Multimedia Representation of Music Pieces Encoded in Symbolic Format: an Approach Based on Csound, MPEG-4, SuperCollider, CML, Chuck and IEEE 1599

Elisa Russo
Laboratorio di Informatica Musicale (LIM)
Dipartimento di Informatica e Comunicazione (DIcon)
Università degli Studi di Milano
Via Comelico 39/41 – I- 20135, Milano, Italia
russo@dico.unimi.it

Abstract— The IEEE 1599 standard offers new potentialities to the most common and effective synthesis systems for the manipulation of timbres. This articles describes how sounds produced in a particular synthesis system can be managed in a complete and intuitive environment thanks by interacting with files designed according to the IEEE 1599 standard and with their contents. It is shown how easy it is to obtain the notes expressed in the syntax of a particular synthesis language starting from IEEE 1599 symbolic information. Therefore, it is possible to let IEEE 1599 files sound with any synthesis algorithm.

Index Terms—IEEE 1599, XML, Computer Music, Sound Synthesis, Symbolic Representation

I. INTRODUCTION

This article examines the multimedia representation of music for synthesis. The data needed for synthesis coding have, among others, characters representing the symbolic nature of music, such as chords and rests, performance and audio information. The XML format describes these aspects of music coding with computer in detail ([1], [2]).

These three aspects - symbols, performance and audio - are needed for synthesis algorithms. In the following paragraphs it is shown how such aspects are represented in IEEE 1599 and how they can be made to interact with some synthesis systems. Some examples will be given to clarify several points and illustrate how IEEE 1599 can be easily integrated with these synthesis systems and increase their potentialities.

The interface to IEEE 1599 allows synthesis systems to interact with most of its contents. For example, they may use all the musical symbols encoded in the LOS layer (4). Thanks to this, any IEEE 1599 score can be directly played with any kind of timbres. Therefore, synthesis systems have the capability of interacting with all the pieces of information contained in IEEE 1599, including symbolic ones.

In fact, IEEE 1599 favors interconnection with various representations of the same musical content, and this approach enables synthesis systems to directly communicate with all of them. For instance, thanks to the Performance Layer, supported synthesis systems can interact with some of the real-time controls as required. Therefore it is possible to control every synthesis instrument in real-time using MIDI, OSC and HID devices, TCP networking real-time controls. Hence, “analysis-based-synthesis” is also managed in this approach thanks to the direct interaction with the audio samples contained in the Audio Layer.

In the second Section it is shown how the IEEE 1599 logical structure can be useful to interface with the most common synthesis systems. This section also mentions some of the most important systems.

The third Section demonstrates the ease in generating sounds synthesized in the most common synthesis languages starting from an IEEE 1599 file.

In the fourth Section, all explicit references of an IEEE 1599 file to Csound and MPEG4-Structured Audio are analyzed.

The fifth Section shows the explicit references of an IEEE 1599 file to MPEG4-Structured Audio.

In the sixth Section, the possibility of interfacing all sounds previously mentioned with all the other contents of an IEEE 1599 file are demonstrated.

II. MULTILAYER IEEE 1599 STRUCTURE FOR MUSIC SYNTHESIS SYSTEMS

A brief overview of the standard is necessary to understand the potentiality of the IEEE 1599 standard in regard to other synthesis systems, while a more complete description is given in [1].

The IEEE 1599 standard is basically organized as a multilayered environment, in which each layer represents a particular aspect of the computer codification of music pieces. The IEEE 1955 layers are: General, Logic (with
two important sublayers: \textit{Spine} and \textit{LOS}), \textit{Structural}, \textit{Notational}, \textit{Performance} and \textit{Audio}.

Logic is made up of two elements: \textit{Spine} and \textit{LOS} (Logically Organized Symbols). The \textit{Spine} marks the significant events in order to reference them from other layers, and the \textit{LOS} describe all the symbols of a score (e.g., chords, rests, and other musical symbols). The \textit{Performance Layer} encodes parameters of notes to be played and parameters of sounds to be created by performance languages. The \textit{Audio Layer} describes the properties of the source material containing audio information about the piece. The other three IEEE 1599 layers not strictly related with the content of this article are, respectively, the General, the Structural and the Notational, where the first contains cataloguing information, the second describes musical objects and their causal relationships, and the third links visual instances of the current piece.

The capability of managing all characteristics of a music piece in a single frame allows a synthesis system to interact, in a single useful framework, with all the needed contents. The synthesis systems in question are the textual and the visual systems. Among the first ones, those of interest are Csound [3], MPEG4-Structured Audio [4], Chuck [5], CLM [6] and SuperCollider [7]. Among the second ones, there are Pure Data [8], jMAX [9] and pure:dyne [10]. For the latter, only free systems are considered, particularly those that contain a software sound synthesis and a music composition package at the same time.

An example of all the above is represented by the direct interaction with audio samples needed in well-known synthesis techniques, such as the one based on analysis.

The manipulation of audio samples represents a precious resource for the creation of interesting and different kinds of sounds. Techniques widely used in research are FOF synthesis and analysis-based-techniques, Phase Vocoder, Heterodyne filter analysis, and Linear predictive analysis.

The following excerpt of IEEE 1599 code shows how every event of a wave file can be represented in relationship to other musical information.

```xml
<audio>
  <track file_name="example.wav" ... ></track>

  <track_event spine_ref="p1v1_1" /> 
  <track_event spine_ref="p1v1_2" /> 
</audio>
```

Every \textit{track event} is connected with all other pieces of information of the music file thanks to the XML element \textit{spine_ref}, as explained in the Section VI.

III. MUSIC SYMBOLS WITH IEEE 1599 AND MUSIC SYNTHESIS SYSTEMS

This section shows how it is easy to obtain all the notes of a synthesis system from the symbolic contents of and IEEE 1599 files, thanks to the clarity and simplicity of the representation.

When all IEEE 1599 symbols are translated into a synthesis language, together with the effective instruments, generation of audio is immediate. The contents produced can be related to the rest of the IEEE 1599 file through immediate and precise rules, as defined in Section VI.

The generation of a synthesis score from symbolic information depends basically on the single annotated sounds, such as chords, notes, rests, etc. The difficulty lies in accurate translation of every symbolic event so that no loss of information occurs.

Usually the core data necessary for the representation of single sounds are the referred timbre, the start time, the duration and the frequency. Figure 2 shows a diminished chord on A and its corresponding IEEE 1599 code.

Looking at the code, it is clear that a \textit{chord} is a single element that contains a number of sub-elements that encode each single note (\textit{notehead}). The duration is related to the chord and the exact pitch is associated to every \textit{notehead} element, as in classical music notation.

The same events of the figure, coded in Csound, have the following structure:

```plaintext
i1 0 4 5000 8.09
i1 0 4 5000 9.00
i1 0 4 5000 9.03
i1 0 4 5000 9.06
```

In Csound, every line represents a single sound event. The first value of every line refers to the timbre to be associated with the current sound, the second one the
time of start expressed in seconds, the third the duration (again in seconds), the fourth is amplitude as an absolute value, the fifth the pitch according to octave-point-pitch-class notation. The duration is calculated with PCM and is equal to 60.

In this case, there are four different sounds, whose start and duration time are the same.

In the following lines it is shown how the task is the same for the other synthesis languages mentioned above.

The translation in MPEG4-Structured Audio is the following:

```
0 string_strum1 4 5000 8.09
0 string_strum1 4 5000 9.00
0 string_strum1 4 5000 9.03
0 string_strum1 4 5000 9.06
```

Here the syntax is very similar to Csound, except that the starting time comes before the id code of the instrument that has to play the current sound.

Also the Chuck representation is easy to deduce from IEEE 1599 symbols. Using only simple sine to play particular pitches, the following file uses the same notes of the example:

```
SinOsc s1 => dac;
SinOsc s2 => dac;
SinOsc s3 => dac;
SinOsc s4 => dac;
440.00 => s1.freq;
554.36 => s2.freq;
622.25 => s3.freq;
739.98 => s4.freq;
4::second => now;
```

The first four lines connect four different sine oscillators to a D/A converter, while the other four let every oscillator sound with frequencies corresponding to the notes of the chord to be played.

Also in CML, single sounds are created by simply specifying the name of the instrument to invoke, its timing and the pitch of the sound created. The CLM representation of the usual example is the following:

```
name_instr    0    1    440.00     0.75
name_instr    0    1    554.36     0.75
name_instr    0    1    622.25     0.75
name_instr    0    1    739.98     0.75
```

The elements are, in order, the name of the instrument, the second of the starting time, the duration, the pitch and the amplitude.

Once again, for SuperCollider the information required is the same.

The visual languages for synthesis, working in real-time, such as Pure Data, are able to acquire XML files and manage contents related to the single note information in a direct way.

To conclude this section, it has to be recalled that the notes of a chord are particular objects of a score and that a detailed management of more complex information, such as different kinds of articulation, tie symbols and irregular groups, requires a deeper approach. For brevity, only the last case, namely *tuplets*, will be treated here, however this example will be useful to understand that IEEE 1599 symbol representation is complete and thus easy to convert in every other format. Only the Csound translation will be shown.

IEEE 1599 allows specification of the actual duration of every element in an irregular group. Tuplets can be represented by considering both the total duration of all the music objects of the group and the single annotated duration of every element. For instance, it is possible to represent a situation where 3 quavers take the place of 2 quavers, as shown in the following example. The attributes of *tuplet_ratio* reflect the instruction: "enter 3 quavers in the space of 2 quavers".

```
i1 0 0.5 10000 9.00
i1 0.5 0.5 10000 9.04
i1 1 0.33 10000 9.02
i1 1.33 0.33 10000 9.04
i1 1.66 0.34 10000 9.05
```

In conclusion, it has been shown that IEEE 1599 provides sufficient information for the creation of complete synthesis scores.

IV. IEEE 1599 DIRECT INTERCONNECTION TO CSEQU ND AND MPEG4-STRUCTURED AUDIO

This section deals with the direct link between Csound contents and IEEE 1599 files. This interconnection enables the storage of information strictly related to exact lines of Csound files. The kernel for communication between Csound codes and IEEE 1599 layers is the structure *csound_instance*, a sub-layer of *Performance* layer.

The corresponding XML element in IEEE 1599 of a Csound score is *csound_score*. The following is an example of its use:

```
<csound_score file_name="example.sco">
<csound_spine_event
```
It can be seen that, for every score, the correspondence between all Csound notes and the related IEEE 1599 events is coded in IEEE 1599 format. The score lines to be linked in Csound are only those in which the first char is an “i”, because, in this case, only the synchronization between sounds and events is of interest. In the csound_score, the purpose of parallelism between Csound and IEEE 1599 refers only to the representation of single notes. Therefore, all information about the function tables – which is expressed in those score lines where the first char in an “f” – should be obtained from the instruments that use them – whose correspondence is coded in csound_orchestra. The following is an example of the last element mentioned:

```xml
<csound_orchestra file_name="example.orc">
  <csound_instrument_mapping instrument_number="1" start_line="10" end_line="20">
    <csound_part_ref part_ref="1"/>
    <csound_spine_ref spine_ref="p1_v1_1"/>
  </csound_instrument_mapping>
</csound_orchestra>
```

In this case, every instrument of the orchestra is univocally characterized by the attribute instrument_number, while start_line and end_line specify the exact lines of the instrument in the orchestra. This way, every instrument can be related to particular IEEE 1599 contents – univocally defined through part_ref and spine_ref).

Finally, SASL and SAOL are managed similarly to Csound. The score is managed in SASL and the code of an example can be similar to the following:

```xml
<mpeg4_score file_name="">
  <mpeg4_spine_event row_num="" spine_ref=""/>
</mpeg4_score>
```

Here we have an example of SAOL:

```xml
<mpeg4_orchestra file_name="">
  <mpeg4_instrument_mapping instrument_number="" start_line="" end_line="">
    <mpeg4_part_ref part_ref=""/>
    <mpeg4_spine_ref spine_ref=""/>
  </mpeg4_instrument_mapping>
</mpeg4_orchestra>
```

V. IEEE 1599 MIDI CONTROL

For the sake of completeness, it is also necessary to mention the management of MIDI contents supported by IEEE 1599. In fact, the Performance Layer provides another sub-layer to enlarge the timbre potentialities of a synthesis system, namely, midi_instance. Here is an example of how a MIDI file can be linked to IEEE 1599 files:

```xml
<midi_instance file_name="ex.mid" format="0">
  { ... }
</midi_instance>
```

This code is useful when synthesis language contents have to be connected to MIDI for sound generation. The most useful utilities for interconnection between MIDI and IEEE 1599 are direct access to MIDI contents, and real time interaction with them.

Besides, the possibility of establishing an interaction between IEEE 1599 and MIDI is useful because of the large availability of musical archives in SFM format and to use a sequencer, and a program that prints music.

VI. INTERLAYER CONNECTION

The logical structure of the interconnection between all the concepts shown above can be simplified with the following scheme:

```xml
<IEEE1599>
  { ... }
  <logic>
    <spine>
      { ... }
      <event id="p1_v1_1" ... />
      <event id="p1_v1_2" ... />
      { ... }
    </spine>
  </logic>
</IEEE1599>
```

The events quoted above are collected in the spine to define the univocal linking in the current file through the defined id. If one wants to know the musical contents related to a specific reference in the spine, only those IEEE 1599 elements have to be searched whose spine_ref values are the same. For example, the algorithms for synchronization between the symbols discussed in the third section and the synthesis or audio information correspondent are based on this logic, and the synthesis score information, as well as the audio, is generated according to the IEEE 1599 rules of coordination defined in the spine.

Therefore, interaction between synthesis score and audio is immediate and, at the same time, easy to use and manipulate.

VI. CONCLUSION

It has been shown how common standards oriented to synthesis can be improved through their integration in IEEE 1599. Software implementation of all the concepts cited in this paper open a new way to integrate different musical contents and contexts.
Elisa Russo received the M.Sc. degree in Informatics from the Department of Information Science (DSI) of the University of Milan in 2006 with full marks, and afterwards the admission to the PhD in Computer Science (Computer-music applications) at the Laboratory of Musical Informatics (LIM) of the DICo (Department of Informatics and Communications) of the same university. Currently she is spending the final year of PhD as visiting researcher at the University of Bath, Department of Computer Science, UK. Her research interests include sound synthesis, timbre programming, automatic audio generation of music pieces encoded in symbolic format, flute sound analysis. Her researches in computer music are greatly motivated by her musical interests. She got a Diploma of Flute from the “Conservatorio G. Cantelli” of Novara in 2004. She plays in concerts with orchestras and teaches at music schools. Contact: russo@dico.unimi.it.