Transforms: The case for Invariant Unique Identifiers in MNX

I propose a mechanism for describing generic transformations or **transforms** to an MNX score. I will then show that this powerful mechanism requires a special type of unique identifier to be optionally applied to elements, and the requirements for this special identifier cannot be fulfilled by the existing XML ID.

I will start with an example which we are all familar with as an application for transforms

Annotation of music for performance

Any musician will say that it is essential to be able to freely annotate music for performance, whether to add dynamics, articulations and text or to correct errors in the score

The current situation

In the current paperful world each musician makes their own annotations in pencil. The director/conductor reads out all his/her annotations and every member laboriously adds these in their copy. Also the section leader will usually have some annotations to add which need to be copied into all copies for that part. There might be other levels of sharing too. If somebody is absent for a rehearsal then they need to borrow somebody's copy to add the missing annotations.

Problems.

- Laborious and error-prone
- When copying annotations from another member there is no easy way of telling which were global and which were personal, or if there are any missing or wrong, or carried over from a previous performance.
- If the composer/arranger makes a new edition of the score after rehearsals have started all the annotations must be gone through and added if necessary on the new score by all members

Annotation in the paperless world - a 'dream' scenario

- There is a single MNX score *shared by all members*. This is the original score without annotations.
- Annotations are merely descriptions of score changes to be displayed in a score viewer.
- There is a group-wide set of annotations managed by the director/conductor.
- There is a set of annotations for each part, managed by each section leader.
- There is a personal set of annotations for each member
- These sets of annotations are applied sequentially and automatically, *and non-destructively* to the score in a viewer which shows the score with annotations applicable to a particular member
- The identity of the author and date/time of each annotation can be revealed, for example, by displaying them in different colours
- Any annotation conflicts are highlighted

- The original score remains editable as far as possible by the composer/arranger without breaking the existing annotations, and places where they are broken is unambiguously detected

Transforms - A way to achieve the 'dream' scenario

Imagine that you could describe a small change to a score in an unambiguous way. This description is called a **transform**.

Now you wrap a set of transforms and store them separately from the score together with any convenient metadata such as author and date/time (for instance in an XML or JSON file).

Now imagine a viewer application which takes the unmodified source document and a set of transforms and allows you to visualise the effect of applying these transforms, perhaps colouring ones by different authors, or showing the transforms in a time-line

By storing a score and separate transforms we see that the original document is always available for comparison. We can see who made what changes, and we can reverse (undo/redo) any changes.

Let us assume that many changes such as insertion, deletion or modification of a single element can be simply described by transforms, but not *all possible* changes to a score can be conveniently or economically described, such as addition of a new part or insertion of many bars. For this reason it is important to allow the source document to be modified and still make a best effort to ensure that any existing transforms remain valid, and if any fail we ensure that the failure can be correctly detected, described and ideally located to a particular measure.

[Aside. This seems like a description of a scriptable score editor, but we are going further by allowing packaged sets of transforms which can be independently applied. We are defining the transforms in a way that will allow them to be standardised, and we are storing metadata in the transforms]

Now it should be clear that transforms can be used for the annotations described in the 'dream' scenario.

Transform implementation

The transform will at least need to describe:

1. How to **insert**, **remove** or **modify** elements.
2. How to **move** certain elements such as wedge ends
3. The unambiguous identity of an existing element of the MNX score.
4. An unambiguous position in the score to insert a new element. In all cases this will be relative to existing elements eg 'on note x' or 'at the start of bar y'

The transform format could be eg XML or JSON, and it could perhaps become an addendum to the MNX standard.

Element identification

An essential part of this implementation is a reliable means of linking the transform descriptions unambiguously with elements in the MNX score, while allowing the original score to be modified directly by editing the MNX file, or indirectly by application of previous transforms

- By index

We could refer to the element by part index, measure index, and index of element type in the measure. eg

```
<insert at="part1-measure34-note23"><dynamic type="sfz"/></insert>
```

Problems.

```
* If the original score is modified and a measure or part or note is inserted or
deleted before this then this transform will end up in the wrong place, and the
error will not be detectable.
```

- Using the existing XML ID field

All MNX elements can have an optional XML ID field. This is defined to be unique and so it could be used for for identifying the element.

Problems.

```
* IMPORTANT If we have a score containing XML IDs on some elements created by some
haphazard rule, and this score requires these references for internal or external
consistency, then we have no way of using these IDs for our scheme, because there
is no convenient way to add new unique IDs to this existing set, not knowing which
'haphazard rule' was used to create them. Unless we have a fixed rule for
generating new unique IDs we cannot add any to a score which contains existing IDs
which need to be preserved.
```

- Using an Invariant Unique Identifier (**IUID**)

When the score is created or prepared for use with transforms then all addressable elements (TBD) including parts and measures are given a unique **IUID** and a Predetermined Generation Rule (**PGR**) is used for creating these IUIDs. If the score is subsequently modified *or transformed* then a) any elements which are modified *must retain the same IUID*, and b) any new elements are given a new IUID using the PGR.

eg

```
<mnx-transforms source="iuid.be478726" by="jdoe" email="jdoe@doe.co">
```

```
<insert id="iuid.34560.23def" at="iuid.23B456.23def"><dynamic type="sfz"/></insert>
```

```
<insert id="iuid.34789.23def" from="iuid.C234.23def" to="iuid.C239.23def"><wedge
type="crescendo"/></insert>

<modify at="iuid.12B.23def" from="iuid.122.23def" to="iuid.134.23def"><wedge></modify>

<modify at="iuid.6573.23def"><note pitch="C4"/></modify>

<delete at="iuid.DA758.23def"><note/></delete>

</mnx-transforms>
```

Notes

```
* The uid is of the form `iuid.<a>.<b>` where a is a unique identifier of the
element and b is the identifier of the part/measure containing the element. This
allows specific error reporting including the part and measure number where the
transform fails (unless these have been deleted from the score).

* The type of the element could be included in the IUID eg "iuid.note.2345.abcd"
for improved readability

* The score itself should have a uid to ensure that the transforms specify
unambiguously which score they refer to

* A IUID must be permitted *in addition to* the existing XML ID on each element as
explained above, and the standard must somehow allow it to be much more narrowly
defined than the normal XML ID. Since it seems probable that others will come
forward with other incompatible requirements for IDs perhaps it is possible to
allow *unlimited* XML IDs on any element, each with a special prefix (eg "iuid.")
and a separate registry of prefixes and the defined usage.
```

Predetermined Generation Rule for IUIDs (PGR)

A couple of possibilities exist for generating IUIDs

An incrementing numeric value.

Notes

```
* The next value needs to be stored in the MNX file. We cannot rely on automatic
detection of this because an element with the latest value may be deleted, and this
could result in reuse of that value.

* This cannot define a IUID for the whole score

* This cannot be used for uids attached to new elements created by transforms as
these can be independently created from the original read-only score and so there
is no way to disambiguate them.
```

A (ms or us) timestamp.

Notes

```
* We assume that 2 identical timestamps cannot be created (but this possibility
could be detected and avoided).

* This can be applied to the whole score and to uids attached to new elements
created by transforms.

* A 32-bit ms timestamp wraps every 49 days. The collision probability seems
acceptably low
```

Other examples of possible uses of transforms.

There are many other situations in which transforms as described above would be advantageous. Some examples are considered here. But I'm sure others will spring to mind

Translation of lyrics

A set of of lyrics for a score in any language can be stored as a set of transforms, so we can store the lyrics separately from the score, and still be able to edit the original score without lyrics

Extraction of parts

Parts could be extracted by application of transforms, including separation of voices on a stave into separate parts, while retaining score editability.

Transposition of parts

A set of transforms to transpose a part can be applied without modifying the original score.

Summary

I have described a mechanism called **transforms** for capturing changes to an MNX score without changing the score itself, with particular application to handling annotations with multiple authors in a group of music performers such as an orchestra or choir.

I have shown that these transforms require a type of Unique Identifier, known as IUID for elements of the score which is incompatible with the existing XML ID, so is an *additional* requirement for MNX.

Responses to questions

Implementation burden

@joeberkovitz IUIDs should be optional, and can be ignored by read-only or write-only applications which don't use them, and can be stripped by read-write applications which don't support them. So they are not necessarily a burden.

Producers and Consumers of MNX

@joeberkovitz In the example of the automatic creation of a choral arrangement of a piano score. The application program could generate a new MNX score with the choral arrangement, or it would be possible for it to generate a transforms file to be applied to the source score. In neither case would it necessarily be required to *write* IUIDs unless the target needs to support transforms