myös merkata lisämääreitä eli attribuutteja. Elementin attribuutti koostuu nimi-arvo -parista, esimerkiksi henkilön sukupuoli voidaan merkata attribuuttina `<henkilö sukupuoli="mies">...</henkilö>`. Attribuutit siis merkataan elementin alkutunnisteen sisään. [Holzner, 2001, sivu 81]

Esimerkissä 1 on esitetty kuvitteellinen tapaus XML-dokumenttiin merkatusta yhdestä tahdista, joka sisältää yhden nuotin. Tahti on luvussa 2.2 esitetyn kuvan 1 sellon ensimmäinen tahti. Tahti sisältää yhden nuotin, joka on pienen oktaavin sävelkorkeuden D kokonuotti. Esimerkin `<tahti>`-elementti on tässä tapauksessa dokumentin juurielementti. `<nuotti>`-elementillä on `tyyppi`-niminen attribuutti, joka määrittelee, että kyseessä on kokonuotti. `<nuotti>`-elementti sisältää myös kaksi lapsielementtiä, `<sävelkorkeus>` ja `<oktaavi>`, joilla määritellään sävelen suhteellinen sävelkorkeus ja oktaaviala.

Esimerkin XML-dokumentti on loogiselta rakenteeltaan hyvin muodostettu, sillä sen elementit ovat hierarkkisen järjestyksen mukaisesti aidosti sisäkkäin ja sillä on tasan yksi juurielementti. Esimerkin XML-merkkaus on täysin esimerkinomainen eikä sillä ole mitään tekemistä varsinaisen MusicXML:n kanssa.

Esimerkki 1. Elementtien ja attribuuttien merkkaus XML-dokumentissa.

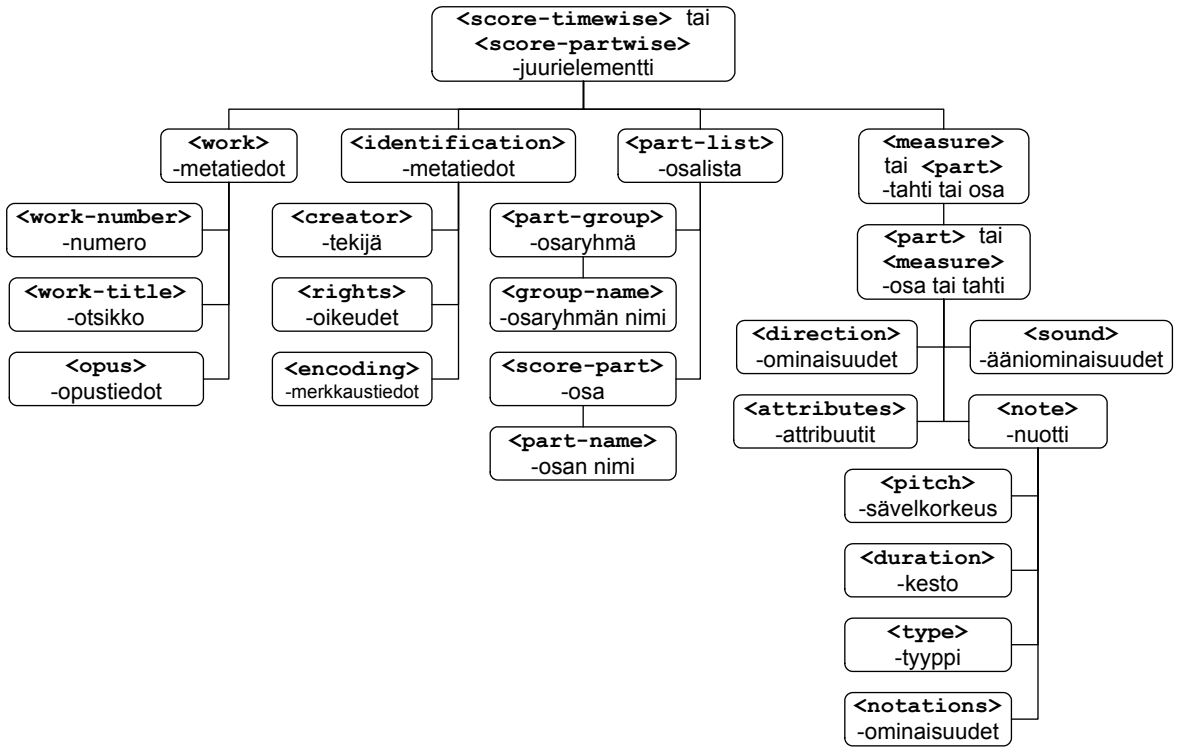
```
<?xml version="1.0"?>
<tahti>
  <nuotti tyyppi="kokonuotti">
    <sävelkorkeus>D</sävelkorkeus>
    <oktaavi>pieni</oktaavi>
  </nuotti>
</tahti>
```

4.3 MusicXML:n syntaksi

Luvussa kuvataan MusicXML-määrittelykielen syntaksia. Luvun esimerkit liittyvät luvussa 2.2 käsitellyn esimerkkikappaleen (katso kuva 1) MusicXML-esitykseen. Liitteessä 1 on esitetty tämä MusicXML-dokumentti kokonaisuudessaan. Liitteen 2 taulukossa L.1 on kuvattu taulukkona yleisimmät viivastonotaation elementit ja niitä vastaavat MusicXML-määrittelykielen tunnisteet. Luku perustuu lähteeseen [Recordare LLC, 2003] ellei toisin ilmoiteta.

4.3.1 MusicXML:n elementtien hierarkia

Kuvassa 7 on havainnollistettu tärkeimpien MusicXML-määrittelykielen elementtien hierarkiaa ja suhdetta toisiinsa. Luvussa 2.1 käsitellyt musiikin eri olomuodot on yritetty sisällyttää MusicXML-määritykseen siten, että kuvattavan musiikkikappaleen musiikillinen sisältö merkataan elementteinä ja graafiseen viivastonotaatioon ja auditiiviseen esitykseen liittyvät asiat elementtien attribuutteina. Tätä periaatetta ei ole pystytty kuitenkaan käytännössä noudattamaan [Good, 2002].



Kuva 7. MusicXML-määrittelykielen elementtien hierarkia.

4.3.2 MusicXML-dokumentin esittelyosa

Esimerkissä 2 on esitetty MusicXML-dokumentin esittelyosa. Jokainen XML-dokumentti alkaa XML-julistuksella, jossa määritellään XML:n versio ja dokumentissa käytetty merkistö. Esimerkin dokumentti on XML 1.0 -määrityksen mukainen. `encoding="UTF-8"` tarkoittaa sitä, että dokumentin merkkauksessa on käytetty `UTF-8`-merkistöä ja `standalone="no"` sitä, että dokumentin tyyppi on määritelty erillisessä tyyppimääritysdokumentissa. Toiselta riviltä alkaa dokumentin tyyppijulistus, joka tarkoittaa, että dokumentti on MusicXML-dokumentti ja se on määritelty tyyppimääritysdokumentissa `partwise.dtd`. XML:n ja MusicXML:n tyyppimäärityksiä käsitellään tarkemmin luvuissa 4.4 ja 4.5.

Esimerkki 2. MusicXML-dokumentin esittelyosa.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE score-partwise PUBLIC
  "-//Recordare//DTD MusicXML 0.7b Partwise//EN"
  "http://www.musicxml.org/dtds/partwise.dtd">
```

4.3.3 MusicXML-dokumentin juurielementti ja osalista

Esimerkissä 3 on MusicXML-dokumentin juurielementti ja osalista. Esimerkin juurielementti `<score-partwise>` koostuu osista ja osat koostuvat tahdeista. Osalla (`<part>`) tarkoitetaan esimerkiksi yhtä instrumenttia tai nuottiviivastoa. Jokaisen MusicXML-dokumentin juurielementti on joko `<score-partwise>`- tai `<score-timewise>`-elementti riippuen jaottelun valinnasta musiikin kahden ulottuvuuden välillä. Juurielementteihin palataan myöhemmin tyyppimääritysten yhteydessä luvussa 4.5.

MusicXML-dokumentin osat määritellään otsikkoelementissä nimeltä `<part-list>`. Tämän elementin sisällä määritellään kaikki dokumentista löytyvät osat tai instrumentit (`<score-part>`), joille annetaan yksikäsitteinen `id`-tunnus. `<score-part>`-elementin ainoa pakollinen lapsielementti on osan nimen määrittelyyn tarkoitettu `<part-name>`-elementti, mutta lisäksi voidaan määritellä esimerkiksi osan käyttämän instrumentin MIDI-kanavan ja -instrumentin numero. Esimerkin osalista koostuu neljästä osasta, joista ensimmäisen osan nimi on `Violin 1`, `id` on `P1`, MIDI-kanavan numero on `1` ja MIDI-instrumentin numero on `41`. Seuraavaksi ovat dokumentin muut osat ja osalistan jälkeen alkaa dokumentin varsinainen datasisältö.

Esimerkki 3. MusicXML-dokumentin juurielementti ja osalista.

```
<score-partwise>
  <part-list>
    <score-part id="P1">
      <part-name>Violin 1</part-name>
      <midi-instrument id="P1-I1">
        <midi-channel>1</midi-channel>
        <midi-program>41</midi-program>
      </midi-instrument>
```

```
    </score-part>
    <score-part id="P2">
      <part-name>Violin 2</part-name>
    </score-part>
    <!-- Muut osat/instrumentit -->
  </part-list>
  <part id="P1">
    <!-- Dokumentin datasisältö -->
  </part>
</score-partwise>
```

4.3.4 MusicXML-dokumentin datasisältö

Esimerkissä 4 on esitetty esimerkkikappaleen MusicXML-esityksen varsinaisen datasisällön alku. Dokumentin ensimmäinen osa alkaa tunnisteella `<part id="P1">`, jonka `id`-attribuutin on viitattava osalistassa määriteltyyn osaan. Ensimmäinen tahti alkaa `<measure number="1">`-tunnisteella, jonka `number`-attribuutilla määritellään kyseisen tahdin järjestysnumero. Tämän jälkeen seuraavat tahdin attribuutit. `<attributes>`-elementin sisällä määritellään sävellaji (`<key>`), tahtilaji (`<time>`), nuottiavain (`<clef>`) ja sävelten soivan keston määrittelemiseen tarkoitettu `<divisions>`-elementti.

`<divisions>`-elementti määrittelee kyseessä olevan osan käyttämän tahtilajin pulssin jaottelun. Sen arvo määrää kuinka moneen osaan yksi neljäsosanuotti jakautuu ajallisesti. Esimerkin kappaleessa ensimmäisen viulun pienin nuotin tai tauon aika-arvo on kahdeksasosa, joten jakajaksi asetetaan `2`, sillä neljäsosanuotti vastaa ajalliselta kestoltaan kahta kahdeksasosanuottia. Jos osassa esiintyisi esimerkiksi kuudestoistaosanuotin pituisia aika-arvoja, niin `<divisions>`-elementin arvoksi määriteltäisiin `4`. `<divisions>`-elementin merkitykseen palataan myöhemmin nuottien aika-arvojen yhteydessä.

Sävellaji määritellään MusicXML:ssä elementin `<key>` avulla ja se määräytyy luvussa 2.2 kuvatun kvinttiympyrän avulla. `<fifths>`-elementissä määritellään kvinttisiirtymien lukumäärä kvinttiympyrässä C-duurista alkaen. Esimerkkikappaleen D-duuri saa kvinttiympyrän avulla määriteltynä arvon `2`, eli kuinka monta askelta C-duurista lähtien kuljetaan kvinttiympyrää pitkin myötäpäivään. B-duuri määriteltäisiin vastaavasti arvolla

-2. Elementillä <mode> määritellään lisäksi, onko sävellaji duuri (*major*) vai molli (*minor*).

MusicXML:ssä tahtilaji määritellään <time>-elementissä. Esimerkkikappaleessa tahtilaji on neljä neljäsosaa ja se määritellään elementtien <beats> ja <beat-type> avulla. Viulun nuottiavain (<clef>) on G-nuottiavain (<sign>) ja sen keskusta osoittaa nuottiviivaston toista viivaa (<line>).

Seuraavaksi vuorossa on ensimmäisen viulun ensimmäinen nuotti, joka määritellään <note>-elemetillä. Elementin sisältä löytyvät nuotin sävelkorkeus (<pitch>), kesto (<duration>), tyyppi (<type>) ja tauon ollessa kyseessä tyhjä <rest>-elementti. Kaksi ensimmäistä nuottia ovat taukoja, joten kahden ensimmäisen <note>-elementin sisällä on tyhjät <rest>-elementit.

Nuotin sävelkorkeus määritellään <pitch>-elementin ja sen lapsielementtien <step> ja <octave> avulla. Nuotin juurisävelen nimi määritellään <step>-elementillä ja nuotin tyyppi <type>-elementillä. Ensimmäisen viulun ensimmäinen soitettava sävel on sävelkorkeudeltaan yksiviivainen *a* ja tyypiltään neljäsosanuotti (*quarter*). MusicXML:ssä oktaavialat (<octave>) on numeroitu siten, että kontraoktaavi saa järjestysnumeron 1 ja loput oktaavialat on numeroitu siitä ylös- ja alaspäin. Näin yksiviivainen oktaavi saa järjestysnumeron 4. Nuotin ajallinen kesto määritellään <duration>-elementillä. Kuten edellä mainittiin, nuottien kestot määritellään MusicXML:ssä neljäsosanuotin jakajaksi asetetun arvon kertoimina. Neljäsosanuotin kesto jakajan ollessa 2 on 2, joten <duration>-elementin arvo on 2.

Nuottiin liittyvät erityisominaisuudet määritellään MusicXML:ssä <notations>-elementin avulla. Esimerkkikappaleessa tällainen on esitysmerkintä *mezzoforte*, joka määritellään elementin <dynamics> lapsielementtinä olevan tyhjän <mf/>-elementin avulla. <dynamics>-elementin attribuutti placement="below" tarkoittaa, että esitysmerkintä sijaitsee nuottiviivaston alapuolella ja relative-x="-9" kertoo sen suhteellisen sijainnin tahdin alusta lähtien. Esitysmerkinnät voidaan määritellä MusicXML-määrittelykielessä myös tahtikohtaisina, jolloin ne määritellään <directions>-elementin avulla

<measure>-elementin lapsielementteinä. Lisäksi absoluuttinen esitysvoimakkuus voidaan määritellä myös <sound>-elementillä. Esimerkissä nuotin voimakkuudeksi määritellään <sound>-elementin attribuutilla dynamics=83 MIDI-standardin voimakkuusarvo 83.

Esimerkki 4. MusicXML-dokumentin datasisältö.

```
<part id="P1">
  <measure number="1">
    <attributes>
      <divisions>2</divisions>
      <key>
        <fifths>2</fifths>
        <mode>major</mode>
      </key>
      <time>
        <beats>4</beats>
        <beat-type>4</beat-type>
      </time>
      <clef>
        <sign>G</sign>
        <line>2</line>
      </clef>
    </attributes>
    <note>
      <rest/>
      <duration>4</duration>
      <type>half</type>
    </note>
    <note>
      <rest/>
      <duration>2</duration>
      <type>quarter</type>
    </note>
    <sound dynamics="83"/>
    <note>
      <pitch>
        <step>A</step>
        <octave>4</octave>
      </pitch>
      <duration>2</duration>
      <type>quarter</type>
      <notations>
        <dynamics placement="below" relative-x="-9">
          <mf/>
```

```
        </dynamics>
      </notations>
    </note>
  </measure>
  <!-- Dokumentin loppuosa -->
</part>
```

4.4 XML:n tyyppimääritykset

Luvussa kuvataan XML:n tyyppimäärityksiä eli DTD:tä (*Document Type Definition*). Luvussa 4.4.1 kuvataan tyyppimäärityksiä yleisesti ja luvussa 4.4.2 niiden syntaksia esimerkkien avulla.

4.4.1 Yleistä XML:n tyyppimäärityksistä

DTD-määritys sisältyy W3C:n XML 1.0 -määritykseen ja se on tarkoitettu XML-dokumenttien loogisen rakenteen kuvaamiseen. DTD on siis käytännössä kielioppi kieliopin sisällä [Nykänen, 2001, sivu 72]. Teknisesti DTD määrittelee säännöt, joita XML-dokumentin elementtien ja attribuuttien nimien, sisällön ja loogisen rakenteen tulee noudattaa. XML-dokumentti on *validi*, jos sen tyyppi esitellään dokumentin esittelyosassa ja sen esiintymäosa noudattaa loogiselta rakenteeltaan tätä tyyppimääritystä. [Nykänen, 2001, sivut 122-124]

XML-dokumentin tyyppimääritys voidaan sijoittaa itse dokumentin sisään, mutta yleensä tyyppimääritykset sijaitsevat erillisessä tiedostossa. Tyyppimääritykset voidaan edelleen jakaa selkeyden vuoksi useampaan tiedostoon ja sijoittaa ne saataville vaikkapa Internetiin [Nykänen, 2001, sivu 124]. Esimerkiksi XHTML 1.0 -määrityksen (*Extensible HyperText Markup Language: A Reformulation of HTML 4.0 in XML 1.0*) mukaisen dokumentin esittelyosa on seuraavanlainen:

Esimerkki 5. XHTML 1.0 -dokumentin esittelyosa

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

Esimerkin dokumentin tyyppi on määritelty tyyppimäärittelydokumentissa `xhtml1-strict.dtd` ja sen sijainti on määritelty URL:n (*Uniform Resource Locator*) avulla.

DTD-määrityksen on vähitellen syrjäyttämässä *XML Schema Language*, joka tarjoaa paremmat ja monipuolisemmat keinot XML-dokumenttien elementtien ja attribuuttien syntaksin ja semantiikan määrittelemiseen. XML Schema Language -määrityksen avulla on esimerkiksi mahdollista määritellä elementin tai attribuutin sisältämän datan tyyppi ja sallitut arvot. DTD-määrityksen avulla tällainen rajoittaminen ei ole mitenkään mahdollista, vaan vastuu datan oikeellisuuden tarkistamisesta on kokonaan XML-dokumenttia käsittelevällä sovelluksella. [Roy, 2001, sivu 37-38]

4.4.2 XML:n tyyppimääritysten syntaksi

Esimerkissä 6 on esitetty luvussa 4.2.2 käsitellyn esimerkkidokumentin tyyppimäärittely eli DTD. XML-dokumentin tyyppimääritys koostuu sarjasta määrityksiä, jotka määrittelevät sallitut elementit, attribuutit ja entiteetit [Harold, 1999, sivu 216].

Dokumentin tyyppimääritys alkaa merkeillä `<!DOCTYPE` ja päättyy merkkeihin `]>`. Juurielementin nimen (`tahti`) on aina vastattava dokumentin tyyppimäärityksen nimeä (`tahti`). Tämän jälkeen seuraavat kaikkien sallittujen elementtien ja niiden attribuuttien määritykset. Avainsanalla `ELEMENT` aloitetaan elementin tyyppimääritys ja jonkin jo määritellyn elementin sallittujen attribuuttien tyyppimääritys avainsanalla `ATTLIST`. [Harold, 1999, sivut 216 ja 300]

Rivillä `<!ELEMENT tahti (nuotti*)>` määritellään `<tahti>`-elementti, joka saa sisältää mielivaltaisen lukumäärän (`nuotti*`) `<nuotti>`-elementtejä. `<sävelkorkeus>`-elementin määrityksessä taas määritellään sen sisältävän jäsennettyä merkkimuotoista dataa (`#PCDATA`, *parsed character data*). Merkkijonolla `<!ATTLIST nuotti tyyppi CDATA #REQUIRED>` määritellään `<nuotti>`-elementin pakolliseksi (`#REQUIRED`) attribuutiksi merkkimuotoista dataa (`CDATA`, *character data*) sisältävä `tyyppi`-attribuutti. [Harold, 1999, sivut 216 ja 302-303]

Esimerkki 6. XML-dokumentin tyyppimäärityksen syntaksi.

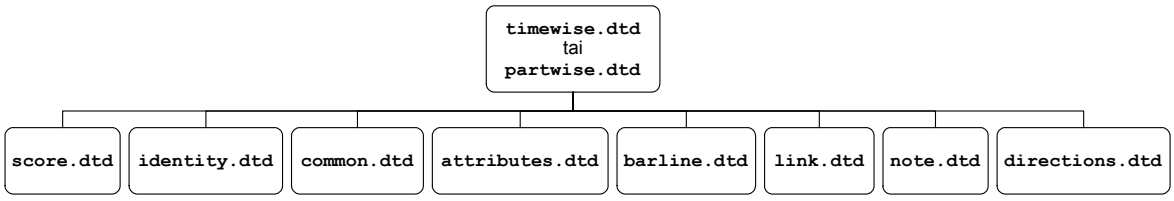
```
<!DOCTYPE tahti [
  <!ELEMENT tahti          (nuotti*)>
  <!ELEMENT nuotti         (sävelkorkeus, oktaavi)>
  <!ELEMENT sävelkorkeus   (#PCDATA)>
  <!ELEMENT oktaavi        (#PCDATA)>
  <!ATTLIST nuotti tyyppi CDATA #REQUIRED>
]>
```

4.5 MusicXML:n tyyppimääritykset

Luvussa 4.3.2 mainittiin esimerkin MusicXML-dokumentin noudattavan tiettyä tyyppimääritystä. MusicXML-määrityksen tyyppimääritykset pyrkivät kattamaan kaiken musiikissa esiintyvän informaation ja ovat siten erittäin laajoja. Ne on jaoteltu selkeyden vuoksi useaan erilliseen tiedostoon, joista hierarkkisesti ylimpinä ovat `timewise.dtd` ja `partwise.dtd`. Seuraavissa luvuissa esitellään MusicXML-määrittelykielen tyyppimäärityksiä tässä tutkielmassa toteutettavan sovelluksen näkökulmasta oleellisin osin. MusicXML:n tyyppimääritykset ovat tätä kirjoitettaessa (18.9.2003) julkisessa beetatestauksessa ja niitä kehitetään edelleen yhdessä niitä käyttävien sovellusten ja sovelluskehittäjien kesken [Good, 2001b, sivu 118].

4.5.1 Yleistä MusicXML:n tyyppimäärityksistä

Kuvassa 8 on havainnollistettu MusicXML-määrittelykielen tyyppimääritysdokumenttien hierarkiaa ja suhdetta toisiinsa.



Kuva 8. MusicXML:n tyyppimääritysdokumenttien hierarkia.

Luvussa 2.4.2 pohdittu musiikin kaksiulotteisuus soittimittain tai ajallisesti jaettuna hierarkiana on ollut yksi keskeisiä asioita MusicXML:n suunnittelussa [Recordare LLC, 2003].

Erilaiset ohjelmistot käsittelevät musiikillista informaatiota eri tavoin ja toinen kuvaustapa voi olla jollekin sovellukselle parempi kuin toinen. Esimerkiksi musiikkia reaaliajassa käsittelevä sovellus käsittelee dokumentteja mieluummin musiikillisten tapahtumien ajallisena virtana (<score-timewise>) verrattuna soittimittain jaettuun informaatioon (<score-partwise>).

MusicXML pystyy esittämään musiikin molemmissa edellä mainituissa muodoissa ja mahdollistaa helpon tavan vaihtaa esitystavasta toiseen. Spesifikaation mukana toimitetaan XSL-muunnosdokumentit (*Extensible Stylesheet Language Transformations 1.0*) muunnosten tekemiseksi näiden kahden esitystavan välillä [Good, 2001a]. XSLT on XML-tyylien muunnoksiin erikoistunut W3C:n suositus. Yhdessä XSL-muuntimen (*Extensible Stylesheet Language Transformations processor*) kanssa XSL-dokumentti (*Extensible Stylesheet Language*) muuntaa yhden tai useamman XML-dokumentin tasan yhdeksi XML-dokumentiksi. XSLT-muunnoksia käytetään paljon esimerkiksi WWW-sivustojen dynaamisessa luonnissa ja ylläpidossa [Nykänen, 2001, sivu 202].

Recordare LLC on MusicXML-määrittelykielen kehitystyön yhteydessä kehittänyt prototyypin sovelluksesta, joka muuntaa XSL-muunnosdokumentin avulla MusicXML-dokumentin MIDI-tiedostoksi (*MIDI Standard File 1.0*, katso luku 3.2). Tämä prototyyppi on ollut yhtenä inkrementaalisen ohjelmistokehitysprosessimallin mukaisena testausvälineenä MusicXML-määrittelykielen kehitystyössä [Good, 2001a]. Tätä tutkielmaa kirjoitettaessa (18.9.2003) tämä sovellus ei ole kuitenkaan ollut julkisesti saatavilla, joten kokemuksia sen toimivuudesta ei ole.

4.5.2 MusicXML-dokumenttien juurielementit

Musiikin kaksiulotteisuus on toteutettu MusicXML-määrityksessä dokumenttien juurielementistä lähtien (katso kuva 7). Jokaisen MusicXML-dokumentin juurielementti on joko <score-partwise> tai <score-timewise>.

Esimerkissä 7 on esitetty MusicXML:n `score.dtd`-dokumentin tärkein osa, jossa on määritelty MusicXML-dokumenttien juurielementit ja niiden sisältämät elementit. Esimer-

kin elementit on määritelty entiteettien (*Entity*) avulla, joiden tunniste on `%`-merkki. Entiteettien avulla tyyppimääritysdokumentit voidaan jäsentää helpommin hallittaviksi osakokonaisuuksiksi ja jakaa esimerkiksi useampaan tiedostoon. Nämä kaksi juurielementtiä vastaavat luvun 2.2 kuvassa 1 havainnollistettua musiikki-informaation kahta ulottuvuutta. Ehdollisessa lauseessa olevat entiteetit `%partwise` ja `%timewise` on esitetty kahdessa ylimmän tason DTD-dokumentissa, `partwise.dtd` ja `timewise.dtd`. Ainoa ero juurielementtien välillä on, että `<score-partwise>`-dokumentti sisältää yhden tai useamman osan, jotka sisältävät yhden tai useamman tahdin, ja `<score-timewise>`-dokumentti sisältää samat asiat käänteisessä järjestyksessä. [Recordare LLC, 2003]

Esimerkki 7. MusicXML-dokumentin juurielementin määrittely.

```
<![ %partwise; [
<!ELEMENT score-partwise (%score-header;, part+)>
<!ELEMENT part (measure+)>
<!ELEMENT measure (%music-data;)>
]]>
<![ %timewise; [
<!ELEMENT score-timewise (%score-header;, measure+)>
<!ELEMENT measure (part+)>
<!ELEMENT part (%music-data;)>
]]>

<!ENTITY % score-header
  "(work?, movement-number?, movement-title?,
    identification?, part-list)">

<!ENTITY % music-data
  "(note | backup | forward | direction |
    figured-bass | print | sound | harmony |
    grouping | attributes | barline)*">
```

Kumpikin MusicXML-dokumentin juurielementti sisältää entiteetit `%score-header` ja `%music-data`, jotka on myös määritelty `score.dtd`-dokumentissa. Näitä entiteettejä käsitellään tarkemmin seuraavissa luvuissa.

4.5.3 Otsikkotietojen määrittely `%score-header`-entiteetin avulla

Esimerkissä 7 on esitetty myös `score.dtd`-dokumentissa määritelty `%score-header`-entiteetti. Se sisältää viiden elementin määrittelyt, joista vain dokumentin osalista on pakollinen elementti.

MusicXML-dokumentin metatiedot: Otsikkotiedot sisältävät metatietoa dokumentista, kuten kappaleen nimen (`<movement-title>`) ja numeron (`<movement-number>`), jos kappale on osa isompaa kokonaisuutta. Elementit `<work>` ja `<identification>` on määritelty sisältämään myös muita elementtejä, jotka on määritelty tyyppimääritysdokumenteissa `identity.dtd` ja `opus.dtd`. [Recordare LLC, 2003]

Tulevaisuudessa metatietoihin tullaan varmasti lisäämään tekijänoikeustietoja, digitaalisia allekirjoituksia ja muita tärkeitä tietoja tietoturvan ja tekijänoikeuksien valvonnan mahdollistamiseksi. [Good, 2001a]

MusicXML-dokumentin osalista: Osalista on otsikkotietojen ainoa pakollinen ja siten tärkein elementti. Luvussa 4.3.3 käsitelty esimerkki MusicXML-dokumentista esitteli osalistan toimintaa. Esimerkissä 8 on esitetty osalista-elementin (`<part-list>`) tyyppimäärittely. Osalista sisältää tiedot dokumentin sisältämistä osista ja osaryhmistä. Yksi osa (`<score-part>`) sisältää osan nimen, ryhmän ja muita osan tietoja. `<score-part>`-elementillä on yksi pakollinen attribuutti, jota käytetään osien nimeämiseen ja erottelemiseen toisistaan. Tämä attribuutti on nimeltään `id` ja sillä on oltava yksikäsitteinen arvo. Jokaisen MusicXML-dokumentin datasisältöosassa olevan osan `id`-attribuutin on viitattava johonkin dokumentin osalistassa määriteltyyn osaan. [Recordare LLC, 2003]

Esimerkki 8. `<part-list>`-elementin tyyppimäärittely.

```
<!ELEMENT part-list (part-group*, score-part,
                     (part-group | score-part)*)>
<!ELEMENT score-part (identification?, part-name,
          part-abbreviation?, group*, score-instrument*,
          midi-device?, midi-instrument*)>
<!ATTLIST score-part
          id ID #REQUIRED>
```

4.5.4 Musiikkidatan määrittely `%music-data`-entiteetin avulla

Esimerkissä 7 on esitetty `score.dtd`-dokumentissa määritelty `%music-data`-entiteetti, joka määrittelee MusicXML-dokumenttien varsinaisen datasisällön muodostavat elementit. Luvussa 4.3.4 esiteltiin MusicXML-dokumentin datasisältöä ja seuraavassa käsitellään sen elementtien tyyppimäärityksiä.

MusicXML-dokumentin attribuutit: `attributes.dtd`-dokumentissa on määritelty MusicXML-dokumenteissa esiintyvä `<attributes>`-elementti. Tällä elementillä määritellään ominaisuuksia, jotka usein liittyvät yksittäiseen tahtiin. Siten sitä useimmiten käytetään `<measure>`-elementin lapsielementtinä. Jos dokumentin juurielementtinä on `<score-timewise>`, niin `<attributes>`-elementti kuitenkin esiintyy `<part>`-elementin lapsielementtinä. Tällöin sitä silti käytetään tahtikohtaisten ominaisuuksien määrittelyyn, sillä `<score-timewise>`-juurielementillisissä dokumenteissa yksi osa käsitellään aina tahti kerrallaan. [Recordare LLC, 2003]

Esimerkissä 9 on esitetty `<attributes>`-elementin tyyppimäärittely. Attribuutit sisältävät tietoja tahtilajista, sävellajista ja mahdollisesta transponoinnista, instrumenteista, nuottiavaimista ja muista tahtiin liittyvistä ominaisuuksista. [Recordare LLC, 2003]

Esimerkki 9. `<attributes>`-elementin tyyppimäärittely.

```
<!ELEMENT attributes (%editorial;, divisions?, key?, time?,
                      staves?, instruments?, clef*,
                      staff-details*, transpose?,
                      directive*, measure-style*)>
```

Yksittäinen nuotti MusicXML-dokumentissa: Yksittäinen nuotti on luonnollisesti MusicXML-dokumentin olennaisin elementti. Sillä määritellään dokumentin varsinaiset nuotit ja niihin liittyvät muotoiluasetukset ja esitykselliset ominaisuudet [Recordare LLC, 2003]. Yksittäisen nuotin tyyppimäärittely on erittäin pitkä ja kattava, joten siitä esitellään seuraavassa vain olennaiset asiat. Esimerkissä 10 on esitetty katkelma `<note>`-elementin tyyppimäärittelystä.

Nuotin sävelkorkeus määritellään <pitch>-elementillä. Se sisältää kolme lapsielementtiä, joilla määritellään nuotin juurisävelen nimi (<step>), sävelen oktaaviala (<octave>) ja mahdollinen kromaattinen sävelkorkeuden muutos (<alter>). Kromaattinen sävelkorkeuden muutos ilmoitetaan kokonaislukuna ja yksi kokonaisluku merkitsee puolisävelaskeleen muutosta nuotin sävelkorkeuteen. Esimerkiksi yksi alennusmerkki ilmoitetaan arvolla `-1` ja korotusmerkintä vastaavasti arvolla `1`. Tauot musiikissa määritellään <pitch>-elementin sijasta tyhjällä <rest>-elementillä. [Recordare LLC, 2003]

<duration>-elementillä määritellään nuotin soiva kesto. Nuottien aika-arvojen määrittelyä käsiteltiin jo luvussa 4.3.4. <time-modification>-elementillä voidaan määritellä nuotin kesto, jos se poikkeaa kappaleen normaalista pulssista. Tällaisia poikkeuksia ovat esimerkiksi triolit ja muut vastaavanlaiset epätavalliset rytmit. [Recordare LLC, 2003]

Yksittäisen nuotin esitykselliset ominaisuudet määritellään sekä <note>-elementin attribuuteilla että sen lapsielementeillä. Attribuutit `dynamics`, `end-dynamics`, `attack` ja `release` vastaavat suoraan MIDI-standardin vastaavia voimakkuusarvoja ja ne määritellään prosentuaalisina muutoksina MIDI:n oletusarvoista. Jos yksittäinen nuotti on tarkoitettu soitettavaksi jollain toisella instrumentilla, kuin mille osa (<part>) on kirjoitettu, se voidaan määritellä <instrument>-elementillä [Recordare LLC, 2003].

Kuten <note>-elementin tyyppimäärityksestä käy ilmi, se on erittäin laaja ja kattava. Muita elementtejä ovat esimerkiksi korunuotin (<grace>), sidotun nuotin (<tie>), nuotin varren (<stem>) ja palkin suunnan (<beam>) määrittelemiseksi käytetyt elementit [Recordare LLC, 2003]. Suurin osa näistä ominaisuuksista liittyy kuitenkin nuottikuvan graafiseen ulkoasuun, joten niitä ei käsitellä tässä tämän tarkemmin.

Esimerkki 10. <note>-elementin tyyppimäärittely.

```
<!ELEMENT note
    (((grace, %full-note;, (tie, tie?)?) |
      (cue, %full-note;, duration) |
      (%full-note;, duration, (tie, tie?)?)),
        instrument?, %editorial-voice;, type?, dot*,
        accidental?, time-modification?, stem?, notehead?,
        staff?, beam*, notations*, lyric*)>
```

```
<!ENTITY % full-note "(chord?, (pitch | unpitched | rest))">

<!ELEMENT pitch (step, alter?, octave)>
<!ELEMENT step (#PCDATA)>
<!ELEMENT alter (#PCDATA)>
<!ELEMENT octave (#PCDATA)>

<!ELEMENT duration (#PCDATA)>

<!ELEMENT chord EMPTY>

<!ATTLIST note
  %position;
  %printout;
   dynamics CDATA #IMPLIED
   end-dynamics CDATA #IMPLIED
   attack CDATA #IMPLIED
   release CDATA #IMPLIED
   pizzicato %yes-no; #IMPLIED>
```

Moniäänisen musiikin kuvaus MusicXML-dokumentissa: Tähän mennessä esimerkkikappaleen sisältö on ollut sellaista musiikkia, jossa on ollut vain yksi nuotti nuottiviivastolla kerrallaan. Esimerkkikappaleen sello on esimerkki polyfonisesta instrumentista, jolla voidaan soittaa useampi sävel samanaikaisesti. MusicXML:ssä on kaksi vaihtoehtoista tapaa määritellä samanaikaisesti soivia nuotteja.

Esimerkissä esitellyllä tyhjällä `<chord>`-elementillä määritellään nuotin kuuluminen sointuun. Jos `<note>`-elementillä on `<chord>`-lapsielementti se tarkoittaa sitä, että se kuuluu samaan sointuun edellisessä `<note>`-elementissä määritellyn nuotin kanssa. [Recordare LLC, 2003]

Toinen vaihtoehto moniäänisyyden määrittelemiseksi on käyttää `<backup>`- ja `<forward>` -elementtejä. Näillä elementeillä liikutaan nuottiviivastolla ajallisesti eteen- ja taaksepäin. Molempien elementtien tärkein lapsielementti on `<duration>`, jolla määritellään kuinka pitkä matka ajallisesti nuottiviivastolla liikutaan. `<duration>`-elementin arvon tulee aina olla positiivinen kokonaisluku ja sen aika-arvo määritellään samoin kuin nuottien ja taukojen aika-arvot. [Recordare LLC, 2003]

5 Muut käytetyt tekniikat

Luvussa kuvataan tässä tutkielmassa kehitettävän sovelluksen kehitystyössä käytettäviä tekniikoita ja työkaluja. Luvussa 5.1 kuvataan XML-työkaluja ja luvussa 5.2 esitellään äänen muodostuksessa käytettäviä menetelmiä ja työkaluja. Luvussa 5.3 esitellään säikeiden ja ajastimien hallinnassa hyödynnettäviä työkaluja ja luvussa 5.4 käyttöliittymän ohjelmoinnissa käytettävää *Swing*-kirjastoa.

5.1 XML-tekniikat

Luvussa kuvataan tässä tutkielmassa kehitettävän sovelluksen kehitystyössä käytettäviä XML-tekniikoita ja työkaluja. Luvussa 5.1.1 kerrotaan aluksi XML-jäsentimistä yleensä esitellen kaksi erilaista lähestymistapaa XML-dokumenttien jäsentämiseen. Luvussa 5.1.2 kuvataan SAX2-rajapinnan ja sen toteuttavan *Xerces* -XML-jäsentimen käyttöä. Lopuksi luvussa 5.1.3 kuvataan yleistä JAXP-rajapintaa.

5.1.1 XML-jäsentimet

XML-dokumenttien käsittelyyn on olemassa useita valmiita kirjastoja. Näistä kirjastoista käytetään termejä *XML-jäsennin* tai *XML-parseri* (*XML Parser*). Valmiin XML-jäsentimen käytöstä saatava hyöty on se, ettei ohjelmoijan tarvitse jokaista sovellusta varten kirjoittaa uusia rutiineja XML-dokumenttien käsittelyyn. XML-jäsentimet lukevat XML-dokumentteja ja tarjoavat niiden sisältämän informaation sovellukselle standardimuotoisen rajapinnan kautta [Nykänen, 2001, sivu 60]. Lisäksi useimmat XML-jäsentimet osaavat myös tarkistaa, validoida, että dokumentti on tyypiltään ja loogiselta rakenteeltaan annetun tyyppimäärityksen mukainen. [Nykänen, 2001, sivu 139]

XML-jäsentimien rajapintojen toteutuksissa on käytetty kahta toisistaan poikkeavaa lähestymistapaa: *tapahtumapohjaista* ja *mallipohjaista* jäsentämistä. Näiden rajapintojen nimet ovat *SAX* (*Simple API for XML*) ja *DOM* (*Document Object Model*). [Roy, 2001, sivu 40; Nykänen, 2001, sivut 64-65]

Tässä kohden on syytä korostaa, että SAX ja DOM eivät ole varsinaisia XML-jäsentimiä. Ne ovat ainoastaan *liittymän* (*interface*) määritteleviä ohjelmointirajapintoja, jonka SAX:n tai DOM:n toteuttavan XML-jäsentimen on toteutettava [McLaughlin, 2000, sivu 13]. Liittymä on verrattavissa olio-ohjelmoinnissa usein käytettyyn termiin *abstrakti luokka*. Jokin konkreettinen luokka edelleen toteuttaa liittymän. Liittymä itse ei sisällä mitään toiminnallisuutta, vaan vain määrittelee rajapinnan, jonka liittymän toteuttavan luokan on toteutettava [Flanagan, 1999, sivu 112].

SAX- ja DOM -rajapinnan toteuttavia XML-jäsentimiä on olemassa lukuisia, joista tässä tutkielmassa kehitettävän sovelluksen kehitystyössä käytettävää *Xerces Java 2* -jäsennintä esitellään tarkemmin luvussa 5.1.2.

5.1.1.1 DOM

DOM on lyhenne sanoista *Document Object Model* ja se on W3C:n suosittama rajapinta XML-dokumenttien käsittelyyn. Sen uusin versio tätä kirjoitettaessa on 2.0 ja siitä käytetään termiä *DOM Level 2*. [W3C, 2003]

DOM-rajapinnan toteuttavat XML-jäsentimet lukevat XML-dokumentin kokonaisuudessaan tietokoneen muistiin ja tarjoavat sen sisältämän informaation sovellukselle puumaisena rakenteena [Roy, 2001, sivu 40]. Kaikki XML-dokumentin sisältämä informaatio on sovelluksen saatavilla puurakenteen solmuissa (*node*) ja datan manipulointi ja liikkuminen puurakenteessa tapahtuu näiden solmujen avulla [Simeoni, 2003, sivu 21]. Koska DOM-rajapinnan tarjoama puurakenne on kokonaisuudessaan tietokoneen keskusmuistissa, XML-dokumentin sisältämän informaation manipulointi on helppoa ja nopeaa. Puussa liikkuminen ja tiedon käsittely on suoraviivaista ja tehokasta, sillä puumaisten tietorakenteiden käsittelyyn tarkoitetut algoritmit ovat vuosikymmenten kuluessa kehittyneet nopeiksi ja optimoiduiksi [McLaughlin, 2000, sivu 13]. Lisäksi nykyiset DOM-rajapinnan toteuttavat XML-jäsentimet tarjoavat myös XML-dokumenttien kirjoituspalveluita mahdollistaen uusien XML-dokumenttien tuottamisen suoraan ohjelmakoodista [Nykänen, 2001, sivu 65].

DOM-rajapinnan huonoja puolia ovat suuri resurssien tarve, koko XML-dokumentin lukeminen kokonaisuudessaan muistiin ja näistä aiheutuva jäsentämisen hitaus. Keskusmuistia tarvitaan DOM-rajapinnan toteuttavan XML-jäsentimen käyttämiseen paljon ja koko XML-dokumentin on oltava saatavilla, ennen kuin sovellukselle voidaan tarjota mitään pääsyä XML-dokumentin sisältämään informaatioon. Lisäksi XML-dokumentin on oltava hyvin muodostettu ja validi, jotta DOM-rajapinnan toteuttava XML-jäsennin pystyy sitä käsittelemään. [McLaughlin, 2000, sivu 13]

Tässä tutkielmassa toteutettavan sovelluksen käyttötarpeet huomioiden DOM tarjoaa hyvät työvälineet MusicXML-dokumenttien käsittelyyn, sillä musiikki-informaation monimutkaiset sisäiset hierarkiat ovat helposti saatavilla puumaisessa rakenteessa. DOM:n käytöstä aiheutuva hitaus rajoittaa kuitenkin soivan ääninäytteen muodostamista ja jopa estää soiton aloittamisen heti dokumentin lukemisen alettua.

5.1.1.2 SAX

SAX on lyhenne sanoista *Simple API for XML* ja sen lähestymistapa XML-dokumenttien jäsentämiseen poikkeaa DOM:n lähestymistavasta paljon. SAX:n perusajatuksena on reaaliaikaisesti etenevä tapahtumapohjainen XML-dokumenttien käsittely. SAX oli aikanaan ensimmäinen yleisesti käytetty ohjelmointirajapinta XML:n käsittelyyn. SAX on alun perin kirjoitettu Java-ohjelmointikielelle, mutta nykyään rajapinnan toteuttavia XML-jäsentimiä on olemassa myös muille ohjelmointikielille. SAX:n uusin versio on tätä kirjoitettaessa (8.10.2003) 2.01 ja siitä käytetään yleisesti lyhennettä SAX2. [McLaughlin, 2000, sivut 46-48]

SAX-rajapinnan toteuttava XML-jäsennin jäsentää XML-dokumenttia tapahtumapohjaisesti. Jäsennin laukaisee tapahtumia sitä mukaa kun se kohtaa elementtejä dokumenttia lukiessaan [Nykänen, 2001, sivu 64]. Tapahtumapohjaisuudestaan johtuen koko XML-dokumentin ei tarvitse olla käytettävissä jäsentämisen alkaessa, kuten on esimerkiksi DOM:n tapauksessa [Roy, 2001, sivu 40; McLaughlin, 2000, sivut 12-13].

Kun SAX-rajapinnan toteuttava XML-jäsennin aloittaa XML-dokumentin lukemisen, se laukaisee tapahtumia kuten *dokumentin alku*, *dokumentin loppu*, *elementin alku* ja *elemen-*

tin loppu. Vastaavanlaiset tapahtumat ovat olemassa myös XML-dokumentin sisältämille mahdollisille virheille. XML-sovellusta kehittävän ohjelmoijan tehtävä on kirjoittaa tapahtumankäsittelijät näille tapahtumille. Tapahtumia käsitteleviä funktioita tai -metodeita kutsutaan takaisinkutsufunktioiksi tai -metodeiksi (*Callback methods*). [Simeoni, 2003, sivu 21; McLaughlin, 2000, sivut 13 ja 64]

Näiden tapahtumankäsittelijöiden avulla SAX-rajapinnan toteuttava XML-jäsennin tarjoaa ohjelmoijalle helpon ja yksinkertaisen liittymän XML-dokumentin sisältämään dataan. SAX-rajapinnan tapahtumankäsittelijöistä kerrotaan tarkemmin seuraavassa luvussa Xerces XML-jäsentimen yhteydessä.

Tässä tutkielmassa toteutettavan sovelluksen näkökulmasta jäsentimen nopeus on kriittinen tekijä soivan ääninäytteen muodostamisessa MusicXML-dokumentista. SAX-jäsentimien ollessa nopeampia kuin DOM-jäsentimet, SAX-rajapinta tuntuisi sopivammalta rajapinnalta MusicXML-dokumenttien jäsentämiseen. Lisäksi tapahtumat mahdollistavat MusicXML-dokumentin auditiiviseksi esitykseksi muuntamisen aloittamisen heti kun dataa on saatavilla. Toisaalta musiikki-informaation sisäiset hierarkiat rajoittavat täysin tapahtumapohjaiseen MusicXML-dokumentin jäsentämiseen perustuvaa toteutusta. Luvussa 2.3.2 esitellyt asiat, kuten musiikki-informaation kontekstiriippuvuus, pakottavat luopumaan ainoastaan tapahtumiin perustuvasta XML-dokumentin jäsentämisestä.

MusicXML-dokumentit sisältävät myös paljon sellaista informaatiota, joka on toteutettavan sovelluksen kannalta täysin merkityksetöntä. Tällaista informaatiota ovat esimerkiksi graafisen viivastonotaation symbolien merkkaukseen käytettävät elementit ja attribuutit. SAX:n avulla tällainen informaatio voidaan yksinkertaisesti jättää käsittelemättä ja täten säästää järjestelmän resursseja.

5.1.2 SAX2-rajapinnan ja Xerces-jäsentimen käyttö

Xerces on WWW-palvelimestaan tunnetun yhdysvaltalaisen *Apache*-projektin kehittämä XML-jäsenninkirjasto. Xerces sisältää lähes kaikki XML-dokumenttien käsittelyssä kuviteltavissa olevat ominaisuudet, kuten SAX- ja DOM-rajapinnat toteuttavat XML-jäsentimet [Nykänen, 2001, sivu 55]. Xerces on yksi maailmalla eniten käytetyistä XML-jäsenti-

mistä [McLaughlin, 2000, sivu 47]. Se on täysin ilmainen ja tätä kirjoitettaessa (8.10.2003) sen uusin versio on 2.5.0. Xerces on saatavissa usealla ohjelmointikielelle mukaan lukien Java, C++ ja Perl.

Seuraavissa kappaleissa esitellään SAX2-rajapinnan ja *Xerces Java 2* -XML-jäsentimen käyttöä esimerkkien avulla. Jäsentimen käyttöä esitellään tässä tutkielmassa toteutettavan sovelluksen kannalta oleellisin osin. Luku perustuu lähteeseen [Megginson, 2002] ellei toisin mainita.

XML-dokumentin alku- ja lopputapahtumien käsittely: XML-jäsentimen lukiessa XML-dokumenttia se laukaisee ensimmäisenä dokumentin alku -tapahtuman. Esimerkissä 11 on esitetty tämän tapahtuman käsittelyyn kirjoitettu tapahtumankäsittelijämetodi. Esimerkissä on mukana myös vastaava dokumentin loppumisen käsittelevä metodi. Esimerkin metodit vain tulostavat näytölle ilmoitukset dokumentin alkamisesta ja loppumisesta.

Esimerkki 11. XML-dokumentin alku- ja lopputapahtumien takaisinkutsumetodit.

```
public void startDocument() throws SAXException {
  System.out.println("Parsing started.");
}

public void endDocument() throws SAXException {
  System.out.println("Parsing ended.");
}
```

Elementtien alku- ja lopputapahtumien käsittely: Elementin alku- tai lopputunnisteen kohdatessaan XML-jäsennin laukaisee tapahtuman `startElement` tai `endElement`. Esimerkissä 12 on kuvattu näille tapahtumille kirjoitetut tapahtumankäsittelijämetodit. Metodit saavat jäsentimeltä parametreinaan elementin nimen kokonaisuudessaan, sen *nimiavaruuden* (*namespace*), paikallisen nimen sekä elementin sisältämät attribuutit. Esimerkin metodit tulostavat näytölle ilmoituksen elementin alkamisesta ja loppumisesta.

Esimerkki 12. Elementin alku- ja lopputapahtumien takaisinkutsumetodit.

```
public void startElement(String l_strNameSpaceURI,
                         String l_strLocalName,
                         String l_strRawName,
                         Attributes l_attributes)
                         throws SAXException {
  System.out.println("Element started: <" +
                     l_strRawName + ">");
}

public void endElement(String l_strNameSpaceURI,
                       String l_strLocalName,
                       String l_strRawName)
                       throws SAXException {
  System.out.println("Element ended: <" +
                     l_strRawName + ">");
}
```

Elementin sisältämän tiedon käsittely: XML-dokumentin elementtien varsinainen tietosisältö käsitellään takaisinkutsumetodilla `characters`. Metodi saa parametrinaan XML-jäsentimeltä merkkijonotaulukon, joka sisältää elementin tietosisällön. Lisäksi se saa parametreinaan alku- ja loppuindeksit merkkijonotaulukkoon. Esimerkissä 13 on esitetty takaisinkutsumetodi, joka muuntaa saamansa merkkijonotaulukon `String`-luokan olioksi ja tulostaa sen näytölle.

Esimerkki 13. Elementin sisältämän tiedon käsittely.

```
public void characters(char[] ch, int start, int end)
                       throws SAXException {
  String l_strElementData = new String(ch, start, end);
  System.out.println("Element data: " + l_strElementData);
}
```

Edellä käsiteltyjen esimerkkien lisäksi SAX2-rajapinnan toteuttava XML-jäsennin tuottaa tapahtumat myös virheille, joiden käsittely tapahtuu vastaavanlaisissa takaisinkutsumetodeissa kuten `error` ja `warning`. Lisätietoja Xerces-jäsentimestä löytyy WWW-muodossa valmistajan Internet-sivuilta osoitteesta `http://xml.apache.org/`.

5.1.3 JAXP

JAXP on lyhenne sanoista *Java API for XML Parsing*. Se on yhdysvaltalaisen Sun Microsystems:n toteuttama ohjelmointirajapinta XML-jäsentimien käyttämiseen Java-ohjelmointikielessä. XML-tekniikoiden yleistyessä ja kehittyessä myös sekaannus eri jäsentimien ja rajapintojen käytössä on lisääntynyt. JAXP on tarkoitettu helpottamaan tätä sekavuutta tarjoten yhtenäisen rajapinnan erilaisten XML-jäsentimien käyttämiseksi. [McLaughlin, 2000, sivu 14]

Haluttaessa vaihtaa käytetty XML-jäsennin johonkin toiseen jäsentimeen, on muutos pahimmassa tapauksessa ilman JAXP:n käyttöä vaatinut koko ohjelmiston muokkausta ja uudelleen kääntämistä. JAXP:n avulla käytetty jäsennin on kokonaan piilotettu rajapinnan taakse ja sen vaihtaminen on mahdollista ilman muutoksia itse ohjelmakoodiin. Käytännössä tämä tarkoittaa sitä, että ohjelmakoodista voidaan jättää pois kaikki viittaukset tiettyyn XML-jäsennintoteutukseen ja käytettävä jäsennin määritellään erillään itse ohjelmakoodista esimerkiksi *jar*-pakettien metatietojen avulla. Varsinainen XML-dokumenttien lukeminen ja käsitteleminen tapahtuu täsmälleen samalla tavoin kuin esimerkiksi Xerces Java 2 -jäsentimen käyttö suoraan ilman JAXP-rajapintaa [McLaughlin, 2000, sivut 200-201]

JAXP kuuluu Java-ohjelmointikielen peruskehityspakettiin (J2SE) ja sen uusin versio tätä kirjoitettaessa (8.10.2003) on 1.2. Lisätietoja JAXP-ohjelmointirajapinnasta saa WWW-muodossa valmistajan Internet-sivuilta osoitteesta `http://java.sun.com/xml/jaxp/`.

5.2 Java Sound API

Luvussa kerrotaan tietokoneiden äänikorteista, niiden toiminnoista ja liitännöistä sekä kuvataan tutkielmassa kehitettävän sovelluksen kehitystyössä käytettävää *Java Sound API* -ohjelmointirajapintaa. Ohjelmointirajapinnan esittelyssä keskitytään toteutettavan sovelluksen kannalta oleellisiin toimintoihin. Luku perustuu lähteeseen [Sun Microsystems Inc., 2001] ellei toisin ilmoiteta.

5.2.1 Tietokoneet ja äänikortit

Äänen muodostamiseen tietokoneella tarvitaan siihen tarkoitettu äänipiiri, joka yleensä löytyy tietokoneeseen liitettävältä äänikortilta. Äänikortti hoitaa kaikki audion käsittelyssä tarvittavat toiminnot ja esimerkiksi kaiuttimet voidaan liittää suoraan äänikortilla oleviin ulostuloliittimiin. Äänikorttien liitäntöihin kuuluvat äänisignaalin sisään- ja ulostulot sekä mikrofoniliitäntä audion äänittämiseen. Äänikorttien toiminnallisuus voidaan jakaa kahteen osaan: ”oikean” äänen muodostamisessa tarvittaviin toimintoihin ja luvussa 3 käsitellyn synteettisen, digitaalisen musiikin MIDI:n käsittelyssä tarvittaviin toimintoihin. ”Oikeasta” äänestä käytetään yleisesti termiä *audio* ja se on äänikorttien eniten käytetty äänen käsittelytapa.

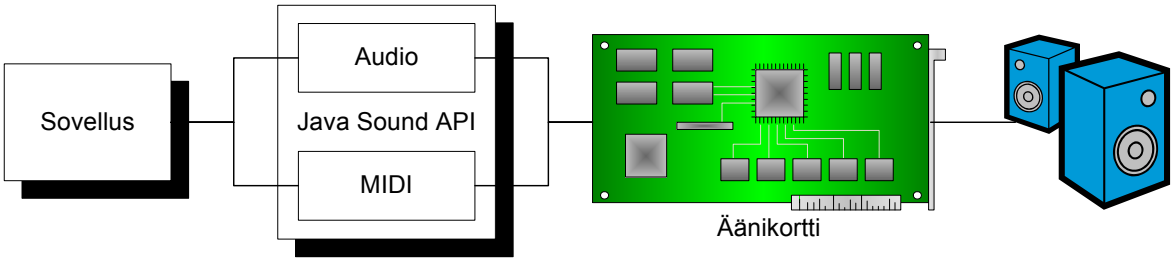
MIDI:n käsittelyyn on äänikortilla (tosin kortin laadusta ja hinnasta riippuen) *MIDI-syntetisaattori*, joka toimii samoin kuin mikä tahansa erillinen MIDI-laite. Se kykenee muuntamaan MIDI-tiedon audioksi, joka edelleen voidaan toistaa äänikortin audion ulostuloliitäntöjen kautta. Lisäksi äänikortilla on usein *MIDI-portti*, johon voidaan kytkeä ulkoisia MIDI-laitteita kuten koskettimistoja tai syntetisaattoreita. MIDI-portin avulla näitä laitteita voidaan ohjata tietokoneelta käsin.

5.2.2 Java Sound API yleisesti

Java Sound API on Sun Microsystems:n kehittämä ohjelmointirajapinta ja se on kuulunut J2SE:iin (*Java 2 Standard Edition*) versiosta 1.3 lähtien. Se tarjoaa ohjelmoijille matalan tason ohjelmointirajapinnan äänikortin toimintoihin Java-ohjelmointikielellä ja se on alusta asti suunniteltu olemaan mahdollisimman joustava ja helposti laajennettavissa. Seuraavassa listassa on lueteltu Java Sound API:n tarjoamia palveluita.

- järjestelmän ääniresursseihin pääsy ja niiden käyttö,
- erilaisten äänitiedostojen luku ja kirjoitus,
- eri tiedostomuotojen tuki ja muunnokset niiden välillä.

Aivan kuten tietokoneiden äänikorttien toiminnot, myös Java Sound API jakautuu kahteen pääosioon, audion ja MIDI:n käsittelyyn. Kuvassa 9 on havainnollistettu Java Sound API:n toimintaa sovelluksen ja äänikortin välisenä rajapintana.



Kuva 9. Java Sound API, sovelluksen ja äänikortin välinen rajapinta.

Tutkielmassa toteutettavassa sovelluksessa äänen muodostukseen käytettävää Java Sound API:n MIDI-osuutta kuvataan tarkemmin seuraavassa luvussa. Audion käsittelyä Java Sound API:n keinoin ei käsitellä tässä tutkielmassa.

5.2.3 Java Sound API:n MIDI-osuuden käyttö

Kaikki MIDI-viestit ovat binäärisiä viestejä, joita laitteet lähettävät ja vastaanottavat bitti ja tavu kerrallaan. Java Sound API piilottaa MIDI-viestien yksityiskohtaisen teknisen toteutuksen yhtenäisen ja helppokäyttöisen rajapinnan taakse. Se tarjoaa sovelluskehittäjälle yhtenäisen rajapinnan avulla kaikki käyttöjärjestelmän tarjoamat MIDI-palvelut, joita ovat esimerkiksi äänikortin MIDI-syntetisaattori ja MIDI-portti.

Java Sound API:n MIDI-osuus voidaan jakaa kahteen pääosioon: reaaliaikaiseen MIDI-datan lähettämiseen ja vastaanottamiseen sekä staattisten, pysyvien MIDI-tiedostojen käsittelyyn. Tässä tutkielmassa toteutettavassa sovelluksessa MIDI:ä käytetään reaaliaikaisesti tuottamaan ääntä yksi sävel kerrallaan, joten seuraavissa kappaleissa paneudutaan Java Sound API:n reaaliaikaiseen MIDI-datan lähettämis- ja vastaanottamisosuuteen. Kaikki Java Sound API:n MIDI-osuuden luokat löytyvät `javax.sound.midi`-paketista.

MIDI-järjestelmä (MIDI System): `MidiSystem`-luokan kautta ohjelmoija pääsee käsiksi kaikkiin käyttöjärjestelmän MIDI-palveluihin. `MidiSystem`-luokka sisältää metodit esimerkiksi MIDI-laitteiden käyttöönsaamiseksi ja MIDI-tiedostojen lukemiseksi. `MidiSystem`-luokasta ei voi tehdä ilmentymiä, sillä sen kaikki metodit ovat abstrakteja.

MIDI-laitteet (MIDI Devices): MIDI-laitteita Java Sound API:ssa edustavat `MidiDevice`-liittymästä perityt `Synthesizer`- ja `Sequencer`-liittymät. `MidiDevice`-liittymän toteuttavat luokat tarjoavat kaikki palvelut, jotka MIDI-IN tai MIDI-OUT -porttien tulee tarjota. `Synthesizer`-liittymä edustaa syntetisaattoria ja `Sequencer` sekvensseriä. Syntetisaattori on ainoa luokka Java Sound API:n MIDI-osuudessa, jolla tuotetaan korvin kuultavaa ääntä. Sekvensserillä luetaan ja kirjoitetaan MIDI-tapahtumien sarjoja, eli sekvenssejä, ja välitetään niitä edelleen toisille MIDI-laitteille. Sekvenssereitä käytetään erityisesti MIDI-tiedostojen käsittelemiseen.

Vastaanottimet (Receivers) ja välittimet (Transmitters): Useimmat MIDI-laitteet pystyvät sekä lähettämään että vastaanottamaan MIDI-viestejä. Vastaanottimet (`Receiver`) ja välittimet (`Transmitter`) ovat liittymiä, joiden avulla MIDI-viestien lähettäminen Java Sound API:ssa tapahtuu. Esimerkiksi syntetisaattorilla on vastaanotin, johon sovellus lähettää syntetisaattorille tarkoitetut MIDI-viestit. Vastaavasti esimerkiksi MIDI-IN-portilla on välitin, josta sovellus lukee portista sovellukselle saapuvat MIDI-viestit. Lisäksi Java Sound API:n avulla on mahdollista ketjuttaa vastaanottimia ja välittimiä siten, että MIDI-tieto voi kulkea vapaasti laitteelta toiselle.

MIDI-viestit (MIDI Messages): `MidiMessage` on abstrakti luokka, joka toimii kantaluokkana kaikille MIDI-viestejä edustaville luokille. Tästä luokasta periytyvät `ShortMessage`, `SysexMessage` ja `MetaMessage`. `ShortMessage` edustaa kaikkia äänen tuottamiseen liittyviä viestejä, `SysexMessage` valmistajakohtaisia erikoisviestejä ja `MetaMessage` MIDI-tiedostoissa esiintyviä metaviestejä. Kaikkia näiden luokkien edustamia olioita voidaan suoraan lähettää jonkin MIDI-laitteen vastaanottimelle ja Java Sound API huolehtii viestien toimittamisen teknisestä toteutuksesta.

Ohjelmakoodiesimerkki: Esimerkissä 14 on esitetty yhden sävelen soitto Java Sound API:n tarjoamin keinoin. Esimerkin ohjelmassa asetetaan ensin `Synthesizer`-liittymän toteuttava olio osoittamaan järjestelmän oletussyntetisaattoria kutsumalla `MidiSystem`-luokan metodia `getSynthesizer`. Tämän jälkeen `Synthesizer` pitää avata kutsumalla sen metodia `open`. Kun `Synthesizer` avataan, Java Sound API varaa sen osoit-

taman MIDI-laitteen käyttöjärjestelmältä sovelluksen käyttöön, jotta muut sovellukset eivät pääse käyttämään sitä samanaikaisesti. Tämän jälkeen asetetaan `Receiver`-liittymän toteuttava olio osoittamaan edellisillä riveillä luodun `Synthesizer`-olion vastaanotinta kutsumalla `Synthesizer`-olion metodia `getReceiver`. Tämän jälkeen sovellus on valmis käyttämään vastaanotinta MIDI-viestien lähettämiseen.

Seuraavaksi vuorossa on MIDI-viestien luominen ja lähettäminen vastaanottimelle. `ShortMessage`-luokassa on valmiit kentät edustamaan eri MIDI-käskyjä, joita ovat esimerkiksi `NOTE_ON` (ääni päälle -viesti), `NOTE_OFF` (ääni pois -viesti) ja `PROGRAM_CHANGE` (instrumentinvaihtoviesti). Esimerkin sovellus luo yhden `ShortMessage`-luokan olion ja asettaa sen edustamaan ääni päälle -viestiä kutsumalla `setMessage`-metodia. Esimerkin MIDI-viestin kentät ovat vasemmalta oikealle viestin tyyppi, kanava, soitettavan sävelen sävelkorkeus ja voimakkuus. Sävelen korkeus ja voimakkuus -arvot vastaavat suoraan MIDI-standardin vastaavia arvoja (katso luku 3.4.2).

Tämän jälkeen MIDI-viesti lähetetään vastaanottavalle laitteelle kutsumalla sen vastaanotinolion metodia `send`. `send`-metodin toinen parametri on MIDI-viestin aikaleima, jolla voidaan hienosäätää viestin toteuttamisen ajoitusta vastaanottavassa laitteessa. Aikaleima on Java Sound API:n oma laajennus MIDI-viesteihin ja siten vain harva MIDI-laite tukee tätä ominaisuutta. Esimerkissä aikaleimaa ei käytetä ja se ilmoitetaan vastaanottimelle asettamalla aikaleimaparametrin arvoksi `-1`.

Seuraavilla riveillä sama MIDI-viesti muutetaan edellisen äänen kumoavaksi ääni pois -viestiksi ja lähetetään se vastaanottimelle. MIDI-laitteet tulee aina käytön jälkeen sulkea, jotta ne olisivat muiden sovellusten käytettävissä. MIDI-laitteen sulkeminen tapahtuu kutsumalla sen metodia `close`.

Esimerkki 14. Java Sound API:n MIDI-osuuden käyttö.

```
Synthesizer synth;
synth = MidiSystem.getSynthesizer();
synth.open();
Receiver recv;
recv = synth.getReceiver();
```

```
ShortMessage midiMsg = new ShortMessage();
midiMsg.setMessage(ShortMessage.NOTE_ON, 0, 60, 100);
recv.send(midiMsg, -1);
midiMsg.setMessage(ShortMessage.NOTE_OFF, 0, 60, 100);
recv.send(midiMsg, -1);
synth.close();
```

5.3 Säikeet ja ajastimet

Luvussa kuvataan tutkielmassa toteutettavan sovelluksen kehitystyössä käytettäviä säie- ja ajastintekniikoita. Luvussa 5.3.1 kerrotaan säikeiden ja luvussa 5.3.2 ajastimien käytöstä Java-ohjelmointikielessä.

5.3.1 Java ja säikeet

Nykyaikaiset tietokoneet ja niiden käyttöjärjestelmät kykenevät suorittamaan montaa sovellusta samanaikaisesti, ja tästä ilmiöstä käytetään termiä *moniajo*. Käyttöjärjestelmä mahdollistaa moniajon samanaikaisesti ajettavilla, rinnakkaisilla *prosesseilla*. *Säie* on yhden käyttöjärjestelmässä ajettavan prosessin sisällä ajettava oma itsenäinen aliprosessi. Sen voidaan ajatella olevan ikään kuin prosessi prosessin sisällä. Sitä suoritetaan samanaikaisesti emoprosessin käynnistämien mahdollisten muiden säikeiden kanssa samassa osoiteavaruudessa. Säie ei voi omistaa muita säikeitä, vaan ne ovat aina rinnakkaisia emoprosessiin nähden. Yksiprosessorijärjestelmissä kuitenkin vain yksi prosessi tai säie voi olla kerrallaan suoritusvuorossa, ja käyttöjärjestelmä vuorottelee eri prosessien ja säikeiden välillä luoden illuusion samanaikaisesti, rinnakkain etenevistä prosesseista. [Booch, 1999, sivu 313]

Säikeitä käytetään usein suorittamaan jokin paljon aikaa vievä operaatio tai jos ohjelmassa yleensä tarvitaan useita eri yhtäaikaisia toimintoja. Graafiset käyttöliittymät, kuten esimerkiksi kaikki *Microsoft Windows* -käyttöjärjestelmässä toimivat ohjelmistot, tarvitsevat usein säikeitä toimiakseen joustavasti. Käyttöliittymä ei voi jäädä käyttäjän esimerkiksi käynnistäessä jonkin toiminnon odottamaan toiminnon päättymistä. Säikeiden avulla toteutettuna käyttöliittymä saadaan toiminnon käynnistyttyä jälleen käytettäväksi uuden säikeen suorittaessa käyttäjän aloittaman toiminnon. [Sommerville, 1995, sivu 289]

Javan peruspaketissa `java.lang` oleva `Thread`-luokka on säikeiden perusluokka. Java-ohjelmointikielessä on kaksi vaihtoehtoista tapaa toteuttaa ajettava säie. Se voidaan luoda perimällä oma luokka `Thread`-luokasta ja kirjoittamalla uudelleen sen `run`-metodi. Toinen tapa on luoda `Runnable`-liittymän toteuttava luokka kirjoittaen siihen `run`-metodi ja antaa tämä luokka `Thread`-luokan muodostajalle. Molemmilla tavoilla toteutettuna lopputuloksena on `Thread`-olio, joka voidaan käynnistää kutsumalla sen metodia `start`. `start`-metodin kutsumisen jälkeen uusi säie alkaa suorittaa `run`-metodiin kirjoitettua ohjelmakoodia ja alkuperäinen käynnistävä säie jatkaa suoritustaan välittömästi säikeen käynnistyttyä. [Flanagan, 1999, sivu 149]

Esimerkissä 15 on esitetty Java-ohjelmointikielen kaksi vaihtoehtoista tapaa luoda ja käynnistää säie. Säikeet luova pääohjelma (tai pääsäie) luo ensin kaksi säiettä ja käynnistää ne. Käynnistyttyään uudet säikeet suorittavat `run`-metodiinsa kirjoitettua ohjelmakoodia ja pääohjelma jatkaa omaa suoritustaan välittömästi säikeiden käynnistyttyä.

Esimerkki 15. Säikeiden käyttö Java-ohjelmointikielessä.

```
/* A thread by subclassing the Thread class.        */
class SubclassedThread extends Thread {
  public void run() {
    /* Time consuming task.                         */
  }
}

/* A thread by implementing the Runnable interface. */
class ImplementedThread implements Runnable {
  public void run() {
    /* Time consuming task.                         */
  }
}

/* Create the first thread and start it.            */
SubclassedThread thread1 = new SubclassedThread();
thread1.start();

/* Create the second thread and start it.           */
ImplementedThread myTask = new ImplementedThread();
Thread thread2 = new Thread(myTask);
thread2.start();
```

Tässä tutkielmassa toteutettava sovellus tulee käyttämään säikeitä kahteen eri tarkoitukseen: MusicXML-dokumentin jäsentämiseen ja siinä olevan musiikki-informaation auditiiviseksi esitykseksi muodostamiseen. Molemmat tehtävät vievät paljon aikaa ja sovelluksen käyttöliittymän on pysyttävä käytettävissä näiden molempien tehtävien edetessä taustalla.

5.3.2 Ajastimet Java-ohjelmointikielessä

Ajastimilla voidaan toteuttaa säännöllisin väliajoin tapahtuvia toimintoja ja tehtäviä. Tällaisia toimintoja voivat olla esimerkiksi käyttöliittymän päivitys tai vaikkapa tiedon tallentaminen tietyin väliajoin. Ajastimien toteutusta varten Java-ohjelmointikielessä on olemassa sen J2SE:n `java.util`-pakkaukseen kuuluvat `Timer` ja `TimerTask` -luokat. [Flanagan, 1999, sivut 536-537]

Ajastinta Java-ohjelmointikielessä edustaa `Timer`-luokka ja sen avulla voidaan ajastaa yksi tai useampia `TimerTask`-luokan olioita ajettavaksi kerran tai toistuvasti tietyin väliajoin. `TimerTask`-luokka toteuttaa `Runnable`-liittymän, joten siitä perittyyn luokkaan on kirjoitettava `run`-metodi. Tämä `run`-metodi suoritetaan joka kerta kun ajastin laukeaa. `Timer`-luokan olio siis luo uuden säikeen jokaista uutta tehtävää varten. Tehtävä voidaan ajastaa ajettavaksi kerran tietyllä ajanhetkellä tai toistuvasti tietyin väliajoin. [Flanagan, 1999, sivut 536-537]

Esimerkissä 16 on havainnollistettu ajastimien käyttöä ohjelmakoodiesimerkin avulla, joka päivittää ohjelman näytön yhden sekunnin välein. Esimerkissä peritään ensin oma luokka `TimerTask`-luokasta, jonka `run`-metodissa kutsutaan `UpdateDisplay`-metodia (tämän metodin toteutus ei ole tässä yhteydessä oleellista). Tämän jälkeen luodaan `Timer`-luokan olio `myTimer` ja ajastetaan oma `UpdateDisplayTask`-luokan olio ajettavaksi toistuvin väliajoin kutsumalla ajastimen `schedule`-metodia. Parametreiksi `schedule`-metodille annetaan ajastettava tehtävä, viive ensimmäiseen suorituskertaan millisekunteina ja viive toistuvien suorituskertojen välillä. Lopuksi pysäytetään ajastettu tehtävä kutsumalla `Timer`-olion `cancel`-metodia. `cancel`-metodin kutsu pysäyttää myös kaikki muut saman `Timer`-olion ajastamat tehtävät.

`schedule`-metodilla ajastettaessa toistuvien suorituskertojen välinen aika lasketaan edellisen suorituskerran päättymishetkestä. Vaihtoehtoinen tapa on käyttää `scheduleAtFixedRate`-metodia, jolla viive lasketaan edellisen suorituskerran alkamishetkestä. `scheduleAtFixedRate`-metodilla voidaan siis ajastaa suurempaa ajastustarkkuutta vaativia tehtäviä. [Flanagan, 1999, sivu 536]

Esimerkki 16. Ajastimien käyttö Java-ohjelmointikielessä.

```
/* Subclass the TimerTask.                              */
class UpdateDisplayTask extends TimerTask {
  public void run() {
    /* A task to be repeated.                           */
    UpdateDisplay();
  }
}

/* Create a timer and schedule a TimerTask object
   with it.                                             */
Timer myTimer = new Timer();
myTimer.schedule(new UpdateDisplayTask, 0, 1000);
myTimer.cancel();
```

Tässä tutkielmassa toteutettavassa sovelluksessa ajastimien käyttö tuntuu luontevalta MusicXML-dokumentin sisältämän informaation auditiiviseksi esitykseksi muuntamisessa. Kehitettävän sovelluksen on jotenkin pystyttävä toistamaan musiikki-informaation sisältämät sävelet tempossa tasaisin väliajoin, joten Java-ohjelmointikielen tarjoamat ajastimet sopinevat tähän tehtävään hyvin.

5.4 Swing

Tässä tutkielmassa kehitettävän sovelluksen käyttöliittymä toteutetaan J2SE:iin kuuluvan *Swing*-käyttöliittymäkomponenttikirjaston tarjoamin komponentein. Swing-kirjasto on Sun Microsystems:n kehittämä käyttöliittymäkomponenttikirjasto, mutta sen tarkempi esittely jää kuitenkin tämän tutkielman aihepiirin ulkopuolelle. Lisätietoa Swing-kirjastosta löytyy WWW-muodossa valmistajan Internet-sivuilta osoitteesta: `http://java.sun.com/products/jfc/tsc/index.html`.

6 Käyttötarkoitus ja vaatimusmäärittely

Luvussa kuvataan tutkielmassa kehitettävän sovelluksen käyttötarkoitusta ja sille asetettuja vaatimuksia. Luvussa 6.1 kuvataan syitä sovelluksen kehittämiselle ja sen erilaisia käyttötarkoituksia, ja luvussa 6.2 esitetään kehitettävän sovelluksen vaatimusmäärittely.

6.1 Sovelluksen käyttötarkoitus

Tutkielmassa kehitettävän sovelluksen tarkoituksena on toimia MusicXML-määrittelykielellä kuvattujen musiikkikappaleiden auditiivisen esityksen muodostajana. Lisäksi sovelluksen tarkoituksena on mahdollistaa tietoverkossa sijaitsevan MusicXML-dokumentin hakeminen ja muuntaminen soivaksi esitykseksi. Jatkossa kehitettävästä sovelluksesta käytetään termiä *MusicXML-soitin*. Seuraavassa listassa on lueteltu syitä MusicXML-soittimen kehittämiselle.

- mahdollistaa MusicXML-dokumentin sisältämän musiikki-informaation kuunteleminen,
- mahdollistaa tietoverkossa sijaitsevan MusicXML-dokumentin sisältämän musiikki-informaation kuunteleminen,
- tutkia MusicXML-määrittelykielen soveltuvuutta yleiskäyttöiseksi musiikin kuvauskieleksi,
- MusicXML-määrittelykieleen liittyvät ohjelmistot ovat tähän asti keskittyneet musiikin visuaaliseen esitykseen (graafiseen viivastonotaatioon), joten vastaavaa sovellusta ei vielä ole saatavilla.

Luvun 4.1.2 kuvassa 6 havainnollistettiin MusicXML-määrittelykielen toimintaa eri käyttötarkoitusten yhteisenä tekijänä. Tutkielmassa kehitettävä MusicXML-soitin sijoittuu kuvaan auditiivisen esityksen muodostajana.

6.2 Vaatimusmäärittely

Sovelluksen tai yleisemmin ohjelmiston *vaatimusmäärittely* (*Requirements Specification*) sisältää informaation siitä, mitä kehitettävän sovelluksen tulee tehdä. Se sisältää yksityis-

kohtaiset tiedot asioista ja toiminnoista, jotka kehitettävän järjestelmän tulee sisältää. Vaatimusmäärittely kirjoitetaan yleensä selkokielellä, eikä se ota kantaa sovelluksen toteutustapoihin tai -menetelmiin. [Sommerville, 1995, sivut 64 ja 68; Rumbaugh, 1991, sivu 150]

Vaatimuksia on kahdenlaisia: *toiminnallisia vaatimuksia* (*functional requirements*) ja *ei-toiminnallisia vaatimuksia* (*non-functional requirements*). Toiminnalliset vaatimukset ovat kuvauksia kehitettävän sovelluksen toiminnoista ja palveluista, jotka sen tulee toteuttaa. Ei-toiminnallisiin vaatimuksiin kuuluvat kaikki muut toteutettavalle sovellukselle asetetut vaatimukset ja rajoitteet. Tällaisia ovat esimerkiksi sovelluksen toteutuksessa käytettävä ohjelmointikieli tai laitteistolle asetetut rajoitukset. [Sommerville, 1995, sivu 64]

Luvussa 6.2.1 on esitetty kehitettävän sovelluksen ei-toiminnalliset ja luvussa 6.2.2 toiminnalliset vaatimukset.

6.2.1 Ei-toiminnalliset vaatimukset

Seuraavassa listassa on esitetty tutkielmassa kehitettävälle MusicXML-soittimelle asetetut ei-toiminnalliset vaatimukset.

- ohjelmoinnissa käytetään Java-ohjelmointikielen versiota 2 (Java SDK 1.4.1),
- MusicXML-dokumenttien käsittely toteutetaan SAX2-rajapinnan toteuttavalla Xerces Java 2 XML-jäsentimellä,
- käytettävässä laitteistossa on oltava äänikortti,
- äänikortissa on oltava MIDI-syntetisaattori tai MIDI-OUT-portti,
- musiikin kuvauskielenä käytetään MusicXML-määrittelykielen versiota 0.7b, joka on tätä kirjoitettaessa (26.8.2003) sen viimeisin versio,
- MusicXML-dokumentin juurielementin on oltava `<score-timewise>`, eli musiikki-informaation on oltava ajallisesti jaettua,
- auditiivinen esitys muodostetaan käyttämällä MIDI 1.0- ja General MIDI 1.0 -standardeja,
- käyttöliittymä toteutetaan Swing-kirjaston käyttöliittymäkomponenteilla.

6.2.2 Toiminnalliset vaatimukset

Taulukossa 1 on esitetty kehitettävälle MusicXML-soittimelle asetetut toiminnalliset vaatimukset.

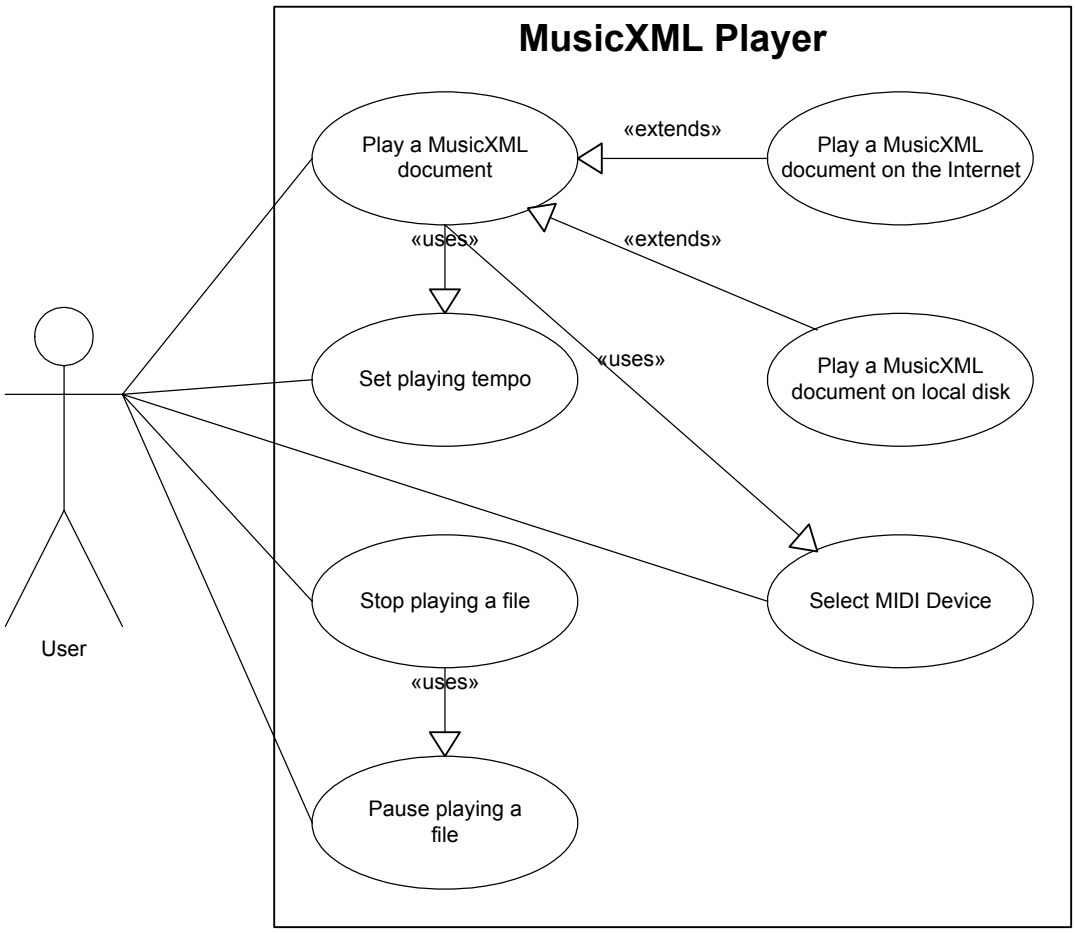
Vaatimus	Selite
Auditiiviseksi esitykseksi muunnettava MusicXML-dokumentti voi sijaita paikallisella levyllä tai Internetissä.	Paikallisella levyllä tarkoitetaan fyysisesti paikallista levyä tai verkkolevyä. Internetissä sijaitsevan dokumentin siirrossa käytetään *http*-protokollaa (*Hypertext Transport Protocol*).
Sovelluksen tulee pystyä jäsentämään ja validoimaan annettu MusicXML-dokumentti.	Dokumentin jäsennys ja validointi toteutetaan käyttämällä Xerces Java 2 -XML-jäsennintä.
Sovellus aloittaa kappaleen soittamisen heti kun MusicXML-dokumentin ensimmäinen tahti on jäsennetty.	Koko MusicXML-dokumentin ei tarvitse olla kokonaisuudessaan jäsennettynä ennen soittamisen alkamista, vaan soittaminen voidaan aloittaa heti kun ensimmäinen tahti on jäsennetty.
Sovelluksen ensimmäisen version tulee tukea erikseen määriteltyjä viivastonotaation elementtejä.	Sovelluksen tulee tukea seuraavia viivastonotaation elementtejä: nuottien sävelkorkeudet, kestot ja voimakkuudet, tauot ja niiden kestot, moniäänisyys ja soinnut, ja useampi nuottiviivasto (instrumentti).
Käyttäjän tulee voida aloittaa, keskeyttää ja lopettaa MusicXML-dokumentin soittaminen.	Käyttöliittymään toteutetaan soitto (*play*), keskeytys (*pause*) ja lopetus (*stop*) -painikkeet näiden toimintojen toteuttamiseksi.
Käyttäjän tulee voida määrittää soitettavan kappaleen tempo eli soittonopeus.	Käyttöliittymään toteutetaan tempon määritys -toiminto.

Vaatimus	Selite
Sovelluksen tulee pystyä soittamaan MusicXML-dokumentti käyttäjän määrittämällä soittonopeudella.	Sovelluksen auditiivisen esityksen muodostamisen tulee pysyä määritetyssä tahdissa eli hidastelemista ja/tai kiihdyttämistä ei saa esiintyä.
Käyttäjän tulee voida valita valittavissa olevista MIDI-laitteista käytettävä MIDI-laite.	Käyttöliittymään toteutetaan MIDI-laitteen valinta -toiminto.
Käyttäjän tulee voida valita, validoidaanko muunnettava MusicXML-dokumentti.	Käyttöliittymään toteutetaan validoinnin valinta -toiminto.

Taulukko 1. Kehitettävän MusicXML-soittimen toiminnalliset vaatimukset.

Sovelluksen päätavoitteena voidaan siis pitää auditiivisen esityksen muodostamisen aloittamista heti kun ensimmäinen tahti on jäsennetty ja tämän jälkeen MusicXML-dokumentin soiton pysymistä määritetyssä tahdissa.

Edellisen taulukon lisäksi kehitettävälle sovellukselle asetettuja toiminnallisia vaatimuksia on havainnollistettu kuvassa 10 *käyttötapauskaavion* avulla (*Use Case Diagram*) [Booch, 1999, sivut 233-244].



Kuva 10. Kehitettävän MusicXML-soittimen käyttötapauskaavio.

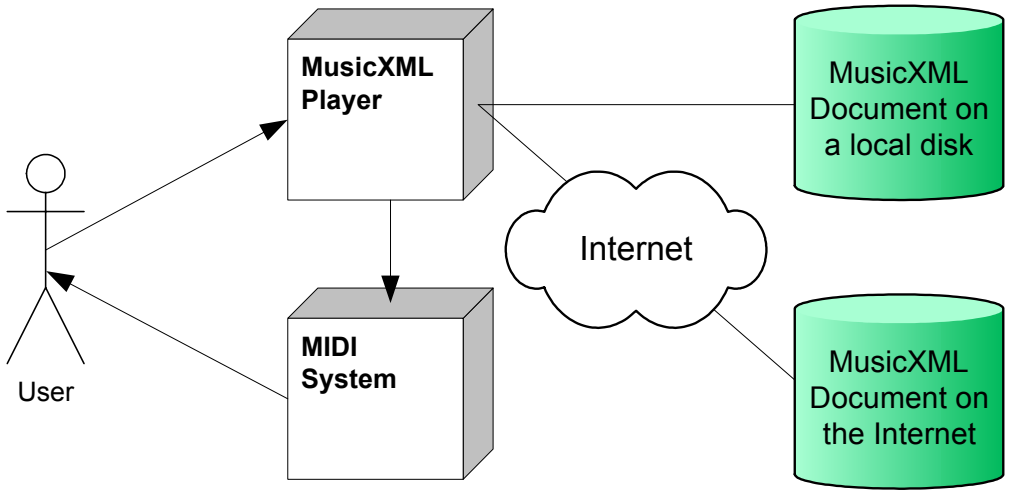
7 Sovelluksen arkkitehtuuri ja toiminta

Luvussa kuvataan tutkielmassa kehitetyn MusicXML-soittimen arkkitehtuuri ja toiminta. Luvussa 7.1 kuvataan sovelluksen arkkitehtuuri ja sen sijoittuminen ympäröivään järjestelmään ja luvussa 7.2 sovelluksen toiminta.

7.1 Sovelluksen arkkitehtuuri

Kuvassa 11 on havainnollistettu kehitetyn MusicXML-soittimen sijoittumista ympäröivään järjestelmään *sijoittelukaavion* (*Deployment Diagram*) avulla. Sijoittelukaaviosta voidaan käyttää myös nimitystä *kontekstikaavio*. [Booch, 1999, sivu 408]

Käyttötarkoituksensa mukaisesti käyttäjä käyttää MusicXML-soitinta, joka muuntaa MusicXML-dokumentin auditiiviseksi esitykseksi. Muunnettava MusicXML-dokumentti voi sijaita paikallisesti samassa tietokoneessa olevalla levyllä tai jollain toisella tietoverkkoon kytketyllä tietokoneella. Sovellus muuntaa lukemansa ja jäsentämänsä MusicXML-dokumentin auditiiviseksi esitykseksi käyttäen ulkoista tai tietokoneesta löytyvää MIDI-järjestelmää ja MIDI-järjestelmä esittää auditiivisen esityksen käyttäjälle esimerkiksi äänikortin tai jonkin ulkoisen äänentoistojärjestelmän avulla.



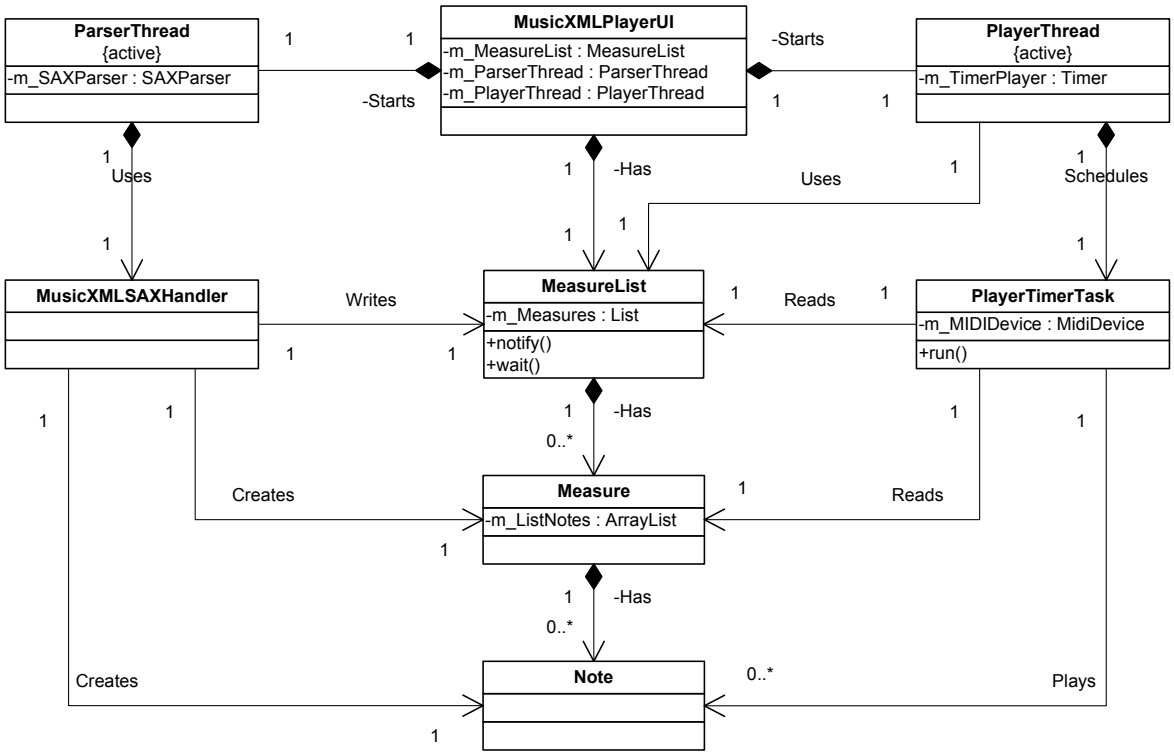
Kuva 11. MusicXML-soittimen kontekstikaavio.

7.2 Sovelluksen toiminta

Luvussa kuvataan kehitettävän MusicXML-soittimen toiminta luokkakuvausten ja UML-kaavioiden (*Unified Modeling Language Diagram*) avulla. UML-kaavioista on esitetty tämän luvun kannalta olennaiset yleistetyt versiot ja vastaavat yksityiskohtaiset versiot kaavioista on esitetty liitteen 3 kuvissa L.1-L.4. Jos liitteiden UML-kaavioihin kuitenkin viitataan, se mainitaan erikseen.

MusicXML-soittimen luokat voidaan jakaa kolmeen pääryhmään, jotka ovat *käyttöliittymäluokka* ja sen apuluokat, *tietorakenneluokat* ja *säieluokat* apuluokkineen. Sovelluksen käyttöliittymäluokkaa kuvataan seuraavassa luvussa, tietorakenneluokkia luvussa 7.2.2 ja säieluokkia luvussa 7.2.3. Luokkien välistä yhteistoimintaa kuvataan luokkakuvausten jälkeen luvussa 7.2.4.

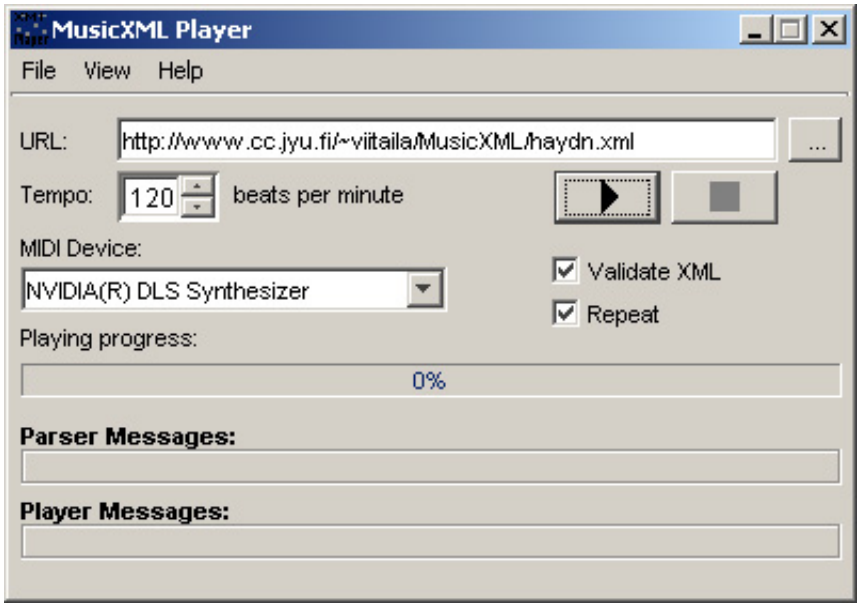
Kuvassa 12 on esitetty MusicXML-soittimen yleistetty *luokkakaavio* (*Class Diagram*) [Booch, 1999, sivut 105-106]. Kuvan luokkakaaviossa on kokonaisuuden hahmottamisen helpottamiseksi esitetty vain sovelluksen tärkeimmät luokat ja niiden olennaisimmat jäsenmuuttujat ja metodit. Sovelluksen yksityiskohtainen luokkakaavio on kuvattu kokonaisuudessaan liitteen 3 kuvassa L.1.



Kuva 12. MusicXML-soittimen yleistetty luokkakaavio.

7.2.1 Käyttöliittymäluokka

MusicXML-soittimen käyttöliittymäluokka on nimeltään `MusicXMLPlayerUI`. Sen voidaan ajatella olevan myös sovelluksen pääluokka, sillä se ohjaa koko sovelluksen toimintaa sekä luo sovelluksen tietorakenteen ja säikeet. Käyttöliittymäluokka käyttää apunaan monia muita Swing-kirjaston luokkia, mutta niiden kuvaaminen ei ole tässä yhteydessä oleellista. Kuvassa 13 on esitetty kehitetyn sovelluksen ensimmäisen version käyttöliittymä.



Kuva 13. MusicXML-soittimen ensimmäisen version käyttöliittymä.

Luokan tehtävät: Käyttöliittymäluokan tehtävä on luoda sovelluksen käyttöliittymä ja täten tarjota käyttäjälle kaikki sovelluksen toiminnallisuus. Luokan tehtävä on myös luoda tarvittavat säikeet ja sovelluksen tietorakenne. Lisäksi käyttöliittymäluokan tehtävä on välittää käyttäjän pyytämät toiminnot säieluokkien olioille.

Luokan toteutus: Käyttöliittymäluokka muodostaa käyttäjälle näkyvän käyttöliittymän Swing-kirjaston käyttöliittymäkomponenttien avulla. Käyttäjän valitessa soitettavan MusicXML-dokumentin ja painaessa soiton käynnistävää painiketta, luokka käynnistää jäsenninsäikeen (`ParserThread`) antaen sille parametrina jäsennettävän dokumentin URL:n eli sijainnin. Lisäksi parametrina välitetään jäsenninsäikeelle tieto siitä, validoidaanko jäsennettävä MusicXML-dokumentti vai ei. Tämän jälkeen luokka käynnistää soittosäikeen (`PlayerThread`) antaen sille tiedon käyttäjän määrittämästä soittonopeudesta (tempo) ja käytettävästä MIDI-laitteesta (MIDI Device). Käyttöliittymäluokka myös pysäyttää ja keskeyttää MusicXML-dokumentin soittamisen välittämällä vastaavat pyynnöt soittosäikeelle.

Käyttöliittymäluokka luo myös sovelluksen tietorakenteen ylimmän tason luokan (`MeasureList`) esiintymän, jonka nimi on `m_MeasureList`. Tietorakennetta kuvataan tarkemmin seuraavassa luvussa.

7.2.2 Tietorakenneluokat

MusicXML-soittimen tietorakenteeseen tallennetaan koko soitettava kappale ja se koostuu kolmesta luokasta: `MeasureList`, `Measure` ja `Note`. `MeasureList`-luokka edustaa järjestettyä listaa koko kappaleen tahdeista ja sen voidaan siten ajatella edustavan koko soitettavaa musiikkikappaletta. `Measure`-luokka edustaa kappaleen yhtä tahtia ja `Note`-luokka yksittäistä nuottia. Sovelluksen kaksi säieluokkaa (`ParserThread` ja `PlayerThread`) lukevat ja kirjoittavat tätä tietorakennetta. Seuraavissa luvuissa kuvataan sovelluksen kolmea tietorakenneluokkaa.

7.2.2.1 `MeasureList`-luokka

Luokan tehtävät: `MeasureList`-luokka edustaa järjestettyä listaa soitettavan kappaleen tahdeista ja se tarjoaa sovelluksen kahdelle säieluokalle tahtien kirjoitus- ja lukupalvelut.

Luokan toteutus: `MeasureList`-luokan tärkein jäsenmuuttuja on kappaleen tahdit sisältävä `m_Measures`, joka on tyypiltään *synkronoitu lista* (`syncronizedList`). Lista sisältää `Measure`-luokan olioita ja luokan tahtien kirjoitus- ja lukumetodit kirjoittavat ja lukevat tätä listaa.

Synkronoitu lista on toiminnaltaan synkronoitu. Se tarkoittaa sitä, että sen kaikki metodit ovat synkronoituja. Kun metodi on synkronoitu, vain yksi säie kerrallaan voi suorittaa sen ja vasta edellisen suorituksen päätyttyä seuraava säie saa suoritusvuoron. Tällä tavoin vältetään vaarallinen tilanne, jossa kaksi tai useampi säie muokkaa samaa oliota samanaikaisesti. [Flanagan, 1999, sivu 151]

Myös kaikki `MeasureList`-luokan metodit ovat synkronoituja, sillä sovelluksen kaksi säieluokkaa käyttävät tätä luokkaa samanaikaisesti. Toteuttamalla säikeiden synkronointi

jo MeasureList-luokassa saavutetaan se hyöty, ettei tietorakenteen muissa luokissa tarvitse huolehtia siitä lainkaan.

7.2.2.2 Measure-luokka

Luokan tehtävät: Measure-luokka edustaa yksittäistä musiikkikappaleen tahtia ja sen voidaan ajatella vastaavan suoraan MusicXML-määrittelykielen <measure>-elementtiä. Luokan tärkein tehtävä on tarjota käyttäjilleen nuottien kirjoitus- ja lukupalvelut. Luokan tehtävä on myös huolehtia nuottiensa suhteellisista kestoista ja siitä, että ne ovat määritelty yhdenmukaisesti.

Luokan toteutus: Measure-luokka tallentaa Note-luokan olioita käyttäen tallentamiseen ArrayList-säiliöluokkatyyppistä jäsenmuuttujaa m_Notes. Luokan nuottien kirjoitus- ja lukumetodit kirjoittavat ja lukevat tätä listaa.

MusicXML-määrittelykielessä nuottien kestot määritellään suhteellisina arvoina neljäsosanuotin jakajaksi asetettuun arvoon verrattuna (katso luku 4.3.4). Measure-luokka tietää nuottiensa jakajan arvon ja huolehtii siitä, että kaikkien nuottien kestot on määritelty käyttäen pienintä mahdollista kappaleessa esiintyvää jakajan arvoa. Näin mahdollistetaan musiikkikappaleen tempon määritys ja sävelten oikea-aikainen soitto ja sammutus.

MIDI-standardin mukaisesti jokainen sävel pitää soittamisen lisäksi myös sammuttaa. Sovelluksen soittosäikeen pyytäessä tietyllä ajanhetkellä soitettavia nuotteja, Measure-luokan nuottien lukumetodi palauttaa samalla myös sillä hetkellä sammutettavat sävelet.

7.2.2.3 Note-luokka

Luokan tehtävät: Note-luokka edustaa musiikkikappaleen yksittäistä nuottia ja MusicXML-määrittelykielessä sitä vastaa suoraan elementti <note>. Luokka sisältää tiedot nuotin soivasta sävelkorkeudesta, kestosta, voimakkuudesta, sijainnista tahdissa, käytettävästä MIDI-instrumentista ja MIDI-kanavasta.

Luokan toteutus: `Note`-luokka on ainoa tietorakenteen luokista, joka on tekemisissä MIDI-standardin kanssa. Se muuntaa esimerkiksi MusicXML-dokumentissa sille määritellyn sanallisen sävelkorkeuden MIDI-standardin mukaiseksi numeeriseksi arvoksi. `Note`-luokan olio voi esiintyä minä tahansa luvussa 3.4.2 esiteltynä ääniviestinä (ääni päälle-, ääni pois- tai instrumentinvaihtoviesti).

`Note`-luokka on peritty `javax.sound.midi`-paketin `ShortMessage`-luokasta. Perinnän avulla toteutettuna luokan olio voidaan sellaisenaan antaa Java Sound API:n `Synthesizer`- tai `Receiver`-luokan oliolle soitettavaksi.

7.2.3 Säieluokat

MusicXML-soittimen kaksi säieluokkaa toteuttavat sovelluksen keskeisimmän toiminnallisuuden, MusicXML-dokumentin auditiiviseksi esitykseksi muuntamisen. Jäsenninsäie (`ParserThread`) lukee ja jäsentää annetun MusicXML-dokumentin ja soittosäie (`PlayerThread`) muuntaa tämän jäsennetyn informaation soivaksi esitykseksi. Seuraavissa luvuissa kuvataan näiden kahden säieluokan tehtävät ja toteutus.

7.2.3.1 Jäsenninsäie (`ParserThread`)

Luokan tehtävät: `ParserThread`-luokan tehtävä on lukea ja jäsentää annettu MusicXML-dokumentti ja kirjoittaa sen sisältämä musiikki-informaatio sovelluksen tietorakenteeseen.

Luokan toteutus: Käyttöliittymäluokka luo ja käynnistää `ParserThread`-luokan olion antaen sille tiedon jäsennettävän MusicXML-dokumentin sijainnista. Jäsennettävä dokumentti voi sijaita paikallisella levyllä tai tietoverkossa. Lisäksi käyttöliittymäluokka välittää jäsenninsäikeelle tiedon siitä, validoidaanko jäsennettävä MusicXML-dokumentti vai ei.

`ParserThread`-luokan tärkein jäsenmuuttuja on `m_SAXParser`, joka on SAX2-rajapinnan toteuttavan Xerces 2 Java XML-jäsentimen instanssi. `m_SAXParser` suorittaa MusicXML-dokumentin jäsentämisen käyttäen apunaan apuluokkaa

MusicXMLSAXHandler. Tämän luokan metodit ovat SAX-rajapinnan tapahtumankäsittelijämetodeita, joita kutsutaan XML-dokumenttia jäsennettäessä. MusicXML-dokumentin jäsentämisen edetessä MusicXMLSAXHandler-luokka luo jäsentämiensä elementtien ja niiden sisällön mukaisesti Measure- ja Note-luokkien olioita ja tallentaa ne sovelluksen tietorakenteeseen MeasureList-luokan avulla.

Lisäksi ParserThread-luokka ilmoittaa sovelluksen soittosäikeelle kun ensimmäinen tahti on valmis soitettavaksi. Ilmoittaminen tapahtuu kutsumalla tietorakenteen MeasureList-luokan notify-metodia, joka herättää sitä odottavan soittosäikeen jatkamaan suoritustaan [Flanagan, 1999, sivu 152].

7.2.3.2 Soittosäie (PlayerThread)

Luokan tehtävät: PlayerThread-luokan tehtävä on lukea ParserThread-luokan jäsentämä musiikki-informaatio sovelluksen tietorakenteesta ja muuntaa se auditiiviseksi esitykseksi.

Luokan toteutus: Käyttöliittymäluokka luo ja käynnistää PlayerThread-luokan olion antaen sille tiedon käyttäjän valitsemasta MIDI-laitteesta ja soitettavan kappaleen tempota.

PlayerThread-luokan tärkein jäsenmuuttuja tietorakenteen sisältämän musiikki-informaation soivaksi esitykseksi muuntamisessa on m_TimerPlayer. Tämä jäsenmuuttuja on tyypiltään java.util-pakkauksen Timer, jonka toimintaa esiteltiin luvussa 5.3.2. Apunaan se käyttää PlayerTimerTask-luokan oliota, joka on peritty java.util-pakkauksen TimerTask-luokasta. PlayerThread-luokka ajastaa PlayerTimerTask-luokan run-metodin suoritettavaksi toistuvasti säännöllisin väliajoin käyttäjän määrittelemän tempon mukaisesti. Ennen ajastimen käynnistämistä PlayerThread-luokka odottaa herätettä ensimmäisen tahdin valmistumisesta ParserThread-luokalta. Odottaminen tapahtuu kutsumalla MeasureList-luokan wait-metodia, joka odottaa kunnes jokin toinen olio (tässä tapauksessa siis ParserThread-luokan olio) kutsuu sen notify-metodia [Flanagan, 1999, sivu 152].

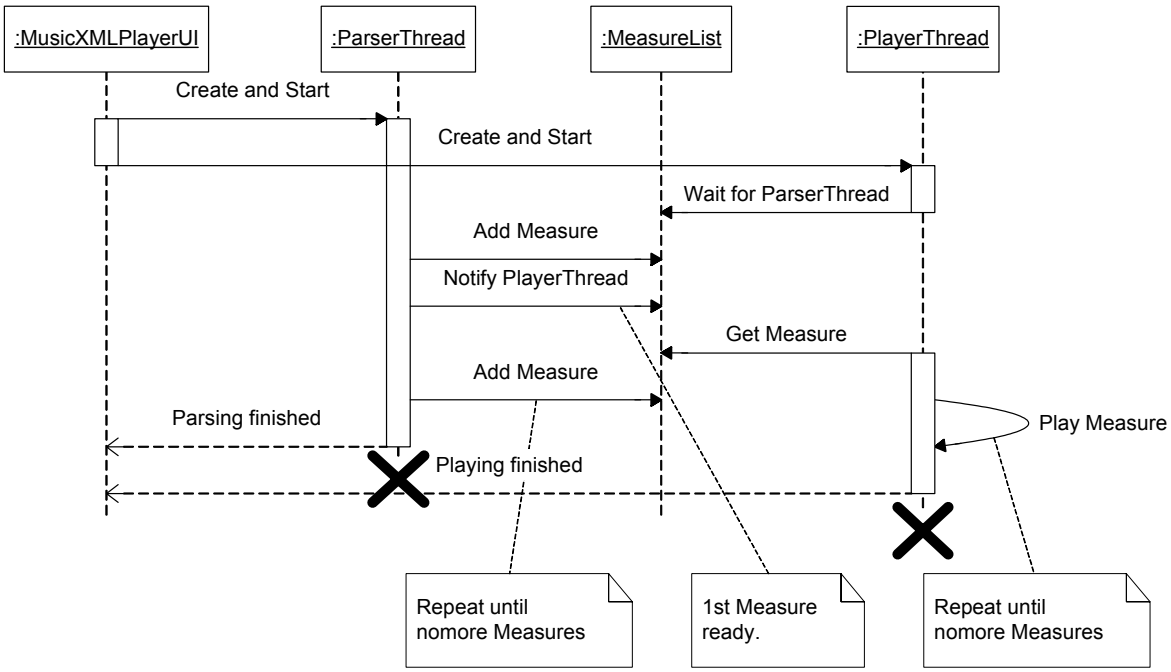
`PlayerTimerTask`-luokka toteuttaa varsinaisen musiikki-informaation auditiivisen esityksen muodostamisen eli korvin kuultavan musiikin. Se avaa käyttäjän valitseman MIDI-laitteen käyttäen Java Sound API:n tarjoamaa `MidiDevice`-tyyppistä jäsenmuuttujaa `m_MidiDevice` ja lukee sille lähetettävät MIDI-viestit eli `Note`-luokan oliot sovelluksen tietorakenteesta. Luokka huolehtii `Note`-luokan olioiden lähettämisestä oikealla ajanhetkellä `m_MidiDevice`-luokan vastaanottimelle (`Receiver`), joka edelleen huolehtii niiden soittamisesta MIDI-järjestelmän välityksellä käyttäjälle.

7.2.4 Luokkien välinen vuorovaikutus

Luvussa kuvataan MusicXML-soittimen luokkien välistä vuorovaikutusta UML-kaavioiden avulla. UML-kaavioista on esitetty tämän luvun kannalta oleelliset yleistetyt versiot ja yksityiskohtaiset versiot on esitetty liitteessä 3.

Kuvassa 14 on havainnollistettu MusicXML-dokumentin soivaksi esitykseksi muuntamisen toiminta yleistettynä sekvenssikaaviona (*Sequence Diagram*) [Booch, 1999, sivu 245]. Saman sekvenssikaavion yksityiskohtainen versio tarkkoine metodikutsuineen on esitetty liitteen 3 kuvassa L.2.

Kuvan sekvenssin käynnistyessä käyttäjä on painanut MusicXML-dokumentin soiton aloittavaa painiketta. Käyttöliittymäluokka (`MusicXMLPlayerUI`) luo ja käynnistää jäsennin- ja soittosäikeen (`ParserThread` ja `PlayerThread`) ja vapautuu tämän jälkeen palvelemaan jälleen käyttäjää. Jäsenninsäie aloittaa MusicXML-dokumentin jäsentämisen ja soittosäie jää odottamaan siltä ilmoitusta ensimmäisen tahdin valmistumisesta. Kun jäsenninsäie on lisännyt ensimmäisen tahdin tietorakenteeseen, se ilmoittaa tietorakenteen (`MeasureList`) avulla siitä soittosäikeelle ja jatkaa dokumentin jäsentämistä. Tämän jälkeen soittosäie aloittaa jäsennetyn musiikki-informaation auditiiviseksi esitykseksi muuntamisen.



Kuva 14. MusicXML-soittimen soiton käynnistämistoiminnon sekvenssikaavio.

Liitteen 3 kuvissa L.3-L.4 on kuvattu yhteistoimintakaavioiden (*Collaboration Diagram*) avulla tietorakenteen käyttöä sekä jäsenninsäikeen että soittosäikeen osalta [Booch, 1999, sivu 245].

Kuvassa L.3 esitetty jäsenninsäie (`ParserThread`) aloittaa MusicXML-dokumentin jäsentämisen ja kohdatessaan `<measure>`- ja `<note>`-elementtejä luo niitä vastaavia lokaaleja (*local*) eli paikallisia `Measure`- ja `Note`-luokan olioita. Se tallentaa luodut `Note`-luokan oliot oikeisiin `Measure`-luokan olioihin ja `Measure`-luokan oliot edelleen sovelluksen tietorakenteeseen globaalin (*global*) `MeasureList`-luokan esiintymän avulla. Ensimmäisen tahdin valmistuttua se ilmoittaa odottavalle soittosäikeelle ensimmäisen tahdin valmistumisesta ja jatkaa dokumentin jäsentämistä, kunnes koko dokumentti on jäsennetty ja tallennettu tietorakenteeseen.

Kuvan L.4 soittosäie (`PlayerThread`) odottaa käynnistymisensä jälkeen ilmoitusta ensimmäisen tahdin valmistumisesta jäsenninsäikeeltä tietorakenteen globaalin `MeasureList`-luokan esiintymän avulla. Ilmoituksen saatuaan se ajastaa `PlayerTimerTask`-luokan olion aloittamaan tietorakenteessa olevan musiikki-infor-

maation muuntamisen auditiiviseksi esitykseksi. `PlayerTimerTask`-luokan olio lukee sovelluksen tietorakennetta (`MeasureList`) käyttäen apunaan lokaaleja `Measure`- ja `Note`-luokkien olioita. Jokaisella suorituskerralla `PlayerTimerTask`-luokan olio lähettää kyseisellä ajanhetkellä soitettavaksi tarkoitetut `Note`-luokan oliot MIDI-laitteen (`MidiDevice`) soitettavaksi.

8 Sovelluksen ja tekniikoiden arviointia

Luvussa arvioidaan tutkielmassa toteutetun sovelluksen lopputulosta ja sen kehitystyössä käytettyjä tekniikoita ja työkaluja. Luvussa 8.1 arvioidaan MIDI-standardia musiikin kuvauskielenä ja sen soveltuvuutta reaaliaikaisen soivan esityksen muodostamiseen. Luvussa 8.2 arvioidaan MusicXML-määrittelykielen soveltuvuutta yleiskäyttöiseksi musiikin kuvauskieleksi erityisesti reaaliaikaisesti tapahtuvan auditiivisen esityksen tuottamisen näkökulmasta. Lopuksi luvussa 8.3 arvioidaan tutkielmassa toteutettua sovellusta ja sen kehitystyössä käytettyjä työkaluja sekä niiden soveltuvuutta tähän tarkoitukseen.

8.1 MIDI:n arviointia

Luvussa arvioidaan MIDI-standardin soveltuvuutta ja käyttökelpoisuutta musiikin kuvauskielenä. Luvussa 8.1.1 arvioidaan MIDI:ä yleisesti ja luvussa 8.1.2 sen toimivuutta ja soveltuvuutta tutkielmassa toteutetun sovelluksen näkökulmasta.

8.1.1 Yleisesti

MIDI täyttää sille sen suunnitteluvaiheessa asetetut vaatimukset yleisesti ottaen erinomaisesti. Huomioon ottaen sen alkuperäisen käyttötarkoituksen elektronisten soitinten välisenä yhteisenä kielenä, sen saavuttama vankka suosio ja yleistyminen osoittavat suunnittelijoiden onnistuneen työssään. MIDI on kestänyt jo kahden vuosikymmenen ajan teknologian kehityksen tuomat paineet ja on edelleen käytännössä ainoa keino elektronisten instrumenttien väliseen kommunikointiin. Luvussa 3.3 käsitellyt rajoitteet ovat toki edelleen MIDI:n rasitteena ja ne aiheuttavat myös ongelmia sen käytössä reaaliaikaisen auditiivisen esityksen muodostamisessa.

Ehkä merkittävin ongelma MIDI:n käytössä on sen aikariippuvuus. Esimerkiksi ääniviestien tapauksessa tämä tarkoittaa sitä, että *ääni päälle* -viestin mukana ei lähetetä mitään tietoa kyseessä olevan sävelen soivasta kestosta. Täten jokaisen soitetun sävelen sammuttamiseksi pitää lähettää erillinen *ääni pois* -viesti, jolloin vastuu sävelen kestosta huolehtimisesta jää kokonaisuudessaan ääniviestejä lähettävälle MIDI-laitteelle. Ratkaisu on peräisin MIDI:n alkuperäisestä käyttötarkoituksesta ja esimerkiksi MIDI-koskettimistot gene-

roivat ääni päälle -viestin kosketinta painettaessa ja ääni pois -viestin koskettimen painamisen loppuessa. Tämä logiikka aiheuttaa kuitenkin tarpeetonta monimutkaisuutta lähettävän MIDI-laitteen toteutuksessa, sillä jokaisen sävelen sammuttaminen vaatii jokaisen sävelen keston määrittämistä ja ääni pois -viestien oikea-aikaista lähettämistä vastaanottavalle MIDI-laitteelle. Toisaalta vastaanottava MIDI-laite voidaan toteuttaa yksinkertaisemmin, sillä sen tarvitsee vain käsitellä ja toteuttaa saamansa MIDI-viestit saapumisjärjestyksessä.

Kanavien liian vähäinen lukumäärä aiheuttaa väistämättä ongelmia missä tahansa käyttöympäristössä. MIDI:n 16 kanavaa on täysin riittämätön määrä nykyaikaisissa järjestelmissä ja useat valmistajat ovatkin puuttuneet tähän omissa MIDI-toteutuksissaan ja -laajennuksissaan. Valitettavasti General MIDI 1.0 -määrityksen 128 instrumenttia loppuvat nekin kesken ja eri sointivärien toteuttaminen eri instrumenttinumeroilla on silkkaa typeryyttä. Eräs ratkaisu tähän voisi olla sointivärien toteutus erilaisilla valittuna olevan instrumentin efekteillä tai asetuksilla kuten monissa syntetisaattoreissa onkin tehty. General MIDI:n puutteisiin on onneksi tiedossa parannusta General MIDI 2.0 -määrityksen myötä, mutta sen vakiintuminen markkinoilla kestänee vielä vuosia.

Lisäksi MIDI:n sarjamuotoinen liikenne aiheuttaa hitautta tiedonsiirrossa ja siitä aiheutuva viive MIDI-viestien välityksessä on muodostunut ongelmaksi monissa käyttöympäristöissä. Erilaiset synkronoinnit ja MIDI-kellon käyttäminen ovat kuitenkin auttaneet tähän ongelmaan.

8.1.2 Toteutetun sovelluksen kannalta

Tutkielmassa toteutetun sovelluksen kannalta MIDI osoittautui oivaksi apuvälineeksi soivan esityksen tuottamisessa. MusicXML-dokumenttien sisältämä musiikki-informaatio on muunnettavissa suhteellisen suoraviivaisesti MIDI-viesteiksi. Tämä rajoittuu kuitenkin käytännössä sävelten sävelkorkeuksien ja voimakkuuksien määrittämiseen. Edellisessä luvussa käsitelty aikariippuvaisuus aiheutti monimutkaisuutta sovelluksen ohjelmointivaiheessa, sillä jokaisen sävelen kesto pitää erikseen laskea ja jokaista soitettua säveltä kohden täytyy muodostaa erillinen ääni pois -viesti. Erityisiä hankaluuksia tämä aiheuttaa

tahtiviivan ylittävien sävelten ääni pois -viestien tuottamisessa ja lähettämisessä. Jos sävelen kesto olisi suoraan määritelty jo ääni päälle -viestissä, tältä monimutkaisuudelta olisi vältytty.

General MIDI 1.0 -määrityksen puutteet aiheuttivat ongelmia myös MusicXML-soittimen kehitystyössä. Esimerkiksi jousisoitinten sointivärin muutoksien merkitsemiseen käytetyt merkinnät pitää tulkita MIDI-viesteiksi muunnettaessa instrumentinvaihdoksina, joka on ristiriidassa todellisuuden kanssa.

MusicXML-soitin saatiin MIDI:n avulla kuitenkin toteutettua sen puutteista huolimatta. Vaihtoehtoinen tapa toteuttaa auditiivinen esitys olisi generoida audiota itse suoraan ohjelmakoodista, mutta menettelyn kohtuuton työmäärä ja vaativuus eliminoivat tämän vaihtoehdon harkitsemisen. Lisäksi syntetisaattorit ja edelleen MIDI ovat alun perin kehitetty juuri tähän tarkoitukseen, eikä pyörän keksiminen uudelleen ole koskaan houkutteleva vaihtoehto.

8.2 MusicXML-määrittelykielen arviointia

Luvussa arvioidaan tutkielmassa tutkitun MusicXML-määrittelykielen soveltuvuutta yleiskäyttöiseksi musiikin kuvauskieleksi. Luvussa 8.2.1 arvioidaan sitä yleisellä tasolla käyttötarkoituksiinsa nähden ja luvussa 8.2.2 sen ominaisuuksia tässä tutkielmassa toteutetun sovelluksen näkökulmasta.

8.2.1 Yleisesti

MusicXML-määrittelykieltä kehittävä yritys on lähtenyt kunnianhimoiseen yritykseen kehittää yleiskäyttöinen kuvauskieli musiikki-informaation kuvaamiseen. Edellisistä vastaavanlaisista yrityksistä (*MuseData*, *HumDrum*, *NIFF* ja *SMDL*, katso luku 2.5) oppineena *Recordare LLC* on määrittelytyötä aloittaessaan tehnyt tietoisen valinnan keskittyä pääosin länsimaisessa taide- ja populaarimusiikissa käytetyssä viivastonotaatiossa esiintyvien elementtien ja ominaisuuksien graafisen ulkoasun kuvaamiseen. Tämä päätös rajoittaa kuitenkin turhaan kehitetyn kuvauskielen mahdollisuuksia yleistyä yleiskäyttöiseksi musiikin kuvauskieleksi. Nämä rajoitukset muodostuvat ongelmiksi erityisesti musiikki-infor-

maatiota reaaliaikaisesti käsittelevissä sovelluksissa. Tällaisia sovelluksia ovat esimerkiksi musiikin auditiivista esitystä käsittelevät sovellukset, joihin myös tässä tutkielmassa kehitetty MusicXML-soitin lukeutuu.

Rakenteisten dokumenttien soveltuvuus musiikin kuvaamiseen: Musiikin kuvaaminen rakenteiseen muotoon XML:n avulla näyttää tämän tutkimuksen perusteella onnistuvan hyvin. Musiikki-informaation sisäisten hierarkioiden kuvaaminen rakenteisiin dokumentteihin onnistuu kohtalaisen suoraviivaisesti, eikä suurempia ongelmia ole. Myös musiikki-informaation sisäisen hierarkian rikkovat poikkeukset saadaan kuvattua XML:n keinoin elementtien attribuutteina. XML:ään perustuvan määrittelykielen suunnittelussa valinta elementtien ja attribuuttien käytön välillä on käytännössä vain valintakysymys, eikä valinta puoleen tai toiseen vaikuta välttämättä kielen käyttökelpoisuuteen. Tärkeintä valinnassa kuitenkin on säilyttää johdonmukaisuus, eikä sen vaikutus esimerkiksi dokumentin jäsennysnopeuteen ole merkittävä.

Musiikin eri olomuotojen kuvaaminen: MusicXML-määritykseen on yritetty liiaksi sisällyttää luvussa 2.1 käsiteltyjä musiikin eri olomuotoja. Alkuperäistä periaatetta musiikillisen sisällön kuvaamisesta elementteinä ja graafisen viivastonotaation ja auditiivisen esityksen ominaisuuksien kuvaamisesta elementtien attribuuteilla ei ole pystytty käytännössä noudattamaan tai se on suunnittelun edetessä kokonaan unohtunut. Nyt sekä elementtejä että attribuutteja käytetään kuvaamaan kaikkia kolmea olomuotoa, ja erittely näiden välillä on vaikeaa.

Erityisesti auditiivisen esityksen piirteitä yritetään kuvata sisällyttämällä MIDI-informaatiota MusicXML-määrittelykieleen. Esimerkkeinä tästä on MIDI-instrumenttinumeroiden sisällyttäminen instrumenttien määrittelyyn MusicXML-dokumentin otsikkotiedoissa (katso esimerkki 3) ja MIDI-informaation sisällyttäminen esitysvoimakkuuksien määrittelyyn (katso esimerkki 4). Samat asiat määritellään kuitenkin myös viivastonotaation näkökulmasta, joten MIDI-informaation lisääminen on turhaa. Kuitenkaan MIDI:iin liittyvät elementit ja attribuutit eivät ole pakollisia, joten MusicXML:ää käsittelevä sovellus ei voi täysin luottaa niiden olemassaoloon. Tällainen menettely heikentää MusicXML-määrittelykielen itsenäisyyttä kuvauskielenä, sillä dokumentin tekijän pitää tässä tapauksessa tietää

myös MIDI:stä. Olisi parempi säilyttää kuvauskielen itsenäisyys jättämällä viittaukset muihin määrittelykieliin MusicXML:stä pois.

Nimiavaruuksien käyttö voisi olla eräs ratkaisu musiikin eri olomuotojen erottelemiseen MusicXML-dokumenteissa. Näin menetellen auditiiviseen esitykseen liittyvät elementit voisivat sijaita omassa nimiavaruudessaan ja graafiseen ulkoasuun liittyvät elementit ja attribuutit omassaan. Näin kukin sovellus voisi jo nimiavaruuden perusteella erotella itselleen oleelliset asiat MusicXML-dokumenteista ja jättää tarpeettomat asiat kokonaan huomioimatta.

Toinen ratkaisu samaan ongelmaan voisi olla hierarkia-ajattelun ulottaminen myös DTD-tyyppimääritysdokumenttien jaottelun suunnitteluun. Nykyinen toteutus (katso luku 4.5.1, kuva 8) yrittää mahdollistaa kaikkien eri olomuotojen kuvaamisen yhdellä ja samalla tyyppimäärityksellä. Jakamalla MusicXML:n tyyppimääritysdokumentit musiikin eri olomuotojen mukaisesti hierarkkiseksi rakenteeksi saataisiin paremmin eroteltua eri käyttötarkoituksiin sopivat tyyppimääritykset omiksi osakokonaisuuksiksi.

MusicXML-dokumenttien tiedostokoko: Eräs ongelma MusicXML-määrittelykielen käytössä tietoverkoissa on dokumenttien suuri fyysinen koko. MusicXML-dokumentit ovat tekstitiedostoja ja usein väljästi kirjoitettuja, joten niiden koko kasvaa helposti todella suureksi. Verrattuna esimerkiksi nuotinkirjoitusohjelmisto *Finale*:n binäärisiin tiedostoihin MusicXML-dokumenttien koot ovat noin kolminkertaisia.

XML-dokumenttien ollessa tekstitiedostoja ne kuitenkin pakkautuvat murto-osaan alkuperäisestä koostaan ja tällä MusicXML-määrittelykieltä kehittävä yritys onkin vähätellyt suuren tiedostokoon aiheuttamia mahdollisia ongelmia [Good, 2001b, sivu 118]. Jatkossa voisikin tutkia esimerkiksi `java.util.zip`-pakkauksesta löytyvien luokkien käyttöä pakattujen MusicXML-dokumenttien purkamisessa reaaliajassa. Pakkaaminen pienentäisi huomattavasti MusicXML-dokumenttien fyysistä kokoa ja täten lyhentäisi myös esimerkiksi Internetistä ladattaessa dokumentin siirtämiseen kuluvaa aikaa.

DTD-määrityksen käytöstä aiheutuvat ongelmat: Myös DTD-määrityksen käyttö MusicXML:n tyyppimäärityksiin on lähtökohdiltaan ongelmallinen. DTD-määrityksen

avulla ei ole mahdollisuutta elementtien tai attribuuttien sisältämän datan tyypitykseen eikä sallittujen arvojen rajoittamiseen. Kuvattava musiikki-informaatio sisältää paljon tekstimuotoista informaatiota, jonka sallittujen arvojen rajoittaminen olisi ehdottoman tärkeää. Esimerkiksi nuotin soivan sävelkorkeuden määrittely tehdään merkkimuotoisena (katso esimerkki 4) ja MusicXML-dokumentin kirjoittaja saa DTD:n avulla määriteltynä kirjoittaa elementin sisällöksi mitä tahansa. Tällöin vastuu datan oikeellisuuden ja järkevyyden tarkistamisesta on kokonaan MusicXML-dokumenttia käsittelevällä sovelluksella.

Ratkaisu näihin ongelmiin voisi olla XML Schema Language -määrityksen käyttö MusicXML:n tyyppimäärityksiin. XML Schema Language -määrityksen avulla sallitut arvot voitaisiin määritellä jo dokumentin tyyppimäärityksessä ja MusicXML-dokumenttia käsittelevä sovellus voisi olla varma elementin sisältämän datan järkevyydestä ja oikeellisuudesta.

Luvussa 4.1.2 käsiteltyihin käyttötarkoituksiinsa MusicXML-määrittelykieli sopii yleisesti ottaen kuitenkin erinomaisesti. XML-standardien mukaisesti määriteltynä se on jo lähtökohdiltaan hyvissä asemissa sovelluskehittäjien ja Internet-maailman silmissä. Valmiit XML-tekniikat ja -työkalut toimivat MusicXML:n kanssa sellaisinaan ja sovelluskehittäjien on siten helppo alkaa kehittämään uusia MusicXML-määrittelykieltä käyttäviä sovelluksia. Tätä kirjoitettaessa (9.10.2003) myös tiedonsiirto eri musiikkiohjelmistojen välillä toimii jo hyvin. Yleiskäyttöisten XML-työkalujen kehittyessä myös MusicXML-dokumenttien sisältämän musiikki-informaation analysointi onnistunee niiden avulla suoraviivaisemmin ja helpommin kuin ennen.

Tuleva standardointi ja laaja tuki musiikkiohjelmistojen valmistajien keskuudessa takaavat MusicXML-määrittelykielen yleistymiselle hyvät lähtökohdat. Tulevaisuus näyttää MusicXML:n menestymisen suhteen hyvältä ja MusicXML tullee täyttämään sille asetetut toiveet ja tavoitteet yleiskäyttöisenä musiikin kuvauskielenä.

8.2.2 Toteutetun sovelluksen kannalta

Tutkielmassa toteutetun sovelluksen kannalta MusicXML-määrittelykieli täyttää tehtävänsä kohtalaisesti. MusicXML-dokumenttien sisältämän musiikki-informaation soivaksi

esitykseksi muuntaminen onnistuu melko suoraviivaisesti, mutta joitain kompromisseja joudutaan kuitenkin tekemään. Tutkielmassa toteutettua sovellusta arvioidaan tarkemmin luvussa 8.3.

Musiikki-informaation kaksi ulottuvuutta: Toteutetun MusicXML-soittimen ainoana todellisena rajoitteena voitaneen pitää sitä, että soivaksi esitykseksi muunnettavan kappaleen musiikki-informaation tulee olla ajallisesti jaettua eli MusicXML-dokumentin juurielementin tulee olla `<score-timewise>`-elementti. Tämä tosin tiedostettiin ja siten asetettiin sovelluksen rajoitteeksi jo vaatimusmäärittelyvaiheessa, joten ongelmaksi tätä ei voida luonnehtia.

MusicXML-määrityksen tyyppimääritysdokumenttien mukana toimitetaan XSL-muunnosdokumentit näiden kahden esitystavan väliseen muunnokseen, joten muunnos olisi helppo toteuttaa MusicXML-soittimeenkin. Tämä kuitenkin pakottaisi koko MusicXML-dokumentin läpikäymisen ja poistaisi mahdollisuuden soiton aloittamiseen heti jäsentämisen alettua. Käyttäjälle mahdollisuus muunnokseen voitaisiin kuitenkin tarjota, joten ominaisuus kirjattiin sovelluksen jatkokehitysideoihin.

MusicXML:n keskittyminen viivastonotaation kuvaamiseen: Tutkielmassa kehitetyn MusicXML-soittimen kehitystyössä kohdattiin muutamia vaikeuksia johtuen MusicXML-määrittelykielen liiallisesta keskittymisestä viivastonotaation graafisen ulkoasun ominaisuuksien kuvaamiseen.

Hankaluuksia aiheuttaa sävelten keston määrittelemiseen käytetyn `<divisions>`-elementin esiintyminen tahtikohtaisena ominaisuutena `<attributes>`-elementin lapsielementtinä. Elementin sijainnista aiheutuu turhaa monimutkaisuutta ja lisälaskentaa reaaliaikaisesti musiikki-informaatiota käsittelevissä sovelluksissa, kuten tässä tutkielmassa toteutetussa MusicXML-soittimessa, sillä seuraavassa osassa muuttuva jakajan arvo pakottaa muuttamaan kaikkien jo käsiteltyjen osien vastaavat arvot jälkeenpäin. Elementin arvo asetetaan kuitenkin vain kerran osaa tai instrumenttia kohden, joten huomattavasti luonnollisempi sijoituspaikka sille olisi `<score-part>`-elementissä jo osan muiden ominaisuuksien määrittelyn yhteydessä MusicXML-dokumentin otsikkotiedoissa.

Toinen ongelmia aiheuttava ominaisuus on <sound>-elementin epämääräinen määrittely. Se voi sijaita kolmen eri elementin lapsielementtinä ja täten sen vaikutusalueen päätteleminen on tehty vaikeaksi. Toisaalta viivastonotaation elementtien kontekstiriippuvuus ja tulkinnanvaraisuus aiheuttavat jo itsessään ongelmia, joten pelkästään MusicXML-määrittelykieltä ei voida tästä syyttää.

8.3 Sovelluksen ja käytettyjen työkalujen arviointia

Tutkielmassa toteutetun MusicXML-soittimen kehitystyö onnistui kokonaisuudessaan hyvin ja kaikki sille luvussa 6 asetetut toiminnalliset ja ei-toiminnalliset vaatimukset täytettiin. Sovellus toimii kuten sen tuleekin toimia ja se palvelee käyttötarkoitustaan kiitettävästi. Joitain ongelmia ja vaikeuksia sovelluksen kehitystyössä kuitenkin kohdattiin ja niitä kuvataan seuraavissa luvuissa.

8.3.1 MusicXML-dokumentin jäsennys

MusicXML-dokumentin jäsennys ja muunnos auditiiviseksi esitykseksi onnistui SAX-rajapinnan toteuttavalla XML-jäsentimellä yleisesti ottaen hyvin, eikä suurempia ongelmia sovelluksen kehitystyössä kohdattu.

Absoluuttinen vs. suhteellinen informaatio: Jo tutkielman alussa (katso luku 2.3) käsitelty viivastonotaation elementtien merkityksen kontekstiriippuvuus pakotti luopumaan pelkästään tapahtumapohjaisesta auditiivisen esityksen muodostamisesta. Lisäksi MusicXML-määrittelykielessä ajallisestikin jaettuna kukin osa tai soitin määritellään tahti kerrallaan, joten ainakin yhden tahdin sisältämä musiikki-informaatio on jäsennettävä kokonaan ennen soivan esityksen soittamisen aloittamista. Tämä ei kuitenkaan muodostunut ongelmaksi, sillä sovelluksen tietorakenne suunniteltiin tämän mukaisesti.

MusicXML-dokumentin jäsentämisen nopeus: Yhtenä sovelluksen vaatimuksena oli soivan esityksen soiton aloittaminen heti kun MusicXML-dokumentin ensimmäinen tahti on saatu jäsennettyä. Tämä ominaisuus toteutettiin, mutta käytetty XML-jäsennin (luvussa 5.1.2 esitelty *Xerces Java 2*) osoittautui niin nopeaksi, että ominaisuudesta saavutetut hyödyt jäivät pieniksi. Tämä tosin ilmenee käytettäessä sovellusta nopealla verkkoyhteydellä

varustetulla tietokoneella ja esimerkiksi hitaan modeemin välityksellä MusicXML-dokumenttia siirrettäessä ominaisuus pääsee paremmin oikeuksiinsa.

Yllätyksellisesti myöskään MusicXML-dokumentin validointi ei vaikuttanut mitenkään jäsentämisen nopeuteen vaan dokumentin sisältämä musiikki-informaatio saatiin tallennettua sovelluksen tietorakenteeseen lähes yhtä nopeasti kuin ilman validointia. Tosin XML-jäsennin hakee kaikki DTD-dokumentit tietoverkosta ennen jäsentämisen aloittamista, joten jäsennyksen alkamiseen kuluva aika kasvaa hieman.

8.3.2 Auditiivisen esityksen muodostaminen

Auditiivisen esityksen muodostaminen Java Sound API:n avulla onnistui todella hyvin. Mitään suuria ongelmia ei ohjelmointirajapinnan kanssa kohdattu, sillä suuritöisimmäksi osuudeksi soivan esityksen muodostamisessa osoittautui säikeiden, ajastimien ja sovelluksen tietorakenteen optimointi.

Java Sound API: MusicXML-dokumentin sisältämän musiikki-informaation varsinainen auditiiviseksi esitykseksi muuntaminen toteutettiin Java Sound API:n tarjoamin keinoin. Koko ohjelmointirajapinta osoittautui esimerkillisen hyvin toteutetuksi, eikä suurempia vaikeuksia alun opiskelun jälkeen ollut. Kaikki kohdatut vaikeudet johtuivat suurimmaksi osaksi MIDI:n puutteista, joille Java Sound API ei voi käytännössä mitään. Jatkossa voisi tutkia rajapinnan `sequencer`-luokan mahdollisuuksia soivan esityksen muodostamisessa. Tämä mahdollistaisi esimerkiksi muodostetun auditiivisen esityksen tallentamisen MIDI-tiedostoksi.

Säikeet, ajastimet ja sovelluksen tietorakenne: MusicXML-dokumentin sisältämän musiikki-informaation muuntamisessa soivaksi esitykseksi haasteellisimmaksi tehtäväksi muodostui musiikkikappaleen soiton tahdistaminen ja rytmin toteuttaminen. Java-ohjelmointikielen ajastimet soveltuivat lopulta tehtävään erinomaisesti, tosin niiden saaminen pysymään tahdissa aiheutti jonkin verran ongelmia. Rytmiikan toteutuksessa sävelten oikea-aikainen soitto on äärimmäisen tärkeää, sillä korva huomaa hyvin herkästi pienenkin hidastelun. Suurimmat virheet ja hidastelut rytmissä saatiin korjattua säikeiden oikealla priorisoinnilla, mutta pientä viivyttelyä soitossa oli havaittavissa silti.

Ongelmia aiheutti musiikkikappaleen sisältämän informaation sisäisen hierarkian rikkovat poikkeukset. Näiden tuomien ongelmien ratkaisemiseksi sovelluksen tietorakenne muodostui alussa liian monimutkaiseksi. Esimerkiksi tahtiviivojen ylittävät sävelten kestot lisäsivät tarvetta käydä tietorakennetta läpi useaan kertaan. Tämä lisäsi vastaavasti suoritusaikaa, joka ilmeni varsinainen auditiivisen esityksen soittamisen hidasteluna.

Oikeaksi ratkaisuksi osoittautui vastuun siirtäminen tietorakenteen synkronoinnista kokonaan tietorakenteen ylimmälle `MeasureList`-luokalle. Tällöin kaikki kriittinen kommunikointi sovelluksen tietorakenteen kanssa tapahtuu tämän luokan välityksellä eikä muiden luokkien (`Measure`- ja `Note`-luokkien) tarvitse huolehtia synkronoinnista lainkaan. Näin menetellen säikeiden odotusajat poistuivat käytännössä kokonaan ja auditiivisen esityksen muodostaminen saatiin rytmisesti tarkemmaksi.

Lisäksi ajastimien toteutusta optimoitiin siten, että jokaisella suorituskerralla ajastimen aivan ensimmäinen tehtävä on sillä ajanhetkellä soitettavaksi tarkoitettujen sävelten soitto. Tämän toteuttamiseksi ajastin hakee jokaisella suorituskerralla sävelten soittamisen jälkeen seuraavalla suorituskerralla soitettaviksi ja sammutettaviksi tarkoitetut sävelet jo valmiiksi sovelluksen tietorakenteesta saataville. Tämäkin ratkaisu tarkensi sävelten oikea-aikaista soittamista ja sammuttamista parantaen auditiivisen esityksen rytmistä kuulokuvaa.

9 Yhteenveto

Onko MusicXML-määrittelykielestä yleiskäyttöiseksi musiikin kuvauskieleksi? Vai onko monimutkaisen musiikki-informaation kuvaaminen tietokoneita ja -järjestelmiä varten jo alun alkaen mahdoton tehtävä? Tämän tutkielman tulosten nojalla kaikki edellytykset MusicXML:n menestymiselle ovat olemassa. Tuleva standardointi ja laaja tuki musiikkiohjelmistojen valmistajien keskuudessa takaavat MusicXML:lle hyvät mahdollisuudet yleistyä ja vakiintua alalla yleisesti käytettäväksi kuvauskieleksi.

Omasta mielestäni näkisin tällaisen yleiskäyttöisen musiikin kuvauskielen vakiintumisen erilaisten musiikkiohjelmistojen yhteisenä tekijänä erittäin tervetulleena asiana. Se helpottaisi montaa käytännön ongelmaa ja edesauttaisi myös musiikin tutkimusta ja analysointia huomattavasti. MusicXML-määrittelykielen kehitystyön liiallinen keskittyminen graafisen viivastonotaation elementtien kuvaamiseen rajoittaa hieman kuvauskielen käyttökelpoisuutta reaaliaikaisesti musiikki-informaatiota käsittelevissä sovelluksissa, mutta nämäkin ongelmat poistunevat määrityksen valmistuessa.

MusicXML-määrittelykieli on vielä tätä kirjoitettaessa (9.10.2003) julkisessa beetatestauksessa ja tässä tutkielmassa ilmenneistä kuvauskielen ongelmista on tarkoitus raportoida MusicXML-määrittelykieltä kehittävälle *Recordare LLC* -yritykselle. Täten myös tässä tutkielmassa kehitetty sovellus (ja sen tekijä) on osaltaan osallistunut MusicXML-määrittelykielen luvussa 4.1.1 mainitun inkrementaalisen prosessimallin mukaiseen ohjelmistokehitykseen.

Tutkielma alkoi aiheen teoriataustan kuvauksella, jossa käsiteltiin länsimaisen taide- ja populaarimusiikin käyttämää viivastonotaatiota ja sen elementtejä sekä musiikki-informaation monimutkaisuutta ja sisäisiä rakenteita. Seuraavaksi esiteltiin elektronisten soitinten yhteenliittämiseen tarkoitettua MIDI-standardia ja musiikki-informaation kuvaamiseen rakenteisiin dokumentteihin tarkoitettua MusicXML-määrittelykieltä. Tämän jälkeen esiteltiin tutkielmassa kehitetyn sovelluksen kehitystyössä käytettyjä tekniikoita ja työkaluja.

Tutkielman empiirinen osuus koostui tutkielmassa toteutetun MusicXML-soittimen määrittelystä ja toteutuksesta. Kehitetyn sovelluksen arkkitehtuuria ja toimintaa kuvattiin UML-kaavioiden ja luokkakuvausten avulla. Lopuksi arvioitiin MusicXML-määrittelykielen soveltuvuutta yleiskäyttöiseksi musiikin kuvauskieleksi sekä MIDI:n käytettävyyttä auditiivisen esityksen muodostamiseen MusicXML-dokumenteista. Lisäksi arvioitiin toteutettua sovellusta ja sen kehitystyössä käytettyjen tekniikoiden ja työkalujen soveltuvuutta tähän tarkoitukseen.

Tutkielman teoriaosuuden kirjallisuuskatsaus ja empiirisessä osuudessa kehitetyn sovelluksen toteutus opettivat tutkielman tekijälle paljon musiikki-informaation monimutkaisuuden aiheuttamista ongelmista sen kuvaamisessa ja käsittelemisessä tietotekniikan tarjoamin keinoin. Lisäksi ne antoivat myös paljon kokemusta XML-tekniikoista ja -työkaluista sekä rakenteisten dokumenttien käytöstä saatavista hyödyistä ja eduista. Laajahkon tutkimustyön tekeminen myös opetti pitkäjänteisyyttä ja määrätietoisuutta, sekä kykyä kriittiseen ajatteluun.

Lähteet

Anthony Don, Cronin Charles, Selfridge-Field Eleanor, *The Electronic Dissemination of Notated Music: An Overview*, kirjassa ”The Virtual Score, Representation, Retrieval, Restoration”, MIT Press, Cambridge, Massachusetts, USA, s. 135-166, 2001.

Apajalahti Hannu, *Intervalli*, kirjassa ”Suuri musiikkitietosanakirja, 3. osa”, Otava, Keuruu, s. 84-85, 1990.

Bellini Pierfrancesco, Nesi Paolo, *WEDELMUSIC Format: an XML Music Notation Format for Emerging Applications*, kirjassa "Proceedings of the First International Conference on WEB Delivering of Music", IEEE, sivut 84-91, 2001.

Booch Grady, Rumbaugh James, Jacobson Ivar, "The Unified Modeling Language User Guide", Addison-Wesley, Reading, Massachusetts, USA, 1999.

Byrd Donald, *Music-notation searching and digital libraries*, kirjassa "Proceedings of the first ACM/IEEE-CS joint conference on Digital libraries", International Conference on Digital Libraries, sivut 239-246, 2001.

Castan Gerd, Good Michael, Roland Perry, *Extensible Markup Language (XML) for Music Applications: An Introduction*, kirjassa ”The Virtual Score: Representation, Retrieval, Restoration”, MIT Press, Cambridge, Massachusetts, USA, s. 95-102, 2001a.

Doan T. T., *Understanding MIDI, The key to creating and conductiong the music to your own MTV video*, Potentials, IEEE, Volume 13, Issue 1, 1994, sivut 10-11.

Flanagan David, ”Java in a Nutshell, Third Edition”, O’Reilly & Associates, Inc., Sebastopol, California, USA, 1999.

Good Michael, ”MusicXML in Practice: Issues in Translation and Analysis”, saatavilla WWW-muodossa osoitteessa <URL: `http://www.recordare.com/good/max2002.html`>, 19.9.2002.

Good Michael, ”MusicXML: An Internet-Friendly Format for Sheet Music”, saatavilla WWW-muodossa osoitteessa <URL: http://www.idealliance.org/papers/xml2001/papers/html/03-04-05.html>, 9.12.2001a.

Good Michael, *MusicXML for Notation and Analysis*, kirjassa ”The Virtual Score: Representation, Retrieval, Restoration”, MIT Press, Cambridge, Massachusetts, USA, s. 113-124, 2001b.

Grande Cindy, *The Notation Interchange File Format: A Windows-Compliant Approach*, kirjassa ”Beyond MIDI: The Handbook of Musical Codes”, MIT Press, Cambridge, Massachusetts, USA, s. 491-512, 1997.

Harold, Elliotte Rusty, ”XML - Tehokäyttäjän opas”, Satku, Helsinki, 1999.

Hewlett Walter B., *MuseData: Multipurpose Representation*, kirjassa ”Beyond MIDI: The Handbook of Musical Codes”, MIT Press, Cambridge, Massachusetts, USA, s. 402-447, 1997a.

Hewlett Walter B., Selfridge-Field Eleanor, *MIDI*, kirjassa ”Beyond MIDI: The Handbook of Musical Codes”, MIT Press, Cambridge, Massachusetts, USA, s. 41-72, 1997b.

Hirvi Jussi & Tuominen Antti Juhani, "Uusi MIDI-kirja", Painatuskeskus Oy, Helsinki, 1995.

Holzner Steven, ”Inside XML”, IT Press, Jyväskylä, 2001.

Huron David, *HumDrum and Kern: Selective Feature Encoding*, kirjassa ”MIDI: The Handbook of Musical Codes”, MIT Press, Cambridge, Massachusetts, USA, s. 375-401, 1997.

McLaughlin Brett, ”Java and XML”, O’Reilly & Associates Inc., Sebastopol, California, USA, 2000.

Megginson David, "SAX: Simple API for XML, API Documentation", saatavilla WWW-muodossa osoitteessa <URL: http://www.saxproject.org/apidoc/>, 25.5.2002.

MIDI Manufacturers Association, "General MIDI 2 Specification", saatavilla WWW-muodossa osoitteessa <URL: http://www.midi.org/about-midi/gm/gm2_spec.shtml>, viitattu 5.10.2003

Newcomb Steven R., *Standard Music Description Language complies with hypermedia standard*, Computer, Volume 24, Issue 7, 1991, sivut 76-79.

Nykänen Ossi, ”XML”, Docendo, Jyväskylä, 2001.

Oksala Yrjö, ”Musiikin perusteet, I osa: Nuottikirjoitus”, Fazer, Helsinki, 1971.

Penfold, R. A., ”Practical MIDI Handbook, Second Edition”, PC Publishing, Tonbridge, Englanti, 1990.

Recordare LLC, ”MusicXML 0.7 Tutorial”, saatavilla WWW-muodossa osoitteessa <URL: http://www.musicxml.org/xml/tutorial.html>, viitattu 28.6.2003.

Romanowski Otto, ”MIDI 1.0, Musiikkilaitteiden tiedonsiirtostandardi”, Valtion painatuskeskus, Helsinki, 1990.

Roy Jaideep, Ramanujan Anupama, *XML Schema Language: Takin XML to the Next Level*, IT Professional, Volume 3, Issue 2, 2001, sivut 37-40.

Rumbaugh James, Blaha Michael, Premerlani William, Eddy Frederick and Lorensen William, ”Object-Oriented Modeling and Desing”, Prentice-Hall, Eaglewood Cliffs, New Jersey, USA, 1991.

Selfridge-Field Eleanor, *Descibing Musical Information*, kirjassa ”Beyond MIDI: The Handbook of Musical Codes”, MIT Press, Cambridge, Massachusetts, USA, s. 3-37, 1997.

Simeoni Fabio, Lievens David, Connor Richard ja Manghi Paolo, *Language Bindings to XML*, Internet Computing, IEEE, Volume 7, Issue 1, 2003, sivut 19-27.

Sommerville Ian, ”Software Engineering, Fifth Edition”, Addison-Wesley, Essex, Englanti, 1995.

Sun Microsystems Inc., ”Java Sound Programmers Guide”, saatavilla WWW-muodossa osoitteessa <URL: http://java.sun.com/j2se/1.4.2/docs/guide/sound/programmer_guide/>, 24.10.2001.

Takala Antti, *Transponoivat soittimet*, kirjassa ”Suuri musiikkitietosanakirja, 6. osa”, Otava, Keuruu, s. 164-165, 1992.

Tiensuu Jukka, *Nuottikirjoitus*, kirjassa ”Suuri musiikkitietosanakirja, 4. osa”, Otava, Keuruu, s. 264-267, 1991.

Tuominen Antti Juhani & Hirvi Jussi, ”MIDI alusta alkaen”, Musiikkiuutiset/Musisoi r.y., Vantaa, 1989.

W3C, "W3C Document Object Model", saatavilla WWW-muodossa osoitteessa <URL: http://www.w3c.org/DOM/>, viitattu 29.9.2003.

Walshaw Chris, "The abc home page", saatavilla WWW-muodossa osoitteessa <URL: http://www.gre.ac.uk/~c.walshaw/abc/>, viitattu 4.11.2003.

Liitteet

Liite 1. Esimerkki MusicXML-dokumentista

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE score-partwise PUBLIC
  "-//Recordare//DTD MusicXML 0.7b Partwise//EN"
  "http://www.musicxml.org/dtds/partwise.dtd">
<score-partwise>
  <part-list>
    <part-group type="start" number="1">
    </part-group>
    <score-part id="P1">
      <part-name>Violin 1</part-name>
      <midi-instrument id="P1-I1">
        <midi-channel>1</midi-channel>
        <midi-program>41</midi-program>
      </midi-instrument>
    </score-part>
    <score-part id="P2">
      <part-name>Violin 2</part-name>
      <midi-instrument id="P2-I2">
        <midi-channel>1</midi-channel>
        <midi-program>41</midi-program>
      </midi-instrument>
    </score-part>
    <score-part id="P3">
      <part-name>Viola</part-name>
      <midi-instrument id="P3-I3">
        <midi-channel>2</midi-channel>
        <midi-program>42</midi-program>
      </midi-instrument>
    </score-part>
    <score-part id="P4">
      <part-name>Cello</part-name>
      <midi-instrument id="P4-I4">
        <midi-channel>3</midi-channel>
        <midi-program>43</midi-program>
      </midi-instrument>
    </score-part>
    <part-group type="stop" number="1"/>
  </part-list>
  <part id="P1">
    <!-- 1st part of the piece -->
  </part>
```

```
<part id="P2">
  <measure number="1">
    <attributes>
      <divisions>1</divisions>
      <key>
        <fifths>2</fifths>
        <mode>major</mode>
      </key>
      <time>
        <beats>4</beats>
        <beat-type>4</beat-type>
      </time>
      <clef>
        <sign>G</sign>
        <line>2</line>
      </clef>
    </attributes>
    <note>
      <rest/>
      <duration>2</duration>
      <type>half</type>
    </note>
    <sound dynamics="83"/>
    <note>
      <pitch>
        <step>A</step>
        <octave>4</octave>
      </pitch>
      <duration>2</duration>
      <tie type="start"/>
      <type>half</type>
      <notations>
        <tied type="start"/>
        <dynamics placement="below" relative-x="-9">
          <mf/>
        </dynamics>
      </notations>
    </note>
  </measure>
  <measure number="2">
    <note>
      <pitch>
        <step>A</step>
        <octave>4</octave>
      </pitch>
      <duration>3</duration>
      <tie type="stop"/>
```

```
        <type>half</type>
        <dot/>
        <notations>
          <tied type="stop"/>
          <slur type="start" number="1"/>
        </notations>
      </note>
      <note>
        <pitch>
          <step>G</step>
          <octave>4</octave>
        </pitch>
        <duration>1</duration>
        <type>quarter</type>
      </note>
    </measure>
    <measure number="3">
      <note>
        <pitch>
          <step>F</step>
          <alter>1</alter>
          <octave>4</octave>
        </pitch>
        <duration>2</duration>
        <type>half</type>
        <notations>
          <slur type="stop" number="1"/>
        </notations>
      </note>
      <note>
        <rest/>
        <duration>2</duration>
        <type>half</type>
      </note>
      <barline location="right">
        <bar-style>light-heavy</bar-style>
      </barline>
    </measure>
  </part>
  <part id="P3">
    <!-- 3rd part of the piece -->
  </part>
  <part id="P4">
    <!-- 4th part of the piece -->
  </part>
</score-partwise>
```

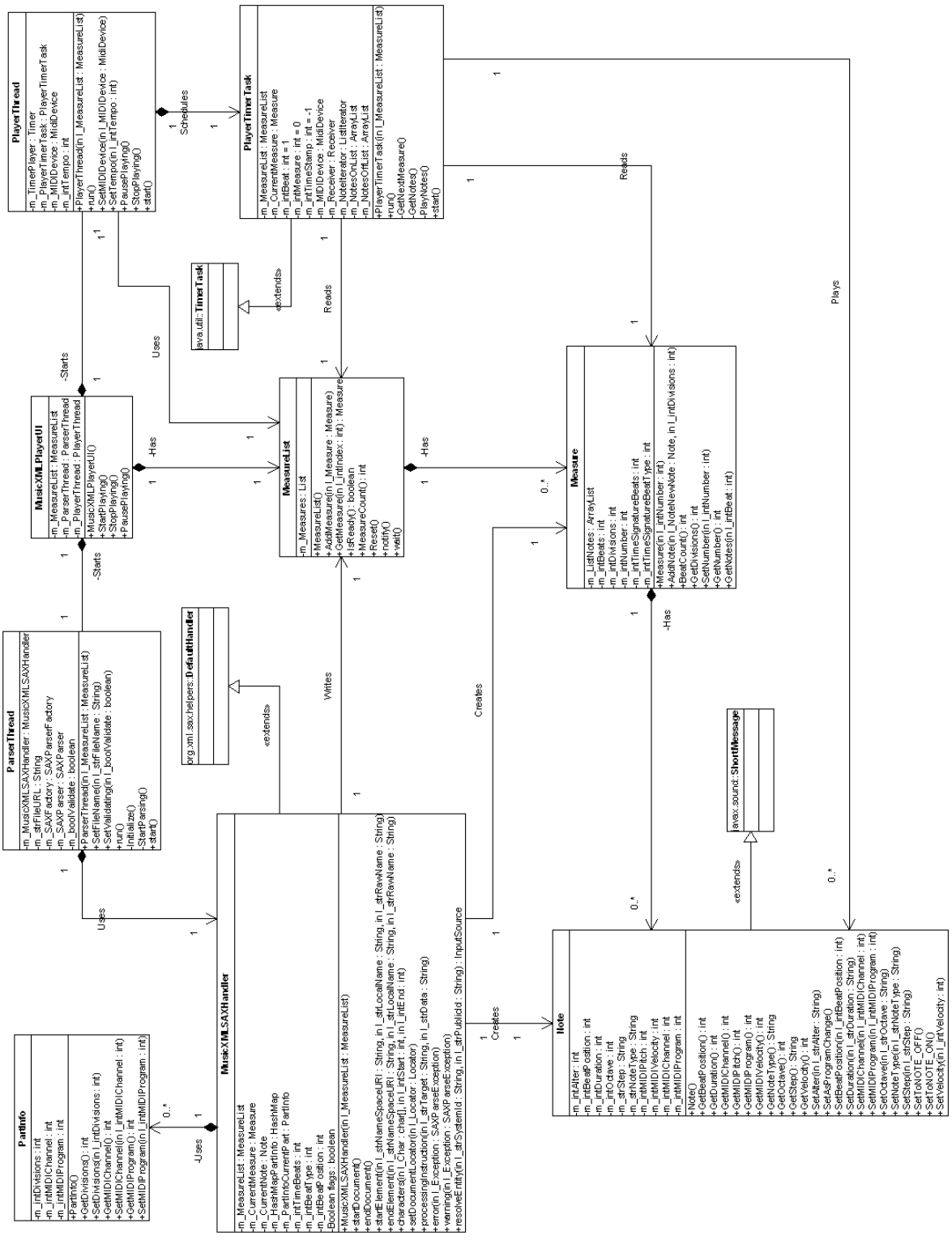
Liite 2. MusicXML:n tunnisteet

Viivastonotaatioelementti	Tunniste MusicXML:ssä
Attribuutit	<attributes>
Auditiiviseen esitykseen liittyvät parametrit	<sound>
Dynamiikka (soittovoimakkuus)	<dynamics>
Eteenpäin siirtyminen	<forward>
Juurielementti (ajallisesti jaettu)	<score-timewise>
Juurielementti (instrumenteittain jaettu)	<score-partwise>
Kromaattinen korotus tai alennus	<alter>
Kvinttiympyrän kvinttisiirtymien lukumäärä	<fifths>
Neljäsosanuotin jakaja	<divisions>
Nuotin kesto	<duration>
Nuotin nimi	<step>
Nuotin sävelkorkeuden oktaaviala	<octave>
Nuotin sävelkorkeus	<pitch>
Nuotin tyyppi	<type>
Nuotti	<note>
Nuottiavaimen tyyppi	<sign>
Nuottiavain	<clef>
Nuottiin liittyvät erityispiirteet	<notations>
Nuottiviivasto (instrumentti tai osa)	<part>

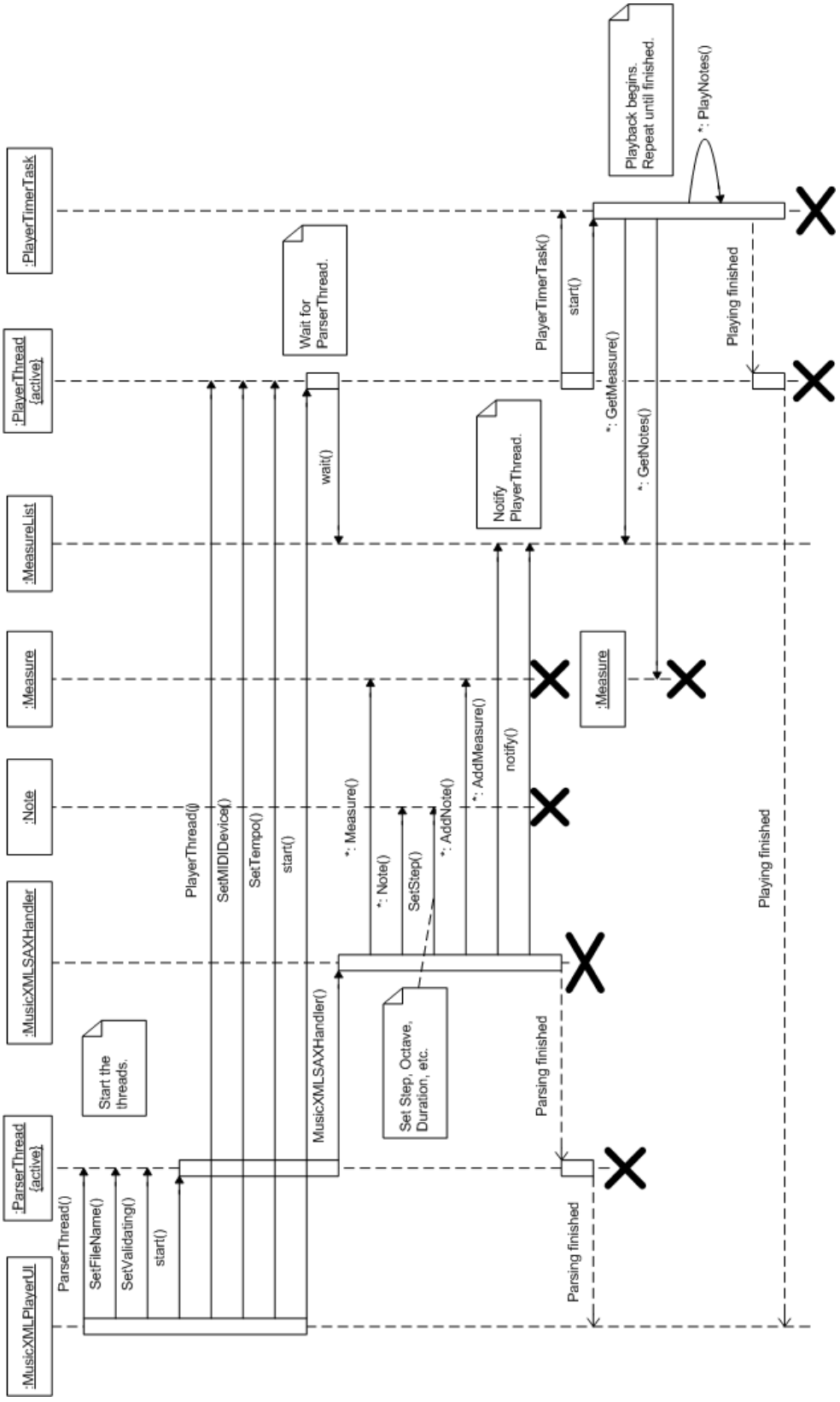
Viivastonotaatioelementti	Tunniste MusicXML:ssä
Nuottiviivastolista (instrumentti- tai osa-lista)	`<part-list>`
Sidontakaari	`<tie>`
Sointu	`<chord>`
Sävellaji	`<key>`
Sävellaji (duuri tai molli)	`<mode>`
Taaksepäin siirtyminen	`<backup>`
Tahti	`<measure>`
Tahtilaji	`<time>`
Tahtilajin osoittamien iskujen lukumäärä	`<beats>`
Tahtilajin osoittamien iskujen tyyppi	`<beat-type>`
Tahtiviiva	`<barline>`
Tauko	`<rest>`

Taulukko L.1. MusicXML:n tunnisteet.

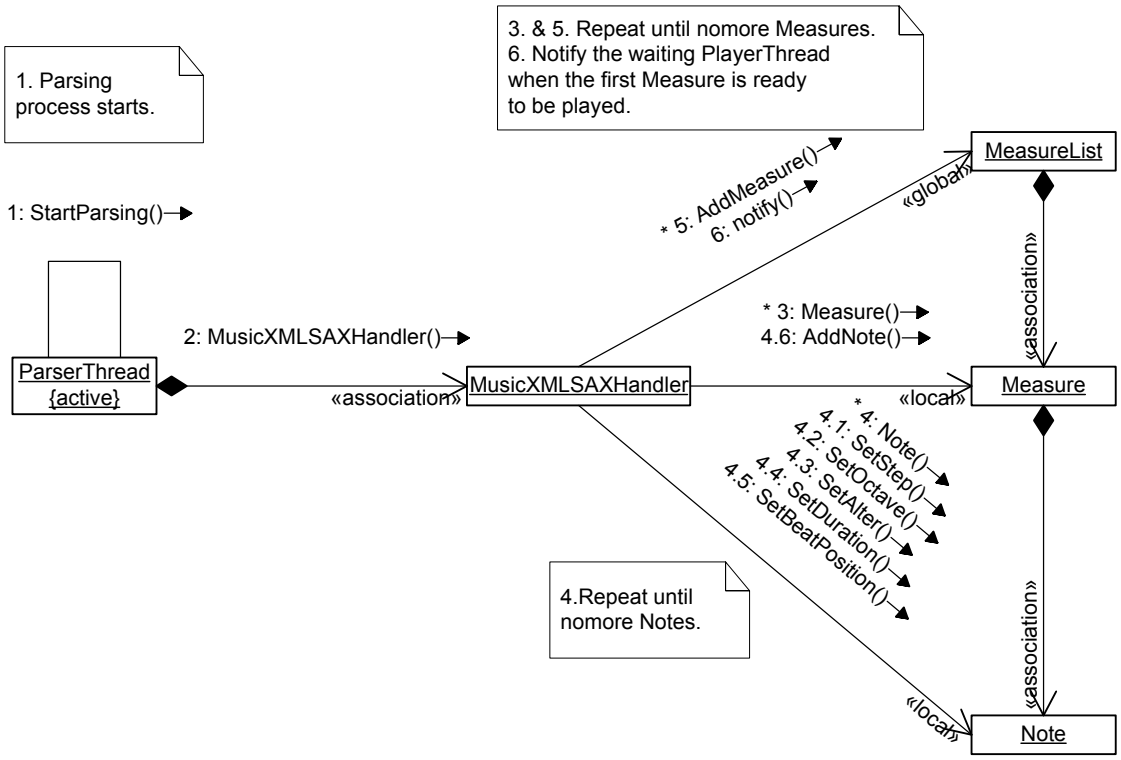
Liite 3. Sovelluksen UML-kaaviot



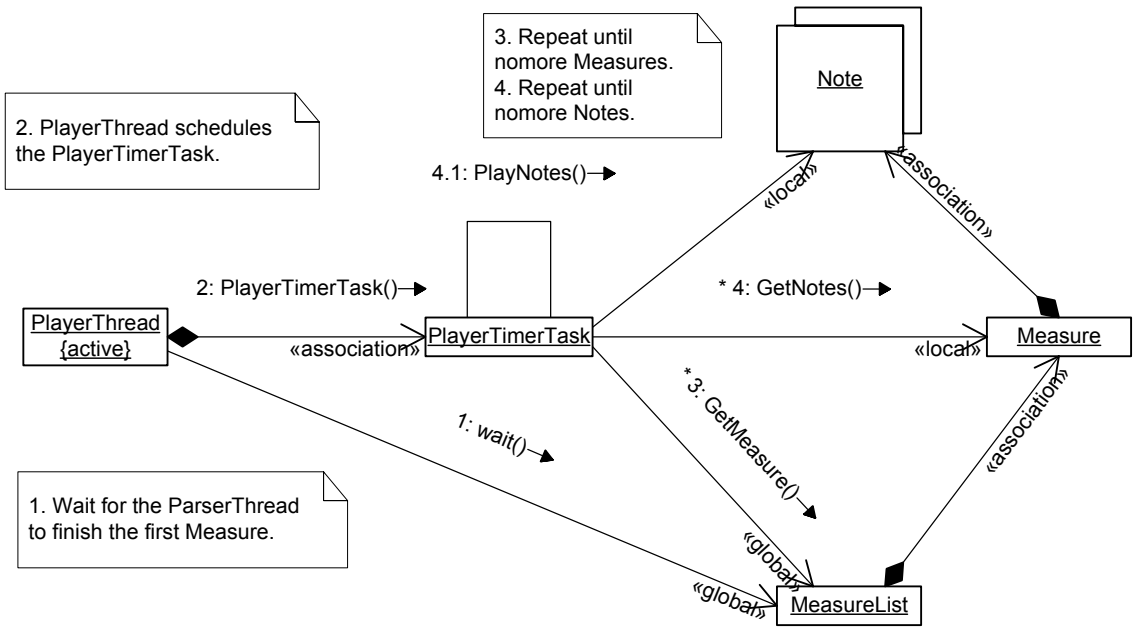
Kuva L.1. MusicXML-soittimen yksityiskohtainen luokkakaavio.



Kuva L.2. MusicXML-soittimen soiton käynnistämistoiminnon sekvenssikaavio.



Kuva L.3. Jäsenninsäikeen ja tietorakenneluokkien välinen yhteistoimintakaavio.



Kuva L.4. Soittosäikeen ja tietorakenneluokkien välinen yhteistoimintakaavio.