CRIM 2020 Development Tasks

Improve Data Entry System

- Update Note Selection and Data Entry System
    - Leader: RV June-August 2020
    - RV: Plans to return computation to the browser and only read once and write once from/to the server. This should bring us back at least to speed levels of the original prototype before Django. EMA processing via OMAS, I believe, is the worst offender here; see Verovio section.

Verovio Rendering and Caching

- Can we make this faster and more reliable?
    - Leader: RV June-August 2020
    - RV: I would re-work these entirely. I imagine replacing OMAS with a simpler in-browser library. Optimize front-end code to avoid pages from hanging.
- Systematic caching server side of each page?
    - Question from RF and AJ: how will the new MEI slices and Verovio be handled by Django? Is there anything that we need to be mindful of in updates to CRIM Django that will then change again when the EMA/OMAS/JS system is updated?

Speed Updates

- Currently many of the Django templates and views involve the harvesting of large amounts of data from the server. We don't actually need all of these with each query.
    - Leader: MJW Late May to July
    - MJW: Daniel R-B. pointed out that more efficient queries could be made in Django to alleviate some of the slow speeds. We've already made an adjustment to allow for lists of "brief objects" (less data per object) in the data view, for the sake of external analysis.
    - RF and AJ:
        - Freddie Gould could help with testing of timing and other issues
        - AJ can implement more CPUs or memory as needed if we want to have parallel processing of some tasks

Missing Metadata Fields

- Some things currently only in Mechanical Transformation are also needed in Non-Mechanical Transformation. This confusion is responsible for overmarking of types.
    - Leader: MJW July/August
    - MJW: Yes, this is an important one. It shouldn't be too hard to change the models, but we'd have to migrate the database again, and then probably to go over and curate the data again afterwards.

Piece Relationship and Piece Observation Lists

- Allow "sort options: in the various views?
- Is this contingent on decisions about indexing, or perhaps is done just with Django View.

Faceted Search

- Is Solr the right tool for the job?
    - Leaders: MJW and AJ

- RV: consider elasticsearch? They both use the same underlying system, but I've had better luck with "up time" with elasticsearch...
- MJW: We might decide this sooner rather than later, as I'll be reinstalling Solr 8 soon. We'll want to enlist Adam's help to make it more secure this time around. Even if Solr is overpowered for our needs, it does have the advantage that we've done it before... so it may be easier to get running again and maintain. It doesn't seem to be missing any major functionality.
- AJ: it's possible that adjustments to CPU or memory could solve the problems, too. But there are other approaches available for indexing.
- AJ: Freddie can test to see what is going on.

Audio Rendering with MEICO

- Leader: MJW, with help from Freddie and Andy.
- Can we integrate with Django server and allow MEICO to play individual EMA slices? Play entire pieces? This will involve passing either full MEI files or MEI slices for Each Observation to MEICO
- We will need to add some content to the Piece and Observation templates in CRIM Django (to launch audio file) see Auralization for mockup.
- RF: We will need to *remove* "oct.ges" from MEI slice, or from all of the MEI files as part of massaging, since these are wrong for Tenor and Altus parts that use the G8 clef: REGEX: oct[.]ges="."
- See Auralization links on CRIM editor site for details, also messages from Axel about these below. He suggests using MEICOpy in a python environment (and thus alongside CRIM Django?)

> Dear Richard,
>
> The integration in Django, sounds more flexible and convenient from the users' perspective but requires probably more development effort.
>
> **JAVA:** Do you have a Java integration in Django? Can you run Java programs there? If so, you can integrate meico as easily as demonstrated in the commandline application (https://github.com/cemfi/meico/blob/master/src/meico/app/Main.java). This is, for instance, the case with MEI Garage where Daniel Röwenstrunk can integrate meico just like a usual programming library and invoke its methods. Here is some demo code:
>
> - Mei mei = new Mei(...); // replace "..." by your input file, InputStream or text string
> - List<Msm> msms = mei.exportMsm(); // this gives you a list of MSMs, one for each <mdiv>
> - Msm msm = msms.get(0); // this gives you the first MSM in the list

- Midi midi = msm.exportMidi(); // now you have the MIDI data that you can store or play back or ...
- MidiPlayer player = new MidiPlayer(midi); // this creates a MIDI player and loads the music into it
- player.play(100500); // this starts playback at time position 100500 milliseconds

I left out exception handling for better readability. You usually check whether the input file really exists and could make an Mei object etc.

PYTHON: MeicoPy does basically the same as the commandline app but out of a Python environment. It instantiates a Java environment (via package JPype) and invokes the meico commands within this environment. That is why the syntax looks a bit different to pure Java. By the way, I heard that the JPype package got much better in past years. MeicoPy is a bit outdated as I never heard of someone using it this way, so I never updated this and it is now a bit behind the state of the commandline app.

If integration in Django is not so easily feasible you can, of course, write your own commandline app, quasi a script (or even a graphical user interface) that automatizes all the stuff from idea a). Thus, you win at least a bit more convenience.
So, there are several ways you can go. Ask your developer about Java support in Django.

There is also another, completely different way you could go: Use meico to convert you music to MIDI - no slices, just the whole pieces. Upload these and do MIDI processing in Django. You can mute/unmute channels (staffs), jump to any playback position, stop playback whenever you want ... MIDI is so old and widespread, it is supported by virtually every programming language. To find out which MEI note id corresponds with which MIDI time position (in ticks) you can use the MSM for lookup. Here, every note element has the original MEI xml:id and a date attribute with the MIDI tick value. This approach reduces the effort of idea a) to a minimum.

I hope this helps.

By the way. I work on an extensive meico update, at the moment, that integrates the Music Performance Markup format (MPM) and according rendering to expressive MIDI sequences. It won't generate performances out of nothing. But it will apply the performance information (tempo, dynamics, articulation) that the MEI source provides. And by editing the MPM file you can create quite rich performances. It will take me some weeks/months, so there will be not much action on the Github page for a while. And I plan to give a first workshop on this at the next Edirom Summer School. So, if this is interesting for you, new things are on the way.
Best wishes, Axel

Differentiated Highlights

- *For possible future development. Not for Summer 2020*
- Some way to show different voice roles via different colors? Or perhaps simply play them selectively with MEICO?

    - MJW: This could be useful and feasible in some limited instances (such as cadence role types), but we'd want to think very carefully before making it another hurdle for analysts to overcome.

Heat Maps and Visualizations

- see: https://crim-vis.herokuapp.com/pieces/
- **Integrate with Piece View in Django**. That is: each individual work will have its own Heat Map loaded via a link.
- Create some **filter features** that allow users to view only selected *types* (as well as *regions* of a piece)
- For Network Viz:
    - Extend meta-data based tree diagrams of "related" musical or relationship types, building on DRB prototypes.
    - Pieces related by similar procedures as well as musical patterns?
    - Refine the finding tools. *Where* are the borrowings? Similarities?
    - Limit by certain musical types or relationship types? Analysts?
- Can we have HeatMaps of the sort we see in JRP: maps of rhythmic and melodic activity in a given piece (quite apart from any set of Observations or Relationships)? Would these provide some indication of pace or density of attacks, for instance?

Data Analysis

- *Grounding Similarity in Music*. We are interested in "similarity in music", which stands behind almost any attempt to explain *style* (which is about what works of a certain "sound" share at the highest level), *genre* (which binds pieces by purpose or pattern) and *borrowing* (when one piece quotes, reworks, or otherwise borrows from another specific piece). Even our attempts to show how a single piece holds together (or not) hinges on our capacity to show how its motives, harmonies, or gestures are "similar" or "not." Musicologists have long been concerned with these questions, but how can machines be used to advance such work? Can they show us more about…
    - *Match Model with Derivative*. This is our corpus' primary information: each of its Masses was created in 'direct reference' to one pre-existent model. Any machine we create should first be able to master the basic task of telling us which model goes with which derivative.

- *Ranking Similarity*. Can we find degrees of similarity among pieces? Aside from those pieces bound by explicit Model-Derivative, what would subsequent degrees of similarity look like?
    - Similar *soggetti* (certain melodic-rhythmic patterns)? Can these be found via HumDrum or similar tools? Via some direct search of MEI? Via some search of tokenized MEI?
    - Similar *counterpoint* (certain interval patterns, or 'interlocks'). See Three-Grams noted below. These could be found in the MEI, or some representation of it.
    - Similar *presentation* types (via metadata, the time or intervals of entry)
    - Similar *procedures* (via relationship type metadata; pieces that share some *habits* of quotation, transformation, etc).
    - *Economy and Consumption*. How much of a model is used? How much of a derivative is new or borrowed? These would also tell us something about how works are like each other? Perhaps this would also include some measure of *where* the borrowed material appears in each piece (example: some mapping of borrowings according to beginning/middle/end of each piece).
    - The metadata would also tell us about similarity within and beyond a given piece:
        - Simply Identity: The same EMA and same Metadata: its the same passage, but only if the Derivative and Model are the same in all instances!
        - Self Similarity: Same Piece, Different EMA but the same Metadata: these are similar passages in the same composition.
        - Exo Similarity: Different pieces (and of course different EMA) but the same metadata. These could be:
            - Model and Mass (that is, some example of borrowing)
            - Any Pair of pieces _not_ otherwise known to be Model and Mass. These share some stylistic or proceduraly similarity!

- Specific Techniques and Tools.
    - jSymbolic, which might be applied as follows:

        - pieces as Bag-of-Notes (including the Movements of Masses as individual pieces in this case)
        - phrases as Bag-of-Notes (using CRIM Phrase data to segment the phrases, perhaps relying in information about words and rests in the melodies)
        - various other Bags such as date, place, genre of model, or genre of Mass Movement
    - Network Graphs using CRIM metadata
        - examples such as DRB's Network of PEns, based on sequences of time or melodic entries of voices. others?
        - possibly networks of soggetti, based on tokenized representations of melodic and rhythmic motion
    - Linguistic Tools, such as Levenshtein Distances.
        - DRB has already experimented to show which Observations are most similar to each other, based on one such routine using tokenized representations of melodic intervals of entry
- Pattern Finding
    - ELVIS technique of three-grams at the level of the minim involving modules of two voices. Here they represent a fundamental motion in a lower part by (a 'vector' of movement up or down by diatonic interval X) surmounted by a pair of vertical harmonic intervals (for instance a sixth followed by a third). This 'triad' represents the basic element of contrapuntal motion. Any succession of voices can be represented as an interlocking chain of such three-grams.

Ontology, LOD and Data Export

- Refine Ontology for CRIM and for OMAC (claim ontology)
- Module for Export of CRIM data as LOD-expressed OMAC or other standard.

Pedagogical Tools and Modules

- Screencasts and Slide Shows to demonstrate concepts and interfaces for
    - musical style and analysis

    - data entry and mark up
    - MEI and Verovio
    - Django and data models
    - Ontologies
    - Data Analysis and Visualizsation
- Code and Research Dashboards that would allow folks to “play” with . .
    - **Learning MEI** (from Music Encoding Initiative, but perhaps adapted to the needs of CRIM).
    - **Mensural Notation Edito**r (from Karen Desmond)
    - **Interact with CRIM Analytic Metadata** (jupyter or similar frameworks that would allow students to interact with CRIM Data, querying certain kinds of information from the CRIM Django database)
    - **Experiment with Similarity Engines for Music Data** (MEI>music21, plus various ‘tools’ for exploring similar melodic patterns or contrapuntal matricdes.
        - See AJ n-gram project notes
        - See AJ n-gram project code
    - **Network Graphs, Heat Maps, and other Visualizations**
        - Heat Maps (different view)
        - Networks of pieces (using music data or analytic metadata)
    - **Friendly environments for the above** that would allow students (DH, musicology, advanced and beginner) to experiment with any of the above, and make new knowledge
        - streamlit.io Streamlit is an open-source app framework for Machine Learning and Data Science teams. Create beautiful data apps in hours, not weeks. All in pure Python. All for free.
        - https://observablehq.com/ Discover insights faster and collaborate seamlessly with interactive Observable notebooks built for data analysis, visualization, and exploration.
        - How to use Django in Jupyter Notebook (Jupyter queries Django directly)

        - Colab (Andy's music21 and n-gram code) for hosted Jupyter-style interaction without need to install things locally.

Computational Essay Platform

- read: Steven Wolfram, "What is a Computational Essay?" (2017). URL: https://writings.stephenwolfram.com/2017/11/what-is-a-computational-essay/
- Resources and platforms that would allow researchers to "publish" arguments involving particular views, code, transformations, or queries
-