Realtime collaborative annotation of music scores with Dezrann

Ling Ma, Mathieu Giraud, Emmanuel Leguy

Realtime collaborative annotation of music scores with Dezrann

Ling Ma, Mathieu Giraud, and Emmanuel Leguy

Univ. Lille, CNRS, Centrale Lille, UMR 9189 – CRIStAL
Centre de Recherche en Informatique Signal et Automatique de Lille
F-59000 Lille, France
dezrann@algomus.fr

Abstract. Music annotation is an important step in several activities on music transcribed in common music notation. We propose a protocol to annotate collaboratively such scores in real time. Based on a paradigm with commutative operations, this protocol guarantees consistency beween distributed editions while providing a fluid user experience, even behind possible network lags. It is being implemented into Dezrann, a web platform for sharing music analysis. We report efficiency and scalability tests on the current implementation, including usage by up to 100 simulated clients.

Keywords: Score annotation, music analysis, collaborative editing, distributed algorithms, operational transformation, operation commutation

1 Introduction

Music annotation is an important step in several activities on music transcribed in common music notation – music analysis, teaching, performance preparation or even composing. As we noted in [5], annotation can be modeled as putting *labels* on the score at various positions, possibly with a duration, and sometimes drawing relations between these labels. Researchers in musicology or in computer musicology often engage with scores through reading, annotating, and analyzing, and they discuss other people's analyses. In talks or lectures on music analysis, history or composition, one frequently needs to *get back to the score*, jumping to different sections, sometimes comparing them, in different places, commenting some elements of a score or of several scores. Most of the time, *paper scores* are very efficient for that. In such lectures, it is now common to see students or people with laptops or tablets. But they often are not as efficient as when they work with annotated paper scores laid on the desk. More generally, many people like to talk about music and to describe what they hear, as demonstrated by presence of music on social networks.

Software for music annotation. Can software help people from different backgrounds efficiently annotate or discuss music? Several software enable to render, and sometimes to annotate, scores on the web[1] [7, 13, 14], or to analyze music [3,

[1] jellynote.com, notezilla.io

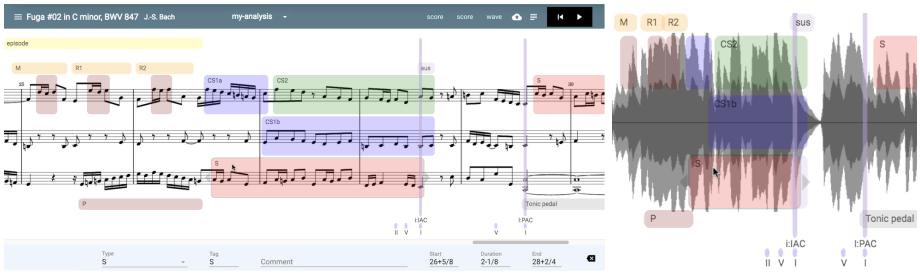

Fig. 1. Annotation with Dezrann (`www.dezrann.net`) on a fugue in C minor by J.-S. Bach, showing both on-stave (subjects, counter-subjects, patterns) and off-stave (harmonic sequence, degrees, cadence, pedal) labels. The music/label synchronization is kept across both a generated score (left) and a waveform from a recording (right).

8, 10]. We introduced the open-source platform Dezrann [6] favoring simplicity over the number of features, focusing on easy usage for people with limited programming background as well as efficiency for people needing to encode music annotations on large corpora. Music is presented either on continuous staves or on waveforms. The user may create, edit and move *labels* on the score – either on a staff or on spaces above or under the score (Figure 1).

Realtime collaborative annotation. Collaborative editing is made possible by fast and ubiquitous network connexions. Initially proposed to edit texts, it is also used in graphics or other domains. Collaborative music applications on the web began to emerge a few years ago, with for example collaborative score edition[2], collaborative patching [12], or collaborative performance [1, 11].

Behind the scene, the challenge of any collaborative editor is to maintain *consistency* between multiple editors separated by the network, and to handle conflicts possibly arising from simultaneous editions. *Lock* mechanisms are reliable, but they limit real-time interaction. More recent models enable actual simultaneous editions by several clients. Operational Transformation (OT) is based on transforming the operations to keep a converging state for each client [4], keeping casual links between operations – one operation possibly being emitted after another one. Decentralized protocols using conflict-free replicated data types (CRDT) guarantee that the user intention is preserved by using only operations that are associative, commutative and idempotent (applying them once or several times yield the same result) [15].

Aims and Contents. Considering the needs of collaborative music annotation, and taking ideas from existing techniques, we propose a distributed protocol to handle simultaneous editions on a set of *labels* on traditional scores. The next Section details our motivations and use cases, Section 3 describes the proposed protocol, Section 4 presents the implementation and the evaluation, and Section 5 concludes on the availibility and the perspectives.

[2] `flat.io`

Fig. 2. Children using Dezrann in a public secondary school in Amiens (France) to annotate sections in music. The school curriculum in music education in France includes a part of "music analysis", without assuming that the children are music readers. Having a way that children annotate themselves music make them active. It improves their learning, and more generally their autonomy.

2 Annotation Model and Use Cases

Annotating music can take different ways, even when one considers only scores in common music notation. We use here the simple notion on *labels* on the score as defined in [2]. Each label has several *fields*. It has a start onset, it may have a duration, or not, and it may concern the whole system, or, more precisely, one or several staves. Common labels to analyze tonal music are patterns, cadences, harmonic markers, or structural elements. This simple modeling does not cover all aspects: Most notably, people analyzing music on paper frequently draw connexions between labels. Here relations between labels are not modeled, but nevertheless labels can have tags or comments. Within this simple modeling, collaborative edition could be worth both in colocated and remote situations:

- *Event-based interactive collaboration.* People, usually in the same place, discuss music while annotating it on their computers, tablets or other mobile devices. They see how the other people interact and annotate the score, for example in the following situations:
  - *specialized music teaching* (general music teaching, or more specialized lectures such as composition, analysis, or history): Students can learn notions such as the sonata form by annotating scores collaboratively;
  - *music education:* Without detailed analysis on the score, music culture and theory can be shared and experimented, for example by identifying sections and other striking events on a waveform (Figure 2);
  - *conference or public concert:* Audience feedback could be organized by letting people annotate the music.
- *Remote collaboration.* People in different places may, over the network, discuss on a score or a waveform. This could be a few collaborating people – artists preparing a concert, or scholars working on scores – or many more people, in a *social network* to annotate music. Like on platforms like *soundcloud*, people may want to share what they hear and comment music.

These use cases are not exclusive: Collaborative edition makes it possible to host *distributed events* on the network. Moreover, one could also *combine "realtime" and "offline" parts*: A collaborative annotation done during a lecture or

another event could be later available for remote viewing or editing. Conversely, artists or students could prepare their own annotation of a score, and thereafter share and edit collaboratively this annotation with others, either to a group of identified users or towards everyone.

Note also that the collaborative edition could be *unrestricted*, where every user may annotate various places in the music. However, one could also have mechanisms to *highlight* the edition or the interaction of one particular user, as a teacher or a presenter. This could include the ability to *follow the session* of such an user, in particular by browsing the music at the same position of her.

3 Operations Modeling and Network Protocol

The following paragraphs describe how we model the collaborative edition between several *user clients* and a *server*. OT and CRDT bring some interesting ideas as the preservation of *casual links* and the *user intention*, the fact that the operation should be applied in any order or the idempotency operations. However, transforming operation in OT is a complex task, and OT modeling is sometimes error-prone [9]. Moreover, in our scenarios, it makes no sense to merge some conflicting operations such as when two users try to move the same label both to the left and to the right. One of these users should know as soon as possible that he is conflicting and then rejoin the shared state. We thus propose a simpler, centralized model based on a centralized linear history but with "Operation Commutations". Compared with OT, we use only trivial transformations on commutative operations, allowing them to be delayed. The consequence is that not all operations are accepted by the server, leading to limited conflicts that may be easily handled by the concerned clients.

State and operations. On a given score, the *state* of the annotation is the result of the application of an *operation history* onto the State 0 – without any labels. We consider three operations: *label-create(id, dict)* creates a label, *label-remove(id)* removes a label, and *label-set(id, dict)* sets some fields of an existing label.

Label *id*s are computed by hashing and are supposed to be unique. The *dict* arguments are dictionnaries containing one or several *(field, value)* pairs. For example, a label may be created by *label-create(5x8, {start: 10, duration: 4})* and another label can be updated by *label-set(d9t, {start: 36})* or *label-set(d9t, {type: 'Cadence'})*.

Operation commutativity. We consider that two operations performed by different users *on different labels* do commute, because labels are independent objects, without casual links. We also consider that, on a same label, two operations *on different fields* do commute. This allows to be very flexible while keeping the server simple. The *label-set(d9t, {start: 36})* and *label-set(d9t, {type: 'Cadence'})* operations do thus commute, enabling to simultanously edit the same label *d9t*, as on Figure 3. In fact, the server is not aware of the actual semantics of the operation: To check whether two operations on a same label commute, it just checks whether they concern different fields or not.

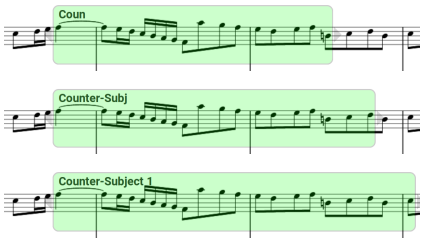

Fig. 3. Andreas is taking a few seconds to enter the comment of a label, here `Counter Subject 1`. One operation is sent by his client at each character stroke. While this, Beatrix is slighlty updating the end of that pattern, resulting again in several operations while she uses her mouse. Her operations do commute with Andreas' operations. Both of them can thus simultanously edit the label.

Client-server handshaking. The server remembers a *state number* s_0 and an *history*[ ] array of size s_0 with all *accepted operations*. User clients may request at any time this full history, in particular when they join the collaborative channel. Each client i, with $i \geq 1$, keeps his own *state* s_i reflecting the last acknowledgement (OK) he received from the server, and a list of his *pending operations* that were sent to the server since this s_i. A client wanting to do an operation op_k sends (s_i, op_k) to the server (Figure 4). When the server receives such a pair:

- If $s_i = s_0$, the client i was aware of the last accepted operation. The server accepts the operation op_k, meaning that he increments $s_0 \leftarrow s_0 + 1$, then stores $history[s_0] = op_k$, and broadcasts to each client $\text{OK}(s_0, op_k)$;
- If $s_i < s_0$, the client i was not aware of the accepted operations since $history[s_i]$. The server checks whether op_k commutes with all $history[s_i + 1] \dots history[s_0]$ operations that were not from the same client:
  - If the operation commutes (or if there were no operations from other clients), then op_k is accepted *at a new position*, that is $s_0 \leftarrow s_0 + 1$, $history[s_0] = op_k$, and broadcast $\text{OK}(s_0, op_k)$;
  - In the other cases, the server sends $\text{DENY}(op_k)$ only to the client i.

When a client receives $\text{OK}(s_0, op_k)$ from the server, he updates his s_i. When op_k was his own operation, he removes it from his list of pending operations. Otherwise, when op_k is commuting with his pending operations, he applies it ($\star$ on Figure 4). When this is not the case ($\bullet$ on Figure 4), or when he receives a $\text{DENY}(op_k)$, the client detects a conflict and rollbacks some operations, possibly asking some of the *history*[ ] to the server. Thus when two or more users are in conflict, some of them will be blocked, but it can be acceptable.

The protocol allows to detect lost messages: Whenever a client receives $\text{OK}(s_0, \ldots)$ with $s_0 > s_i + 1$, he can ask again some of the *history*[ ] to the server. As the history is linear and as the only transformation is to switch commutative operations, the protocol tries to preserve some form of casual links and user intention. Note that all the update operations are idempotent, and could thus applied several times in case of re-emission following network failures.

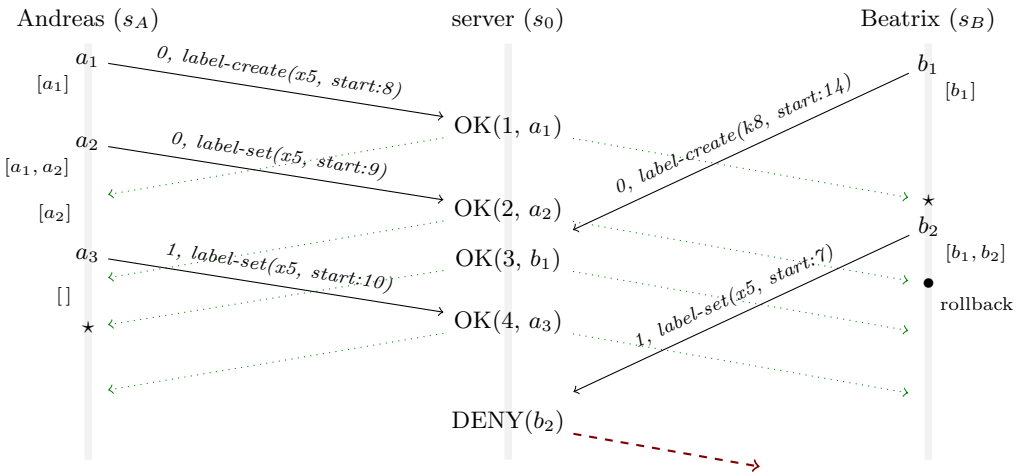

Fig. 4. Example of protocol based on Operation Commutations for collaborative music annotation. Andreas emits operations a_1 and a_2. The server accepts these operations, stores them in *history*[], and broadcasts the acknowledgements. Each time Andreas receives the acknowledgements, he update his s_A with s_0 and, when the concerned operation was from him, he deletes it from his list of pending operations. Beatrix emits the operation b_1 = *label-create(k8, start:14)*, without being aware of operations a_1 and a_2. However, b_1 commutes with a_1 and a_2, and thus is accepted, and a_3 is also accepted because it commutes with b_1. Contrarily, the further operation b_2 = *label-set(x5, start:7)*, where Beatrix want to move the label *x5* in a conflicting way, does not commute with a_2 nor a_3. The server sends thus a DENY back to Beatrix. However, Beatrix could have detected the conflict before, when she received OK$(2, a_2)$ at (•).

4 Implementation and Evaluation

4.1 Implementation

The Dezrann codebase includes several user interface parts, using the *Web components* model (`Polymer.js`), that will be not discussed here [6]. The server part, written in `node.js`, handle user authentication through JWT (json web tokens) and storage of corpora and annotation files, together with their metadata. The collaborative annotation protocol was implemented above `socket.IO`. The payload of each operation message is (s_i, op_k), as explained above, together with other data such as hashes to uniquely identify operations.

The conflict handling is now not optimized on the client side: The clients do not detect by themselves the conflicts, but rather wait for DENY messages from the server. Moreover they request the full state after each conflict, and do not check for lost messages. The protocol still works with such unoptimized clients, except when messages are lost. To benchmark the server part, we both tested the actual client, but also simulated fake clients running regular requests.

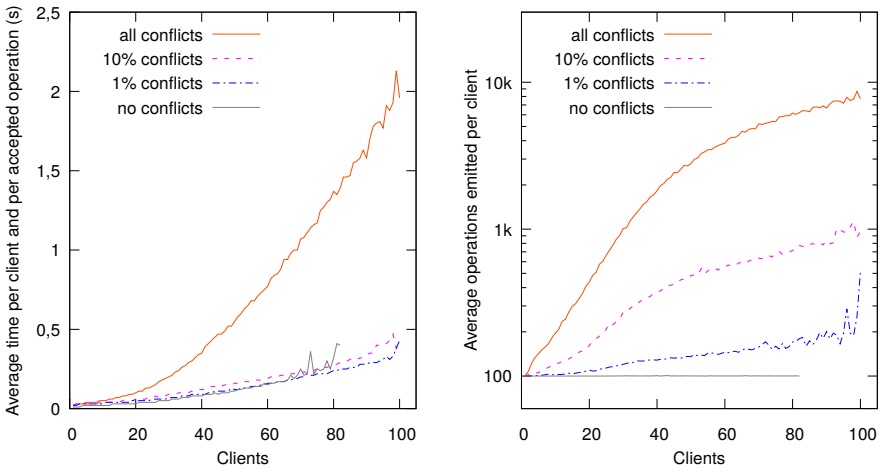

Fig. 5. Time (left) and operations (right, logscale on y-axis) for handshaking 100 operations per client under several scenarios. Parallel clients and server are simulated on a same laptop computer (8 cores, 2.60 GHz, 16GB RAM).

4.2 Evaluation

We simulated a server and many clients performing either commuting or conflicting operations. In the *no conflicts* scenario, all clients move each a different label. In the *all conflicts* scenario, all clients perform conflicting operations on a same label. Indermediate scenarios are when clients update one out of 100 or 10 random attributes of a label (*1% and 10% conflicts*). Parallel clients attempt operations every 0.01 second. When a client receives a DENY, he tries another operation, until 100 of his operations have been accepted. Such a stress test is not realistic but enable to see how robust the protocol is.

Figure 5 shows that the best-case scenarios behave well with up to 100 simultaneous clients. The clients can update labels even without having received previous operations: Starting from 3 clients, more than 99% of the (commutative) operations are here delayed. The worst-case scenario handles until about 20 clients simultanously emitting conflicting operations. Above this number, the server slows down but is stil not stalled.

Finally, we performed client-side evaluation, with until 10 people working on the actual Dezrann clients and annotating collaboratively a score. No server stall was recorded, even when one sometimes feel that an operation was denied.

5 Conclusion, Availibility and Perspectives

We proposed a protocol based on Operation Commutations to handle simultaneous collaborative editions on labels on a music score. This simple protocol is between OT and lock mechanisms. Conflicts do occur but may be ef-

ficiently handled. Simulations show that it enables a scalable use by multiple clients, in scenarios both without and with conflict, and human evaluation confirms that the platform works for a few simultaneous people, answering the needs for the envisaged use cases. Prototype implementation is available in the `dez-{server,client}/dez-collab` directories on `gitlab.dezrann.net`.

Perspectives on the protocol includes improving client (re)joining a collaborative channel, notably by better handling and compressing history to avoid spurious transfers of the full *history*[] table and also by using the local storage of the web application. The protocol could also use more OT techniques for text edition in tags and comments. On the development part, perspectives include improving the user interfaces to make the fully useable by everyone, using authentification mechanisms to grant collaborative edition accesses, and finally conducting more user tests both with adults and children groups.

References

1. J. T. Allison et al., NEXUS: Collaborative performance for the masses, handling instrument interface distribution through the web. In *New Interfaces for Musical Expression (NIME 2013)*, 2013.
2. G. Bagan et al., Modélisation et visualisation de schémas d'analyse musicale avec music21. In *Journées d'Informatique Musicale (JIM 2015)*, 2015.
3. P. Couprie. iAnalyse : un logiciel d'aide à l'analyse musicale. In *Journées d'Informatique Musicale (JIM 2008)*, 2008.
4. C. A. Ellis and C. Sun. Operational transformation in real-time group editors: issues, algorithms, and achievements. In *ACM Computer supported cooperative work*, 1998.
5. M. Giraud. Using Dezrann in musicology and MIR research. In *Digital Libraries for Musicology (DLfM 2018)*, 2018.
6. M. Giraud et al., Dezrann, a web framework to share music analysis. In *Int. Conf. on Techn. for Music Notation and Representation (TENOR 2018)*, 2018.
7. H. H. Hoos et al., The GUIDO music notation format. In *Int. Computer Music Conf. (ICMC 1998)*, 1998.
8. D. Huron. Music information processing using the Humdrum toolkit: Concepts, examples, and lessons. *Computer Music Journal*, 26(2):11–26, 2002.
9. A. Imine et al., Proving correctness of transformation. In *Eur. Conf. on Computer-Supported Cooperative Work*, 2003.
10. G. Lepetit-Aimon et al., INScore expressions to compose symbolic scores. In *Int. Conf. on Techn. for Music Notation and Representation (TENOR 2016)*, 2016.
11. J. Malloch et al., A network-based framework for collaborative development and performance of digital musical instruments. In *Int. Symp. on Computer Music Modeling and Retrieval (CMMR 2007)*, 2007.
12. E. Paris et al., KIWI : Vers un environnement de création musicale temps réel collaboratif. In *Journées d'Informatique Musicale 2017* (JIM 2017), 2017.
13. L. Pugin et al., Verovio: A library for engraving MEI music notation into svg. In *Int. Society for Music Information Retrieval Conf. (ISMIR 2014)*, 2014.
14. C. S. Sapp. Verovio Humdrum Viewer. In *Music Encoding Conf. (MEC 2017)*, 2017.
15. M. Shapiro et al., Conflict-free replicated data types. In *Symp. on Self-Stabilizing Systems*, 2011.