Music Performance Markup: Format and Software Tools Report

Axel Berndt
Center of Music and Film Informatics
University of Applied Sciences and Arts OWL, Germany
axel.berndt@th-owl.de

Abstract

With Music Performance Markup (MPM) we introduce a new XML format for describing musical performances in a systematic way. The format builds upon a series of mathematical models that capture the characteristics of performance features such as continuous tempo and dynamics transitions, articulations, and metrical accentuations. Bundled with MPM comes an infrastructure of documentations and software tools. This paper aims to provide an overview of and introduction to MPM, its infrastructure and ongoing development activities.

1 Introduction

Music culture of the past century has been strongly influenced by audio media. Music has been and continues to be published on all kinds of audio media to an overwhelming extent. The study and analysis of such audio documents poses special challenges to musicology. Typical questions are: How can the connection between an audio document and a musical score be drawn? How can the musical realization of a performer be described and put in relation to printed performance instructions and to the performances of other musicians? The tools available to musicology for this purpose still leave much to be desired. Music Performance Markup (MPM) is a contribution to enrich the musicological toolbox with a novel description format.

While music notation can be described in a wide variety of symbolic music formats (MusicXML, MEI, Humdrum, ABC), its performance is a complex transformation process that only partly derives from the notation. MPM offers a systematic and comprehensive approach to the phenomena of musical performance (such as timing, dynamics, and articulation). It enables the publication and reuse of research results from musicological performance research and edition for a variety of application and research contexts, such as musicology, timbre research, music psychology, Music Information Retrieval, and music production. In doing so, MPM and its tools also inspire new research designs such as corpus analyses and analysis-by-synthesis methods. The 'simulation' or digital 'reconstruction' of musical performances allows, for instance, the experimental testing of hearing impressions and hypotheses about how exactly a performance has been made and with which expressive means a certain effect has been achieved.

However, for MPM to become an accepted and practically usable standard, a variety of further efforts are required beyond the mere definition of the format itself. An infrastructure must be created, consisting of software tools (editor, converter, MIDI and audio renderer), guidelines, sample encodings and tutorials as well as development guidelines, which can promote a community-driven further development of the format. In the following, we report on this ongoing effort.

2 Conception

The expressive performance of a piece of music given in symbolic representation – e.g., in common western music notation – is, formally speaking, the entirety of all transformations that are necessary to make the piece sound or to convert it into an equivalent audio signal. This includes the chronological progression of sound events as well as their specific execution. In the course of several years of fundamental research, a series of mathematical models has been developed by which these transformations can be plausibly emulated [1, 4]. These models currently capture the following performance features:

- timing: tempo (incl. discrete and continuous tempo changes such as ritardando and accelerando), rubato, asynchrony, random/unsystematic deviations from exact timing;
- dynamics: macro dynamics (incl. discrete and continuous dynamics changes such as crescendo and decrescendo), metrical accentuations, random/unsystematic deviations from exact dynamics;
- articulation: with absolute and relative effects on tone duration, loudness, tuning, and timing (e.g., agogic accents) as well as random/unsystematic fluctuations of duration and tuning.

MPM is an XML-based data format and can be regarded as a scripting language for describing musical performances by means of these models. Simply put, it specifies when to apply a performance feature to a musical part. In this respect, MPM is distinct from the generative models that dominate the literature on this subject; for an overview, see [6]. The models in generative systems map a musical input to a specific performance. MPM's feature models, on the other hand, are purely descriptive. Their input parameters serve the purpose of a user-friendly customization of such features (e.g., the shape of a tempo curve, the effect of an articulation, the stochastic character of dynamics fluctuations). Generative systems may use MPM as a construction kit to render their generated performance plans, though.

Given the inherent structure of MPM's descriptions and the fact that one and the same piece of music can be performed in many different ways, we opted against expanding an already existing, structurally incompatible data standard (like MEI) but to conceive MPM as a format to be used in tandem with other symbolic music formats (incl. MEI) [2, 3]. Thus, those symbolic music formats constitute the notes to be played while MPM describes how they are played.

3 Schema Definition and Documentation

At the beginning of the process, we executed an analysis of the application scenario, i.e., the musicological work with markup formats for music encoding and description. MPM is an XML-based data format. Working with XML editors, most notably the Oxygen XML Editor, is common practice. For special tasks, more comfortable editors with optimized input masks or graphical user interfaces can also be found, such as the metadata editor MerMEId [9, 10] and the Vertaktoid tool for measure annotation [11]. Nevertheless, and especially for advanced users, the XML editor remains the primary tool. For productive work, two editor functions are of essential importance: automatic code completion and (live) validation. Both require a fully comprehensive format definition in a suitable schema language.

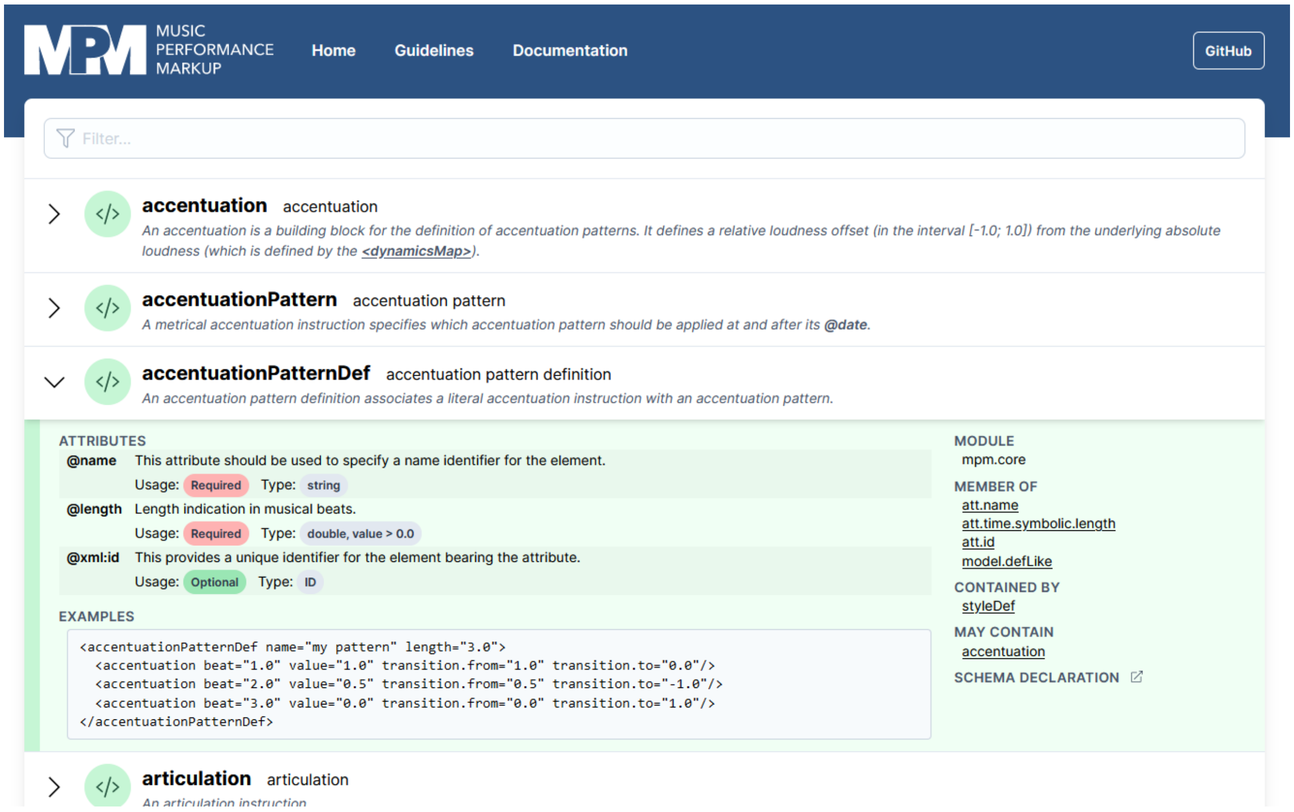
The MPM schema definition is therefore of essential importance for productive work in the XML editor. The schema was created by means of the TEI ODD meta language, which can be compiled into a number of different other schema languages, such as RNG, XSD and DTD, so that maximum compatibility with the commonly used XML parsers is provided. The MPM schema is published in a GitHub repository[1] under the BSD-2 and CC-BY-4.0 open source licenses. The precompiled release assets include an RNG and XSD compilation.

In its current version 2.1.0, the MPM schema is fully specified. Beyond the purely syntactic definition, it also implements several Schematron rules for content-based tests. These are also included in the validation and detect content-related errors, such as non-existent references, violations of value ranges or invalid value combinations. Everything is communicated to the user through appropriate warning and error messages.

The complete textual documentation was also realized directly in ODD and can be exported to various formats, including HTML and PDF. The documentation includes all elements and attributes, always with compact

1 https://github.com/axelberndt/MPM (accessed January 12, 2022).

code examples, furthermore an extensive guidelines text, which serves as a practical handout and introduction to MPM. The latter introduces the underlying concepts, mathematical models and their representation in XML, explains the structure of MPM documents and supplements this with more detailed code examples.



accentuation accentuation

An accentuation is a building block for the definition of accentuation patterns. It defines a relative loudness offset (in the interval [-1.0; 1.0]) from the underlying absolute loudness (which is defined by the <dynamicsMap>).

accentuationPattern accentuation pattern

A metrical accentuation instruction specifies which accentuation pattern should be applied at and after its @date.

accentuationPatternDef accentuation pattern definition

An accentuation pattern definition associates a literal accentuation instruction with an accentuation pattern.

ATTRIBUTES

@name	This attribute should be used to specify a name identifier for the element. Usage: Required Type: string
@length	Length indication in musical beats. Usage: Required Type: double, value > 0.0
@xml:id	This provides a unique identifier for the element bearing the attribute. Usage: Optional Type: ID

EXAMPLES

```
<accentuationPatternDef name="my pattern" length="3.0">
  <accentuation beat="1.0" value="1.0" transition.from="1.0" transition.to="0.0"/>
  <accentuation beat="2.0" value="0.5" transition.from="0.5" transition.to="-1.0"/>
  <accentuation beat="3.0" value="0.0" transition.from="0.0" transition.to="1.0"/>
</accentuationPatternDef>
```

MODULE
mpm.core

MEMBER OF
att.name
att.time.symbolic.length
att.id
model.defLike

CONTAINED BY
styleDef

MAY CONTAIN
accentuation

SCHEMA DECLARATION

articulation articulation

An articulation instruction

Figure 1: A screenshot of the MPM web page in the documentation area. All elements, model and attribute classes are listed and documented here. This also includes short exemplary code snippets. A filter function makes it easier to find entries quickly.

In addition to the transformation from ODD to HTML provided by the TEI OxGarage and XSLT, a custom transformer (`doc_generator`) was developed. The web page[2] created with it includes both, documentation and guidelines, and marks the best entry point for anyone willing to work with MPM (see Figure 1). The `doc_generator` is published in the MPM repository along with the MPM schema and is part of the continuous integration pipeline. Thus, in the future it will suffice to upload schema updates in the ODD source code; the generation of the web page and all release assets is fully automated.

Furthermore, the repository contains sample encodings currently for three pieces of music:

- monophonic piece: Johann Sebastian Bach, Minuet No. 2 from Cello-Suite, BWV 1007 (one performance);
- homophonic piece: Melchior Vulpius, four-part chorale harmonization of „Die helle Sonn' leucht' jetzt herfür“ (three contrasting performances);
- polyphonic piece: Georg Philipp Telemann, Grave from TWV 51-D7 (three contrasting performances).

Each encoding includes the raw MIDI data (resp. symbolic music without performance), the MPM file with its performances, a PDF file of the score with performance annotations, and an audio rendering of each performance.

2 https://axelberndt.github.io/MPM/ (accessed January 12, 2022).

4 Software Infrastructure

Working with MPM should not be limited to the comparatively inefficient and unintuitive editing of XML code in XML or text editors. More convenient software tools are developed. Also, these tools should not have to implement the basic functionalities for parsing, creating, editing, and storing MPM data each time anew. It is the task of the Application Programming Interface (API) to provide such a reusable interface for application development.

4.1 Application Programming Interface and Performance Rendering Engine

In parallel to the schema definition, an API for the MPM format was developed. The MPM API was implemented as part of the Java library *meico*[3] [5]. *meico* is already a well-developed conversion tool for several music formats (including MEI) and provides some functionality that usefully interacts with the MPM API. *meico*'s MEI-to-MSM (Musical Sequence Markup) export has been extended to export MPM data from MEI encoded performance instructions as well.

A fully featured performance rendering engine has been integrated into the MPM API. This allows the performances described in MPM to be applied to score material in MSM format to create expressive MIDI sequences. MSM is designed as a partner format to MPM with a similar structure and syntax. Originally, it served as an intermediate format in *meico* and can currently be generated from both MEI and MIDI data. The expressive MIDI sequences can then be saved as MIDI files or converted to audio (WAV, MP3).

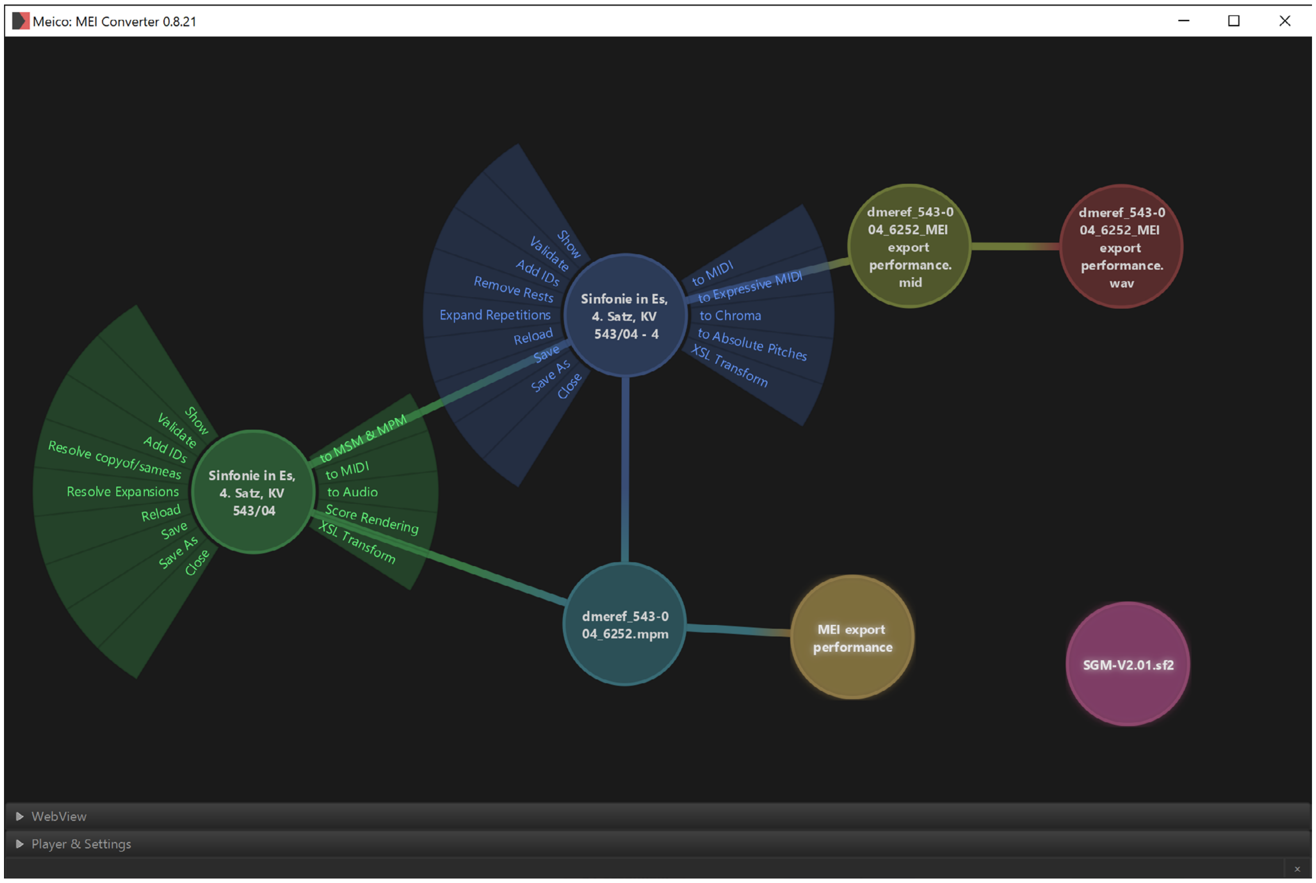


Figure 2: Screenshot of the *meicoApp* showing the integration of MPM. From the MEI data (left), MSM (top, dark blue) and MPM (bottom, light blue) data are exported. The MPM here contains only one performance, i.e., "MEI export performance" (bottom), that is activated. This allows the MSM data to be rendered "to Expressive MIDI" (top, yellow) and the MIDI data to be played directly and converted to audio (top right). At the bottom right, an external soundfont has been activated, which produces better quality instrumental sounds than the standard timbres provided by the Java Runtime Environment or the operating system.

3 https://github.com/cemfi/meico (accessed January 12, 2022).

However, *meico* is not just a programming library, but also provides two application programs in the form of the *meicoApp* package. The command line application can be integrated into XML editors or a scripting environment (e.g., in Python programs) as an external invocation. The graphical application serves as a standalone tool and offers more interaction possibilities as well as more diverse conversion paths in a convenient graphical user interface. *meico*'s main application scenarios so far have included proof listening as part of the music editing process and the creation of sound samples for the music encoded. The MPM API has been integrated into both *meicoApp* applications (see Figure 2), so that users can now easily listen to performances and save them as MIDI or audio. At the same time, this fulfills a wish expressed several times, i.e., to translate not only the notes but also the performance instructions encoded in MEI into MIDI and audio.

4.2 Graphical Editor: MPM Toolbox

Furthermore, a graphical editor called MPM Toolbox is developed.[4] Its user interface design was led by basically three major use cases:

1. the free creation of performances (creative use case);
2. the analysis of performance scores and interpretation or annotation of signs in the autograph (analytical use case); and
3. the analysis of audio recordings.

The graphical user interface (GUI) depicts four main widgets. An MSM tree representation of the plain note information is required to match time indications and voice assignments (e.g., to assign articulation X to note Y), but does not need to be editable. An MPM tree representation serves as a fully functional editor.

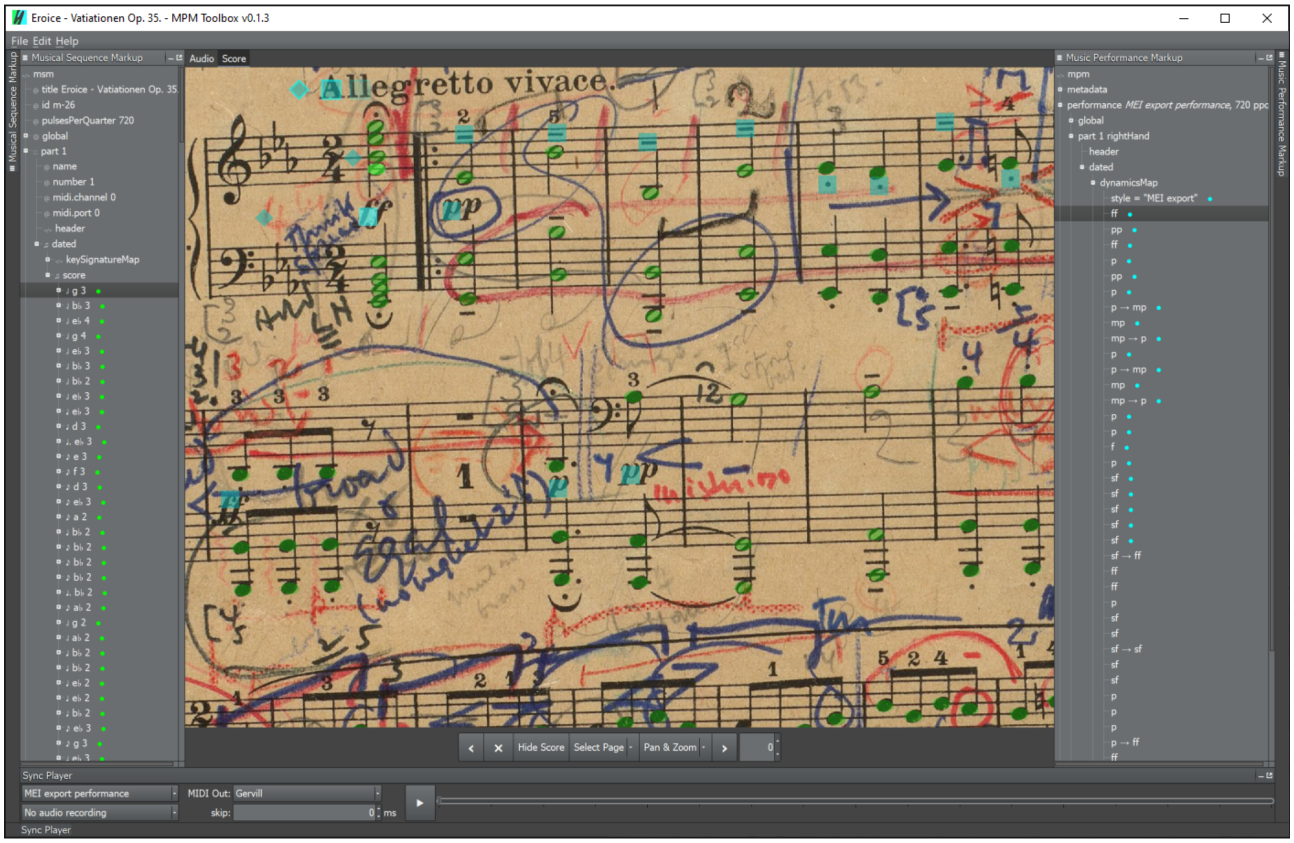
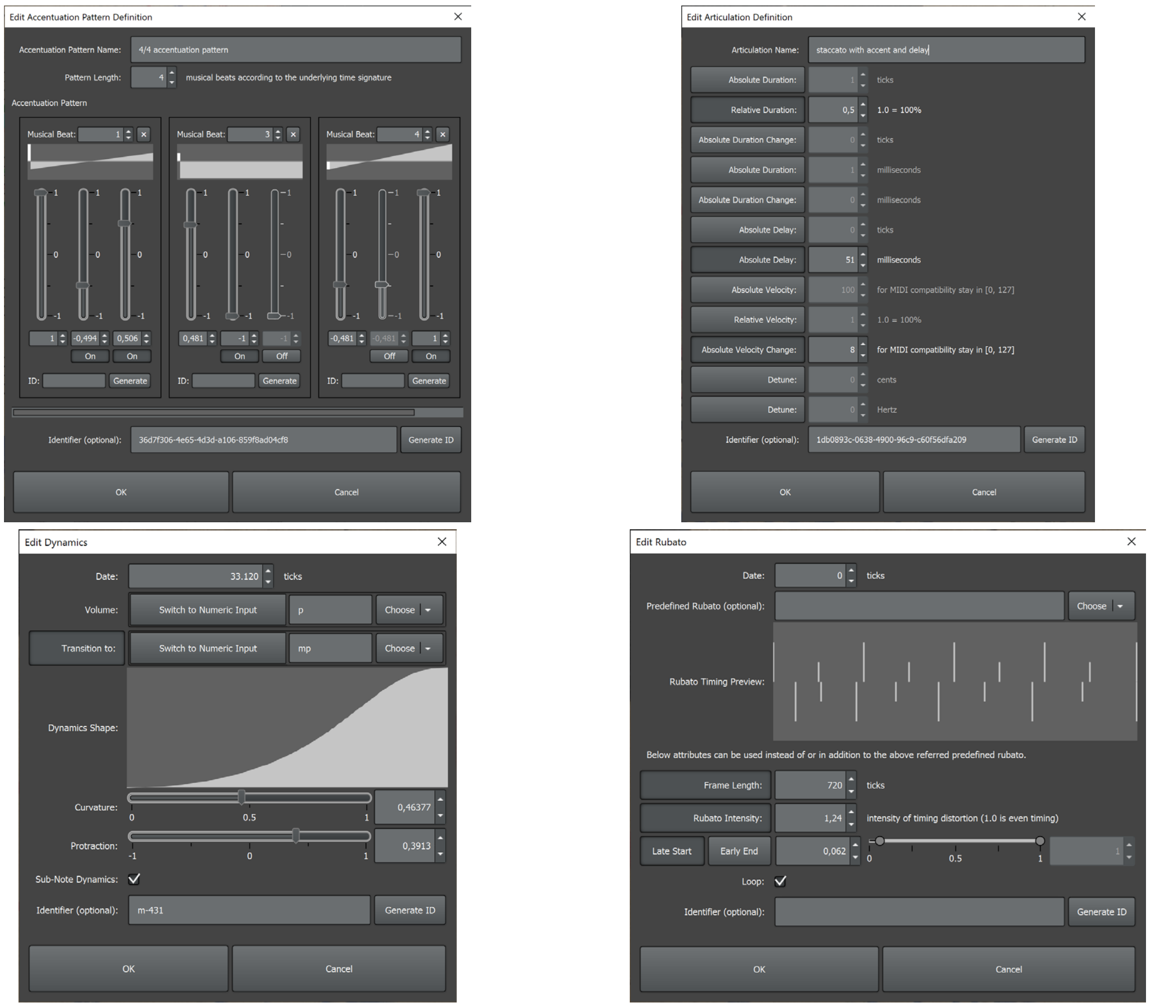


Figure 3: MPM Toolbox's main window. The colored overlays in the score (center) are references to notes from the MSM (green) and performance instructions from the MPM (light blue). Performances can be created either directly in the MPM tree (right) or in the score image.

4 https://github.com/axelberndt/MPM-Toolbox (accessed January 12, 2022).

A score display is used to incorporate autographs, e.g., of performance scores, or any other instance of scanned sheet music. Notes from the MSM and performance instructions from the MPM can be assigned to pixel positions in the score, creating a rich, interactive overlay. It is also possible to create new performance instructions here. This makes the score widget the most convenient and, hence, primary interaction widget in MPM Toolbox's GUI. The fourth widget holds the playback controls. This is not only a means to render a performance description into MIDI and play it back with only one mouse click. Moreover, it enables the playback of an audio file alone or in parallel with the rendering of a performance, facilitating listening analyses of recorded performances, i.e., an analysis-by-synthesis process starting with a rough performance and iteratively refining it in order to match performance rendering and audio recording. A screenshot of the MPM Toolbox's main window is shown in Figure 3.

References to notes from the MSM and performance instructions from the MPM are displayed in the score as colored overlays. They constitute annotations that interpret the notated symbols and assign semantics to them. However, it is not practical to visualize all the information of the performance instructions directly in the score, since it is usually printed rather compactly anyway and the remaining space is important for readability. For this reason, the overlays created with MPM Toolbox are implemented as semi-transparent and compact symbols. The detailed specifications of the performance instructions are done in separate editor dialogs. Each of these dialogs not only provides the respective input options, but also actively ensures that the input is valid, i.e., validates against the MPM schema. Several further convenience functions and auxiliary visualizations support the user in this process. Examples of such editor dialogs can be seen in Figure 4.



Edit Accentuation Pattern Definition

Accentuation Pattern Name: 4/4 accentuation pattern

Pattern Length: 4 musical beats according to the underlying time signature

Accentuation Pattern

Musical Beat: 1 — 1, -0,494 (On), 0,506 (On) — ID:

Musical Beat: 3 — 0,481, -1 (On), -1 (Off) — ID:

Musical Beat: 4 — -0,481, -0,481 (Off), 1 (On) — ID:

Identifier (optional): 36d7f306-4e65-4d3d-a106-859f8ad04cf8

Edit Articulation Definition

Articulation Name: staccato with accent and delay

Absolute Duration: 1 ticks

Relative Duration: 0,5 1.0 = 100%

Absolute Duration Change: 0 ticks

Absolute Duration: 1 milliseconds

Absolute Duration Change: 0 milliseconds

Absolute Delay: 0 ticks

Absolute Delay: 51 milliseconds

Absolute Velocity: 100 for MIDI compatibility stay in [0, 127]

Relative Velocity: 1 1.0 = 100%

Absolute Velocity Change: 8 for MIDI compatibility stay in [0, 127]

Detune: 0 cents

Detune: 0 Hertz

Identifier (optional): 1db0893c-0638-4900-96c9-c60f56dfa209

Edit Dynamics

Date: 33.120 ticks

Volume: p

Transition to: mp

Dynamics Shape:

Curvature: 0,46377

Protraction: 0,3913

Sub-Note Dynamics: ✓

Identifier (optional): m-431

Edit Rubato

Date: 0 ticks

Predefined Rubato (optional):

Rubato Timing Preview:

Below attributes can be used instead of or in addition to the above referred predefined rubato.

Frame Length: 720 ticks

Rubato Intensity: 1,24 intensity of timing distortion (1.0 is even timing)

Late Start / Early End: 0,062 1

Loop: ✓

Identifier (optional):

Figure 4: Four exemplary editor dialogs in MPM Toolbox. These give detailed access to all attributes of the performance instructions or to the parameters of the underlying mathematical models. These individual customizations allow for differentiated analyses and descriptions of how certain instructions were realized by a performer.

5 Things to Come and Final Remarks

The development of the MPM Toolbox continues. There are many major and minor enhancements and convenience features that will be added in future updates. Beyond the mere listening analysis, further tools for audio analysis, possibly interacting with existing tools such as the Sonic Visualiser [7, 8], will be added throughout 2021 and 2022. We also think about a suggestion to incorporate a tool for analyzing MIDI-recordings of performances. The listening analysis tool (parallel audio and rendering playback) will be expanded, too. It should be possible to play the generated performances with custom soundfonts and to export them as MIDI and audio for further treatment, for instance in music production software / Digital Audio Workstations. We also want to provide the MPM Toolbox with documentation as well as tutorial (videos) and best practice tips.

Furthermore, we plan to add further sample encodings to the MPM repository. Currently, they do not include MEI-encoded music such as those from the MEI sample encodings collection. We are open to suggestions and contributions from the community.

We offer practical workshops to provide a more interactive introduction to MPM, its concepts, productive usage and tools. A first such workshop was already held at Edirom Summer School 2020 in Paderborn, Germany. While that event was mainly focused on the MPM syntax and working in the XML editor and *meico*, future workshops will build upon MPM Toolbox and its more efficient workflow.

Such workshops and user feedback will help us to further develop and improve MPM and its tools. Anyone interested in actively participating and contributing to this process is welcome. The repository includes a contribution guide and a Wiki where we collect suggestions for future additions to MPM's set of performance features.

Acknowledgments

The author likes to express thanks to Tilo Hähnel for the collaboration in the mathematical model development, Benjamin W. Bohl for his contributions to MPM's conception, Peter Stadler for his ODD support and Simon Waloschek for developing the `doc_generator`. The development of MPM and its tools is kindly funded by the Fritz Thyssen Foundation.

Works Cited

[1] Berndt, Axel. "Formalizing Expressive Music Performance Phenomena" in *Works in Audio and Music Technology*, ed. Axel Berndt. Dresden: TUDpress, 2015, 7–128.

[2] Berndt, Axel, and Benjamin W. Bohl. "XML Music Performance Description: Reflections on and Future Developments of the Music Performance Markup Format" in *Music Encoding Conference Proceedings* (MEC 2016), 91–94.

[3] Berndt, Axel, and Benjamin W. Bohl. "Music Performance Markup: Formale Beschreibung musikalischer Interpretationen" in *editio* 32, no. 1 (2018), 185–204, https://doi.org/10.1515/editio-2018-0012.

[4] Berndt, Axel, and Tilo Hähnel. "Studying Music Performance and Perception via Interaction" in *Works in Audio and Music Technology*, ed. Axel Berndt. Dresden: TUDpress, 2015, 129–153.

[5] Berndt, Axel, Simon Waloschek, and Aristotelis Hadjakos. "Meico: A Converter Framework for Bridging the Gap between Digital Music Editions and its Applications" in *Proceedings of the Audio Mostly 2018 on Sound in Immersion and Emotion* (AM 2018), 1–7, https://doi.org/10.1145/3243274.3243282.

[6] Cancino-Chacón, Carlos E., Maarten Grachten, Werner Goebl, and Gerhard Widmer. "Computational Models of Expressive Music Performance: A Comprehensive and Critical Review" *Frontiers in Digital Humanities* 5 (2018), https://doi.org/10.3389/fdigh.2018.00025.

[7] Cannam, Chris, Christian Landone, and Mark B. Sandler. "Sonic Visualiser: An Open Source Application for Viewing, Analysing, and Annotating Music Audio Files" in *Proceedings of the ACM Multimedia 2010 International Conference* (MM 2010), 1467–1468, https://doi.org/10.1145/1873951.1874248.

[8] Cannam, Chris, Christian Landone, Mark B. Sandler, and Juan P. Bello. "The Sonic Visualiser: A Visualisation Platform for Semantic Descriptors from Musical Signals" in *Proceedings of the 7th International Society for Music Information Retrieval Conference* (ISMIR 2006), 324–327, https://doi.org/10.5281/zenodo.1416388.

[9] Crandell, Adam. "'MerMEId: Metadata Editor and Repository for MEI Data' by The National Library, Danish Centre for Music Publication (review)" *Notes* 71, no. 3 (2015), 543–544, https://doi.org/10.1353/not.2015.0037.

[10] Geertinger, Axel T., and Sigfrid Lundberg. "MerMEId: Creating Thematic Catalogues Using MEI Metadata" in *Music Encoding Conference Proceedings* (MEC 2013), 122–126.

[11] Mexin, Yevgen, Aristotelis Hadjakos, Axel Berndt, Simon Waloschek, Anastasia Wawilow, and Gerd Szwillus. "Tools for Annotating Musical Measures in Digital Music Editions" in *Proceedings of the 14th Sound and Music Computing Conference* (SMC 2017), 279–286, http://smc2017.aalto.fi/media/materials/proceedings/SMC17_p279.pdf.