Chapter 1

Transforming musical notations for universal access to performance and composition

Silas S. Brown and Peter Robinson

Abstract

Some disabilities can be assisted by using alternative notations and alternative input methods. This paper describes a system for transforming between different musical notations, which can be customised to an individual's requirements, hence supporting many unusual needs that did not specifically have to be accounted for in the initial design. The customisation is brief, which encourages experimentation because new ideas can be explored more quickly.

Keywords: music, transformation, low vision, blind, typing difficulties, markup languages, notations, braille.

1.1 Introduction

Musical notations are coded instructions for musicians to perform music. They represent co-ordinated events in a stream of time. Internationally, several written notations are in widespread use, such as Western staff notation, Chinese *Jianpu* notation, *sol-fa*, many instrument-specific notations such as guitar tablature and Japanese *koto* notation, and Braille music for the blind, which has numerous different versions across the world. See Figure 4 for examples.

It is often possible to transcribe a piece of music from one notation into another in order to make it accessible to a greater number of musicians. Software exists to effect such transcriptions, such as the music-to-Braille projects MFB [Langolff et al., 2000] and Goodfeel [McCann, 1997]. Such software works by obtaining a semantic (symbol-based) computer representation of the source notation, and algorithmically transforming this into a representation of the desired notation which is then realised by a suitable output device. However, existing systems are limited to dealing with a few specific notations; if a highly unusual or customised notation is required, this will often call for specialist programming or manual intervention.

Customised notations. All of these notations can be customised for training purposes, or for particular tasks such as rapid overview (this is particularly useful in Braille music). Printed notations can further be customised to address print disabilities such as low vision or dyslexia, by using a modified set of symbols or a modified layout-for example, a person with tunnel vision benefits if musical directions are moved closer to the notes to which they apply, especially if the print is large. Ideally, software should support such customisation in an open-ended way, to facilitate needs that the designer did not originally anticipate.

Input. Musical composition and input presents another challenge. It is possible to enter music into a computer by playing it on an electronic keyboard or other instrument, but the results do not always match the user's intentions due to quantisation errors, and many disabled users are excluded because of the dexterity required. A music notation editor is frequently more appropriate. However, notations that are optimal for reading are not necessarily optimal for writing or editing-conceptual similarity between reading and writing is useful, but this can be overshadowed by a disabled person's accessibility needs. For example, someone with typing difficulties might prefer a terse input notation even if it means more training. People with print disabilities often find direct manipulation music publishing systems such as *Sibelius* [Finn and Finn, 2001] difficult to use, and prefer character-based music languages that can be written in any text editor, including any specialist or customised text editing environments they may have.

Several different character-based notations are in use by music typesetting systems and online repositories, and it is possible to design new ones. The ideal notation will vary with the style of music and the individual's method of composing or editing, as well as their disability and input device. For this reason it is useful to support flexibility in input (“write only”) as well as output (“read only”) notations when supporting musical activities with software.

1.2 Related work

Besides the specialist Braille transcription software that has already been mentioned, and numerous music typesetting tools and other software that is capable of dealing with more than one notation (e.g. recent versions of *Sibelius* can convert between Western staff and guitar tablature), there are also some efforts to generalise the problem of transcribing between musical notations so that new or rarely-used notations can be supported as needed. In an earlier project [Brown, 2000], the first author represented musical scores as databases with each record corresponding to an event in the music; a special reporting language was used to generate various forms of Braille as well as data for music typesetting software. The main limitations were the difficulty of supporting new input formats and the verbosity of the languages used.

The problem can also be addressed by considering musical data as an example of general structured data, and utilising a generalised transformation framework such as XSLT, the XML transformation system [W3C, 1999] or TXL [Cordy et al., 1988] to effect the transcriptions. These languages are verbose, so customising them involves considerable work, particularly for print-disabled people—those with blindness, low vision, dyslexia or another impairment that restricts the use of print. There is no built-in support for multi-dimensional data, but music is inherently multi-dimensional, and forcing it into a hierarchical structure introduces arbitrary assumptions about its processing order and introduces difficulties when there are exceptions to the structure [Castan et al., 2000]. This increases verbosity.

1.3 Implementation

The authors' transformation framework 4DML [Brown and Robinson, 2002] has been used as the basis of a transformation system for musical notations. The 4DML framework consists of four main components:

1. An internal representation of structured data with multi-dimensional structures,
2. *Matrix markup language* (MML), a generalised markup language designed to facilitate the input of multi-dimensional data,
3. A transformation tool that takes XML or MML as input, uses the above internal representation, and produces output in any text-based language by following a *model* of the desired structure,
4. *Compact model language* (CML) for representing the model (models may also be represented using XML).

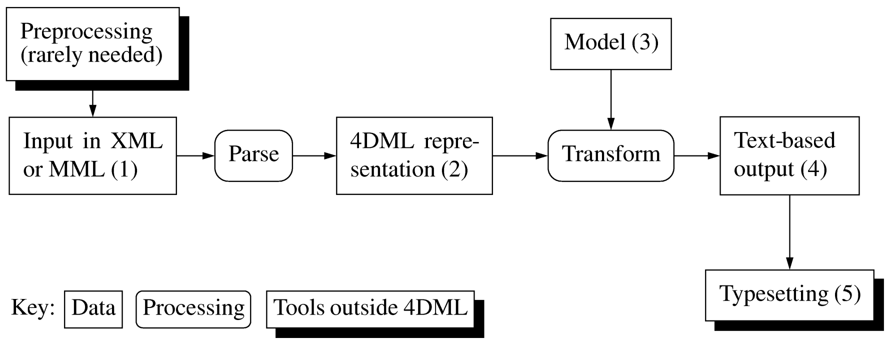


Figure 1: Overview of the 4DML transformation framework

As shown in Figure 1, data in XML or MML (1) is first converted into 4DML (2)—a process which needs no external information as XML and MML are both self-describing formats—and then transformed into any text-based output language (4) under the direction of a model (3). The entire process may be surrounded by other transformations, such as the passing of the output through a typesetting system (5).

Matrix markup language (MML)

It can be cumbersome to hand-code multi-dimensional data in a hierarchical markup language like XML, since the markup is very verbose and repetitive. For example, in coding the lyrics of a song, one might have to enclose each syllable in a `<SYLLABLE>...</SYLLABLE>` pair, whereas it would be easier to define a separator (for example, hyphen) to stand for "next syllable" (other separators can advance the word, verse number or translation). In the general case, one can construct a parser for an arbitrary input language, but this can be a significant amount of effort for an end-user. There is scope for a markup language that provides for some simple re-definitions (such as "whitespace means next syllable") while not being as complex as a complete parser generator tool.

Matrix Markup Language (MML) is a text-based language that can represent structure in several ways. It consists of a mixture of directives and data. For example, the `!block` directive begins a matrix-like block of data that starts with directives such as

```
    Have paragraph ¹ newline ² whitespace ³ - ⁴
    as system ¹ verse ² word ³ syllable ⁴
```

which defines how the text is to be parsed—paragraphs represent "systems", lines represent "verses", whitespace separates "words" and hyphens separate "syllables" (the numbers are for illustrative purposes only and are not part of MML). The `have` directive takes a list of input tokens, the word `as` and then a list of the corresponding components in the structure being described. These tokens are then referred to by the model (see below) during output. If desired, the lists may be built up from several `have...as` directives. Punctuation and other arbitrary strings may also be defined as separators, and there are facilities for representing overlapping sets of independent markup via multiple `have` directives separated by the word `also`. The data is also checked for consistency as it is processed, so any errors such as missing data are reported.

A "system" is a unit of physical layout, and the layout will probably change with the transformation. Nevertheless, representing the original layout (if any) often facilitates error correction and cross-referencing.

Compact model language (CML)

4DML uses a "model" to outline the structure of the desired output, which facilitates adjusting the output notation as needed. The data is automatically rearranged into the structure given by the model—the model guides a complex sorting operation, and also specifies any extra typesetting instructions in the language of whatever typesetting system is to be used. This means there is very weak coupling between the design of the input and that of the output; each can be customised independently of the other.

CML is a text-based language designed to facilitate the brief coding of models. It consists of literal text to be output directly, interspersed with code that generates output from the 4DML representation. In practice, most models have a repetitive structure; they express such things as “for each song, for each verse, for each syllable,...” which is expressed as `song/verse/syllable`. CML also has other operators and can represent any hierarchical document, but its syntax is designed for representing typical models concisely.

1.4 Evaluation

People with print disabilities should be able to program this transformation system by themselves, to assist with their musical work. To demonstrate this, an individual with low vision has used the system for the tasks described in this section. We hope to find other interested individuals in the future.

```
!block hand
have whitespace , character . as bar beat note string

r,r,r,Bd e,dB,e,dB A,B,D,EF GB,AG,E,ED E,r,r,Bd e,dB,e,dB
A,B,D,EE GB,AG,E,ED E,r,r,EF G,GB,d,cB A,A,B,GA B,g,ed,Bd
e,r,r,Bd e,dB,e,dB A,B,D,EF GB,AG,E,ED E,r,r,Bd etc
!endblock

!block hand
have whitespace , / character as bar beat note string

r,r,r,r EG,r,EA,r D,r,r,r E,r,r,r r,r,E,r EG,r,EA,r
D,r,r,r E,r,r,r r,r,r,r E,r,EGB,r DE,r,EG,r EG,r,EG,r
EG,r,EG,r EG,r,EG,r E,r,r,r E,r,r,r B/G,EB/G,EGB,r etc
!endblock
```

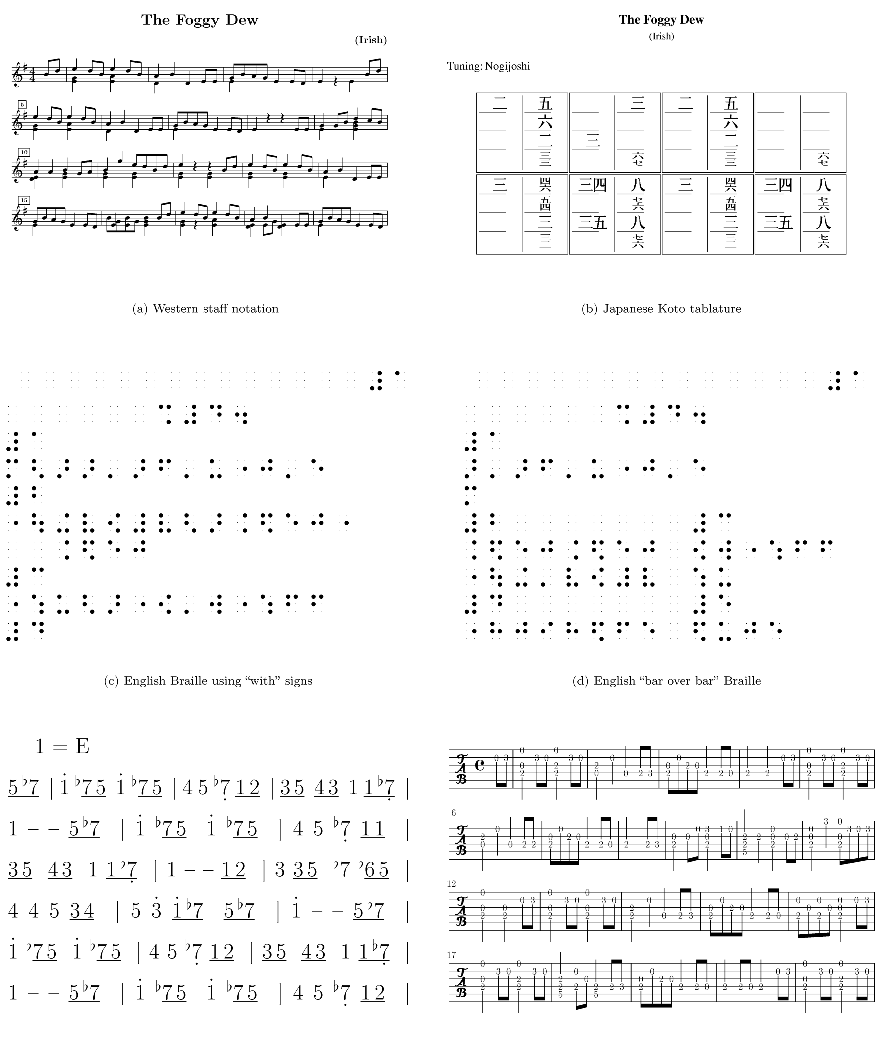
Figure 2: Input in MML using a syntax designed by the user.

The individual arranged some music for the Japanese Koto and encoded in a text editor using a notation that was invented for the purpose and appropriate to the music. This was achieved by means of MML and is shown in Figure 2—notice that the notation changes half way through the figure. Koto tablature was then produced by using 4DML to drive the layout engine Lout [Kingston, 1993], which is a general document preparation system that takes a description of page layout and typesets it as PostScript or PDF. The Japanese characters were implemented as images, as Lout does not support Unicode. The model is shown in Figure 3, which is less “cluttered” when the comments are removed. It consists of a translation table of notes to symbols, and then nested loops over bars, hands, beats and notes—notice that the nesting order is different from that of the input and its transposition is automatic. Other 4DML models allowed the same music to be sent to Western music typesetting systems, Braille printers using multiple versions of

Braille, and other formats as shown in Figure 4. It is possible to implement models for new types of output as needed.

```
@Include{koto.setup} @Doc @Text @Begin
@Display clines @Break { @Heading 18p @Font {The Foggy Dew}
(Irish) }    (alternatively, the titles could be kept in the MML)
Tuning: Nogijoshi @PP    (Now follows a preparatory translation table)
[[cml chord export-code /
 string begin="@OneCol {" end="}" / (    (for each string,)
   note value=D/"@IncludeGraphic kanji2.ps",
   note value=E/"@IncludeGraphic kanji3.ps",
   note value=F/"@IncludeGraphic kanji3.ps",    (re-writing F as E)
. . .  (more notes follow)
)]]    (End of translation table; start of layout proper)
@RightDisplay -90d @Rotate  (because Koto is read in columns from right to left)
21c @Wide {ragged 1.5vx}@Break {
[[cml bar group-size=20 group="} @RightDisplay -90d @Rotate 24c @Wide {ragged
   1.5vx}@Break {" /(    (for each bar, with new page every 20 bars)
  ]]# Bar [[cml bar count]]    (comment the bar number—useful in debugging)
3.9c @Wide @Box {    (each bar is a 3.9cm-wide box before rotating downward)
      [[cml hand between="//0.1c @FullWidthRule //0.1c"/(]]  (for each hand)
         90d @Rotate 2c @Wide {    (each hand is a 2cm box rotated left)
            [[cml beat between="//0.1c 1.3c @Wide @LocalWidthRule //0.1c"/(]]
               @Centre {    (two cases—1 or 2 notes in the beat
                     —handle differently because it affects scaling)
                  [[cml note total=1 no-strip call=chord]]    (call the
                  [[cml note total=2 no-strip call=chord      translation table)
                        begin="{0.8 0.5} @Scale " between=" // "]]
               }
            [[cml )]]    (end of code for each beat)
               //0.1c    (this Lout code means 0.1cm vertical gap)
         }
      [[cml )]]    (end of code for each hand)
   }
[[cml )]]    (end of code for each bar)
} @End @Text
```

Figure 3: 4DML model as CML embedded in Lout. The literal text is shown in roman type, the code in **bold sans serif** type, and the comments in *italics* are added here for explanatory purposes only and are not part of CML.



(a) Western staff notation

(b) Japanese Koto tablature

(c) English Braille using "with" signs

(d) English "bar over bar" Braille

(e) Chinese Jianpu

(f) Guitar tablature

Figure 4: Output in various notations

```
begin music
begin part

!block pitch
have whitespace character as bar note

r rrrd ddddfca aarrd ddddfca aadce gfcdfeca gfcddfeeg
gfbagffg dcfffg feaabb ddcc bbaa gfgaabaadfa ddcbbdf
baggbdgfe egfee gfeebdrad daffaaaad drdcbbdf baggbdgfe egfee
etc
!endblock

!block duration
have whitespace character as bar note

0 2488 8881144 28888
8881144 28114 11481148
114111148 11481148 114848
114848 4882 4882 88888883333
etc
!endblock
```

Figure 5: Part of a piece in an aspect-oriented encoding. Other aspects (not shown) are octaves, enharmonics, dots, tuplets, phrasing, articulation, ornaments, dynamics, text, time and key changes and typographic adjustments.

```
bar between="&#10;&#10;" / part between="&#10;" (
  uptext begin="U: " end="&#10;",
  keychange/posn number=1 after=" ",
  note between=" " / (
    tie/posn number=1 end=" ",
    pitch, accidental, dot, duration,
    octave, shift, tuplet, articulation,
    tie/posn start-at=2 begin=" ",
    dynamics begin=" "
   ),
  keychange/posn number=2 before=" "
  )
```

Figure 6: CML code to interleave Figure 5 into the format of M-Tx (a music typesetter)

Another experiment involved the use of “aspect-oriented” music encodings, imitation of aspect-oriented programming [Elrad et al., 2001]. Different aspects of the music, such as note letters, octaves, durations, enharmonics, ornaments, etc, were coded on separate passes through the score (Figure 5), and the model interleaved them when producing the typesetting instructions (Figure 6). This facilitated the transcription of already-written music because the user need consider only one aspect at a time, avoiding the need to switch rapidly between many different features of a complex input language; the user was able to encode a complex score which he had been unwilling to attempt using conventional methods.

Aspect-oriented encoding also proved beneficial for original composition, the different aspects of the composition being added at different times. In this case the "aspects" were not always aspects of musical notation; they also included aspects of the compositional framework defined by the user (such as "arpeggio type" and "time distortion") which were converted into musical notation by the user's model.

The system was also used to typeset a large number of Chinese songs in various formats including an invented *sol-fa* like notation; in this case most of the work was in arranging for the model to produce and typeset pronunciation aids in an accessible form, and this is discussed elsewhere [Brown and Robinson, 2003].

1.5 Conclusion

A transformation system for musical notations has been constructed using the 4DML framework. This allows people with unusual accessibility needs to customise both the presentation of musical notations and the means of inputting them to their individual requirements, and allows music to be transformed between different presentations for different people. This should increase the accessibility of music as an educational subject, a vocation and an avocation. The aspect-oriented method of encoding music that was introduced also holds potential for music publishers and repositories, because it could be used to divide encoding skills among several people.

4DML's primary contribution is the brief-but-readable nature of its models, which aids in the rapid prototyping of transformations. It encourages a consideration of the notations themselves rather than the algorithmic methods for their transformation, hence allowing new notations to be experimented with more easily. In future it could be used to assist in experimenting with completely new ways of presenting music, such as via sign language, pictorially, or in tactile forms other than Braille (some physical conditions preclude good Braille reading but allow other tactile forms of communication). This would make music accessible to an even greater number of individuals.

4DML has also been used for the transformation of other notations; a forthcoming thesis will demonstrate its applicability to mathematics, diagrams, websites, experimental data and personal notes. Virtually all information-society applications involve notations, and the transformation of these between different versions is a component part of universal access, since it can help to cater for special needs and for differing tasks and environments. Tools that support the programming of such transformations, such as 4DML, can make it easier to create new notations on demand and to implement universal design.

1.6 References

Brown, S.S. (2000). An extensible system for conversion of musical-notation data to braille musical notation. *Computing in Musicology*, 12:45-74. The original was an undergraduate dissertation entitled “A Representation and Conversion System for Musical Notation”, Cambridge University Computer Laboratory, 2000.

Brown, S.S. and Robinson, P. (2002). Automatically rearranging structured data for customised special-needs presentations. In Keates, S., Clarkson, P. J., Langdon, P., and Robinson, P., editors, *Universal Access and Assistive Technology: proceedings of the Cambridge Workshop on UA and AT*, pages 109-118.

Brown, S.S. and Robinson, P. (2003). Addressing print disabilities in adult foreign-language acquisition. In Stephanidis, C., editor, *Proceedings of the 10th International Conference on Human-Computer Interaction (HCII 2003), Vol.4: Universal Access in HCI*, pages 38-42. Lawrence Erlbaum Associates.

Castan, G., Good, M., and Roland, P. (2000). Extensible markup language (XML) for music applications: An introduction. *Computing in Musicology*, 12:95-102.

Cordy, J.R., Halpern, C.D., and Promislow, E. (1988). TXL: A rapid prototyping system for programming language dialects. In *Proceedings of The International Conference of Computer Languages*, pages 280-285, Miami, FL.

Elrad, T., Filman, R.E., and Bader, A. (2001). Aspect-oriented programming: Introduction. *Communications of the ACM*, 44(10):29-32.

Finn, B. and Finn, J. (2001). *Sibelius: The Music Notation Software*. Sibelius Software Ltd, Cambridge, `http://www.sibelius-software.com/`.

Kingston, J.H. (1993). The design and implementation of the Lout document formatting language. *Software—Practice and Experience*, 23:1001-1041.

Langolff, D., Jessel, N., and Levy, D. (2000). MFB (music for the blind): A software able to transcribe and create musical scores into braille and to be used by blind persons. In *Proceedings of the 6th ERCIM Workshop on “User Interfaces for All”*, number 17 in Short Papers, page 6. ERCIM.

McCann, B. (1997). *GOODFEEL Braille Music Translator*. Dancing Dots Braille Music Technology, `http://www.dancingdots.com/`.

World Wide Web Consortium (1999). *XSL Transformations (XSLT) Version 1.0, W3C Recommendation*. `http://www.w3.org/TR/1999/REC-xslt-19991116`.