EXPRESSION MARKS FOR PROGRAMMING INTERACTIVE MUSIC

Juan Carlos Martinez Nieto
Georgia Tech Center for Music Technology
jcm7@gatech.edu

Jason Freeman
Georgia Tech Center for Music Technology
jason.freeman@gatech.edu

ABSTRACT

The present work uses common Western music notation to represent logical and systematic behaviours of computer music processes in the context of score-oriented interactive music. The algorithmic representation is described by adding programming annotations in a controlled natural language to a musical staff as expression marks in the score. We implemented a computational environment that is able to translate these expression marks into coding instructions and execute them in real-time during a live performance of an interactive-music piece. A collection of short interactive music exercises for MIDI-controlled piano based on the proposed notation was composed and edited using music engraving software. During the compilation stage, an encoded version of the score in MusicXML format is translated into scripting code, and during live performance the computational environment executes the code in real time in sync with the human-performed parts. This paper introduces the syntax of expression marks for programming interactive music through a classic "Hello World" example in the context of interactive music and explains the technical details behind the implementation of the computational environment. The main motivation behind this work was to evaluate the viability of creating a cohesive symbolic representation of interactive music that is independent of specific software and hardware frameworks, and is strongly connected with the western music tradition.

1. INTRODUCTION

In the context of score-oriented interactive music, creating and preserving repertoire is not straightforward. One reason is that performance information regarding an interactive piece is not entirely represented in a musical score, so an important part of the information resides inside the computational framework on which the piece runs. This issue creates strong dependencies with a particular technology, a factor that has made some composers move away from the computer music genre [1].

In this paper, we propose to create a cohesive representation of an interactive music piece by keeping both performance instructions for human/acoustic musicians and performance instructions for a computational process in a single music score. Logical descriptors are added as expression marks in a musical staff (see section 4) and interpreted in real time by a programming engine during a live performance. This work presents a novel approach that fuses algorithmic thinking and traditional Western music notation. This approach is particularly well suited for modeling incremental music processes that usually are present in minimalist aesthetics [2].

This work is a first step in extending the technology independent representation of music, a common ground for pure instrumental music, into the field of interactive music. Selfridge-Field asserts that "since the representation of music is entirely independent of the use of computers, there is every reason to expect that codes designed for the representation of music in computer applications will eventually be entirely independent of both hardware configurations and software processes" [3].

Expression marks in common Western music notation have been used in musical scores since the eighteenth century to represent variations of tempo, intensity, and articulation [4]. The term is misleading as the scope goes beyond expressiveness in music performance [5]. In connection with the extensibility of expression marks, it is useful to bring the definition of the *Harvard Dictionary of Music* "Symbols and words or phrases and their abbreviations employed along with musical notation to guide the performance of a work in matters other than pitches and rhythms" [6]. The multi-purpose implicit characteristic of expressive marks, along with the fact that they are text-based signs, makes expression marks well suited for the purpose of extending a musical score with an algorithmic descriptor.

In regards to score-oriented interactive music, sometimes referred as score-driven interactive music [7], temporal relationships between human-performed musical events and automatic music processes play a fundamental role when creating interactive music [8]. In musical scores representing time-relationships among discrete-time events is simple and accurate. This fact motivates us to employ common Western music notation to represent systematic behaviors in interactive music. From this perspective, the score acts as a symbolic *source code* where the composer abstracts and clearly records structural relationships in time among the different musical entities (human performed instruments or automatic computer processes).

In summary, we propose to extend common Western music notation to describe systematic procedures by adding programming annotations as expression marks. The proposed approach allows representation of both the instru-

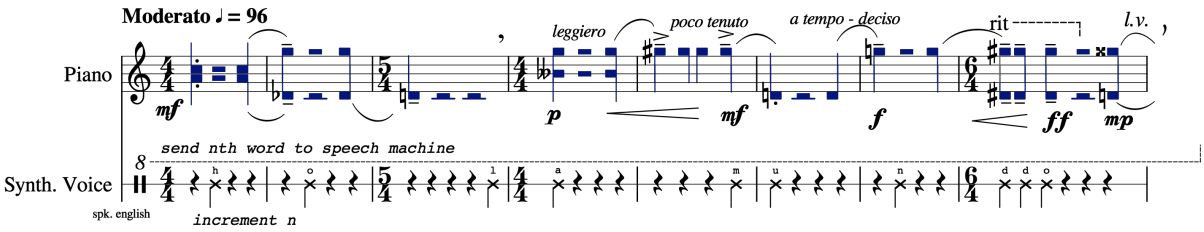

Figure 1. Score of the interactive exercise 'Hola Mundo'.

mental music and the algorithmic process in a single and well-understood standardized document such that the music is readily readable by a human, processable by a machine, and recreatable in other systems into the distant future. In the following sections we first discuss some related work, then walk through a simple example to introduce the basic concepts behind programming-expression marks, and finally the software implementation and a case study will be detailed in section 6.

2. BACKGROUND

The present work is loosely related to score languages, which have a long tradition in the field of computer music [9]. Score language refers to a text-based list of actions arranged in absolute or symbolic time. In that sense Score language is more related to a data structure with basic programming functionality. Max Mathews developed at Bell Labs a series of score languages known as MUSIC-N in the field of audio-synthesis; the first one of those languages appeared in 1957, and the last one MUSIC-V around 1969 [10]. MUSIC-V became popular in the academic and scientific world and was extensively used in the computer music field during the second half of the twentieth century.

The Score Language pattern paradigm has been used in many music programming languages since then; mainly in the field of audio-synthesis. Common Lisp Music [11] and Csound [12] are direct descendants of MUSIC-V languages. Nowadays, Antescofo [13], one of the most popular environments for interactive music that runs embedded on Max/Msp and Pure-data environments [14], provides a text-based Score Language for describing customized actions.

Score Languages are essentially sequences of events in their core conception, so modeling high level interactions among music entities can only be done at a very basic level. Dannenberg in his survey of Music Representation Systems states that:"This approach is straightforward, but it makes it difficult to encode structural relationships between notes" [15]. Modern Score languages address this issue by embedding custom-language programming scripts. For example, RTCmix, a score language started by Lansky, includes MinC, a C-style scripting language [16] and Antescofo enables the combination of score based instructions with data structures and control-flow logic all within a single script, but with a score-following paradigm [13].

However, from the symbolic representation perspective, these approaches create two different and simultaneous models of the same music that is represented. One is the score-representation for performance and the other one is the logic-representation for the computational framework. Our work addresses the representation of music interaction in a different way, by keeping the representation of the music in a unique and cohesive symbolic source and taking advantage of the multiple semantic connotations of common Western music notation for modeling structural relationships among musical entities.

This work provides a written representation that models automatic music processes in interactive music as an extension of common Western music notation. From the machine perspective, it implies building a programming engine that is able to interpret the programming-expression marks notated in the score.

Thanks to open standards for encoding a musical score, such as MusicXML [17] or MEI [18], an encoded file version of a musical score can be understood by a machine. For the purpose of this research, the composer can model and record an interactive process directly in the score by adding programming-expression marks during the editing phase using a third-party notation software, and the encoded version of the score is interpreted by a specific purpose programming engine.

3. HELLO WORLD

This section presents the basics of expression marks for programming interactive music using the classic "Hello World" approach as a walk-through example. Figure 1 shows the score of *Hola Mundo*, a very short interactive exercise for MIDI-controlled piano and synthesized voice. The bottom staff represents the synthesized voice part that is played automatically in Supercollider [19], and the top staff shows the human-performed part that is played on the piano. The score uses square shaped note-heads only to emphasize visually the algorithmic character, and they do not have semantic meaning.

Figure 2 shows the script library that is imported by the interpreter engine at the compilation stage. The technical details about how all different levels of information are connected to be able to synchronize and execute the instructions during a live performance will be explained later in section 6. Here we will present the underlying concepts.

```
var words = [];
words[0]=  'h';
words[1]=  'o';
words[2]=  'l';
words[3]=  'a';
words[4]=  'm';
words[5]=  'u';
words[6]=  'n';
words[7]=  'd';
words[8]=  'd';
words[9]=  'o';
```

Figure 2. JavaScript code for 'Hola Mundo'

3.1 Roles in Interaction

Note that each part plays a different role in the interaction (see Figure 1). The piano part acts as control signal (input) used by the programming engine to estimate the current symbolic time-position in the score (i.e. measure and subdivisions) during a live performance. In this paper, we label this interaction type as control role, as the input controls the pace at which the instructions are executed. Furthermore, the human-performed part acts as an external synchronization clock that adjusts the internal clock of the programming engine every time that a control signal is received.

The second staff plays two simultaneous and different roles in the *Hola Mundo* exercise. First, it increments a variable in steps of one. In other words, it defines a systematic behavior, which in this Hello World exercise is constrained to increment the variable `n` every time a quarter note appears on the bottom staff. We named this interaction type as logic role. The second role of the bottom part is derived from the 'sent' expression mark text. This mark is translated in sending a message to the speech machine (synthesized voice) every time a quarter note is notated. This type of interaction represents the input parameters of an external computer music process (Supercollider in this case). We labeled this interactive role as process role.

3.2 Programming-Expression Marks

The programming-expression marks in the bottom staff of *Hola Mundo* are entered in a controlled natural language. As an example, take the expression "*send nth word to speech machine*". When the encoded version file (e.g. MusicXML) of the music score is compiled (i.e. transpiled is the proper term for describing a translation among different source codes), the compiler looks for the following syntactic text structure: *action [expression] to [output]*, then a text disambiguation operation is applied to the string literal to get each part of the text. Next, the compiler checks if the expression is a key that maps to a stored script literal and if so replaces the original expression by the mapped script version. In this Hello World exercise, the string "nth word" is mapped to the script variable *words[n]*.

This *text-to-code* mapping mechanism strongly contributes to the ease of building and preserving interactive music repertoire for the following reasons. First, the symbolic representation (staff notation) acts as a descriptor of an algorithmic behavior at a higher level, allowing the composer to abstract and record the logic of the interaction, which along with the instrumental parts should be sufficient to recreate the piece in the future without the participation of the original performers and technicians. Second, an ambiguity is introduced in the symbolic representation as the technical implementation details are not recorded in the musical score. Thus, the music can be adapted to technological and aesthetic changes in the future, avoiding that the piece being frozen in time, in a similar way as expression marks work in the context of instrumental music performance.

4. THE MEANING OF PROGRAMMING-EXPRESSION MARKS

In a general sense, a programming-expression mark is a text-based descriptor of an algorithmic function that is applied only to the specific score-part where it is defined. It is executed every time that a note appears, and it is not executed on rests. This approach enables building a parallel environment where the algorithmic functions could run at different rates as each score-part could potentially evolve independently over time.

The programming-expression marks scope can be defined to cover the whole section or to be restricted to a single note. In the first case, usually the commands are repeated until it reaches the double-bar which enables reducing the information in the score. In the second case, the command is executed just one time in that specific note which is very useful for initializations. The following paragraphs will provide more details about the syntax and semantics of programming-expression marks.

In the Hello World exercise, the expression mark "*send nth word to speech machine*" appears in the first measure. After making the text disambiguation by the parser in the compilation stage, the text is split as follows: 'send' (action), 'nth word' (identifier), 'to' (connector) and 'speech machine' (identifier). The action *send* is interpreted during live performance as sending an OSC message [20] every time a note appears. The compiler searches internally for the identifier value (nth word), if this entry exists, the compiler replaces the identifier with the stored coding expression associated with this entry. In the *Hola Mundo* score, the identifier "nth word" is mapped to the script expression "words[n]". Using a similar approach, the identifier "speech machine" is mapped to a pre-defined OSC message template. Additionally, the send action in *Hola Mundo* associates the variable "words[n]" to the content of the OSC message. Before the compilation stage, a mapping table that associates each identifier with its equivalent is added.

All actions are assumed by the compiler to be repeated every time a note appears until the music section ends (separated by double barline) except when the modifier 'once' is present. The 'once' modifier constrains the scope of the action only to the current note (in contrast to the whole section). Table 1 shows a complete list of actions for the ex-

ACTION	MEANING
send	send expression-message to port
increment	var = var + 1
compute	execute expression
assign	var = expr
jump to	goto measure
print	print(expression) in console
stop	clear all actions on the current section

Table 1. List of actions

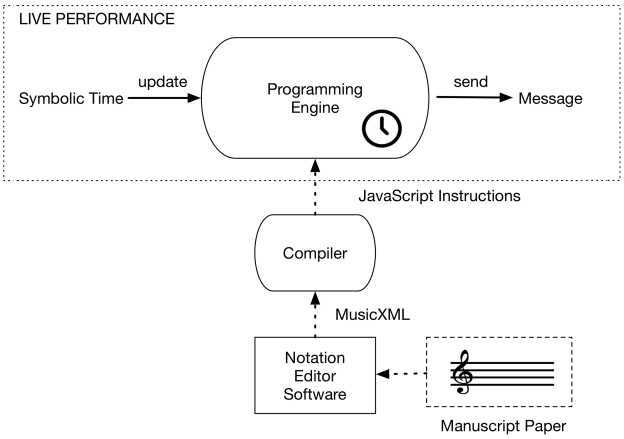

Figure 3. Implementation block diagram.

amples in this paper. The second column shows an equivalent pseudo-code of how the action is translated to the language engine during the compilation phase.

5. IMPLEMENTATION

Dynamic Programming Languages such as JavaScript are well suited for live environments as they enable interpreting and executing code in real-time. This programming approach is often referred as to Just-In-Time compilation or dynamic compilation [21]. Our implementation of the computational environment that interprets programming expression marks was written in C/C++ and has an embedded JavaScript engine to perform the Just-In-Time compilation of scripting code during live performance.

Figure 3 shows the block diagram of the actual implementation of the real-time environment based on the proposed notation for music interactive systems. The core of the implementation is the programming engine that interprets the programming expression marks in the score during a live performance. We use the music notation editor software *MuseScore* [1] for creating, editing, and exporting the score to MusicXML format.

The first step is to compile (transpile) the encoded version of the musical score. This step involves text disambiguation of the programming-expression marks in the score and translation of these commands into machine instructions. During this compilation stage, an encoded version of the score in MusicXML format is mapped to an intermediate scripting version in JavaScript that is stored in the programming engine, and it contains the symbolic-music-time locations where the instructions should be executed. Now the environment is ready for execution.

The programming engine has an internal clock that estimates the current symbolic music time, and based on that time, the corresponding scripting instructions are executed. As shown in the block diagram, an external signal with the current symbolic time feeds the programming engine to update the internal music-symbolic-time clock. Based on this update the internal music-time is estimated. This external input signal is derived from the live performance of the control-role parts(i.e. the human performed parts). In our implementation the external control signal is received via OSC. Furthermore, as shown on the block diagram, the output of the system consists of OSC messages that are sent to an external computational music framework.

6. CASE STUDY

A collection of short interactive exercises for MIDI controlled piano were composed to evaluate the viability of the proposed notation. Figure 4 shows the architecture of the implemented environment. In this setup, a human performed digital piano sends MIDI messages to a simple score following system implemented in Objective C that essentially detects chords events. The score following system estimates the current symbolic time position in the score, and sends the value to the programming engine via OSC.

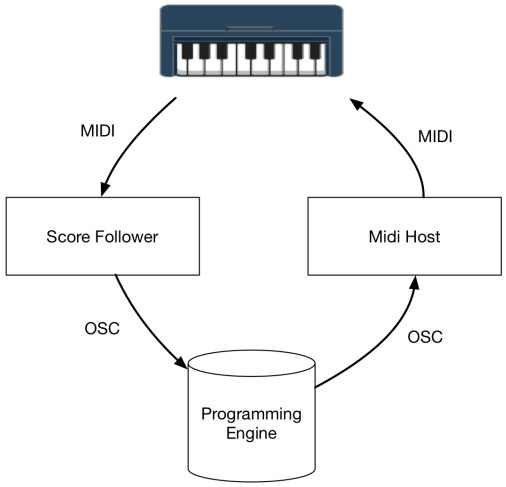

Figure 4. Implementation diagram.

As shown in Figure 4, the interactions in these piano exercises are focused on enhancing the human performance by adding an automatic counter-part played by the MIDI-Host application. The interactions are in essence minimalistic but in the variety of process music [22], meaning that one of the parameters of a music entity is gradually changed, and it is the process itself which determines the overall form of the piece [2]. Furthermore, this minimalistic approach to music composition is well-suited for evaluating a symbolic representation of logical behaviors.

[1] https://musescore.org/.

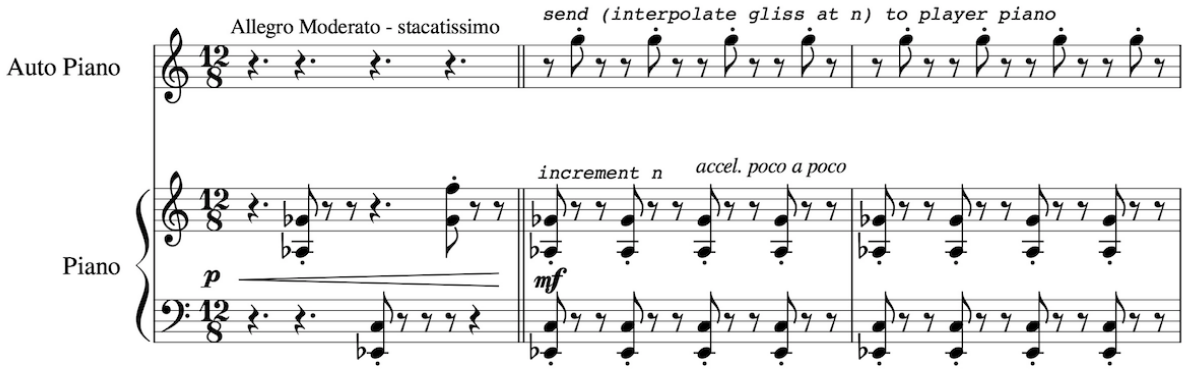

Figure 5. Score of 'Cencerro Deslizante'.

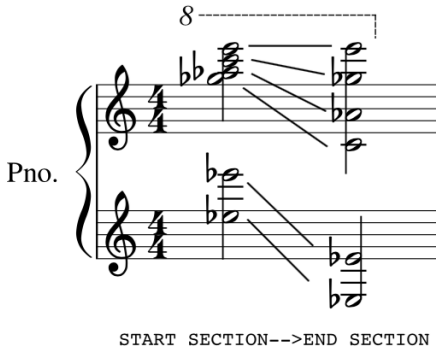

Figure 6. Process description for 'interpolate gliss at nth'.

During a live performance, OSC string messages are sent from the programming engine to a MIDI Host application developed in Objective C. The messages are strings in JSON format, and they contain the chord notes to be played by the MIDI Host Application. Note that the MIDI Host does not add any logical layer to the interaction environment. The JSON messages are mapped to MIDI messages in the MIDI Host controller application and played back in the Digital Piano. In this context the digital piano behaves as a hyper-instrument.

Figure 5 shows the first three measures of `Cencerro Deslizante`, one of the exercises of the collection, and Figure 6 shows a segment of the performance notes of this interactive exercise. In this exercise, the automatic piano part plays off-beat chords computed from an incremental process that interpolates between two chords. Each incremental process runs over a complete section (double barline) of the piece. It is explained in the performance notes and notated in the score by the programing-expression mark *'interpolate gliss at nth'*.

7. CONCLUSIONS

This research shows that it is possible to implement a programming engine that understands a cohesive score representation of interactive music that is independent of any computational framework by extending music notation to an algorithmic context. The present work proposes a new compositional approach that does not intend to be applicable in all cases of scored interactive music. Instead, it introduces a new compositional mechanism for interactive music which is strongly connected with traditional practices of writing music through notation and takes advantage of the multi-functional semantic scope of expression marks. We will focus our future research on developing a cohesive representation of interactive music by defining a formal syntax of programming-expression marks and creating a broad set of pieces to explore and enrich the different dimensions of the introduced compositional practice. This approach is well summarized by the following statement of Roger Dannenberg: "Music evolves with every new composition. There can be no 'true' representation just as there can be no closed definition of music" [15].

8. REFERENCES

[1] J. C. Risset, "Composing in real-time?" *Contemporary Music Review*, vol. 18, no. 3, pp. 31–39, 1999.

[2] S. Reich, "Music as a gradual process," in *Writings on Music, 1965-2000*. Oxford University Press, 2002.

[3] E. Selfridge-Field, *Beyond MIDI: the handbook of musical codes*. MIT press, 1997.

[4] R. Rastall, *The Notation of Western Music: An Introduction*. JM Dent and Sons, 1983.

[5] L. Treitler, *Reflections on musical meaning and its representations*. Indiana University Press, 2011.

[6] D. M. Randel, *The Harvard dictionary of music*. Harvard University Press, 2003, vol. 16.

[7] J. Drummond, "Understanding interactive systems," *Organised Sound*, vol. 14, no. 2, pp. 124–133, 2009.

[8] M. Stroppa, "Live electronics or live music? towards a critique of interactio," *Organised Sound*, vol. 18, no. 3, pp. 41–77, 1999.

[9] G. Wang, "A history of programming and music," in *The Cambridge Companion to Electronic Music*. Cambridge University Press, 2007.

[10] M. V. Mathews, J. E. Miller, F. R. Moore, J. R. Pierce, and J. Risset, *The technology of computer music*. MIT press, 1969, vol. 9.

[11] H. Taube, "Common music: A music composition language in common lisp and clos," *Computer Music Journal*, vol. 15, no. 2, pp. 21–32, 1991.

[12] B. Vercoe and D. Ellis, "Real-time csound: Software synthesis with sensing and control," in *Proc. of the Int. Conf. on Computer Music ICMC 1990*, Scotland, 1990, pp. 209–211.

[13] A. Cont, "Antescofo: Anticipatory synchronization and control of interactive parameters in computer music," in *Proc. of the Int. Conf. on Computer Music ICMC 2008*, Belfast, 2008, pp. 33–40.

[14] M. Puckette, "Pure data: another integrated computer music environment," in *Proc. of the Int. Conf. on Intercollege Computer Music Concerts*, Tachikawa, 1996, pp. 37–41.

[15] R. Dannenberg, "Music representation issues, techniques, and systems," *Computer Music Journal*, vol. 17, no. 3, pp. 20–30, 1993.

[16] D. Topper, "Rtcmix for linux (part 1)," *Linux Journal*, vol. 78, no. 1, p. 5, 2000.

[17] M. Good and G. Actor, "Using musicxml for file interchange," in *Proc. of the Int. Conf. on Web Delivering of Music, IEEE WEDELMUSIC-2003*, Leeds, 2003, p. 153.

[18] R. Perry, "The music encoding initiative (mei)," in *Proc. of the Int. Conf. on on Musical Applications Using XML - MAX 2002*, Milan, 2002, pp. 55–59.

[19] J. McCartney, "Rethinking the computer music language: Supercollider," *Computer Music Journal*, vol. 26, no. 4, pp. 61–68, 2002.

[20] M. Wright and A. Freed, "Open sound control: A new protocol for communicating with sound synthesizers," in *Proc. of the Int. Conf. on Computer Music -ICMC 1997*, Thessaloniki, 1997, p. 10.

[21] J. Aycock, "A brief history of just-in-time," *ACM Computing Surveys (CSUR)*, vol. 35, no. 2, pp. 97–113, 2003.

[22] I. Quinn, "Minimal challenges: Process music and the uses of formalist analysis," *Contemporary Music Review*, vol. 25, no. 3, pp. 283–294, 2006.