

A MusicXML Test Suite and a Discussion of Issues in MusicXML 2.0

Reinhold Kainhofer, reinhold@kainhofer.com

Vienna University of Technology, <http://www.fam.tuwien.ac.at/>
GNU LilyPond, <http://www.lilypond.org/>
Edition Kainhofer, Music publishing, <http://www.edition-kainhofer.com/>

Linux Audio Conference 2010, Utrecht, Netherlands
May 4, 2010

Overview

- 1 **What is MusicXML?**
 - MusicXML Specification by Recordare
- 2 A MusicXML 2.0 Test Suite
- 3 MusicXML 2.0: Semantic Ambiguities
- 4 Sub-Optimal XML Design
- 5 Missing Features
- 6 Issues in the conversion from MusicXML to LilyPond
- 7 Conclusion and Acknowledgements

What is MusicXML?

- **XML format** to represent **western-style music notation**
 - Musical content (Notes, chors, dynamics, time, key, clef, etc.)
 - Exact page layout (MusicXML 2.0)
 - Audio representation (like MIDI, not performance recording)
- Defined originally via **Document Type Definition (DTD)** files and later also via **XML Schema (XSD)** files.
- Defined by Recordare LLC, plugins for Finale, Sibelius, etc.
- Support (import and/or export) by many applications (notation, scanning, sequencers, etc.)

What is MusicXML?

- **XML format** to represent **western-style music notation**
 - Musical content (Notes, chors, dynamics, time, key, clef, etc.)
 - Exact page layout (MusicXML 2.0)
 - Audio representation (like MIDI, not performance recording)
- Defined originally via **Document Type Definition (DTD)** files and later also via **XML Schema (XSD)** files.
- Defined by Recordare LLC, plugins for Finale, Sibelius, etc.
- Support (import and/or export) by many applications (notation, scanning, sequencers, etc.)

What is MusicXML?

- **XML format** to represent **western-style music notation**
 - Musical content (Notes, chors, dynamics, time, key, clef, etc.)
 - Exact page layout (MusicXML 2.0)
 - Audio representation (like MIDI, not performance recording)
- Defined originally via **Document Type Definition (DTD)** files and later also via **XML Schema (XSD)** files.
- Defined by Recordare LLC, plugins for Finale, Sibelius, etc.
- Support (import and/or export) by many applications (notation, scanning, sequencers, etc.)

An example: Schubert's Ave Maria (excerpt)

```
<?xml version="1.0"
  encoding="UTF-8"?>
<!DOCTYPE score-partwise
  PUBLIC [...] >
<score-partwise
  version="2.0" >
  <work>
    <work-number>D.
      839</work-number>
    <work-title>Ave
      Maria</work-title>
  </work>
  <identification>
    <creator
      type="composer">F.
        Schubert</creator>
    <encoding>
      <software>Finale 2005 for
        Windows</software>
    </encoding>
  </identification>
  <defaults>
  [...]
    <music-font
      font-family="Maestro"
      font-size="18"/>
  </defaults>
  <part-list>
    <score-part id="P1">
      <part-name>Voice</part-name>
  [...]
  </score-part>
  [...]
  </part-list>
  </>
```

```
<part id="P1">
  <measure number="1">
    <attributes>
      <divisions>48</divisions>
      <key>
        <fifths>-2</fifths>
        <mode>major</mode>
      </key>
      <time symbol="common">
        <beats>4</beats>
        <beat-type>4</beat-type>
      </time>
      <clef>
        <sign>G</sign>
        <line>2</line>
      </clef>
      <staff-details
        print-object="no"/>
    </attributes>
    <note>
      <rest/>
      <duration>192</duration>
      <voice>1</voice>
    </note>
  </measure>
  <measure number="2">
    <note>
      <rest/>
      <duration>192</duration>
      <voice>1</voice>
    </note>
  </measure>
  </part>
```

```
<measure number="3" width="654" >
  <print new-system="yes"/>
  <barline location="left">
    <bar-style>heavy-light</bar-style>
    <repeat direction="forward"/>
  </barline>
  <note default-x="122" >
    <pitch>
      <step>B</step>
      <alter>-1</alter>
      <octave>4</octave>
    </pitch>
    <duration>72</duration>
    <voice>1</voice>
    <type>quarter</type>
    <dot/>
    <stem
      default-y="-55.5" >down</stem>
  <lyric default-y="-82" number="1">
    <syllabic>begin</syllabic>
    <text>A</text>
  </lyric>
  <lyric default-y="-104"
    number="2">
    <syllabic>begin</syllabic>
    <text>A</text>
  </lyric>
  <lyric default-y="-127"
    number="3">
    <syllabic>begin</syllabic>
    <text>A</text>
  </lyric>
  </note>
  <note default-x="326" >
```

Observations about MusicXML

- Extremely verbose! (e.g. first page of Ave Maria has 8768 lines / 250kB in XML)
- Score is structured into parts (here: vocal voice + Piano) ⇒ typically separate staves
- Each part structured into measures, each measure contains notes, rests, markup, etc.

Advantages

- Standardized exchange format
- Support by many applications
- Good support

Problems

- Large size / verbosity
- Specification sometimes unclear / ambiguous
- No free reference implementation, no test cases

Observations about MusicXML

- Extremely verbose! (e.g. first page of Ave Maria has 8768 lines / 250kB in XML)
- Score is structured into parts (here: vocal voice + Piano) ⇒ typically separate staves
- Each part structured into measures, each measure contains notes, rests, markup, etc.

Advantages

- Standardized exchange format
- Support by many applications
- Good support

Problems

- Large size / verbosity
- Specification sometimes unclear / ambiguous
- No free reference implementation, no test cases

Overview

- 1 What is MusicXML?
- 2 **A MusicXML 2.0 Test Suite**
 - Why a Test Suite?
 - Structure of the Test Suite
 - Some Examples of Unit Tests
 - Sample Renderings of the Test Cases
 - Availability
- 3 MusicXML 2.0: Semantic Ambiguities
- 4 Sub-Optimal XML Design
- 5 Missing Features
- 6 Issues in the conversion from MusicXML to LilyPond
- 7 Conclusion and Acknowledgements

Why a Test Suite

- No free reference implementation available (advice: Use the proprietary Dolet plugin for Finale)
- Only comments in the specification
- Only some complex sample files available at MusicXML homepage, showing off what MusicXML is able to do
- No set of basic unit test files available

Aim of this Unit Test Suite

- Full coverage including all possible elements and all combination not possible
- ⇒ Create representative test cases to catch as many common combinations as possible
- Small test cases, where a bug in one feature does not influence other cases
- Cover also some less used musical notation elements (but no cross-influences with other elements)

Why a Test Suite

- No free reference implementation available (advice: Use the proprietary Dolet plugin for Finale)
- Only comments in the specification
- Only some complex sample files available at MusicXML homepage, showing off what MusicXML is able to do
- No set of basic unit test files available

Aim of this Unit Test Suite

- Full coverage including all possible elements and all combination not possible
- ⇒ Create representative test cases to catch as many common combinations as possible
- Small test cases, where a bug in one feature does not influence other cases
- Cover also some less used musical notation elements (but no cross-influences with other elements)

Why a Test Suite

- No free reference implementation available (advice: Use the proprietary Dolet plugin for Finale)
- Only comments in the specification
- Only some complex sample files available at MusicXML homepage, showing off what MusicXML is able to do
- No set of basic unit test files available

Aim of this Unit Test Suite

- Full coverage including all possible elements and all combination not possible
- ⇒ Create representative test cases to catch as many common combinations as possible
- Small test cases, where a bug in one feature does not influence other cases
- Cover also some less used musical notation elements (but no cross-influences with other elements)

Why a Test Suite

- No free reference implementation available (advice: Use the proprietary Dolet plugin for Finale)
- Only comments in the specification
- Only some complex sample files available at MusicXML homepage, showing off what MusicXML is able to do
- No set of basic unit test files available

Aim of this Unit Test Suite

- Full coverage including all possible elements and all combination not possible
- ⇒ Create representative test cases to catch as many common combinations as possible
- Small test cases, where a bug in one feature does not influence other cases
- Cover also some less used musical notation elements (but no cross-influences with other elements)

Why a Test Suite

- No free reference implementation available (advice: Use the proprietary Dolet plugin for Finale)
- Only comments in the specification
- Only some complex sample files available at MusicXML homepage, showing off what MusicXML is able to do
- No set of basic unit test files available

Aim of this Unit Test Suite

- Full coverage including all possible elements and all combination not possible
- ⇒ Create representative test cases to catch as many common combinations as possible
- Small test cases, where a bug in one feature does not influence other cases
- Cover also some less used musical notation elements (but no cross-influences with other elements)

Why a Test Suite

- No free reference implementation available (advice: Use the proprietary Dolet plugin for Finale)
- Only comments in the specification
- Only some complex sample files available at MusicXML homepage, showing off what MusicXML is able to do
- No set of basic unit test files available

Aim of this Unit Test Suite

- Full coverage including all possible elements and all combination not possible
- ⇒ Create representative test cases to catch as many common combinations as possible
- Small test cases, where a bug in one feature does not influence other cases
- Cover also some less used musical notation elements (but no cross-influences with other elements)

Why a Test Suite

- No free reference implementation available (advice: Use the proprietary Dolet plugin for Finale)
- Only comments in the specification
- Only some complex sample files available at MusicXML homepage, showing off what MusicXML is able to do
- No set of basic unit test files available

Aim of this Unit Test Suite

- Full coverage including all possible elements and all combination not possible
- ⇒ Create representative test cases to catch as many common combinations as possible
- Small test cases, where a bug in one feature does not influence other cases
- Cover also some less used musical notation elements (but no cross-influences with other elements)

Why a Test Suite

- No free reference implementation available (advice: Use the proprietary Dolet plugin for Finale)
- Only comments in the specification
- Only some complex sample files available at MusicXML homepage, showing off what MusicXML is able to do
- No set of basic unit test files available

Aim of this Unit Test Suite

- Full coverage including all possible elements and all combination not possible
- ⇒ Create representative test cases to catch as many common combinations as possible
- Small test cases, where a bug in one feature does not influence other cases
- Cover also some less used musical notation elements (but no cross-influences with other elements)

Structure of the Test Suite

- 12 large feature categories (separate aspects of MusicXML, e.g. basic musical notation, staff attributes, note-related elements, page layout, etc.)
- Each category split into more specific aspects
- Each such aspect gets several different, non-overlapping test cases
- Structured by file name!
- More than 120 small unit test cases
- Current files: <http://www.kainhofer.com/musicxml/>

File naming scheme

*AREA***letter**-*AreaDescription*-*TestcaseDescription*.xml

where *AREA* is a number between 00 and 99, identifying the large feature area, *letter* is a running letter to enumerate the test cases within a category, and the other file name parts are human understandable descriptions.

E.g. 01b-Pitches-Intervals.xml, 21e-Chords-PickupMeasures.xml,
46e-PickupMeasure-SecondVoiceStartsLater.xml

Structure of the Test Suite

- 12 large feature categories (separate aspects of MusicXML, e.g. basic musical notation, staff attributes, note-related elements, page layout, etc.)
- Each category split into more specific aspects
- Each such aspect gets several different, non-overlapping test cases
- Structured by file name!
 - More than 120 small unit test cases
 - Current files: <http://www.kainhofer.com/musicxml/>

File naming scheme

*AREA*letter~*AreaDescription*-*TestcaseDescription*.xml

where *AREA* is a number between 00 and 99, identifying the large feature area, *letter* is a running letter to enumerate the test cases within a category, and the other file name parts are human understandable descriptions.

E.g. 01b-Pitches-Intervals.xml, 21e-Chords-PickupMeasures.xml,
46e-PickupMeasure-SecondVoiceStartsLater.xml

Structure of the Test Suite

- 12 large feature categories (separate aspects of MusicXML, e.g. basic musical notation, staff attributes, note-related elements, page layout, etc.)
- Each category split into more specific aspects
- Each such aspect gets several different, non-overlapping test cases
- Structured by file name!
- More than 120 small unit test cases
- Current files: <http://www.kainhofer.com/musicxml/>

File naming scheme

*AREA*letter~*AreaDescription*-*TestcaseDescription*.xml

where *AREA* is a number between 00 and 99, identifying the large feature area, *letter* is a running letter to enumerate the test cases within a category, and the other file name parts are human understandable descriptions.

E.g. 01b-Pitches-Intervals.xml, 21e-Chords-PickupMeasures.xml, 46e-PickupMeasure-SecondVoiceStartsLater.xml

Structure of the Test Suite

- 12 large feature categories (separate aspects of MusicXML, e.g. basic musical notation, staff attributes, note-related elements, page layout, etc.)
- Each category split into more specific aspects
- Each such aspect gets several different, non-overlapping test cases
- Structured by file name!
- More than 120 small unit test cases
- Current files: <http://www.kainhofer.com/musicxml/>

File naming scheme

*AREA*letter~*AreaDescription*-*TestcaseDescription*.xml

where *AREA* is a number between 00 and 99, identifying the large feature area, *letter* is a running letter to enumerate the test cases within a category, and the other file name parts are human understandable descriptions.

E.g. 01b-Pitches-Intervals.xml, 21e-Chords-PickupMeasures.xml, 46e-PickupMeasure-SecondVoiceStartsLater.xml

Structure of the Test Suite

- 12 large feature categories (separate aspects of MusicXML, e.g. basic musical notation, staff attributes, note-related elements, page layout, etc.)
- Each category split into more specific aspects
- Each such aspect gets several different, non-overlapping test cases
- Structured by file name!
- More than 120 small unit test cases
- Current files: <http://www.kainhofer.com/musicxml/>

File naming scheme

*AREA*letter~*AreaDescription*-*TestcaseDescription*.xml

where *AREA* is a number between 00 and 99, identifying the large feature area, *letter* is a running letter to enumerate the test cases within a category, and the other file name parts are human understandable descriptions.

E.g. 01b-Pitches-Intervals.xml, 21e-Chords-PickupMeasures.xml, 46e-PickupMeasure-SecondVoiceStartsLater.xml

Structure of the Test Suite

- 12 large feature categories (separate aspects of MusicXML, e.g. basic musical notation, staff attributes, note-related elements, page layout, etc.)
- Each category split into more specific aspects
- Each such aspect gets several different, non-overlapping test cases
- Structured by file name!
- More than 120 small unit test cases
- Current files: <http://www.kainhofer.com/musicxml/>

File naming scheme

*AREA*letter~*AreaDescription*-*TestcaseDescription*.xml

where *AREA* is a number between 00 and 99, identifying the large feature area, *letter* is a running letter to enumerate the test cases within a category, and the other file name parts are human understandable descriptions.

E.g. 01b-Pitches-Intervals.xml, 21e-Chords-PickupMeasures.xml,
46e-PickupMeasure-SecondVoiceStartsLater.xml

Feature area categories

01-09 ... Basics
01 Pitches
02 Rests
03 Rhythm
10-19 ... Staff attributes
11 Time signatures
12 Clefs
13 Key signatures
14 Staff details
20-29 ... Note-related elements
21 Chorded notes
22 Note settings, heads, etc.
23 Triplets, Tuplets
24 Grace notes
30-39 ... Dynamics, artic., spanners
31 Dynamics and other single symbols
32 Notations and Articulations
33 Spanners
40-44 ... Parts
41 Multiple parts (staves)
42 Multiple voices per staff
43 One part on multiple staves

45-49 ... Measures and repeats
45 Repeats
46 Barlines, Measures
50-54 ... Page-related issues
51 Header information
52 Page layout
55-59 ... Exact positioning
60-69 ... Vocal music
61 Lyrics
70-75 ... Instrument-specific
71 Guitar notation
72 Transposing instruments
73 Percussion
74 Figured bass
75 Other instrumental notation
80-89 ... MIDI and sound
90-99 ... Other aspects
90 Compressed MusicXML files
99 Compat. with broken MusicXML

Testing multiple possible element uses vs. separation of separate item

Example: Parenthesized noteheads (<notehead parentheses=.../>)

- Parenthesized normal noteheads
- Parenthesized non-standard noteheads
- Parenthesized noteheads inside a chord
- Parenthesized chords (all noteheads)
- Parenthesized rests (default position)
- Parenthesized rests (explicit position)

The test case 22d-Parenthesized-Noteheads.xml for parenthesized noteheads tests all these cases in one file, but each of the settings on separate notes:



Example 1: Two tied notes (33b-Spanners-Tie.xml)

```
<?xml version="1.0" encoding="ISO-8859-1"
  standalone="no"?>
<DOCTYPE score-partwise PUBLIC
  "-//Recordare//DTD MusicXML 0.6 b
  Partwise//EN"
  "http://www.musicxml.org/dtds/partwise.dtd">
<score-partwise>
<identification>
  <miscellaneous>
    <miscellaneous-field name="description">Two
      simple tied whole
      notes</miscellaneous-field>
  </miscellaneous>
</identification>
<part-list>
  <score-part id="P1"/>
</part-list>
<part id="P1">
  <measure number="1">
    <attributes>
      <divisions>1</divisions>
      <key>fifths>0</fifths></key>
      <time>
        <beats>4</beats>
        <beat-type>4</beat-type>
      </time>
      <staves>1</staves>
      <clef number="1">
        <sign>G</sign>
        <line>2</line>
      </clef>
    </attributes>
```

```
<note>
  <pitch>
    <step>F</step>
    <octave>4</octave>
  </pitch>
  <duration>4</duration>
  <tie type="start"/>
  <voice>1</voice>
  <type>whole</type>
  <notations>X<tie type="start"/></notations>
</note>
</measure>
<measure number="2">
  <note>
    <pitch>
      <step>F</step>
      <octave>4</octave>
    </pitch>
    <duration>4</duration>
    <tie type="stop"/>
    <voice>1</voice>
    <type>whole</type>
    <notations>X<tie type="stop"/></notations>
  </note>
</measure>
</part>
</score-partwise>
```



Example 2: Key signatures with microtones (33b-Spanners-Tie.xml)

[...]

```

<measure number="1">
  <attributes>
    <divisions>1</divisions>
    <key>
      <key-step>4</key-step>
      <key-alter>-1.5</key-alter>
      <key-step>6</key-step>
      <key-alter>-0.5</key-alter>
      <key-step>0</key-step>
      <key-alter>0</key-alter>
      <key-step>1</key-step>
      <key-alter>0.5</key-alter>
      <key-step>3</key-step>
      <key-alter>1.5</key-alter>
    </key>
    <time>
      <beats>2</beats>
      <beat-type>4</beat-type>
    </time>
    <clef>
      <sign>G</sign>
      <line>2</line>
    </clef>
  </attributes>
  <note>
    <pitch>

```

[...]



- Very exotic case!
- All possible alterations are checked!
- Observe bad XML design (see later!)

Connection to LilyPond

- Originally: Some **test files for musicxml2ly** (Converter from MusicXML to LilyPond; <http://www.lilypond.org/>)
- Still resides inside LilyPond source code repository
- Automated **sample renderings** can be done of MusicXML test case (No reference renderings!):
 - `musicxml2ly` is just one particular implementation with one particular interpretation of ambiguities!
 - `musicxml2ly` does not support every aspect perfectly
 - The MusicXML specification leaves many things open (⇒ left to each importing application!)
- Future plan: Include sample renderings from other applications, too. (Need to extend `lilypond-book` for this!)

Connection to LilyPond

- Originally: Some test files for `musicxml2ly` (Converter from MusicXML to LilyPond; <http://www.lilypond.org/>)
- Still resides **inside LilyPond source code repository**
- Automated **sample renderings** can be done of MusicXML test case (No reference renderings!):
 - `musicxml2ly` is just one particular implementation with one particular interpretation of ambiguities!
 - `musicxml2ly` does not support every aspect perfectly
 - The MusicXML specification leaves many things open (⇒ left to each importing application!)
- Future plan: Include sample renderings from other applications, too. (Need to extend `lilypond-book` for this!)

Connection to LilyPond

- Originally: Some test files for musicxml2ly (Converter from MusicXML to LilyPond; <http://www.lilypond.org/>)
- Still resides inside LilyPond source code repository
- Automated **sample renderings** can be done of MusicXML test case (No reference renderings!):
 - musicxml2ly is just one particular implementation with one particular interpretation of ambiguities!
 - musicxml2ly does not support every aspect perfectly
 - The MusicXML specification leaves many things open (⇒ left to each importing application!)
- Future plan: Include sample renderings from other applications, too. (Need to extend lilypond-book for this!)

Connection to LilyPond

- Originally: Some test files for musicxml2ly (Converter from MusicXML to LilyPond; <http://www.lilypond.org/>)
- Still resides inside LilyPond source code repository
- Automated **sample renderings** can be done of MusicXML test case (No reference renderings!):
 - musicxml2ly is just one particular implementation with one particular interpretation of ambiguities!
 - musicxml2ly does not support every aspect perfectly
 - The MusicXML **specification leaves many things open** (\Rightarrow left to each importing application!)
- Future plan: Include sample renderings from other applications, too. (Need to extend lilypond-book for this!)

Connection to LilyPond

- Originally: Some test files for musicxml2ly (Converter from MusicXML to LilyPond; <http://www.lilypond.org/>)
- Still resides inside LilyPond source code repository
- Automated **sample renderings** can be done of MusicXML test case (No reference renderings!):
 - musicxml2ly is just one particular implementation with one particular interpretation of ambiguities!
 - musicxml2ly does not support every aspect perfectly
 - The MusicXML specification leaves many things open (\Rightarrow left to each importing application!)
- Future plan: Include **sample renderings from other applications**, too. (Need to extend lilypond-book for this!)

Sample Renderings of the Test Cases

[<< Test cases]
[To p[ICo]s menu[Index I ?]
[>>]

[< 12 ... Key signatures]
[Up : Test cases]
[21 ... Chorded notes >]

14 ... Staff attributes

[14a-StaffAttributesLineChanges.ly](#) The number of staff lines can be modified by using the staff-lines child of the staff-details attribute. This can happen globally (the first staff has one line globally) or during the part at the beginning of a measure and even inside a measure (the second part has 3 lines initially, 4 at the beginning of the second measure, and 3 starting in the middle of the third measure).



[<< Test cases]
[To p[ICo]s menu[Index I ?]
[>>]

[< 14 ... Staff attributes]
[Up : Test cases]
[22 ... Note settings heads etc. >]

21 ... Chorded notes

[21a-Chord-Basic.ly](#) One simple chord consisting of two notes.



[21b-Chords-Techniques.ly](#) Some subsequent (identical) two-note chords.



[21c-Chords-ThreeNotesDuration.ly](#) Some three-note chords, with various durations.



[21d-Chords-ShiftsAndMeter.ly](#) Chords in the second measure, after several ornaments in the first measure and a p at the beginning of the second measure.



[21e-Chords-PickupMeasures.ly](#) Check for proper chord detection after a pickup measure (i.e. the first beat of the measure is not aligned with multiples of the time signature!)



[21f-Chord-ElementBetween.ly](#) Between the individual notes of a chord there can be direction or harmony elements, which should be properly assigned to the chord (or the position of the chord).



[<< Test cases]
[To p[ICo]s menu[Index I ?]
[>>]

[< 21 ... Chorded notes]
[Up : Test cases]
[22 ... Triplet/Tuplet >]

Availability and Download of the Test Suite

Availability of the Test Suite

- Web page: <http://kainhofer.com/musicxml/> (Download, sample renderings)
- Git repository: <http://git.sv.gnu.org/gitweb/?p=lilypond.git> (GNU)

License of the Test Suite

- MIT License (Basically BSD license): Can be used for any purpose, as long as the copyright notice (or LICENSE file) is left intact!

Availability and Download of the Test Suite

Availability of the Test Suite

- Web page: <http://kainhofer.com/musicxml/> (Download, sample renderings)
- Git repository: <http://git.sv.gnu.org/gitweb/?p=lilypond.git> (GNU)

License of the Test Suite

- MIT License (Basically BSD license): Can be used for any purpose, as long as the copyright notice (or LICENSE file) is left intact!

Overview

- 1 What is MusicXML?
- 2 A MusicXML 2.0 Test Suite
- 3 MusicXML 2.0: Semantic Ambiguities**
 - Semantic Ambiguities
 - Only Syntax Definition
 - Voice-Based
 - Attributes
 - Chords
 - Lyrics
 - Others
- 4 Sub-Optimal XML Design
- 5 Missing Features
- 6 Issues in the conversion from MusicXML to LilyPond

Semantic Ambiguities

- **MusicXML is a syntax definition**
- Music notation is very complex, has many inherent semantic restrictions.
 - These cannot be properly expressed in a XML specification (via DTD or XSD)
 - Some MusicXML import plugins: Very strict about syntax, but happily accept non-sensical musical content
- MusicXML tries to provide features of different GUI applications!
- Many unclear issues in the spec; discussion (if anyone asks) on a mailinglist without public archives; no definitive documentation for future implementors

Semantic Ambiguities

- MusicXML is a syntax definition
- Music notation is very complex, has many inherent semantic restrictions.
 - These cannot be properly expressed in a XML specification (via DTD or XSD)
 - Some MusicXML import plugins: Very strict about syntax, but happily accept non-sensical musical content
- MusicXML tries to provide features of different GUI applications!
- Many unclear issues in the spec; discussion (if anyone asks) on a mailinglist without public archives; no definitive documentation for future implementors

Semantic Ambiguities

- MusicXML is a syntax definition
- Music notation is very complex, has many inherent semantic restrictions.
 - These cannot be properly expressed in a XML specification (via DTD or XSD)
 - Some MusicXML import plugins: Very strict about syntax, but happily accept non-sensical musical content
- MusicXML tries to provide features of different GUI applications!
- Many unclear issues in the spec; discussion (if anyone asks) on a mailinglist without public archives; no definitive documentation for future implementors

Semantic Ambiguities

- MusicXML is a syntax definition
- Music notation is very complex, has many inherent semantic restrictions.
 - These cannot be properly expressed in a XML specification (via DTD or XSD)
 - Some MusicXML import plugins: Very strict about syntax, but happily accept non-sensical musical content
- MusicXML tries to provide features of different GUI applications!
- Many unclear issues in the spec; discussion (if anyone asks) on a mailinglist without public archives; no definitive documentation for future implementors

a) MusicXML is a syntax definition, no semantic

- Music has many semantic restrictions for the contents to make sense
- Cannot be expressed in restrictions to the DTD / XSD

Examples of additional semantic restrictions

- Spanners in MusicXML (e.g. slurs `<slur number="1" type="start"/>` ... `<slur number="1" type="stop"/>`) can be arbitrarily overlapping
- Impossible to specify that each spanner must be closed properly
- Crescendo / Decrescendo cannot be overlapping in the same voice



- Can overlap for different voices (e.g. Flute 1 & 2 shown in one staff)



a) MusicXML is a syntax definition, no semantic

- Music has many semantic restrictions for the contents to make sense
- Cannot be expressed in restrictions to the DTD / XSD

Examples of additional semantic restrictions

- Spanners in MusicXML (e.g. slurs `<slur number="1" type="start"/>` ... `<slur number="1" type="stop"/>`) can be arbitrarily overlapping
- Impossible to specify that each spanner must be closed properly
- Crescendo / Decrescendo cannot be overlapping in the same voice



- Can overlap for different voices (e.g. Flute 1 & 2 shown in one staff)



b) Voice-Basedness of MusicXML

- MusicXML allows different voices on a staff, but does not enforce concept of voices (many notes at the same time allowed)
- MusicXML provides `<voice>1</voice>` element to specify belonging to a particular voice
- No clear definition what a voice in MusicXML means!
- `<voice>` is OPTIONAL, many applications leave it out
 - Side-question: What does a missing `<voice>` mean? voice 1? different from voice 1?
 - It is up to the importing application!
 - Each application will handle it differently
 - Advantage of a proper specification lost
- ⇒ No information which notes belong to together to form a melody line

b) Voice-Basedness of MusicXML

- MusicXML allows different voices on a staff, but does not enforce concept of voices (many notes at the same time allowed)
- MusicXML provides `<voice>1</voice>` element to specify belonging to a particular voice
- No clear definition what a voice in MusicXML means!
- `<voice>` is OPTIONAL, many applications leave it out
 - Side-question: What does a missing `<voice>` mean? voice 1? different from voice 1?
 - It is up to the importing application!
 - Each application will handle it differently
 - Advantage of a proper specification lost
- ⇒ No information which notes belong to together to form a melody line

Importing applications will need to split up the notes in a part according to their needs ⇒ Even if <voice> given, it might not be used (overlapping notes...)



(From: Piano reduction of Mahler's 8. Symphony)

Which notes belong together? Good luck, if you don't have any voice attributes in the MusicXML file!

c) Staff and Measure Attributes

- Key, Clef, Time signature, etc. given in <attributes> blocks for a part
- What does presence of <attributes> indicate? The visual display?
- Some applications create <attributes> block for every measure, others only when a change happens
 - Case 1: Presence indicates display – breaks for apps writing attributes for every measure
 - Case 2: Presence does not force display – up to each application, imported MusicXML file might look different; No way to force a “cautionary” clef or key change!

c) Staff and Measure Attributes

- Key, Clef, Time signature, etc. given in <attributes> blocks for a part
- What does presence of <attributes> indicate? The visual display?
- Some applications create <attributes> block for every measure, others only when a change happens
 - Case 1: Presence indicates display – breaks for apps writing attributes for every measure
 - Case 2: Presence does not force display – up to each application, imported MusicXML file might look different; No way to force a “cautionary” clef or key change!

c) Staff and Measure Attributes

- Key, Clef, Time signature, etc. given in <attributes> blocks for a part
- What does presence of <attributes> indicate? The visual display?
- Some applications create <attributes> block for every measure, others only when a change happens
 - Case 1: Presence indicates display – breaks for apps writing attributes for every measure
 - Case 2: Presence does not force display – up to each application, imported MusicXML file might look different; No way to force a “cautionary” clef or key change!

c) Staff and Measure Attributes

- Key, Clef, Time signature, etc. given in <attributes> blocks for a part
- What does presence of <attributes> indicate? The visual display?
- Some applications create <attributes> block for every measure, others only when a change happens
 - Case 1: Presence indicates display – breaks for apps writing attributes for every measure
 - Case 2: Presence does not force display – up to each application, imported MusicXML file might look different; No way to force a “cautionary” clef or key change!

d) Chords in MusicXML

```

<note>
  <pitch>
    <step>F</step>
    <octave>4</octave>
  </pitch>
  <duration>960</duration>
  <voice>1</voice>
  <type>quarter</type>
</note>
<note>
  <chord/>
  <pitch>
    <step>A</step>
    <octave>4</octave>
  </pitch>
  <duration>960</duration>
  <voice>1</voice>
  <type>quarter</type>
</note>
<note>
  <chord/>
  <pitch>
    <step>C</step>
    <octave>5</octave>
  </pitch>
  <duration>960</duration>
  <voice>1</voice>
  <type>quarter</type>
</note>

```



- Chords are subsequent notes, 2nd has <chord/> element
 - Note with <chord/> must be after a note without <chord/>!
 - Can NOT be expressed (easily) in a DTD!
 - Introduced in PVG profile of Open Score Format (OSF) in XSD
- <forward.../> or <backward.../> elements before chorded note are allowed in spec... ⇒ Nonsense!
- What does it mean if different notes of a chord belong to different voices? How shall notation programs handle that?

d) Chords in MusicXML

```

<note>
  <pitch>
    <step>F</step>
    <octave>4</octave>
  </pitch>
  <duration>960</duration>
  <voice>1</voice>
  <type>quarter</type>
</note>
<note>
  <chord/>
  <pitch>
    <step>A</step>
    <octave>4</octave>
  </pitch>
  <duration>960</duration>
  <voice>1</voice>
  <type>quarter</type>
</note>
<note>
  <chord/>
  <pitch>
    <step>C</step>
    <octave>5</octave>
  </pitch>
  <duration>960</duration>
  <voice>1</voice>
  <type>quarter</type>
</note>

```



- Chords are subsequent notes, 2nd has <chord/> element
 - Note with <chord/> must be after a note without <chord/>!
 - Can NOT be expressed (easily) in a DTD!
 - Introduced in PVG profile of Open Score Format (OSF) in XSD
- <forward.../> or <backward.../> elements before chorded note are allowed in spec... ⇒ Nonsense!
- What does it mean if different notes of a chord belong to different voices? How shall notation programs handle that?

d) Chords in MusicXML

```

<note>
  <pitch>
    <step>F</step>
    <octave>4</octave>
  </pitch>
  <duration>960</duration>
  <voice>1</voice>
  <type>quarter</type>
</note>
<note>
  <chord/>
  <pitch>
    <step>A</step>
    <octave>4</octave>
  </pitch>
  <duration>960</duration>
  <voice>1</voice>
  <type>quarter</type>
</note>
<note>
  <chord/>
  <pitch>
    <step>C</step>
    <octave>5</octave>
  </pitch>
  <duration>960</duration>
  <voice>1</voice>
  <type>quarter</type>
</note>

```



- Chords are subsequent notes, 2nd has <chord/> element
 - Note with <chord/> must be after a note without <chord/>!
 - Can NOT be expressed (easily) in a DTD!
 - Introduced in PVG profile of Open Score Format (OSF) in XSD
- <forward.../> or <backward.../> elements before chorded note are allowed in spec... ⇒ Nonsense!
- What does it mean if different notes of a chord belong to different voices? How shall notation programs handle that?

d) Chords in MusicXML

```

<note>
  <pitch>
    <step>F</step>
    <octave>4</octave>
  </pitch>
  <duration>960</duration>
  <voice>1</voice>
  <type>quarter</type>
</note>
<note>
  <chord/>
  <pitch>
    <step>A</step>
    <octave>4</octave>
  </pitch>
  <duration>960</duration>
  <voice>1</voice>
  <type>quarter</type>
</note>
<note>
  <chord/>
  <pitch>
    <step>C</step>
    <octave>5</octave>
  </pitch>
  <duration>960</duration>
  <voice>1</voice>
  <type>quarter</type>
</note>

```



- Chords are subsequent notes, 2nd has <chord/> element
 - Note with <chord/> must be after a note without <chord/>!
 - Can NOT be expressed (easily) in a DTD!
 - Introduced in PVG profile of Open Score Format (OSF) in XSD
- <forward.../> or <backward.../> elements before chorded note are allowed in spec... ⇒ Nonsense!
- What does it mean if different notes of a chord belong to different voices? How shall notation programs handle that?

e) Lyrics in MusicXML

```

<note>
  <pitch>
    <step>G</step>
    <octave>4</octave>
  </pitch>
  <duration>1</duration>
  <voice>1</voice>
  <type>quarter</type>
  <lyric number="1" name="Verse">
    <syllabic>begin</syllabic>
    <text>Verse1A</text>
  </lyric>
  <lyric number="1" name="Chorus">
    <syllabic>begin</syllabic>
    <text>Chorus1A</text>
  </lyric>
  <lyric number="1" name="Chorus">
    <syllabic>begin</syllabic>
    <text>AnotherChorus1A</text>
  </lyric>
  <lyric number="2" name="Chorus">
    <syllabic>begin</syllabic>
    <text>Chorus1A</text>
  </lyric>
</note>
</note>

```

Verse1A - Chorus1A - 2B

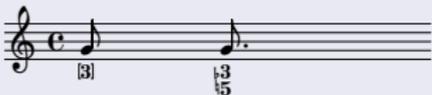
- Lyrics in MusicXML are <lyric> sub-elements of <note>
- Different stanzas can be identified by number and name attribute!
- No clear definition how to determine which syllables belong together (if no name or number or both are given)
 - Up to importing applications
- Vertical position of syllables is more important than values of name or number elements ⇒ Separation of musical content and visual display broken!

f) Figured Bass, Harp Pedals etc.

```

<figured-bass parentheses="yes">
  <figure><figure-number>3</figure-number></figure>
  <duration>4</duration>
</figured-bass>
<note>
  <pitch><step>G</step><octave>4</octave></pitch>
  <duration>4</duration>
  <voice>1</voice>
  <type>eighth</type>
</note>
<figured-bass>
  <figure><prefix>flat</prefix><figure-number>3</figure-number></figure>
  <figure><prefix>natural</prefix><figure-number>5</figure-number></figure>
  <duration>6</duration>
</figured-bass>
<note>
  <pitch><step>G</step><octave>4</octave></pitch>
  <duration>6</duration>
  <voice>1</voice>
  <type>eighth</type>
  <dot/>
</note>
</measure>

```



- Bass figures are always assigned to “first regular note that follows”
 - In XML order? i.e. if `<backward.../>` follows before next note \Rightarrow different time
 - In time order? Hard to determine the next following note!
 - Problem is that restriction (`<note>` has to follow immediately) is not mentioned / defined in specification!!!
- slash of the `<suffix>` child element does not distinguish forward/backward slashes (same meaning, different display, up to importing applications)

f) Figured Bass, Harp Pedals etc.

```

<harp-pedals>
  <pedal-tuning>
    <pedal-step>D</pedal-step>
    <pedal-alter>0</pedal-alter>
  </pedal-tuning>
  <pedal-tuning>
    <pedal-step>C</pedal-step>
    <pedal-alter>-1</pedal-alter>
  </pedal-tuning>
  <pedal-tuning>
    <pedal-step>B</pedal-step>
    <pedal-alter>-1</pedal-alter>
  </pedal-tuning>
  <pedal-tuning>
    <pedal-step>E</pedal-step>
    <pedal-alter>0</pedal-alter>
  </pedal-tuning>
  <pedal-tuning>
    <pedal-step>F</pedal-step>
    <pedal-alter>0</pedal-alter>
  </pedal-tuning>
  <pedal-tuning>
    <pedal-step>G</pedal-step>
    <pedal-alter>1</pedal-alter>
  </pedal-tuning>
  <pedal-tuning>
    <pedal-step>A</pedal-step>
    <pedal-alter>-1</pedal-alter>
  </pedal-tuning>
</harp-pedals>

```



- Harp pedals: pedal states recommended in order D, C, B, E, F, G and A pedal.
- What if different order is used in MusicXML? Shall XML order be used or always the default order?
- No way to customize where the vertical separator will be displayed.

Overview

- 1 What is MusicXML?
- 2 A MusicXML 2.0 Test Suite
- 3 MusicXML 2.0: Semantic Ambiguities
- 4 Sub-Optimal XML Design**
 - Strict Element-Order
 - XML Element Naming
 - Metronome Markings
 - Enumerated Data Types
- 5 Missing Features
- 6 Issues in the conversion from MusicXML to LilyPond
- 7 Conclusion and Acknowledgements

Sub-Optimal XML Design Issues

- Not everything in the MusicXML specification is consistent!
- Backward compatibility in future versions ⇒ Can not be changed any more

Sub-Optimal XML Design Issues

- Not everything in the MusicXML specification is consistent!
- Backward compatibility in future versions \Rightarrow Can not be changed any more

a) Strict Order of Elements

DTD definition of the <note> element

```
<!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*)>
```

a) Strict Order of Elements

DTD definition of the <note> element

```
<!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*)>
```

- Forces a fixed order of the children!
- Counter-intuitive order: duration (time length), then voice, then type (visual display)!
- Historically: Need restriction that some elements can only be there once
⇒ Cannot be done (easily) in a DTD without fixing element order!
- Now: Would be possible in XSD, but for backward-compatibility fixed order is kept in the XSD, too

a) Strict Order of Elements

DTD definition of the <note> element

```
<!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*)>
```

- Forces a fixed order of the children!
- Counter-intuitive order: duration (time length), then voice, then type (visual display)!
- Historically: Need restriction that some elements can only be there once
⇒ Cannot be done (easily) in a DTD without fixing element order!
- Now: Would be possible in XSD, but for backward-compatibility fixed order is kept in the XSD, too

a) Strict Order of Elements

DTD definition of the <note> element

```
<!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*)>
```

- Forces a fixed order of the children!
- Counter-intuitive order: duration (time length), then voice, then type (visual display)!
- Historically: Need restriction that some elements can only be there once
⇒ Cannot be done (easily) in a DTD without fixing element order!
- Now: Would be possible in XSD, but for backward-compatibility fixed order is kept in the XSD, too

a) Strict Order of Elements

DTD definition of the <note> element

```
<!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*)>
```

- Forces a fixed order of the children!
- Counter-intuitive order: duration (time length), then voice, then type (visual display)!
- Historically: Need restriction that some elements can only be there once
⇒ Cannot be done (easily) in a DTD without fixing element order!
- Now: Would be possible in XSD, but for backward-compatibility fixed order is kept in the XSD, too

a) Strict Order of Elements

DTD definition of the <note> element

```
<!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*)>
```

Which of the following snippets is correct?

```
<note>
  <pitch>
    <step>G</step>
    <alter>1</alter>
    <octave>2</octave>
  </pitch>
  <accidental>sharp</accidental>
  <duration>1</duration>
  <type>quarter</type>
  <voice>1</voice>
</note>
```

```
<note>
  <pitch>
    <step>G</step>
    <alter>1</alter>
    <octave>2</octave>
  </pitch>
  <duration>1</duration>
  <voice>1</voice>
  <type>quarter</type>
  <accidental>sharp</accidental>
</note>
```

a) Strict Order of Elements

DTD definition of the <note> element

```
<!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*)>
```

Which of the following snippets is correct?

Incorrect

```
<note>
  <pitch>
    <step>G</step>
    <alter>1</alter>
    <octave>2</octave>
  </pitch>
  <accidental>sharp</accidental>
  <duration>1</duration>
  <type>quarter</type>
  <voice>1</voice>
</note>
```

Correct

```
<note>
  <pitch>
    <step>G</step>
    <alter>1</alter>
    <octave>2</octave>
  </pitch>
  <duration>1</duration>
  <voice>1</voice>
  <type>quarter</type>
  <accidental>sharp</accidental>
</note>
```

a) Strict Order of Elements

DTD definition of the <note> element

```
<!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*)>
```

Which of the following snippets is correct?

Incorrect

```
<note>
  <pitch>
    <step>G</step>
    <alter>1</alter>
    <octave>2</octave>
  </pitch>
  <accidental>sharp</accidental>
  <duration>1</duration>
  <type>quarter</type>
  <voice>1</voice>
</note>
```

Correct

```
<note>
  <pitch>
    <step>G</step>
    <alter>1</alter>
    <octave>2</octave>
  </pitch>
  <duration>1</duration>
  <voice>1</voice>
  <type>quarter</type>
  <accidental>sharp</accidental>
</note>
```

b) Element Naming for Pitch Information

Normal note pitch

```
<note>
  <pitch>
    <step>E</step>
    <alter>-1</alter>
    <octave>2</octave>
  </pitch>
  <duration>1</duration>
  <accidental>flat</accidental>
</note>
```

Root pitch of chord

```
<harmony>
  <root>
    <root-step>E</root-step>
    <root-alter>-1</root-alter>
  </root>
  <kind>major</kind>
</harmony>
```

Tuning of Tab staves

```
<attributes>
  <staff-details>
    <staff-lines>6</staff-lines>
    <staff-tuning line="1">
      <tuning-step>E</tuning-step>
      <tuning-alter>-1</tuning-alter>
      <tuning-octave>3</tuning-octave>
    </staff-tuning>
  [...]
</staff-details>
</attributes>
```

- All provide alteration / octave information for containing element!
- Why not use the same element and take context into account?

Normal note pitch

```
<note>
  <pitch>
    <step>E</step>
    <alter>-1</alter>
    <octave>2</octave>
  </pitch>
  <duration>1</duration>
  <accidental>flat</accidental>
</note>
```

Root pitch of chord

```
<harmony>
  <root>
    <step>E</step>
    <alter>-1</alter>
  </root>
  <kind>major</kind>
</harmony>
```

Tuning of Tab staves

```
<attributes>
  <staff-details>
    <staff-lines>6</staff-lines>
    <staff-tuning line="1">
      <step>E</step>
      <alter>-1</alter>
      <octave>3</octave>
    </staff-tuning>
  [...]
</staff-details>
</attributes>
```

b) Element Naming for Pitch Information

Normal note pitch

```
<note>
  <pitch>
    <step>E</step>
    <alter>-1</alter>
    <octave>2</octave>
  </pitch>
  <duration>1</duration>
  <accidental>flat</accidental>
</note>
```

Root pitch of chord

```
<harmony>
  <root>
    <root-step>E</root-step>
    <root-alter>-1</root-alter>
  </root>
  <kind>major</kind>
</harmony>
```

Tuning of Tab staves

```
<attributes>
  <staff-details>
    <staff-lines>6</staff-lines>
    <staff-tuning line="1">
      <tuning-step>E</tuning-step>
      <tuning-alter>-1</tuning-alter>
      <tuning-octave>3</tuning-octave>
    </staff-tuning>
  [...]
</staff-details>
</attributes>
```

- All provide alteration / octave information for containing element!
- Why not use the same element and take context into account?

Normal note pitch

```
<note>
  <pitch>
    <step>E</step>
    <alter>-1</alter>
    <octave>2</octave>
  </pitch>
  <duration>1</duration>
  <accidental>flat</accidental>
</note>
```

Root pitch of chord

```
<harmony>
  <root>
    <step>E</step>
    <alter>-1</alter>
  </root>
  <kind>major</kind>
</harmony>
```

Tuning of Tab staves

```
<attributes>
  <staff-details>
    <staff-lines>6</staff-lines>
    <staff-tuning line="1">
      <step>E</step>
      <alter>-1</alter>
      <octave>3</octave>
    </staff-tuning>
  [...]
</staff-details>
</attributes>
```

b) Element Naming for Pitch Information

Normal note pitch

```
<note>
  <pitch>
    <step>E</step>
    <alter>-1</alter>
    <octave>2</octave>
  </pitch>
  <duration>1</duration>
  <accidental>flat</accidental>
</note>
```

Root pitch of chord

```
<harmony>
  <root>
    <root-step>E</root-step>
    <root-alter>-1</root-alter>
  </root>
  <kind>major</kind>
</harmony>
```

Tuning of Tab staves

```
<attributes>
  <staff-details>
    <staff-lines>6</staff-lines>
    <staff-tuning line="1">
      <tuning-step>E</tuning-step>
      <tuning-alter>-1</tuning-alter>
      <tuning-octave>3</tuning-octave>
    </staff-tuning>
  [...]
</staff-details>
</attributes>
```

- All provide alteration / octave information for containing element!
- Why not use the same element and take context into account?

Normal note pitch

```
<note>
  <pitch>
    <step>E</step>
    <alter>-1</alter>
    <octave>2</octave>
  </pitch>
  <duration>1</duration>
  <accidental>flat</accidental>
</note>
```

Root pitch of chord

```
<harmony>
  <root>
    <step>E</step>
    <alter>-1</alter>
  </root>
  <kind>major</kind>
</harmony>
```

Tuning of Tab staves

```
<attributes>
  <staff-details>
    <staff-lines>6</staff-lines>
    <staff-tuning line="1">
      <step>E</step>
      <alter>-1</alter>
      <octave>3</octave>
    </staff-tuning>
  [...]
</staff-details>
</attributes>
```

b) Element Naming for Pitch Information

Normal note pitch

```
<note>
  <pitch>
    <step>E</step>
    <alter>-1</alter>
    <octave>2</octave>
  </pitch>
  <duration>1</duration>
  <accidental>flat</accidental>
</note>
```

Root pitch of chord

```
<harmony>
  <root>
    <root-step>E</root-step>
    <root-alter>-1</root-alter>
  </root>
  <kind>major</kind>
</harmony>
```

Tuning of Tab staves

```
<attributes>
  <staff-details>
    <staff-lines>6</staff-lines>
    <staff-tuning line="1">
      <tuning-step>E</tuning-step>
      <tuning-alter>-1</tuning-alter>
      <tuning-octave>3</tuning-octave>
    </staff-tuning>
  [...]
</staff-details>
</attributes>
```

- All provide alteration / octave information for containing element!
- Why not use the same element and take context into account?

Normal note pitch

```
<note>
  <pitch>
    <step>E</step>
    <alter>-1</alter>
    <octave>2</octave>
  </pitch>
  <duration>1</duration>
  <accidental>flat</accidental>
</note>
```

Root pitch of chord

```
<harmony>
  <root>
    <step>E</step>
    <alter>-1</alter>
  </root>
  <kind>major</kind>
</harmony>
```

Tuning of Tab staves

```
<attributes>
  <staff-details>
    <staff-lines>6</staff-lines>
    <staff-tuning line="1">
      <step>E</step>
      <alter>-1</alter>
      <octave>3</octave>
    </staff-tuning>
  [...]
</staff-details>
</attributes>
```

c) Metronome Markings and Non-Standard Key Signatures

Contrast the over-correctness for `<*-step>` and `<*-alter>` (ignoring context, new name for basically same functionality) to Metronome marks and Non-Standard Key Signature definitions:

DTD for Metronome marks

```
<!ELEMENT metronome (
  beat-unit, beat-unit-dot*,
  ( ... |
    (beat-unit, beat-unit-dot*) )
)>
```

```
<metronome>
  <beat-unit>quarter</beat-unit>
  <beat-unit-dot/>
  <beat-unit-dot/>
  <beat-unit>half</beat-unit>
  <beat-unit-dot/>
</metronome>
```

- Tempo changes "old value = new value"
- Optional dots
- second unit can not be obtained directly!

DTD for Non-std. keys

```
<!ELEMENT key (
  ( cancel?, fifths, mode?) |
  ((key-step, key-alter)* ) ),
  key-octave*
)>
```

```
<key>
  <key-step>0</key-step>
  <key-alter>-2</key-alter>
  <key-step>4</key-step>
  <key-alter>2</key-alter>
  <key-octave
    number="1">2</key-octave>
  <key-octave
    number="2">4</key-octave>
</key>
```

- Used to define accidentals for non-standard key signatures
- Step and alteration alternate
- Optional octave identifiers follow later!!!

c) Metronome Markings and Non-Standard Key Signatures

Contrast the over-correctness for `<*-step>` and `<*-alter>` (ignoring context, new name for basically same functionality) to Metronome marks and Non-Standard Key Signature definitions:

DTD for Metronome marks

```
<!ELEMENT metronome (
  beat-unit, beat-unit-dot*,
  ( ... |
    (beat-unit, beat-unit-dot*) )
)>
```

```
<metronome>
  <beat-unit>quarter</beat-unit>
  <beat-unit-dot/>
  <beat-unit-dot/>
  <beat-unit>half</beat-unit>
  <beat-unit-dot/>
</metronome>
```

- Tempo changes "old value = new value"
- Optional dots
- second unit can not be obtained directly!

DTD for Non-std. keys

```
<!ELEMENT key (
  ( cancel?, fifths, mode? ) |
  ((key-step, key-alter)* ) ),
key-octave*
)>
```

```
<key>
  <key-step>0</key-step>
  <key-alter>-2</key-alter>
  <key-step>4</key-step>
  <key-alter>2</key-alter>
  <key-octave
    number="1">2</key-octave>
  <key-octave
    number="2">4</key-octave>
</key>
```

- Used to define accidentals for non-standard key signatures
- Step and alteration alternate
- Optional octave identifiers follow later!!!

c) Metronome Markings and Non-Standard Key Signatures

Contrast the over-correctness for `<*-step>` and `<*-alter>` (ignoring context, new name for basically same functionality) to Metronome marks and Non-Standard Key Signature definitions:

DTD for Metronome marks

```
<!ELEMENT metronome (
  beat-unit, beat-unit-dot*,
  ( ... |
    (beat-unit, beat-unit-dot*) )
)>
```

```
<metronome>
  <beat-unit>quarter</beat-unit>
  <beat-unit-dot/>
  <beat-unit-dot/>
  <beat-unit>half</beat-unit>
  <beat-unit-dot/>
</metronome>
```

- Tempo changes "old value = new value"
- Optional dots
- second unit can not be obtained directly!

DTD for Non-std. keys

```
<!ELEMENT key (
  (cancel?, fifths, mode?) |
  ((key-step, key-alter)* ) ),
key-octave*
)>
```

```
<key>
  <key-step>0</key-step>
  <key-alter>-2</key-alter>
  <key-step>4</key-step>
  <key-alter>2</key-alter>
  <key-octave
    number="1">2</key-octave>
  <key-octave
    number="2">4</key-octave>
</key>
```

- Used to define accidentals for non-standard key signatures
- Step and alteration alternate
- Optional octave identifiers follow later!!!

d) Data Types in DTD / XSD (Enumerations and Integers)

- **DTD:** Mostly **#PCDATA** for all attributes
 - Possible values for enumerations described in comments
 - Inaccessible to syntax checkers!
 - Meaning/Handling of other values undefined
- **XSD:** **Enumerations**
 - All possible values listed
 - Available to syntax checkers
 - MusicXML cannot be extended (new values cannot be added)

d) Data Types in DTD / XSD (Enumerations and Integers)

- **DTD**: Mostly **#PCDATA** for all attributes
 - Possible values for enumerations described in comments
 - Inaccessible to syntax checkers!
 - Meaning/Handling of other values undefined
- **XSD**: **Enumerations**
 - All possible values listed
 - Available to syntax checkers
 - MusicXML cannot be extended (new values cannot be added)

Overview

- 1 What is MusicXML?
- 2 A MusicXML 2.0 Test Suite
- 3 MusicXML 2.0: Semantic Ambiguities
- 4 Sub-Optimal XML Design
- 5 Missing Features**
 - Credit Elements: Header markup and purpose of credits
 - System separators and cadenzas
- 6 Issues in the conversion from MusicXML to LilyPond
- 7 Conclusion and Acknowledgements

Missing Features in MusicXML: Headers and Credit elements

- Document-wide headers/footers
 - `<credit page="..">` only allows page number (1 by default, `xsd:positiveInteger` in XSD)
 - Document-wide headers the same for all / all even / all odd pages
 - Suggestion: Allow "all", "even" and "odd" for the page attribute:

Suggestion for document-wide headers / footers

```
<credit page="even">
  <credit-words default-x="955"
    default-y="20">Even
  footer</credit-words>
</credit>
```

- Purpose of credit elements
 - All header, title, author labels are credit elements
 - `credit` stores only position on page, but not what information it displays
 - Impossible to extract metadata information about page layout (e.g. the arranger is placed on the upper left of the score)
 - Suggestion: Add an enumerated type attribute to `<credit>` element

Suggestion for information about credit element

```
<credit type="title">
  <credit-words
    default-x="624" default-y="1387"
    justify="right">Score
  title</credit-words>
</credit>
```

Missing Features in MusicXML: Headers and Credit elements

- Document-wide headers/footers
 - `<credit page="..">` only allows page number (1 by default, `xsd:positiveInteger` in XSD)
 - Document-wide headers the same for all / all even / all odd pages
 - Suggestion: Allow "all", "even" and "odd" for the page attribute:

Suggestion for document-wide headers / footers

```
<credit page="even">
  <credit-words default-x="955"
    default-y="20">Even
    footer</credit-words>
</credit>
```

- Purpose of credit elements
 - All header, title, author labels are credit elements
 - `credit` stores only position on page, but not what information it displays
 - Impossible to extract metadata information about page layout (e.g. the arranger is placed on the upper left of the score)
 - Suggestion: Add an enumerated type attribute to `<credit>` element

Suggestion for information about credit element

```
<credit type="title">
  <credit-words
    default-x="624" default-y="1387"
    justify="right">Score
  title</credit-words>
</credit>
```

Missing Features in MusicXML: Headers and Credit elements

- Document-wide headers/footers
 - `<credit page="..">` only allows page number (1 by default, `xsd:positiveInteger` in XSD)
 - Document-wide headers the same for all / all even / all odd pages
 - Suggestion: Allow "all", "even" and "odd" for the page attribute:

Suggestion for document-wide headers / footers

```
<credit page="even">
  <credit-words default-x="955"
    default-y="20">Even
    footer</credit-words>
</credit>
```

- Purpose of credit elements
 - All header, title, author labels are credit elements
 - credit stores only position on page, but not what information it displays
 - Impossible to extract metadata information about page layout (e.g. the arranger is placed on the upper left of the score)
 - Suggestion: Add an enumerated type attribute to `<credit>` element

Suggestion for information about credit element

```
<credit type="title">
  <credit-words
    default-x="624" default-y="1387"
    justify="right">Score
  title</credit-words>
</credit>
```

Missing Features in MusicXML: Headers and Credit elements

- Document-wide headers/footers
 - `<credit page="..">` only allows page number (1 by default, `xsd:positiveInteger` in XSD)
 - Document-wide headers the same for all / all even / all odd pages
 - Suggestion: Allow "all", "even" and "odd" for the page attribute:

Suggestion for document-wide headers / footers

```

<credit page="even">
  <credit-words default-x="955"
    default-y="20">Even
    footer</credit-words>
</credit>
  
```

- Purpose of credit elements
 - All header, title, author labels are credit elements
 - credit stores only position on page, but not what information it displays
 - Impossible to extract metadata information about page layout (e.g. the arranger is placed on the upper left of the score)
 - Suggestion: Add an enumerated type attribute to `<credit>` element

Suggestion for information about credit element

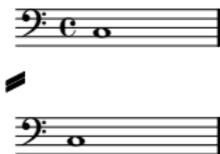
```

<credit type="title">
  <credit-words
    default-x="624" default-y="1387"
    justify="right">Score
  title</credit-words>
</credit>
  
```

Missing Features: System separators and Cadenzas

- System separator

- Systems in full scores separates by two slashes, currently not possible in MusicXML



```
Suggestion for system delimiter (in global defaults)
<defaults>
  <system-layout>
    <system-separator>double-slash</system-separator>
  </system-layout>
</defaults>
```

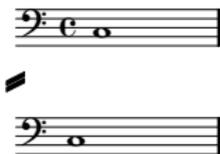
- Cadenzas

- No way to properly encode a cadenza and detect it as a cadenza
- A measure can have arbitrary number of beats (irrespective of time signature!)
- No way to mark the beginning of the cadenza
- No way to distinguish a real cadenza from an incorrect measure
- Problems with applications trying to check a MusicXML for (musical) correctness

Missing Features: System separators and Cadenzas

- System separator

- Systems in full scores separates by two slashes, currently not possible in MusicXML



Suggestion for system delimiter (in global defaults)

```
<defaults>
  <system-layout>
    <system-separator>double-slash</system-separator>
  </system-layout>
</defaults>
```

- Cadenzas

- No way to properly encode a cadenza and detect it as a cadenza
- A measure can have arbitrary number of beats (irrespective of time signature!)
- No way to mark the beginning of the cadenza
- No way to distinguish a real cadenza from an incorrect measure
- Problems with applications trying to check a MusicXML for (musical) correctness

Overview

- 1 What is MusicXML?
- 2 A MusicXML 2.0 Test Suite
- 3 MusicXML 2.0: Semantic Ambiguities
- 4 Sub-Optimal XML Design
- 5 Missing Features
- 6 Issues in the conversion from MusicXML to LilyPond**
 - Staff-Assigned Items
 - Voice-Based vs. Measure-Based
 - Page Layout and Metadata
 - Musical Content vs. Graphical Representation
 - Workarounds in Some GUI Applications
- 7 Conclusion and Acknowledgements

Staff-Assigned Items

- MusicXML: "Directions" like dynamics assigned to staff position or note
- LilyPond: Everything assigned to note (possibly invisible spacer note "s")

⇒ All staff-assigned items need to be assigned to appropriate note in LilyPond

- Which note? The nearest note? What if there is no near note?
- Horizontal offsets to/from the note?

Special case: Staves with multiple voices/instruments



- First "p" applies to both voices (two notes present)
- "f" only to first voice (only first voice present)
- "mf" only to second
- last "p" applies only to first voice (even though both voices present!)

To generate instrumental parts, you want the dynamics assigned to correct voice (in many cases to both voices!)

Staff-Assigned Items

- MusicXML: "Directions" like dynamics assigned to staff position or note
 - LilyPond: Everything assigned to note (possibly invisible spacer note "s")
- ⇒ All staff-assigned items need to be assigned to appropriate note in LilyPond
- Which note? The nearest note? What if there is no near note?
 - Horizontal offsets to/from the note?

Special case: Staves with multiple voices/instruments



- First "p" applies to both voices (two notes present)
- "f" only to first voice (only first voice present)
- "mf" only to second
- last "p" applies only to first voice (even though both voices present!)

To generate instrumental parts, you want the dynamics assigned to correct voice (in many cases to both voices!)

Staff-Assigned Items

- MusicXML: "Directions" like dynamics assigned to staff position or note
 - LilyPond: Everything assigned to note (possibly invisible spacer note "s")
- ⇒ All staff-assigned items need to be assigned to appropriate note in LilyPond
- Which note? The nearest note? What if there is no near note?
 - Horizontal offsets to/from the note?

Special case: Staves with multiple voices/instruments



- First "p" applies to both voices (two notes present)
- "f" only to first voice (only first voice present)
- "mf" only to second
- last "p" applies only to first voice (even though both voices present!)

To generate instrumental parts, you want the dynamics assigned to correct voice (in many cases to both voices!)

Staff-Assigned Items

- MusicXML: "Directions" like dynamics assigned to staff position or note
 - LilyPond: Everything assigned to note (possibly invisible spacer note "s")
- ⇒ All staff-assigned items need to be assigned to appropriate note in LilyPond
- Which note? The nearest note? What if there is no near note?
 - Horizontal offsets to/from the note?

Special case: Staves with multiple voices/instruments



- First "p" applies to both voices (two notes present)
- "f" only to first voice (only first voice present)
- "mf" only to second
- last "p" applies only to first voice (even though both voices present!)

To generate instrumental parts, you want the dynamics assigned to correct voice (in many cases to both voices!)

Staff-Assigned Items

- MusicXML: "Directions" like dynamics assigned to staff position or note
 - LilyPond: Everything assigned to note (possibly invisible spacer note "s")
- ⇒ All staff-assigned items need to be assigned to appropriate note in LilyPond
- Which note? The nearest note? What if there is no near note?
 - Horizontal offsets to/from the note?

Special case: Staves with multiple voices/instruments



- First "p" applies to both voices (two notes present)
- "f" only to first voice (only first voice present)
- "mf" only to second
- last "p" applies only to first voice (even though both voices present!)

To generate instrumental parts, you want the dynamics assigned to correct voice (in many cases to both voices!)

Staff-Assigned Items

- MusicXML: "Directions" like dynamics assigned to staff position or note
 - LilyPond: Everything assigned to note (possibly invisible spacer note "s")
- ⇒ All staff-assigned items need to be assigned to appropriate note in LilyPond
- Which note? The nearest note? What if there is no near note?
 - Horizontal offsets to/from the note?

Special case: Staves with multiple voices/instruments



- First "p" applies to both voices (two notes present)
- "f" only to first voice (only first voice present)
- "mf" only to second
- last "p" applies only to first voice (even though both voices present!)

To generate instrumental parts, you want the dynamics assigned to correct voice (in many cases to both voices!)

Measure Length: Voice-Based vs. Measure-Based

Different handling of measure lengths

- MusicXML: **Measures explicitly defined** in .xml file, can contain arbitrary number of beats
- LilyPond: **Music expressions** for each voice separately; only (optional) bar line checks, but no explicit concept of measures; Bar can only contain beats according to time signature

In LilyPond, voices are independently split into measures according to time signature, later voices are synchronized. ⇒ each voice must have same number of beats!

Overlapping notes (with or without explicit voice)

- MusicXML: Several **notes can overlap**, whether they belong to different voices or to the same
- LilyPond: Each voice can only have **one active note**/chord at a time

⇒ Need to split up overlapping notes into different voices in LilyPond (hard to get right!)

Page Layout and Metadata

Handling of Metadata and of Title, Author, Header, ... data

- MusicXML:
 - **Metadata** stored in **<identification>** tag (never displayed),
 - Title, author, header, etc. printed via **score-wide <credit> tags** – No attribute for type of information shown!
- LilyPond:
 - **Only metadata** explicitly entered (header block containing title, author, etc.)
 - Title, author, header, etc. **automatically generated** from metadata

To produce the same layout as MusicXML: need to extract metadata information from the <credit> elements – **Not Possible!!!**

Different coordinate systems

MusicXML places headers and other credit markup at absolute coordinates on page, not possible in lilypond (only relative coordinates!)

Page Layout and Metadata

Handling of Metadata and of Title, Author, Header, ... data

- MusicXML:
 - **Metadata** stored in **<identification>** tag (never displayed),
 - Title, author, header, etc. printed via **score-wide <credit> tags** – No attribute for type of information shown!
- LilyPond:
 - **Only metadata** explicitly entered (header block containing title, author, etc.)
 - Title, author, header, etc. **automatically generated** from metadata

To produce the same layout as MusicXML: need to **extract metadata information from the <credit> elements** – **Not Possible!!!**

Different coordinate systems

MusicXML places headers and other credit markup at absolute coordinates on page, not possible in lilypond (only relative coordinates!)

Page Layout and Metadata

Handling of Metadata and of Title, Author, Header, ... data

- MusicXML:
 - **Metadata** stored in **<identification>** tag (never displayed),
 - Title, author, header, etc. printed via **score-wide <credit> tags** – No attribute for type of information shown!
- LilyPond:
 - **Only metadata** explicitly entered (header block containing title, author, etc.)
 - Title, author, header, etc. **automatically generated** from metadata

To produce the same layout as MusicXML: need to extract metadata information from the <credit> elements – **Not Possible!!!**

Different coordinate systems

MusicXML places headers and other credit markup at absolute coordinates on page, not possible in lilypond (only relative coordinates!)

Page Layout and Metadata

Handling of Metadata and of Title, Author, Header, ... data

- MusicXML:
 - **Metadata** stored in **<identification>** tag (never displayed),
 - Title, author, header, etc. printed via **score-wide <credit> tags** – No attribute for type of information shown!
- LilyPond:
 - **Only metadata** explicitly entered (header block containing title, author, etc.)
 - Title, author, header, etc. **automatically generated** from metadata

To produce the same layout as MusicXML: need to extract metadata information from the `<credit>` elements – **Not Possible!!!**

Different coordinate systems

MusicXML places headers and other credit markup at **absolute coordinates on page**, not possible in lilypond (only relative coordinates!)

Musical Content vs. Graphical Representation

- LilyPond is a WYSIWYM application: You enter the music content, it formats it according to best-practices from centuries of music engraving (can be tweaked).
- MusicXML also mostly describes the musical content; adds layout information in extra sub-elements and attributes
- Some elements are tied to a horizontal position on the staff, e.g. dynamics:



- The position of the "f" in the music context can only be deduced from the graphical layout!

Workarounds in Some GUI Applications

Chant example provided by Recordare as MusicXML sample file



```

<direction placement="below">
  <direction-type>
    <words relative-y="69">|</words>
  </direction-type>
  <offset>-1</offset>
</direction>
  
```

- Divisio minima (short tick through the top-most staffline) faked by "|" text markup, appropriately shifted!!!!
- Can never be correctly imported!

Overview

- 1 What is MusicXML?
- 2 A MusicXML 2.0 Test Suite
- 3 MusicXML 2.0: Semantic Ambiguities
- 4 Sub-Optimal XML Design
- 5 Missing Features
- 6 Issues in the conversion from MusicXML to LilyPond
- 7 Conclusion and Acknowledgements**
 - Conclusion

Conclusion

- Finally a MusicXML test suite is freely available:

Homepage of the test suite

<http://kainhofer.com/musicxml/>

- Sample renderings available (created via `musicxml2ly` and LilyPond)
- MusicXML is a good industry standard for music notation exchange
- Several minor issues; discussed here for future implementors to know some problems / pitfalls
- Future versions of MusicXML will probably solve many of the mentioned problems

Acknowledgements

- LilyPond developers and community!
- MusicXML mailing list (in particular Michael Good, author of the MusicXML specification)

Conclusion

- Finally a MusicXML test suite is freely available:

Homepage of the test suite

<http://kainhofer.com/musicxml/>

- Sample renderings available (created via `musicxml2ly` and LilyPond)
- MusicXML is a good industry standard for music notation exchange
- Several minor issues; discussed here for future implementors to know some problems / pitfalls
- Future versions of MusicXML will probably solve many of the mentioned problems

Acknowledgements

- LilyPond developers and community!
- MusicXML mailing list (in particular Michael Good, author of the MusicXML specification)