The Design of a Pen-Based Music Notation System

Annick Leroy, Giovanni Müller, and Guy E. Garnett

Center for New Music and Audio Technologies (CNMAT)
Department of Music
University of California, Berkeley
1750 Arch Street, Berkeley, CA 94709
(510) 643–9990
annick.leroy@irisa.fr, mueller@cnmat.berkeley.edu, guy@cnmat.berkeley.edu

Abstract

We present the basic design of a music notation system that uses a pen device and handwritten symbols for fast, flexible input capability. The objective is to rapidly obtain a description of musical ideas that is suitable for further processing, not only for printing, but for use with a digital composer's assistant as well as in representations for synthesis control and performance. In contrast to the mouse, the pen provides more precise gestures due to the direct feedback from a touch-sensitive display. We focus on a system for composers, not for engravers: composers are skilled in handwritten notation and editing and need to sketch out musical ideas rapidly while retaining the ability to make alterations at both local and global levels.

1. Introduction

In a general way, our system can be partitioned into two interdependent parts: a recognition part, and a representation part. After describing the system from the perspective of the user requirements, we will present the overall system architecture that we think best fulfills those requirements. After that, we will describe the representation layer since this is crucial to understanding the recognition and display mechanisms which we will describe last.

1.1 User Requirements

The primary requirement is to make entering and editing music notation simple and fast. Composers and copyists are skilled in hand-scripted notation and editing; we wish to provide a tool to use these skills directly to obtain, ultimately, engraver quality results.

A secondary requirement stems from the fact that composers often need to sketch out musical ideas rapidly retaining the possibility of making alterations at both local and global levels. It should be simple to say both, "make this F an F-sharp," and "transpose this passage in F to F-sharp."

Third, we want the underlying system to be more than just a notation system. In this age of electronic performance, we expect the system to support performance of notated music via computer and other electronic instruments including, but not restricted to MIDI, ZIPI, and other real and non-real-time synthesis methods. In fact we want the system to support a variety of notions of score and not just Common Music Notation (CMN). The underlying mechanism should be strong enough to support different user interfaces. It is for related reasons we believe the editing operations should be focused on the musical structures and not on the layout properties.

Fourth, we want the system to provide full documentation of the *process* of composing or notating. The user should have coherent and usable access to all sketches and versions of the score.

Fifth, we want the input and editing to largely correspond to the way we work with paper and pencil. We do not want automatic behavior to interfere with user input. For example, on paper a note remains in the same place on the staff even if the user changes a clef in front of it. In most computer notation systems, the note is redisplayed in a new position because its *pitch* is thought of as its primary attribute rather than its *position*. In this sense it is more "natural" that entering or changing a clef with the pen device should not affect the way the rest of the document looks. We can state this user paradigm as: "*local* editing should not result in *global* changes."

Finally, a requirement too often overlooked or at least too infrequently achieved especially in music software, we want the whole working environment to be robust. We don't want erratic, unpredictable behavior.

1.2 System Design Issues

The requirements outlined above, though fairly broadly stated, substantially narrow the choices for system design. These choices will now be defined and explained.

We have opted for a pen as the primary user interface tool largely because it allows users to make the most of the substantial skills they already posses. Its use is very simple to understand and should be quicker than other methods. In this, music notation is very different from handwriting—whereas most people can learn to type faster than they write by hand, it is not clear whether this can ever be the case with music except

possibly the simplest kinds because of the extreme variety in symbols and ways of joining symbols together. Opting for a pen based system however, means we must make a substantial investment in the design and integration of a very flexible recognizer. This recognition section will be covered in greater detail in a later section.

A number of our requirements push us in the direction of developing a strong music representation at the heart of our system. For example, since we have made the requirement to allow global as well as local changes, and in fact we see global changes as being generalized to include process oriented modifications to an existing music document as well, we need to have a substantial structure underlying what the user sees. The system can not simply represent graphical elements but must reify musical concepts at some deep level. For a second example, since we want to be able to support more than CMN, we want to be able to support MIDI and ZIPI and probably other performance-oriented presentations of the musical data, the underlying structure must be strong enough and general enough to include attributes targeted for these applications. Furthermore, a substantial structure is necessary to meet the primary goals of supporting a full-blown CMN and the detailed history we desire. Finally, we believe that in a system as complicated as the one we are designing, it is absolutely necessary, from the software engineering point of view, to have a very robust architecture with clearly defined components and communications.

In order to handle the complexity of the overall system, we have opted for an object-oriented software model and, what we believe to be one of the most coherent and flexible architectures for complex user interface structures, the Model-View-Controller (MVC) paradigm. These will be very briefly explained as there is a great deal of literature on them [Goldberg 83].

1.3 Model/View/Controller (MVC) Paradigm

The MVC paradigm is a technique to modularize a system by dividing a complex user-interface into three principle components that communicate, as all objects do in an object-oriented system, by messages. In the MVC paradigm, the model is an encapsulation of the basic conceptual structure, the data structures associated with it, and the functionality; the view is an encapsulation of all the necessary data and functionality needed to create some kind of display; the controller is a means for the user to effect things in the model—it is often closely connected with the view because it needs to know, for example, what displayed object the user was pointing to and only the view knows what object is where in the view. In an anthropomorphic sense, there is an object (the model), a way of looking at it (the view), and a way of manipulating it (the controller).

2. System Architecture

We apply this paradigm in a straightforward manner to obtain the following large-scale system architecture. We have what we call a Notation Layer as the Model; it is an abstract music representation system. It has information about note values and timing, for example. A Layout Layer is the View; it handles the representation of the page description and layout, and, ultimately, the display of the state of the Notation Layer. It includes, for example, relative position information and page and margin boundaries. Finally, we have a Recognition Layer as the controller structure; it receives pen input from the user and makes a decision about what the input means. It may ask the layout model for information on what the spatial environment of a stroke contains, and eventually it makes a decision and informs the Notation Layer, "hey, someone just added an eighth note here!" The Notation Layer, as the Model in the MVC paradigm, makes the appropriate changes to its own data and passes a message to the Layout Layer telling it to update its state.

2.1 Layer Structure

This is, in fact, all there is to the primary system architecture. However, in sufficiently complicated systems, such as we have here, each of these parts may need to be structured in more detail. In practice, often even the subparts of an MVC structure (that is, the Model, the View, and/or the Controller) themselves take on characteristics of embedded MVC structures. For example, in our case the Layout Layer needs to contain so much internal structure that is still abstract in nature (for example, all the pages that are not being explicitly looked at presently) that it is thought best to create a quasi-MVC structure to hold the different components of this one layer. In this more detailed level of structure (as can be seen in Figure 1.), the Layout Layer consists of a Layout Model and a Layout View (and could also, but does not in this case, consist of a layout controller as well). Its view, the Layout View, is what actually gets around to displaying the current page or score area. In the same manner, the Recognition Layer is actually designed as a smaller scale MVC structure as well. Its model, called the Recognition Model, handles the basic representation of stroke data and the functionality of the decision making process. Its view, called the Recognition View handles display of stroke information (the "ink" on the screen). Its controller, the Recognition Controller receives input from the pen and informs the Recognition Model of the pen's location. The Recognition Model then updates its stroke data and informs the Recognition View to

update itself, which it does by displaying the new data on the computer screen.

this history mechanism to the user input and recognition is described later in this paper. Now we

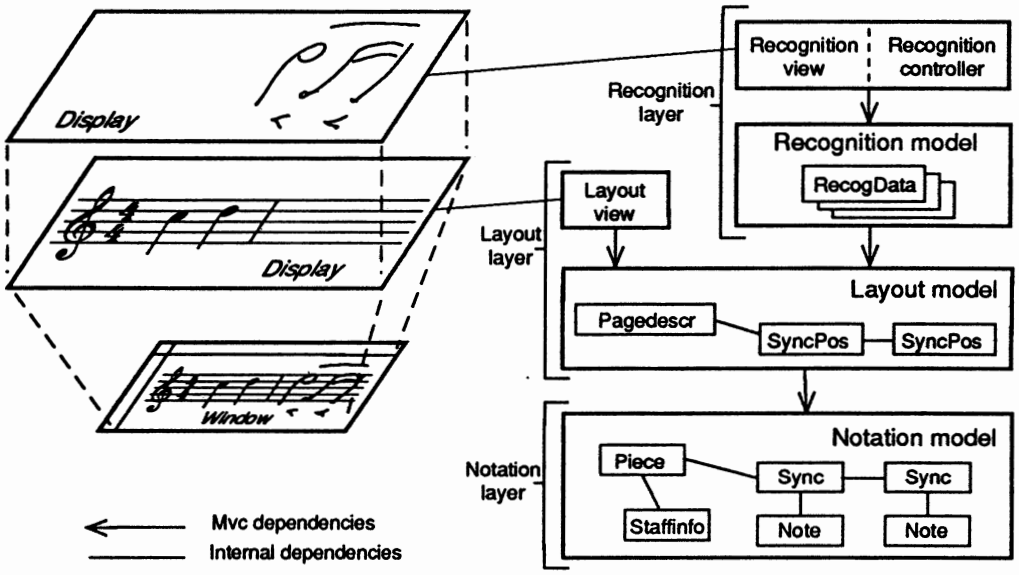

Figure 1.

These layers will be described in more detail below. For now we turn to another part of the basic system, the history mechanism.

2.2 History mechanism

One further aspect of this system design is necessitated by our desire to have a very full history mechanism to document not only the final state of the notation, but to record the entire process, including the strokes input, in such a way that the user can retrieve any previous state (with some limitations) and create a new document proceeding from there.

The history mechanism is document oriented, i.e., the objects which need to be saved are linked into a "history document". When the document is being saved we add a unique identification tag to all its objects that are newly created, the ones being saved for the first time. In a second step we store all objects there were changed since the last save by adding a time stamp and by making a read only copy. Notice that all references between objects are stored by the identification tag, so that the document is self-contained and it might be externally stored on a file and read in another context without loss of consistency.

Any saving step of the document is a snapshot of the document state at that time. The whole history may be traversed and the objects may be read but not changed anymore. In this sense there is no undo operation that restores a previous state, rather one traverses the sequence of states starting from the original up to any point including the current state. The relationship of

describe the structure of the layers.

3. Notation layer

The notation layer contains two levels of abstractions for music representation. The lower level is defined according to the Smallmusic Object Kernel (SmOKe) [Pope92], an object-oriented description language for musical parameters, events and structures. The higher level of abstraction is represented by the "notation model". This is an abstraction derived from CMN that describes the score only in terms of pitch and timing, independent of graphic properties. The purpose of the notation level is to define a thin interface between the world of musical events and the world of the musical notation symbols.

The concept of pitch is quite straightforward and does not need to be handled here.

The concept of timing is much more difficult to handle than pitch, not only because it is non linear, but because there are two kinds of timing, one of which is described for the first time here. First, the "notation timing" is given by the "timed notation symbols" (i.e. notes, chords, rests). In the score almost all vertically aligned symbols correspond to the same time. In order to keep this syncronicity in the model we define as a "sync" the collection of all synchronous elements.

On the other hand the timed symbols (notes, rests) are grouped into horizontal sequences called "parts" (e.g., voices). Basically, a piece is a sequence of syncs which refer to notation symbols that may be collected in parts. See Figure 2.

The sequence of syncs defines a notation timing grid, according to the conventions of CMN (Figure 2. upper scale). This grid is computed by an algorithm based on the minimal timing differences between the non synchronous timed notation symbols [Müller 90]. Many of the non-timed notation symbols, such as dynamics and articulations, are defined as relations to timed symbols and therefore their position is well defined. There is another class of symbols, however, which cannot be defined in this way.

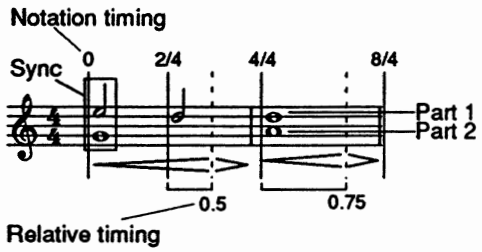

Figure 2.

The use of notation timing is common in music notation systems, but it leads to many problems when we try to define the position of certain notation symbols (such as grace notes and dynamic hairpins) which are not synchronous with timed symbols and do not imply a definite time at all. In fact, the timing and spacing of these symbols is not defined by notation value and place in a sequence (as it is for a sequence of quarter notes, for example), rather it is defined by their relative graphic placement in the score. For this purpose we define a second kind of timing, which we call "relative timing." Relative timing is defined in a qualitative way (i.e., before or after a sync) or in a quantitative way by the proportional position between two syncs (see Figure 2 lower scale). In the figure, the first hairpin crescendo keeps its relative temporal position (reaching its maximum at about the fourth quarter of the measure) even if we delete the note at the timing position 2/4. The coordination algorithm works by recomputing the new proportional position when a sync is deleted. Therefore the new proportional position of the crescendo will be as in the second measure. In the case of the relation "before" (i.e. grace notes) the element remains before the sync it is attached to yet after any other timed notation symbols that may be added before it in sequence. In this way a grace note remains at the same position relative to its sync even if notes are added to the part before the sync.

4. Layout layer

The layout model transforms the pitch and timing coordinates of the notation model to the graphic coordinates of the display. It contains the data to describe the location of all the notation symbols on a page. Typically the size of the staff (and therefore of the notes), the size and the margins of the page are described in the layout. On the other hand also the local data such as the positions of the syncs in the page are necessary. This is because the layout model must not only be able to display notation symbols, but must be able to return which elements are located at some position on the page.

5. Recognition Layer

The Recognition Layer consists, as described above, of three basic components: first there is the Recognition View. This view displays the user's pen strokes on the display surface. This display is conceived of as a transparent "sketch sheet" that is overlaid on top of what is currently displayed by the Layout Layer. When starting a new composition, for example, the Layout Layer may be empty except for staves that have been previously defined with a staff system template. The sketch sheet is empty, the staves are drawn (under it) by the Layout Layer. The user begins entering strokes which are displayed by the Recognition View as "ink" on the sketch sheet—thus overlaying the layout staves.

The second component of the Recognition Layer is the Recognition Controller. This is the mechanism by which user commands and strokes enter the system. The Recognition Controller receives information from the pen device; it makes Pen-up, Pen-down decisions and hands over to the Recognition Model the time-stamped pen location data. In order to simplify the recognition task, it is up to the user to decide when sufficient information has been presented to begin a recognition (indicated by clicking the pen in a "button," for example). By making the user decide when to attempt recognition we avoid the problem of the recognizer trying to make a decision based on a partially defined symbol (such as a filled note head without a stem). This simplifies the task of the recognizer. We also leave it up to the user to "accept" the results of the recognition or go back and clean up if something was not clear. In addition, we wait until the user accepts the recognizer's results to define the boundary of a sketch to be preserved by the history mechanism. That is, the user can make all kinds of strokes with erasures and other changes, but only when the recognition is accepted does the current state of the Recognition View get stored in the history and the Notation Layer get notified of the changes. This simplifies the history mechanism as well.

The third, and by far most complicated, part of the Recognition Layer is the Recognition Model. It handles the inputs from the Recognition Controller maintaining and updating the current state: including the current stroke data, the current editing mode, etc. It also communicates with the other two system layers to gather a context for the recognition. When the Recognition Controller informs the Recognition Model that the user is requesting recognition, the

Recognition Model informs the Recognizer to begin operating on the current stroke and context state stored in the Recognition Model. Though this is a simplification of how the overall system functions, it is the basic story. We can now present a few details concerning the Recognizer itself.

6. The Recognizer

Musical handwriting has a lot of particularities which distinguish it from text handwriting. These characteristics strongly influence the recognition method as well as the data structure:

- There are a minimum of sixty different symbols.
- The position on the staff and the size helps comprehension.
- Music has a very precise grammar.
- Strokes are not written in a left to right order: the user will often write all the note-heads before the stems in a tuplet, go back to put in slurs, etc. It is thus necessary to wait until the user is finished with a coherent part to exploit it.
- The symbol combinations are infinite (the presence or absence of an alteration, of a dot or an accent varies, as does the stem length, and the number of ledgerlines above or below the staff).
- There are many ways for grouping notes: beamed, slurred, tied, in chords or in voices.
- Interpretation follows time order which is from left to right, but each note can be bound on top or underneath to interpretation symbols (dynamics, articulations, etc.).
- Synchronous notes are vertically aligned either in chords or in voices (unless they are a step apart, in which case they must be offset for legibility), generally on the same staff for an instrument, except for keyboards, and on several staves for orchestrations.
- Some symbols have fixed size (relative to the staff size), but others are expandable like note stems.
- Some symbols are related to one staff but others cover several staves (such as cross-staff beaming in keyboard parts).
- Some symbols are composed of a single stroke (G-clef or a measure line), while others are composed of several strokes (a quarter-note or an F-clef).
- Depending on the writer, a given symbol might be composed of one or several strokes (an eighth-note can be written with one or two strokes).
- Depending on the context a given writer will form a symbol differently (a quarter-note might be composed of one stroke when isolated but of two when in a tuplet).
- Musical handwritings are as diverse as text handwritings. A simple quarter note can be written from top to bottom or the opposite, the head might be shaped like a dot, a spiral, a line, or a big scribble.
- Several distinct elements of a tuplet can be represented by one stroke, for example many writers use one stroke for the stems and the bar of a duplet.
- Composers are used to creating new symbols and modifying the existing syntax.
- A certain shape of stroke might belong to different symbols (the dot appears in dotted notes, in an F-clef, in bar-lines, etc.).
- There is often no more space between two different symbols than between the strokes of the same symbol (the space between a note-head and its stem will often be as wide as the space between two notes).
- Symbols overlap in width and height (a tuplet with one or several beams across the notes, accents above or below the notes, notes of different voices overlapping, etc.).

6.1 . Remarks about on-line musical handwriting

It is important to retain the timing information when the pen data is made for three reasons: first, since a stroke is defined simply as any pen data between a pen down and a pen up, we can easily segment the data into separate strokes and distinguish overlapping strokes. Second, with the time-stamped information we know how each stroke was drawn. Third, we can exploit the order in which strokes were written.

6.2 . Recognition algorithm

The previous two sections show where the problems are, that is, reordering the symbols from left to right, grouping them into musical entities, and segmenting or recognizing strokes representing different elements of a symbol. In no way is it possible to have a classifier provide one label per symbol, there are far too many and it is sometimes impossible to distinguish them without a context. This is why we chose an analytical approach with several levels of recognition. In a prototype, we have shown that the algorithm works at least on a limited set of symbols. We are now gathering data to train the recognizer on a larger set.

We use a bottom-up approach summarized as follows:

- stroke labeling

We use a TDNN (Time Delay Neural Network) [Guyon, et al, 1991] to provide a list of labels for each stroke (see Figure 3, step 2). These labels can be shapes (vertical line, dot, etc.) or symbols (G-clef, sharp, etc.). The labels are eventually disambiguated using global features such as the size or the position on the staff or the distance to other symbols.

- symbol grouping

We next group the strokes into elementary music symbols using a music symbol graphic grammar. These elementary symbols are notes, alterations, clefs, etc. (see Figure 3, step 3). For each symbol, a set of rules describes the position of the strokes relative to one another with a distance tolerance. These distances can be learned and adapted to a user.

From a few strokes we can build several different elementary notation symbols. In Figure 3, step 3, for example, we see various hypotheses for the few strokes entered.

- validation

We next validate these first hypotheses using a symbol classifier, such as a TDNN. This gives us probabilities for the symbols. For example, if the nearly filled circle (third box from the left in step 2) is notehead size, the probability of it being an augmentation dot is decreased and the probability of it being a notehead is increased.

- high-level grouping

Then we group the elementary symbols into musical units, for example, we connect an alteration to the following note, a dot to a previous note, etc.

- building relations

It is at this point that we build up relationships between the above musical units, for example, we determine tuplets or the binding of slurs with note groups, etc.

Now we have generated a set of high level hypotheses based on the strokes. We must pick a set of the best ones (in term of probabilities). This is how we proceed:

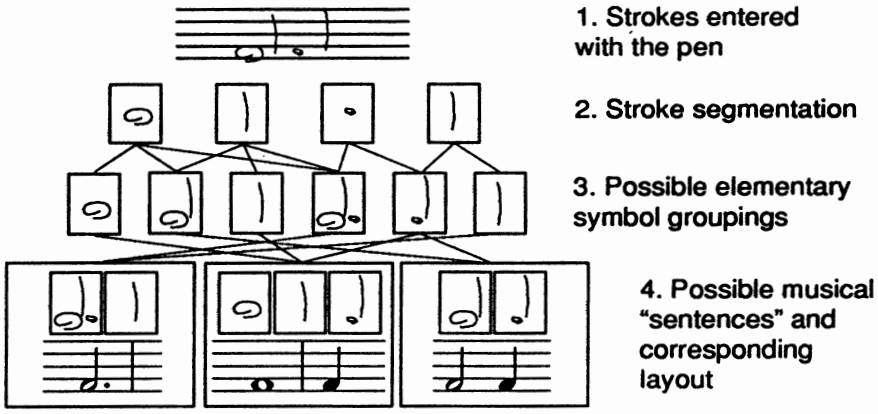

Figure 3.

- sentence construction

We construct all the possible musical "sentences" using all the strokes only once and compute their associated probabilities (see Figure 3, step 4).

- selection

It is now possible to select the most probable sentence and present it to the user. If the choice is accepted, it is passed on to the Notation Level and propagated through the system resulting in the recognized symbols being displayed on the screen.

6.3 . Learning

TDNN are well suited for on-line recognition especially since they offer an adaptation capability as well as the ability to introduce new symbols. The network performs feature extraction which is followed by a classification step [Guyon, et al. 1991]. The learning phase needs a lot of data and gathering them is a difficult task. We need skilled and patient people.

We cannot ask composers or musicians to decompose a symbol into strokes in order to label them, it would be too boring even for very patient ones! As mentioned before, the number of symbols is quite high and the combinations unlimited. So we need to limit

the data collected to very significant samples and extrapolate.

We need to be able to label any stroke, to know the distance and position of one stroke relative to the others in a symbol. We also need to train on some elementary symbols and learn the distances and relative positions of these symbols in groups and relations.

The examples we have designed are composed of several symbols, this obliges us to do a preliminary learning on elementary symbols. After this, the learning algorithm for the sentences is identical to the recognition algorithm with a few differences :

- The stroke labels are provided using an a priori description of the elementary strokes.

- Large a priori distances are chosen providing more hypotheses.

We match the possible sentences with the solution and associate a label with each stroke, record the distances and associate groups of strokes with elementary symbol labels.

This phase has not been implemented yet, since we do not have enough data. We have not trained a dedicated music recognizer, but have used the adaptation capabilities of an existing one in order to introduce the musical elements.

6.4 . Editing

Editing with a pen offers a lot of possibilities. It allows us to use quick, natural gestures to erase, select, or move a symbol. These gestures can eventually be defined by the user and associated with existing commands. Because the number of symbols in a music editing system is large, and they can occur in varied and complex contexts, it is best to use separate recognizers for the different tasks, and different editing modes for the music symbols and for the command gestures. The modes we use are similar to the command mode and the insertion mode in many text editors.

There are three basic editing modes we plan for: first, straightforward input of the stroke information. Second, simple editing of that stroke information prior to recognition. This is to enable the user to clean up minor slips of the pen, so to speak. The third activity is editing already recognized symbols. Since we have a transparent sketch sheet overlaying the display created by the Layout Layer, all of these activities appear to the user in a uniform manner. That is, the user makes strokes of various kinds on the sketch sheet. In the first mode we simply input the strokes and pass them to the Recognition Model. In the second mode (possibly initiated by touching an "eraser" button on screen, for example) we allow the user to select any stroke and delete it or move it. We have to be careful not to allow partial erasure of strokes because this adds too much complexity to the data representation. In the third mode, the user sees the fully recognized symbols displayed by the Layout Layer and can make gestures (on the overlying sketch sheet) to select, cut, copy, and paste. In this mode the user is manipulating higher level objects than simple strokes.

7. Conclusions

We have described in some detail the overall system architecture and the principle design criteria for a very sophisticated music editing tool. The recognition algorithms have been tested in simplified form using a network designed for text recognition. We are currently prototyping the Notation and Layout Layers in Smalltalk. In the near future we will add a TDNN to our prototype and begin constructing the recognizer and editing functionality. Our prototypes lead us to believe we are on the right track.

8. References

[Goldberg 83] Goldberg, Adele, and David Robson. 1983. *Smalltalk-80: The Language and Its Implementation.* Addison-Wesley. Menlo Park.

[Guyon 91] Guyon, I. and Albrecht, P. and Le Cun, Y. and Denker, J. and Hubbard, W. 1991. "Design of a Neural Network Character Recognizer for a Touch Terminal". *Pattern Recognition*, February.

[Leroy 94] Leroy, Annick. 1994 "A Pen Based Music Editor," in *Fundamentals in Handwriting Recognition.* Sebastiano Impedovo, Ed. Springer-Verlag .

[Müller 90] Müller Giovanni. 1990. *Interactive Music Notation Editing*, Ph. D. Thesis, ETH Zürich.

[Pope 92] Pope, Stephen Travis, et al. 1992. "Smallmusic Objet Kernel: A Music Representation, Description Language, and Interchange Format." Document named "OOMR.ps.Z" available via Internet file transfer from the server "ccrma-ftp.stanford.edu" in the directory "pub/st80."