SNAPSHOTS: NEW POSSIBILITIES FOR SOCIAL DIGITAL MUSIC-MAKING ARISING FROM THE STORAGE OF HISTORY

Samuel Aaron
University of Cambridge
Computer Laboratory
Cambridge, England
sja55@cam.ac.uk

Jenny Judge
University of Cambridge
Centre for Music & Science
Cambridge, England
judge.jennifer@gmail.com

ABSTRACT

In this paper, we propose that digital music systems need to be designed with a view to collaborative, social, ‘non-cochlear’ musical engagement. In order for digital music to become more truly social, performers and audiences alike need to be able to see to some extent inside the ‘black box’ of the systems being used. This is particularly important in the case of electronic improvisation, where a lack of understanding of the processes being manipulated and how they are reflected in the sonic result can lead to a sense of alienation for audiences and performers. We propose that the capacity for storage of history represents a powerful tool for the design of interactive musical systems. We introduce ‘Snapshots’, a fully generic timeline recording and retrieval system, which is capable of storing independent named streams of OSC messages, as well as a novel query language for the bespoke specification of interesting stream subsets or streams. ‘Snapshots’ can be used with any physical interface that is OSC-compatible, making it compatible with embodied approaches to digital sound control. The system could also be used to create semantic visualizations of the patterns being used in the manipulation of the data structures. We argue that this is an important step towards making digital more comprehensible, communicative and ‘non-cochlear’, paving the way for a social and collaborative digital music environment.

1. MUSIC AS SOCIAL BEHAVIOUR: TOWARDS ‘NON-COCHLEAR’ MUSIC

1.1. Snapshots

The capacity of digital musical systems to store history is a potentially powerful tool that could be deployed in an attempt to set the scene for meaningful musical interaction in digital contexts. We introduce ‘Snapshots’, an environment in which the history of a performance is stored as streams of OSC events. Queries result in ‘snapshots’ of immutable sequences of events, which can be manipulated and used as generic input to other systems. Our system allows for greater power in terms of event recall, but it also provides us with a way of adding coherent structure to improvisations by referencing and manipulating past events, thereby increasing the structural intelligibility of improvised digital performances, for both performers and audiences. Since Snapshots operates on streams of OSC events, it is compatible with any physical interface that is OSC-compatible, increasing the possibilities for embodied communication with the audience and with other performers. Snapshots could also be used to create semantic visualizations of the data structures in use, which would help the audience (and, indeed, the collaborating performers) to see further into the ‘black box’ of what is happening on stage.

For example, consider the ability to record the sequence of interactions of a pianist during a given performance. If the timing, press, release, velocity and pitch of each on-screen event are recorded into a unique stream, then this event can be treated as standard programming data to be processed and manipulated. Using Snapshots, it is possible to specify a query which will return a specific subset or riff embedded in the larger performance. This subset then may be subsequently manipulated via any arbitrary function; the resulting data set may then be used for a variety of purposes: for instance, as a control signal for audio processing modules or seed data for visualisations. Figure 1 shows an example of this kind of data manipulation: the melodic contour of a past event could be queried and used to modulate a high-pass filter.

1.2. What do we mean by ‘music’?

Attempts to define music have met with considerable difficulty. The standard ‘dictionary definitions’ of music, which generally take it to be an art form in which sounds are combined in order to express meaning with a view to beauty or form, work well for Western classical music, but run into difficulty when we consider other types of music. In particular, the intimate link between music and culture in most world musics, as studied by the discipline of ethnomusicology, calls into question the notion that music should be seen as an art form. This is perhaps best understood in terms of how the cultures that practice it actually use it in social interaction.

1.3. Why should we pay attention to the music of other cultures?

Bruno Nettl [11] notes that ‘standard’ definitions of music lead us to expect a certain level of formal complexity, which may rule out tribal chanting, for instance, from consideration as a type of music. The notion that music is made ‘with a view to beauty’ may lead us to expect that all music must sound pleasant. This is of course problematic, since beauty, too, is notoriously difficult to define, and may well be fundamentally culture-dependent. Moreover, it is difficult to say whether or not other cultures actually have the concept of ‘music’ as we understand it. Until relatively recently, as Nettl points out, ethnomusicologists took it for granted that all cultures had the concept of music. However, the Hausa of Nigeria have no term for music in general. Some native American societies, though they do have concepts for specific instances of what we would call music, have no general concept to tie all instances of it together under one unified category. Perhaps, then, music is not a universal concept at all – perhaps no universal definition of it is possible.

1.4. Music as social life

Thomas Turino takes up this idea, arguing that music is best understood as a type of social life, rather than as an abstract cultural artefact [14]. The main thesis of his book is that the social nature of music, rather than the details of which process could simultaneously be shown via semantic visualisations, in order to enable both the performers and the audience to achieve a more fine-grained understanding of the unfolding structure of the improvisation.

We propose that this is a dynamic new line of inquiry in the attempt to make digital musical performance more truly social, collaborative and meaningful for performers and audiences alike. Before describing in detail the architecture of Snapshots, it is important to discuss the motivations for the design of this system, which stem from concerns raised in ethnomusicology about the expressive and social nature of music-making. Any digital musical system must begin by interrogating what ‘music’ means to begin with; it is to this question that we now turn.

1.5. Different ‘kinds’ of musical performance: Pre-scientific versus participatory music

Turino’s distinction between prescriptive and participatory live music highlights another sense in which traditional dictionary definitions of music leave something out. Most western art music is, he argues, prescriptive: the musicians are doing, the audience merely listening. There...
SNAPSHOTS: NEW POSSIBILITIES FOR SOCIAL DIGITAL MUSIC-MAKING ARISING FROM THE STORAGE OF HISTORY

Samuel Aaron
University of Cambridge
Computer Laboratory
Cambridge, England
sja58@cam.ac.uk

Jenny Judge
University of Cambridge
Centre for Music & Science
Cambridge, England
judge.jennifer@gmail.com

Abstract

In this paper, we propose that digital music systems need to be designed with a view to collaborative, social, ‘non-cochlear’ musical engagement. In order for digital music to become more truly social, performers and audiences alike need to be able to see to some extent inside the ‘black box’ of the systems being used. This is particularly important in the case of electronic improvisation, where a lack of understanding of the processes being manipulated and how they are reflected in the sonic result can lead to a sense of alienation for audiences and performers. We propose that the capacity for storage of history represents a powerful tool for the design of interactive musical systems. We introduce ‘Snapshots’, a fully generic OSC-compliant multi-threaded storage and retrieval system, which is capable of storing independent named streams of OSC messages, as well as a novel query language for processing streams of OSC events, it is compatible with any physical interface that is OSC-compatible, making it possible for embodied approaches to digital music control. The system could also be used to create semantic visualizations of the data structures in use, which can be manipulated and used as generic input to other systems. Our system allows for greater power in terms of event recall, but also provides us with a way of adding coherent structure to improvisations by referencing and manipulating past events, thereby increasing the structural intelligibility of improvised digital performances, for both performers and audiences. Since Snapshots operates on streams of OSC events, it is compatible with any physical interface that is OSC-compatible, increasing the possibilities for embodied communication with the audience and with other players. Snapshots could also be used to create semantic visualizations of the data structures in use, which would help the audience (and, indeed, the collaborating performers) to see further into the ‘black box’ of what is happening on stage. For example, consider the ability to record the sequence of interactions of a pianist during a given performance. If the timing, press, release, velocity and pitch of each individual interaction or event are recorded into a unique stream, then this event can be treated as standard programming data to be processed and manipulated. Using Snapshots, it is possible to specify a query which will return a specific subset or riff embedded in the larger performance. This subset then may be subsequently manipulated via any arbitrary function; the resulting data set may then be used for a variety of purposes: for instance, as a control signal for audio processing modules or seed data for visualisations. Figure 1 shows an example of this kind of data manipulation: the melodic contour of a past event could be queried and used to modulate a high-pass filter, thus changing the details of which process could simultaneously be shown via semantic visualisations, in order to enable both the performers and the audience to achieve a more fine-grained understanding of the unfolding structure of the improvisation.

We propose that this is a dynamic new line of inquiry in the attempt to make digital musical performance more truly social, collaborative and meaningful for performers and audiences alike. Before describing in detail the architecture of Snapshots, it is important to discuss the motivations for the design of this system, which stem from concerns raised in ethnomusicology about the expressive and social nature of music-making. Any digital musical system must begin by interrogating what ‘music’ means to begin with; it is to this question that we now turn.

1.2. What do we mean by ‘music’?

Attempts to define music have met with considerable difficulty. The standard ‘dictionary definitions’ of music, which generally take it to be an art form in which sounds or tones are combined in order to express meaning with a view to beauty or form, work well for Western classical music, but run into difficulty when we consider other types of music. In particular, the intimate link between music and culture in most world musics, as studied by the discipline of ethnomusicology, calls into question the notion that music should be seen as an isolated art form at all — perhaps it is best understood in terms of how the cultures that practice it actually use it in social interaction.

1.3. Why should we pay attention to the music of other cultures?

Bruno Nettl [11] notes that ‘standard’ definitions of music lead us to expect a certain level of formal complexity, which may rule out the possibility of individuals and groups expressing themselves in as simple a form as possible, once again assuming that music is practised socially all over the world. Maybe Western art music is actually unrepresentative: perhaps to make generalisations about music based on consideration of Western art music alone is to make a false step. For Turino, playing with others allows him to transcend his own experience and feel united with the others in the group: ‘I think that what happens during a good performance is that the multiple differences among us are forgotten and we are fully focused on an activity that emphasized our sameness — of time sense, of musical sensibility, of musical habits and knowledge, of patterns of thought or action, of spirit, of common goals — as well as our direct interaction and concerted frame of musical performance that sameness is felt as total.’ (Turino, p. 18).

This unifying aspect of music-making has been discussed by other authors (for example [5]) as a possible reason for music’s being a target of evolutionary selection at the group level, the idea being that a group that practices music together will be more united and hence more successful. This feeling of total synchrony with fellow members of a musical group is quite elusive: in many cases, it is the motivation for musicians seeking new groups to perform with. In addition, Turino points to the possibility that some are more sensitive to this synchrony than others: one member in a band may feel that everyone is well-synchronised, while the other band members may all feel that he is in fact out of time. “Like the good human relationships they index,” says Turino, “good musical relationships are difficult to achieve and require continual work to sustain.” (Turino, p. 20) The pursuit of this feeling of synchrony with others is, for many musicians, a primary motivation to keep playing.

1.5. Different ‘kinds’ of musical performance: Pre-sentational versus participatory music

Turino’s distinction between presentional and participatory live music highlights another sense in which traditional dictionary definitions of music leave something out. Most western art music is, he argues, presentional: the musicians are doing, the audience merely listening. There
is a physical divide between the performers and the listeners. Participatory music, on the other hand, has no such clear divide between performers and listeners; all present are involved in simultaneously doing and listening. This type of music-making is less on music as 'art' as on the social dimension of the act of music-making itself. Non-western music is often participatory rather than pre- sentational. Of course, the distinction is blurred in some cases: for example, it seems that local musical improvisation performances, such as those found in jazz, might be more participatory than presential, at least where the musicians are concerned. However, it is important that musicians must focus on each other's actions as well as their own, in or- der to be able to react to unscripted changes as the music progresses. However, it seems that this social focus appar- ent in participatory music is neglected by dictionary def- initions of music, perhaps because of a latent assumption that music is to be understood as presen- tational.

1.6. Composition and improvisation: two ends of the same process

Performance and listening, then, may be understood as different facets of the same activity, at least in the case of participatory music-making. Nettl makes a similar point about composition and improvisation, arguing that the two are "opposite ends of the same process" (Nettl, p. 30). He discusses the various ways in which some of the 'great' composers are said to have worked. Schubert, for in- stance, was known to scribble songs on the backs of menus in cafes. Mozart was similarly adept in turning out compositions quickly and with ease. Beethoven and Brahms, however, seemed to have a much more laboured composi- tion process. Brahms worked on his first symphony for several years. Beethoven agonized over his 9th symphony for two decades. Nettl argues that Schubert or Mozart's quick compositions were effectively improvisations without the real-time performance dimension. In a similar vein, a great improviser may only be able to be as fluent as she is due to hours spent digesting the different styles of other performers, familiarity with the genre in question and practice on a particular instrument. North Indian sitarists, for in- stance, can create a new performance on the spot, but only because they practice exercises for hours every day, and have an extensive group of software programmes commonly used in live performance. Electronic music, as a result, is seen by many as 'individualistic' given this lack of a performance-driven, 'non-cochlear' social dimension. The laptop orchestra, in its various forms, is an attempt to combat this individual- ism by creating an environment where electronic musicians can play together as part of an ensemble. The motivation behind the development of such ensembles can be that the performers need for a more non-cochlear digital music: one that is grounded in real-world interactions and embodied gestures, as opposed to being focused purely on the sonic output of a system. Such ini- tiatives highlight a growing interest in making digital mu- sic more social and communicative, both for performers and listeners; to a large extent, it is certainly a step in the right direction.

2.3. Laptop orchestras and black boxes: alienated audi- ences

However, even in the case of laptop orchestras, the perfor- mance of live electronic music can often be an alien- ating experience for the audience. The lack of an in-depth understanding of the music, to use Benjamin's term, is relocated rather than dispersed by technological reproduction. Returning to Turino's concerns for privilging the social in music, we may ask whether digital music lends itself to a social understanding. Thornton's discussion of 'club cultures' supports this idea; she argues that discos, clubs and raves evolved into new sites for music listening, where technol- ogy was made audible by enacting the idea of the music. Listening to digital music certainly seems to be something that groups of people do together in social contexts, where the enjoy- ment of the music is linked fundamentally to the group ex- perience. Cochrane [4] discusses the importance of joint attention in musical listening, arguing that it is our aware- ness of others in a group listening situation that constitutes much of the aesthetic appeal of the act of listening itself. Digital music, it would seem, lends itself to joint atten- tion, and hence a social aesthetic appreciation, in much the same way as any other kind of music.

2.4. What about electronic music improvisation? Is a truly social creativity possible?

Moving away from purely presenational music, we might ask what the state of affairs is when it comes to participa- tory music-making – for instance, by considering the na- ture of the experience of the performers themselves. This becomes particularly relevant when considering the pos- sibility of improvising with digital music, as well as col- laborative composition, which we consider as occupying 'opposite ends' of the same process, following Nettl. In the case of digital music ensembles, much the same prob- lems occur as those outlined above with regard to presen- tational music: that is, because of the presence of 'black boxes' in the ensemble, it can be difficult for the listener to understand what the others are doing. In traditional ensembles, the technologies are stable: the symphony or- chestra, for instance, has been much the same for the last hundred years. Even though there have been some changes, the pace of the evolution of the symphony orchestra has been extremely slow. For that reason, it is possible for each player to have a good under- standing of the ranges, styles and parameters involved in each instrument. With digital ensembles, however, any combination of software and hardware could be in use at any given time. Even if each player happens to know the combination of software and hardware that is in use by each of the other players, there is much less of a chance that she will have an in-depth, fine-grained understanding of the parameters of each of those combinations.

2.5. Lack of constraints in electronic improvisation: a barrier to creativity

The nature of electronic music, in its open-endedness, also hampers communication among players. Jazz players can improvise fluently because they understand the pre-given harmonic structures of the piece, know the general struc- ture of an improvisation in that particular genre and have an in-depth embodied knowledge of their instrument. In other words, they know the rules, in much the same way as the North Indian sitarist does. Johnson- Laird [8] argues that this knowledge reduces the load on their working memory, allowing them to dedicate more resources to the improvisation. Performance researchers can face duality a situation where there are no pre-given struc- tures, harmonic or otherwise; there is no generally-agreed- upon structure of the improvisation; and the instability of NIMEs means that they have a limited embodied understanding of their particular interface. It is no wonder, then, that electronic improvisation poses such a difficulty for performers and listeners alike.

2.6. How might we make digital music more comprehensi- ble and more social?

Leman [9] argues for a greater emphasis on embodied control of digital technology, in order to bridge this gap in understanding. Much of the current research in NIMEs...
is a physical divide between the performers and the listeners. Participatory music, on the other hand, has no such clear divide between performers and listeners; all present are involved in simultaneously doing and listening. This type of music-making is less concerned with music as ‘art’ as on the social dimension of the act of music-making itself. Non-western music is often participatory rather than presentational. Of course, the distinction is blurred in some cases; for example, it seems that in some traditional improvisation performances, such as those found in jazz, might be more participatory than presentational, at least where the musicians are concerned, in the sense that the musicians must focus on each other’s actions as well as their own, in or- der to be able to react to unscripted changes as the music progresses. However, it seems that this social focus appar- ent in participatory music is neglected by dictionary def- initions of music, perhaps because of a latent assumption that music is to be understood as presentational.

1.6. Composition and improvisation: two ends of the same process

Performance and listening, then, may be understood as different facets of the same activity, at least in the case of participatory music-making. Nettl makes a similar point about composition and improvisation, arguing that the two are “opposite ends of the same process” (Nettl, p. 30). He discusses the various ways in which some of the ‘great’ composers are said to have worked. Schubert, for in- stance, was known for scribbling songs on the back of menus in cafés. Mozart was similarly adept in turning out com-positions quickly and with ease. Beethoven and Brahms, however, seemed to have a much more laboured compo- sition process. Brahms worked on his first symphony for years. Beethoven agonized over his 9th symphony for two decades. Nettl argues that Schubert or Mozart’s quick compositions were effectively improvisations without the real-time performance dimension. In a similar vein, a great improvisator may only be able to be as fluent as she is due to hours spent digesting the different styles of other improvisers, familiarity with the genre in question and the ability to connect with other musicians. Non-Western music is often participatory rather than presentational, at least where the musicians are concerned, in the sense that the musicians must focus on each other’s actions as well as their own, in order to be able to react to unscripted changes as the music progresses. However, it seems that this social focus apparent in participatory music is neglected by dictionary definitions of music, perhaps because of a latent assumption that music is to be understood as presentational.

2. 2.1. Can listening to digital music be social?

Scepticism about digital music can be, at least in part, traced back to Walter Benjamin’s influential essay ‘The work of art in the age of mechanical reproduction’, first published in 1936. [1] For Benjamin, technological repro- duction of art undermines the artwork itself, dispersing its ‘aura’, or its fundamental aesthetic appeal. Although digital music wasn’t established in Benjamin’s day to the extent that it is now, he would almost certainly have ar- gued that digital music isn’t ‘special’ in the way that more traditional artworks are, due to its capacity to be in- finitely reproduced, and the fact that it is frequently not performed live. Thornton challenges this idea [13]. That the music is neither performed live nor recorded in high- fidelity – that is, as a document of what was at some point performed live – does not, for Thornton, compromise its authenticity. Thornton argues that the aura of the music, to use Benjamin’s term, is relocated rather than dispersed by technological reproduction. Returning to Turino’s concerns for privileging the social in music, we may ask whether digital music lends itself to a social understanding. Thornton’s discussion of ‘club cultures’ supports this idea; she argues that discos, clubs and raves evolved into new sites for music listening, where technol- ogy and listening to digital music certainly seem to be something that groups of people do together in social contexts, where the enjoy- ment of the music is linked fundamentally to the group ex- perience. Cochrane [4] discusses the importance of joint attention in musical listening, arguing that it is our aware- ness of others in a group listening situation that constitutes much of the aesthetic appeal of the act of listening itself. Digital music, she argues, lends itself well to joint atten- tion, and hence a social aesthetic appreciation, in much the same way as any other kind of music.

2.2. What about performance of digital music?

Performance of digital music, however, seems to be some- what more problematic when it comes to social engage- ment. Electronic music tends to be produced by indivi- duals in studios; often, the heavy reliance on record- ing technology means that the music doesn’t lend itself to its performance easily. Electronic music doesn’t seem to be social in the same way that, say, jazz improvisa- tion is, since it often involves only one musician, who composes ‘offline’ by building a piece of music incremen- tally to then perform a piece that was presumably composed offline. Of course, ‘classical’ composers mostly composed offline too, but the difference is that classical music is performed live by musicians in real time. With electronic music, the distinction is that the link between the audience and the performer – the establishment of fellowfeeling through the music – may be compromised.

2.3. Laptop orchestras and black boxes: alienated audiences

However, even in the case of laptop orchestras, the perfor- mance of live electronic music can often be an alien- ating experience for the average listener, performing the- tation of the sound are happening in plain sight: there is no black box in the case of a symphony performance. This understanding on the part of the audience is often absent in the case of digital music. For one thing, the average musical listener is often not conversant with the various software programmes commonly used in live per- formance. However, even for an audience that is so con- versant, there is often no way of knowing firstly what soft- ware is being used, and secondly what parameters are being altered at any given time, and how the actions of the musicians are reflected in her sonic out- put. There are simply too many possibilities. Because of the complex process of interaction between the digital link and the audience and the performer – the establishment of fellowfeeling through the music – may be compromised.

2.4. What about electronic music improvisation? Is a truly social creativity possible?

Moving away from purely presentational music, we might ask what the state of affairs is when it comes to participa- tory music-making – for instance, by considering the na- ture of the experience of the performers themselves. This becomes particularly relevant when considering the pos- sibility of improvising with digital music, as well as col- laborative composition, which we consider as occupying ‘opposite ends’ of the same process, following Nettl. In the case of digital music ensembles, much the same prob- lems occur as those outlined above with regard to presen- tational music: that is, because of the presence of ‘black boxes’ on the ensemble, it is very difficult for the musician to understand what the others are doing. In traditional ensembles, the technologies are stable: the symphony or- chestra, for instance, has existed for much the same in- struments for the last hundred years. Even though there have been some changes, the pace of the evolution of the symphony orchestra has been extremely slow. For that reason, it is possible for each player to have a good under- standing of the ranges and controls involved in each instrument. With digital ensembles, however, any combination of software and hardware could be in use at any given time. Even if each player happens to be familiar with a combination of software and hardware in use by each of the other players, there is much less of a chance that she will have an in-depth, fine-grained understanding of the parameters of each of those combinations.

2.5. Lack of constraints in electronic improvisation: a barrier to creativity

The nature of electronic music, in its open-endedness, also hampers communication among players. Jazz players can improvise fluently because they understand the pre-given harmonic structures of the piece, know the general struc- ture of an improvisation in that particular genre and have an in-depth embodied knowledge of their instrument. In other words, they know the rules, in much the same way as the North Indian sitarists mentioned in section 1. Johnson [10] and Laird [8] argue that this knowledge reduces the load on performers and listeners alike.

2.6. How might we make digital music more comprehensible and more social?

Leman [9] argues for a greater emphasis on embodied control of digital technology, in order to bridge this gap in understanding. Much of the current research in NIMEs...
constitutes an important step towards creating new, em-
doubled digital interfaces, which will go towards solving
at least part of the puzzle faced by electronic instrumen-
talists. However, the problem of fine-grained understand-
ing of the parameters of the music itself remains, even if
we achieve stability of digital interfaces, as well as the
problem of the excessive freedom of electronic improvi-
sation. We propose that the capacity of digital ensembles
for the storage of history suggests a potential solution to
this problem, paving the way for a more truly social digi-
tal music.

3. SNAPSHOTS: QUERYABLE HISTORY

3.1. Background: understanding sounds

Insofar as musical listening involves sounds, it is reason-
able to assume that musical sounds ought to be ‘under-
stood’ in some way by the listener. An understanding of
the causal chain implicated in sound production is gener-
ally present in audience experiences of acoustic musical
performance, due to familiarity with the instruments used
and also an embodied understanding of how the sound is
produced. However, as discussed above, the case of dig-
tal music listening, in particular the case of improvised
digital music, poses problems for the listener, who typi-
cally does not have the same level of understanding about
how the sound is being produced. Moreover, due to the
lack of genre conventions in digital improvisation, the lis-
tener may not grasp how sonic structures in the improvi-
sation relate to each other. This is also true to some extent
for the performers themselves, who may have little or no
knowledge about what the other performers are doing or
even whose actions are resulting in which sound, which
makes intelligible improvisation difficult. That is, there
is a lack of understanding of how the actions of the play-
ers are manifested in features of the sound, and also in
how the sonic contributions of each player relate to those
of the improvisation in general. The result of this lack
of communication through sound is that communicative,
social interaction is rendered difficult for performers and
audiences alike. Given that, as we have proposed above,
music may be understood as involving social interaction
at a fundamental level, this poses deep problems for the
project of communicative digital music. In rendering dig-
tal music more transparent, Snapshots addresses these is-

3.2. Implementation

Snapshots is implemented entirely within Clojure[7]—an
expressive JVM-hosted Lisp with an emphasis on concur-
rency and functional programming. There were two main
motivations for this choice. Firstly, Clojure’s persistent
data structures[12] and support for state-of-the-art lock-
free concurrent data structures makes it well-suited to build-
ing multi-threaded systems capable of supporting multiple
simultaneous users. This functionality is vital in the con-
text of collaborative, social music-making, where multi-

ple players will necessarily be using the system at once.
In the Snapshots implementation, individual user requests
map on to independent threads of activity, which need to
currently modify and read a given stream of events in
isolation from other streams; this is typically a com-
plex and error-prone problem. Secondly, Snapshots has
been developed as part of the suite of libraries supporting
Overtone[2]. However, despite being part of the Overtone
suite of libraries, Snapshots has been designed to be used
entirely independently of Overtone itself, only requiring a
JVM implementation, which is freely available on all ma-
jor platforms. Snapshots is freely available from Github
[1] under an MIT X11 license and the architecture for this
system is presented in Figure 2.

The Snapshots server exposes itself as a generic OSC
server sitting on a specific host and listening on a specified
port. For example, the following code creates a server on
the local machine listening on port 9851:

(def c (osc-client "localhost" 9851))

3.3. Event Storage

In order to store an event, it must be encodable as a stan-

dard Open Sound Control (OSC) message, which is es-

sentially a list of values of a small set of defined types: in-
tegers, floats, strings and binary blobs (specifics for these
types can be found in the OSC specification[15]). A typi-

cal OSC message contains a path followed by 0 or more
arguments. For example, consider the following encoding
of a MIDI-like piano event, which represents note 60 be-
ing triggered at volume 10 (within the standard range of
0-127):

/value/on 60 10

In order to store this message in Snapshots, it is necessary
to firstly create an OSC client with the server’s details:

(def c (osc-client "localhost" 9851)))

Once the client has been created, the piano event can be
sent as arguments to the /store command:

(osc-snd c "/store"
"/my-stream"
"/note/on")

Notice that the /store command also requires an extra
argument (i.e. /my-stream) representing the name of the
stream in which the event is to be stored. This allows the
server to store any number of independent streams.3
For each event stored, Snapshots stores an immutable
snapshot of the event with the persistent nature of Clo-
jure’s datastructures, this snapshot is guaranteed to repre-
sent a valid stream state (i.e. not in a liminal state such
as the partial completion of a storage update). Also, the
algorithmic complexity of creating this snapshot is essen-
tially free, O(1), and does not interfere with the ability
for other threads to be concurrently updating the original
stream. Once the snapshot has been created, the thread
then starts sending each element in the snapshot to the
specificed location as separate OSC messages. Clearly,
the cost of this streaming is O(n) in the size of the stream.
However, as it is operating on an immutable snapshot
and not the original stream, new events stored in the original
stream between the snapshot creation and the completion
of OSC data transmission will not be reflected in the snap-
shot and therefore not sent as an OSC event as the reply
to a /fetch message. Finally, as the snapshot creation
and subsequent transmission happen in an isolated thread
that holds no locks, it does not block the server’s ability
to continue to concurrently handle additional requests.

3.5. SnaQL - Snapshot Query Language

It is not always desirable to retrieve the entire contents
of a specific stream. It is therefore important to be able to
describe interesting subsets of a given stream of events and
formulate descriptions through its query language SnaQL (Snapshot
Query Language). This subset could be as simple as rep-
resenting a point in time in the stream where a specific
filter expression was true or false and consists of predicate
types such as compare and set (CAS) which ensures the atomic succes-
sion of stored values.

Once events have been stored, we may then retrieve
them, filter them and even generate new named frenum
event streams to be further referenced, filtered and re-
trieved. Retrieved event streams are then available as ba-
sic native data structures to be manipulated, played back
or stored to disk from any programming language with an
OSC library.

3.4. Event Retrieval

Snapshots supports the retrieval of all the messages stored
in any of the named stores as a series of individual OSC
messages. Retrieval is achieved using the /fetch com-
mand, which expects four arguments: a unique identifier
for this particular query, the name of the stream to fetch
and the hostname and port of an OSC server, to which
the stored events are to be sent. For example:

(osc-snd c "/fetch" 78
"/my-stream"
"localhost" 9851)

On receiving a /fetch message, an independent thread
on the server first creates an immutable snapshot of the
stream’s current state. Due to the combination of the stream’s
use of CAS for updates and the persistent nature of Clo-
jure’s datastructures, this snapshot is guaranteed to repre-
sent a valid stream state (i.e. not in a liminal state such
as the partial completion of a storage update). Also, the
algorithmic complexity of creating this snapshot is essen-
tially free, O(1), and does not interfere with the ability
for other threads to be concurrently updating the original
stream. Once the snapshot has been created, the thread
then starts sending each element in the snapshot to the
specified location as separate OSC messages. Clearly,
the cost of this streaming is O(n) in the size of the stream.
However, as it is operating on an immutable snapshot
and not the original stream, new events stored in the original
stream between the snapshot creation and the completion
of OSC data transmission will not be reflected in the snap-
shot and therefore not sent as an OSC event as the reply
to a /fetch message. Finally, as the snapshot creation
and subsequent transmission happen in an isolated thread
that holds no locks, it does not block the server’s ability
to continue to concurrently handle additional requests.
constitutes an important step towards creating new, em-
dabled digital interfaces, which will go towards solving at
least part of the puzzle faced by electronic instrument-
talists. However, the problem of fine-grained understand-
ing of the parameters of the music itself remains, even if
we achieve stability of digital interfaces, as well as the
problem of the excessive freedom of electronic improvis-
ations. We propose that the capacity of digital ensembles
for the storage of history suggests a potential solution to
this problem, paving the way for a more truly social digi-
tal music.

3. SNAPSHOTS: QUERYABLE HISTORY

3.1. Background: understanding sounds

Insofar as musical listening involves sounds, it is reason-
able to assume that musical sounds ought to be ‘under-
stood’ in some way by the listener. An understanding of
the causal chain implicated in sound production is gener-
ally present in audience experiences of acoustic musical
performance, due to familiarity with the instruments used
and also an embodied understanding of how the sound is
produced. However, as discussed above, the case of dig-
tal music listening, in particular the case of improvised
digital music, poses problems for the listener, who typi-
cally does not have the same level of understanding about
how the sound is being produced. Moreover, due to the
lack of genre conventions in digital improvisation, the lis-
tener may not grasp how sonic structures in the improvis-
ations relate to each other. This is also true to some extent
for the performers themselves, who may have little or no
knowledge about what the other performers are doing or
even whose actions are resulting in which sound, which
makes intelligible improvisation difficult. That is, there
is a lack of understanding of how the actions of the play-
ers are manifested in features of the sound, and also in
how the sonic contributions of each player relate to those
of the improvisation in general. The result of this lack of
communication through sound is that communicative,
social interaction is rendered difficult for performers and
audiences alike. Given that, as we have proposed above,
music may be understood as involving social interaction
at a fundamental level, this poses deep problems for the
project of communicative digital music. In rendering dig-
tal music more transparent, Snapshots addresses these is-
suely directly.

3.2. Implementation

Snapshots is implemented entirely within Clojure[7], an
expressive JVM-hosted Lisp with an emphasis on concur-
rency and functional programming. There were two main
motivations for this choice. Firstly, Clojure’s persistent
data structures [12] and support for state-of-the-art lock-
free concurrency semantics makes it well-suited to build-
ing multi-threaded systems capable of supporting multiple
simultaneous users. This functionality is vital in the con-
text of collaborative, social music-making, where multi-
ple players will necessarily be using the system at once.
In the Snapshots implementation, individual user requests
map on to independent threads of activity, which need to
corrently modify and read a given stream of events in
isolation from other streams; this is typically a com-
plex and error-prone problem. Secondly, Snapshots has
been developed as part of the suite of libraries supporting
Overtone [2]. However, despite being part of the Overtone
suite of libraries, Snapshots has been designed to be used
entirely independently of Overtone itself, only requiring a
JVM implementation, which is freely available on all ma-
jor platforms. Snapshots is freely available from Github
[1] under an MIT X11 license and the architecture for this
system is presented in Figure 2. The Snapshots server exposes itself as a generic OSC
server sitting on a specific host and listening on a specified
port. For example, the following code creates a server on
the local machine listening on port 9851:

(def c (osc-client "localhost" 9851))

3.3. Event Storage

In order to store an event, it must be encodable as a stan-
dard OSC message, which is essentially a list of values of
a small set of defined types: integers, floats, strings and binary blobs (specifics for these
types can be found in the OSC specification [15]). A typ-
ical OSC message contains a path followed by 0 or more
arguments. For example, the following encoding of a MIDI-like piano event, which represents note 60 be-
ing triggered at volume 10 (within the standard range of
0-127):

/notes/on 60 10

In order to store this message in Snapshots, it is necessary
to firstly create an OSC client with the server’s details:

(def c (osc-client "localhost" 9851))

Once the client has been created, the piano event can be
sent as arguments to the server command:

(osc-snd c "/store" "/my-stream" "/note/on" 60 10)

Notice that the /store command also requires an extra
argument (i.e. /my-stream) representing the name of
the stream in which the event is to be stored. This allows
the server to store any number of independent streams.1
For each event stored, Snapshots associates with it an
OSC path (i.e. /note/on) and a list of arguments (i.e. (60 100)). In addition, a timestamp is stored, which records the time
Snapshots received the event as well as the sender’s host-
name and port. Each of these events is represented with an
immutable associative map such as the following:

{| (path "/piano/on")
 | (ARGS [60 100])
 | (ts 132735574447)
 | (sr-c-port 64546)
 | (sr-c-host "localhost")
|

The store commands may be sent from an arbitrary num-
ber of external clients and may request the storage of his-
ory in an arbitrary stream of named stores. Snapshots
 guarantees that these storage commands will never con-
flict with each other. This guarantee is ultimately met
at the CPU level with the instruction Compare and Set
(CAS) which ensures the atomic succession of stored val-
ues.

Once events have been stored, we may then retrieve
them, filter them and even generate new named frozen
event streams to be further referenced, filtered and re-
trieved. Retrieved event streams are then available as ba-
sic native data structures to be manipulated, played back
or stored to disk from any programming language with an
OSC library.

3.4. Event Retrieval

Snapshots supports the retrieval of all the messages stored
in any of the named stores as a series of individual OSC
messages. Retrieval is achieved using the /fetch com-
mand, which expects four arguments: a unique identifier
for this particular query, the name of the stream to fetch
and the hostname and port of an OSC server, to which the
stored events are to be sent. For example:

(osc-snd c "/fetch" 78 "/my-stream" "/note/on" 9873)

On receiving a /fetch message, an independent thread on
the server first creates an immutable snapshot of the
stream’s current state. Due to the combination of the stream’s
use of CAS for updates and the persistent nature of Clo-
jure’s datastructures, this snapshot is guaranteed to repre-
sent a valid stream state (i.e. not in a liminal state such
as the partial completion of a storage update). Also, the
algorithmic complexity of creating this snapshot is essen-
tially free, O(1), and does not interfere with the ability
for other threads to be concurrently updating the original
stream. Once the snapshot has been created, the thread
then starts sending each element in the snapshot to the
specified location as separate OSC messages. Clearly, the
cost of this streaming is O(n) in the size of the stream.
However, as it is operating on an immutable snapshot and
not the original stream, new events stored in the original
stream between the snapshot creation and the completion of
OSC data transmission will not be reflected in the snap-
shot and therefore not sent as an OSC event as the reply
to a /fetch message. Finally, as the snapshot creation
and subsequent transmission happen in an isolated thread
that holds no locks, it does not block the server’s ability
to continue to concurrently handle additional requests.

3.5. SnaQL - Snapshot Query Language

It is not always desirable to retrieve the entire contents of
a specific stream. It is therefore important to be able to de-
scribe interesting subsets of a given stream of events and for
mulate queries that describe these subsets. This can be done
through the /query message and supports subset descriptions through its query language SnaQL (Snapshot Query Language). This subset could be as simple as rep-
resenting the entire event, in which case, it would act similarly to /fetch or it might contain more
specifics. The SnaQL query language implements this filtering process using a cascading filter-chain gener-
ated from the SnaQL statements. Note that the cardinality of the original snapshot n is less than or equal to the car-
dinality of the result snapshot m, i.e. m <= n. SnaQL statements consist of one or more filter expres-
sions, see Table 1. Each filter expression is constructed by using one of the supplied filter types such as drop, take and remove and passing the appropriate param-
eter. A parameter is either a number or a predicate. A
predicate can be seen as a function which returns either
ture or false and consists of predicate types such as and,
or and not=, see Table 2. Each predicate takes one or
more clauses which may be other predicates or values.
A value is an immutable number or string. Values
can either be explicitly entered into the predicate, or may be dynamically retrieved from the current event not being
filtered. The syntax for referencing values within an event
is (event /event-value arg=). A value is one of the entries in Table 3.

As an example, consider the following interest state-
ment: “The first 10 /note/on events with a velocity be-
tween 40 and 80”. In SnaQL this may be expressed as:

(drop (not= (event /path) /note/on) filter (> (event /arg 1) 40)
drop n  Returns the snapshot except for the n events.
take n  Returns first n events from the snapshot.
drop-while p  Returns the snapshot except for the n consecutive events for which predicate p returns true.
take-while p  Returns the first n consecutive events for which predicate p returns true.
filter p  Returns only the events for which predicate p returns true.
remove p  Returns only the events for which predicate p returns false.

<table>
<thead>
<tr>
<th>Table 1. Filter types.</th>
</tr>
</thead>
<tbody>
<tr>
<td>and &amp; cs  Returns true if all of the clauses cs return true.</td>
</tr>
<tr>
<td>or &amp; cs  Returns true if only one of the clauses cs return true.</td>
</tr>
<tr>
<td>not= &amp; cs  Returns true if all of the clauses cs are equal.</td>
</tr>
<tr>
<td>= &amp; cs  Returns true if all of the clauses cs are not equal.</td>
</tr>
<tr>
<td>&lt; &amp; cs  Returns true if clauses cs are in monotonically increasing order.</td>
</tr>
<tr>
<td>&gt; &amp; cs  Returns true if clauses cs are in monotonically decreasing order.</td>
</tr>
<tr>
<td>not c  Returns true if clause c is logical false, false otherwise.</td>
</tr>
</tbody>
</table>

The SnaQL engine parses these statements and generates a four-stage cascading filter-chain with which to filter the contents of a specific event stream snapshot. The first filter is passed all of the events within an immutable snapshot of a stream. Applying the first filter to this stream results in a new immutable snapshot which removes any events that do not have the path /note/on. This subset snapshot is then passed to the second filter, which only keeps the events for which the first argument is greater than 40. This snapshot is fed, in turn, as input to the third filter, which only keeps the events for which the first argument is less than 80. The final filter then returns only the first 10 events. The snapshot generated as the output of the chain of filters represents the resulting subset of interesting events.

3.7. Query Snapshots

In order to further facilitate the sharing of history between users, it is possible to create named frozen immutable snapshots of a given filter-chain output. This is made possible with the /snapshot command. This command has two required arguments and zero or more SnaQL filter expressions. The required arguments are: unique ID, name of the event stream to query; hostname to reply to; and port to reply to. These are similar to the arguments for the /fetch command. In fact, if no SnaQL filter expressions are passed to /query, it has identical semantics to /query; it snapshot the event stream and then sends each event as a separate OSC message to the specified host and port. However, if filter expressions are specified, the snapshot is filtered appropriately before being sent as individual OSC messages; that is to say, a subset of the original stream is always returned. The above SnaQL example can be expressed as a snapshot:

```
:path /snapshot
:args n  The path of the event.
:arg n  The n-th argument of the event.
:ts  The timestamp of the event.
:port p  The port the event originated from.
:host h  The host the event originated from.
```

4. CONCLUSIONS & FUTURE WORK

We have described ‘Snapshots’, a system that facilitates the querying of the history of a digital improvisation session and the subsequent manipulation of the query results. This, we argue, represents an important first step towards making digital improvisation more intelligible, creating as it does the possibility of the use of past material to influence or modulate current and future events. Being implemented within Clojure, Snapshots has built-in concurrency semantics, making it ideal for group scenarios where there are multiple inputs to the system. This, we argue, vital if digital music is to become more truly social and participatory.

We have described a basic system of querying and naming events. Future work could include building functionality for performing computations on these immutable events in order to generate interesting patterns in the improvisation. Another intriguing prospect is the generation of semantic visualizations from the operations of Snapshots, which might represent a further step in crossing the communication barrier, giving the audience visually rich clues as to the processes that are implicated in the sonic results of the improvising group. In other words, by layering the groundwork for interesting manipulation of data on the fly, we are setting the scene for performers and audiences to achieve a more fine-grained understanding of the processes involved in electronic music. We feel that this is an exciting prospect for a more collaborative, outward-looking and expressive digital music.

5. REFERENCES

drop n
Returns the snapshot except for the first n events.

take n
Returns first n events from the snapshot.

drop-while p
Returns the snapshot except for the first n consecutive events for which predicate p returns true.

take-while p
Returns the first n consecutive events for which predicate p returns true.

filter p
Returns only the events for which predicate p returns true.

remove p
Returns only the events for which predicate p returns false.

and & cs
Returns true if all of the clauses cs return true.

or & cs
Returns true if only one of the clauses cs return true.

not= & cs
Returns true if all of the clauses cs are not equal.

= & cs
Returns true if all of the clauses cs are equal.

< & cs
Returns true if clauses cs are in monotonically increasing order.

> & cs
Returns true if clauses cs are in monotonically decreasing order.

not c
Returns true if clause c is logical false, false otherwise.

Table 2. Predicate values. The & denotes variable arity.

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>filter (&lt; (event :arg 1) 80)</code></td>
<td>Takes the first 10 events in the stream. The snapshot generated as the output of the chain of filters represents the resulting subset of interesting events.</td>
</tr>
<tr>
<td><code>take 10</code></td>
<td>Takes the first 10 events in the stream. The snapshot generated as the output of the chain of filters represents the resulting subset of interesting events.</td>
</tr>
</tbody>
</table>

3.7. Query Snapshots

In order to further facilitate the sharing of history between users, it is possible to create named frozen immutable snapshots of a given filter-chain output. This is made possible with the `snapshot` command. This command has two required arguments and zero or more SnaQL filter expressions. The required arguments are: unique ID, name of the event stream to query; hostname to reply to; and port to reply to. These are similar to the arguments for the `show` command. This command allows the user to embed SnaQL statements in stream and then sends each event as a separate OSC message.

Example:

```osc
(filter (< (event :arg 1) 80))
```

4. CONCLUSIONS & FUTURE WORK

We have described 'Snapshots', a system that facilitates the querying of the history of a digital improvisation session and the subsequent manipulation of the query results. This, we argue, represents an important first step towards making digital improvisation more intelligible, creating as it does the possibility of the use of past material to influence or modulate current and future events. Being implemented within Clojure, Snapshots has built-in concurrency semantics, making it ideal for group scenarios where there are multiple inputs to the system. This, we argue, vital if digital music is to become more truly social and participatory.

We have described a basic system of querying and naming events. Future work could include building functionality for performing computations on these immutable events in order to generate interesting patterns in the improvisation. Another intriguing prospect is the generation of semantic visualizations from the operations of Snapshots, which might represent a further step in crossing the communication barrier, giving the audience visually rich cues to as the processes that are implicated in the sonic results of the improvising group. In other words, by layering the groundwork for interesting manipulation of data on the fly, we are setting the scene for performers and audiences to achieve a more fine-grained understanding of the processes involved in electronic music. We feel that this is an exciting prospect for a more collaborative, outward-looking and expressive digital music.