Greenhalgh, Chris and Benford, Steve and Hazzard, Adrian (2016) ^muzicode$: composing and performing musical codes. In: Audio Mostly 2016, 4-6 Oct 2016, Norrköping, Sweden.

^muzicode$: Composing and Performing Musical Codes

Chris Greenhalgh, Steve Benford, Adrian Hazzard
School of Computer Science, The University of Nottingham
Jubilee Campus, Wollaton Road, Nottingham NG8 1BB, UK
{chris.greenhalgh, steve.benford, adrian.hazzard}@nottingham.ac.uk

ABSTRACT
We present muzicodes, an approach to incorporating machine-readable 'codes' into music that allows the performer and/or composer to flexibly define what constitutes a code, and to perform around it. These codes can then act as triggers, for example to control an accompaniment or visuals during a performance. The codes can form an integral part of the music (composition and/or performance), and may be more or less obviously present. This creates a rich space of playful interaction with a system that recognises and responds to the codes. Our proof of concept implementation works with audio or MIDI as input. Muzicodes are represented textually and regular expressions are used to flexibly define them. We present two contrasting demonstration applications and summarise the findings from two workshops with potential users which highlight opportunities and challenges, especially in relation to specifying and matching codes and playing and performing with the system.

Author Keywords
Musical codes, performing, music information retrieval

ACM Classification
H.5.2 [Information Interfaces and Presentation] User Interfaces–Input devices and strategies, H.5.5 [Information Interfaces and Presentation] Sound and Music Computing.

INTRODUCTION
There is a rich tradition of embedding 'codes' within music, from J.S. Bach hiding his name within a sequence of notes [1] to more readily discernable leitmotifs in operas and films [2]. The advent of computational algorithms that can extract a diverse range of features from live musical performance with increasing accuracy raises the possibility of musical codes that can be recognized by computers as well as (to a greater or lesser extent) by humans. Such computer-regonisable musical codes could serve various purposes, for example triggering media, special effects or musical control during performance.

We introduce *muzicodes,* an approach to the composition of computer recognizable musical codes and their subsequent use during live performance. Instead of recognizing the final rendered form of a musical fragment (i.e., a waveform), muzicodes focuses on underlying musical structures embedded within a musical work. In our first realization, these take the form of sequences of notes whose pitches and rhythms may be more or less tightly specified, ranging from a precise sequence that must be played tightly to more general melodic or rhythmic shapes that can be flexibly embedded into a performance.

A musician may define any number of muzicodes for a given performance, associating each with a different action, for example, employing one to trigger the display of particular background images, another to change a backing track and yet another to change effects or instrument settings. The flexible nature of muzicodes means that musicians can play the same code in many different ways, allowing them to improvise around codes and choose whether to hide or reveal them to the audience. Thus, performers are free to decide exactly *when* to embed *which* code within their playing and also *how* to perform that code.

We begin by reviewing related work, based on which we then present a more complete description of the muzicode concept and approach. We then describe our initial implementation before presenting and reflecting on our iterative development process, including two demonstration applications and two workshops with potential users.

RELATED WORK
Musical codes
Many composers have played with musical cryptography in their compositions, including J.S. Bach, Shostakovich [1] and also Elgar who implanted messages to family and friends in his music [3]. Musical codes have also provided a compositional framework for large-scale musical works. For example, in Serial Composition (also known as 12 Tone Composition), the building block of a composition comprises a defined arrangement (i.e. row) of the 12 notes of the chromatic scale, which cannot be played out of sequence [4]. Compositional variation is achieved via a range of inversions, transpositions, and the distributions of the row across instruments.

In contrast to these 'hidden' codes are examples of musical codes that are more obvious to the listener. Leitmotifs were first used in the operas of Richard Wagner and are still liberally employed in film soundtrack scoring. Leitmotifs typically represent a character and may appear alongside their narrative representation or without, in which case they suggest the presence of a character without the need for them to be seen on stage or screen. An oft-quoted example can be heard in the film Jaws, where the ominous alternating semi-tone theme represents the presence of the approaching shark [2]. Leitmotifs are often re-composed and presented in many varied forms to represent the changing emotional states of its character at different points in the narrative.

In contrast to systems such as Chirp [5] that directly encode data in audio form our interest lies in codes that are composed and performed by humans and that can be flexibly embedded into performances in ways that range from cryptographic-style hiding to appearing as discernable motifs. Of course, we also require them to be recognisable by computers as we now consider.

Music recognition by computers
Music recognition is a large and complex endeavor spanning several fields of research. *Music information retrieval* (MIR) concerns the extraction and use of information from music, which

might be drawn from of an audio signal or a symbolic representation such as a traditional musical score [6]. A range of software tools are used to conduct MIR tasks such as the MatLab MIR Toolbox [7] and VAMP plugins [8] which focus on the extraction of a broad range of audio and musical features from audio signals, including pitch, rhythm, timbre, tonality, note onsets, segmentation, chord progressions and loudness. These form the basic building blocks for many MIR applications.

Automatic music transcription describes a process where an audio file is converted to another format, for instance a symbolic representation such as sheet music [9]; this kind of symbolic representation (and/or other features) derived from a live performance forms the first stage of our approach.

Audio fingerprinting concerns the recognition of a specific audio recording (version identification) [10], with the mobile app Shazam [11] being a prominent commercial example. In contrast, we are primarily concerned with live performances, each of which is unique. Query-by-humming relies solely on the melody features extracted from songs rather a specific recording [10]. Our interest lies in what a performer might *do* with such enabling technologies.

Music information retrieval tools are also employed in live performance settings, as the following examples illustrate. *Automatic score following* (audio-to-score) concerns the automatic synchronisation between a live audio or MIDI input (performer) and a pre-composed score [12]. This can then be used to track the nuances of a live soloist so as to align a computer generated accompaniment or sound manipulations (e.g. digital signal processing effects) as well as cueing extra-musical events such as lighting or visual media [13]. Rowe developed *Cypher* [14], a 'real-time interactive music system' that analyses incoming MIDI data and extracts key, chord, beat and phrase group features which are then used to generate musical accompaniment. *Sound Analyser* [15], a plugin for DAW environments, extracts real-time audio features from its input that can be mapped to Open Sound Control (OSC) messages to control live visuals. Rather than following an entire score, or generating a continuous accompaniment or control channel, we focus on distinct recognizable musical fragments that performers can embed at different points and in different ways within an overall performance.

THE CONCEPT OF MUZICODES

We now introduce the concept of muzicodes, describing how they are tailored to the context of musical performance, drawing out key requirements that arise from this and specifying a structure for muzicodes that addresses these.

The performance context

Muzicodes are intended for use in a live setting, such as a performance, rehearsal or lesson. We begin with the case of a single performer or instrument, illustrated in figure 1. During set-up the instrument, mic, direct line or MIDI source would be connected to the muzicode system's input. Typically at the same time, the muzicode system would be configured (perhaps also incorporating input from the composer) to support a particular 'experience', tailoring the operation and behaviour of the system for the purpose at hand. During setup, the system would also be linked to external output devices such as display or light controllers, other performers' devices or digital audio systems although it may also have its own output capabilities such as the screen of the device it is running on. Then while the performer plays (or sings) the muzicode system continuously monitors its input, checking for any of the codes associated with this experience, and triggering outputs accordingly.

In any given setting there might be more than one muzicode system, for example there might be one for the venue and/or each musician might have their own. In other cases the input to a single system may include several instruments, e.g. from a mic or mix. Where there are multiple muzicode systems these could communicate and coordinate with each other, perhaps requiring combinations of codes from different performers to trigger coordinated outputs.

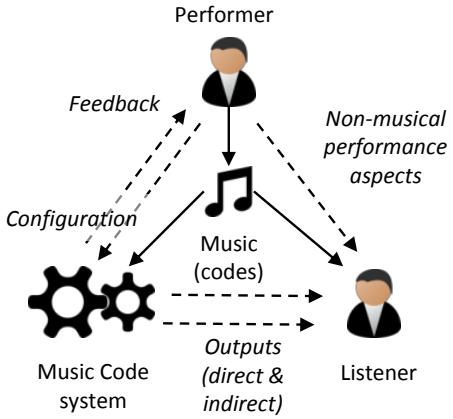

Figure 1: Muzicode performance setting

Requirements for muzicodes

This performance context fundamentally shapes what a muzicode should be. An inevitable complexity – and richness – of the situation is that the system and the performer may never agree perfectly about what is and is not a code. Consider a performer intending to trigger a particular action: they play the code, but every execution is slightly different (e.g. in timing, tone, or errors); noise is inevitably present (e.g. in the audio signal, timing or bit errors); and the algorithms used by the system are imperfect (e.g. mis-classifying or missing notes). There are also many situations (e.g. teaching) in which we might like the 'same' code to be triggered by different performers, each with their own idiosyncrasies, using different instruments, perhaps even completely different types of instrument, and in different settings.

Consequently, **codes need to be defined at an appropriate level of abstraction**. For live performance, this could be at the level of notes and rhythms, but not at the level of specific timbre or waveform, so that the 'same' code can be achieved at different times and by different performers. Codes might also be defined more abstractly or semantically, for example "a chord of D-major".

A second key consideration is for the composer and/or performer **to be able to choose to what extent to hide or reveal the codes for the audience**. The composer requires the flexibility to compose more or less obvious codes, for example, the use of obvious musical motifs versus more subtly hidden features. In turn, the performer should be able to decide how noticeable or subtle their playing of the codes might be, enjoying considerable latitude with regard to how they interpret them, perhaps including embellishing and improvising around them.

In addition, when defining codes **a balance has to be struck between the rates of false positives and false negatives**, i.e.

non-code music being heard as a code versus codes being missed. For example one possible approach might be for a performer to define an unusual musical 'prefix' to their codes so that they do not play a code by accident, and/or defining a set of similar codes or patterns to trigger the same action, to take account of common variations in how the system responds.

Finally, the performer should have the facilities **to incorporate the system's characteristics into the performance as a whole**, including artistic responses to its flaws and limitations. For example, the performer might develop strategies to cope with missed triggers, such as repeating riffs. Or the muzicode system itself might be presented as an explicit agency within the performance that sometimes needs to be 'persuaded' to act. Thus the idiosyncrasies of the live performance setting add an additional layer of complexity to the task of embedding codes into a musical work, when compared to the challenges of serial or leitmotif composition.

Muzicode structure and format

A muzicode specifies a sequence of notes that might be played at some point within a piece of music. In its simplest and most constraining form each note would be specified as a particular pitch and duration (and perhaps even tone and articulation) and the sequence played contiguously, with the notes sounding one after the other without any other intervening music. However, given the above requirements, we want to provide a richer set of options for greater flexibility:

- **Pitch and/or rhythm:** the notes might be specified by *pitch only*, so playable with any rhythm or timing, *rhythm only*, playable with any melody or on a percussion instrument, or a combination of the two.
- **Absolute or relative**: pitch and/or durational values might be specified *absolutely*, i.e. the right pitch or duration must be played, or *relatively*, for example based on the first note(s) of the code, so that it can be played in any key or at any speed.

The composer then has to be able to express the muzicode, which in our case they do concretely as a text string. This is broken down into two stages.

First the composer chooses how sequences of notes will be represented as text. Consider for example the following simple fragment of melody played at a steady 60bpm (see figure 2):

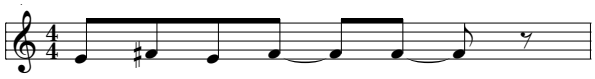

Figure 2: Sample melody

This might be represented textually as a sequence of: note names irrespective of octave or timing, "`E,F#,E,F#,F#`"; note names with timings in seconds (or beats), "`E/0.5,F#/0.5,E/0.5,F#/1,F#/1.5`" (where "`/0.5`" indicates a duration of 0.5s); notes in defined octaves with relative timing, "`E4/1,F#4/1,E4/1,F#4/2,F#4/3`" (where "`/1`" indicates a relative duration of 1); intervals relative to the final note irrespective of time, "`-2,0,-2,0,0`"; and so on. This gives the composer an initial level of control over how precise or flexible that muzicode will be. The composer could also specify how polyphony should be handled, for example whether simultaneous notes should all be included in the text or perhaps only the lowest or highest note.

Second, the composer identifies the *particular* textual string(s) that correspond to the muzicode they are defining. In the simplest case this would be a single string such as those above, which the system would look for. More generally the composer could specify a *pattern* that identified a set of similar or equivalent textual forms for a single muzicode. In principle there are many ways that this could be done; we are currently exploring the use of regular expressions, which correspond to the simplest class of automata [16], finite state machines. This opens up various possibilities when specifying muzicodes such as: using wildcards ('`.`'), ranges ('`[1-3]`') or choices ('`…|…`') to allow alternative notes or durations; forcing the code to be played at the beginning ('`^…`'), end ('`…$`') or entirety ('`^…$`') of a sequence; and allowing notes or groups of notes ('`(…)`') to be omitted ('`…?`'), repeated ('`…+`') or either ('`…*`').

PROOF OF CONCEPT SYSTEM

We have implemented a proof of concept prototype[1] that supports the muzicodes system described in section 3.3; the prototype's functional architecture is shown in Figure 3. It is implemented as a web-based client-server system, with the majority of the functionality implemented within the browser client. The system includes an *'experience' editor* (Figure 3, top-right), which allows the composer/performer to configure the system for a particular performance.

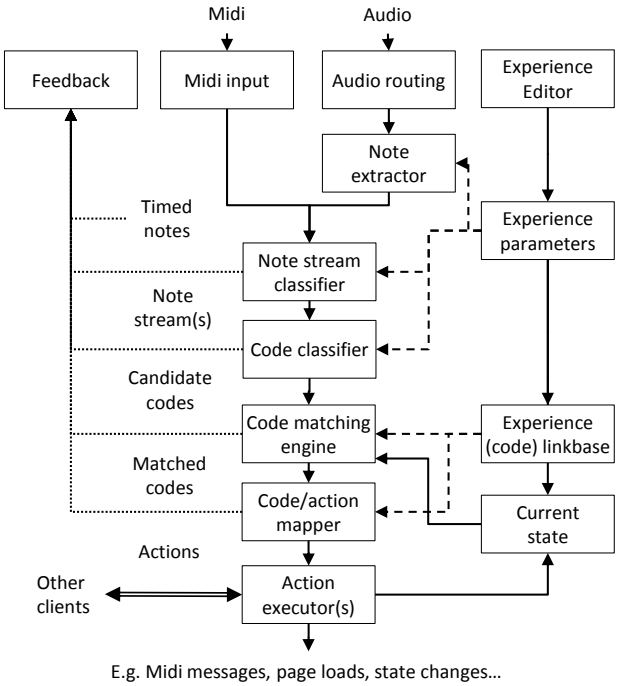

Figure 3: Muzicodes system architecture

Audio and *MIDI input* use the WebAudio API. Audio input depends on a *note extractor*, currently the Silvet note transcription VAMP plugin [10] in 'live' mode, to estimate *timed note* onsets (frequency, time and velocity); this plugin is executed on the server. Alternatively MIDI note events can be injected directly at this point. Notes are then assigned to one or more distinct *note streams*; this provides initial control over which notes are

[1] https://github.com/cgreenhalgh/musiccodes

considered together as potential codes. The current prototype divides sequences of notes into configurable pitch ranges and time intervals.

Note streams are then converted to *candidate codes* based on the specific musical features that have been chosen to define the codes (as described in section 3.3). To accommodate these different musical feature representations within the candidate codes four different *code formats* are currently defined and supported:

- Code format 'n': note name (pitch information) only. The stated pitch(s) will be recognised in any octave position. An example code might be 'A,C,E'.
- Code format 'no': note name and octave position. This code format uses two character positions, the first for the pitch name and second for a pitch's octave position, for example 'A3,C3,E3'.
- Code format 'mrle0': midi note relative to last note equal to 0. This format concerns interval relationships between pitches, i.e. relative pitch rather than absolute pitch. This is a numerical format, where positive (descending) or negative (ascending) values represent the interval distance from the final note of the stream, which is assigned a value of '0'. For example '-2,0,-2,0,0' would define the melodic fragment in figure 2.
- Code format 'crle4': count relative to last note equal 4. This fourth format concerns the duration between note onsets. This format offers options for rhythmic based codes where pitch information is ignored. As the VAMP plugin is not aware of tempo or time signature we chose to frame this format around the time between note onsets, while the note value (i.e. the duration that a note is sounding) is ignored. The relationship of durations is set by nominally assigning the value of '4' to the delay between the last two notes in the sequence. The example in would be represented in this format as '2,2,2,4'; the musical fragment has five notes but there are only four time intervals *between* those notes.
- Combined formats: beyond the four basic code formats, a combination of these formats can also be applied to further extend the complexity of a code. For instance the previous two code formats could be employed together (i.e. 'mrle0/crle4') to form a code that uses both interval and count relationships. Thus our example in would then be defined as: '-2/2,0/2,-2/2,0/4,0'. Alternatively, a format of 'no/crle4' would integrate pitch name, octave position and relative count between onsets.

As part of the experience, the composer/performer specifies the *experience 'linkbase'*, a set of mappings from codes to *actions*, including the selected code format for each code instance. In the current prototype codes are specified and matched as textual regular expressions. When a candidate code is matched, the resulting action is currently to output a MIDI message or publish a URL to listening applications (using socket.io); this URL might correspond to audio-visual media to display alongside the performance. The applications receiving these URLs can be on other networked devices, e.g. a laptop controlling a projected display, another performer's tablet or an audience member's phone.

Each experience also has a *current state*, a set of internal variables, which can enable and disable particular actions at different points in the performance. The current prototype also provides continuous *feedback* to the performer through a live timeline visualisation of recent notes and note streams, and a representation of which codes are being matched by the current notes (see Figure 4).

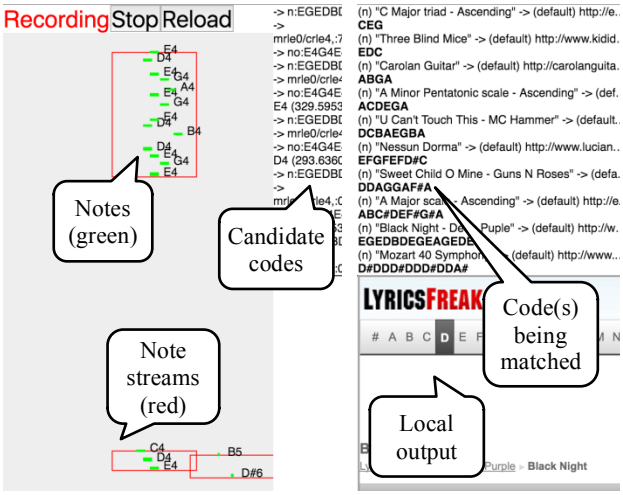

Figure 4: Muzicode feedback interface (annotated)

EXPERIENCE AND REFINEMENT

We are engaged in an ongoing process of iterative refinement and evaluation of the muzicodes concept and prototype, which has involved three iterations to date, successively focusing on: technical validation; demonstration applications; and initial user workshops. We briefly summarise these iterations, before presenting our reflections to date.

Iteration 1

The first iteration of muzicodes supported the core functionality of the muzicode system, and was used by the authors to assess the feasibility of the concept. This version demonstrated the technical integration of the client/server system, including the WebAudio APIs and VAMP plugin. It also supported feedback to the performer of the notes and note streams being 'heard' by the system (Figure 3, left). In this version there was no experience editor, and the system had to be configured by editing a textual (JSON) configuration file. Code formats 'n' and 'no' were implemented in this first iteration.

Iteration 2

The key addition for the second iteration was to provide visual feedback to the performer about muzicodes that were currently being matched (Figure 4, top right). As each note is received the system checks to see how much of each code pattern (regular expression) has been matched so far and whether the whole code can potentially be matched. This is fed back to the performer by highlighting (red) successfully matched portions of each code and by greying out those codes which can no longer be matched by the current note stream. This allows the performer to check if a code is being matched, and potentially to adapt their current playing within a single phrase. Code formats 'mrle0', 'crle4' and combination formats were implemented into this second iteration. Using the second prototype the authors, who are musicians themselves, developed two contrasting demonstration applications. We briefly describe each of these applications and our experiences with them.

Muzicodes for Irish tunes

Working with this second iteration our first demonstration was based around traditional Irish music [16], exploring how these musicians might employ muzicodes in order to draw on diverse digital materials on the Internet when learning and performing. We designed a suite of muzicodes to embody distinctive fragments of Irish tunes. In this case we chose to work with pitch and disregard timing information as Irish tunes tend to have distinct melodies but are harder to identify rhythmically due to sharing just a few rhythmic styles; consequently formats 'n' and 'no' were used exclusively. As a simple example, below is the first part of Egan's polka, a well-known beginner's tune (Figure 5). The following muzicode in format 'no' encodes its distinctive first two bars played in the register of a fiddle: "`F#5A5B5A5F#5A5B5A5`".

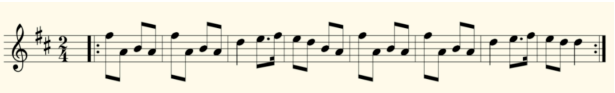

Figure 5: Elgan's Polka

Regular expression string matching can then be used to refine this basic code in various ways, for example we might employ the wildcard '`.`' to make the code recognizable when played in any register: "`F#.A.B.A.F#.A.B.A.`" (or equivalently we could use code format 'n': "`F#ABAF#ABA`").

We designed further codes to be playable by accompanying instruments such as guitars, citterns and bouzoukis. Our approach here was to specify the code as the notes of the chord played in sequence, for example a rising chord of D-major played across any range of octaves: "`^D.F#.A.D.$`". In practice, chords can have many different voicings and a muzicode designed to capture the high-level musical concept of a 'broken chord of D' would need to accommodate all of them. On the other hand, the system might respond differently to each voicing, offering accompanists a range of interactive possibilities even when playing simple chords.

We employed these codes to create a range of different experiences. The first was to support learning, where playing a memorable muzicode would connect to a tuition web page containing notes on the tune, sheet music (in standard and ABC formats [18]) and also an accompanying video that could be played along to when practicing. We created one variant for melody instruments and another for accompanists. The second experience was to support live performance on stage in which the musician could trigger playback of an accompanying video. The third was to support jamming at Irish sessions in which playing the code triggers the display of the name of the tune, answering the common 'what was that tune?' question.

Muzicodes for live performance

Our second demonstration prototyped a performance piece, where a soloist uses muzicodes to trigger prepared musical fragments at different points throughout the piece to accompany their instrumental performance. The piece used a MIDI keyboard controlling a virtual instrument in Apple Logic Pro X, with the MIDI notes also being fed directly into the muzicodes system, and twelve pre-composed musical fragments (hosted audio files) to be triggered by muzicodes. Typically, the codes we composed were relatively short strings and simple in their melodic and rhythmic construction. Different code representations were employed that used all of the code formats in conjunction with regular expressions. For instance absolute pitches separated with wildcards ('`.*`') permit for a code to be played in order, or with any number of additional pitch events placed in-between allowing it to be embedded into numerous other manifestations (e.g `"C#[0-9].*G#[0-9].*D#[0-9]"`). Other codes used interval matching rather than absolute pitches (i.e. code format 'mrle0'), enabling the soloist to perform a code in different keys or registers. The following code example shows a simple ascending melodic shape: `"^-9/[^,]*,-5/[^,]*,-2/[^,]*,0$"` (in this example a regular expression ('`[^,]*`') is used to match for any duration value against the interval). The score representation in Figure 6 shows the code with a starting pitch of 'B'.

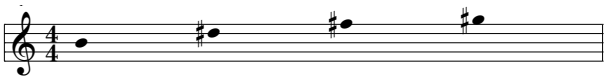

Figure 6: a simple ascending melodic shape

Our initial exploration revealed a growing desire to compose several different codes to trigger the same musical fragment. Typically, these sets of related codes used different code formats, for example, one code composed around a melody without duration data and another requiring just the duration data that would work with any pitch combination. In addition to offering musical variation, this also provided an alternative method of triggering an action should the performer experience difficulty performing one or the other codes. These emerging methods of code design reflect some of the traditional compositional approaches used in the variation of musical themes (i.e. leitmotifs, thematic development and serial composition), and when placed in a 'live' setting presented a flexible approach for triggering media that would not constrain the range of a performers musical expression.

We found that the composition of codes as triggers for accompanying musical elements often had their musical construction related to these elements. Several codes, for example, drew directly on the melodic motifs contained within their triggered fragments. Thus, the presentation of these codes foreshadowed the introduction of the triggered fragments by hinting at the music to follow. This observation highlights the underlying 'musical' nature of the muzicode approach.

Iteration 3

Based on our experience in developing the demonstrations we created a third prototype. The main changes in the third iteration were: the introduction of a browser-based 'experience' editor, to make editing experiences simpler; support for less technical users to install and update the prototype; support for multiple output clients with different actions (URLs) being sent to different clients (e.g. a performer view vs an audience view); and support for state within an experience, giving more control over when codes could/couldn't be triggered. This third prototype was used as the basis for two workshops with potential users, which are described in more detail in the next section.

User Workshops

Using the third prototype system we ran two workshops with potential users. Each participant brought their own instrument and laptop. Each workshop lasted approximately three hours, and comprised: informed consent; support with installing the muzicodes prototype and connecting their instrument to it; an

introduction to muzicodes, the system, code formats and regular expressions; individual time to test out and ideally create their own muzicodes; a plenary demonstration of what they had done; and finally a plenary discussion.

The first workshop involved 10 participants (8 male, 2 female) recruited by email and word of mouth from staff and research students at the authors' institution. The second workshop involved 5 participants (4 male, 1 female) recruited by email and word of mouth from staff and research students of another institution with a specialism of computer music. 11 of the participants (including all of those in the second workshop) had prior experience of programming and (at least to some extent) regular expressions whereas four did not. The instruments the participants played were: electric guitar (4), acoustic guitar (2), bass guitar (1), mandolin and electronic bagpipes (1), MIDI keyboard (3), synclavier (1), synth with audio output (1), whistle (1) and augmented grand piano (1). All participants were amateur musicians, ranging from relatively novice (learning) to extremely experienced (regularly performing publicly at gigs or concerts), and played a wide range of musical styles.

Findings

We begin by summarizing the main successes and challenges in the workshops. All participants were able to run the muzicode system and – is some cases with support – control it using their instrument. The participant who initially attempted to use the digital bagpipes abandoned them in favour of the mandolin, because the note transcription appeared to be extremely poor with the bagpipes (which have quite a distinctive tone and drone). Two participants also switched from acoustic guitar (using a microphone) to electric guitar to increase the reliability of note capture in the shared workshop setting.

All participants successfully created and triggered at least one muzicode of their own, although some participants found this very difficult. Some participants found *specifying* muzicodes difficult, due to subtle mismatches between the code text and the type of code they had selected, or due to misunderstanding the exact syntax of regular expressions. Some participants found *triggering* muzicodes difficult, due to difficulties in notes being consistently recognized due to overtones or shortness of notes when using audio input, and difficulties in performing rhythms with the precision required by the system. Most participants started by creating simple melody-only codes without any regular expression features and some participants (a minority) extended or elaborated their initial codes with (e.g.) wildcards or alternatives or tried other code types.

Potential applications of muzicodes which were prototyped or discussed by the participants included: triggering a backing track to play long with; summoning the musical score from stating the introduction of the piece being played; triggering a synchronized sound effect at a key point in a performance without 'leaving' their instrument; changing effects or patches (being a "musician" not just a "button pusher"); generating Open Sound Control (OSC) messages to control external devices; detecting and rewarding "correct" playing, e.g. in teaching; detecting and responding to specific gestures which playing, e.g. depending on precisely how the person presses the piano keys; and as a "partner" in a live improvisation.

In addition to specific clarifications and usability issues participants also had a range of suggestions for enhancing and extending the muzicodes system or concept. A common request was for alternative means of entering codes, for example by 'playing in' an example phrase or by editing a DAW-style piano-roll view within the system. Alternative ways of filtering note groups were also discussed, for example using frequency ranges correspond to a particular instrument or register, or filtering out very quiet notes as likely overtones. For those using the audio input there was generally a wish to reduce the latency (which was up to 1 second between starting a note and seeing it on the interface) and increase the accuracy of the note detection. It was suggested that the note matching and regular expression functionality might be done at the level of notes rather than the textual representation. There was also discussion of whether the system could "learn" regular expressions (or equivalent) for codes if the system were given several examples of the "same" phrase. Finally, there was significant discussion in both workshops about ways of adjusting the "looseness" of a code, i.e. being able to adjust how flexibly performance of a code would be matched. This included various facets such as variations of tempo, feel, grace notes and ornamentation, slides, missed or extra notes, and errors in playing.

Discussion

The workshops included participants with a range of technical and musical experience, but they were all able to use and create muzicodes at a basic level, and some participants were able to develop more complex codes and test the boundaries of the system. The basic concept made sense to all of the participants, and a range of possible applications were identified, suggesting that there is merit in the muzicodes concept. We expand briefly on two key themes of codes and performing.

Code Matching and Looseness

Our approach of using regular expressions to govern how input streams of recognised notes are matched to target codes is potentially powerful, giving a common framework in which codes can be elaborated with alternatives, ranges and repetition. For example, the use of alternatives and wildcards allowed codes to be embedded within extended or more florid melodic and rhythmic statements, and dealt with variations in playing such as embellishments. However, we acknowledge that as Computer Scientists we are comfortable with the approach and notation of regular expressions whereas many musicians are not. Therefore a key question for future research is whether and how this approach of flexible matching can be harnessed by musicians. While there are some forms of music that express music as text strings (including the ABC notation used for traditional music [18]) future work needs to explore whether this will be tractable for musicians and how these can then be embedded into future compositional and performance interfaces. Immediate technical steps would be to lift the domain of the regular expressions from text characters (as in the current prototype) to notes. Syntax-driven editing and the ability to enter codes by example would allow anyone to safely enter and experiment with regular expression enhanced codes.

A related key theme that emerged from the workshops is that of the "looseness" with which codes are matched. This is a rich and ambiguous musical concept which needs to be further explored and unpacked. The current prototype provides fragmented control over aspects of looseless through the choice of code format (i.e. pitch and/or rhythm, absolute or relative), and the use of regular expressions such as wildcards and start/end markers can also

extend the number of matched representations of a code with its associated action.

The integration of regular expressions and different facets of looseness might be more musically framed, perhaps as a set of checkbox options or sliders on the user interface that allow the performer/composer to more explicitly manipulate these elements within their codes. For example, these could take the form of 'make code transposable' (i.e. system converts to a relative pitch format) or 'distribute code across an extended phrase' (i.e. system inserts wildcards), or 'reduce accuracy of count' slider when using the 'crle4' rhythm format.

Performing and Adapting

Both the workshops and our demonstrations reveal how players have to learn to perform the codes. The robustness and success of matching a code in a 'live' setting is dependent on a number of factors. MIDI input is more responsive and delivers a "cleaner" signal (as the VAMP plugin is by-passed), whereas audio inputs from an acoustic instrument can introduce undesirable artefacts such as harmonic overtones or ambient environmental sounds recognized as note events by the VAMP plugin, which were particularly prominent on stringed instruments. This results in a significant increase in false negatives, i.e., codes not triggering when desired, with performers having to adapt to 'play down' to the system, slowing down their playing, separating out notes and sometimes damping strings rather than letting notes ring. Codes based on note durations (i.e. rhythms) were particularly hard to achieve consistently, as they required the performer to be highly accurate in their execution of a rhythmic pattern and the note extraction stage did not detect very short notes. On the other hand, the pitch-only codes can be played with rhythmic flexibility, for example introducing a 'lilt' to slow playing that makes them still broadly paletteable.

The real-time visual feedback that is available through muzicodes user interface has proved essential to being able to play codes reliably but also as expressively as possible. This visual feedback shows which notes are being recognized, enabling the musician to adapt their playing so as to strike a balance between expressivity and reliability and also to become aware of the effect of background noise such as voices and other instruments that can cause problems in busy environments. Further work is required on how best to present this visual representation of musical events in a performance setting, which is often a complex environment that often requires a performer to attend and respond to a number of different sensory events. Further work is also required to refine the note recognition as much as possible, and to allow greater looseness in the matching of rhythmic codes in particular. However, complementary to this, more research is needed – through longer engagements by performers with the system – to explore whether and how players can adapt their playing style and effectively integrate the muzicodes system within a performance setting which also includes their choices of communication protocol integration (e.g. MIDI, OSC, URL, DMX). Future work should also explore how composers can balance the various factors that appear to make for a good muzicode – distinct within a particular musical context, recognizable to both humans and computers, and being open to embellishment – with the feedback that will enable musicians to perform them in a reliable but also suitably expressive manner.

CONCLUSIONS

We have just begun to explore the creative possibilities of muzicodes, but we believe that they show great potential. Their embedding within a live performance, rehearsal or learning setting creates a rich space of interaction between the musician, the muzicode system and other listeners. As with any instrument, part of developing a performance that uses muzicodes relies on the performer learning how to get the best out of the system by preparing and rehearsing a performance, and considering the idiosyncrasies of the codes, instrument(s) and signal input. From the initial workshops we have identified priorities for further refinement of the prototype in the entering and editing of codes and their "looseness", which we hope to test in a further round of workshops. In addition, we aim to explore the performative aspects of muzicodes more fully through extended engagements leading to use in performance.

ACKNOWLEDGEMENTS

This work was supported by the UK Engineering and Physical Sciences Research Council [grant numbers EP/L019981/1, EP/M000877/1].

1. REFERENCES

[1] A. Shenton. 2008. *Olivier Messiaen's system of signs: notes towards understanding his music*. Ashgate Publishing, Ltd.

[2] J. Wingstedt, S. Brändström, and J. Berg. 2010. Narrative music, visuals and meaning in film. *Vis. Commun.* 9, 2, 193–210.

[3] E. Sams. 1970. Elgar's Cipher Letter to Dorabella. *Music. Times*, 151–154.

[4] Perle, George. 1972. *Serial composition and atonality: an introduction to the music of Schoenberg, Berg, and Webern*. Univ of California Press.

[5] Chirp. http://www.chirp.io/ verified 2016-01-21.

[6] J. S. Downie. 2003. Music information retrieval. *Annu. Rev. Inf. Sci. Technol.* 37, 1, 295–34.

[7] O. Lartillot and P. Toiviainen. 2007. A Matlab toolbox for musical feature extraction from audio, In *International Conference on Digital Audio Effects*, 237–244.

[8] C. Cannam, M. Mauch, M. E. Davies, S. Dixon, C. Landone, K. Noland, M. Levy, M. Zanoni, D. Stowell, and L. A. Figueira, 2013. MIREX 2013 entry: Vamp plugins from the centre for digital music. MIREX.

[9] E. Benetos and S. Dixon. 2012. A Shift-Invariant Latent Variable Model for Automatic Music Transcription. *Comput. Music J.* 36, 4 (Dec.2012), 81–94.

[10] J. Salamon, E. Gomez, D. P. Ellis, and G. Richard. 2014. Melody extraction from polyphonic music signals: Approaches, applications, and challenges. *Signal Process. Mag. IEEE* 31, 2–134.

[11] A. Wang. 2006. The Shazam music recognition service. *Commun. ACM* 49, 8, 44–48.

[12] A. Jordanous and A. Smaill. 2009. Investigating the Role of Score Following in Automatic Musical Accompaniment. *J. New Music Res.* 38, 2 (June 2009), 197–209.

[13] C. Joder, S. Essid, and G. Richard. 2011. A Conditional Random Field Framework for Robust and Scalable Audio-

to-Score Matching. *IEEE Trans. Audio Speech Lang. Process.* 19, 8 (Nov. 2011), 2385–2397.

[14] R. Rowe. 1992. Machine Listening and Composing with Cypher. *Comput. Music J.* 16, 1, 43.

[15] A. M. Stark. 2014. Sound Analyser: A Plug-In For Real-Time Audio Analysis In Live Performances And Installations. In *Proceedings of New Interfaces for Musical Expression* (NIME), London, 2014.

[16] John E. Hopcroft, Rajeev Motwani and Jeffrey D. Ullman. 2013. *Introduction to Automata Theory, Languages, and Computation* (3rd ed.). Pearson. ISBN:1292039051.

[17] Steve Benford, Peter Tolmie, Ahmed Ahmed, Andy Crabtree, Tom Rodden. 2012. Supporting traditional music-making: designing for situated discretion. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work* (CSCW '12). ACM, New York, NY, USA, 127-136.

[18] ABC Notation, http://abcnotation.com verified 2016-05-20