Interaction Design for Mobile Music Composition

Master Thesis

by

Sebastian Fichtner

Konstanz University
Faculty of Computer- and Information Science
Departmant of Human-Computer Interaction

Prof. Dr. Harald Reiterer

Konstanz, 2015

Contents

Abstract

In this master thesis, we investigate a possibly innovative interaction concept for a music composition tool on tablets. We present a prototype that communicates a novel domain model that we had prepared and introduced through previous works [17, 18, 19]. Further, we report from a study of the prototype that we conducted with users of music sequencer software and interpret the produced qualitative data in terms of our design. We are able to derive valuable insights into the user group, its working practices, our particular design and CST design in general. We are also able to refine the requirements on which our design is based and initiate a re-design based on the study results.

Chapter 1

Introduction

Our drive to create meaning distinguishes us from animals and machines. It is what makes us human. To help people create meaning is to help them express their true nature as human beings. And that may be the most noble purpose to which software can ever be applied. That is why we dedicated our master subject to the development of software tools for creativity support.

Creativity-support tools (CSTs) depend, like hardly any other type of software, on the art and science of Human-Computer Interaction (HCI). The interaction design of CSTs is a distinct area within HCI research and the primary scientific context for this thesis.

As our concrete application domain, we chose music composition. It offers a perspective on CST design that the HCI community seldom picks up. The frame of reference for research and innovation in this field are digital audio workstations (DAWs) like Apple Logic Pro, Ableton Live and Avid ProTools. The core metaphor of this type of software is the classic recording studio in which producers and sound engineers monitor, record and mix some musician's (semi-)live performance.

DAWs originally served producers and the recording- and mixing process. They did not target musicians or the composition process. In that sense, they apply to the technical- rather than the creative side of music making.

Musicians and composers have adapted to this situation and accepted

DAWs as the standard. And DAWs themselves continue in their attempt to cover all aspects of music making and to please everyone. However, they never overcame the mismatch between their core metaphor and the creative process. To address exactly that mismatch is our motivation for this work.

The preceding seminar [19] and project [18] set the context for this thesis at hand. The seminar in particular is quite extensive and allows us to introduce our subject in rather broad strokes. In the following, we'll summarize both previous works.

1.1 Seminar

In the master seminar [19], we identified requirements for audio- and music composition software. We distilled those requirements from a large body of scientific literature from HCI, computer music, musicology and psychology. In the past decades, a particularly interesting field of research grew at the intersection of HCI and computer music [8, 16, 20, 22].

We are confident that these literature-based requirements express a true need and potential market gap. Not only do they heavily overlap with our personal experience from years of music software use, but most of the related works are also based on user studies.

We identified four broad themes that pervade research results on interaction design for music composition. They serve as categories for our 36 requirements:

1. Simplicity
2. Freedom
3. Exploration
4. Abstraction

These categories roughly reflect the needs that CSTs in general need to support. The abstraction category, however, emerged as a particularly important aspect of music composition tools. We'll get back to the exact abstraction requirements in a moment.

The works of Duignan et al. [11, 12, 13] became the center of our interest because they specifically asked how DAWs fail to support the composition process, and they investigated this question in a profound way.

Duignan et al. [12] started from a bird's eye view that allowed them to develop a taxonomy of music composition (or *sequencing-*) software. Based on their taxonomy, they identified a type of sequencer that would be ideal for creative users but is practically non-existent for it is a unique combination of characteristics:

1. *Graphical* Mode (rather than *textual* mode)
2. *Custom* Abstractions (rather than *predetermined* abstractions)
3. *Delayed* Linearization (rather than *eager* linearization)
4. *Control* Flow (rather than *data* flow)
5. *General* Purpose (rather than *special* purpose)

We might roughly describe the "GCDCG-Sequencer" as a classic DAW enhanced by more live performance capabilities (delayed linearization) and custom abstractions. Related works support the idea that this sequencer type would better serve creativity in music making, for instance research on the preconditions of flow in composition software [29, 30, 31] or the prevalent cognitive styles of composers [5, 14].

As we laid out in the seminar, custom abstractions are a precondition of delayed linearization. When the user wants to define live (on the fly) which parts of a piece of music are being played next, some abstract notion of a "part" must exist in the composition tool.

Custom abstractions are much more fundamental than delayed linearization, and when they are made more available, capabilities of delayed linearization automatically increase as well. So, an essential step towards innovation of composition interfaces would be to add custom abstractions to the kind of graphical control flow sequencers that is the centre of traditional DAWs.

Duignan and his colleagues realized this need and initially stated they were developing such a GCDCG-Sequencer [12], but they never published the result. Instead, Matthew Duignan [11] dedicated his PhD thesis to a deeper assessment of the exact shortcomings of DAWs for music composition. Based on a user study with 17 participants, he revealed how a lack of custom abstractions in DAWs profoundly disrupts the creative flow of the composition process. In a summarizing paper, Duignan et al. [13] stated:

> "This framework helps us understand and clearly identify issues that need to be resolved in the next generation of DAW user interfaces."

And this is were the story ended. Although this framework is a milestone, no one has, yet, taken the next step and put it into practice in order to design this "next generation of DAW user interfaces". One reason may be that there are many diverse issues to be resolved, and some of the implied requirements are conflicting or, at least, not trivial to align with one another. Through our master project and -thesis, we pick up where Duignan et al. left off and hope to add another chapter to the story.

1.2 Project

The master project was actually a long process going through countless iterations of contemplation, sketching and implementation. We provided a simplified and streamlined description of that process in the project paper [18]. Here, we briefly summarize its challenges and results.

Early on in the seminar, we learned that if we take this seriously and strive for real innovation, we have to acknowledge that *interaction design* involves more than cosmetics and goes deeper than *interface design*. We have to rebuild the interface of sequencing software from its core, starting with the conceptual model of composed music that it ultimately communicates to the user. Therefore, the most important result of the project is neither code nor mockups but a domain model that is in line with the requirements.

So, how did we approach those requirements? In the master project, we were able to narrow them down to four core requirements. We also implicitly attended to Requirement #19, which expresses our rather general premise. A sequencer that satisfies those five can easily be extended to satisfy most of the other 31 requirements later on:

#19 *GCDCG-Sequencer*: The tool's sequencer type should be: graphical mode, custom abstractions, delayed linearization, control flow and general purpose [12].

#21 *Voice Groups*: The tool should support multi-level voice groups. Groups should act as voices, including their representation, editing, reuse and so forth.

#25 *Temporal Abstraction*: The tool should provide multi-level temporal abstraction for sequencing and composing musical objects at all temporal scopes.

#32 *Preparation*: The tool should provide a place to store currently unused material and make this material the context of future work.

#33 *Global Reuse*: The tool should enable access to- and seamless reuse and combination of all previously created material through one integrated library.

Basically, we want to allow the user to recursively build voice- and temporal abstractions and to freely store, reuse and combine those musical objects from within the composition tool. For graphical sequencers, this would add a radically new concept and empower the user in unimaginable ways. However, the groundwork for that already turned out to be far from trivial.

Figure (1.1) depicts our domain model as a simple class dependence diagram. Because we keep model and actual implementation congruent, the diagram reflects both.

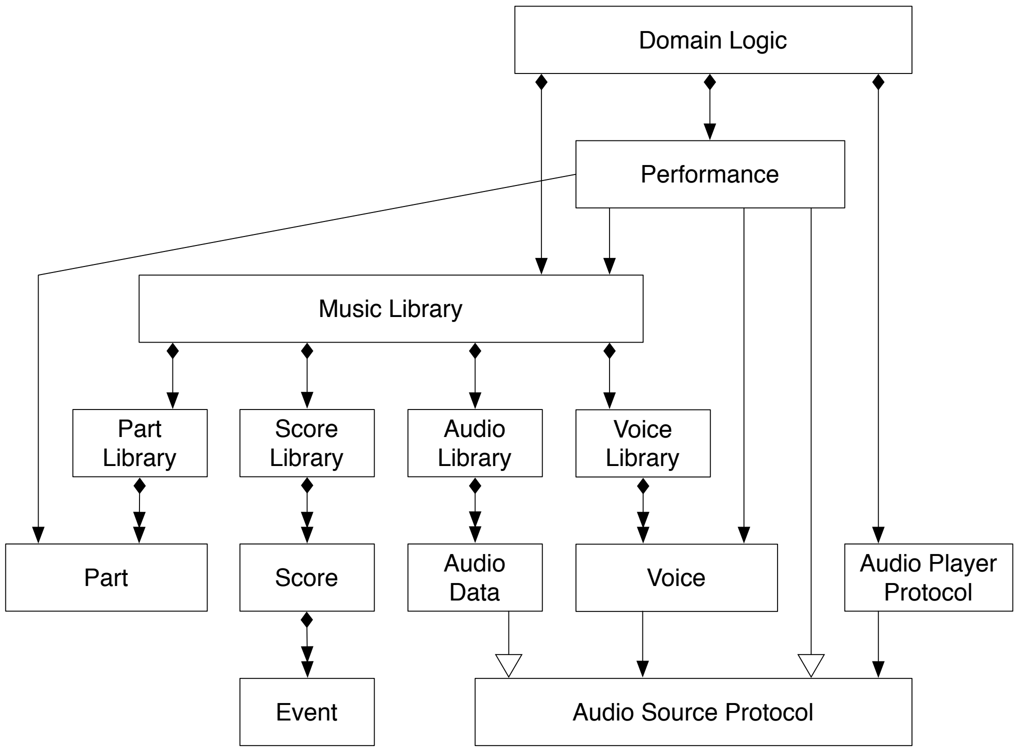


Figure 1.1: Overview of the domain model as a dependence diagram

A major break through of domain modeling was the realization that we have to strictly decouple voice- and temporal abstractions, and that forced us to abandon the central role and conventional notion of a *composition*. In our model, there is no distinct composition entity. Instead, we have a *Performance* that is of a much more fleeting nature. It is an ad-hoc combination (current user selection) of a temporal abstraction (*part*) and a voice abstraction (*voice*). We'll illustrate the musical meaning of those entities when we introduce our design approach.

As another result of seminar and project, we decided on tablets as our target platform. Compared to conventional systems, the mobility, touch interaction and simplicity of tablets promises to be a far better match for today's users, creativity-support tools, music creators and many of our particular requirements. At the same time, tablets offer more screen space than smart phones, and this larger canvas is crucial for creative content production.

The direction in which the requirements point us and the mobility and pragmatic minimalism that tablets promote amplify each other. And that synergy may bring about the next generation of music sequencers and CSTs.

A more general and simple domain model not only supports a more creative process but also allows us to mediate that process through the relatively small screen of a tablet. And the tablet itself is a wonderful constraint that forces us to design for the focus and mono-tasking that creative work requires.

A convention of sequencer software that we adapt in only slightly generalized form is the 2-dimensional "editing plane" that distinguishes voices (sounds) on one axis and time on the other. This is were the user actually arranges musical events. We had to bring this basic building block to the tablet before we could add the higher level navigation and editing of custom abstractions.

As it turned out, "translating" the simple editing of events for different sounds in time from the desktop to the tablet required much more sensitivity, experimentation and implementation effort than we imagined. The project ultimately focused on that very step.

1.3 About this Work

For this thesis, we developed our design concept further and complemented our implementation with in-screen mockups. In the next chapter, we'll demonstrate the concept as a whole, using screen shots in conjunction with mockups.

In Chapter 3, we will review our overall methodology, explain the evaluation method that we applied and present the results of our evaluation process.

In Chapter 4, we will discuss, interpret and (as far as possible) generalize the results of our evaluation. We will also present how we advanced our concept based on what we learned from those results.

And finally, Chapter 5 will discuss the limitations of our contribution and the directions future research and development should take.

Chapter 2

Design Concept

Our frame of reference for explaining our *Human-Audio Interaction Lab* (HAIL) is Apple's GarageBand. GarageBand is the most commonly used music software on tablets and perfectly represents DAWs on that smaller platform.

We occasionally compare HAIL to the sequencer component of GarageBand in order to explain what we do differently than conventional sequencers and why those differences might be beneficial. In the project [18], we already used GarageBand to illustrate the starting point of our design process. For a more detailed introduction to GarageBand on the iPad, see Apple's online guide at *http://help.apple.com/garageband/ipad*.

2.1 Basics

We laid out the development of our domain model in the project paper, but we want to give an even more concise and understandable explanation of the results because they are the foundation of all that follows and because we advanced model and design for this thesis.

2.1.1 Parts

Parts represent time intervals that may have a specific meaning to the composer. A part may be a symphony, a song, a new idea, a verse, a chorus, a bridge, just a single bar or any other temporal abstraction. Such categories or labels are not innate to the domain model but are arbitrary meanings assigned by the composer. The composer might express how he thinks of a certain part by naming it.

Figure (2.1) illustrates how the composer may label different parts and how he might relate them to each other.

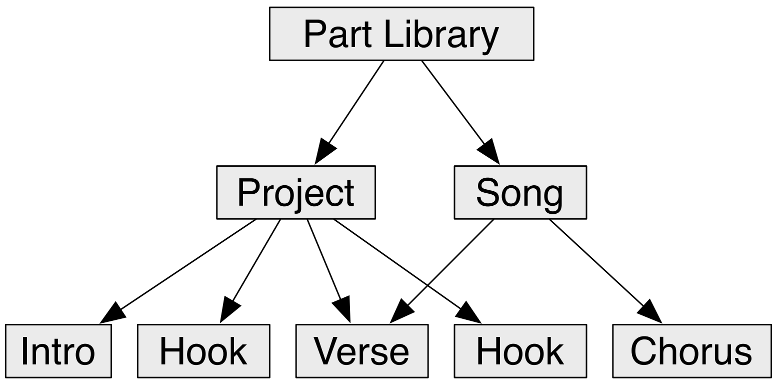


Figure 2.1: Example composition of parts

A part may just be a list of other parts. For example, there may be a part named *Project* which is made of the list <*Intro, Hook, Verse, Hook*>. We sometimes call this type of part a *Composed Part* or a *Part Group.*

Composed parts can list (reference) other parts multiple times, just like *Project* references *Hook* at two positions in its list. It is also possible that different composed parts "share" one or more parts, just like *Project* and *Song* both make use of *Verse.*

Composed parts may reference other composed parts, and there is no principle limit to the depth of recursion. However, it is impossible for a composed part to reference itself, neither directly nor through other references.

What we get is a directed acyclic graph (DAG) of parts, so we'll often speak of *Nodes*, *Children* or *Ancestors* to describe our domain model. We'll also use the terms *super*- and *sub-nodes* or *super*- and *sub-parts* as synonyms for parents and children.

The only part that necessarily pre-exists is what we call the *Part Library*. The part library is the default parent of all parts that would, otherwise, be roots themselves. Therefore, there is always exactly one global root or "source" in the part graph.

2.1.2 Voices

Voices represent sounds that can be played. A voice may be a piano key, a vocal recording, a drum hit, some arbitrary noise, some sampler instrument, a band, a whole orchestra or any other abstraction of this kind. Like with parts, such labels don't come from the domain model itself but are attributions made by the composer, which is the whole point of *custom* abstractions.

How the composer can relate voices to each other is totally analog to parts. Figure (2.2) shows an example graph of voices.

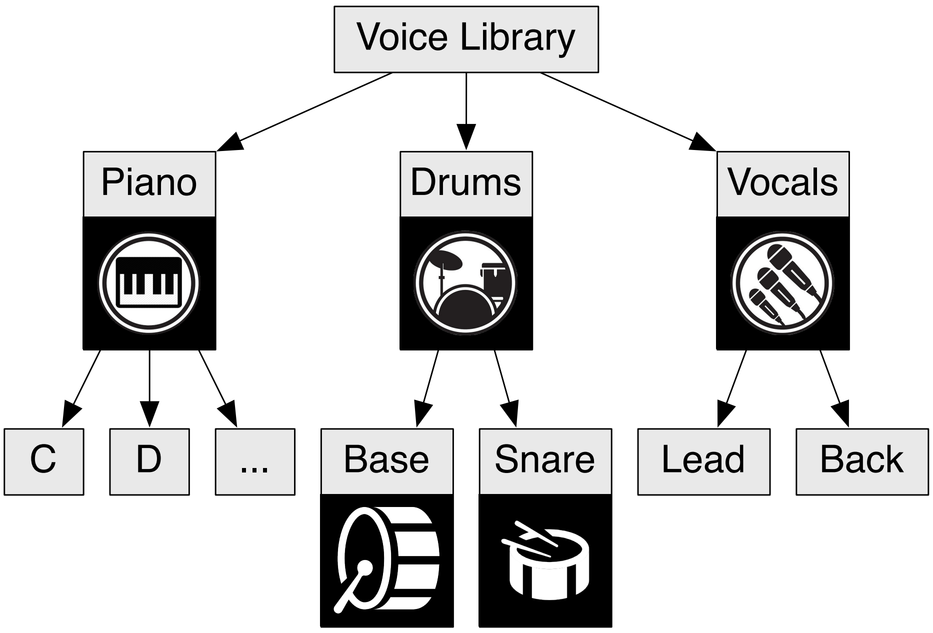


Figure 2.2: Example composition of voices

In the example, the user has already attached images to five voices. This should also help the reader to understand the figures in this work and relate mockups to the voice graph.

2.1.3 Performances

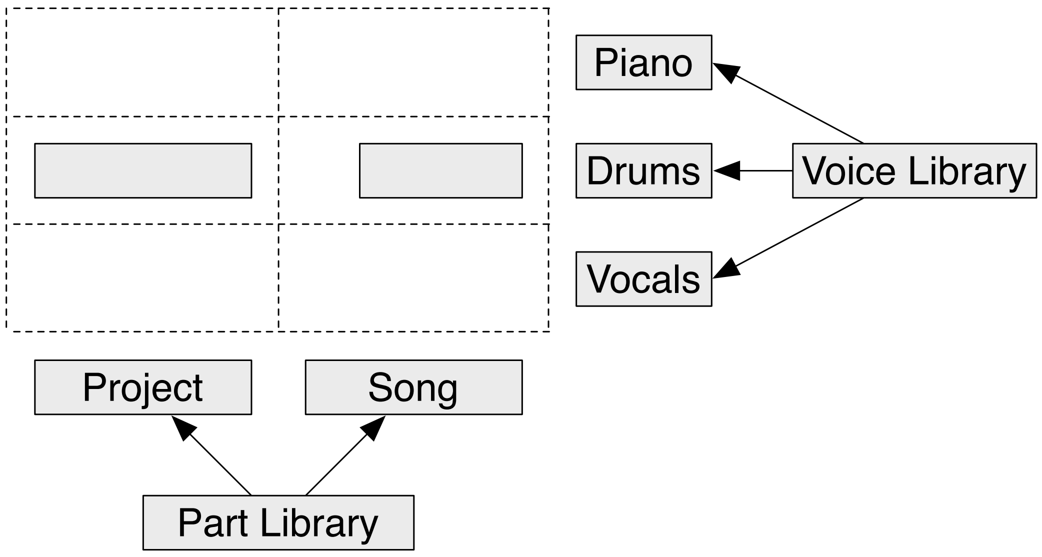
A *Performance* is a combination of one part with one voice. Any combination of a node from Figure (2.1) with a node from Figure (2.2) would make a performance. Figure (2.3) shows the performance <*Part Library, Voice Library*> and how it looks in the prototype.

At any time, the main view of the interface displays exactly one performance. Sometimes, we call it the "current" performance since the user ultimately determines which performance is being displayed.

A performance associates exactly one voice with one part, but the content that interests the user relates to the corresponding sub-nodes, i.e. the direct children of that voice and part. In Figure (2.3), we see two sub-parts and three sub-voices.

These sub-parts and sub-voices identify a matrix (dashed lines in the diagram) that we call the *Score Matrix*. The interface presents the current performance and its score matrix as a table in which columns represent sub-parts and rows represent sub-voices, similar to a spread sheet.

This 2-dimensional representation in which voices or keys are mapped to the vertical- and time is mapped to the horizontal axis is not just a design convention that pervades all kinds of music notation systems and interfaces, it also roots in image schemas and the conceptual metaphors that people universally apply to make sense of music [3, 24, 25, 39]. Of course, we find that in GarageBand as well [18]. Since this convention is also the defining principle of traditional score notation, we are safe to assume that any composer is familiar with it.



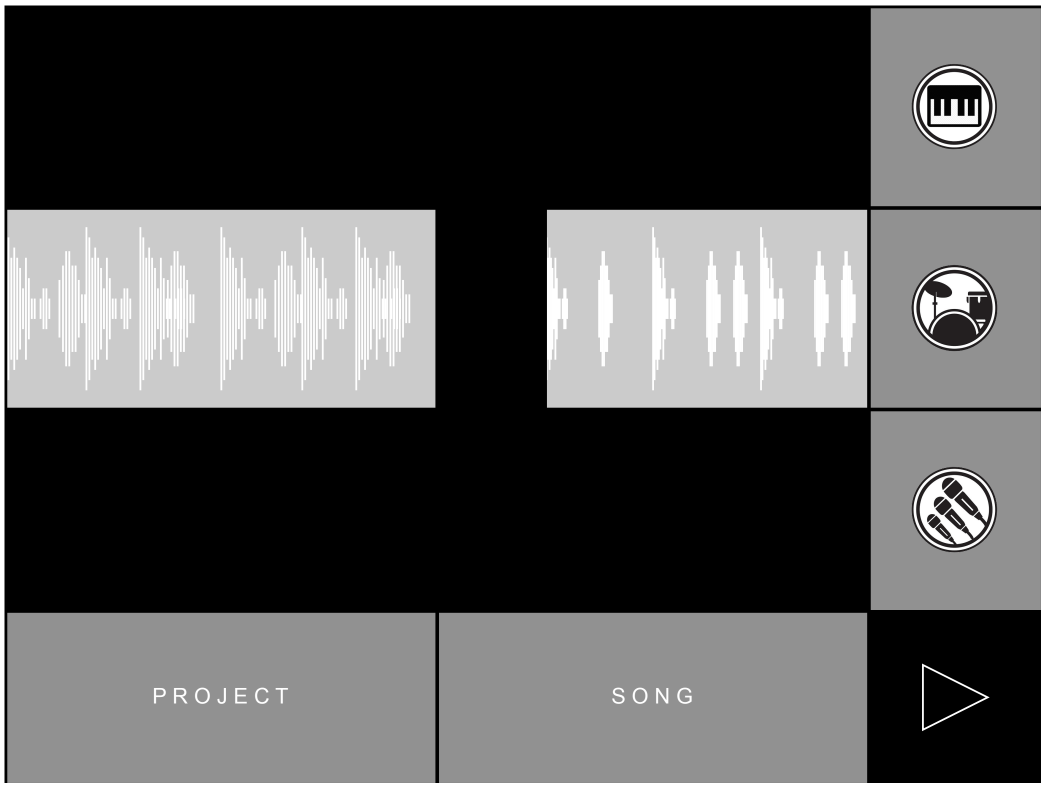


Figure 2.3: The performance <*Part Library, Voice Library*> as a diagram and in the prototype

So, our *Score Matrix* itself is nothing new. The addition that we offer is that the user can choose the level of detail or abstraction that the matrix represents in either of both dimensions. Thereby, we not only make temporal abstractions available to the user as explicit objects but also let him integrate all previously created material into his process. In Figure (2.3), for example, we see an overview of all material in the library and, through the score matrix, the interfaces provides clues about where the actual content sits. We'll elaborate on that content in the following section.

2.1.4 Events

The basic principle of how musical content is modeled, visualized and edited is trivial and an innate part of the design convention that we explained in the previous section. In common DAWs, the most prominent manifestations of that convention are the piano roll editor and the arranger. In the former, content is made of "notes". In the latter, content is made of "regions". In the project paper [18], we described what those terms mean and how we merged both contexts into one. As a result, our model doesn't distinguish notes and regions but only knows *Events*.

Like a note or a region, an *Event* associates a certain time interval with a certain voice. It means that, during the interval, the voice actively plays.

As a result of making parts explicit and treating them as an analog to voices, events cannot stretch across multiple sub-parts in the current performance. In terms of the score matrix, each event locates within the confines of one cell. Of course, there can be a whole list of events in a single cell.

However, Requirements #21 and #25 demand that we make this notion of events work on *all* abstraction levels, and that is an essential challenge to which we provided no real solution in the project paper.

The question we had to answer is: What is the meaning of events as they relate to composed (grouped) voices and parts? The answer had to be applicable to all levels. From an event that plays one piano key to an event that plays a whole orchestra. From an event that relates to a bar of 4 beats to an event that relates to a whole song. The user should be able to arrange the large-scale structure of a symphony the same way he edits the single notes (events) of the first violin. We decided to start with the simplest answer that could possibly work:

1. An event that relates to a voice group triggers, by default, all the sub-voices in the group. The sub-voices "inherit" the "incoming" event and play in unison over its exact time interval.

2. A composed part translates incoming events into events for its sub-parts so that those sub-part events still cover the exact same time interval that the incoming event covers.

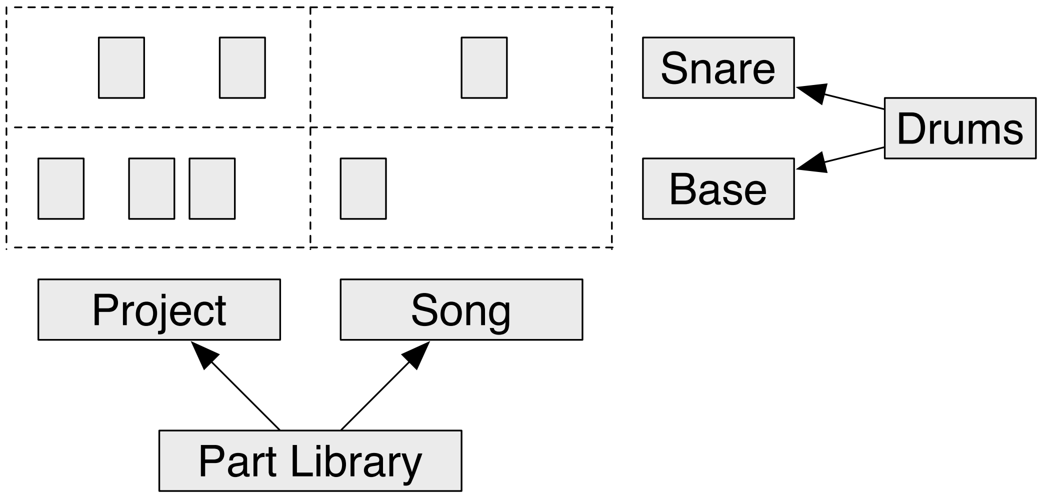
However, to really be able to employ composed voices in a musically meaningful way, there must also be the option that a composed voice may apply a more sophisticated interpretation to an incoming event than just forwarding the event to all sub-voices.

So, if the sub-voices of a voice group have there own event lists in the part in which the group is triggered, these event lists are played by the sub-voices in place of the incoming event.

While all this may sound cumbersome in words, it's actually the most natural and intuitive interpretation of "global" events in the context of our domain and in terms of using the interface.

Let's look at our example. In Figure (2.3), we see two events one which relates to the *Song* and the *Drums.* At the position within the *Song* at which the event begins, the *Drums* start playing. We know from the voice-graph in Figure (2.2), that the *Drums* are composed of *Base* and *Snare*, so when our event plays (triggers) the *Drums*, actually *Base* and *Snare* play. What exactly *Base* and *Snare* play in the *Song* is determined by their respective event lists in the *Song* and those can be seen in the score matrix of the performance <*Part-Library, Drums*> as displayed by Figure (2.4).

In Figure (2.5), on the other hand, we see how the global drum event is "split in two" and passed on to the parts *Verse* and *Chorus.* Those "inher-



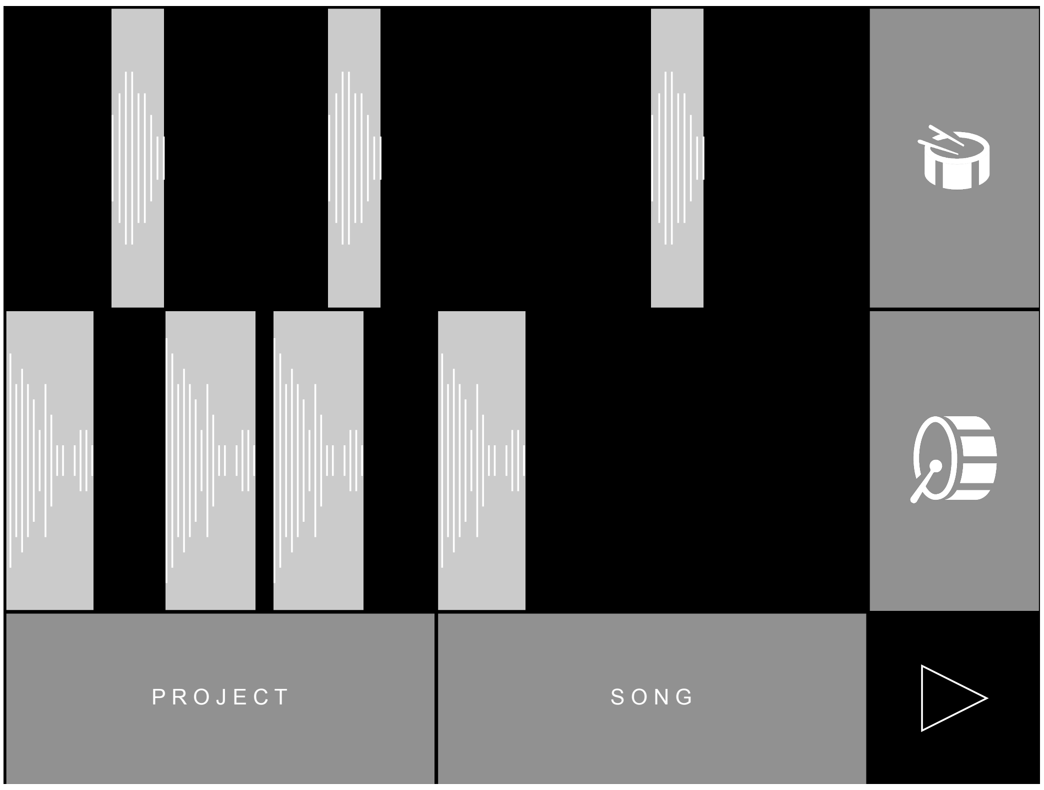


Figure 2.4: The performance <*Part Library, Drums*> as a diagram and in the prototype

ited" events that actually are defined on higher abstraction levels are just implied on lower levels and cannot be edited there, which is why they are pictured darker. They are also in the background and can't actually collide the events on those lower levels.

2.1.5 Loops

The previous section and Figure (2.4) may have provoked this question: Why are there only two events (beats) defined for *Base* and *Snare* over the whole duration of the song? Also, the audio preview in Figure (2.3) clearly suggests that there is more going on in the drums during the event that plays them in the *Song*.

This has to do with the role that the *Score Matrix* plays in our model. Figures (2.5) and (2.6) display two more performances from our example. The meaning of the events in all figures will be clear, after we revisited the *Score Matrix* in more depth.

The event list that a score matrix cell "contains" can be shorter than the original length of the associated part (column). In that case the system understands that the list is supposed to be repeated (looped) to "fill up" the whole part. To make these shorter lists editable without zooming, the interface graphically scales them to the whole width of the cell (column, part). Of course that also has disadvantages and the user might ultimately need a way to switch to a "real time" view in which the time scales of all cells are comparable and all repetitions of a loop are visible. But for communicating the concept, this loop centered view is sufficient.

Our simple looping mechanism provides multiple benefits:

1. It obviously gives a lot more meaning to events that relate to voice groups.

2. It reflects a rhythmical and repetitional aspect of music that parts alone can't express. Repetition occurs on different scales, depending on voice and part. That means, within the same part, one voice might

repeat a long segment only a few times, while another might repeat a short segment many times. This cannot be modeled with hierarchically segregated parts because their partitioning of the time axis applies across all voices.

3. It lets the user exploit the hierarchical nature of rhythm, reducing redundancy in the music notation and, thereby, increasing the efficiency (practical directness) of interaction.

4. As a side effect, it introduces an elegant form of variation. In our example, a drum loop is defined on the whole song and that loop is then extended (varied) in the chorus because the chorus defines its own loop that happens to be of the same length (Figure 2.6). In the seminar, we still thought this to be the most difficult requirement to satisfy.

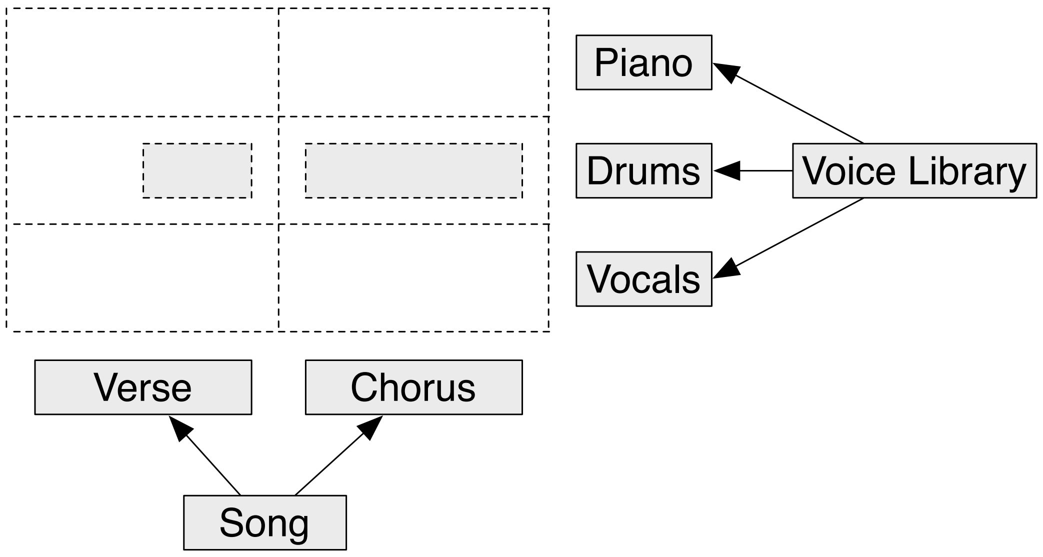
The combination of part-graph and looping makes a powerful tool. Remember that, in one and the same part, different voices can loop event lists of different lengths, and these loops can add up to complex rhythmic patterns (polyrhythms).

Concerning events, there's one detail left to mention. Those events for *Base* and *Snare* in the *Song* (Figure 2.4) make up a short loop that is repeated throughout the song. Now, we saw in Figure (2.3), that the global *Drums*-event begins playing the drums some time into the *Song*. No matter where that global event would exactly start, the drum loop would always start playing from the start of its own definition, that is: on the base beat.

Ultimately, it was necessary to make event editing consistent between atomic and composed voices. An event for an atomic voice like a single piano key must directly represent the corresponding audio being played from the exact position of the event.

Again, this doesn't lend itself well to verbal description and may all sound complicated but should feel very natural when using the interface. Users expect voices to start playing exactly where events start. They don't expect the second *Drums*-event to just turn on a "channel" that is already playing something at this point. This principle is in line with traditional score notation as well as with sequencing software. It provides even more benefits together with loops, as we'll see in the next section.

In the following, we'll describe how certain tasks are performed in the prototype, starting with navigating through part- and voice graphs and then proceeding with graph- and event editing.



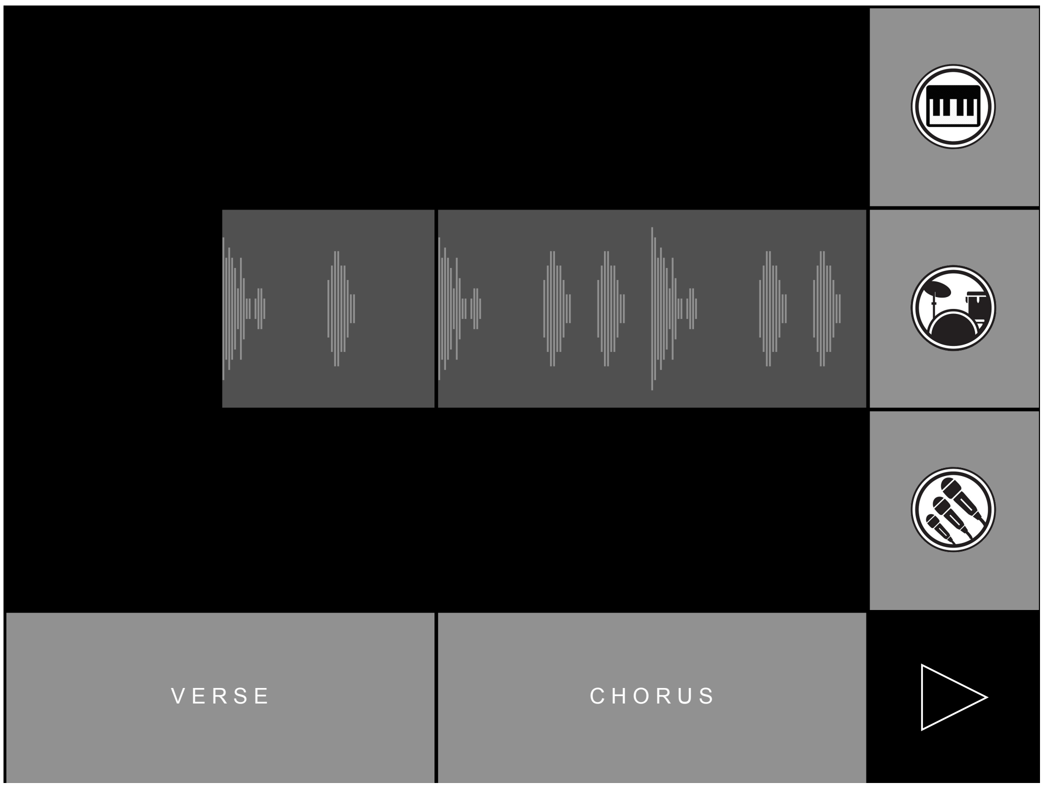
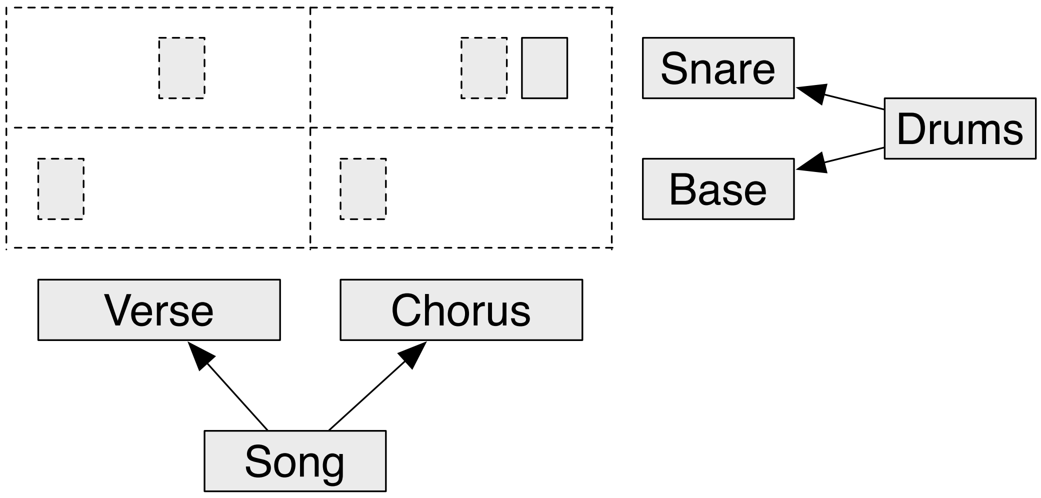


Figure 2.5: The performance <*Song, Voice Library*> as a diagram and in the prototype



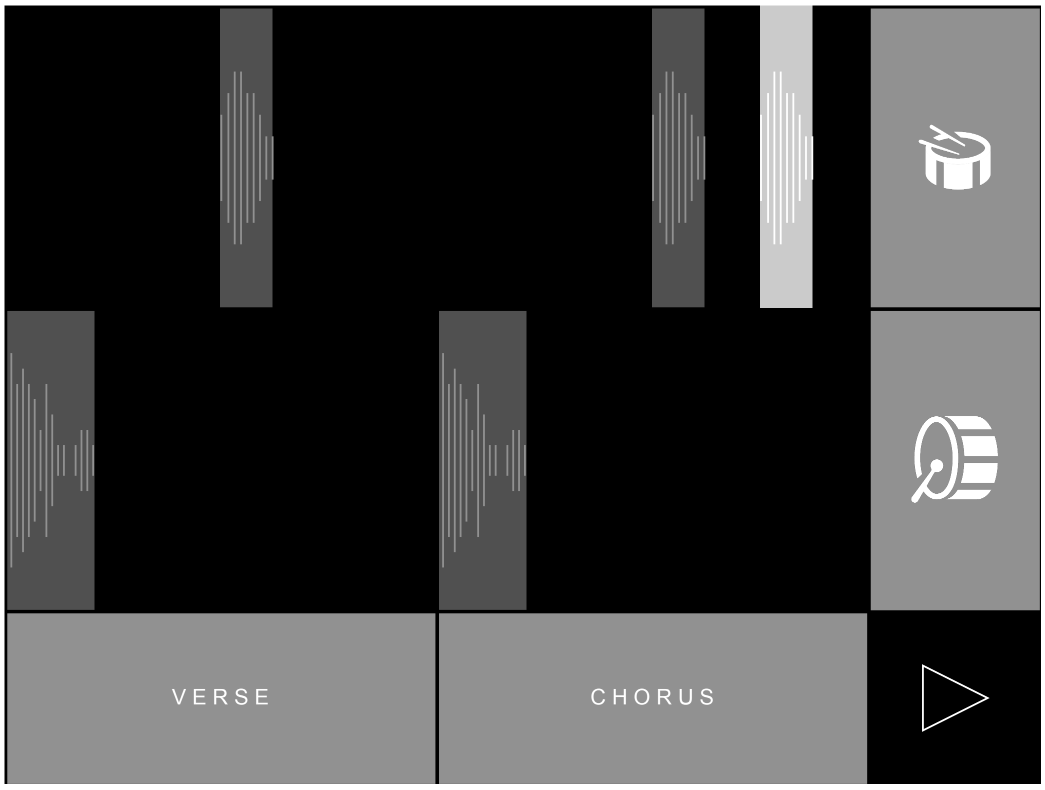


Figure 2.6: The performance <*Song, Drums*> as a diagram and in the prototype

2.2 Navigation

We have two types of nodes: Parts are temporal abstractions and represent time intervals. Voice abstractions refer to some sound. Here, atomic voices refer to single audio files and, consequently, composed voices (voice groups) refer to a set of audio files. When we talk of a node and don't specify it further, it might be atomic or composed.

The user can build DAGs of both of these abstractions. And we try to treat both equally so that the interface presents them as analogs of each other. This helps to communicate the domain model and its available operations as well as to simplify the interface.

As we mentioned earlier, the current performance holds the user's current selection of a part and a voice. This combination defines what he sees and edits. The score matrix depicts the actual content in form of events. There may be no events at all for the current performance. In that sense, we may understand the current performance as a 2-dimensional query and the events in its score matrix as the query result.

2.2.1 Navigating Down

Basically, two context dependent (local) navigation operations are sufficient for browsing and selecting all material in the library: Moving down in a DAG and moving up.

Moving down means the user chooses a sub-node of the current performance that will become its new scope and reference point. For example, he might want to view and edit a certain chord of the currently selected chorus. Or he might want to "enter" the toms from the currently selected drum set.

Figure (2.4) shows how the current performance would change from Figure (2.3) if the user moved down to the composed voice labelled as *Drums*. At this level in the example, one cell of the base drum contains three events and the other contains one.

To navigate down to the drums, the user needs the graph editor, which he can slide in with a simple swipe. In Figure (2.7), a swipe-up on some sub-part revealed the part editor, while in Figure (2.8), a swipe-left on some sub-voice revealed the voice editor. Since both are analogs of each other and look similar, it will suffice to focus on the voice editor.

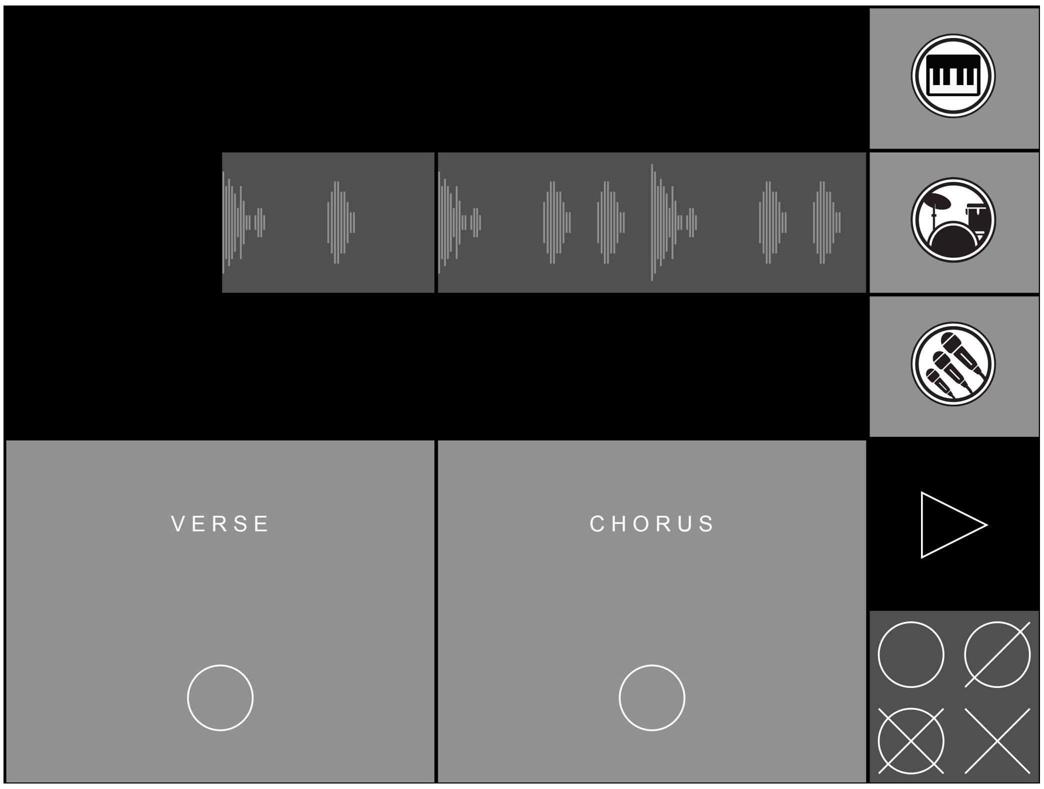


Figure 2.7: A mockup of the part editor

When navigating down to the drums, the screen depicted in Figure (2.8) turns into the one in Figure (2.9), which corresponds to Figure (2.6).

Since the sub-nodes of the current performance determine the content that is actually being displayed and edited, there are several ways to design the act of "entering" or "opening" them. In our design, a double tap on the graph-editor side of a sub-node selects that sub-node as the new scope. We chose the double tap for a few reasons:

1. It resembles *opening* files and programs.

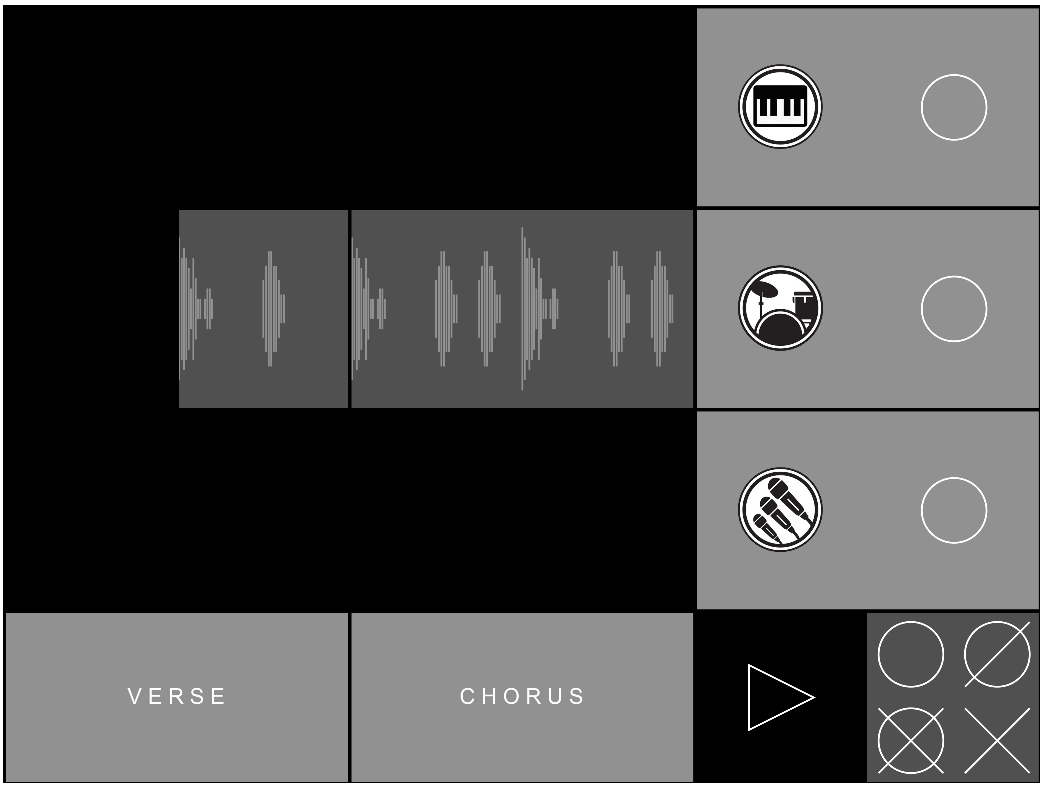


Figure 2.8: A mockup of the voice editor

2. It prevents the user from accidentally changing the context.

3. It is in line with our overall approach to ...

   (a) use simple and natural gestures for frequently used direct operations like local content editing.

   (b) use more advanced gestures for infrequently used meta operations like changing the editing context.

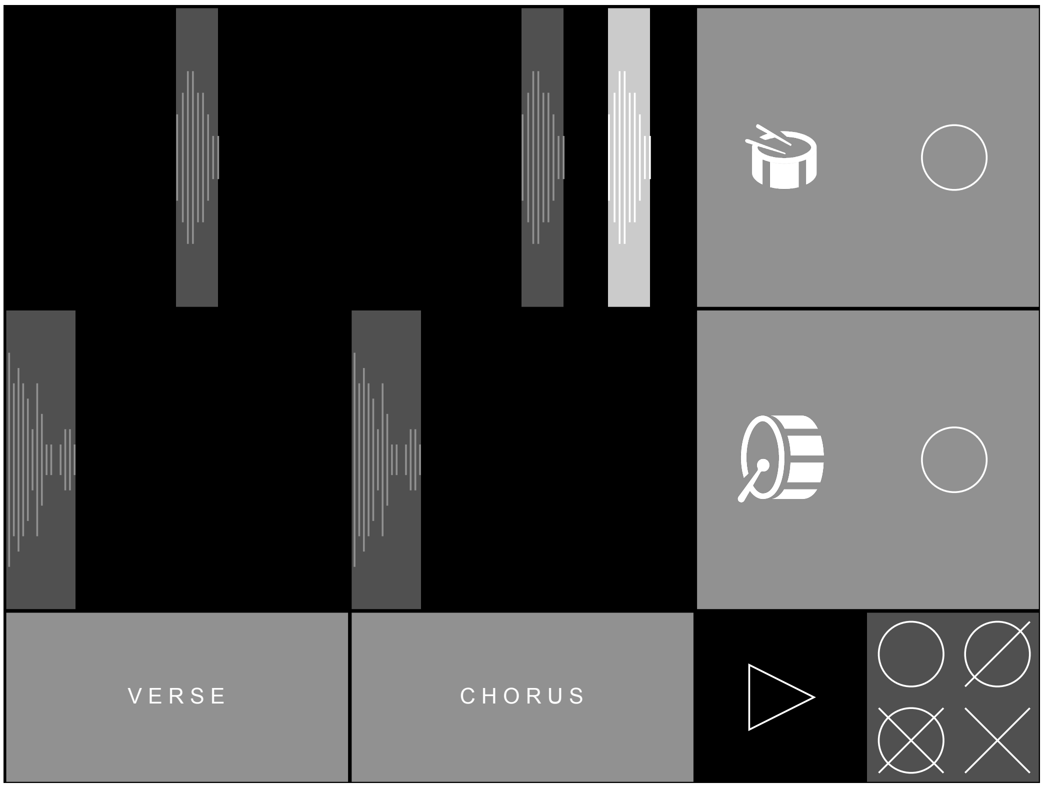


Figure 2.9: The performance <*Song*, *Drums*> and the voice editor

2.2.2 Navigating Up

One important reason for why we have part- and voice abstractions in the first place is that they allow the user to reuse material. He may use the same part with different voices (instrumentations) and vice versa.

But even more, he may reuse (reference) the same node within different super-nodes. For example, a pop- and a jazz instrumentation may both employ the same type of piano. Or two verses may employ the same chords but in different orders. In these cases, there are nodes (piano and chords) that have multiple parents in their respective graphs. Technically, DAGs are not really trees but our abstractions are better understood and visualized as hierarchies.

Nodes that have only one sub-node would typically be redundant. But most nodes would have only one super-node. Moreover, when the user navigates up, he most often goes to that super-node from which he came. So, to keep things simple, we don't make the user decide which parent he means when navigating up but, instead, always go to the one from where he came. It is still possible to navigate to all nodes in the library because the library acts as the default root of all nodes that would otherwise have no parents and be roots themselves.

In our design, a two-finger tap on the editor-side of sub-parts or sub-voices shifts the interface one level higher in the respective dimension. The two fingers don't need to tap the same sub-node but can be anywhere on either sub-parts or sub-voices. This is analog to panning and zooming, where two fingers are needed to grab the whole score matrix and, thereby, relate to a higher abstraction level.

2.3 Graph Editor

The user can perform five basic editing operations on the current performance. These five are a minimal set of operations that is still sufficient to build any thinkable ordered DAG:

1. Adding another sub-node
    (a) Group one or more existing sub-nodes
    (b) Create a new node (in the library)
    (c) Reuse an existing node (from the library)
2. Splitting (or ungrouping) a sub-node
3. Removing a sub-node (from the performance)
4. Deleting a sub-node (from performance and library)
5. Moving a sub-node to a different position

The first four of these operations can be performed through the buttons that Figure (2.10) highlights.

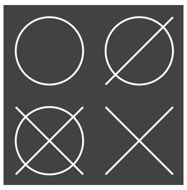


Figure 2.10: The graph editor allows to create (group), split (ungroup), remove and delete sub-nodes.

To perform certain operations on one or more sub-nodes, the user must first select those nodes. He can do that by tapping sub-nodes in the graph editor. In Figure (2.11), the user did select *Drums* and *Vocals.*

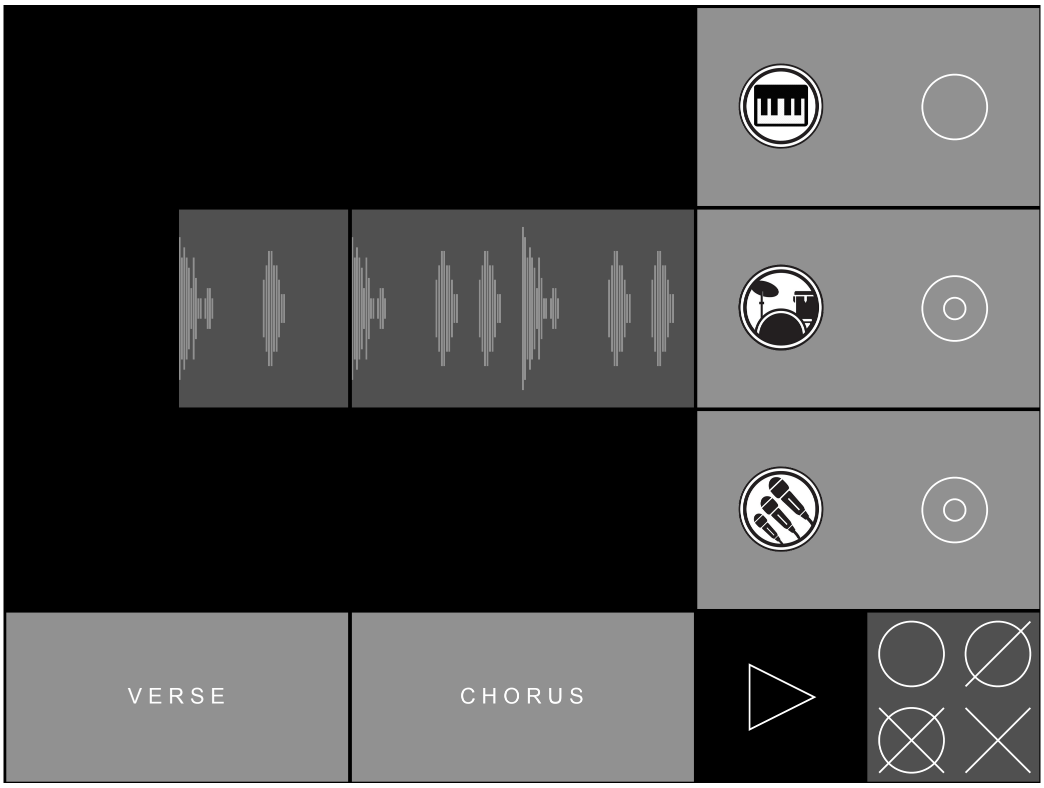


Figure 2.11: The sub-nodes *Drums* and *Vocals* have been selected for performing graph editing operations.

In the following, we'll explain in more detail *what* these operations mean in terms of graphs.

2.3.1 Adding

There are three ways the user might want to add another sub-node to the current performance.

Grouping

Grouping replaces one or more existing sub-nodes of the performance by a new node that becomes their parent. For example, the user may turn the three tom voices of a drum set into one voice group called *Toms*. Or he can create a piano instrument by grouping all piano keys. On the time-axis, he may summarize 8 bars as a part called *Verse 1*. Grouping re-structures the form (syntax) and does not change the audio output (semantics) of the performance.

Creating

When no sub-node is selected, the create/group button simply creates a new leaf node and inserts it into the performance. In terms of parts, the user may start creating a new song in the context of an album, insert a new bar into a chorus or start a totally new idea in the library. In terms of voices, he may add a sample to an instrument or create a new voice as a container for other voices.

Note that users can compose those graphs through a global top-down approach by enriching and splitting high-level nodes as well as through a more local bottom-up approach by grouping low-level nodes.

Reusing

Reusing means the user can choose an existing node from the library and insert it into the performance. Thereby, he can reuse material across all songs, projects, ideas, instrumentations etc.

Of course, the interface offers only those nodes that are not an ancestor of the current performance because the DAGs must remain acyclic. For example, a song cannot contain itself as a sub-part.

That said, a node *can* reference one and the same sub-node multiple times. For example, a song may use the same 2-bar transition from verse to chorus whenever a verse leads to a chorus (repetition). Or there simply are two different instruments of the same type and the user wants to write their notes separately.

2.3.2 Ungrouping

Ungrouping a sub-node removes it from the performance and replaces it with all its children. If the original sub-node is not referenced by any other node, it will also be deleted from the library. Like grouping, this operation does not change the audio output.

At this point, one might ask about merging and splitting composed nodes. Those would be reasonable operations, but they can be achieved through grouping/ungrouping. However, grouping/ungrouping cannot be achieved through merging/splitting, so the latter pair does not offer an alternative minimal set of operations.

2.3.3 Removing

Removing a sub-node disassociates it from the performance. The (former) sub-node will still exist in the library and may still be referenced from other nodes.

2.3.4 Deleting

Deleting a sub-node removes it like described above but also deletes it permanently from the library.

However, there is a restriction to the deleting operation. The removed sub-node or any of its descendants will not be deleted if they are being referenced by other nodes. That way, the delete operation will not effect other material.

For example, the user deletes a verse from a song. The verse uses four bars. The third of those bars is also used in another verse or song. The operation will delete the verse and three bars but keep the third bar in the library.

2.3.5 Repositioning

A node defines an order on its sub-nodes. The interface displays the sub-nodes of the performance in that order. The user may want to edit the orders that the performance defines on its sub-parts and sub-voices. To that end, he can simply long-press a sub-node, drag it to its new position and drop it there. This should be reminiscent of moving columns and rows around in spread sheet software.

2.4 Event Editor

2.4.1 Panning

To move (pan, scroll) the score matrix (content view) around, the user first has to grab it with two fingers. The interface signifies that the user grabbed the whole thing by marking it with a half transparent red overlay as can be seen in Figure (2.12).

The user can then zoom and pan with one or both fingers. He can even temporarily release all fingers, the score matrix will keep its momentum and slowly decelerate (inertia). As long as it moves, the user can re-grab it and proceed panning with one finger. As soon as it stops, scroll mode ends and the overlay disappears. We provided the reasoning behind this interaction design in the project paper [18].

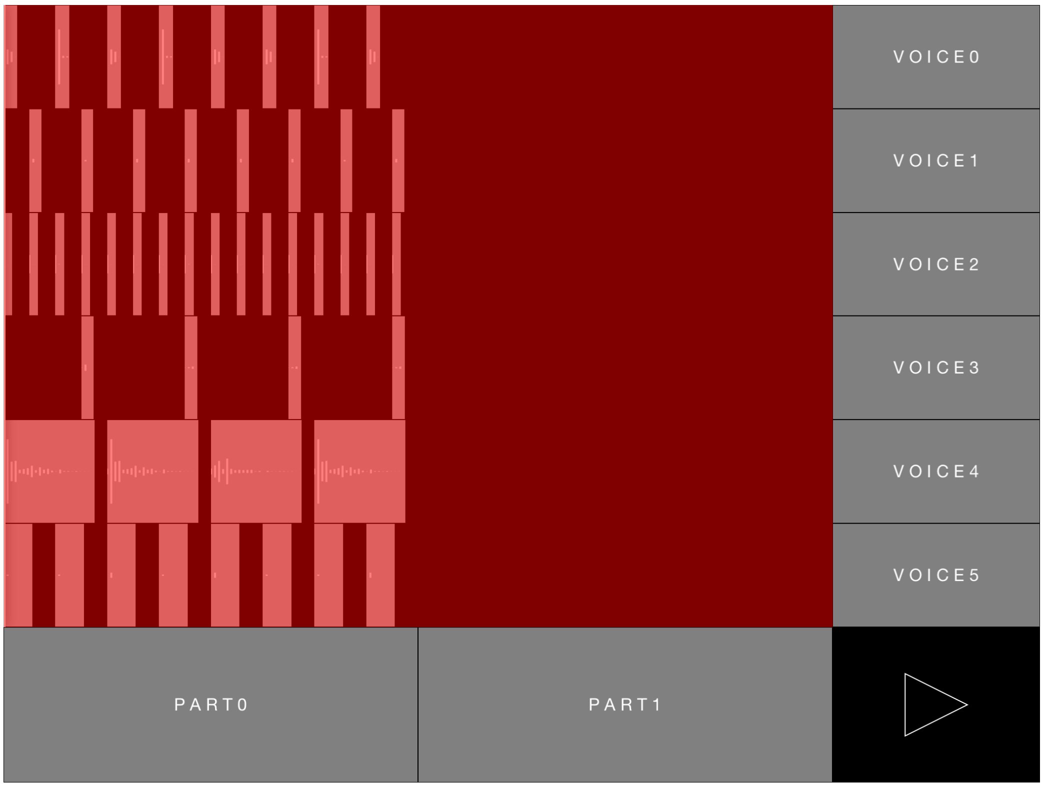


Figure 2.12: The interface in panning/zooming mode

2.4.2 Zooming in

The user can zoom in on a particular location with a two-finger pinch, like on a regular zoomable scroll view (Figure 2.13). Like with panning, the whole content view (score matrix) is visually locked in scroll mode as soon as both fingers touch the content view.

2.4.3 Zooming in on a Part or Voice

Because parts and voices are independent concepts, the interface should treat them as such. Consequently, zooming is direction-sensitive in our implementation. How much a pinch gesture zooms into voices and into parts depends on how much the fingers actually pinch in either dimension. The user can zoom in only on voices or only on parts or do anything in between Fig-

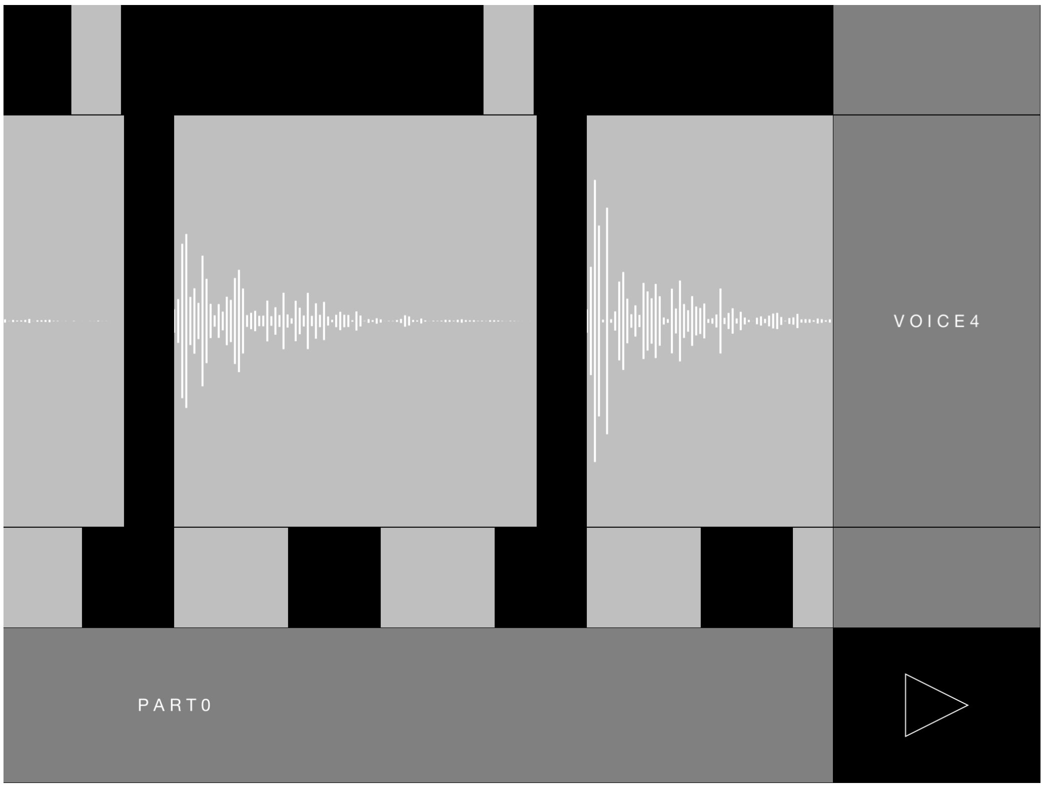


Figure 2.13: The interface zoomed in

ures (2.14) and (2.15).

This also leads to greater articulatory directness than conventional scroll views. Since both fingers remain on the exact spot in the editing plane during the pinch gesture, it feels much more like actually grabbing the content instead of performing an arbitrary gesture which the interface then translates into zooming.

2.4.4 Zooming out

Of course, the pinch gesture can also zoom out. The difference is that zooming out is not location-sensitive. A zoom out reflects the intent to bring back into view what is currently outside of it, i.e. to regain overview.

In conventional scroll views, zooming out is location sensitive. That means that, while zooming out, the viewport might collide with the bound-

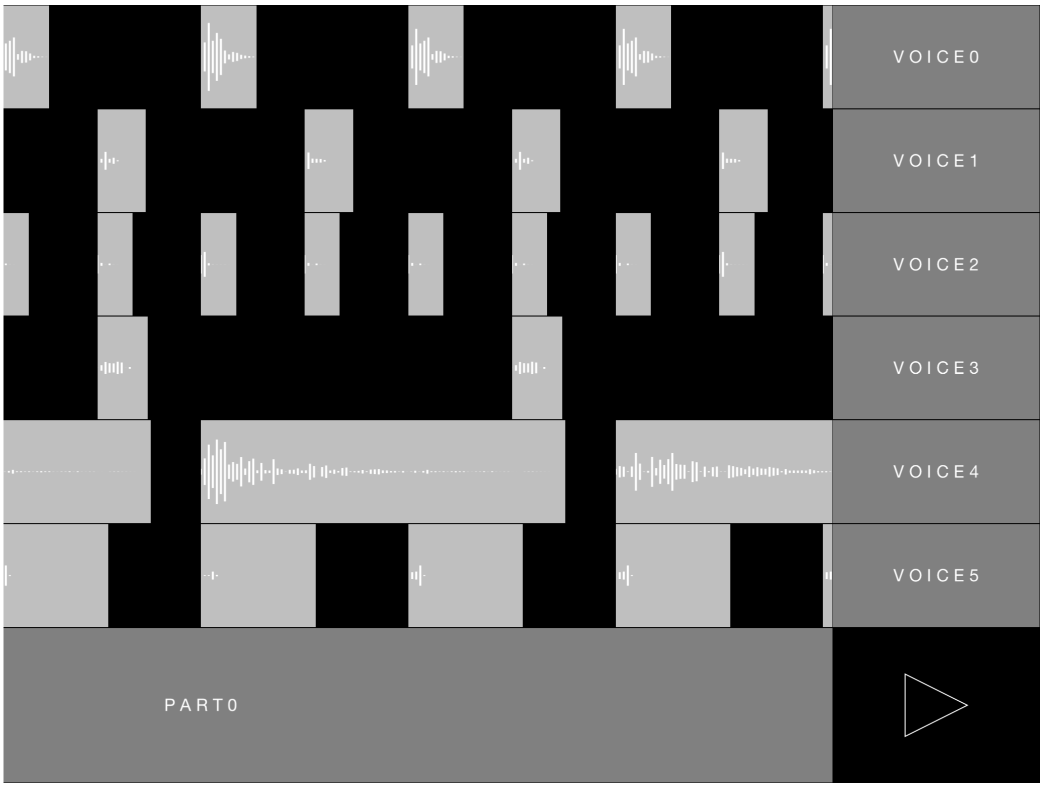


Figure 2.14: The interface zoomed only horizontally.

aries of the scroll content view. Typically the scroll view first pretends as if it was possible to move the viewport beyond the content view, and after the gesture is complete, the viewport abruptly snaps back into the bounds of the content view.

In contrast, our "scroll view" implementation performs a smooth transition on zoom out. It not only changes the zoom level but also the viewport position so that the viewport position perfectly fits the content view (score matrix) when the zoom level is back to 1.0 or "normal".

This is especially helpful in our interface concept because, here, all content (voices and parts) is visible by default. The user can quickly return to this view with one gesture. The detail level with which he interacts is not just determined by zoom level but also more explicitly by selecting a certain depth in voice- and part graph. Therefore, our design makes it unnecessary to have

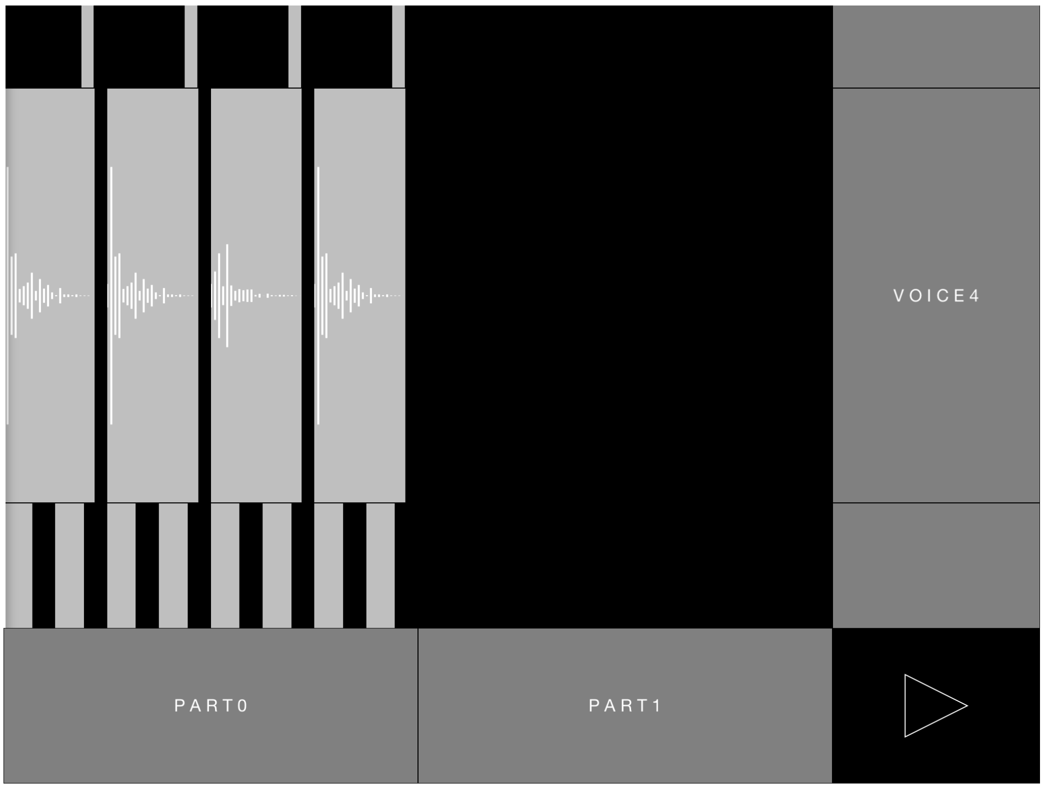


Figure 2.15: The interface zoomed only vertically.

a score matrix that is so complex that it can't be summarized on one screen, like it is often the case in the arrange view of conventional DAWs.

2.4.5 Adding an Event

The user adds an event (note) by simply drawing it with a one finger swipe, thereby he also determines the length of the event. Actually, a one finger swipe does not directly create an event but marks a time interval within a score matrix cell. If no events yet exist in that interval, the marking turns into a new event. Otherwise, all events overlapping with the marking are deleted. The marking is restricted by musical quantization.

2.4.6 Deleting an Event

Figure (2.16) shows how the user can delete several events with one gesture by drawing a marking over them.

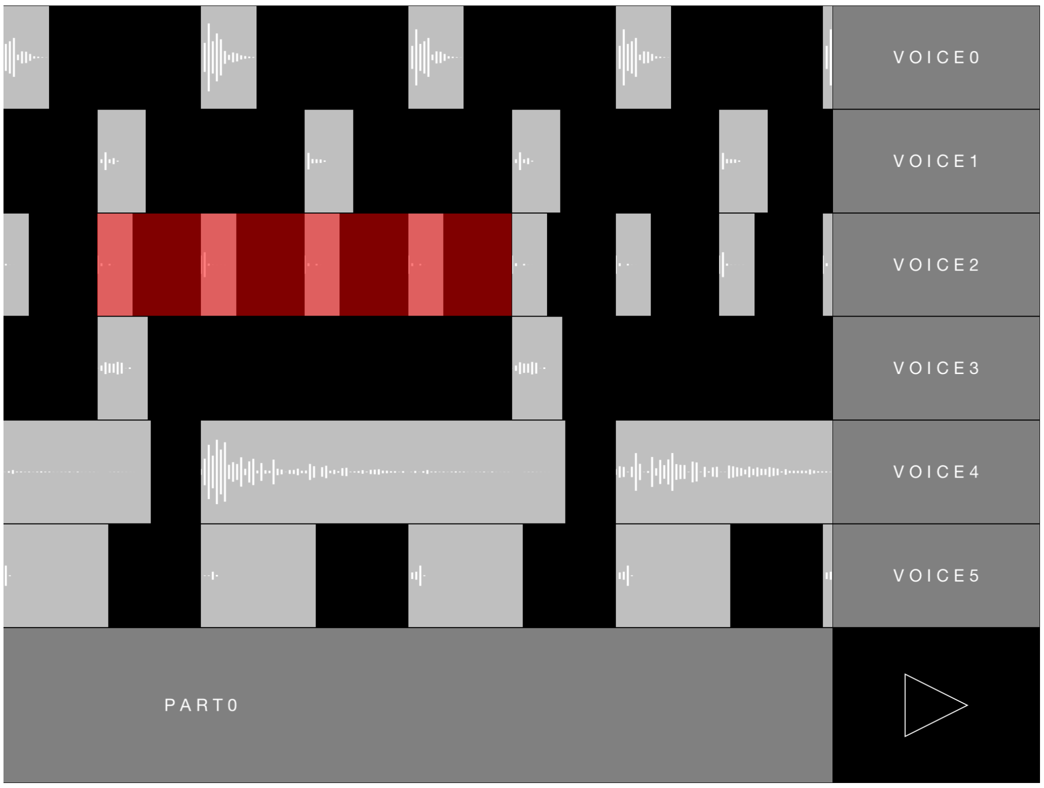


Figure 2.16: In this case, the editing range will delete four events.

Quantization ensures that editing gestures never effect more detail than the user actually can control. Apple recommends that interactive elements are at least 44 points big in either dimension. The interface is supposed to always pick the most detailed musical quantization raster that satisfies that requirement for the current zoom level. However, the implemented component of our prototype doesn't yet realize this adaptation.

The resulting *Quantization Cells* act as our interactive elements. So the user does actually not span an interval of arbitrary length but touch a discrete number of these cells. One advantage of this is that a simple tap without any swipe already marks a single cell, and that allows the user to delete and create events with a single tap.

Before we move on to our evaluation, we have to mention that we totally skipped the whole topic of actually playing the current performance which would, of course, be central to any music application. One reason for that is that we're investigating a much more fundamental layer of music sequencers to which playback isn't even that essential. The other reason is that we don't have a fully functional product and our implementation has no caching mechanism that would actually allow real time playback.

Chapter 3

Evaluation

3.1 Methodology

In the seminar, we had laid out our overall approach and specific methodology in great detail. Here, we just re-establish that context.

Not to utilize user feedback at all would be a huge mistake. It would also be a huge mistake to build the whole design exclusively on user feedback. Finding the sweet spot between those extremes is more of an art than a science.

Our orientation for this thesis is Johnston's [26] approach to research and design of new musical interfaces (Figure 3.1). His idea of the process involves listening to users and understanding their belief systems. At the same time, it relates the user study to a specific novel design and, thereby, allows the design process to be driven by a grand vision.

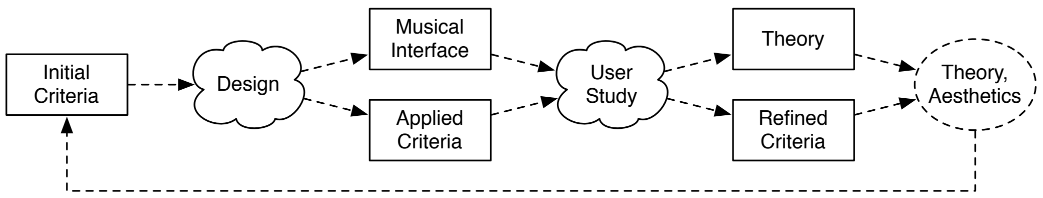


Figure 3.1: Research- and design process for new musical interfaces, according to Johnston [26]

This latter aspect is crucial. The standards that users already know limit their perspective. Observing how they manage within that old frame of reference is not enough to create a new frame of reference. Therefore, in order to design a new interface, we must confront users with a new interface.

In terms of Johnston's methodology, we now have a new musical interface as well as applied criteria. The next step is to design and conduct a user study. We explained in the seminar how Johnston understands the user study more specifically as a "user experience study". This study is supposed to tackle three questions:

1. Do the instruments [musical interfaces] that have been created meet the design criteria identified during design?
2. How do musicians experience them?
3. What are the relationships between the characteristics of the instruments [musical interfaces] and the musicians' experiences?

3.2 User Study

Measuring creativity is a tricky task. Most quantitative methods to measure the creativity of people or the creativity-support of systems rely on self reports and language analysis.

For example, Latulipe et al. [4, 7] did incredible work on developing a quantitative index for creativity-support. Their work reminds us of the creativity factors that we have to look out for in our evaluation: "Collaboration, Enjoyment, Exploration, Expressiveness, Immersion, and Results Worth Effort".

Other research used hard data from system logs of user interaction but, then, had to interpret that data in terms of creativity [31]. In a publication on the Evaluation of CSTs, Hewett et al. [21] stated:

> "... qualitative methods [...] are particularly well-suited to gaining a deep understanding of the needs and methods of a commu-

> nity of practice. When performed by an experienced researcher, in-situ observations and semi-structured interviews can yield a rich set of data in a relatively short period of time (Millan, 2000), providing the information necessary for subsequent design and evaluation phases."

Particularly in computer music research, it is very common to conduct qualitative studies with a small number of musicians because such investigations can go into depth and actually illuminate the sometimes intricate working practices of creative users. Much of the data that is gathered comes from interviews.

For instance, Carter, Eaglestone and their colleagues [5, 14] analyzed the cognitive styles of composers. Their initial study was based on interviews with four composers. Donin and Theureau [9] analyzed the long term process of just one composer.

Even in general HCI research, the benefits of assessing small groups and then moving on to the next iteration have been acknowledged. Jakob Nielsen [34] stated:

> "Elaborate usability tests are a waste of resources. The best results come from testing no more than 5 users and running as many small tests as you can afford."

We conducted a qualitative evaluation of our interaction design through assessment sessions with five participants. The participants are regular users of sequencer software. Their musical expertise ranges from amateur- to professional level. On page (101) in the appendix, we list their detailed profiles.

The evaluation had to satisfy three criteria: It had to be fast and cheap. It had to produce a lot of qualitative data. It had to be applicable at a relatively early stage and produce valid feedback even with just mockups. To provide for all that, we employed a blend of five different evaluation techniques:

Feature Inspection Nielsen [33] described Feature Inspection as one of eight usability inspection methods. The developer analyses how well features are implemented and how well they complement each other in specific tasks.

In Section (3.4), we'll adapt this technique to our needs. We'll interpret our requirements as the features that we want to offer, estimate how well they are implemented and reflect on how they work together to serve the intention of our proposed model. To put the estimations in perspective, we'll also compare them to GarageBand.

Field Study Field studies have been used in HCI to observe actual behaviors of users in their real world setting [27, 37, 40].

We observed and interviewed our participants in their own working environment, that is: at their home, in their basement studios and in the sound studio of the Konstanz University of Applied Sciences. This made it possible to open the participant's own projects and see their working habits live. It also enabled the participants to exemplify their statements with real world examples and to reflect on their own material.

Paper in Screen Prototyping Particularly for mobile applications, in screen prototypes have proven to be highly effective in gathering feedback on design concepts at an early stage of development, in HCI research [2] as well as in the industry at Apple Inc. [1].

We created partially interactive mockups in Keynote and presented them to the participants on an iPad, which is our intended platform. Thereby, we were able to demonstrate not only the navigational structure of the interface but also particular actions like going in and out of sub-nodes, opening the detail pane of a sub-node or performing graph editing operations like grouping and un-grouping.

Interview Interviewing is a very basic evaluation technique that lets the interface designer assess specific aspects of the interface or the user's thought process and feelings [23, 28, 38].

Part of our assessment sessions were basically unstructured interviews. We had a rough structure prepared and made sure to cover certain aspects, but the questions were mostly open. The interviewing technique also allowed us to respond to what participants found important or interesting. Thereby, we could create a relaxed atmosphere that, in turn, encouraged participants to think aloud.

Think Aloud According to Nielsen [32], "Thinking aloud may be the single most valuable usability engineering method." And indeed, it is has a long history in HCI [35, 41]. Think Aloud intents to let users literally think aloud about what they try to achieve, what irritates them etc. It is a simple, cheap and very effective method that can reveal the most important insights with just a handful of participants.

Throughout the assessments, we encouraged participants to think aloud and freely associate, and we posed some hypothetical tasks to them to further facilitate thinking aloud.

We captured all sessions as audio recordings and also got some sketches from participants. The recordings have an average length of 102 minutes. In most cases, our correspondence and conversation with the participants extended beyond what was recorded. Nonetheless, we had 8.5h of audio recordings which we analyzed afterwards.

The sessions were structured as follows. Stages (1) - (3) are completely independent of our own ideas and it was our intent to assess the participant's unbiased thought process and needs before introducing them to our design concept:

1. We recorded some personal data (sex, age, profession) and assessed the participant's musical experience regarding instruments, live performance, composing, software, platforms etc.

2. We assessed the participant's understanding and practice of the music composition process and the role that software plays in it. After taking

a general account of that, we made sure that the following particular aspects were covered:

(a) Mobility

(b) Preparation and Reuse

(c) Voice abstractions

(d) Temporal abstractions

3. We assessed the participant's wishes and visions for better composition software. We specifically asked what annoys the participant in his setup and software and whether he has to do any workarounds to achieve a goal.

Stages (4) - (6) examined how the participant's characteristics manifest in their approach to learn and understand the concept of our prototype:

4. With the help of object graphs and sketching, we first established a mutual language together with the participants. This step resembled a bit of Domain-Driven Design [15]. We got clues about their specific mental models of music composition and about how they relate to our proposed model and the graph-related operations that come with it.

5. Using partially interactive in-screen mockups in Keynote on the iPad, we presented a conventional scenario in the model and visual language of the prototype and also pointed out the simplifications and possibilities that our design offers. The example scenario that we employed here is the same that we outlined in Chapter (2). During this introduction, we encouraged the participants to freely associate, comment or ask questions.

6. To see how the participants actually perceived our concept, we asked how they would apply it to specific use cases. In the end, we also casually encouraged an open discussion to elicit what was really going through their minds.

Figures (3.2) and (3.3) show three participants in their working environment reflecting on our design.

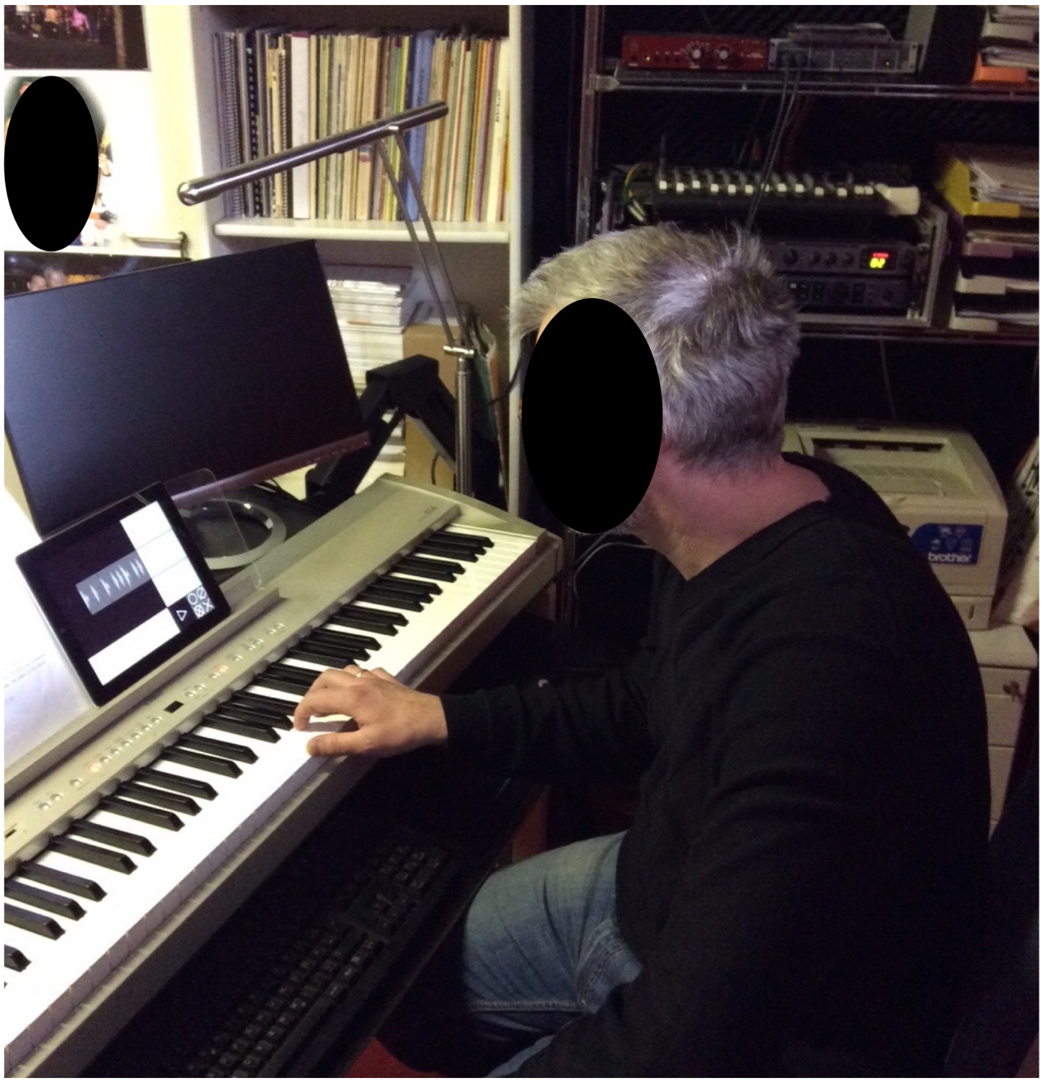


Figure 3.2: Participants B during his assessment session

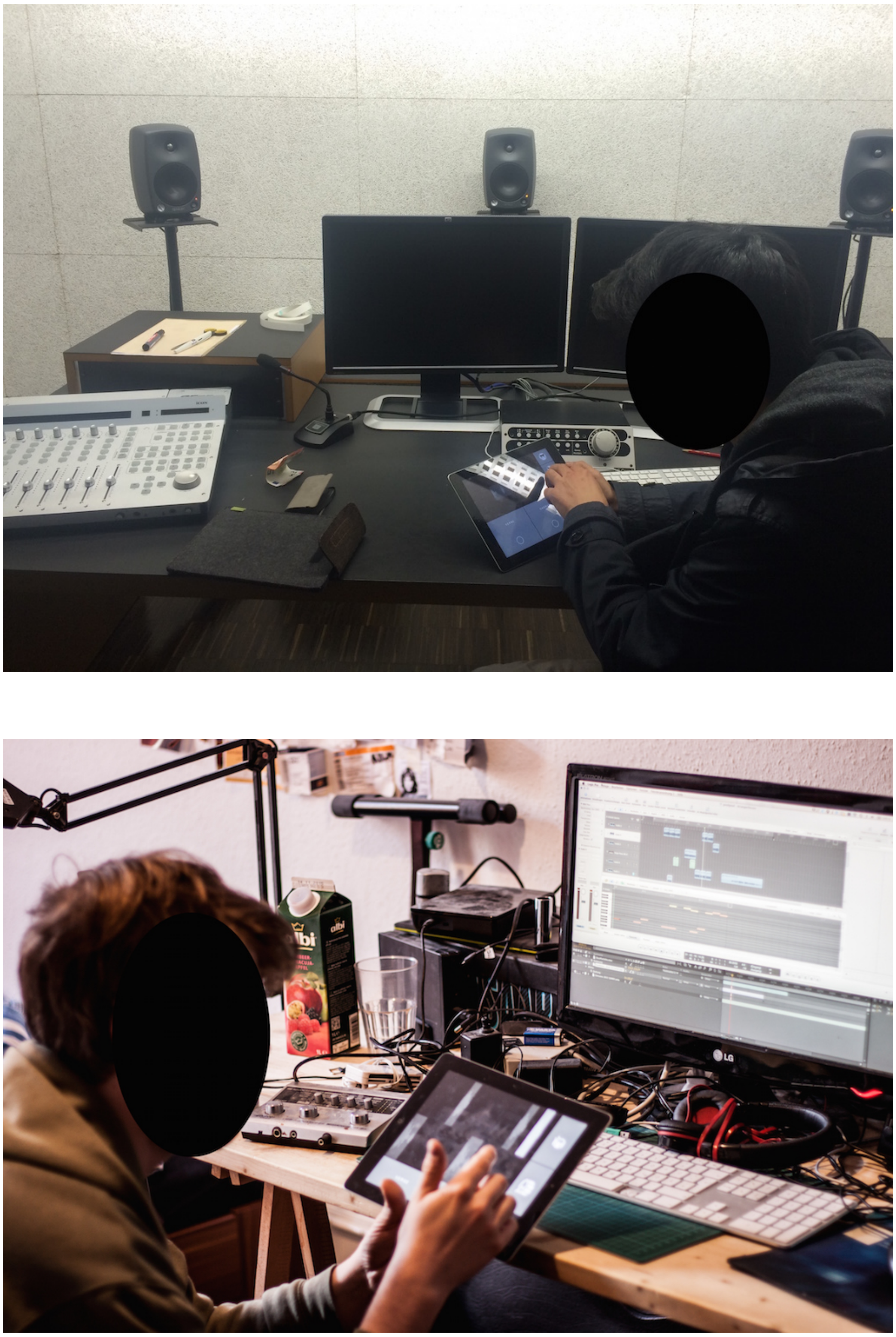

[redacted]
[redacted]

Figure 3.3: Participants D and C during their assessment sessions

3.3 How the Participants Usually Work

With regards to general working practices, we observed surprisingly many commonalities between our participants. This allows us to list just those characteristics here that apply to all participants:

3.3.1 The Process and how it Relates to Software

1. They do most of their composing solo, i.e. not through collaboration.
2. Their inspirations and new ideas often come independently of any software use, on the go, spontaneously or when playing instruments.
3. They use software to capture ideas on the go, either with a Macbook (four participants) or with an iPad (one participant).
4. Sometimes, they have to capture ideas very rapidly, leaving no time for technicalities, clean structure or naming anything.
5. They care about the creative composition phase and have little knowledge of- or interest in the mix and production of their projects.

3.3.2 Accessing and Re-using Material

1. They collect many ideas on their computers, most of which lie there for long times and never get "finished".
2. They rarely come back to old ideas that haven't been on their minds for some time.
3. They prefer starting new projects over perfecting old ones.
4. They practically never combine different ideas from different project files into a new project. One reason for that is that, when they start a new project, they clearly identify it by the unique idea, vision or mood that they have in mind for it. That specific intention separates projects

from one another. Even over the course of working on a project, they do not think of it in terms of formal characteristics like its key, how it employs tonic, subdominant and dominant and so on, not even its specific chord progressions.

5. Within project files, they dump or store ideas related to the project for later use. They either put them on muted tracks or behind the actual piece of music (in time).

3.3.3 Input and Instruments

1. They employ software early in the composition process. On the other hand, software use is always intertwined with playing and recording real instruments. They record material not only for the end result but also as an intermediate step for experimentation and reflection during the whole process.

2. They use prebuilt content and combine virtual- with real instruments.

3. When composing melodies or chords that are to be recorded with real instruments, they sketch out, pre-listen and develop those with the help of virtual instruments.

4. They do not, as one might expect, record scores for virtual instruments by playing a MIDI-keyboard live and recording that. Instead, they create scores manually, note by note, either in the piano roll editor with the mouse or with MIDI- and regular keyboard in Finale.

3.3.4 Handling Temporal Structure

1. Even without software, they work with- and think in terms of temporal abstractions like verse and chorus. The order of those parts and the exact temporal structure of a piece evolve organically over the whole composition process.

2. They edit temporal structure and part order by manually editing all material with the mouse through operations like cutting, marking, copy/paste, drag and drop.

3. They manually move notes around on the micro-level scale of mili seconds to introduce either groove (swing) or natural irregularities.

4. In addition to mere repetition, they often employ some form of looping in their projects.

3.3.5 Using the Software

1. They use a linear software sequencer as their central application, either Apple Logic Pro (three participants), the linear sequencer in Ableton Live (one participant) or Sonar (one participant).

2. They have issues with technical aspects of their music software like performance, reliability, compatibility, self-sufficiency (as explained in [19]) and the combination of different programs.

3. They are annoyed by- or at least aware of the cumbersome way their desktop sequencer provides panning and zooming. But they somehow got (had to get) used to that.

4. They use only the most basic features of their sequencer software and are far from exploiting its production power.

3.4 From Requirements to Design

Here, we answer the first of Johnston's [26] three questions for the user study:

> "Do the instruments [musical interfaces] that have been created meet the design criteria identified during design?"

Of course our "design criteria" are the 36 requirements on which we elaborated in the seminar [19] and, more specifically, the four core requirements on which we focused in the project [18].

To tackle the above question, we will not only interpret our assessment sessions, but also compare HAIL against GarageBand in a more formal (if not quantifiable) fashion. For this requirement inspection, we distinguish five levels of support:

Unsupported The requirement is unaffected by the interaction concept and would be hard or impossible to satisfy.

Prepared (+) The requirement is installed in the interaction model and could be integrated into the interface.

Partially satisfied (++) The requirement is somehow satisfied but could be improved significantly.

Mostly satisfied (+ + +) The requirement is mostly satisfied. More work would improve it only a little bit.

Fully satisfied (+ + ++) The requirement cannot be met with more precision.

In the following, we'll assign a support level to each requirement and comment on that rating in more or less depth. Of course, the rating reflects only a qualitative estimation and not a quantitative measurement.

The first two requirements are actually not part of any of the four big categories. They address aspects of cognition.

#1 and #2 Cognitive Styles and Dimensions (+++) It seems our approach is better aligned with the required cognitive styles and dimensions than conventional sequencers. Yet, to analyse this in depth is beyond the scope of this work.

In any case, our participants seemed to tend towards the global style. Participant A said she preferred simpler interfaces, "because you don't have to think about it much, like what kind of beat you want to use, when you don't even have a matured idea. You're not distracted by that [too many options]."

3.4.1 Requirements of Abstraction

Core Requirements

First of all, our design obviously satisfies the identified core requirements of abstraction to a great extent.

#19 GCDCG-Sequencer (+++) Our design is graphical, provides custom abstractions, follows a control flow paradigm and is very general purpose.

Delayed linearisation is enabled in principle although the exact design decisions around this aspect allow for a wide variety of solutions. More experimentation at the interface level is needed to optimise improvisational interaction. However, our design definitely supports the functionality of selecting and playing material at any scope.

#21 Voice Groups (+++) The model as well as the interface representing it allow to form ad hoc groups from arbitrary sets of voices.

However, the requirement demands that composed voices behave just like atomic voices, and our user study amplified the impression that there are more ways to realize global events with which we might want to experiment in the future. Our current solution is probably not optimal.

#25 Temporal Abstraction (++++) Our design also allows to group any ordered selection of existing time ranges (parts) as one bigger time range. The user can treat time- and voice abstractions independently.

Temporal abstractions are pretty straight forward and the solution space is not as vast as for voice groups.

#32 Preparation (++++) Whatever the user creates necessarily ends up in the library. He can follow a bottom-up approach just as well as a top-down approach. That means he can build small snippets and later integrate them into a larger structure, or he can first sketch out the larger structure and then fill in the details by splitting nodes and inserting new ones.

#33 Global Reuse (+++) Part- and voice library provide access to all the user's previous work. When adding a node to the current performance, the user can import any perviously prepared node. Thereby, the existing material acts as a set of building blocks out of which the user can make new creations.

However, the way material can be combined across different keys, tempi and lengths can surely be optimized.

Other Requirements of Abstraction

In addition to mostly satisfying the core requirements, our design is, at least, prepared for all other abstraction requirements.

#20 Transient Voices (+++) The separation of frequency- and time abstractions in our model already resolves this issue to a great extent. Transient voices can be collapsed into voice groups. This is especially valuable since it can be done at any abstraction level. With our approach, transience should no more waste screen space or overwhelm the user.

However, the problem is not completely solved at the interface level. When the instrumentation of a performance drastically changes over time, the fixed mapping of sound sources (instruments) onto the interface's vertical axis can still be awkward. This can be the case, when unrelated projects are viewed side by side like in the performance <*Part-Library, Voice-Library*>. On the other hand, that should not be the default mode of working and it is only a problem when the user still hasn't embraced reusing voices.

#22 Emergent Mix Groups (++) This is one of those features that almost come for free as a result of the underlying model. Through the elegance of the model, mix groups are a given. Our in-screen prototype involves a volume-panorama control in the detail pain of every voice.

However we didn't design or evaluate the interaction with mix groups in any more detail. An issue may be that mixing one voice-group can change its output level and that effects the mix of its super-voice. Overall, we haven't spent enough focus on this requirement to assume it is already mostly satisfied.

#23 Juxtaposability (+++) This issue mostly resides on the interface level. It is a question of how the model is presented. Our design allows to put voices into groups, which resolves the issue to a great extent. Also, the user can change the order of sub-voices in a voice group, so he can view and edit voices of his choice side by side. Through parts, the interface also allows to see formerly unrelated material side by side. Because the user can re-use parts and voices, he can create different "views" on the same material. He can utilize this feature for editing.

But juxtaposability could be taken further. If voices have no common ancestor (voice group) in the voice graph, the user should still be able to edit them side by side for a common time line. Also, with complex voice hierarchies, the effort to get a certain set of voices into view may be discouraging.

One solution would be a temporal virtual performance entity that only serves the purpose of editing. Users would put arbitrary voices into this entity like they would bookmark a page or add an item to a shopping cart.

Another would be that parts and voices could be collapsed and expanded in the performance view, so that different abstraction levels could be viewed and edited at the same time.

#24 Groove (+) This is also an interface issue. Our design distinguishes the discrete quantization raster from with micro level offsets. The question for future design will be on what kind of graphical parameter to map these offsets.

#26 Liveness (++) As we commented on requirement #19, more work needs to be done on the improvisational interaction of our interface although interactive triggering of material and immediate playback are quite advanced in our design.

#27 Direct Audio Processing (+) Conventional sequencers do not provide time ranges as explicit objects. Therefore they can apply audio processing only to the abstract concept of streams (channels). In our model, on the other hand, everything is based on concrete audio data and composed material like a performance is identified by its voice and time range, which makes the future implementation of direct audio processing straight forward.

Note how this reflects Requirement #19 as it abandons the data-flow paradigm in favor of the control-flow paradigm.

#28 Scalable Audio Processing (+) In order to play high-level abstractions reliably in real time, we would need some caching mechanism. Hierarchical data models perfectly lend themselves to caching. In our model, two independent hierarchies are conjointly used to access the same data. This makes the implementation of caching more tricky but opens up a range of potential solutions.

We haven't yet implemented a caching mechanism. Improving on that will be an interesting, complex, technical endeavor with great returns for usability.

In contrast, GarageBand has no voice groups and not even a "freeze" function for tracks, so everything has to be rendered in real time. Hardware performance limits the complexity of projects.

#29 Rich Audio Processing (+) Since custom abstractions are given and the model supports direct audio processing, applying multiple processors to one composition entity poses no conceptual cost.

#30 Visual Audio Processing (++) Our tree-like data structures and audio-based model make the presentation of composed objects as audio data feasible and easy to implement. Our prototype already previews the content (resulting effect) of events as waveform sums.

#31 Nondestructive Audio Processing (+) Since everything is based on concrete audio anyway and possibly on hierarchical caching, the original version of a piece of material could quickly be restored.

#34 Versioning (++) Versioning is only supported as an implicit possibility. The user can create different nodes, exchange and compare them. Most of all, he can utilize re-use to create different versions.

Again, the model of conventional sequencers is far from prepared for this because it doesn't have a real structural organization for musical material. There are no explicit sub-entities to which versioning could even be applied. Versioning only starts to make sense when the core requirements of abstraction are satisfied.

#35 Referencing (++) Referencing is intrinsically built into our model. All composed nodes reference their sub-nodes. But the design can be improved in the visual communication of repetition and looping. Also, there is no possibility, yet, to reference composed event data, for example to play the same arpeggio on different keys.

#36 Variation (++) As we mentioned in the seminar paper, variation is the most ambitious requirement. It combines versioning and referencing. And although we didn't explicitly design for this, the concept we propose actually offers a powerful variation mechanism. Through looping and global events, the user can vary and refine a global theme by adding events to it in selected sub-parts.

We didn't tackle the question how events of a theme could be permitted in sub-parts, and there are certainly other approaches to variation that users might wish for.

3.4.2 Requirements of Simplicity

Finally, we briefly reflect on the remaining, more general requirements.

The various aspects of simplicity that the requirements point to were also echoed by our participants. They were likely overwhelmed by software and find retreat in acoustic instruments to maintain a creative mindset. Participant A said about Logic: "I'm too lazy to get into it deeply. This mantle of functionality does really overwhelm you."

Our platform decision alone already greatly elevates the satisfaction of most requirements of simplicity [19].

#3 Constraints (++++) A prototype is by its very nature constrained, but our model also facilitates a certain focus on creating composed audio.

#4 Concentration (++++) The focused functionality, custom abstractions and fluent zoom of our prototype ensure that the user can easily match the interface's focus with his thought process. What the interface presents is exactly what he cares about in any moment.

#5 Accessibility (+++) This requirement seems trivial but is crucial for creative tools. Mobile devices offer the greatest – but not total accessibility.

#6 Mono-tasking (++++) Mobile devices typically enforce mono-tasking in the sense that only one app can be opened and visible at any one moment. Our prototype also hides the system bar, so that only the composition tool is visible while composing.

#7 Single Interaction Context (++++) Our design basically employs only two contexts: navigating the graphs and editing a node. Still, both are part of the same interface and can be visible at the same time. Due to the elegance of the model, many compositional tasks can be accomplished through those contexts. Conventional sequencers employ far more different interaction contexts but allow for less compositional complexity.

#8 Single Physical Mode (+++) The mobile platform encourages the app designer to replace external input controllers like keyboards with the device's own touch screen. Even compared to a desktop with no musical controllers, the tablet's modality is much more focused and its articulatory distance much smaller. In alignment with all requirements, our design is focused on editing, which also contributes to its focused physical mode.

Again, participant's confirmed what we extracted from the literature, as they like to use their hands for creative work. In the context of composing the overall structure of a piece of music, Participant A said: "I noticed I can work very well with pen and paper. [...] I definitely need some doodling."

#9 Scalable Graphics (++++) At least iOS, for which we implemented the prototype, uses a resolution-independent screen metric called points. This, the quite fixed screen size and the necessity to make interactive elements large enough for touching nullify the issue that once arose on the desktop.

#10 Fluid Panning and Zooming (+++) On the desktop, setting the focus within the editing plane might require the user to shift his attention to the periphery of the plane and to set four little sliders with the mouse. On the tablet, it requires one swipe gesture directly performed on the content.

In contrast to GarageBand, zooming out to regain overview is especially fast with our prototype. But our implementation is not as fine-tuned.

#11 Relative Mixing Relative mixing is a bit removed from the other requirements of simplicity and it isn't as much affected by the platform.

Relative mixing would be a great complement to other features like custom abstractions and emergent mix groups, but it is in itself an isolated non-trivial problem that involves statistics, algorithm complexity and acoustics. In terms of Direct Manipulation [17], relative mixing together with touch interaction would make audio mixing far more semantically direct.

3.4.3 Requirements of Freedom

#12 Complementarity (+) Our design is definitely focused on the creative aspect, which is creating composed constructs of audio. However, while our model very much encourages it, the implementation doesn't yet provide file import or the use of other apps as sound sources.

#13 Interchange (+) With our design, the user builds all compositions from plain old WAV files, but the implementation doesn't provide true file import and export. Our model lends itself to saving material in an XML-based format but we didn't implement such functionality.

Also, it is typically impossible for mobile apps to directly communicate in real time. Michael Tyson solved this issue for audio apps on iOS. AudioBus enables us to implement full interchange functionality in the future. And that allows us, today, to focus our design on composing higher-level constructs from the output of different sources/apps.

#14 Self-sufficiency (++++) Our design is, of course, restricted in many ways. This helps the tool and its produced content to be self-sufficient, but it will be a challenge to keep this promise of self-sufficiency when other features are added.

Just like the literature suggested, our participants also had problems with data losses, format compatibility and accessing old material because music software and its output artifacts are generally not self-sufficient and don't promote interchange.

#15 Generality (++++) This follows directly from our model and isn't even much affected by the interface design. Abstractions are by definition generalizations and, through structural recursion, the user can build complex compositions from simple blocks of any kind of audio.

3.4.4 Requirements of Exploration

#16 Emergent Exploration (++++) To distinguish this requirement from others, we must ask if model and design enable and encourage the user to explore possibilities beyond the current editing context. We want to argue that this is very much the case because the model enables endless combinations of existing building blocks and the library invites the user to try them out.

Also, the appeal of custom abstractions is that anything that the user edits and creates is immediately available to him as a simple building block, so he is always tempted to test his creation in a larger context. This possibly endless recursion is unknown in existing sequencer software.

#17 Interface Exploration (++++) Can the interface be learned iteratively? Because the interface is a direct reflection of our generalizing model, it hasn't even many layers, which is good. The few layers that it has (subnodes, graph editor, detail pane) can be revealed and hidden as needed, which allows the user to explore them iteratively. The model and our emphasis on simplicity will help a great deal to keep this requirement satisfied when more features are added.

#18 Direct Exploration (++++) As mentioned above, interface and model are in congruence. There isn't much in the interface that isn't part of the model and vice versa. The user deals with a set of existing material and the currently edited performance.

3.4.5 Summing Up

The following tables roughly compares our design to GarageBand. First there are requirements #1 - #18, which cover cognition, simplicity, freedom and exploration.

#	Requirement	GarageBand	HAIL
1	Cognitive Styles	++	+ + +
2	Cognitive Dimensions	++	+ + +
3	Constraints	+ + +	+ + ++
4	Concentration	++	+ + ++
5	Accessibility	+ + +	+ + +
6	Mono-tasking	+ + +	+ + ++
7	Single Interaction Context	+ + ++	
8	Single Physical Mode	++	+ + +
9	Scalable Graphics	+ + ++	+ + ++
10	Fluid Panning and Zooming	+ + +	+ + +
11	Relative Mixing		
12	Complementarity	++	+
13	Interchange	++	+
14	Self-sufficiency	++	+ + ++
15	Generality	++	+ + ++
16	Emergent Exploration	++	+ + ++
17	Interface Exploration	+ + +	+ + ++
18	Direct Exploration	++	+ + ++

Then, there are requirements #19 - #36, which make up the abstraction category with its sub-categories of temporal-, voice-, processing- and reuse-and-versioning abstraction.

#	Requirement	GarageBand	HAIL
19	GCDCG-Sequencer		+ + +
20	Transient Voices		+ + +
21	Voice Groups		+ + +
22	Emergent Mix Groups		++
23	Juxtaposability	+	+ + +
24	Groove	+	+
25	Temporal Abstraction		+ + ++
26	Liveness		++
27	Direct Audio Processing		+
28	Scalable Audio Processing		+
29	Rich Audio Processing	+ + +	+
30	Visual Audio Processing	+	++
31	Nondestructive Audio Processing	++	+
32	Preparation	+	+ + ++
33	Global Reuse		+ + +
34	Versioning		++
35	Referencing		++
36	Variation		++

GarageBand has such a hard time mostly because it doesn't provide custom abstractions. Most of the requirements involve an applicability to all abstraction levels and that kind of generalization is absent in the domain model that GarageBand presents.

In summary, adding custom abstractions to a graphical control flow sequencer seems to promote vast improvements across many requirements of the "creator experience". This is especially compelling since the support of most requirements is positively affected, while we tackled only a few core requirements.

This was possible since we decided, from the beginning [19], not to focus on the interface itself but on its underlying domain model. We created an example of Domain-Driven Interaction Design.

3.5 From Design to User Experience

As mentioned earlier, the user study is supposed to answer three questions [19, 26]. Here we answer question number two: How did the participants experience our design? In the following , we'll give an account of our observations of these experiences without interpreting them.

The participant's reaction and feedback to our design was much more diverse and doesn't appear as generalizable as their working practices. Only a few of the following points of feedback were universally expressed by all participants. The other points are still worth mentioning because of how urgently they were expressed or how frequently they came up or because of how critical or specific they are.

3.5.1 Event Editor

Participants were uniformly a bit conflicted about panning and zooming in our implementation. They were, at first, delighted by the way the score matrix zooms direction-sensitive and provides detail and overview in a seamless and intuitive manner. Participant A said: "This is cool. I like that."

Participant E stated on panning/zooming in Logic, in contrast to our implementation: "That is also the annoying thing about Logic, that you first have to [scroll/zoom] in this direction then in that direction ... That would be really great, if you can do that intuitively with one gesture."

But after playing around with it for a few minutes, participants became annoyed by one or more of the following impediments:

1. Scrolling is not smooth due to performance issues. This caused a general sense of uncertainty or fragility and amplified the other concerns about panning/zooming.

2. Participants missed the bouncing effect they know from iOS scroll views. Sometimes, they were not sure whether they had hit the boundary of the event view. One said he didn't "feel the confines" of that content view.

3. Some participants proceeded scrolling with two fingers after they grabbed the view, instead of releasing one finger. The resulting two-finger pinches lead to unintended changes in zoom level, which was especially irritating since vertical- and horizontal zoom levels are independent.

4. Some complained they were frequently zooming in an unintended direction. For example, they wanted to zoom in on parts and accidentally also zoomed a little bit on voices, which they had to correct with the corresponding zoom out gesture.

Participants were quite pleased with how event editing generally works, but they missed a couple of features and pointed out some quirks of the implementation:

1. Some participants found it funny or strange that drawing gestures work from left to right but not from right to left. This is kind of a bug.

2. Some participants had difficulties understanding the meaning of events because the visualization of audio samples is inconsistent within a voice. They wondered if different events for one voice actually represented different audio files. In the project paper, we explained what implementation issue causes this graphical inconsistency.

3. Participants liked the musical quantization but asked how they could change its granularity and how to place events with more precision, especially when zoomed in on the time axis.

4. Participants tried to drag and drop events around and said they would want to do that even across voices.

5. Some participants wanted to change the length of events by ”pulling them longer”.

6. Some asked about cutting events like it can be done in DAWs.

The assessments revealed no significant controversy or conceptual problem with the interaction design around events but clearly highlighted the limits of our implementation.

3.5.2 Graphical Representation

The participants were aware of the principle difference between the model and its graphical representation. Participant D stated: ”I found that very interesting because, while you explained it [model and design] to me, I recognized subareas that I know from other programs, where I see you try to bring them together and that the main problem is to present that [our model] in an interface because that is simply super difficult, also to do that in a way so that you don't have to attach a manual to it that explains how it works, that's most difficult about that.”

Several participants commented on the graphical preview displayed on global events. They would prefer to see the ”contained” events instead of a waveform sum.

Some participants would want to have more graphical feedback on the internal temporal structure of the sub-parts of the current performance, particularly with regards to repetition and variation. The technique of color coding came up where participant B (professional musician) was very affirmative while participant D (communication designer) warned against it since it would only allow to distinguish about seven part types.

When navigating up or down, the relation between one and the next abstraction level was a bit unclear to some participants and they wished for smoothly animated transitions when navigating up and down in part- and voice hierarchy.

Because of our loop and variation mechanism, we had to graphically distinguish inherited and innate editable events. In our mockups, inherited events are significantly darker to signify they belong to a higher (more general) abstraction level. However, some users first perceived the active and innate events as being "selected", which is only true in the sense that those events are exposed and highlighted.

3.5.3 Understanding the Domain Model

All participants were able to understand our design in its entirety, including part and voice graph, loops and their variation as well as the graph editor. Here and there, participants were irritated for a moment or they needed a few minutes to fully grasp a particular aspect, but we are safe to say that our design posed no principle conceptual problem to them.

The "*of course, how else?*"-attitude with which four participants received our design and responded to it surprised us and actually made us worry that we haven't pushed the envelope hard enough.

A typical example of this is how our model dissolves the concepts of notes and instruments and simply merges them into "voices". Participants accepted that surprisingly quickly and offered different ideas of how to use this aspect of the model, once they got a feeling for it. One participant suggested grouping keys (atomic voices) into chords and then grouping those chords into a harmony instrument for simple chord-based song writing.

It took participant A a little more time and effort to let go of her ideas about how sequencer interfaces are supposed to work and look like. However, like with the other participants, after she opened up and began to grasp the much more general domain model of the prototype, she began to appreciate the elegance, flexibility and power that this approach offers.

Some of the most rewarding experiences of this whole endeavor of developing the Human-Audio Interaction Lab where those aha-moments that participants had when they let go and started to see the bigger picture. For instance, participant A erupted in the middle of our demonstration: "Ah! I understand the basic concept. Slowly."

Participants liked how freely our design lets the user combine material. Participant C stated: "I find it great, indeed, like that you can combine all this stuff that is otherwise [in conventional software] really separated. Of course, it requires adjusting your habits."

All participants really liked the custom abstractions manifested in part and voice graph. The notion of hierarchically grouping voices was particularly obvious to all participants. Participant A said: "Yeah, that makes sense. You also understand that quickly."

While all participants liked the idea of having all parts and voices they ever created stored in a library, some disliked how the prototype treats that library. They wished for a more prominent representation of the library as well as more functionality for browsing, searching and pre-listening.

Participant D saw the main advantage and application area of our model in collaborative settings, which is also interesting because the kind of collaboration that he suggested is roughly our ultimate vision behind the Human-Audio Interaction Lab, beyond the goal of this thesis at hand.

Participants D and E strongly suggested that we analyze Ableton Live because they saw parallels between our model and the non-linear interface of Live.

3.5.4 Parts and Temporal Structure

Participant B said on parts as explicit nodes in a graph: "That would be the form of the piece [of music], I find that very important because something [software] like that really doesn't exist ... the whole aspect of musical forms." And participant E said about the part graph: "Yeah, that's great. It's basically more like a hierarchy for ... well not a *library* but kind of a network of all that stuff [parts]."

But all participants also expressed some concern that the part graph may not benefit every type of music. Participant C said about the custom part abstractions: "It makes sense, indeed. Well, I'd say maybe not for every type of music." We asked for which type of music it would be appropriate. He said, "Well, everything that is to some extent relatively clearly arranged [overview], or a bit simpler structured. I think, if you have some extreme jazz piece where something different comes all the time [hardly anything repeats], you can, of course, maybe not combine [parts] as easily. But in itself it's [the part DAG], of course, definitely interesting."

Some participants were, at first, irritated by the fact that, on the highest level in the part library, unrelated songs would share a common timeline. Participant D said: "Ah, ok ok, because I thought that this is separated in the middle into 'project' and 'song'."

Participant A said: "It looks like one song, but it's actually two." It took her a moment to embrace the idea that there is no predetermined concept of "project" or "song" and that a whole body of work can be seen as one piece and, just as well, a single verse can be viewed as a mini library that holds some bars for future use.

The musical quantization in the event editor raised questions about the internal temporal structure of atomic parts. Participant D asked: "What is actually the smallest [shortest part] that you can enter? Are there, in the end, just single bars left or something when you enter the chorus itself ... because a pre-chorus would probably not make sense somehow, I think. I'd rather have bar 1 and bar 2 [and so on] at the end [at the leafs in the part DAG] [...] otherwise it would get too confusing I could imagine."

Participants were curious about the length of parts and how it is measured (beats, seconds). But it didn't stop there. They also inquired about the actual meaning of parts and what it means to reuse them. They even offered some opinions on that question:

1. A part defines a certain rhythmic structure.

2. A part only defines a number of beats.

3. A part should convey no temporal data at all. It should only serve to structure material semantically, i.e. to distinguish temporal entities

like verses, ideas, songs or projects from one another.

3.5.5 Global Events

Because events are visualized as a preview of the actual data they refer to, users understandably wished to jump right to the data by "opening" or "going into" the [composed] events, i.e. they would navigate more directly through the content.

Participant E stated: "I believe it makes more sense, when you don't open the next layer here [sub-node representations] but there [in the score matrix through its cells/events] [...] so that, when I click on that [an event], the level inside that opens. And not here, that doesn't make sense somehow. So that you say *ok, what is inside this clip [event]?* and then again *what is in this clip in this clip ...*"

From there, a profound discussion with participant E illuminated the whole issue of global events and hierarchy. He expressed the wish that custom abstractions should follow a very simple principle, although his ideas seem already to be realized by our model:

> "But I would actually find it greatest if you, for now, couldn't do stuff like MIDI or so, but the only thing you could do would be to drag really just one sample onto such a track [atomic voice], and the only action that would be available would be to play that sample by clicking [creating an event for the voice] [...] And because you could go deeper and deeper into it [create more abstraction layers] it [the musical structure] becomes complex."

The sketches he drew to explain his point, made his idea clearer and showed that he had grasped our intention and vision on a deeper level than the other participants.

Translated into our model and terminology, his suggestion amounts to the following: Atomic voices and their events would not necessarily refer to wave files (samples) but could actually refer to a performance. Thereby, complex hierarchical abstractions could be built, without even making use of voice groups and "global" events. This would bring back the benefits and elegance that we lost when abandoning "naive abstraction" [18].

Interestingly, he sketched events as triangles that look like play buttons. This may signify several intentions: The content that events refer to is not looped but played exactly once. The length of the content is significantly shorter than the part in which the event is played. The starting time is the most important property of the event. The event is really just a marker that plays (triggers) some content (performance or sample) rather than being a container of content itself.

3.5.6 Loops and Variation

All participants stressed the importance of variation within repetitions. That is, it should be possible for a repetition or loop to deviate a little from its original. Participant D stated: "Well, in my process, I try not do make everything the same. Maybe with beats, there it would totally make sense, but with a chord or fill-in, it would also be interesting to be able to say *But in the third verse, I'd like to have a little alternative.*"

The inheritance and variation mechanism that our global loops offer caused some concerns that it might be too complicated (participants A, C) or simply unnecessary because the part graph already allows for repetition (participant E). Participant E said:

> "Yeah loops are an interesting issue. [...] In Ableton, you basically only loop the clips. [explains how] I could imagine that one probably doesn't always want to loop everything. [...] There is so much [software] out there with loops and, actually, If you do it in that structure [our model], it's almost that you don't need loops anymore. I would possibly try the idea to get away from loops, honestly, because I also know several [musicians] who say *there is already so much loop-based music.* ... rather try to do something with the linear aspect again."

The concern about our looping mechanism is also understandable since our screen designs didn't yet communicate that aspect in great detail.

A discussion with participant D illuminated a crucial aspect about our suggested mechanism for loops and variation. First, we clarified how our design allows the user to create any kind of variation – not just progressively changing loops.

We discussed an example like the one in Figure (3.4). Two verses, *Verse A* and *Verse B* in a song may have a chorus between them. The user wants both to be of the "type" *Verse* but to also to have them expose some unique little variations of their base type.

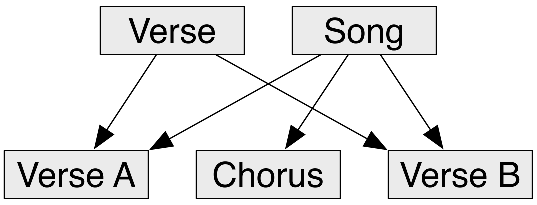


Figure 3.4: Example graph that uses a composed part to define a part type

The *Verse*-part that contains both verses and, thereby, defines their type or basic structure would not itself be part of the song because, in the song, the two verses have a chorus between them. Therefore, the *Verse*-part would not be contained in the temporal hierarchy of the song. It would not serve to define the temporal structure of the song but only to provide both verses with an exclusive common parent.

And that poses two design challenges: First, how does the user decide which parent of a verse he means when navigating upwards, the more general verse or the song? And second, how does the interface signify in the song that the verses have another common parent aside from the song, i.e. that they are of the same type?

3.5.7 Input and Integration with other Software

Some participants stressed the fact that recording their own samples with their own real instruments is very important to them, particularly in the early and creative phase of experimentation and composition. Participant B became very explicit at this point:

> "If you want to create a software like that, always come from the instrument [keep real instruments in mind]. I find that very important. What I find interesting now [about the idea of HAIL], you could tinker a lot with sounds, you know, you have different songs and ideas there and juggle with these parts, and that brings enormously many possibilities to be creative. That is great. But you must never forget the instrument with which you first bring in the sound [into the software]. Even if it's just a single guitar chord that you record. And then you put that in different parts. You must not loose sight of that or [of] how the musician thinks who plays his instrument."

Participant D asked: "But what is ultimately the goal? Do you want to create an interface with which you can completely produce something, or is it more like a preceding step." We answered that it is more like the preceding step. He said: "So, it's more something like a sketch book, right? [...] But then, I'd find a recording functionality definitely necessary, so that you can build little loop machines, so to speak."

Some participants even imagined our proposed concept as kind of a plugin in a conventional sequencer. With participant B we discussed a certain spectrum between using prebuilt sampler instruments that are based on samples of single notes and, on the other end, recording the whole song by yourself live. Many musicians work in between those extremes, recording snippets and chords, building custom sample instruments. In this context, Participant B offered an interesting idea:

> ”Yes, exactly, that’s true, I could imagine that very well here [in HAIL]. If I have a rhythm sample and can record that quickly and use it in other software, you know. Create something by yourself and then being able to access it again and again wouldn’t be bad, of course. Because here [in Sonar and his setup], it’s always program-specific, and you always have to load this again and that again. And the time [it takes] to get into [learn] a program is also enormous.”

What he implied was the idea that our system would be used to create, store and organize recordings and then provides these recordings across different software sequencers as an instrument plug-in. Of course, that would pose the question what would happen to our time dimension, since our design represents a sequencer in itself. But the idea is inspiring.

3.5.8 Limitations

Overall, confronting participants with our model, in screen mockups and implementation showed that the proposed concept is not as conceptually challenging to music software users as we expected. And the evaluation provided many valuable specific clues about how we should proceed development.

However, mere understanding of our design concept doesn’t imply that it inspires creativity or is even practical. Because our mockups and implementation didn’t allow for very much user testing, it was difficult to provoke many questions and comments and we got only little direct feedback on how the design would effect creative processes in practice.

For example, participants were able to learn what the graph editing operations mean in the prototype. However, we’re not sure how quickly they would be able to interpret and apply those operations in terms of musically meaningful actions.

Our prototype fosters and rewards a certain mindset or approach to the composition process that is quite untraditional. For instance, the idea to reuse time sections (parts) from other "songs" seemed rather foreign to most users. In the timeframe of the assessments, participants could hardly absorb the fact that time sections are to be used in a more general sense and that categories like "song" somewhat evaporate. This was especially the case with participants who have mainly experienced linear sequencers and rarely experimented with other types of music software like trackers, programming environments, live performance tools or Ableton Live.

The novel degrees of freedom that our design offers may spawn more creativity in amateurs or users that have preserved themselves a "beginner's mind". But further studies are needed to reveal how exactly our design approach would effect actual work flows and creative outcomes.

So, the limitations of our evaluation leave many open questions. On the other hand, participants were not only able to understand the design but also expressed great appreciation for its overall concept. Participant E who seemed to have grasped the vision behind that concept better than other participants commented on it in a follow up email: "I thought a few more times about your idea and have to say I like it more and more, that really has potential!"

Chapter 4

Discussion

4.1 On the Validity of our Evaluation

We could only test a prototype and not a released product. Moreover, the prototype partly consists of mockups. Therefore, some negative outcomes that the assessments highlighted are consequences of an incomplete implementation and tell us nothing about requirements or design. Examples are:

1. A lack of performance impedes panning/zooming.
2. Panning lacks the bouncing effect.
3. Events can't be drawn from right to left.
4. The waveform previews on events are inconsistent.

The fact that participants spent less than an hour with a prototype in an artificial setting instead of one week with a real product at home limits the validity and expressiveness of this kind of evaluation in even more ways:

1. The prototype lacks the amount and diversity of prebuilt content that it would contain as a real product.
2. The prototype focuses on some requirements and does not yet satisfy all of them. However, all the requirements work together to realize a certain concept. None of them is completely redundant.

3. Participants had hardly any time to experiment with the interface. This is critical since adapting to a novel domain model takes time.

4. Participants could not use the prototype in the context of their own material and process.

That said, we are pleased that our assessment sessions still tell us something about how its outcomes relate to requirements.

4.2 Creative Applications

Our hopes that participants would, by themselves, come up with interesting applications of our proposed system were a bit too optimistic. It was already difficult to bring this artistically inclined user group to reflect on their own working processes. It was even more difficult to elicit feedback on a mere concept for a tool in less than an hour.

We occasionally steered the conversation so that participants were able to notice, with as little assistance as possible, some of the more interesting ways of using our proposed system. This helped them to get a feeling for the bigger intention behind our design and that, in turn, provoked questions and moved the conversation forward.

So, in the assessment sessions, we established some creative applications of our concept together with the participants. As we mentioned earlier, participants had no problem following the working principle of the proposed interface concept.

Now, the way they responded to those advanced applications told us that they would have discovered those possibilities by themselves had they been able to play around with a fully functional implementation. For instance, Participant E arrived at the realization that chords could be triggered by single events: ”... so you would basically actually put C, E, G or so into one level [composed voice] and then go one [abstraction] level higher and play the whole thing. Yes, cool, I find that awesome.”

However, the only conclusion about creative applications that we can safely draw from our evaluation is that participants not only understood how the design enables these advanced use cases but they also appreciated those possibilities.

Many of the ideas that we developed with the participants relate to voice groups and the emulation or "creation of instruments". Some of these applications are:

1. Creating chord instruments
2. Creating instruments from their own samples
3. Creating instruments with custom scales
4. Creating instruments that mix tonal and atonal sounds
5. Creating instruments from vocal recordings

Another utilization of voices was to reference the same voice several times in order to create complex rhythms by layering different events for the same voice within the same part.

This is analog to repeating parts on the time-axis. Here, our approach to treat and present both dimensions equally paid out since participants used the analogy to deduct that the same can be done with voices. This repetition was enabled by – and is in line with Requirement #35 [19].

Of course, the voice-related use cases refer directly to Requirement #21. In particular, the custom recursive structuring that this requirement demands opens up all the creative options described above.

Most of these options are unavailable in traditional DAWs like GarageBand, an they're definitely not available at the level of simplicity and uniformity that is innate to our design. Our design conveys all these different use cases through only a minimal number of contexts. Thereby, it adheres firmly to the fundamental Requirement #7 [19].

As mentioned earlier, participants were a bit more reluctant to utilize and reuse parts (temporal abstractions). Still, they found some interesting use cases, which are only possible because our design satisfies Requirement #25:

1. Creating a setlist, playlist or album
2. Creating different variants of a song structure
3. Exploiting navigation and part order for live improvisation
4. Comparing unrelated material side by side

4.3 Refining Requirements and Design

Finally, we answer the third of Johnston's [26] three questions for the user study:

> "What are the relationships between the characteristics of the instruments [musical interfaces] and the musicians' experiences?"

We already have an account of the user experience from answering Johnston's second question, and that necessarily involves the relation between experience and design. What we're, now, interested in are the deeper characteristics of our design that are manifestations of underlying requirements.

In other words: How do the requirements determine user experience? The answer to that will allow us to enter the next development iteration where we refine those requirements and approach a re-design.

4.3.1 Mobility

First, we should explicate the tablet platform as a new requirement. We saw in Section (3.4) and many other places how well the tablet suits our purpose [17, 18, 19]. Now, part of refining the requirements is to make them address the conditions of mobility and book-sized touch screens.

Requirement 37 (Tablet) *HAIL should run on tablets and make optimal use of mobility, touch input and screen space.*

These refined requirements now apply more specifically to our design and prototype, which we called the "Human-Audio Interaction Lab", so we articulate them as <*HAIL should ...*> instead of <*The tool should ...*>

4.3.2 Event Editor

Zooming

Participants sometimes zoomed in unintended directions. This somehow fails Requirement #10 [19], which demands that navigating 2-dimensional content representations (like the score matrix) should be made simple and fast through fluid panning and zooming.

Our choice of the tablet platform is a result of many requirements and this is only one which tablets greatly support. However, none of the requirements assumes tablets being used, thus they don't specify how zooming and panning should work with a touch interface. No requirement is really missing but the ones that apply may be too non-specific.

We see two possible approaches that might solve the problem:

1. We might lock the zoom direction like GarageBand does. This way, users can only zoom in one direction with one gesture, the disadvantage being that they have to release all fingers and start a new gesture whenever they intent to change both zoom levels.

2. The problem of unintentional zooming arises particularly when both fingers are close to each other because, then, even small movements are relatively large compared to finger distance. It is likely possible to process touch events in a way that is more sensitive to the user's intent. For example, we might start such experiments by introducing a minimal distance between fingers for performing a zoom.

Editing

Some participants said they would want to manipulate events by moving, cutting, shortening and extending them. These operations are absent in our prototype. The requirements demand temporal- and voice abstractions but say nothing about the role of notes, events or score data.

Again, the requirements are quite general and don't prematurely fix design features that can only result from experimentation. Event editing is in line with the overall imperative of simplicity (Requirements #3 - #11) but specific requirements for editing events with a touch interface are missing.

We already implicitly applied certain criteria when we employed touch gestures in our design. Before we add new editing gestures, we should make those underlying criteria explicit:

Requirement 38 (Proximity) *HAIL should present entities spatially closer together the tighter they are semantically related [6].*

Requirement 39 (Economy) *HAIL should expect easy gestures for frequent operations and complicated gestures for rare operations.*

Requirement 40 (Directness) *HAIL should employ touch gestures in a way that maximises articulatory- and semantic directness [17].*

These three requirements may seem quite general, common sense or even banal. But together, they tightly restrict our design of further editing operations. For example, picking a tool (like in Logic Pro) or picking an action to perform after tapping an event (like in GarageBand) appear as two sub-optimal solutions in the light of the above requirements.

Now we require HAIL to offer the set of editing operations that users apparently need the most.

Requirement 41 (Event Editing) *HAIL should offer ways to edit events, including operations to create, delete, move, shorten, extend and cut.*

And this is how we would map the editing operations to gestures in a way that balances all the above requirements:

Operation	Importance	Gesture
Delete	Very High	Tap
Create	Very High	Tap or draw
Move	High	Drag and drop (not the ending if visible)
Shorten	Medium	Drag and drop ending (if visible)
Extend	Medium	Drag and drop ending (if visible)
Cut	Low	Long press, swipe outside (up or down) at intended position
Connect	Very Low	Long press, swipe outside (left or right) to neighbour event

To make length changes very direct, we'd have to give up the interval marking (see the project [18]) and, with that, the possibility to delete several events through one gesture.

4.3.3 Graph Editor

Sometimes, participants got confused by the abrupt switches when navigating up or down in our keynote prototype. They tended to lose the context of what happened during those steps.

We observed that navigating custom abstractions lacks a bit of continuity and context. Our design failed to completely satisfy the exploration requirements, particularly #17 and #18 [19]. This is not a question of the task itself or the underlying model but of their presentation through the interface. We might improve our design with the following changes:

1. Navigating up to a super-node should be visualised through an animation that continuously collapses the currently visible sub-nodes into one while expanding the newly visible sub-nodes.

2. Navigating down to a sub-node should be visualised through an animation that continuously expands the selected sub-node while collapsing all the others.

3. The user should be able to enter a sub-voice *and* a sub-part with one gesture by double-tapping in the event-editor, analog to entering either one of both sub-node types.

4. The user should be able to navigate up to the super-part *and* the super-voice with one gesture by a two-finger tap in the event-editor, analog to moving up to either one of both super-node types.

Improvements (1) and (2) would provide more continuity and context to the user and help satisfying Requirement #17. Improvements (3) and (4) would increase the directness of navigation and better satisfy Requirement #18.

Together, those four improvements would give rise to a powerful variant of semantic zooming. The user could smoothly zoom in on a cell of the score matrix, revealing more details and editing options. But still, there would be discrete abstraction levels, all using the same visual language, so the user would always know exactly what details he can effect at the current abstraction level and how he would do so.

4.3.4 Parts

Requirement #25 demanded custom temporal abstractions. This requirement is the reason that parts exist in the prototype. Participants loved the idea of having explicit parts that can be moved around like columns to edit the overall temporal structure of whatever they may compose.

However, as we described before, the exact nature of parts is not as fixed, and our study inspired us to reflect on aspects like the internal structure of atomic parts and the relation between parts and score (event-) data. Our design sketches out one approach, but the assessment sessions hinted that there may be others and that the design should be more specific about those aspects.

To effectively explore the solution space, we should explicate our goals. We need to clarify the nature of parts beyond the characterization that Requirement #25 provides. While Requirement #25 should be made more specific, requirements for quantization and (composed) score data have yet to be written. In the following, we'll explain the ramifications of re-thinking parts.

4.3.5 Atomic Parts and Quantization

It didn't surprise us that users wanted to position events with more precision than the arbitrary quantization of our implementation allows. In contrast to the voice graph, event editing requires a fine grained differentiation within atomic parts. The user must be able to position events with millisecond precision within atomic parts.

Also, we envisioned that the quantization would ultimately have to adapt to zoom level. Because touch interaction requires input elements to have a minimal size on screen, the interface should always apply the most detailed quantization raster with which the smallest quantization cell is still big enough on screen at the current zoom level. Therefore, the user could only position events as precisely as the current zoom level allows.

Zoom level would determine the detail level of data visualization. It would also set the detail level for data manipulation. In terms of Direct Manipulation, output- and input language would be close together [17], which is a really good thing.

Quantization involves a discrete subdivision of time intervals (parts) into shorter time intervals. It is very obvious that the quantization has to be musically meaningful and realize true *musical quantization*. We summarize this as a new requirement:

Requirement 42 (Quantization) *HAIL should employ a musically meaningful quantization raster that adapts to zoom level.*

The question we have to answer is how we recursively subdivide atomic parts in order to provide this meaningful zoom-dependent quantization.

In the seminar paper [17], we mentioned that, "typically, time spans are recursively partitioned into a prime number of sections of equal length." And that applies at all temporal scales, particularly around bars, beats and fractions of beats, where rhythmic patterns unfold. We see two ways to implement these insights:

1. For atomic parts, the user may choose a custom chain of prime numbers, i.e. a rhythmic structure.
2. The interface chooses the most likely rhythmic structure for atomic parts. Without considering additional context, this would mean to always divide into two intervals of equal length.

Ultimately, the interface will have to provide a combination of both approaches. A reasonable first step would be to implement only variant (2). This would suffice to demonstrate the adaptive quantization while covering a wide range of application scenarios.

4.3.6 Parts and Scores

One reason why it is a bit hard to grasp what role score objects (groups of events) could play in our model is how tightly related event lists are to the temporal structure of parts. We want to use that structure for the quantization of events, not just with respect to visualization and editing but possibly also in the way event lists are actually stored in a data structure. Now that we have explained how a quantization tree might look like (previous section), we can approach the notion of a *Score* as a distinct entity in our model.

Participants liked the idea of building complex rhythms by looping different event lists of different musical length within the same part. We would greatly support this if each event list could be edited in relation to its own time structure. Therefore, each event list would have a quantization tree associated with it. To make such an association, we finally need to acknowledge those event lists as explicit *Score* entities.

Let's just say an *Atomic Score* has a list of non-overlapping events which is ordered by the start time of events. It also has a length, measured in the same units as the length of events.

We still might ask, whether the quantization tree is an innate quality of the score or whether it is just a way the interface communicates that score. Does the model actually use the quantization tree as a data structure to store the timing of events or does quantization just help the user to edit events?

Both options have their advantages. The latter allows us to present the same score in the context of different quantization trees, but the relation between that structure and events might be lost. Event timing could not anymore be derived from discrete numbers and be more of a floating point value.

4.3.7 Musical Length

Whatever the relation between quantization tree and score might be, atomic parts would not anymore be associated with a quantization tree. So what data do atomic parts convey at all? At least, they must somehow help to determine the actual duration of parts and performances in seconds.

We intentionally left the question open in what unit the timing (start time and length) of events and the length of scores is measured because we need to keep scores general enough so that different scores can be combined within the same part. And while it might suffice to leave all the timing of scores in relative measures, parts must, at some point, introduce an absolute measure.

We are confronted with a trade-off between reusability and simplicity.

Within that trade-off lie multiple possible approaches to delineate the quality of length and duration of parts. We'll briefly describe one approach that appears as a reasonable compromise. However, this aspect demands further experimentation.

We suggest, the design should somehow adapt the way practically all music software and music notation systems handle length and duration: Beats serve as the currency for musical length, and a tempo in beats per minute (bpm) determines actual duration.

Every atomic part would have a length in beats. And every part may or may not define a tempo. If a part does not define a tempo, it inherits the tempo from the next ancestor that defines one. The part library as the ultimate ancestor of all parts would always define a tempo and that would act as a default value. Also, new atomic parts could have a default length, so the user wouldn't need to worry about any of that unless he wants to change something.

We're aware that, in some cases, we'd need a way to decide which of the possibly multiple ancestors determines the tempo of a part. We may introduce the notion of an "active parent" but we won't go into such details here.

4.3.8 Composed Scores

An atomic score, as we introduced it over the previous sections, basically holds a rhythmic pattern that can sit in a cell of a score matrix. It relates to just one voice. However, most often, the events for several different voices together make up musically meaningful scores like melodies, arpeggios, chords, and even drum beats as we saw in Figure (2.4).

Even conventional DAWs incorporate some notion of such composed score objects in the form of regions on MIDI tracks, where they can be moved, copied, referenced, looped and even be used across different instruments (voices). So the purpose of identifying such objects is to apply and reuse them in different contexts.

In our model and design, a composed score would comprise one or more or all the atomic scores in the column of a score matrix. It would be applicable on all levels of abstraction and it should be reusable and available to all projects and working contexts.

Since we have no explicit requirements for composed scores yet, let's summarize the above in a new requirement:

Requirement 43 (Composed Scores) *HAIL should offer the user score objects that can hold tonal scores for composed voices. He can apply them at any abstraction level and reuse them within different voice groups.*

To make this work, we need to introduce a level of indirection. Music software and score notation use tonal keys and drum types for that, and we can very well adapt this convention.

However, the general nature of our design requires the user to assigned keys or drum types to the sub-voices in a voice group before he can apply some existing composed score to that group. That way, the system can map the atomic scores in the composed score onto the sub-voices.

Another issue would be whether and how the user could be enabled to apply composed scores in the way of MIDI effects, or more general, how he might concatenate composed scores.

For example, he has a composed voice that represents a piano instrument. And there is a composed score that plays a melody on that piano. Now he might append another composed score to that which would transform every note that is played into an arpeggio on that note.

He could not just create arpeggios but also chords, rhythms, melodies, delays and any kind of pattern. Because the transformation is always relative to the incoming (base) key, concatenation could perform all kinds of score transformation, even such simple ones as transposition.

So, composed scores pose a huge challenge to us as designers and offer powerful tools to the user. May be the challenge can't be met perfectly as it requires decisions on trade-offs, but we'll surely tackle it in the future.

4.3.9 Global Events

Our design includes the possibility to edit global events, that relate voice- and part groups. Without such a concept nothing can be edited at higher abstraction levels and we were hardly surprised when participants missed that functionality in our implementation. Strictly speaking, we need global events to just really satisfy Requirements #21 and #25.

But the assessments made it clear that our solution to those Requirements is not the only possible solution, much less the optimal one. In particular, the way events relate to (composed) voices didn't always match the participant's intuition.

Suppose a composed voice has event lists for its sub-voices but does *not* loop these event lists but, instead, has a non-repeating score over the whole length of the part. Global events that trigger the whole group wouldn't make much sense. On that higher level, the user would rather draw one event that reaches from the very beginning to the end of the part to make sure the score inside the group gets played.

In that respect, we fully understand why participant E wanted those global events to reference performances instead of (composed) voices.

For once, it is easier to imagine the event referencing a clip that is significantly shorter than the part in which it is referenced. The samples of atomic voices already have that property. They have another simplifying property: The sample file that an atomic voice references applies across all parts, while the internal score of a composed voice depends on the part. And that is also one reason why we we abandoned the idea of voices referencing performances when we experimented during the project.

However, participant E inspired us to bring this idea back for atomic voices. Atomic voices should allow to not only reference sample files but also to reference whole performances as long as that doesn't introduce any cycle. Of course, the voice would still be atomic and, although its sound may be complex, that sound (the referenced performance) would still apply across all parts.

High-level editing clearly requires us to extent our solution and experiment with some more variants.

4.3.10 Loops and Variation

We didn't even expect the subject of variation (Requirement #36) to come up at any point. But our looping mechanism opened up a wide range of applications to the user and our participants were able to see how loops can be utilized to realize types and inheritance.

However, the discussion with participant D also showed that this way of using the system somehow requires the user to know what he is doing. The *Verse* part in Figure (3.4) references two verse variants. The *Verse* itself, together with whatever voices the song uses, would make a performance that "plays" both verse variants. The fact that the *Verse* is not used in this way and rather stores a type is not explicit in the model but rather implicit and in the user's mind and intention. Participant D alluded to how difficult that would actually make collaboration. A different user might not know whether a part is intended as a type or actually as some piece of music.

There are some other issues with this kind of variation. It only works with loops, because they can repeat over each sub-part which has the effect that the whole loop is "passed on" to each sub-part.

Types would have to be visualized so that the user would a) immediately identify a variant as a mix of reference and original and b) immediately see which material is of the same type.

Another issue is that, in the current design, the user is responsible to match the lengths of event lists so that they are in sync with each other in a meaningful way.

The question we'll have to tackle in the future is whether and how our design should support the use case of variation through inheritance more explicitly.

4.4 Lessons for CSTs

Here, we address the aesthetics/theory part of Johnston's methodology [26]. First, we ask what general hypotheses about CST design we can derive from our work.

4.4.1 CST Design

While the requirements of simplicity, freedom and exploration translate fairly easy to other domains, the interesting aspects of our work surely are custom abstractions and their re-use across the boundaries of conventional definitions. What made this particularly difficult for us to actualise is the 2-dimensional nature of musical compositions, as they are time-dependent *and* deeply layered. In addition, music is not intrinsically visual but rather abstract, requiring us to find an effective visualisation.

Other creative products don't exhibit all these three intricacies in combination. For example, video is time-dependent but not deeply layered and also visual. Text and program code is linear but not deeply layered and has a natural visual representation. Graphics are not time-dependent, may be deeply layered and are visual. And so on.

Compared with music composition, it would be quite trivial to provide custom abstractions and an integrated library of material in these other domains. And indeed, we find glimpses of these qualities in other tools.

The most advanced domain in this regard seems to be software development, where custom abstractions and extensive re-use are innate to the process. We also find custom abstractions in graphics design. For example, OmniGraffle lets the user group objects together. He can treat a group like any other object and also have groups within groups.

On the other hand, extensive re-use as Requirements #32 and #33 demand is practically non-existent beyond the domain of software development. Considering how essential this is for creative work [19], future CST research and design should tackle the question of how to evolve data organisation from separate project files to one integrated library.

4.4.2 CST Research

Now, let's review the the methodology that we applied. We experienced an interesting tension between our desire to explore unknown territory and our desire to do so through the scientific method. The literature, including Johnston's work, emphasises the special role of creativity support tools [19], and the thesis at hand magnified that aspect.

With novel CST interfaces, the question is not so much how they apply to certain use cases. The question is how they inspire users to come up with interesting use cases. The interface is supposed to offer a new perspective on a known subject and, thereby, allow its users to objectify and challenge their unconscious belief systems, promoting open-mindedness and creativity.

So, the need to innovate the tool is ingrained in the objective to make it suitable for creative work. We can't encourage creativity without being creative. In a publication on evaluating CSTs, Hewett et al. [21] stated:

> "If one optimizes current practices (that is, *what* people do), one can expect only incremental improvements in computer-based support for the creative process. On the other hand, understanding *why* people do what they do can lead to new insights into the forms computational support can take."

To understand *why* creative practitioners do what they do, researchers and designers must immerse themselves deep into the respective domain and that, by its very nature, limits somewhat the possibility of deriving general guidelines from their domain-specific work.

Evaluation must be tailored to these circumstances, where evaluating incremental improvements of established interfaces types may be possible but is insufficient.

So what about evaluating leaps that redefine the very DNA of an interface? We suggest that CST researchers should be skeptical of short term studies because real innovation of CSTs involves more than new graphical mappings for known concepts and is often not as flashy. And this also highlights the limited expressiveness of our own evaluation.

Despite all efforts to break down the research and design of CSTs through methodologies, principles and guide lines, most authors acknowledge that CSTs require more of us: While placing one foot on the solid ground of science, we must be confident enough to step, with the other, into a place of uncertainty, where ideation and intuition must be applied. The goal to support creativity requires not just ergonomics but actual *interaction design*. And that, as Donald Norman [36] suggests, is unfortunately not yet an engineering discipline but still an art form.

In this context, art and engineering don't really present a trade-off, because one side cannot easily be traded for the other. Instead, one has to find a balance (the sweet spot) between them.

This leads to an important question for the science of CST design: Should researchers tweak existing interfaces in spectacular ways to investigate specific boundaries or should they develop new comprehensive interface concepts that have a chance of actually progressing real products? The context of academic research seems to lend itself more to the former approach, and that makes us even more thankful for the opportunity to also explore the latter approach in this master thesis.

Chapter 5

Conclusion

5.1 Summary

We've come along way. Back in the seminar paper [19], we listed 9 potential contributions that we aspired to deliver over the course of seminar, project and thesis. By now, we've tackled all of them, and it is time to reflect.

Seminar The challenge in the seminar [19] was to find an entry point that lets us make use of domain knowledge and intuitive insights but also adheres to a structured methodology. This amounted to a long but all the more substantial orientation process. We contributed an extensive literature survey and introduction to the domain as well as a rather complete list of requirements for novel music sequencers.

Project The challenge in the project [18] was to narrow the requirements down to the most essential ones and find a model that enables an interface to satisfy these core requirements while being compatible with all the other requirements. Another challenge was to document this process as a linear progression to make it understandable. We contributed a domain model, a clean and extendable code basis which reflects that domain model and a rudimentary implementation of an interface prototype.

Thesis The challenge of this thesis was to demonstrate a big idea based on one design concept exemplified by an incomplete implementation and, then, to evaluate potential innovation of creativity-support. We contributed an example concept with the help of mockups, an account of user experience with that partially implemented prototype, an analysis of how user experience relates to the design and its criteria, refined criteria, approaches to develop the design further and a discussion of the consequences that all of this has for CSTs.

The idea behind our *Human-Audio Interaction Lab* is too comprehensive to be realized within one master subject. It may have been beneficial for this thesis to narrow its objective down even more strictly.

On the other hand, we gave an introduction to the vast field of HCI research that grows around computer music, offered a broad differentiated view on the rich application domain of music composition, identified a distinct unfulfilled need and, most of all, developed a clear vision for novel music sequencer interfaces and for creativity-support tools in general.

5.2 Outlook

Before even thinking about any further conceptualization, future work faces some specific implementation challenges that result from our refinements of requirements and design.

A caching mechanism is necessary to improve performance and visualization. Panning/zooming has to be fine-tuned. For event editing, we have to fix bugs and add features. Navigating through the graphs has to be enriched with new gestures and animations. And much more.

Then there are also challenges regarding modeling and design, for instance the library, score- and event groups, tempo and duration, and more.

The third challenge is to satisfy more of all the requirements on which we did not focus, for example regarding relative mixing, mix groups, audio processing and so forth.

Another challenge will be to adjust the evaluation approach and to re-evaluate HAIL at a more advanced state. It is also possible that a broader perspective that contrasts fundamentally different solutions would create better results than endlessly iterating one concept [10].

Finally, we might look even farer out into the future and think about how HAIL could integrate with other apps through AudioBus, data formats and storage systems, how it might benefit from a combination with a desktop version and, ultimately, how it might fit into the context of sharing and collaboration.

Bibliography

[1] J. T. Bernstein, L. Dong, J. Missig, and M. Hauenstein. Prototyping: Fake it till you make it. Talk at the *WWDC 2014*, *Apple Developer Videos for Enterprise App Developers*, `https://developer.apple.com/videos/enterprise/#56`, 2014.

[2] D. Bolchini, D. Pulido, and A. Faiola. Feature: ”paper in screen” prototyping: an agile technique to anticipate the mobile experience. *interactions*, 16(4):29–33, 2009.

[3] C. Brower. A cognitive theory of musical meaning. *Journal of Music Theory*, 44(2):323 – 379, 2000.

[4] E. A. Carroll, C. Latulipe, R. Fung, and M. Terry. Creativity factor evaluation: towards a standardized survey metric for creativity support. In *Proceedings of the seventh ACM conference on Creativity and cognition*, C&#38;C ’09, pages 127–136, New York, NY, USA, 2009. ACM.

[5] J. Carter, B. Eaglestone, N. Ford, and P. Holdridge. An analysis of interviews with composers from a cognitive styles perspective. In *International Computer Music Conference*, pages 391–394, Ann Arbor, MI, 2009. International Computer Music Association, MPublishing, University of Michigan Library.

[6] D. Chang and K. V. Nesbitt. Developing gestalt-based design guidelines for multi-sensory displays. In *Proceedings of the 2005 NICTA-HCSNet Multimodal User Interaction Workshop - Volume 57*, MMUI ’05, pages

9–16, Darlinghurst, Australia, Australia, 2006. Australian Computer Society, Inc.

[7] E. Cherry and C. Latulipe. Quantifying the creativity support of digital tools through the creativity support index. *ACM Trans. Comput.-Hum. Interact.*, 21(4):21:1–21:25, June 2014.

[8] D. Cronin. Feature: Into the groove: lessons from the desktop music revolution. *Interactions*, 15(3):72–78, 2008.

[9] N. Donin and J. Theureau. Theoretical and methodological issues related to long term creative cognition: the case of musical composition. *Cogn. Technol. Work*, 9(4):233–251, 2007.

[10] S. Dow. How prototyping practices affect design results. *interactions*, 18(3):54–59, 2011.

[11] M. Duignan. *Computer mediated music production: A study of abstraction and activity*. PhD thesis, Victoria University of Wellington, 2008.

[12] M. Duignan, J. Noble, and R. Biddle. A taxonomy of sequencer user-interfaces. In *International Computer Music Conference*. Ann Arbor, MI: MPublishing, University of Michigan Library, 2005.

[13] M. Duignan, J. Noble, and R. Biddle. Abstraction and activity in computer-mediated music production. *Computer Music Journal*, 34(4):22–33, 2010.

[14] B. Eaglestone, N. Ford, P. Holdridge, J. Carter, and C. Upton. Cognitive styles and computer-based creativity support systems: Two linked studies of electro-acoustic music composers. In *Computer Music Modeling and Retrieval. Sense of Sounds*, volume 4969 of *Lecture Notes in Computer Science*, pages 74–97. Springer Berlin Heidelberg, 2008.

[15] Evans. *Domain-Driven Design: Tackling Complexity In the Heart of Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[16] S. Fels and M. Lyons. Interaction and music technology. In P. Campos, N. Graham, J. Jorge, N. Nunes, P. Palanque, and M. Winckler, editors, *Human-Computer Interaction – INTERACT 2011*, volume 6949 of *Lecture Notes in Computer Science*, pages 691–692. Springer Berlin Heidelberg, 2011.

[17] S. Fichtner. Direct manipulation. Seminar: *Theories and Models in HCI*, `http://hailbringer.com/writings/dm_seminar_paper.pdf`, 2013.

[18] S. Fichtner. Hail: Developing the human-audio interaction lab. Master-Project, `http://hailbringer.com/writings/developing_the_human_audio_interaction_lab.pdf`, 2014.

[19] S. Fichtner. What music composition interfaces require. Master-Seminar, `http://hailbringer.com/writings/what_music_composition_interfaces_require.pdf`, 2014.

[20] M. Gurevich. Editor's notes. *Computer Music Journal*, 34(4):4–5, 2014/02/18 2010.

[21] T. T. Hewett, M. Czerwinski, M. Terry, J. Nunamaker, L. Candy, B. Kules, and E. Sylvan. Creativity support tool evaluation methods and metrics. In B. Shneiderman, G. Fischer, M. Czerwinski, B. Myers, and M. Resnick, editors, *NSF Workshop Report on Creativity Support Tools*, pages 10 – 24. National Science Foundation, Washington, DC, 2005.

[22] S. Holland, K. Wilkie, P. Mulholland, and A. Seago. Music interaction: Understanding music and human-computer interaction. In *Music and Human-Computer Interaction*, Springer Series on Cultural Computing, pages 1–28. Springer London, 2013.

[23] Y. Hong and T.-J. Nam. A method to get rich feedbacks from users in an interview for design concept decision. In *CHI '10 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '10, pages 3907–3912, New York, NY, USA, 2010. ACM.

[24] A. Jandausch. Conceptual metaphor theory and the conceptualization of music. In *International Conference of Students of Systematic Musicology*, volume 5, 2012.

[25] M. Johnson and S. Larson. *Architectural Metaphors in Music Discourse and Music Analysis*, volume 50 of *Yearbook of Comparative and General Literature: Mutability. Architecture, Music and the Chicago School*, pages 141 – 154. Bloomington: University of Indiana Press, 2002.

[26] A. Johnston. Beyond evaluation: Linking practice and theory in new musical interface design. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, pages 280–283, 2011.

[27] J. Kjeldskov and M. B. Skov. Was it worth the hassle?: Ten years of mobile hci research discussions on lab and field evaluations. In *Proceedings of the 16th International Conference on Human-computer Interaction with Mobile Devices &#38; Services*, MobileHCI '14, pages 43–52, New York, NY, USA, 2014. ACM.

[28] P. Mannonen, M. Aikala, H. Koskinen, and P. Savioja. Uncovering the user experience with critical experience interviews. In *Proceedings of the 26th Australian Computer-Human Interaction Conference on Designing Futures: The Future of Design*, OzCHI '14, pages 452–455, New York, NY, USA, 2014. ACM.

[29] C. Nash. *Supporting Virtuosity and Flow in Computer Music.* PhD thesis, University of Cambridge, 2011.

[30] C. Nash and A. Blackwell. Tracking virtuosity and flow in computer music. In *International Computer Music Conference*, pages 575–582, Ann

Arbor, MI, 2011. International Computer Music Association, MPublishing, University of Michigan Library.

[31] C. Nash and A. Blackwell. Liveness and flow in notation use. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, Ann Arbor, MI, 2012. University of Michigan.

[32] J. Nielsen. *Usability Engineering.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[33] J. Nielsen. Usability inspection methods. In *Conference Companion on Human Factors in Computing Systems*, CHI '95, pages 377–378, New York, NY, USA, 1995. ACM.

[34] J. Nielsen. Why you only need to test with 5 users. *Nielsen Norman Group*, `http://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/`, 2000.

[35] J. Nielsen, T. Clemmensen, and C. Yssing. Getting access to what goes on in people's heads?: Reflections on the think-aloud technique. In *Proceedings of the Second Nordic Conference on Human-computer Interaction*, NordiCHI '02, pages 101–110, New York, NY, USA, 2002. ACM.

[36] D. A. Norman. Interaction design is still an art form.: Ergonomics is real engineering. *interactions*, 13(1):45–60, Jan. 2006.

[37] J. Preece, Y. Rogers, and H. Sharp. *Interaction Design.* John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 2002.

[38] E. Raita. User interviews revisited: Identifying user positions and system interpretations. In *Proceedings of the 7th Nordic Conference on Human-Computer Interaction: Making Sense Through Design*, NordiCHI '12, pages 675–682, New York, NY, USA, 2012. ACM.

[39] K. Wilkie, S. Holland, and P. Mulholland. What can the language of musicians tell us about music interaction design? *Computer Music Journal*, 34(4):34–48, 2010.

[40] S. Wiltschnig, B. Onarheim, B. T. Christensen, P. Dalsgaard, H. Korsgaard, L. J. Ball, J. Chan, A. Houssian, and A.-M. Hebert. Integrating laboratory paradigms and ethnographic field studies for advancing analyses of creative processes. In *Procedings of the Second Conference on Creativity and Innovation in Design*, DESIRE '11, pages 365–366, New York, NY, USA, 2011. ACM.

[41] T. Zhao and S. McDonald. Keep talking: An analysis of participant utterances gathered using two concurrent think-aloud methods. In *Proceedings of the 6th Nordic Conference on Human-Computer Interaction: Extending Boundaries*, NordiCHI '10, pages 581–590, New York, NY, USA, 2010. ACM.

Appendix

The data that we list here in the appendix to represent the assessment sessions is only a selection of observations that we made while re-listening to our audio recordings.

The quotes and comments are partly un-edited, meaning they represent our notes and are not part of the polished main body of this work. What we already used in the main text doesn't appear in this appendix.

Participant Profiles

Participant A

Age	27
Sex	Female
Profession	Interaction Designer
Instruments	1. Electric guitar for 11 years 2. Singing in a choir for 4 years 3. Keyboard for 3 years 4. Songwriting (incl. singing) for 2 years
Activities and Experience	1. Songwriting and recording as a hobby 2. A bit of live experience

Software and platforms	1. Logic X Pro on Mac OS to record, edit, compose and mix songs 2. GarageBand on Mac OS to record, edit, compose and mix songs 3. MuseScore on Mac OS to compose and print scores 4. Voice Memos on iPhone to capture raw ideas
Type of music	singer-songwriter, pop songs, natural sounds, mainly recordings, some virtual instruments
Process Overview	1. Create lyrics and melody, possibly with guitar, possibly capture in Voice Memos 2. Add chords using guitar 3. Flesh out song structure on guitar, finalise on paper or in a word processor, possibly use MuseScore 4. Possibly transcribe score in MuseScore for further composition and printing scores for other musicians 5. Record song in GarageBand or Logic 6. Edit and mix in GarageBand or Logic

Participant B

Age	47
Sex	Male
Profession	Musician (composer, performer, teacher)
Instruments	1. Trombone for 37 years 2. Piano for 40 years 3. (Base) guitars and other instruments on demand

Activities and Experience	This is a selection: 1. Leading, playing, composing and background singing in a jazz and latin band for 13 years 2. Playing in a Big Band for 10 years 3. Leading a trombone choir for 8 years 4. Playing in a Latin and Salsa Band for 6 years 5. Playing in another band for 6 years 6. Leading a Big Band for 4 years 7. Leading and playing in another latin jazz band 8. Composing and arranging scores for his bands 9. Recording two albums with one band 10. Coaching young bands 11. Teaching instruments 12. Composing scores for a score publisher
Software and platforms	1. On Windows: Finale for score writing, editing, arranging and print 2. On Windows: Sonar for inspiration, recording, composing, arranging 3. On the iPad Mini: different Apps for capturing ideas and composing
Type of music	Jazz & Latin, Songwriter Rock/Pop
Process Overview	1. Start new ideas withs instruments: Guitar, piano ... or with lyrics ?. Possibly write idea down on paper (text and harmonies) 2. Write note for note through a combination of MIDI-piano and keyboard into Finale (actual composing and arranging) 3. Export Finale files as MIDI files and import MIDI files into Sonar 4. When the large scale structure (arrangement) is fleshed out, record guitar, base, vocals into Sonar

Participant C

Age	25
Sex	Male
Profession	1. Multidisciplinary artist 2. University student of Communication Design
Instruments	1. Acoustic and electric guitar for 19 years 2. Base guitar for 7 years
Activities and Experience	1. Song writing and recording for 8 years 2. Home recording solo artist, but compositions aim at a band instrumentation 3. Experience with playing and improvising in a band 4. A bit of experience with collaborative song writing
Software and platforms	1. Logic Pro on a Macbook at home for recording instruments 2. GarageBand on the Macbook for capturing ideas on the go (without external hardware audio interface) 3. Smart phone for practicing singing 4. Years ago: Cubase on a Windows computer
Type of music	Psychedelic rock (instrumental, no singing vocals)

Process Overview	1. Create and develop new ideas through playing guitar 2. Quite early in the process: record ideas, part by part, in Logic Pro or Garageband. Some recorded pieces are very short samples. 3. Build loops from recorded pieces by manually copying them 4. Playing and recording alternate and inspire each other, developing the song and its overall structure iteratively. 5. Virtual instruments are added, but the guitars remain the heart of most songs. 6. Utilizes no other externalization (like hand writing) than Logic Pro and Garageband to capture and develop ideas

Participant D

Age	24
Sex	Male
Profession	Designer and film maker
Instruments	1. Violin for 11 years 2. Electric and acoustic guitars for 8 years 3. Piano for several years 4. Cello for a few years 5. Singing for at least a few years
Activities and Experience	1. Playing violin in a jazz band 2. Playing guitar in a metal core band 3. Working for a music label

Software and platforms	1. Logic Pro on a Macbook 2. GarageBand on the Macbook 3. PhotoBooth on the Macbook (to record videos of playing ideas on the guitar) 4. Currently learning to use Ableton Live (on the Macbook)
Type of music	Singer-Songwriter (solo) with recorded and virtual instruments, artful movie scores
Process Overview	1. Creates and develops ideas through guitar- or piano playing 2. Constantly records with Logic Pro. Recording and composing alternate quickly and co-evolve (iterative approach, rapid cycles). 3. Sketches out ideas with virtual instruments in Logic Pro, before recording the real instrument

Participant E

Age	16
Sex	Male
Profession	Musician (jazz pianist, composer)
Instruments	1. Piano and Keyboard for 12 years 2. Violin vor more than 6 years 3. Trumpet for 6 years 4. Base guitar for 4 months

Activities and Experiences	1. Regularly plays live with several bands 2. Is band leader in at least one of those bands 3. Composes material for his main band 4. Has always been composing music and always used software to assist that 5. Composed an orchestral soundtrack for a short film 6. Composed a piece for a bigband that also has been performed live
Software and platforms	1. Ableton Live on a Macbook 2. Finale on the Macbook for score writing 3. MainStage on the Macbook for live performance 4. Native Instruments Maschine 5. Reaper (back in the early days)
Type of music	Jazz and neo soul (live), modern classical (scores), experimental and electronic (personal projects in Ableton Live)
Process	1. Process depends heavily on context and goal 2. Typically creates and develops compositions by piano playing 3. In some cases, he writes scores, note by note, in Finale 3. Creates and develops electronic and experimental pieces completely in Ableton Live and through a MIDI keyboard. 4. Also develops songs collaboratively with members of his band

General Processes

Collecting, Accessing and Reusing Material

Participant B

- there was already enough material for the album when the band came together: "I've had lots of ideas, the ideas emerge at the computer, still today, and they add up, you know?"
- "But then there are also songs where you have snippets and you tinker with them, over a year, and still it doesn't please you, and then it just lies there."
- On managing and saving ideas: "A long time ago, I also worked without a computer, but then it was really hard. So I often sat at the piano and wrote down the chords. And when I was interrupted [...] the idea was gone the next day. [...] Now with the computer, it's great. You can capture [remember] all ideas you come up with."
- Our question "Do you know from the project names you have assigned what ideas they contain?" His answer: "Always. Yes."
- On reliability: "There is a new [version of] Sonar that I would like to buy, or I could update it, but I still have Windows XP there. And everything is running just fine. And I don't dare to install the new Windows 7 or 8 now, because I don't know what won't work anymore."
- compatibility, self-sufficiency: About 500 score files created over 20 years would have to be re-edited when opened in the newest version of Finale!
- On reuse of existing material: Our question: "Does it sometimes occur that you open one project file and then think *Oh there was this other idea [project file] that would well fit together with this one*?" His answer: "That I mix them together? I think that hasn't happened yet. [...]

That probably happens before [using Sonar], during composing itself [with guitar]."

- ideas have such a unique and clear character that they don't mix together, so they're not reused with one another, although working on one can take months. All its elements remain in one project file. Combining across different keys doesn't occur because the key is part of what delineates the character of the song.
- Uses commercial prebuilt content in form of MIDI loops.
- On distribution of prebuilt content: "That is coming more and more, I also worked with musicians who create samples and then sell them to labels. Now, there are many musicians who work like this."

Participant C

- He rarely re-uses older existing ideas as building blocks for new projects. If he re-uses an old idea, he doesn't combine the projects but re-records the old idea into the new project.
- Uses muted tracks to dump material for later use within the same project.
- "The cubase projects are gone, because they were on the old computer. [...] I don't mind, it's old stuff anyhow, so..."
- On accessing project files from the operating system: "However, I always give them totally shitty names, so that I don't know at all what is actually what. [...] But I also wouldn't know how to name them better."
- He uses additional sample libraries and instruments (beyond what comes with Logic Pro) but: "I always have problems with disk space on my computer." Our question: "Like you're using one violin, but need to

have the whole orchestra package installed that requires several gigabytes ..." His answer: "Yes, exactly."

- On identifying old unfinished projects when going back to them: "Now and then, I listen to them, well, not to all of them but I just look what the last ten things were that I created. [...] it doesn't happen that frequently that I resort to that [old] stuff. [...] I'm just not so disciplined and constantly simply start new things that just come right now. And the things that are very important I keep in my head, after all."

Participant D

- uses the project file in Logic Pro to dump or park material: "I use the first five tracks or so for snippets and ideas ..."
- further on the experimental mode of Ableton live: "Well, it just is this preparation for the linear mode, for collecting ideas. It's just some pot, like a box of legos."
- On whether he uses Ableton's non-linear mode to collect unrelated ideas or whether he creates a new project file for each new idea: "I open a new project because I can, from the mood alone, roughly assign [identify, separate] it."
- On re-using parts across these ideas or project files: "No, I rarely do that. Somehow it seems to be relatively fixed what belongs where."
- On whether he recognizes an idea/mood/vision from the name he has assigned to the project file: "If it was a serious project, yes. And if not, it's just like this [example project on screen], then there are just these snippets [...] which is somehow also quite awesome. [...] About music making I find the aspect of *development* [progress, change] quite intriguing."

- "Actually, it's also great to be able to let things [snippets/ideas] lie there [not work on them for longer]."
- regularly (at every significant break), he saves the project file under a new name (versions number) to keep (conserve) that state for later comparison or for going back to some earlier state.
- While none of the participants use to merge projects or ideas, Participant D does sometimes actually split a project when the ideas stored within the project file start to diverge. Like other participants, he sees project files as little material libraries...

  On whether he deliberately puts tracks into some order: "When I'm [still] trying things out, I indeed sort them according to what [other tracks] they relate to. [...] And slowly, I develop those little clusters. But mostly, when I notice *these just belong together*, I open a new project [to extract the cluster and have it as its own project] [...] or the rest is just left there. [...] It's mostly really like a pot where I throw stuff in and then, progressively, select things from it."

Participant E

- On whether Ableton Live project files can be exchanged between musicians: "I tried it once and it didn't work. It was just a very simple file. That is not done that well with Ableton, I think, no that is not that easy. With the plugins anyway, but with samples it's also an issue [of] where they are stored. What also has happened to me is that I emptied my trash and then several projects didn't work anymore because some samples were still stored there. [...] I had thrown some sample away and it probably was still in... yeah, I probably have hundred open projects on which I don't work. I rarely finish things."
- On navigating and accessing his material in the Mac OS Finder: "Actually, I don't use that at all. I do all of that out of Ableton [the navigator inside Ableton Live]."

- he can not identify (recall) the content of project files from the file names he has assigned: "No it is completely chaotic. [...] However, a lot of people do it like that. You're just too lazy to think of an appropriate name and type in some arbitrary shit." But he recognizes every piece once he opens the file.

- On whether he comes back to older projects: "Yes, sometimes. Sometimes, for example, I look into old things, completely old things, just for fun, and possibly find something of value. But most of the time it's actually like *half complete, well, on to the next!*"

- he rarely brings older existing ideas into new ideas/projects.

- On whether Ableton allows to preview or pre-listen to project files from within its own file navigator: "No, you also can't pre-listen. But that would indeed be cool, yes."

Inspiration and Handling New Ideas

Participant B

- On originality: "Sometimes I wake up at night and go downstairs [into the basement studio] and make sure that I get it done. I'll show you one song were I got up at 02:00 at night and worked non-stop until 11:00, and then the song was done. It was in my head and had to get out."

- On handling new ideas "And then, you have to bring this song or this underlying thought into a certain shape, and mostly I first do that with score notation software. There, you can better think about how the parts lead towards following parts and so on."

- On why he merges all audio on a track into one file: "I think that's not bad to avoid clicks [glitches] [...] My computer is neither the fastest nor the newest and sometimes things happen. Like, during recording suddenly everything slows down [...] because the CPU is overburdened."

- uses prebuilt content like loops and samples to get inspiration and new ideas

- on having new ideas: "That really only emerges in my head, not at at the computer. The computer is more for bringing it into a form. But then [when starting to capture it in software] it's already complete in my head. A lot also emerges on a sheet of paper, when I sit to write it down, I write the chords."

Participant C

- On why he writes nothing down on paper: "What I play on the guitar I can remember pretty well."

- Determines song tempo in the beginning by recording one of the initial guitar licks and then matching the tempo to it.

Participant D

- On sketching new ideas with virtual instruments: "That is the useful thing about it. Before I record anything with the violin... I mean it sounds shitty first [the virtual instrument] but you can plug in a violin sound and see how it generally fits there."

- uses Photo Booth on his Macbook to record videos of himself playing new ideas on the guitar, partly to also capture fingering (the left hand)

- particularly with movie scores, he starts the composition process with an atmosphere or mood in mind

Work Process

Participant A

- about the prospect of composing in Logic: "That would also distract me too much, I have to say. I'm someone who is quickly overstrained by screens. My brain can't process that [composing] as well and I'm not very creative anymore when I sit in front of it [the screen]".
- further on software complexity: "MuseScore is extremely complicated to handle, I think. Not particularly intuitive."
- struggles to clearly identify, compare and combine parts: "It would be easier if it were clear. With my new one [song], I came up with the chorus, and I needed a long time to write the verses for that, and wasn't completely happy with that. [...] Either I complete it, or it lies around somewhere as an idea."

Participant B

- On the goal of composing and drawing the finish line: "It has to communicate something. It has to become [overall] a coherent entity. Whether it sounds perfect [...] [is not important] at some point you are ready to say *now it's done.*"
- On using synthetic instruments: "I do a lot at the computer. And back then, when I wasn't as proficient with instruments, I tried a lot with virtual instruments and to optimize their sound so that it would express some human feeling. [...] On the second album, the percussion is completely made of samples. [...] I did that all here at the keyboard. [...] This is where the software – with *what* you work – really makes a difference. I had the best experiences with Sonar."
- On using a click for recording (and timing) a song: "Good musicians practice – and are able to play – with a click. This is very very important."

- On recording with a click vs. the creative dynamic of recording songs by performing them as a live band: "Nowadays, with a computer, you can do that alone, you can perfectly flesh out the idea that you have and then hand it over to the musicians. [...] That [the click] is never a limitation, it just requires you to be more explicit about what you want to do. It results in greater certainty."
- Our question: "What would you say is more important in a band that works together, the musicality or the training?" His answer: "The musicality. [...] Score notation is always only a means to an end."
- Solo composition precedes collaborative recording/performance: "Now, I have composed five new songs that have vocals, and I want to bring them onto a CD in the near future but, firstly, I don't yet have the right musicians for that and, secondly, I have to improve the songs some more."
- On working with MIDI loops in Sonar: "The wonderful thing is you don't have to quantize it. It's all ready to use. You drop it in there and it automatically plays through [the drawn time range]."
- It's better to start a new project than endlessly perfecting an existing one
- Used virtual instruments on album production
- the click is does not a restrict creativity, mostly because tempo changes can be programmed in the sequencer
- Using software to create and notate music greatly enhances the communication of ideas and musical content between musicians, especially if those musicians are amateurs and not classically trained.
- turns MIDI tracks int audio tracks by recording them, in order to save CPU performance.

- would like to temporarily hide or trim down graphical representation in favor of CPU power for real time recording.
- uses project files to communicate ideas with his collaborators
- He doesn't create sounds or instruments. Those are always just applied in the context of a concrete composition. "I'm not a sound tinkerer. [...] I don't want to produce but bring the composition onto a stage [to real musicians]."
- Doesn't create mix groups as that is more on the production side.
- "There is often very very much specific information, also here [in Sonar] that you can click on [and] that you, now in this moment, don't need at all, you know. And that is much too much information that rather distracts you from what you're actually doing, and where it takes a lot of time to click in there [interact]."
- "What has helped me is that here [...] I assign these little instruments [instrument pictures to tracks]. Then you immediately see *There is a guitar, there is a drum set [...]*"
- Criticizes the separation of mixer and tracks into different windows in Sonar.
- creates loops (like with the drums) through manual copying (!)

Participant C

- On why Garageband is sufficient in most cases: "It's [the song is] always more or less like a sketch, and I don't have the ambition to get a fully produced song out of it. [...] because I also don't know much about mixing and things like that."
- His songwriting involves more than chords and is not strictly chord-based. It is more guitar-oriented and also roots more in licks than in riffs.

- On developing songs to completion from within the sequencer: "Unfortunately, it's very often the case that I don't finish these things [songs], but I just start and tinker with that piece and think *Well, at some time, when I have use for it, I'll come back to it. But it's nothing complete.*"
- has problems with panning/zooming in Logic Pro because he works a lot with Video software where that works differently...
- on editing notes (events) for virtual instruments: "Most of the time I do, indeed, click [create/draw] them there, note by note."
- Manually moves events around and away from the quantization raster to introduce some kind of randomness, i.e. natural timing.
- uses the macbook keyboard to record only a rhythm on a single key, then edits the tonal aspect (key) of each recorded note manually.
- records rhythms and single chords through a MIDI keyboard
- manually creates and edits complex drum patterns and fills in the pianoroll editor, without actual drum playing experience
- The structures in temporal and voice-dimensions develop in interdependence, inspiring each other iteratively. As the song grows longer and its temporal structure delineates, the number of tracks grows with it. The example song was about 5.5 minutes long and had about 30 tracks, most of them audio tracks.
- On collaborative writing: "In my experience, when I perform with other people, totally different ideas come up than when I perform only by myself. When I record stuff by myself I'm always quite restricted, I don't know why, but when I write some chord progression and record it and record overdubs on that, the result is rarely as good as when I play to material of others. I think one reason is that, as soon as I record something solo, there is this thought in my head *this is being recorded and it has to fit*, and then I'm not as free to just play around."

- He uses only a very minimal set of features of the software, to the point of not being aware of very basic features and having workflows that are unnecessarily cumbersome. But, on the other hand, he stays deeply connected to his inspiration, creative intention and creative result. (admirable)
- He does not change the order of tracks. Their order is determined by when and where he creates them. This doesn't seem to inhibit "juxtaposability" for him, or he doesn't require it very much (editing musically related tracks side by side, in relation to each other).
- On not naming tracks or regions: "In my process it's always like that: When I'm really at it, that all has to go very fast. I don't know it any other way. As long as the flow is there, it has to go fast, and then I can't name that stuff or put it in order."

Participant D

- Learns and plays by listening. "I hate score notation, always hated it."
- On his example song: "Particularly with guitar, I make music by just playing. That is, I don't think about music theory at all because I also don't know much about it."
- On his iterative process: "Actually, I constantly record and build on what I just did. Either I work on it or replace something, that is an enduring process. [...] Even though it doesn't actually all happen simultaneously [composing and recording], I jam with myself, so to speak. What I also frequently do is, like if I only have one riff or so, I loop it and jam over it and see what comes out of that."
- "At some point the decision has to come when to really record it [the idea/song/instrument]."
- He sketches ideas and melodies with virtual instruments.

- On whether he also uses GarageBand: "Yes, but that is... well, currently ... I mean Logic and GarageBand are basically exactly the same thing. [...] I use GarageBand whenever I don't have Logic available."

- "I'm a big proponent of the attitude that these things are tools. So, with which software you work, with mac or windows is so irrelevant. [...] What I like about the Mac is just the interface."

- "What I just find great, for example now that I'm learning Ableton [Live] [...] there is this classic linear [sequencer interface] and then also a kind of experimental mode, and I really embrace that at the moment. That is the way I actually make music, that is: playing, then recording something in between, modifying and looping, and for that Ableton [Live] is just fucking awesome. [...] Of course, when I ultimately record music, I need that linear mode, because on a pure logical level it's just made for that. It's just a timeline, exactly like with film. But until I am at this state [to record in that linear mode] [...] I mean, the tracks that I have then are so short but I have 20 of them. And Ableton [Live] simply does exactly that for me, because I'm actually not yet in that linear mode but in that *I'll just loop everything I have and look how it might fit together.*" Our question: "So you don't yet have to decide on the order." His answer: "Exactly, exactly. Because that [order] is supposed to just evolve out of that [different loops/snippets]." That is a classic articulation of the requirement for delayed linearization [19].

- On his decision not to go into music professionally but to keep it as a hobby: "Because as soon as you *have* to do it ... no, this little soundtrack niche for little movies is totally sufficient for me. And I also really celebrate it that I can make music whenever I'm really inspired and then I'm also more motivated than if I'd have to do it."

- uses no mix groups and has, overall, no experience with – or interest in – the subtleties of mixing and producing.

Participant E

- On MainStage: ”It can’t do that much but it’s just mega reliable, so that it doesn’t crash or something...”
- ”Actually, I don’t use Logic at all, but by now I also have that.”
- Used an intricate setup for his orchestral movie soundtrack: he routed MIDI data from Finale into MainStage, i.e. he used MainStage as a standalone orchestra. Then, by using Soundflower, he directed the Audio output of MainStage into Ableton Live to record it.
- describes the restrictions and complications that the composer faces when composing a piece that is supposed to be played by a real ensemble, like a bigband or an orchestra. for example, not every melody can necessarily be played on a specific instrument.
- On zooming and scrolling in Finale: ”Normally, I go into normal zoom [level] and then I scroll [with the mouse, Finale does not utilize the touchpad]. [...] in the beginning, that was totally odd, particularly when you come from Ableton [Live]. In Ableton, you just click into the location and press play. An here [in Finale], you really have to specify the bar number. [...] But you get used to it. Most of all, it [Finale] makes you work more accurately, while in Ableton, you can just tinker with it until it fits, but when you write a score [like in Finale] you got to be a little precise.”
- On the nature of his countless unfinished ideas/projects: ”Often, they’re really just loops. Mostly, they’re not fleshed out, well, the details are not done and they’re not mixed at all. Well, by now I can mix a little but it’s not that fun. [...] while I’m still working [developing the piece], I already mix more than other people.”
- more on the role of mixing: he doesn’t use specific monitor boxes for that. this rough mixing is part of developing the character of the piece

and weighting voices relative to each other. It follows a creative not a technical intent.

- creates and edits much score data (notes, events, MIDI) manually with the mouse. Then he often moves those notes around to make their timing less clean and more natural.

- On the timing raster of Ableton Live: "However, that's a thing that I don't like at all. When you play jazz and things like that, there are not just triplets but also [german: Quintolen und Septolen] ... You can't do that here."

- when he has ideas on the go, he captures them with his Macbook: "The least amount of the time I sit down here [in his basement studio]. I have it [the Macbook] upstairs and also when I'm on the way [outside, anywhere], I always have it with me, even during rehearsals. If this should break, a lot of stuff [compositions] would be gone. Really everything is on that."

Handling Temporal Structure

Participant A

- Participant A on working out a temporal structure: "Of course [I do that]. It [the composition] has to be logical [i.e. meaningful, coherent] after all."

Participant B

- On the temporal structure of an example song: "From A to part J. This is how many parts occur in that song."

- On custom markers on the timeline in Sonar: "That is also important when you make recordings, because I record the base guitar myself and [when I have to re-record] then I only need to go there, like I go to the pre-chorus, you know, and play [record] it again if I didn't like it."

- Our question: "Suppose you would like to edit the structure of the song, like inserting a verse there, how would you do that [in Sonar]?" Answer:"Yes, exactly, I often do that. Then I cut it through [audio material on tracks] and there [on the timeline] I can set markers, like for one bar."
- he uses markers to orient himself in the time dimension, manually cuts all tracks, marks all events and moves them through drag and drop to edit the large-scale temporal structure
- On why this form of editing is not as cumbersome as it sounds: "Before I record all tracks I have, like, only the guitar, drum and base tracks. That are three tracks, that provides clear overview. And there you can say *Ah let's insert another verse here*"
- identifies verses as "parts", also uses other specific terms for temporal abstractions: chorus, pre-chorus, fill, break, bridge, transition, variation
- very common: looping the chorus a spontaneous number of repeats

Participant C

- on the structure of his example song: "There are roughly two different parts, and they [an organ instrument] only play in the chorus like part."
- Our question: "How do you maintain an overview on the structure of the song, like where which parts are?" His answer: "Good question... basically, I just remember what I have recorded. I always have this part A with which it starts, and then here comes the chorus at some point, after this gap where the break comes [...] that occurs once at the beginning, then the part A comes again, and then it goes, well, where

is the transition... [plays the song at that position] ... well, I don't really set markers [on the timeline] where I am."

- On how the temporal structure of the song evolved: "I really have to say, that emerged during recording and tinkering with it. I didn't somehow create a concept or anything like that [...] I just started playing around. I create one part, then I create another, then I say *OK, now I want to proceed with the first part.*"

- Edits the temporal song structure (arrangement) by manually selecting, moving, copying audio- and MIDI regions.

- On temporal overview and transient voices: "When I edit at some position, I listen to it. And then, when I search anything, like I want to change a specific bit, I listen to it [the song at an estimated position] and see *Ok, it has to be there* [the correct position]."

Participant D

- on how the temporal structure (order of parts) evolves during the process: "It's not predetermined, particularly with melodies, I create them still in this jam stadium and record it as kind of a placeholder and look what comes out of that. and then, out of that, I pick parts that I like and re-record these specific parts. and then, most of the time, I have the melody."

- on (temporal) structural differences between singer-songwriter and soundtrack projects: "That parts are repeated is more the case with singer-songwriter stuff, but there I'm not at the computer most of the time when I'm doing singer-songwriter projects. Of course, they are also much much simpler, they're mostly just chord progressions where you can say: *verse, verse, chorus, bridge... and then done.* [...] well, that [temporal structure] then emerges from the working process. Yeah, maybe that is the main difference. When I make music with the computer, I mostly have visual references. The goal is simply different. There [when composing soundtracks], I don't just make music to make music, but I make music purposefully to underscore or amplify a mood."

Assessment of our Design

Domain Model and DAGs

Participant B

- Immediately understands the concept of parts (verse and chorus) inheriting the loop of their super-part (song): "Well, I like that, yeah. I find it provides overview. When you can structure it like that. [...] That would be great. You see exactly were you are in the song."

- on having the same sub-part (like a chorus) referenced multiple times from its super-part (song): "That would, of course, be much faster to handle. Because, here [in Sonar], you have to click, copy, move, cut again and so on."

- On re-use: "Yeah, nowadays, there is more this thing, like you have a song, and now there is the single version and there is the groove version and disco version. Very many [musicians] do that nowadays. And it would be great if you could automate [assist] that, of course. [...] Yeah, exactly, that you can access single parts and say *On those parts, I wanna hear a different beat for once. Or I wanna have that in a higher key.* If you could juggle like that, that wouldn't be bad."

Participant C

- "In itself, it all makes sense, indeed, particularly with different tempos. If you have one and the same tempo, it's easy to piece everything together. But if there are different tempos [in different parts] ... [...] I don't know how it would be [in our design], but you can stretch MIDI instruments and all [freely change tempo], but with audio recordings you couldn't necessarily do that [in our design], or could you?"

- On the inherited beats in verse and chorus: "That's a bit complicated."
- On the possibility of removing those keys from an "instrument" (composed voice) that have no events: "Of course, to simplify that makes sense, so that you only see what is really there."
- "Of course, it would be cool if I could try it out in more detail, but I understand the overall concept. And it also makes sense. Like I said, maybe not for everything."

Participant D

- On how the DAGs for parts and voices might effect the way of composing: "I can imagine this to be pretty good in a collaborative setting. Because, since you can always go one [abstraction] level higher, the single part loses its significance. [...] I could imagine this could become interesting when multiple [composers] participate in it. So that you have this sound pool, this library, but so that it is set up in a way that [people] can exchange [material] between each other. [...] At least in the design industry, everybody has, indeed, his pride and wants to bring in his own stuff. And here [in our model], it is more democratic because the higher you go in these [abstraction] levels the less you see the details, and maybe it's easier this way to mix and combine [material from different people]."
- Asks how a part like a verse that is repeated throughout a song can then be varied a little with every repeat. "Could you, if you have a verse, and if it [the defined score/loop] is just one chord [like 4 beats], could you basically attach variations and alternatives to it? So that,

if you press longer on it you're offered verses A to D because you just want that the second verse ends a little differently."

"It also is very genre-dependent. Because if you want to make a 4/4 electro, it's super because that is not that complex."

- he recommends: "Look into Ableton [Live]. That also works with building blocks. [...] The approach is similar [to ours]. Of course, it aims at something different, it is made to work live."

Participant E

- On the voice graph: "Well, you should definitely look into Ableton, I think. Because, it's not exactly the same but it does contain ideas like groups. This [our voice graph] is something that is not in there [in Ableton Live], that everything is so extremely connected. This here [the part graph] is quite close to what Ableton has as an idea. And that here [the voice graph] is definitely a great idea, yes."

- a major confusion unfolded because, in our mockups, the example parts in the library were named 'project' and 'song'. He read these names as categories of the model rather than as custom names assigned by the user. the names in the example should be changed into something more authentic.

- On the inheritance that our hierarchical loop definitions offers: "Yes, that is indeed great. [...] Yeah, that does make sense."

- Brings up a sequencer software (name unknown) that supposedly is the only software that lets the user apply effects not just to voices but to sections on the timeline (parts) just as well and easily (without time-dependent programming parameters of the effect or voice). He really likes that feature. strongly suggests that we treat parts in analogy to voices when it come to applying effects.

- further suggests that effects and audio processing should be applicable to a specific combination of part and voice (which we called *Performance* in our model). strongly suggests that this should not be the current performance, because that would require the user to enter and leave a sub-part and a sub-voice when he wants to apply an effect to what we call a cell in the score matrix. The user should rather maintain overview over all sub-nodes and select and edit a single cell ...

- ”For now, I wouldn’t do that much with effects and so [audio processing] [...] but first, when you start [to implement HAIL], only [implement] these samples [as the basis for custom abstractions], because I believe you can already do a lot with these.”

Graphical Representation

Participant B

- On the content preview displayed on events for the composed drum voice: ”you could also display that in other formats, for example as a score. [...] Because that would be interesting [...] If you display base and snare separately, you’d see exactly what the base plays and what the snare plays.”

- On timeline and cursor: ”If that would also be marked through colors ... or a little time marker would run there [directly on the part].”

- On the cursor running over a loop repeatedly before it enters the next part: ”That wouldn’t be necessary at all. It would suffice if the time cursor would simply run through [the sub-part] down there [directly on the subpart representation]. And up there [in the event editor] it would only display *what* it plays. That would totally be enough for a musician. I even like that more than if the cursor would constantly move over that [the loop in the event editor].”

- "Of course, if you don't use only one loop there [...] but when you [...] wanna have a fill after the third bar, that should be possible by tapping it, it should be able to display that. [...] I'd work a lot with color there [to signify variation and repetition over time]" he means: color coding the variation of a loop that the sub-parts apply. "If the colors are similar [like applied in sonar] [...] you immediately see *That must be the same.*"
- Criticizes that, in Sonar, variations like fills within audio material is not visualized which confines orientation in the time dimension: "Here I only see one thing [actually] made of different loops. If there would be different colors, then I could immediately recognize *Ah, it plays different things there.* And [on every third bar], if I had that darker, I could recognize [the difference]. And when it's light again [...], there it plays the same thing again. It would be clear on first sight and I wouldn't need to click there and listen to it first."
- explicitly states that one editable layer is enough but more feedback on the content of sub-nodes is needed: "If you need more layers [edit details] you tap on it [the sub-nodes], and separate that single part [of interest]."

Participant E

- He suggests that entering (opening/selecting) a part should be initiated through a zoom gesture, i.e. by literally zooming into the part. He really emphasizes that the transitions should be smooth and feel like zooming in and out of details rather than like opening and closing files.
- on the graphical content preview displayed on events: "Graphically, I wouldn't make this a waveform but really as a clip [rectangle] and inside the clip [if it triggers a performance] display the clips [preview the events of that performance]. Then you also see when you are inside the deepest layer [where the voice refers to samples]."

Timeline and Quantization

Participant B

- On the time line: ”You always need a certain raster to understand that and [communicate that] with other people.”

Participant D

- On the timeline of the part library: ”So for the lib there’ll also be an overall timeline?”

Participant E

- likes the idea of being able to partition atomic parts by just assigning some list of small prime numbers, like (5, 2, 3) which would partition the part into 5 sub-sections and partition each of these 5 sub-sections into 2 sub-sections and so on...

Global Events

Participant B

- On editing events for different levels of voice abstractions, from notes to big loops: ”Yeah that is, indeed, understandable.”

Participant C

- On applying (reusing) arpeggios or some form of composed events: ”I would do that manually. [...] If’d hat to do that automatically [like with MIDI effects], until I’d have configured exactly what it should play automatically, what kind of arpeggio ... [it would take too long]” Our question: ”Would you make use of it if you could create it manually but then apply it to the whole track/instrument?” His answer: ”No, I don’t think so because I always would... well I can’t really know that now.”

Participant E

- On what events for composed voices should mean: ”From practice I'd say ... wait, maybe I should reopen Ableton again. [...] There [in Ableton Live] it is basically the same problem. [explains MIDI regions and how they are edited in Ableton Live]” From there, a deep discussion unfolded about custom abstractions and what the nature of global events should naturally be.

- expresses the strong intent to ”go into [global] events” and would like to understand [as far as we understood him] the events themselves as references to containers/pieces rather than as references to voices and voice groups

Recording, Input and Integration with other Software

Participant B

- ”I could very well imagine that as an additional program to that [Sonar] in the program itself [as a plugin in Sonar] within a big interface, where you'd open such an editor [HAIL] and you could arrange the single parts like that [...] and simply insert them again there [reuse/repeat]. I recorded that part already, drums are in there and all, now I want it to repeat. Or I want to insert another verse in the song, a fourth one or so.”

Participant D

- ”An optional connection to analog interfaces I'd also find important. That I have the option to connect my keyboard or my MIDI-controller or so. That would be great because to do everything with touch becomes annoying at some point. Things that tend towards [drawing and zooming] gestures are cool but, ultimately, this [hardware knobs and sliders] is indeed more awesome.”

- "And that's what I really find great with Ableton [Live] that this mapping works super easily [mapping any hardware control onto any control in the interface]."

Participant E

- Asks whether our proposed system could be an extension [plugin] for an existing sequencer
- suggests we use *Max for Live* [an audio programming environment] for implementing HAIL, based on what he understood of the part library and because Max for Live integrates well with Ableton Live.