

From NIFF to Musical Braille

Didier Langolff, Pierre Gradit, Nadine Jessel, Monique Truquet (TOBIA)
Institut de Recherche en Informatique de Toulouse
Université Paul Sabatier
E-mail : langolff@irit.fr

1. Introduction

Our purpose is to make faster the transcription of printed score into Braille. After a study of several Musical notation "standards", we decided to develop a software which will make the transcription from NIFF (Notation Interchange File Format) to Musical Braille. In the following paragraph we will present some key concepts of the NIFF notation, NIFF standard is a computer-oriented description of printed scores. Then we will describe the transcription software called "Editor" and some specific features.

The two used languages (NIFF and Musical Braille), are probably near as their main constraint is the same: the writing of music scores being made with a single sequence of characters. The basic step of our method was to find a way of editing musical Braille near the description of a score in a NIFF notation. So, the first step was to structure the Braille characters sequence with a tree-structure to develop then a complete editor of this structure.

2. NIFF notation

NIFF is a file format designed to allow the interchange of music notation data between and among music notation editing and publishing programs and music scanning programs. Its design is a result of combined input from many commercial music software developers, music publishers, and experienced music software users¹.

The lack of an accepted standard format for music notation has been during several years a source of great frustration for computer musicians, engravers and publishers. Numerous attempts have been made in the past to create a standard format. The effort resulting in NIFF is interesting for us for different reasons:

- NIFF files can be extracted from printed scores with different processes. We opted for the scanning program MIDISCAN which proposes a NIFF automatic


¹ NIFF 6.A, Notation Interchange File Format, 1995. <http://mistral.ere.umontreal.ca/~belkina/NIFF.doc.html>

translation, the other way being the result (the NIFF output) of editing programs. One of the main feature of NIFF is that the description of a score in NIFF format is purely graphic. For example, a note is described not by its value (A, B, C...) but by its position on the staff.

NIFF basic structure is a tree (See Appendix 1). A NIFF file is divided into two sections: the Setup Section, containing general information relative to the whole score, and the Data Section, containing the music symbols and layout information. The tree structure is inherent of the NIFF file structure, but also of the use of the anchor concept. We will describe in the following paragraphs these key features of NIFF format.

2.1 Lists and "chunks"

Lists and "chunks" are the basic components of a NIFF file. All the lists and "chunks" available to create a NIFF file are presented in the NIFF specifications¹.

Notehead "chunk" specification		Printed score sample	Interpreted NIFF code
Notehead	Chunk		note: [6] (4, 5*, 0, 1, 0, 16**)
Shape	Byte 1 = double whole 2 = whole 3 = half-whole 4 = quarter-whole ... 14 = open triangle		
Staff step	SIGNEDBYTE		
Duration	RATIONAL		
Default anchor:	Time-slice		
Default reference point:	The center of the bounding box of the notehead		"[6]" is the size of the body. *: Position on the staff, first line has the value 0. **: Duration of the note: 1/16th
Tags:	Part ID, Voice ID, placement tags, MIDI, Performance, Grace Note, Cue Note...		

2.1.1 Data "chunk"

The "chunk" may be separated in two classes: setup "chunk" and symbol "chunk". Symbol "chunks" describe the music score (See example above). Setup "chunks" define the context of the description.

A "chunk" is the basic structure of a NIFF file (which is an extension of RIFF (Resource Interchange File Format)). A "chunk" is composed of a header and a body. The header is written over 8 bytes. Four bytes give the name of the "chunk", and four give the size of the body.

In the following table we present a NIFF file sample concerning the notehead "chunk".

2.1.2 Lists

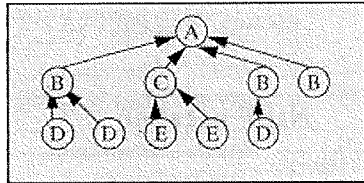
A list is a type of "chunk" which can contain nested "chunks". Its code of four characters is "LIST". The type of the list consists of a "four characters code" following the size field, which is followed by a series of chunks or other lists. Pages, systems, and staves are all represented by lists.

Staff "list" specification		NIFF notation
Staff	LIST	LIST: [2382] syst
Location:	System list	syhd: [0] ()
Required:	Staff Header chunk	LIST ¹ : [1252] ² staf ³
Optional:	Any number of the following:	Sthd: [18] (...)
	Time-Slice chunk	... ⁴
	Any music symbol chunk	..
	NIFF Font Symbol chunk	
	Custom Shape Graphic chunk	1: Four characters code (always "LIST")
	Text chunk	2: Size of field
	Line chunk	3: Type of list
		4: series of "chunks" or other lists

The "chunks" given in the body of a list "chunk" are the child of this "chunk". List "chunks" generate a tree, called reading tree, where nodes are the list "chunks", and leaves the data "chunks". The last level of the reading tree is the staff. So all symbols describing a staff are at the same level as the reading tree. But, informally, the tree structure continues deeper than the staff level. Measure, Stem, Notehead could be considered as nodes of a tree, taking into account the concept of anchor/dependant relationship.

2.1.3 The anchor/dependant relationship

In NIFF, a music symbol whose placement depends on one or more other symbols is called a "dependent" symbol, and the symbol or other "chunk" type on which its placement depends is called its "anchor." For each symbol "chunk" type, a default anchor "chunk" type is defined (except for the root: the page header "chunk"). An example of this is given in the above table concerning the representation of the notehead "chunk": you can see that the default anchor of the notehead is the "time-slice". This means that every symbol "chunk" is child of the last previous "chunk" corresponding to the anchor type. For example, let be $(A \leftarrow B, A \leftarrow C, B \leftarrow D, C \leftarrow E)$ a set of (Anchor \leftarrow Dependant) relationship. Then to the following sequence *ABDDCEEEDB*, corresponds the following tree structure:



Where:

A: could represent a time-slice (measure-start event),

B: could represent a notehead "chunk",

C: could represent a stem "chunk",

D: could represent an accidental "chunk",

E: could represent a chord "chunk".

2.2 Interest for our "Editor software"

So, there are two "tree structures" in a NIFF file:

- Explicit tree: the reading tree, which nodes are lists and data "chunks" are leaves. This tree is the NIFF file.
- Implicit tree: The final tree, in which data "chunks" may be nodes. The basic notion associated to the final tree is the anchor concept and the memory table. To build this tree, external information is needed, common to all NIFF files: the anchor/dependant relationship.

In this software, we give the notes relatively to their height in music (A,B,C...), their octave is given only if needed. We adopt this notation (different from NIFF notation) only to correspond to the Musical Braille definition.

3 The Software

3.1 Musical Braille notation for stave description

We now focus on musical Braille notation. The first official normalization of this notation takes place in the nineteenth century (Cologne Key 1888), of course without any global structure description as a regular grammar. For the transcription, we choose to use the musical Braille notation described in the New International Manual of Braille Music Notation².

But as a staff depicted in Braille is a sequence of characters, other musical objects are in fact represented by "sub-sequence". These sequences, which represent a musical object, are called frames. Frames studied in our work are regular, that is to say if a child frame starts in a father frame, it has to end before this father frame.

These structures may be represented by trees. In those trees, called frame trees, nodes are frames and leaves are the characters.

A Braille character has 6 dots³ which permit to obtain only 64 combinations. So in some cases we need several Braille characters to represent only one classical item. For example the "key" notation requires 3 Braille characters. On the contrary, a single Braille character represents a note or a rest (name and value). The names of the notes are obtained with the dots 1,2,4 and 5; the dots 3 and 6 determine the note values.

3.2 Tree concept

It is now clear that the tree structure is the main structure used in our work. As we developed our software in an object-oriented language, we defined abstract class defining trees.

The software may be seen as trees manipulation. Many instances of the tree structure are used:

- the frame tree: the internal describing language of the software, which nodes are frames and leaves are characters. The action associated to score, page and line is *void update()*, that guarantees the validation of pagination constraints (number of characters per line, hyphenation, number of lines per page). Updating is automatic.

² New International Manual of Braille Musical Notation, Bettye Krolick, World Blind Union, 1996.

³ 3 lines, 2 columns.

the score tree: the Braille display format, which nodes are score, page, line and the leaves are characters.

the reading tree: the NIFF file format, the element action is *int read()*, *read* returns the number of bytes read. This is an automatic action.

the intermediate tree: the intermediate format, the element action is *void translate()*. It gives the corresponding frame and inserts it in the context (contextual insertion).

the dictionary tree: the database organization using the standard file system, which nodes are folders and leaves are files. The element action is *void insert()*. It inserts in the context the frame described in the selected file.

3.3 Editor: set of basic tools

We now focus on tools, components of the editor (See Appendix 2). All these components are browsers of tree.

The Cursor viewer is a frame tree browser.

The Braille panel is a score tree browser.

The NIFF translator is an intermediate tree browser.

The Dictionary is a file tree browser.

Each browser has its own cursor. So at a time, only one browser and only one element of the corresponding tree are active. The state display of the software is the display of the String representation of the current element. The set of command at this instant is:

- Change active browser (Next Tool/ Prev Tool).
- Move in the current tree (Next/Prev/Outside/Inside)
- Action the current element (Action)

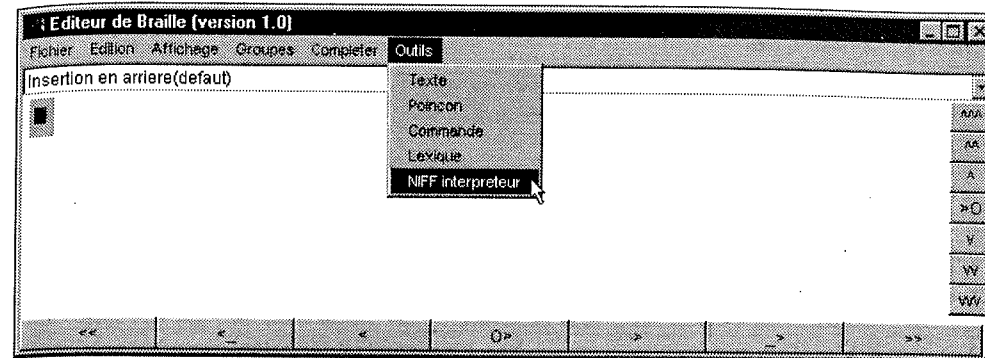
3.4 Editor: Example of use

For a better understanding, we will develop an example of transcription with the following staff:



The first step is to acquire a NIFF file of this staff, for example with MIDISCAN (as mentioned in the paragraph 2).

We set up the program and we chose to begin the transcription, so we have to activate the "NIFF interpreteur".



This program takes the NIFF file corresponding to the above staff, and gives the following tree.

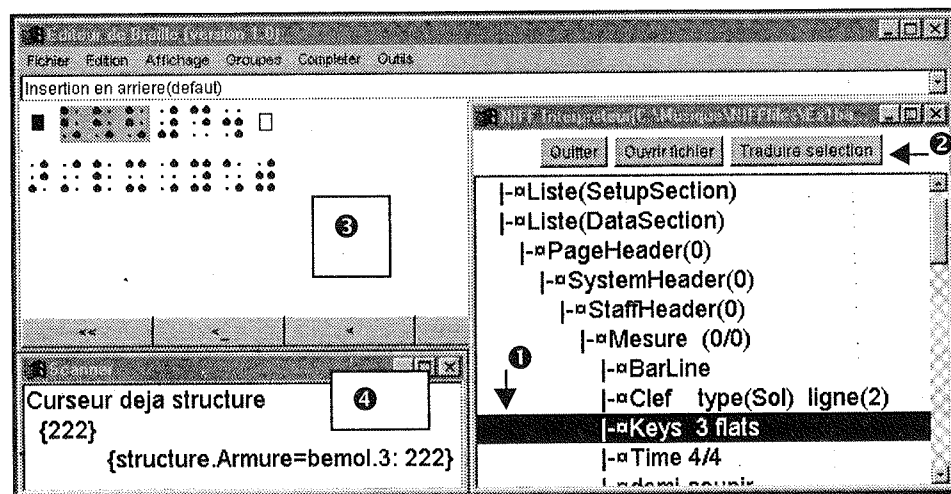
This section corresponds to the framed part of the score:

*: Intermediate tree

At the beginning, you have only the root of the tree (called "Fichier") in the windows of the NIFF translator.

You can expand the tree by double clicks on the nodes until a leaf is encountered. For example, when you activate the node fichier, the nodes Liste(SetupSection) and Liste(DataSection) appear, according to the NIFF structure (See Appendix 1).

When you are in the NIFF Translator, you can browse the file (Intermediate tree) and decide to transcribe what you want by selecting the part (❶) you have to activate the transcription button (❷):



In the above example, the user selects the key signature and activates the transcription button. The result of the operation appears in two different windows if they have been opened by the user:

- The Braille panel (❸) where we can see the current translation highlighted.
- The window called "Scanner" (❹) which permits to browse the frame tree.

In the transcription step, the user can translate the whole score, a part of a score or just one item of a score.

4 Some specific features

4.1 Contextual insertion

The contextual insertion manages automatically the insertion of symbols into frames at the right place. The Braille notation enforces some rules in the order of the appearance of musical items. For example, if the Editor has to transcribe a sequence containing a note, an

accident and a dot, according to the Braille notation, the definitive sequence will be: the accident, the note and the dot.

A basic attribute called priority is used to manage contextual insertion. The routine *int getPriority()* returns a value, which permits to insert each frame at the right place (according to the Braille notation).

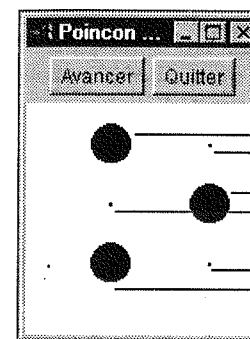
4.2 Creation of Braille Scores

The user has the possibility to create a score directly in Braille notation (without a NIFF translation). The software offers two different possibilities:

- The user knows the Braille musical notation and use indifferently the tools called "poinçon" or "dictionary".
- The user doesn't know the Braille musical notation and he has to use only the "dictionary".

4.2.1 The "Poinçon"

The "poinçon" is a tool available in the "tools menu" of the Braille panel. The user can write Braille characters via the keypad and the keys 1, 2, 4, 5, 7, 8.



Button "Avancer" allows to insert the current character and wait for the entry of a new character.

Button "Quitter" desactivates the use of the "poinçon"

Key "8" of the keypad,
Key "7" of the keypad,

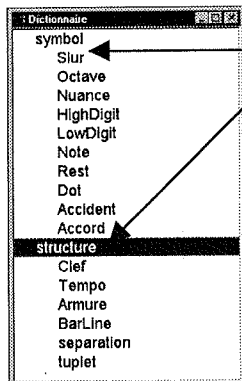
Key "5" of the keypad,
Key "4" of the keypad,

Key "2" of the keypad,
Key "1" of the keypad,

If an user does not find a Braille character or a sequence of Braille characters in the dictionary he can create them and store them in the dictionary with the help of the "poinçon".

4.2.2 The "Dictionary"

Based on the Dictionary tree (Cf. paragraph 3.2), the dictionary tool allows to an user (knowing or not Braille notation) to write a score according to the Braille notation. The principle is the following: the user has to browse the Dictionary tree, choose the element he wants to insert, and a double click allows to insert the element at the right place.



The dictionary tree has two main nodes:

Symbol and
Structure.

You have to develop the nodes as it's made in the example (on the left), and to go on with the other nodes until the needed leave is encountered.

The leaves of the Dictionary tree correspond to the Braille characters.

5 Conclusion

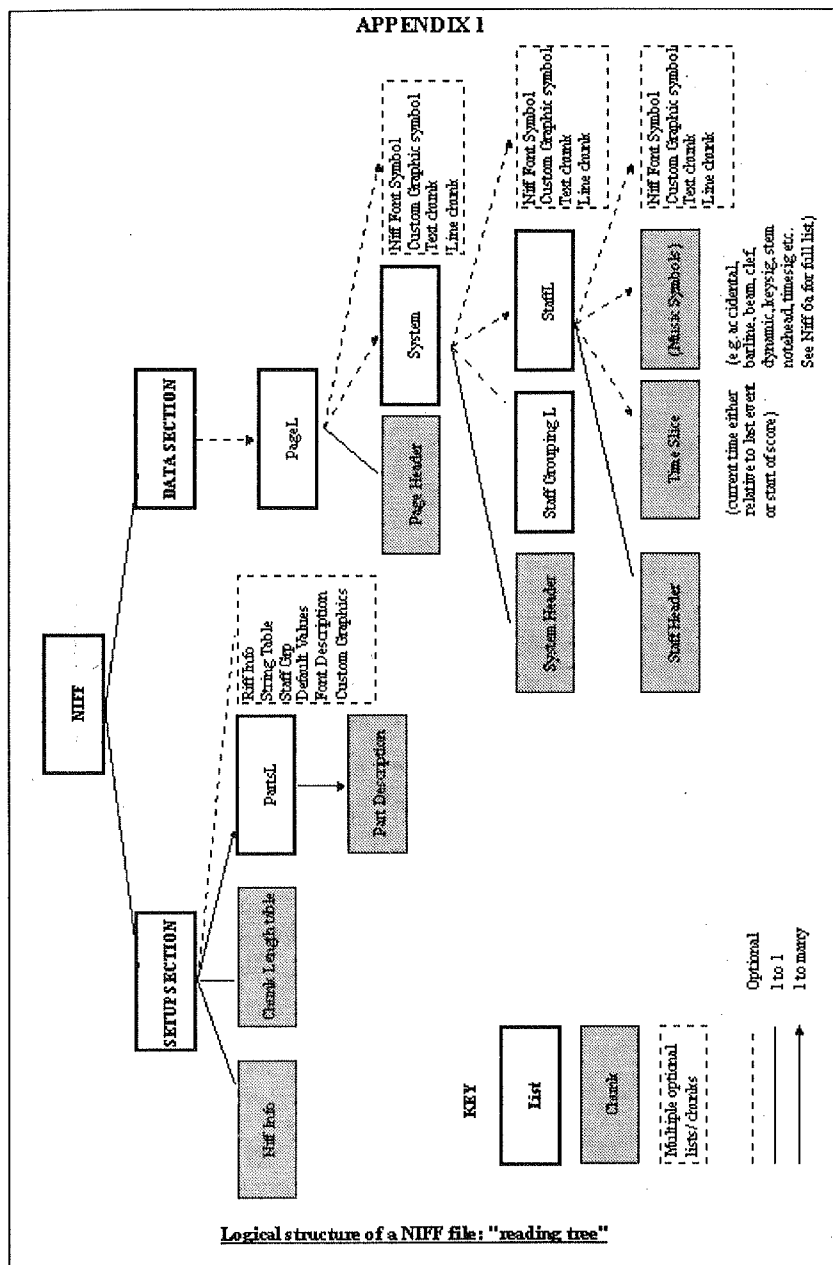
At the present time, the software is accessible by a sighted person. This person, who could know or not the music Braille notation, can transcribe a part of a score. Some parts of the Braille transcription software have to be developed in order to transcribe the whole score.

We plan to adapt this software to allow a blind person to use it with the help of a screen reader and text to speech synthesis. The use seems to be possible due to some features of the software:

- Only one browser is active at one time.
- The four keypad arrows represent the moves (left, right, outside, inside).
- The space key triggers the action associated to the element.
- Page-Up/Page-Down change the current tool.
- Some keys of the Braille sensitive line could be used too.

Another extension of this software could be used for the learning of music. In the example presented above, we saw that it's possible to browse a score and to know at any

time the meaning of each symbol, in Braille notation and classical notation as well (these indications appears in two separate windows). With the help of multimedia devices and the improvement of our software, it would be possible to listen to the whole score or a part of the score.



Designing a Sound Object Library

Victor Lazzarini e Fernando Accorsi

Núcleo de Música Contemporânea
 Departamento de Arte / Departamento de Ciência da Computação
 Universidade Estadual de Londrina

Abstract

The Sound Object Library is being designed to be employed by programmers and composers to develop sound manipulation applications. Its classes encapsulate all the processes involved in sound creation, manipulation and storage. The library can be used as a framework or, alternatively, as a set of objects to be patched together in a program. It is being developed to be portable across the UNIX and Windows platforms. The library class hierarchy is founded on three base classes, corresponding to sound processing objects, maths function-tables and sound input/output objects. A number of classes have already been implemented. A programming example shows one of the possible applications of the library objects. A GUI interface for the objects is proposed, using the V C++ framework.

1 Introduction

The research described here derived from some ideas that evolved during the work on the software *Audio Workshop* (Lazzarini, 1998). In that process, it was noted that the development of a set of objects for audio signal processing could be very useful in the design of new programs. It would help provide higher level tools for audio programming and software with a more intuitive user interface. This set would be used in two ways: as finished objects, to be employed directly in a program (or in a visual patching application), and as a framework, to which developers could add their own specialized code. Consequently, programs to carry out specific tasks would be easily developed by connecting the available objects and providing standard control-of-flow. Also, a text-based sound compiler could be designed to make use of the library. Using derivation and inheritance, new objects could be created from the existing ones, thus leaving the development possibilities open. The creation of a Sound Object Library seemed a worthwhile, though complex, task to pursue.

The project took shape as some important premises were being defined. Firstly, the library objects should encapsulate all the processes involved with production, manipulation and storage