NEW DATA STRUCTURE FOR OLD MUSICAL OPEN WORKS

Sergio Canazza
Centro Polifunzionale di Pordenone
University of Udine
Via Prasecco 3/A - 33170
Pordenone
+39 0434239404
sergio.canazza@uniud.it

Antonina Dattolo
Dept. of Informatics and Mathematics
University ofUdine
Via delle Scienze 206 - 33100
UDINE
+39 0432558757
antonina.dattolo@dimi.uniud.it

ABSTRACT

Musical open works can be often thought like sequences of musical structures, which can be arranged by anyone who had access to them and who wished to realize the work. This paper proposes an innovative agent-based system to model the information and organize it in structured knowledge; to create effective, graph-centric browsing perspectives and views for the user; to use authoring tools for the performance of open work of electro-acoustic music.

Keywords

Musical Open Work, Multimedia Information Systems, Software Agents, zz-structures.

1. INTRODUCTION

A classical musical composition (a Beethoven's symphony, a Mozart's sonata, or Stravinsky's *Rite of Spring*) posits an assemblage of sound units that the composer arranged in a closed, well-defined manner before presenting it to the listener. He converted his idea into conventional symbols, obliging (more or less) the (eventual) performer to reproduce the format devised by the composer himself. On the contrary, a number of music pieces (or, more general, of multimedia works) are linked by a common feature: the considerable autonomy left to the individual performer[1] in the way he chooses to play the work. Thus he is not merely free to interpret the composer's instructions following his own discretion (as happens in traditional music), but he must impose his judgment on the form of the piece, as when he decides in what order to group the sounds: a real act of improvised creation. In *Klavierstück XI*, Karlheinz Stockhausen presents to the performer a single large sheet of music paper with a series of note groupings; he then has to choose among these groupings, first for the starting unit and, next, for the successive ones in the order in which he elects to weld them together: in this way, he can *mount* the sequence of musical units in the order he chooses, changing the *combinative* structure of the piece. In Pierre Boulez' *Third Sonata for piano*, the first section (*Antiphonie*, *Formant 1*) is made up of ten different pieces on ten corresponding sheets of music paper. These can be arranged in different sequences like a stack of filing cards, though not all possible permutations are permissible. A particularly representative example of musical open work is *Scambi*, an analogue tape work created in 1957 by the Belgian composer Henri Pousseur at the Studio di Fonologia in Milan. By means of a specific process, called dynamic filtering (realized by using a special equipment, designed by Alfredo Lietti, the engineer of the studio), the composer is able to extract from noise animated time structures, then to process them further in different parameters, and thus to produce 32 sequences. These sequences can be arranged by anyone who had access to them and who wished to realize the work, according to certain rules regarding their order and possible overlapping.

Today several multimedia artistic works are 'in progress', 'open' in structure or in realization. Here the audio recording is included in a complex procedure of audio signal processing and where different writing systems (video, text, static pictures, gestures) flow together. In this field emerges the necessity to define suitable data structures and to convey, in a single (digital) medium, verbal and musical documents, pictures, audio and video signals.

The aim of this paper is to provide an innovative system for the performance of electro-acoustic music open works, considered as a representative subset of multimedia artistic works. Will be introduced innovative authoring tools able to make open a musical work, enabling the user to become self-author of new versions for a given work.

These topics are faced using an agent-based extension of the ZigZag model. Conceiving a system as aggregation of autonomous and cooperative agents is one of the most exciting aspects of the challenging arena known as multi-agent system (MAS); this perspective revolutionizes radically the way in which a model may be conceived and work [3]. In addition, the presence of ZigZag model guaranties a graph-centric browsing tool, which supports the representation of persistent context: user visualizes the unit of interest in relation to other associated units. But more than this, he sees the unit of interest from that unit's position relative to multiple perspectives or orientations. As case study, we chose the cited, complex electro-

[1] As it is well known, the practical intervention of a performer (the musician who plays a music piece) is different from that of an interpreter in the sense of consumer (somebody who listens to a musical composition performed by somebody else). In this context, however, both cases are seen as different manifestations of the same interpretative attitude [5].

acoustic open work (*Scambi* by Henri Pousseur), characterized by a variety of sequences and of different performance degrees of freedom.

2. ACTOR–BASED ZIGZAG MODEL

In order to present our model in section 3, this section introduces in two separate subsections (2.1 and 2.2) the basics of the ZigZag model and a brief description of the actors, a particular class of computational agents.

2.1 The ZigZag model

ZigZag [6] introduces a new, graph-centric system of conventions for data and computing; it separates the structure of information from its visualization (i.e. the way the data – text, audio, video - is presented to the user); therefore a ZigZag structure handles all the different visualizations necessary to realize an Electronic Edition of musical works.

The main element present in the ZigZag model is the zz-structure: it can be viewed as a multigraph where edges are colored, with the restriction that every vertex, called zz-cell, has at most two incident edges of the same color; the sub-graphs, each of which contains edges of a unique color, are called *dimensions*. The cells in a same dimension are linked into linear and directed sequences, called *ranks*. Each dimension can contain a number of parallel ranks, which are a series of distinct cells connected sequentially.

Since there is no canonical visualization, the pseudo-space generated by zz-structures is called zz-space and may be viewed in various ways. A view is a presentation of some portion of zz-space and is presented by a view program, which visualizes, for example, a region around a particular cursor.

A 2D view can be drawn picking a single cell as a focal point, and drawing the neighborhood around that cell along two chosen dimensions. By changing the chosen pair of dimensions, we can visually reveal, hide, and rearrange nodes in interesting ways. Considering that a zz-structure may be very large, and that there is usually not enough room in the 2D view for all of the cells, we restrict the dimension of the 2D view.

Some observations are necessary on the zz-cells; they are the principal unit of the system and they are conceived not only as passive containers of primitive data (i.e., text, graphics, audio, etc.) but they can have types, based either on their functions or on the types of data they contain. Thus, a zz-cell may have a variety of different properties and functions, such as executable program or scripts (this type of cell is called progcell), or represent the package of different cells (this type of cell is called referential cell).

Analogous observations can be made on the dimensions; in fact, they can be passive and nominal (merely receiving and presenting data) or operational, programmed to monitor changing zz-structures and events, and calculate and present results automatically (for example, the dimensions d.cursor and the dimension d.clone). From these considerations it turns out that it is reductive to associate zz-cells to passive entities meant simple nodes are in a graph. So, we have considered the opportunity to model a zz-cell by means of a specific class of computational agents, the actors.

2.2 Brief description of the actor model

The actor model [1] is a model of concurrent computations in distributed systems; it is organized as a universe of inherently autonomous computational agents, called *actors*, which interact with each other by sending messages, improving on the sequential limitations of passive objects.

Each actor is defined by three parts: a **passive part**, which is a set of local variables, termed *acquaintances*, that constitute its internal state; an **active part**, that reacts to the external environments by executing its procedural skills, called *scripts*. This active part constitutes actor's behaviour; the third part is represented by the actor's **mail queue**, which buffers incoming communication (i.e. messages).

Each actor has a unique name (the uniqueness property) and a given behaviour; it communicates with other actors via asynchronous messages. Actors are reactive in nature, i.e., they execute only in response to messages received.

An actor can perform three basic actions on receiving a message: *create* a finite number of actors with universally fresh names, *send* a finite number of messages and *assume* a new behaviour.

The actor's behaviour is deterministic in that its response to a message is uniquely determined by the message contents and its internal state. Furthermore, all actions performed on receiving a message are concurrent.

In order to describe the actors in our model we adopt the formalism used in [3]:

```
(DefActor ActorName
 [inherits-from Class-Name]
    <acquaintances list>
    {scripts list} )
```

Therefore an actor is described by specifying its superclass, its data part and its script part; the script part represents the set of scripts that can be executed.

3. THE MODEL

The architecture of our model is organized in two layers: a *component layer* contains the zz-cells, those are actors specifically designed to model the audio documents domain; a *meta layer* contains the actors classes specialized, for example, to manage connections among zz-cells or to generate specific views on them. The interaction between actors is defined using the diagrammatic language AUML (Agent Unified Modelling Language) [8], extension of UML for agents.

3.1 Component layer

The component layer is defined in relation to the magnetic tape structure: each open reel is usually composed by several physical segments, i.e. pieces of magnetic tape connected by means of adhesive tape (called junction).

In each segment, the audio signal is recorded in one, two or more tracks. Following this structure we define the actors Source, PhysicalSegment, and DigitalSignal. Moreover, the actor LogicalSegment is introduced, with the aim to compare the sources on the basis of a segmentation that is different by the physical one. The actor Source represents the overall characteristics of the document, such as the tape width (typical values are 1/4, 1/2, 1, and 2 inch) and the cataloguing fields.

```
(DefActor Source
    <physicalSegments width archive shelfMark inventory
      conservationCondition …>
    {calculateDuration …})
```

It contains also a list of phisicalSegments, which compound the open reel tape. This actor is able to perform several actions, e.g. the script calculateDuration asks each physical document for his length and rate and it calculates the total duration of the tape. The LogicalSegment carries out a virtual partition of the Source.

```
(DefActor LogicalSegment
      <start length source rate equalization
      noiseReductionSystem
      tracksLayout digitalSignal audioQuality …>
      {getQuality getSignalProperties …})
```

Its acquaintances are the start time and the duration of the segment, a pointer to the corresponding source, the recording rate (tipical values are 7.5 or 15 inch/s), the equalization (e.g . IEC1/CCIR), the noiseReductionSystem (e.g. Dolby or dbx), the tracksLayout (i.e. the tracks number and width), a pointer to the digitalSignal representing the audio recorded on each track, and an audioQuality index, that can be subjectively specified by the user. If the audioQuality field is left blank, the script getQuality asks the digitalSignal to estimate the signal to noise ratio of each track and returns an index of quality. The script getSignalProperties asks the digitalSignal for the digital audio of each track and estimates if the audio signal is monophonic, stereophonic, or polyphonic. This information doesn't always correspond to the number of tracks, because, following the practice of magnetic recording, a monophonic signal can be recorded on a multi-track format, writing the same signal in all the tracks. A specialization of this actor class is the PhysicalSegment actor that specifies geometrical features of the segment, such as the angle of his junction.

```
(DefActor PhysicalSegment
  [inherits-from LogicalSegment]
     <junctionAngle …>
     {getFadeDuration …})
```

The script getFadeDuration, starting from the geometrical properties of the junction, calculates the duration of the fade-in and fade-out of the audio at the segment edges. The NumericSignal actor represents the audio signal recorded on the tracks.

```
 (DefActor NumericSignal
      <size samplingRate resolution data[i,j]>
      {estimateSNR calculateSTFT
   calculateAmplitudeEnvelope})
```

Its acquaintances are related to the numeric format: number of samples, sampling rate, and resolution. data[i,j] is a matrix, with a row for each track and a column for each sample. This actor can perform several actions that, using signal-processing techniques, calculate signal to noise ratio, short-time Fourier transform, amplitude envelope or other kinds of representations.

3.2 Meta layer

In order to manage the information in a zz-structure context, a meta layer is added. Following the rank definition introduced in ZigZag model, the actor Rank can be described as an ordered list of actors, belonging to the same dimension. Each specific dimension contains one or more ranks. Examples of dimensions are:

- *source* dimension, that composes given sources as an ordered sequence of segments;
- *equivalent segments* dimension: links segments with a common music content present in different sources.

For each dimension, we define specialized classes of actors, (such as SourceRank and EquivalentSegmentsRank).

```
(DefActor SourceRank
      [inherits-from Rank]
      <source …>
      {swapSegment …})
```

```
(DefActor EquivalentSegmentsRank
      [inherits-from Rank]
      <sources …>
      {compareQuality …})
```

Another typology of meta-actor is the generativeProcess; this class is devoted to generate dynamic new virtual hyperdocuments.

```
(DefActor generativeProcess
      <… >
      {generateView createSource})
```

4. CASE STUDY

We propose now the analysis of an interesting case study, in which we use a subset of scripts and actors defined above: the Pousseur's electronic work *Scambi*. Writing about his work in 1959, Pousseur ended by envisaging the day when technology would allow listeners to make their own realizations of the work (either following his 'connecting rules' or not) and to give the, now active, listener the experience of a temporal event open to his intervention and which could therefore be elevated in type, as vital, creative freedom [7]. The active listener becomes, in effect, a composer; reception and interpretation are expressed as (musical) production.

In our paper, Pousseur's invitation to interpret his work creatively as re-composition has been extended to remix and other types of appropriation that were not only permitted but welcomed by the composer [7] (a position that associates him with popular-music culture in which such freedom is assumed).

In our case study, we have collected the original 32 audio sequences realized by Pousseur, thanks to the 'Scambi Project', Lansdown Centre for Electronic Arts, School of Arts, Middlesex University, UK (http://www.scambi.mdx.ac.uk).

Thanks to the cooperation of different classes of actors, our system allows user-author to surf among the existing performances of Scambi (by the composer, Luciano Berio and others) and to create a new virtual source, automatically picking up the audio sections following the 'connecting rules' proposed by the composer. Moreover, the user-author can establish himself stochastic rules.

Thanks to the separation of structure (dimensions) from content (zz-cell) in zz-structures, different users can customize their workplace and the visualization of same contents, creating new dimensions or new virtual performances. This is achieved activating specific actor collaboration schemes and is shown in following section 4.1.

4.1 New performances

The four acoustic parameters taken into account by Pousseur are:

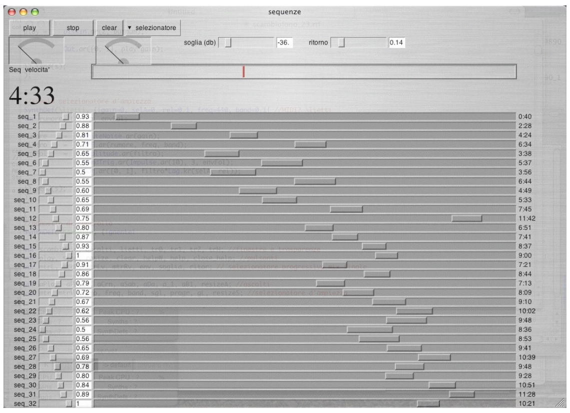
1) statistical tempo (from slow to fast);
2) relative pitch (from low to high);
3) homogeneity of the sound pattern (from dry to strong reverb);
4) continuity (from long breaks to continuous sound). In Figure 1 is detailed the start- and end-situations for each sequence.

We define *equivalent segment* as the sequences that can follow the current sequence on the basis of Pousseur's intention to ensure a transition without break (in the four parameters) from one sequence to the next (a sort of 'continuity principle').

Family	Sequence	Pitch	Speed	Hom.	Cont.	Dur.
1	1-2	0/1	1/1	1/0	0/0	42"
2	3-4	0/1	1/1	0/1	1/1	42"
3	5-6	1/0	1/1	0/0	0/1	42"
4	7-8	1/0	1/1	1/1	1/0	42"
5	9-10	1/1	1/0	1/0	1/0	30"
6	11-12	1/1	1/0	0/1	0/1	30"
7	13-14	1/1	0/1	0/0	0/0	30"
8	15-16	1/1	0/1	1/1	1/1	30"
9	17-18	0/0	0/1	1/0	0/1	30"
10	19-20	0/0	0/1	0/1	1/0	30"
11	21-22	0/0	1/0	0/0	1/1	30"
12	23-24	0/0	1/0	1/1	0/0	30"
13	25-26	1/0	0/0	1/0	1/1	42"
14	27-28	1/0	0/0	0/1	0/0	42"
15	29-30	0/1	0/0	0/0	1/0	42"
16	31-32	0/1	0/0	1/1	0/1	42"

Figure 1. Characteristic per sequence (from: Decroupet [4]).

This segmentation process can be iteratively applied to all the sequences, obtaining a set of audio segments linked along two dimensions. The user-author can generate new performances mixing different sequences also in polyphonic structure. To do so, user can apply deterministic laws (given by the composer), stochastic models or self-oriented choices; this allows user to generate new 'reading' performances of an open work.



soglia (db) -36. ritorno 0.14

4:33

seq_1	0.93	0:40
seq_2	0.88	2:28
seq_3	0.81	4:24
seq_4	0.71	6:34
seq_5	0.65	3:38
seq_6	0.55	5:37
seq_7	0.5	3:56
seq_8	0.55	6:44
seq_9	0.60	4:49
seq_10	0.65	5:33
seq_11	0.69	7:45
seq_12	0.75	11:42
seq_13	0.80	6:51
seq_14	0.87	7:41
seq_15	0.93	8:37
seq_16	1	9:00
seq_17	0.91	7:21
seq_18	0.86	8:44
seq_19	0.79	7:13
seq_20	0.72	8:09
seq_21	0.67	9:10
seq_22	0.62	10:02
seq_23	0.56	9:48
seq_24	0.5	8:36
seq_25	0.56	8:53
seq_26	0.65	9:41
seq_27	0.69	10:39
seq_28	0.78	9:48
seq_29	0.80	9:28
seq_30	0.84	10:51
seq_31	0.89	11:28
seq_32	1	10:21

Figure 2. A screenshot of the system. X-axis: time, Y-axis: sequences. The user can be realize a polyphonic structure and modify the pitch, the duration and the volume of each sequence.

We assume that an user is interested in creating the new performance starting from a segment src_n; this request is captured from the meta-actor generativeProcess and it is forwarded from it to the source rank r^{src} (that manages the logical segments src_1, …, src_{32}). r^{src} sends a synchronous multicast message (CalculateRule) to all its logical segments. In order to enable the comparison among its features and that of others segments, each segment (src_j, j=1,...32) assigns this task to the rank $r^{e\text{-}sj}$ (that manages the logical segments able to follow src_n on the basis of Pousseur's 'continuity principle'). Each $r^{e\text{-}sj}$ contacts (in synchronous multicast way) all its components ($e\text{-}src_{js}$, s = 1, ..., m) and, following stochastic law defined off-line by the user or by means of user input, it choices the components with the best matching. This information is returned to generativeProcess. This last actor collects the segments and creates the new requested performance. In Figure 2 a screenshot of the system is showed.

5. CONCLUSION

The era of high modernism, in which concept of the open work was a radical resistance to this dominant aesthetic, has been relegated to history. The contemporary western culture, as it is well-known, assumes that all musical works are *open* to perpetually renewed interpretation by listeners, musicologists, analysts, and performers [2]. In particular, in multimedia domain no work is permitted to resist endless (interactive) interpretation. This contemporary situation is partly the effect of the invention of the concept of the open musical work, in which Pousseur was a precursor. For this reason, the interest in *Scambi* is particularly high today, as also proved by the success obtained by the *Scambi Project* (www.scambi.mdx.ac.uk/). One effect of our work might be to free the historical musical open work from its iconic status as history, to revive and redefine its specific openness within general (digital and interactive) openness, and to return a continuous presence to it by opening it up to interpretive renewal.

6. REFERENCES

[1] Agha, G. *Actors: A Model of Concurrent Computation in Distributed Systems*, MIT Press, Cambridge, MA, 1986.

[2] Ayrey, C. Pousseur's Scambi (1957), and the new problematics of the open work, *Proc. of Symposium on* Scambi *at Goldsmiths College*, University of London 2005.

[3] Dattolo A. and Loia, V. Distributed Information and Control in a Concurrent Hypermedia-oriented Architecture. *Inter-national Journal of SEKE*, Vol. 10, n. 6, pp. 345-369, 2000.

[4] Decroupet, P. Vers une théorie generale – Henri Pousseurs "Allgemeine Periodik" in *Theorie und Praxis in MusikTexte* 98, pp. 31-43, 2003.

[5] Eco, U. *The role of the reader: explorations in the semiotics of texts*, Indiana University Press, USA, 1979.

[6] Nelson, T. H. Cosmology for a Different Computer Universe. *Journal of Digital Information*, Vol. 5, Issue 1, 2004.

[7] Pousseur, H. Scambi. In *Gravesaner Blätter* IV, pp. 36-54, 1959.

[8] Winikoff, M. Toward making agent UML practical: a textual notation and a tool. *First International Workshop on Integration of Software Engineering and Agent Technology*, Melbourne, Australia, pp. 401-412, 2005.