jωL

Engineering a time-sensitive XSLT/SaxonJS application

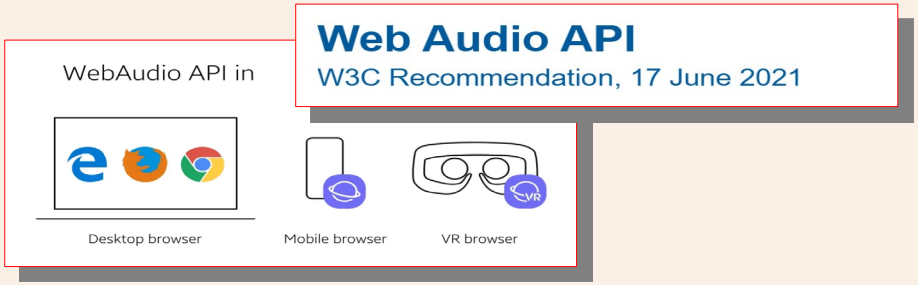
John Lumley

jωL

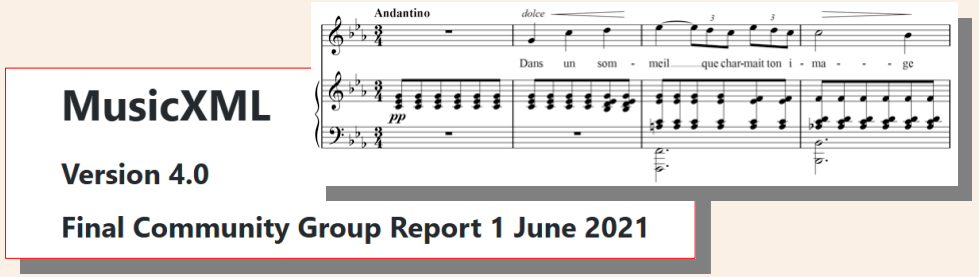
Michael Kay made a notable remark

SaxonJS works pretty well and can use Javascript libraries – could we put these three together and make something like an XSLT piano?



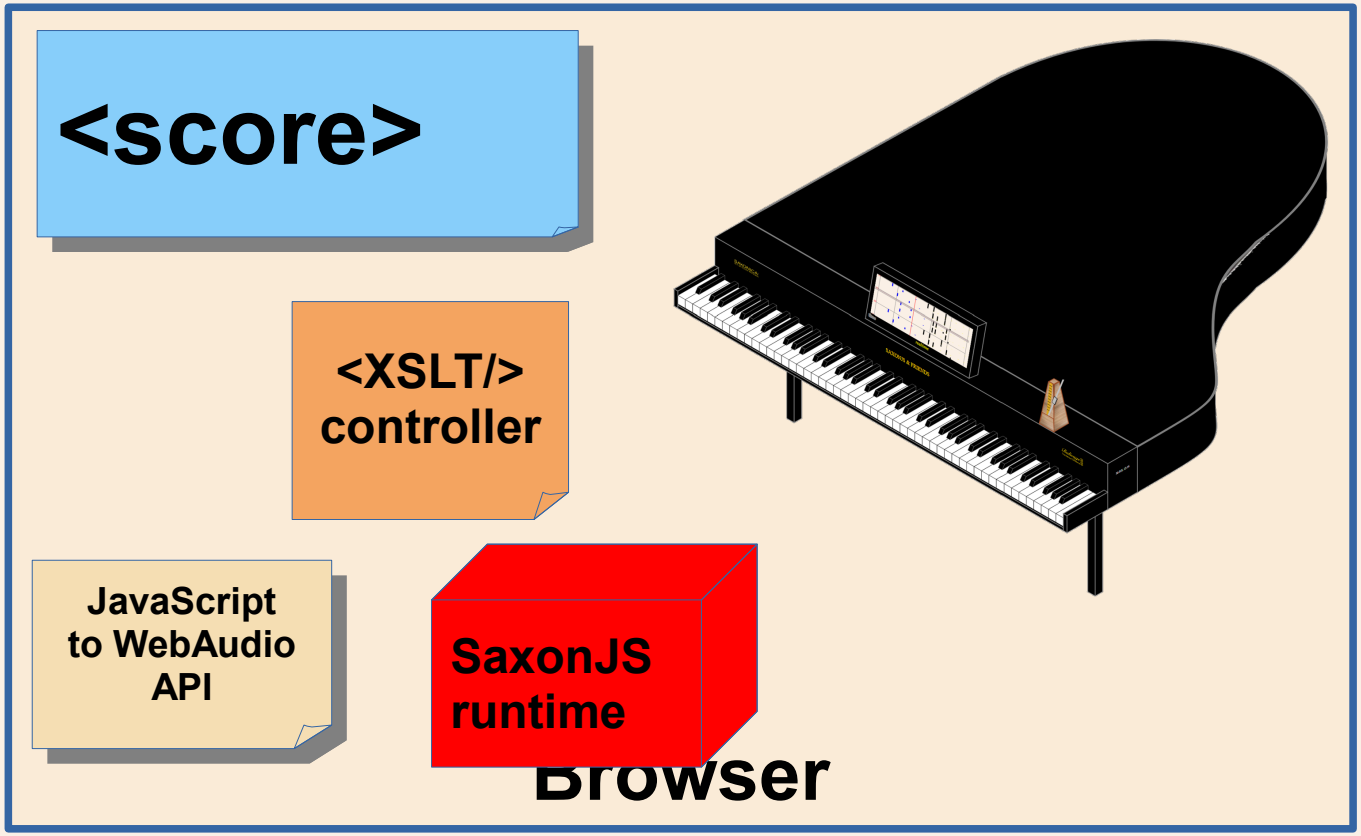


SAXONICA.COM
XSLT AND XQUERY PROCESSING
SaxonJS



jωL

General Architecture



jωL

Demo of final player-piano Mockingbird

Executed in Browser
0

jωL

WebAudio API

duration

volume

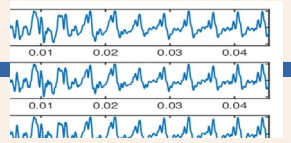
t_{start}

freq

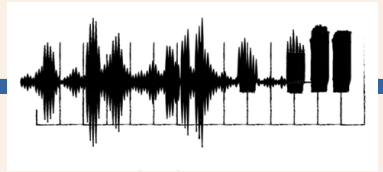
voice

playnote()

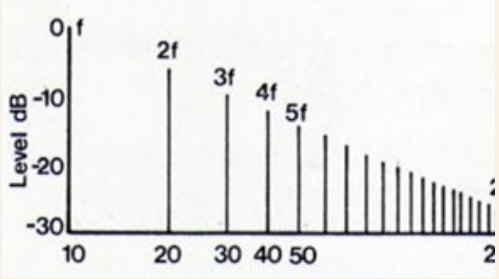
Oscillator
(**OscillatorNode**)



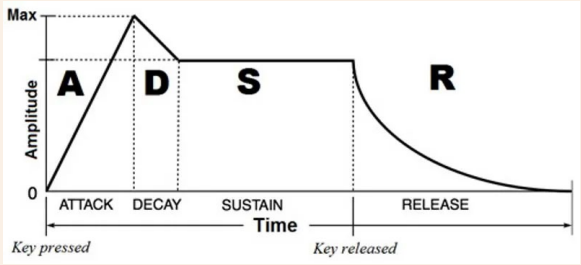
Gain
(**GainNode**)



Audio
Destination



Possibly varies by frequency



Suitable parameters for the given voice/instrument

Theoretically we could implement most of this connectivity in XSLT, but it's probably best left to ~400 lines of JavaScript!

jωL

JavaScript Interface

```
function playnote(frequency, type, duration, volume, delay) {
  let inst = activeInst.find (type)
  let record = {
    time: inst._ac.currentTime + delay,
    frequency: frequency,
    velocity: (volume == null ? 1: volume),
    duration: (duration == null ? Instrument.toneLength: duration),
    gainNode: null,
    oscillators: null,
  };
  inst._makeSound(record);
}
function lift() {/* Stop all active instruments */
  for (const key in activeInst.members) {
    let inst = activeInst.members[key];
    for (const r of inst._startSet) {
      r.gainNode && r.gainNode.disconnect();
      r.gainNode = null
    }
  }
}
```

APPROXIMATE
(to give sense of code type)
~450 lines JavaScript code
+ data tables.

```
_makeSound(record) {/* Setup & connect oscillator & gain */
    let timbre = record.timbre || this._timbre,
    starttime = record.time + InstrumentXtimeOffset,
    releasetime = starttime + record.duration;
    ...
    let g = ac.createGain();
    g.gain.value = 0;
    g.gain.setValueAtTime(0, starttime);
    ...
    g.gain.setTargetAtTime(record.velocity, decaystart, decayT);
    ...
    g.connect(this._out);
    // Hook up the main oscillator.
    let o = this.makeOscillator(timbre.wave, record.frequency);
    o.connect(g);
    o.start(starttime);
    o.stop(stoptime);
    ...
   this._startSet.push(record);
  }
}
```

jωL

Demo of an HTML ‘button’ piano

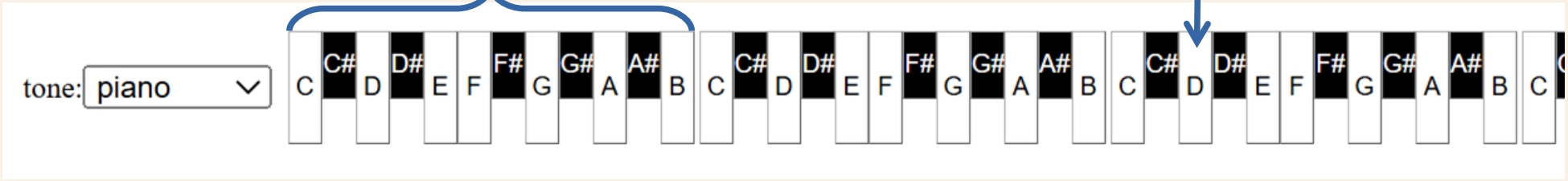


jωL

Just making sounds

<button/>...

<button class="key white D">D</button>




```
<xsl:template match="button[contains-token(@class, 'key')]" mode="ixsl:onmousedown">
  <xsl:variable name="note" select="tokenize(@class)[last()]" as="xs:string"/>
  <xsl:variable name="octave" select="../@octave" as="xs:integer"/>
  <xsl:variable name="of" select="$octave-frequencies($octave)"/>
  <xsl:sequence select="js:playnote($of * $base-frequencies-all($note),
                        f:getValue('tone'), 10)"/>
</xsl:template>

<xsl:template match="button[contains-token(@class, 'key')]" mode="ixsl:onmouseup">
  <xsl:sequence select="js:lift()"/>
</xsl:template>
```

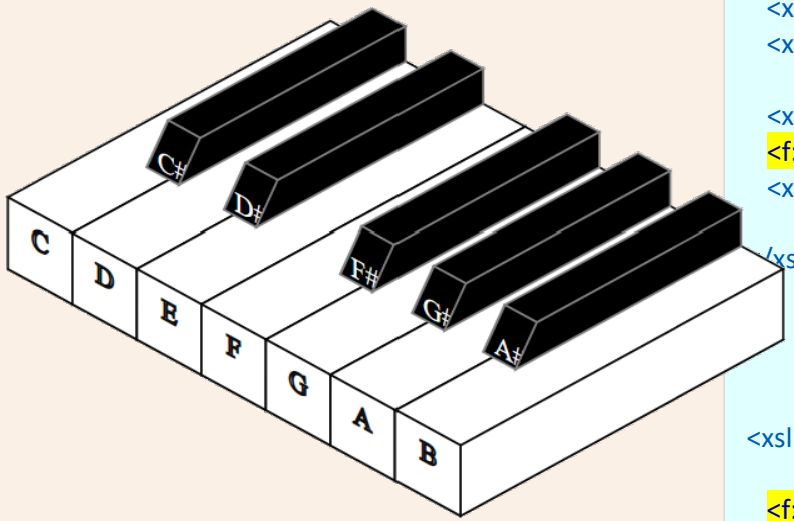
jωL

Demo of an SVG keyboard



jωL

A responsive SVG keyboard




```
<xsl:template match="svg:*[contains-token(@class, 'key')]"
              mode="ixsl:onmousedown">
  <xsl:variable name="note" select="@note" as="xs:string"/>
  <xsl:variable name="octave" select="(ancestor::svg:g/@octave)[1]"
                as="xs:integer"/>
  <xsl:variable name="of" select="$octave-frequencies($octave)"/>
  <f:depress key="."/>
  <xsl:sequence select="js:playnote($of * $base-frequencies-all($note),
                                    f:getValue('tone'), 10)"/>
</xsl:template>


<xsl:template match="svg:*[contains-token(@class, 'key')]"
              mode="ixsl:onmouseup">
  <f:lift key="."/>
  <xsl:sequence select="js:lift()"/>
</xsl:template>
```

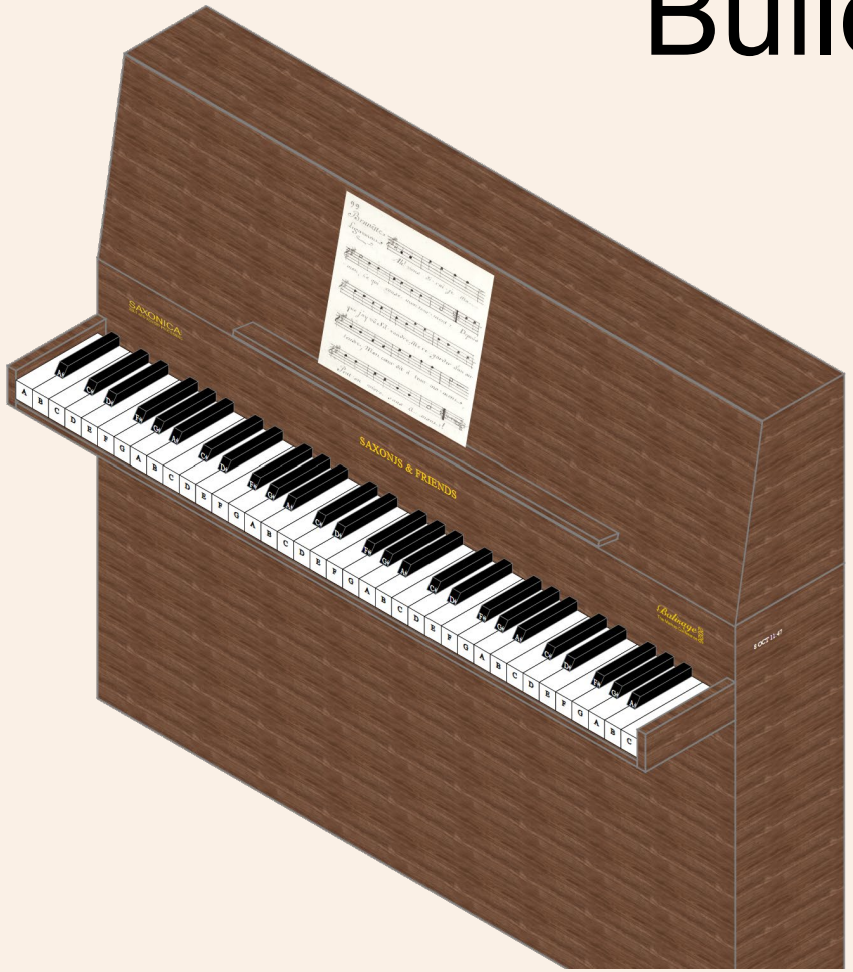
jωL

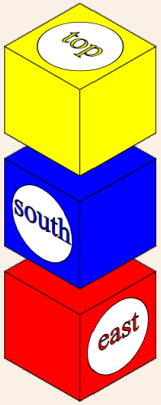
Demo of an SVG upright piano



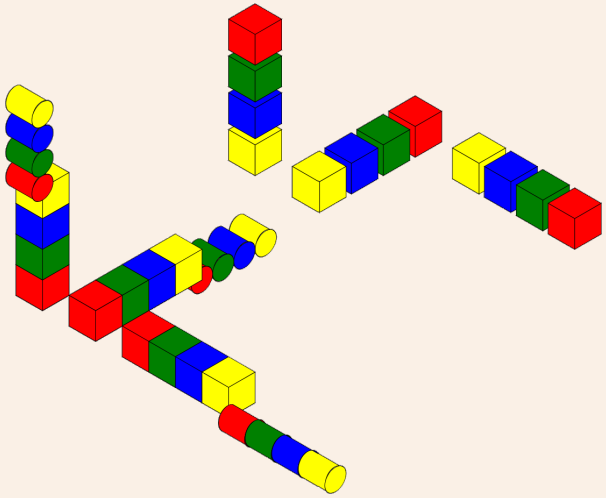
jωL

Building a piano





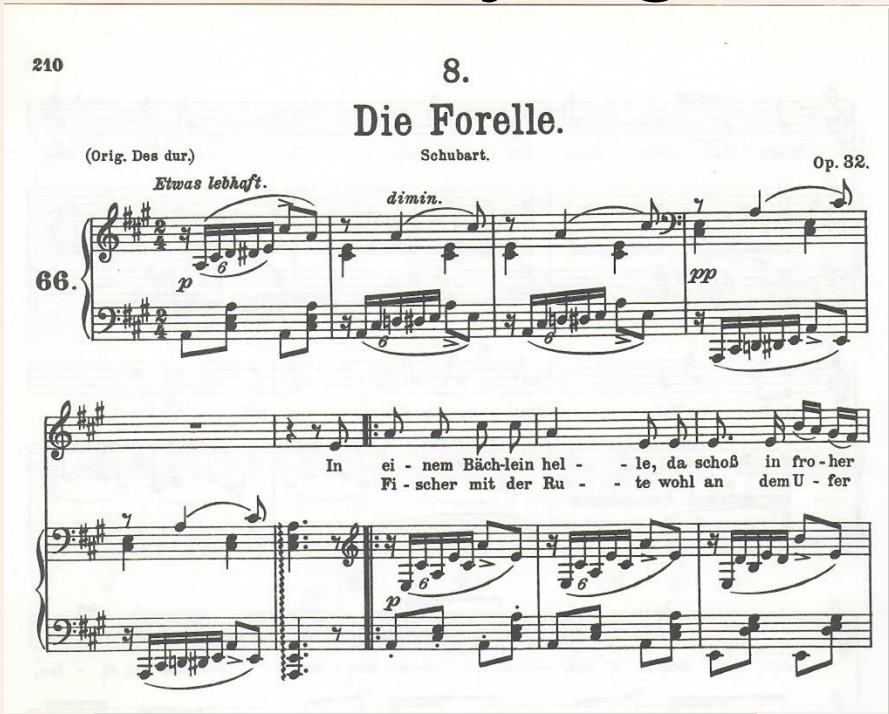
Parametrised
blocks/cylinders,
with face content
→Isometric projection



Relative placement
directives

jωL

Playing a Score – MusicXML



Design focus on describing printed scores

```
<score-partwise>
 <work>
  <work-number>The Trout</work-number>
  <work-title>Die Forelle</work-title>
 </work>
 <identification>
  <creator type="composer">Franz Schubert</creator>
  ...
 </identification>
 <part-list>
  <score-part id="P1">
   <part-name>piano</part-name>
  </score-part>
  <score-part id="P2">
   <part-name>piano</part-name>
  </score-part>
  <score-part id="P3">
   <part-name>chorus</part-name>
  </score-part>
 </part-list>
 <part id="P1">...
 <part id="P2">...
 <part id="P3">...
</score-partwise>
```

But contains (somewhere) all information needed

(at least mechanically – emotional play needs different approaches)

jωL

MusicXML structure - ii

A <part> is some general information (key, timing quanta etc.) & a sequence of <measure>s (bars) containing (mostly) <note>s

```
<part id="P1">
  <measure number="1">
    <direction placement="above">...
      <sound tempo="80.00"/>
    </direction>
    <attributes>
      <divisions>120</divisions>
      <key>
        <fifths>3</fifths>
        <mode>major</mode>
      </key>
      <time>
        <beats>2</beats>
        <beat-type>4</beat-type>
      </time>
    </attributes>
    ...
    <note>...
    ...
  </measure>
```

```
<note>
  <pitch>
    <step>C</step>
    <alter>1</alter>
    <octave>4</octave>
  </pitch>
  <duration>20</duration>
  <voice>1</voice>
  <type>eighth</type>
  <time-modification>
    <actual-notes>6</actual-notes>
    <normal-notes>2</normal-notes>
  </time-modification>
  <beam number="1">continue</beam>
</note>
```

Pitch (Middle C#)

Length (1/6th of a beat)

The start of a <note> is determined (usually) by the end sum(preceding-sibling::note/duration)

Tempo – 80 beats per minute

Beat time quanta, e.g. 60 = ½ beat

Measure = 2 (crotchet) beats

jωL

Extracting the notes/keys

MusicXML

Necessary ‘notes’+

Minimal data: arrays/maps



Ornamentations
(trill, arpeggio, turn)
expanded



Scheduled measures/notes

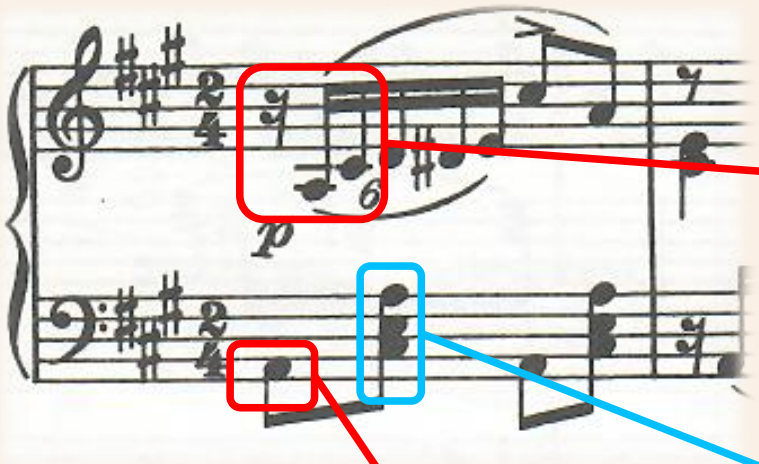


Key bindings



jωL

Necessary ‘music’ – by part

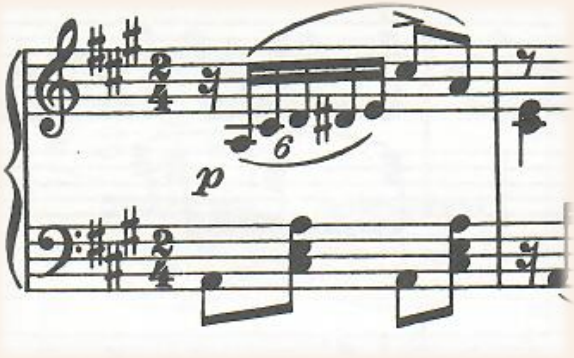


```
<part voice="piano" id="P1" bar-length="2" beat-type="4">
  <measure number="1">
    ...
    <!--<dynamics loudness="p"/>-->
    <rest duration="0.167"/>
    <note duration="0.167" volume="0.22"
          note="A" octave="3" freq="220"/>
    <note duration="0.167" volume="0.22"
          note="C#" octave="4" freq="277.12"/>
    ...
  </measure>
  ...
</part>
```

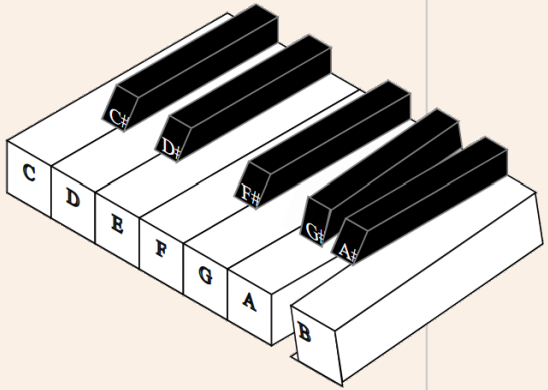
```
<part voice="piano" id="P2" bar-length="2" beat-type="4">
  <measure number="1">
    <note duration="0.5" volume="0.22" note="A" octave="2" freq="110"/>
    <note duration="0.5" volume="0.22" note="C#" octave="3" freq="138.56"/>
    <note duration="0.5" volume="0.22" chord="" note="E" octave="3" freq="164.8"/>
    <note duration="0.5" volume="0.22" chord="" note="A" octave="3" freq="220"/>
    <note duration="0.5" volume="0.22" note="A" octave="2" freq="110"/>
    ...
  </measure>
  ...
</part>
```

jωL

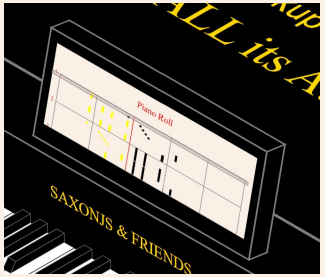
Scheduling priorities

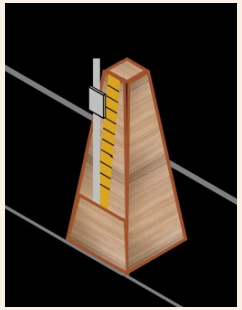


1)All notes start as close to beat position as possible
(JavaScript)



2)Key press/lift in correct order and
at approximately correct time
(SVG @class, SaxonJS or JavaScript)



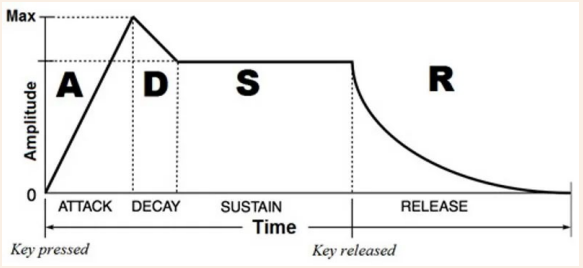


3)Metronome swing and piano-roll scroll
(SVG animate-transform)

jωL

Scheduling Mechanisms







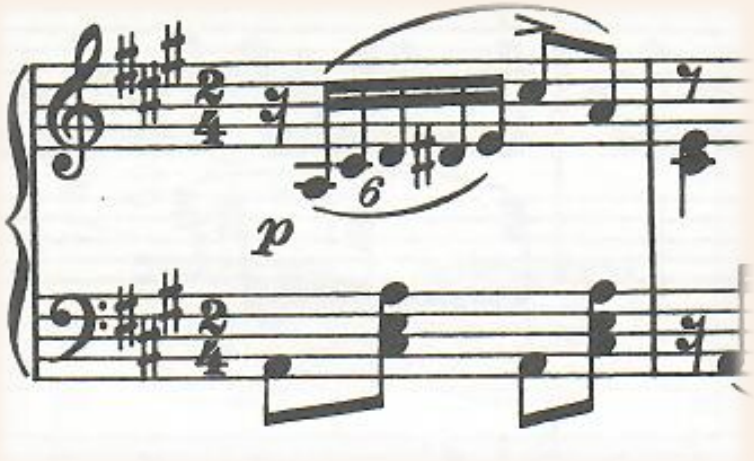
```
setTimeout(functionName, delay_in_ms, arg0, arg1...)
```

SAXONICA.COM
XSLT AND XQUERY PROCESSING
SaxonJS

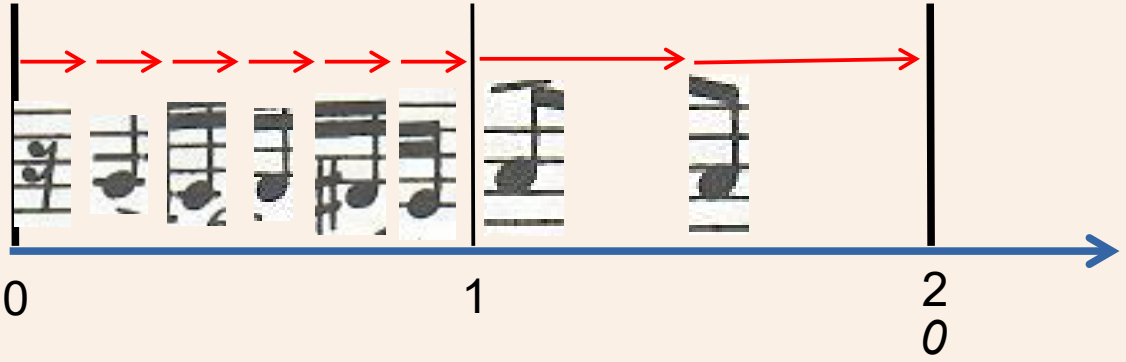
```
<ixsl:schedule-action wait="$duration_in_ms">
    <xsl:call-template name="process"/>
</ixsl:schedule-action>
```

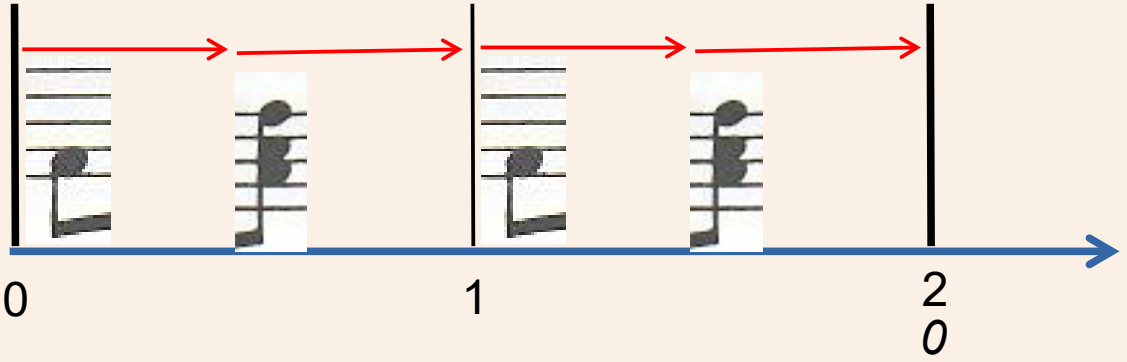
jωL

Event scheduling - differential



12 single-voice events
Invoking notes and scheduling events is *not instantaneous*





jωL

Note-by-note scheduling

```
<xsl:template name="f:sequence">
  <xsl:param name="notes" as="element()*"/>
  <xsl:if test="exists($notes)">
    <xsl:variable name="note" select="head($notes)"/>
    <xsl:variable name="remainder" select="tail($notes)"/>
    <xsl:variable name="duration" select="60 * xs:double($note/@duration) div $tempo"/>
    <xsl:choose>
      <xsl:when test="exists(head($remainder)/chord)">
        <f:sequence notes="$remainder"/>
      </xsl:when>
      <xsl:otherwise>
        <ixsl:schedule-action wait="$duration * 1000">
          <f:sequence notes="$remainder"/>
        </ixsl:schedule-action>
      </xsl:otherwise>
    </xsl:choose>
    <xsl:for-each select="$note[@freq]">
      <xsl:sequence select="js:playnote(@freq, $tone, $duration, volume * $volume)"/>
      <f:pressKey key="f:getKey(@note, @octave)" duration="$duration * 1000"/>
    </xsl:for-each>
  </xsl:if>
</xsl:template>
```

Process next note

Process next note
$duration later

jωL

Demo of an SVG player grand piano – separate parts



jωL

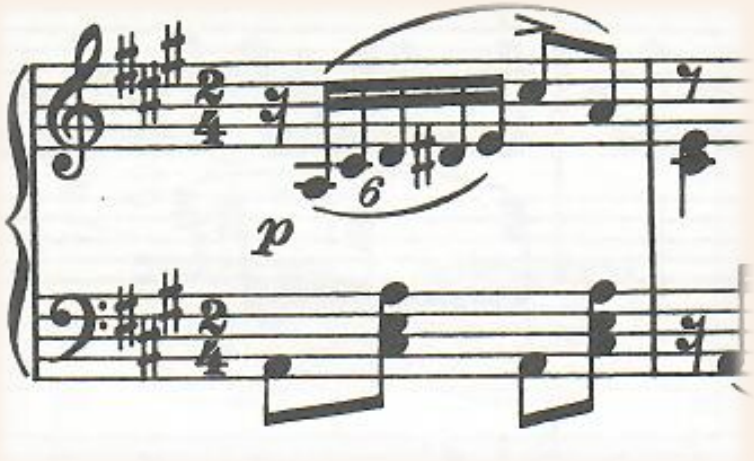
Necessary ‘music’ – by measure

```
<measure number="1">
  <part id="P1" voice="piano">
    <rest duration="0.167" start="0"/>
    <note duration="0.167" volume="0.22" start="0.167" note="A" octave="3" freq="220"/>
    <note duration="0.167" volume="0.22" start="0.333" note="C#" octave="4" freq="277.12"/>
    ...
  </part>
  <part id="P2" voice="piano">
    <note duration="0.5" volume="0.22" start="0" note="A" octave="2" freq="110"/>
    <note duration="0.5" volume="0.22" start="0.5" note="C#" octave="3" freq="138.56"/>
    ...
  </part>
  <part id="P3" voice="chorus">
    <rest duration="2.0" start="0"/>
  </part>
</measure>
```

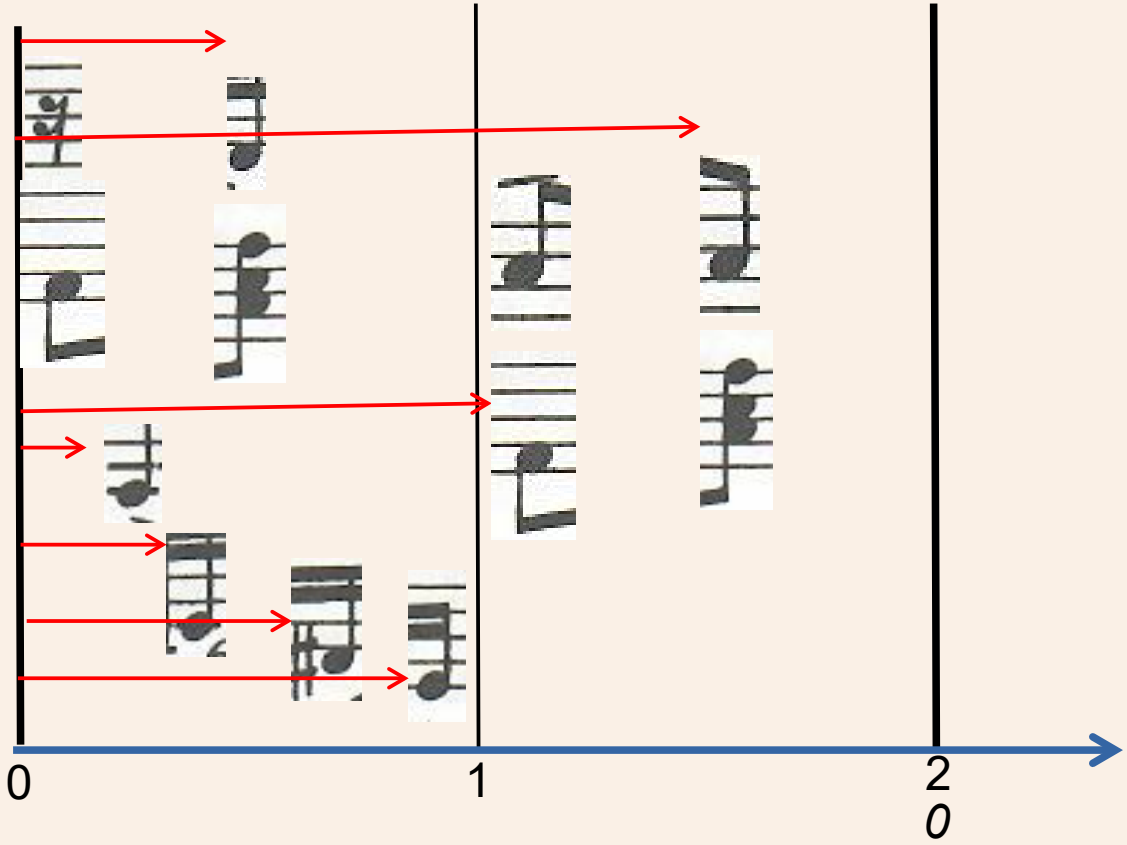
...

jωL

Event scheduling – measure relative



7 multi-voice events



jωL

Compiled ‘music’ – by measure

```
<measure number="1">
  <note start="0" duration="0.5" volume="0.22" freq="110" voice="piano"/>
  <note start="0.167" duration="0.167" volume="0.22" freq="220" voice="piano"/>
  <note start="0.3337" duration="0.167" volume="0.22" freq="277.12" voice="piano"/>
  <note start="0.5" duration="0.167" volume="0.22" freq="293.6" voice="piano"/>
  <note start="0.5" duration="0.5" volume="0.22" freq="138.56" voice="piano"/>
 ...
  <key-down beat="0"> <key ki="20"/> </key-down>
  <key-down beat="0.167"> <key ki="32"/> </key-down>
  ...
  <key-down beat="1.5"> <key ki="44"/><key ki="34"/>...</key-down>
  <key-up beat="0.304"> <key ki="32"/> </key-up>
  <key-up beat="0.47"> <key ki="20"/> </key-up>
  ...
  <key-up beat="1.97"> <key ki="44"/><key ki="34"/>...</key-up>
</measure>
```

...

jωL

Scheduling by measure

```
<xsl:template name="playMeasure">
  <xsl:param name="measure" as="element()"/>
  <xsl:param name="beat" as="xs:double"/>
  <xsl:param name="volume" tunnel="true"/>
  <xsl:variable name="notes" as="element()*" select="$measure/note"/>

  <xsl:for-each select="$measure/key-down">
    <xsl:variable name="keys" select="array {key/xs:integer(@ki)}"/>
    <xsl:sequence select="js:keysDown(980 * $beat * xs:double(@beat), $keys)"/>
  </xsl:for-each>
  <xsl:for-each select="$measure/key-up">
    <xsl:variable name="keys" select="array {key/xs:integer(@ki)}"/>
    <xsl:sequence select="js:keysUp(980 * $beat * xs:double(@beat), $keys)"/>
  </xsl:for-each>

  <xsl:for-each select="$notes[@freq]">
    <xsl:variable name="duration" select="$beat * @duration"/>
    <xsl:variable name="tone" select="@voice" as="xs:string"/>
    <xsl:variable name="start" select="$beat *  @start"/>
    <xsl:sequence select="js:playNote(xs:double(@freq), $tone, $noteLength,
        xs:double(@volume) * $volume, $start)"/>
  </xsl:for-each>
</xsl:template>
```

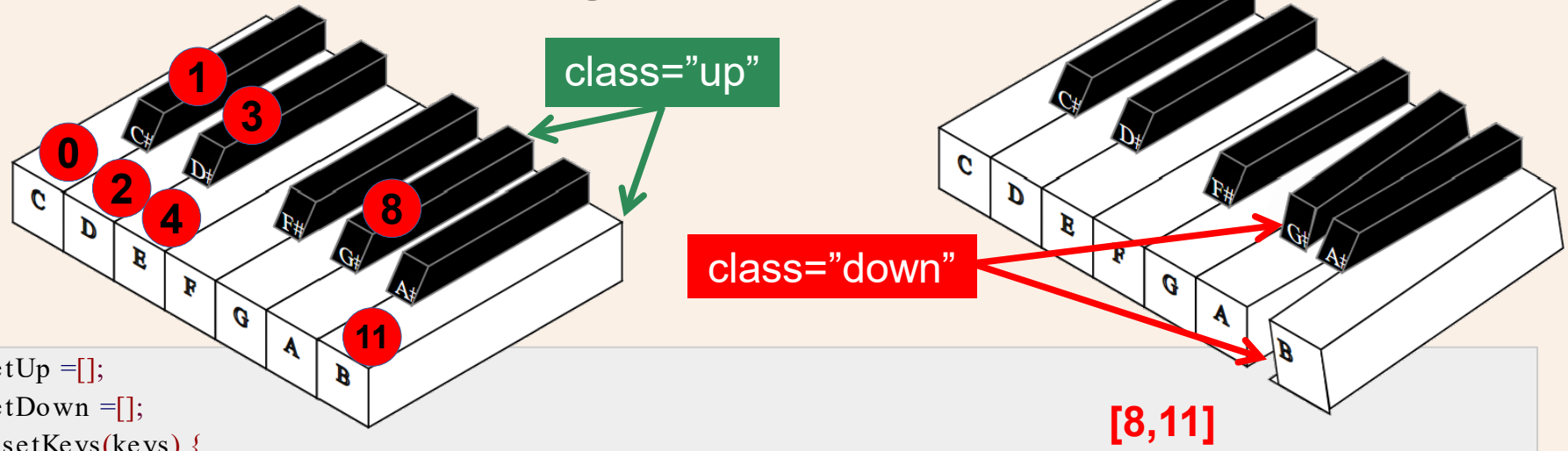
jωL

Demo of an SVG player grand piano – by measure



jωL

Tickling the Ivories




```
var keySetUp =[];
var keySetDown =[];
function setKeys(keys) {
  keySetUp =[];
  keySetDown =[];
  for (const k of keys) {
    const u = k.getElementsByClassName('up').item(0);
    const d = k.getElementsByClassName('down').item(0);
    keySetUp.push(u);
    keySetDown.push(d);
  }
}
function keysDown(time, keys) {
```

jωL

Minimal compilation via JSON

```
[{"notes":[
  {"start":0,"duration":0.5,"volume":0.22,
   "freq":110,"voice":"piano"},
  {"start":0.166,"duration":0.167,"volume":0.22,
   "freq":220,"voice":"piano"},
  {"start":0.333,"duration":0.167,"volume":0.22,
   "freq":277.12,"voice":"piano"},
  {"start":0.5,"duration":0.167,"volume":0.22,
   "freq":293.6,"voice":"piano"},
  ...],
 "keyDown":[
  {"beat":0,"keys":[20]},
  {"beat":0.166,"keys":[32]},
  {"beat":0.333,"keys":[46]},
  {"beat":0.5,"keys":[40,34,29,32]},
  ...],
 ...},
...]
```

```
<xsl:sequence select="js:playMeasureJSON($measure/@number - 1,
                      $beat, $volume)"/>
```

```
var currentScoreJSON = null;
function setCurrentScoreJSON(score) {
  currentScoreJSON = JSON.parse(score);
}
function playMeasureJSON(no, beat, volume) {
  const measure = currentScoreJSON[no]
  for (const n of measure.notes) {
    playNote(n.freq, n.voice, n.duration * beat,
             n.volume * volume, n.start * beat);
  }
  for (const kd of measure.keyDown) {
    keysDown(980 * beat * kd.beat, kd.keys)
  }
  for (const kd of measure.keyUp) {
    keysUp(980 * beat * kd.beat, kd.keys)
  }
}
```

jωL

Demo of player grand piano & pipe organ – by ‘fully JSON compiled’ score

Executed in Browser
5

*j*ωL

Conclusion

- Working at high level in XSLT can get the ‘data’ architecture established
- Successive passes of ‘compilation’ towards minimal sufficient data
- Migration of such structures to the JavaScript/browser level