Web Applications for Automatic Audio-to-Score Synchronization with Iterative Refinement: Appendix A

Adriano Baratè, Goffredo Haus
Luca A. Ludovico, Giorgio Presti
Laboratory of Music Informatics
Department of Computer Science
University of Milan
{first.last}@unimi.it

Stefano Di Bisceglie
Alessandro Minoli
Department of Computer Science
University of Milan
{first.last}@studenti.unimi.it

Davide A. Mauro
Department of Computer
and Information Technology
Marshall University
maurod@marshall.edu

1. SCORE-INFORMED BEAT TRACKING

Excerpts of the implementation for the score-informed beat tracking algorithm, including the computation of chroma vectors from the score, and the DTW computation.

1.0.1 *chromaFromScore()*

```
var chords = xmlDoc.getElementsByTagName("chord");
transpose = parseInt(document.getElementById("transpose").value);
for(var i = 0;i < chords.length;i++){ //for every chord
    var chord = chords[i];
    var timing_start = timings[chord.getAttribute("event_ref")];
    var dur = getElementDuration(chord);
    var timing_end = timing_start + (quarter_vtu * dur * 4);
    var pitches = chord.getElementsByTagName("pitch");
    //for every note in the chord
    for(var j = 0; j < pitches.length;j++){
        var pitch_obj = pitches[j];
        var chroma = 0;
        var chroma = getChromaFromPitch(pitch_obj,transpose);
        if(chroma == -1)
            continue;
  var cv_start = insertIntoChromasVtus(chromas_vtus[chroma],timing_start,true);
  var cv_end = insertIntoChromasVtus(chromas_vtus[chroma],timing_end,false);
  var dist = deleteBetween(chromas_vtus[chroma],cv_start,cv_end);
  cv_end = cv_end - dist;
  if(chromas_vtus[chroma][cv_start-1][1] == true){
            deleteFrom(chromas_vtus[chroma],cv_start);
            cv_end--;
  }
  if(chromas_vtus[chroma][cv_end+1][1] == false){
    deleteFrom(chromas_vtus[chroma],cv_end);
  }
    }
}
for(var chroma = 0;chroma < 12;chroma++){
    var j = 1;
    while(j < chromas_vtus[chroma].length-1){
        var flag1 = chromas_vtus[chroma][j];
  j++;
  var flag2 = chromas_vtus[chroma][j];
  insertIntoChvecsScore(flag1[0],flag2[0],chroma);
  j++;
    }
}
```

1.0.2 *calcTrackIndexing()*

```
//compute new track indexing and add it to xmlDoc
var track_indexing = xmlDoc.createElement("track_indexing");
track.appendChild(track_indexing);
track_indexing.setAttribute("timing_type","seconds");
var spine = xmlDoc.getElementsByTagName("spine")[0];
var curTiming = 0;
for(var i = 0;i < spine.getElementsByTagName("event").length;i++){
    //for every event
    var event = spine.getElementsByTagName("event")[i];
    curTiming = curTiming + parseInt(event.getAttribute("timing"));
    var seconds = vtuToSeconds(curTiming);
    var track_event = xmlDoc.createElement("track_event");
    track_event.setAttribute("event_ref",event.getAttribute("id"));
    track_event.setAttribute("start_time",seconds);
    track_indexing.appendChild(track_event);
}
```

1.0.3 *getDistance()*

```
matrix = [];
path_matrix = [];
    for ( var i = 0; i < ser1.length; i++ ){
    matrix[i] = [];
    path_matrix[i] = [];
    for ( var j = 0; j < ser2.length; j++ ){
        var c = distFunc(ser1[i], ser2[j]);
        if(i == 0 && j == 0){
            matrix[i][j] = c;
    path_matrix[i][j] = -1;
  }else if(i > 0 && j == 0){
            matrix[i][j] = matrix[i-1][j] + wh*c;
            path_matrix[i][j] = 0;
        }else if(j > 0 && i == 0){
            matrix[i][j] = matrix[i][j-1] + wv*c;
            path_matrix[i][j] = 1;
        }else{
            var h = matrix[i-1][j] + wh*c;
            var v = matrix[i][j-1] + wv*c;
            var d = matrix[i-1][j-1] + wd*c;
            var min = Math.min(v,d,h);
```

```
22              if(min == d){
23                  matrix[i][j] = d;
24                  path_matrix[i][j] = 2;
25              }else if(min == h){
26                  matrix[i][j] = h;
27                  path_matrix[i][j] = 0;
28              }else if(min == v){
29            matrix[i][j] = v;
30          path_matrix[i][j] = 1;
31      }
32    }
33      }
34  }
35  return matrix[ser1.length-1][ser2.length-1];
```

2. ONSET DETECTION FUNCTIONS

Concerning the reduction of a signal to an ODF, there are mainly two categories: energy-based approaches and phase-based ones. Please note that ODFs detect the physical onsets of a signal. The more the attacks of the notes are slow and/or soft, the more these offsets tend to deviate temporally from the ones perceived during the listening.

Energy-based approaches take their cue from the following observation: the occurrence of onsets in the signal is usually accompanied by sudden increments in energy. These changes tend to appear in the spectrum as broadband events. In addition, in the portions of the signal not affected by transients, energy is usually concentrated in the low frequencies, so these changes should be more easily detectable at high frequencies. The ODF called *High-Frequency Content* (HFC) [1] exploits this observation and, for each frame, weighs each frequency bin proportionally to its frequency. As a drawback, it does not consider the temporal evolution of the signal, thus the value of the function at each instant depends exclusively on the current frame.

Other energy-based ODFs consider the spectrum variation between adjacent frames. In particular, they measure the distances between the magnitude of two consecutive frames. For example, by choosing L1-norm as the distance metrics, we obtain the *Spectral Difference* (SD), while L2-norm defines the *Spectral Flux* (SF) [2]. Another approach called *Percussive feature detection* (or, simply, percussive) is based on an algorithm similar to SF but calculated on a logarithmic amplitude scale [3].

In general, energy-based ODFs can be implemented in a simple and efficient way. They give excellent results in the presence of percussive onsets, namely notes with a sharp attack, such as the ones produced by drums or piano. Energy-based ODFs performances get worse in the case of non-percussive signals and when the transients begin to overlap, as in complex signals.

Phase-based approaches rely on the idea that, during a transient, the phase of the frequency components changes significantly with respect to the increase expected considering previous time instants. A measure of the temporal instability of the signal phase can be constructed by quantifying the derivative of the phase over time for each frequency component and then adding these quantities to obtain a single time-varying signal [4]. Unlike energy-based ODFs, phase-based ones are also able to detect onsets with loosely defined attacks. However, they are very sensitive to noise, especially for those signal components carrying very low energy.

In recent years, in order to try to solve the open problem of onset detection, machine learning techniques have been added to the more classic techniques of signal analysis and processing. For example, neural networks can be successfully applied [5].

3. FURTHER TESTS

Figures 1 to 4 show the deviations of the original timings and the adjusted ones with respect to the ground truth for 4 of the test tracks.

Then, the solution was tested on excerpts from the *Onset_Leveau* dataset. Compared to the scenario of IEEE 1599 documents, which also contain a *Logic* level with the symbolic description of the events, in this case the score is not available. For this reason, we prepared IEEE 1599 documents which in the *Audio* layer already contained ground-truth timings. In an ideal case, the algorithm should not change them, as they already correspond to the correct time position of the onsets. The best results will be those where $RMSE$ and MAE are closest to 0.

The results achieved on the dataset tracks are shown in Table 1. The tracks in the upper area perceptively present sharp note attacks, and those in the lower area smooth attacks.

By weighing each track by its number of onsets, the means and standard deviations were calculated, grouping results by track family and by the characteristics of their onsets. These measures are respectively denoted by symbols μ_{RMSE}, μ_{MAE}, σ_{RMSE} and σ_{MAE}, and reported in Table 2. Figures 5 to 8 show their trend as the size of the neighborhood changes; the bars represent the standard deviation of the timings from their mean. Finally, Figures 9 and 10 show the deviations of adjusted timings with respect to the ground truth for some tracks in Table 1.

4. IEEE 1599 STRUCTURE

The standard IEEE layered structure is depicted in Fig. 11

5. REFERENCES

[1] P. Masri, "Computer modelling of sound for transformation and synthesis of musical signals," Ph.D. dissertation, University of Bristol, 1996.

[2] C. Duxbury, M. Sandler, and M. Davies, "A hybrid approach to musical note onset detection," in *Proc. Digital Audio Effects Conf.(DAFX,'02)*. sn, 2002, pp. 33–38.

[3] D. Barry, D. Fitzgerald, E. Coyle, and B. Lawlor, "Drum source separation using percussive feature detection and spectral modulation," *IEE Irish Signals and Systems Conference (ISSC)*, pp. 13–17, 2005.

[4] P. Brossier, J. P. Bello, and M. D. Plumbley, "Real-time temporal segmentation of note objects in music signals," in *Proceedings of ICMC 2004, the 30th Annual International Computer Music Conference*. University of Surrey, 2004.

WAV file	duration	# onsets	type	narrow neighborhood		large neighborhood	
				$RMSE$	MAE	$RMSE$	MAE
guitar3	15 s	56	P	8 ms	7 ms	8 ms	7 ms
jazz2	14 s	47	CM	20 ms	15 ms	21 ms	16 ms
piano1	15 s	19	P	12 ms	10 ms	12 ms	10 ms
pop1	15 s	27	CM	13 ms	11 ms	13 ms	11 ms
rock1	15 s	59	CM	12 ms	10 ms	24 ms	15 ms
synthbass	7 s	23	M	6 ms	5 ms	21 ms	9 ms
techno2	6 s	56	CM	17 ms	15 ms	17 ms	15 ms
cello1	14 s	61	P	23 ms	17 ms	43 ms	32 ms
clarinet1	30 s	35	M	22 ms	17 ms	86 ms	80 ms
classic2	20 s	38	CM	23 ms	19 ms	40 ms	29 ms
classic3	14 s	4	CM	28 ms	28 ms	28 ms	28 ms
distguit1	6 s	19	P	10 ms	7 ms	10 ms	7 ms
guitar2	15 s	36	P	18 ms	12 ms	34 ms	22 ms
jazz3	11 s	53	CM	21 ms	16 ms	35 ms	24 ms
sax1	12 s	9	M	28 ms	23 ms	64 ms	55 ms
trumpet1	14 s	56	M	21 ms	19 ms	21 ms	19 ms
violin2	15 s	73	P	20 ms	14 ms	41 ms	29 ms

Table 1: Results achived on *Onset_Leveau*.

category	narrow neighborhood				large neighborhood			
	μ_{RMSE}	σ_{RMSE}	μ_{MAE}	σ_{MAE}	μ_{RMSE}	σ_{RMSE}	μ_{MAE}	σ_{MAE}
M	19 ms	3.91 ms	16 ms	3.52 ms	43 ms	9.54 ms	37 ms	9.44 ms
P	17 ms	2.15 ms	12 ms	1.52 ms	29 ms	4.71 ms	21 ms	3.36 ms
CM	18 ms	1.27 ms	14 ms	0.99 ms	25 ms	2.22 ms	19 ms	1.46 ms
sharp onsets	13 ms	1.08 ms	11 ms	0.90 ms	17 ms	1.43 ms	12 ms	1.02 ms
smooth onsets	21 ms	1.30 ms	17 ms	0.99 ms	41 ms	2.84 ms	32 ms	2.32 ms

Table 2: Aggregated results achived on *Onset_Leveau*.

[5] J. Schlüter and S. Böck, "Improved musical onset detection with convolutional neural networks," in *2014 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE, 2014, pp. 6979–6983.

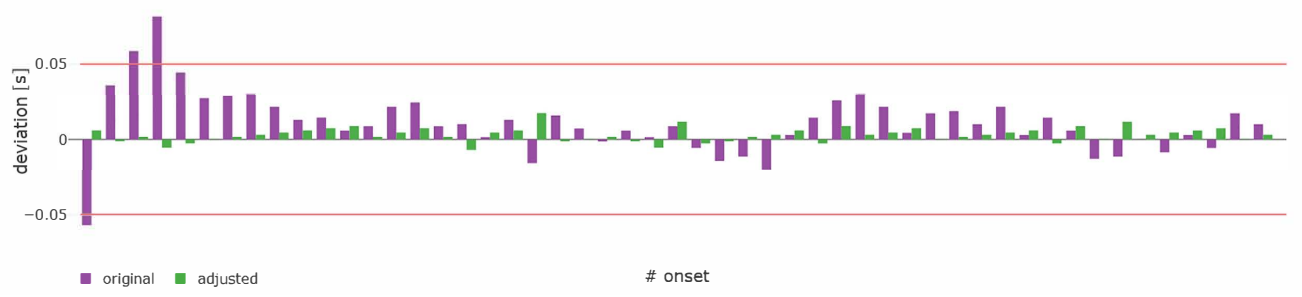


Figure 1: Results for piano1.mp3.

Figure 2: Results for violin1.mp3.

Figure 3: Results for string_4et.mp3.

Figure 4: Results for jazz_4et.mp3.

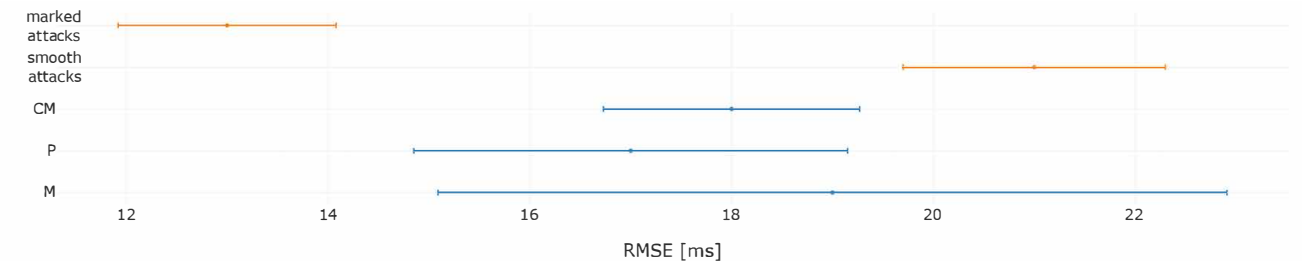


Figure 5: *RMSE* of the adjustments with a narrow neighborhood.

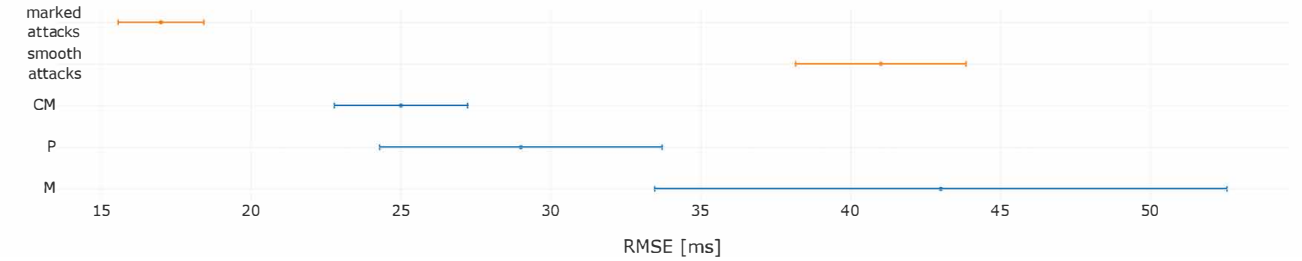


Figure 6: *RMSE* of the adjustments with a large neighborhood.

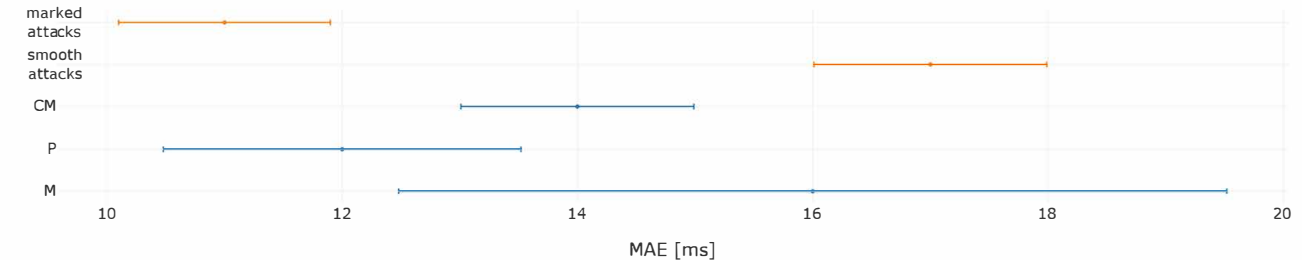


Figure 7: *MAE* of the adjustments with a narrow neighborhood.

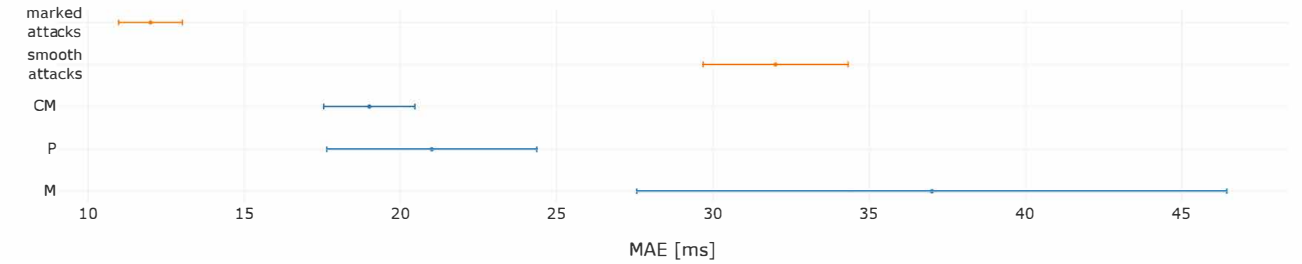
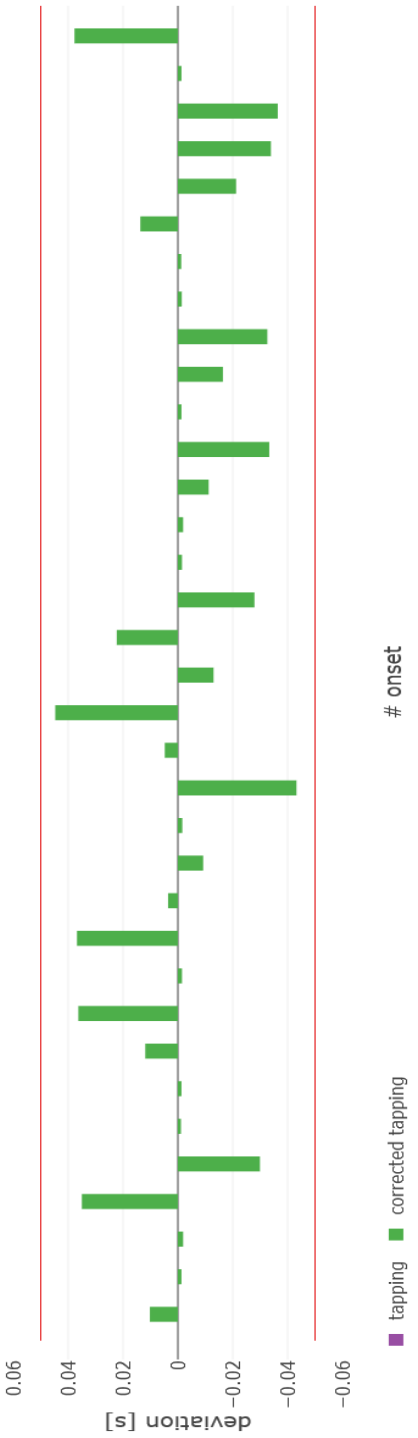
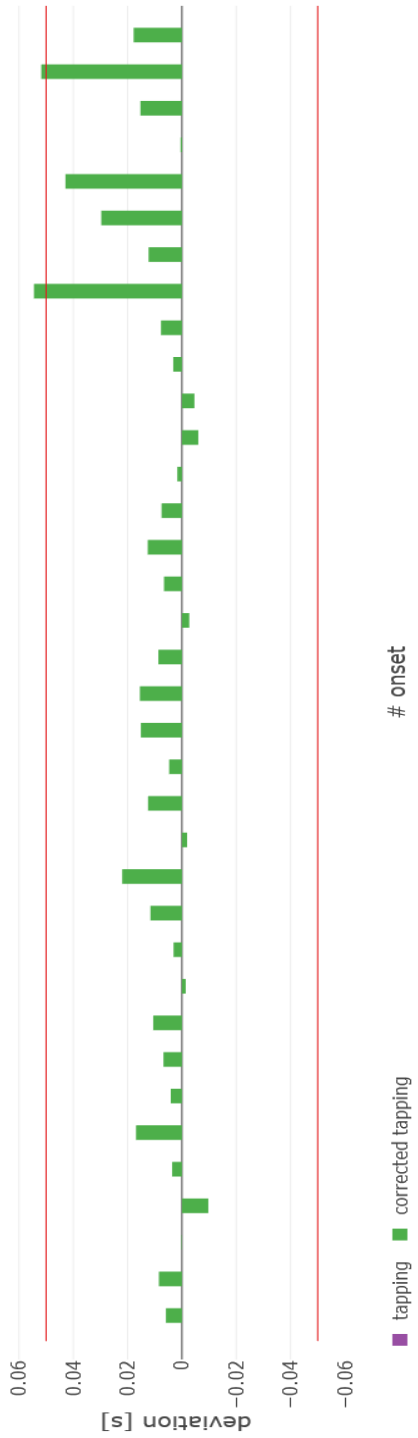


Figure 8: *MAE* of the adjustments with a large neighborhood.



clarinet1.wav



guitar2.wav

Figure 9


deviation [s]
0.06
0.04
0.02
0
−0.02
−0.04
−0.06
onset
tapping
corrected tapping
classic2.wav
pop1.wav

Figure 10

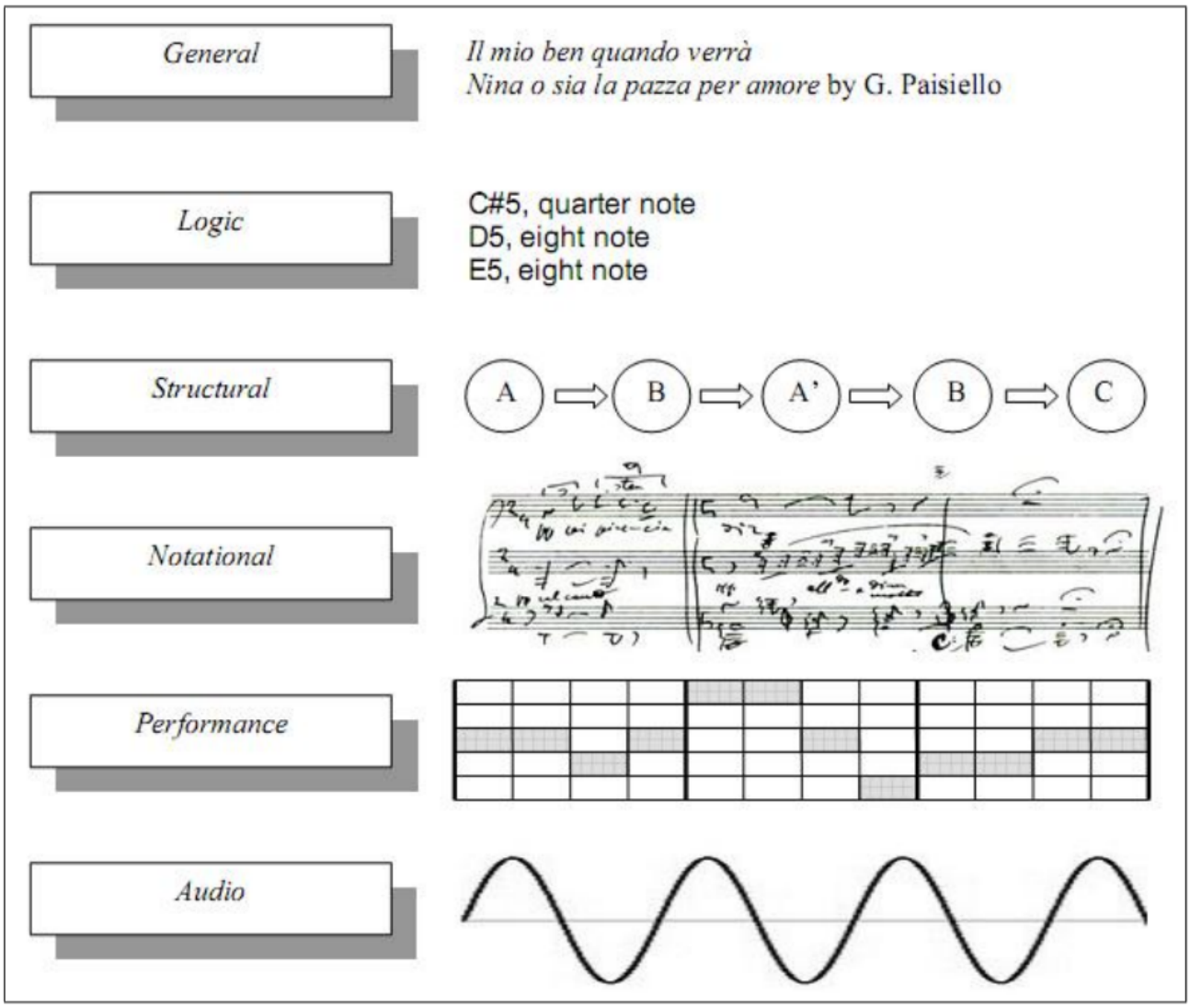


Figure 11: IEEE1599 Layers