Generating and analyzing MIDI files using Hindustani Music Notation

Report submitted in partial fulfillment
of the requirements for the degree
of
Bachelor of Technology
by

Shubham Gupta
11CS30035

Under the supervision of
Prof. Pallab Dasgupta

Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur
WB 721302, India

May,2015

CERTIFICATE

This is to certify that the report entitled "Generating and analyzing MIDI files using Hindustani Music Notation" submitted by Shubham Gupta, in the Department of Computer Science and Engineering, Indian Institute of Technology, Kharagpur, India, for the award of the degree of Bachelor of Technology, is a record of an original research work carried out by him under my supervision and guidance. This report fulfills all the requirements as per the regulations of this institute.

Prof. Pallab Dasgupta
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur
Kharagpur, India, 721302

ACKNOWLEDGEMENT

I am deeply grateful to my supervisor Prof. Pallab Dasgupta who gave me the opportunity to work on this project. I am thankful to him for his aspiring guidance, invaluably constructive friendly advice during the course of this project. I am sincerely grateful to him for sharing his truthful and illuminating views on a number of issues related to the project. I would also like to express my gratitude to Sudipa Mandal and Antonio Bruto Da Costa for their help at all stages of the project.

I am also thankful to my teachers of the Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur, who have instilled in me the scientific spirit of inquiry, experimentation, observation and inference, without which I would not have been able to produce this work. I hope I will be able to rise up to the high standards set by my predecessors in this department.

Shubham Gupta

Contents

1 Introduction

1.1 Motivation and Aim

Ragas have been a fundamental construct of Indian (both Hindustani and Carnatic) classical music for thousands of years. In language theoretic parlance, each raga represents a context free language over the sequence of notes used in rendering that raga. Ragas have well defined grammars in addition to its Arohana (notes in the ascent), Avarohana (notes in the descent), Pakar (signature sequence), vadi and vivadiswaras (admissible and inadmissible notes). Different renditions of the same raga may use remarkably different sequences of notes, but both renditions must satisfy the grammar of the raga. The quality of the rendition depends on the choice of note sequences, as well as other factors such as the timbre (or voice quality), tonal variations (such as meends) and more esoteric features that are perceived by the audience but are hard to define formally. Since a raga is defined by its grammar, it is anticipated that the grammar plays a role in the classification. While this may be true, there also exist prominent exceptions. Two ragas having widely different grammars often belong to the same class. This suggests that it is not the grammar alone that is responsible for such classification there may be features that are hidden in the traditional renditions of these ragas, which are not explicitly defined, but are responsible in associating a raga with the mood of a season. Such hidden features may have been passed on through generations through the tutelage of traditional ways of rendering a raga, without being explicitly elucidated by the teachers. Given a rendition and the grammar of a raga, determining whether the rendition follows the grammar of the raga is a parsing problem. We wish to extract note sequences from audio files for raga renditions by expert musicians, parse the renditions into ragas and also be able to score the renditions based on certain features.

Notation in Hindustani music arose in a musical culture which was and continues to be primarily an oral tradition. In all musical traditions, notation can only be satisfactorily interpreted by musicians who have knowledge of the musical culture, and this is particularly the case in Hindustani music. No representation is complete, and the traditional representation system omits as much as it captures. This system of notation has often been criticized for omitting essential elements or greatly simplifying performances.

If fact, its abstraction can be an asset for the researcher, focusing attention on certain categories and simplifying or eliminating other aspects. Traditionally, over a thousand years before notation, the concept of notes and discrete pitch levels for notes was a firmly established part of Hindustani music. For the re-

searcher, modern symbolic notation, even with its limitations, is a rich source of information about pitch and durational information in Hindustani music.

A **machine readable format of symbolic music** is useful in a variety of applications. Large symbolic music repos- itories can be built using this format. They serve the pur- pose of preserving compositions and pedagogy. From the viewpoint of MIR research, they are primarily useful for large scale statistical comparative studies. They can be used for generating synthetic melodies and storing automatically generated transcriptions. Other MIR applications where such a system of representation would be useful are melodic transcription, melody prediction and continuation through melodic sequence modeling.

In this part of the project we were interested in building a tool that could act as an interface between the annotated notes that we have defined for our grammar and the corresponding music file associated with it. More specifically, we would like our tool to take the annotated notes as an input and generate a MIDI (Musical Instrument Digital Interface) file for it. Also we would like the tool to take as input a MIDI file and generate its notation in the format devised by us.We are also interested in inquiring whether the MIDI format is sufficient to cover all the varied aspects and the many ornamentations that are inherent in Hindustani Classical Music

2 Background: The Journey of Indian Music

The origin of music is said to have come from God himself. Indian music can be traced back to the Vedic hymns in the Hindu temples nearly 3000 years ago. The Vedas constitute the oldest layer of Sanskrit literature and the oldest scriptures of Hinduism. **Rig-Veda** is a collection of poems that tell the stories of creation and of the Indian gods. The preservation and transmission of the Rig-Veda became the responsibility of the Brahmins. *These stories were eventually preserved in sung chants, not written down until recently.*

When these texts and chants were re-arranged as hymns they were set to a special collection of tunes called the *Samagana.*These in turn became the basis for secular musical compositions.The oldest treatise on the arts, the *Natyasastra*, links music as an adjunct to drama. The purpose of music (or artistic experience in general) is arouse the emotions in specific ways. Consequently, each raga and tala are designed to illicit specific s and emotions. A musician should aim to attain a state of self-abandonment in order that he fuse with the Supreme Reality.

Indian music has developed through very complex interactions between different peoples of different races and cultures over several thousand years. However, references to music in ancient texts, aesthetic formulations, and depictions and written discussions of musical instruments can offer clues. In rare instances an ancient musical style may be preserved in an unbroken oral tradition. For example, musical notes or the structure of a raga, as we know them today, must have had their origins in the Samavedic times.

Indian music belongs to one of the oldest and richest living traditions of monophonic modal music, and as such, has had an influence on many important music traditions (including western art music), which have incorporated some of its basic musical concepts. Over the centuries, Indian music had been open to and subjected to, influences from other traditions, which has further widened its musical base and appeal. It has developed a wealth of instruments and styles, which have developed right up to the present day. Although Indian music is a so-called classical tradition, with an ancient and highly developed theoretical base, it also has important links to the folk and popular styles. The built-in possibility for creative input by performers through improvisation and composition, even within the constraints of the traditional structures, has kept Indian music in a continual state of renewal and readiness to adjust to changes in tastes and society. This

has also helped it extend beyond narrow borders of social classes and geographic borders.

Ravi Shankar who introduced the western world to the rich treasures of Indian Music had this to say about the essence of music:

To us, music can be a spiritual discipline on the path to self-realization, for we follow the traditional teaching that sound is God - Nada Brahma: By this process individual consciousness can be elevated to a realm of awareness where the revelation of the true meaning of the universe - its eternal and unchanging essence - can be joyfully experienced. Our ragas are the vehicles by which this essence can be perceived.

2.1 Introduction to Ragas

Indian classical music is defined by two basic elements - it must follow a Raga (classical mode), and a specific rhythm, the Taal. In any Indian classical composition, the music is based on a drone, ie a continual pitch that sounds throughout the concert, which is a tonic. This acts as a point of reference for everything that follows, a home base that the musician returns to after a flight of improvisation. The result is a melodic structure that is easily recognizable, yet infinitely variable.

Raga is a noun derived from the Sanskrit root ranj, meaning to colour red, to delight. Red is the colour of passion, hence raga implies the emotional content of a song which delights the listener. Ragas have a particular scale and specific melodic movements; their sound should bring delight and be pleasing to the ear.A raga can be regarded as a tonal framework for composition and improvisation which is far more precise and much richer than a scale or mode, and much less fixed than a particular tune.

A Raga is popularly defined as a specified combination, decorated with embellishments and graceful consonances of notes within a mode which has the power of evoking a unique feeling distinct from all other joys and sorrows and which possesses something of a transcendental element. In other words, a Raga is a characteristic arrangement or progression of notes whose full potential and complexity can only be realised in exposition. This makes it different from the concept of a scale in Western music. A Raga is characterised by several attributes, like its Vaadi-Samvaadi, Aarohana-Avrohana and Pakad, besides the sequence of notes which denotes it. It is important to note here that no two performances of the same Raga, even two performances by the same artist, will be identical.

Raga is a melodic abstraction around which almost all Indian classical music is prepared. A raga is most easily explained as a collection of melodic gestures and it is a technique for developing them. The gestures are sequences of notes that are often inflected with various micro-pitch alterations and articulated with an expressive sense of timing. Longer phrases are built by joining these melodic atoms together

2.2 Structural Features of Raga

Indian Classical Music has two related, but distinct, traditions: Hindustani and Carnatic.The basic scale of Hindustani music, like western scale, has 12 notes: **Sa Re Re Ga Ga Ma Ma Pa Dha Dha Ni Ni** Raga is a subset of these notes together with a set of rules to combine these notes effectively and create a particular mood. It must have atleast 5 of these 12 notes.They must contain the tonic (Sa) and at least either the fourth (Ma) or fifth (Pa). the structure of a Raga follows a fixed grammar or a fixed sequence of notes that govern the progression in some specific direction. This fixed grammar of the basic unit of Indian Classical Music enables one to model its structure computationally.

In Indian Classical Music there are certain properties on the basis of which the Ragas are defined. The Properties are Aarohan, Avorohan, Jati, Time of Singing, Pakad, Register, Position of notes, Vadi Notes, Samvadi Notes

Pakad is a string of notes characteristic to a Raga to which a musician frequently returns while improvising in a performance. Pakada literally means a grip. In musical parlance, pakada is the signature phrase for the raga. Given two very similar ragas, they differ at least in their pakadas. Formally, the pakada is any string which conforms to the Arohana and Avarohana. It is important from a computational point of view since it is unique for the raga. The Pakad also serves as a springboard for improvisational ideas; each note in the Pakad can be embellished and improvised around to form new melodic lines. One common example of these embellishments is the splitting of Pakad into several substrings and playing each of them in order in disjoint portions of the composition, with the repetition of these substrings permitted. In spite of such permitted variations, Pakad is a major identifying characteristic of a Raga and is used even by experts of Indian classical music for identifying the Raga been played. The pakada might not be present in its entirety in the composition, but parts of it may be distributed over the composition as the underlying theme of the composition. Such hurdles force a nondeterministic way of identifying the pakada.

Thaat: A Thaat is the parent scale of a Raga.Thaat is a way of classification

of Ragas. Each Thaat is a set of seven notes in an octave. Classification of Raga into a Thaat is empirical rather than strictly formal, because a raga might use more than 7 notes. For example, thaat Khamaj contains the notes S, R, G, 'M, P, D, n but the raga desh belonging to this thaat uses both n(komal ni) and N(suddha ni).As a matter of fact there are 10 thaats defined in Hindustani classical music out of a possible 32

Aarohan & Avarohan: The sequence of notes of a particular raga or thhat in ascending order of the frequencies starting from the tonic of the scale of performance is called the Aroha and the sequence of notes starting from the double frequency of the tonic of the scale to the tonic of the scale in descending order of frequencies is called the Aboroha. By these two properties a Raga can be decided to belong to a Thhat. As example, In Raga Multaani

Aarohan NiSa, ga MA Pa, Ni SA.
Avarohan SA, Ni dha Pa, MA ga, re Sa
. From these two note streams it is clear that the Raga Multaani is composed of the notes used in the Thhat Torhi. So. It can be decided that the Multaani is belonging to the Thhat Torhi.

Aalap: Aalap of a raga is a rendition of the raga in which part the possible legal combinations of the used notes are performed without any fixed rhythm. Here in the beginning portion the performer starts from the tonic and reaches to higher to higher frequencies according to ability and expertise and then comes downward to reach to the tonic gradually.
Shrutis: The linking between notes is called the Shrutis. To get more pleasant and melodious music the Shrutis are used by the expert musicians. Shrutis are interlinked and merged with the notes. In between the transition from one note to another note these shrutis are present.

Vadi,Samvadi,Anuvadi and Vivadi: These notes are specific to a raga.Vadi is the most important note in a raga.Samvadi is the second most important note in a raga.Anuvadi notes are those notes which are neither highlighted nor downplayed. Vivadi notes are those notes that are either not included in a raga or are used very rarely.
Other attributes: Nyasa is the last note of a specific phrase of notes,which leads to its end- ing.Poorvang and Uttarang are lower and higher regions of an octave.Tirobhav is the process of concealing a raga on a temporary basis.Avirbhav is that technique of presenting the raga, in which the raga is noticeably expanded

and exhibited.Jati refers to the classification of musical compositions as per the tones.

2.3 Classification of Ragas

Indian ragas have been traditionally classified in several ways. A raga must evoke a particular emotion or create a certain mood.Each Raga has a name. It also has a character, which can be devotional, erotic, bold and valorous, or tragic.Some of the common classification systems are:

Season: Ragas are classified into seasons such as summer, monsoon, autumn, spring winter.Some Ragas are related to seasons. For example, Raga Malhar.
Mood: Some Ragas are known to offer specific moods, such as serenity, sadness, happiness, solitude, etc
Time of day: Ragas are classified into dawn, day, dusk, night ragas and sometimes in more detailed periods of the day/night (such as the prahar of the day/night).Time of the day when it is best performed is usually specified as a 3 hour interval. However the beauty of the raga is not affected by the time of the day it is sung.

Raga Desh is famous for its feeling of belongingness while raga Lalit is known to impart a sense of beginning. Every raga is associated with a time of the day when it is supposed to be rendered. Thus Bhairav is an early morning raga whereas Yaman is suitable for the evening and Malkauns is to be rendered late in the midnight. Some ragas are associated with seasons too. Miyan ka Malhar is ideally sung in the rainy season while Vasant is the coveted raga of spring.

Ragas can be classified using different criterias. For example, on basis of the number of notes used. Ragas that contain all 7 notes in ascent and descent are Sampurna, those with 6 notes are Shadav, those with 5 are Audav. All ragas are divided into two groups as Poorva Ragas and Uttar Ragas. Poorva Ragas are sung between 12 noon and 12 midnight, Uttar Ragas between midnight and 12 noon.

Another division of ragas is the classification of ragas under five principal ragas : Hindol, Deepak, Megh, Shree and Maulkauns. From these five ragas, other ragas are derived.

The first derivatives of the ragas are called raginis, and each of the five ragas

have five raginis under them.These raga and raginis also have derivatives. this results in each principal raga having 16 secondary derivatives known as upa-raga and upa-raganis.

Ragas are not static. Some can be traced back to ancient or medieval times, others originated only a few centuries or even a few decades ago.However, all ragas have undergone transformations over the centuries. The light ragas, used in folk melodies and popular songs, allow some freedom to add extra notes.The serious ragas have more definite rules and they can be elaborated in performances. A Raga may produce the effect it normally does on an audience not only because of the structure of its Arohana and its Avarohana, but because of some standard phrases used in its rendering at times. Such Ragas may not be totally specified by simply modelling its Arohana and Avarohana, and might call for more detailed treatment.

An important way of Raga classification is the **thaat** system.A certain arrangement of the seven notes with the change of shuddha, komal and teevra is called a thaat.Every raga has a fixed number of komal or teevra notes, from which the thaat can be recognised. A thaat is named after the popular raga of that thaat. For example, bhairavi is a popular raga and the thaat of the raga bhairavi is named after the raga. A thhat is a musical scale, conceived of as a Western musical scale can have, with the seven notes presented in their order of ascent (aroha). In Indian Classical Music there are 10 thhats from each of which many ragas are created. The names of these ten thhats are - *Kalyan, Bhairav, Kafi, Asavari, Bilabal, Khamaj, Bhairavi, Purbi and Torhi.*

subsectionPitch Bending in Hindustani Music In both Western as well as Indian Classical Music, the ability to be able to slide from one note to other is paramount to be able to effectively generate realistic, expressive music. In Western Music, Guitars, Violins and bass instruments continually slide notes up and down.**Vibrato** is a musical effect consisting of a regular, pulsating change of pitch. It is used to add expression to vocal and instrumental music. *Vibrato* is typically characterised in terms of two factors: the amount of pitch variation ("extent of vibrato") and the speed with which the pitch is varied ("rate of vibrato").Similarly, **Portamento** is a musical term that describes pitch sliding from one note to another while a **Glissando** refers to a glide from one pitch to another.

In Hindustani Music,the **Meend** is arguably the most important of all ornaments. It is somewhat similar to the glissando of western music in the sense that it is a smooth glide from one note to another.It is a compulsary ornament

in many Raga of Shuddha Geeti or Gaurhar Bani. Such Raga cannot be properly presented without the necessary Meend. In Hindustani music, the ornament is absolutely at least as important as the note itself. For example, the descending progression from Ma to Re is an essential element of any Raga belonging to the Malhar group. But, a mere movement from Ma to Re will not characterise a Raga as one of the Malhar family if the absolutely mandatory Meend from Ma to Re is absent.

Similarly, there are occasions where the primary difference between two Raga is that while the one dictates the use of Meend, the other dictates that Meend must not be used, or used minimally, if at all. For example, Bhoopali is a Raga that dictates an absence of Meend while Shuddha Kalyan demands that Meend must be used. The two Raga have virtually similar notes. Similar is the case with, for example, the Raga Darbari Kanada, where Meend is most absolutely de rigueur and the Raga Adana, where Meend must not be used.

3 Musical Notation

3.1 Historical Overview

In Hindustani classical music, we do not notate music for performance purposes because a classical music artist is by definition one who is capable of extemporaneous raga development, and a classical music performance, by definition, is an act of extemporaneous raga development. Further, notation cannot be completely comprehensive and capture all the subtle elements of a performance, which mainly depends on the performer's virtuosity. We do, however, use notation to teach and learn music, and as an aid to memory. When you learn a new raga, you notate a few basic melodic phrases, patterns and simple compositions in that raga so that you can recall them later.

It is well known that Indian music is based on an oral tradition. However, it is often erroneously presumed that this oral tradition precluded any musical notation. This is not the case; musical notation in India extends back to the Vedas. Musical notaton, known as swar lipi has existed in India from ancient Vedic age up to the modern internet age.The history of Indian musical notation is very rich.The Vedic hymns were typically sung in three notes. The central note was referred to as the "swarita". This was the default state and needed no notational element. The upper note was called the "udatta". This was denoted with a small vertical line over the syllable. The lower note was called the "anudatta" and was denoted with a horizontal line underneath the syllable.

Modern musical notation may be said to have begun with **Vishnu Digambar Paluskar** at the turn of the 20th century. Paluskar's notational system was used by music colleges in Northern India for the next few decades. An example of Paluskar's notation is shown below:

Although Paluskar's system was precise, it was difficult. It was soon to be replaced with an equally precise system, but one which was more intuitive. This system was introduced by **Vishnu Narayan Bhatkhande**. Today it is his system which has become the standard. An example of Bhatkhande's notational system is shown below:

There are a few other minor systems that may sometimes be found. One of which is **Western staff notation**. Although this makes Indian music accessible to Europeans and Americans, it has a poor acceptance within India (this will be discussed later).

Figure 1: Paluskar's Musical Notation

3.2 Bhatkhande Notation

The Traditional Notation System

There have been many systems of notation in Hindustani classical music over the centuries, but a system proposed by musicologist Vishnu Narayan Bhatkhande (1860-1936) gained widespread acceptance during the early 20th century, and is commonly used to this day in music textbooks and other situations within the Hindustani classical music community. The Bhatkhande system uses the Devanagari (Hindi) script for the notes and the lyrics, and a few other simple symbols. Let us become familiar with the particulars of Bhatkhande's notational system. The previous example is reshown below with annotations to make it easier to follow:

The above example shows two lines of a sthai in rag Basant. We see that there is a melody line with the corresponding lyrics underneath. This particular example is in tintal so the four vibhags are delineated with vertical lines. here are also occasional grace notes which may be indicated.

The Bhatkhande system is a model of elegance and simplicity. The basic notational elements are shown in the figure below:

In the above table we see that one simply has to write out the Sa, Re, Ga, etc. The komal swar (flattened notes) are designated with a horizontal bar beneath. The only note which may be sharpened is the Ma, this is designated

Figure 2: Bhatkande Notation

with a vertical line over it. The various claps of the tal are designated with their appropriate number (e. g. "2" for the second clap, "3" for the third clap etc.) the khali is designated with a zero. The sam is designated with an "X". Some authors may use a "+". The vibhag is just a vertical line. A rest is indicated with a dash. The register is indicated by placing dots either above or below the swar. Finally complex beats (matras) are indicated by a crescent beneath the notes.

3.3 Humdrum Syntax

Humdrum syntax, developed by David Huron, provides a framework within which more generalized forms of notation, known as representation schemes, can be defined. It is a linear text-based representation for efficient data entry is described.It is an tool to systematically encode the long existing western music notation into a machine readable format. Each scheme consists of a mapping between concepts we wish to represent (signifieds) and how we want to represent them (signifiers). Hudrum syntax imposes two basic constraints: 1) signifiers must be composed of ASCII characters, 2) concurrent events occur at the same horizontal level on the page while sequential events proceed vertically from top to bottom. In practice most Humdrum representations consist of parallel vertical columns. Each column represents sequences moving through time. Records from different columns which are aligned horizontally are interpreted as simultaneous. Different columns can

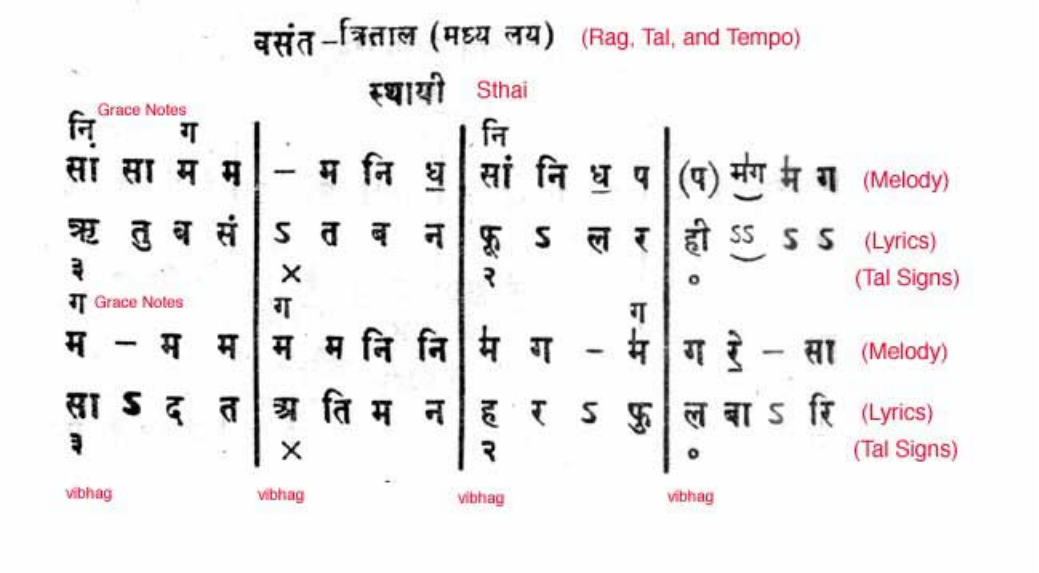

Figure 3: Bhatkande with Annotations

Figure 4: Notational Elements in Bhatkande

represent attributes of a single note or event, or depending on the representation system could represent different notes.

A given representation scheme is composed of three elements: data records, comment records and interpretation records. Comment records are used to annotate encodings and consist of free form text preceded by !!. Humdrum specifies a series of standardized comment records such as author, performer, date, etc. Interpretation records allow us to give information necessary to decode data tokens in a given column. They are of two types: exclusive and tandem interpretations. An exclusive interpretation tells us what scheme is being used and is specified by a double asterisk (**). For example, **kern is a representation that used to encode Western music. Within that scheme, 'A' refers to a pitch. In another scheme, the symbol could be defined to have an entirely different meaning. A tandem interpretation, denoted with a single asterisk (*),

provides additional information that might be used when processing the data. We might, for example, note that the pitches in a given spine are from an equal-

tempered scale by writing *equal-tempered before the start of the data. Tandem interpretations differ from comments in that they are used by other programs to help process the data. If we specify *equal-tempered as a tandem interpretation, we might use it if we wish to convert pitch names to frequencies.

3.4 Intermediate Representation of Bandish

Bhatkhandes notation system consists of symbols for notes indicating their pitch, and symbols for ornaments such as glissandi (meend) and turns (khatka). Note names, written in Devnagari script, are abbreviations of the traditional names given to different pitch-classes. The pitch height is given by a dot that is either present above or below the note, or absent. When absent this is taken to mean the middle octave, whereas the dot above indicates the octave above, and the dot below the octave below. Rests are indicated by the use of a dash (). Note names are indicated by the Hindi character for the traditional names of the scale tones. In the Western alphabet, these are Sa, Re, Ga, Ma, Pa, Dha, and Ni. These are sometimes abbreviated to simply the first letter (e.g. S). The second, third, sixth, and seventh scale degrees have minor or flat forms, much analogous to the Western diatonic scale, which are represented by lowercase letters (e.g. re).

Grace notes (kan swars) are indicated by slightly smaller notes placed immediately above another note for example Ni above Dha on the first line. Slurs above notes indicate a meend, gliding from the note where the slur begins to that above which it ends. Khatkas, which start on the upper neighbor, proceed quickly to the main note and past to the lower neighbor before returning to the main note (a turn in Western music), are indicated by the placement of parentheses around the note. For example (S) would indicate the movement RSNS taken rapidly. The exact neighboring notes taken are dependent on the raag in question. For example, if only the minor seventh scale degree was present, then (S) would indicate SRnS.The placement of the symbols on the page gives information about the duration and sequence of notes, as well as their relationship to the metric scheme. Time proceeds from left to right, skipping to the next line when one is completed, as in standard text. In Bhatkhande's system, beats are separated by spaces. Notes that occur in the span of the same beat are written next to each other without gap, with a slur underneath to indicate they all occur in the same beat.Durations are inferred by assuming that all symbols separated by a space take the space of one beat, and that all symbols within a beat are of equal duration

id: bhatk4663a
antara: Chhin bhangur sab
vol: 4
page: 663
raag: darbari kanada
taal: teentaal
tempo: madhyalaya

$// > PMP - / > nd > ndn/SSS/ > SnSSS/$
$// > PSS > P[n/P] > PMP/ > PMP > Pn/ > Mn- > SRS/$

Each line begins with // and contains the number of beats that are in the rhythmic cycle. For example, in this case, there are 16 beats, corresponding to tintal. Every beat is separated by one or more spaces. When the bandish ends before the end of the cycle or begins somewhere besides the first beat, silent rests are used (=) to fill those beats. The silent rest is used in order to distinguish between rests within the phrase structure and those which fall outside of it. Divisions within the rhythmic cycle, as indicated by vertical lines in Bhatkhande, are indicated with the / marker, making the line easier to read and error check. It can be seen that traditional notation does not fully account for all possible rhythmic figures. If subdivisions of the beat are not equal, then this notation becomes unwieldy. However, both because of the musical context and becuse of the intention of encoding traditional notation, this does not pose a serious problem.

3.5 Musical Notation and the Internationalization of North Indian Music

There have been two overall approaches to the internationalization of north Indian music notation. One approach is to translate everything into staff notation, and the other is to use a Bhatkhande notation, but shift the script to Roman script.

The use of staff notation for Indian music is a very controversial issue. It is true that staff notation has the widest acceptance outside of India. This is no doubt a major advantage. Unfortunately, the use of staff notation distorts the music by implying things that were never meant to be implied.

The biggest false implication of staff notation is the key. Western staff notation inherently ties the music to a particular key. This is something that has never been implied in Indian music. The key is merely a question of personal convenience.

Material is routinely transposed up and down to whatever the musician finds comfortable. Over the years a convention of transposing all material to the key of C has been adopted; unfortunately, this convention is usually not understood by the casual reader.

One other problem associated with staff notation is the implication of equal-temperament. This clearly is not implied in Indian music.

Staff notation is not the only approach to the internationalization of North Indian music, simply writing in Roman script is the another approach. There are advantages and disadvantages to this.

The biggest advantage of writing Bhatkhande notation in Roman script is that it does not distort the original material. Since Bhatkhande's notation was never actually tied to any particular script, it is arguable that this is really no change at all. Furthermore, the widespread acceptance of Roman script, even in India, means that it has a wide acceptance.

However, the use of Roman script / Bhatkhande notation is not without its deficiencies. The biggest problem is that it absolutely requires a firm understanding of the structure and theory of North Indian music. The practical realities of international book distribution and more especially the Internet, means that information should be instantaneously accessible. One should not expect a casual visitor to a website, or a musician browsing through a music book, to invest the energy required to master the Bhatkhande notation.

4 A Tool to Synthesize Music from Annotated Notes

4.1 Choice of MIDI as our output file format

MIDI is a technical standard that describes a protocol, digital interface and connectors and allows a wide variety of electronic musical instruments, computers and other related devices to connect and communicate with one another.A single MIDI link can carry up to sixteen channels of information, each of which can be routed to a separate device. MIDI carries event messages that specify notation, pitch and velocity, control signals for parameters such as volume, vibrato, audio panning, cues, and clock signals that set and synchronize tempo between multiple devices. These messages are sent to other devices where they control sound generation and other features. This data can also be recorded into a hardware or software device called a sequencer, which can be used to edit the data and to play it back at a later time Before electronics, music was expressed exclusively as

written symbols. By translating musical parameters into digital data, MIDI can express not only the types of musical events written into sheet music, but other parameters as well (such as the amount of pitch bend or degree of vibrato).

4.1.1 The MIDI interface

The MIDI interface has three ports. MIDI: IN, OUT and THRU. The IN port allows data to be received by the machine. The OUT port is used for transmitting data. The THRU port creates a replica of the input signal. This is used to connect more than one MIDI device. There are many reasons why MIDI is used by such a large number of people. The most significant being that it is the system that all keyboard manufacturers have used in the design of their digital synthesizers.

4.1.2 Composing music using MIDI

Composing music using MIDI is very simple. What you play on the keyboard is recorded by the sequencer. For a single keyboard up to two tracks can be recorded simultaneously. When a key is pressed the data corresponding to the key being pressed is sent to the sequencer via the OUT port on the keyboard. Once the sequencer has this information it can be converted into musical score. MIDI plays music not by digitizing the actual audio coming out of all the electronic instruments but rather by recording the MIDI OUT of all these messages. MIDI files contain no sound. They only contain performance data.

4.2 The MIDI format

The information to MIDI is transmitted in "MIDI messages", which can be thought of as instructions which tell a music synthesizer how to play a piece of music. A MIDI message is made up of an eight-bit status byte which is generally followed by one or two data bytes. There are a number of different types of MIDI messages. The synthesizer receiving the MIDI data must generate the actual sounds.MIDI controller is a device which is played as an instrument, and it translates the performance into a MIDI data stream in real time (as it is played). A MIDI sequencer is a device which allows MIDI data sequences to be captured, stored, edited, combined, and replayed. The MIDI data output from a MIDI controller or sequencer is transmitted via the devices MIDI OUT connector. The MIDI data stream is usually originated by a MIDI controller, such as a musical instrument keyboard, or by a MIDI sequence.

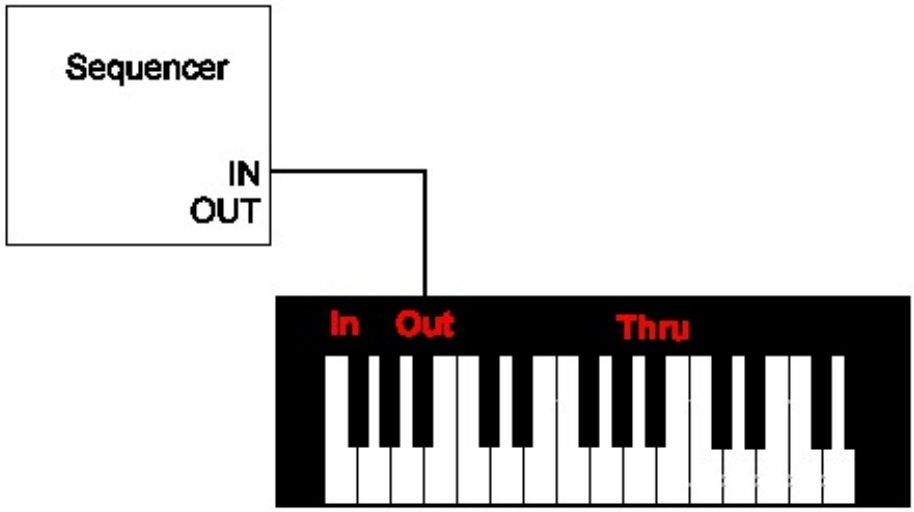

Figure 5: Sequencing the data from a keyboard

4.2.1 Note ON/OFF Messages

In MIDI systems, the activation of a particular note and the release of the same note are considered as two separate events. When a key is pressed on a MIDI keyboard instrument or MIDI keyboard controller, the keyboard sends a Note On message on the MIDI OUT port. The keyboard may be set to transmit on any one of the sixteen logical MIDI channels, and the status byte for the Note On message will indicate the selected Channel number. The Note On status byte is followed by two data bytes, which specify key number (indicating which key was pressed) and velocity (how hard the key was pressed).

The key number is used in the receiving synthesizer to select which note should be played, and the velocity is normally used to control the amplitude of the note. When the key is released, the keyboard instrument or controller will send a Note Off message. The Note Off message also includes data bytes for the key number and for the velocity with which the key was released. The Note Off velocity information is normally ignored.

4.2.2 The Pitch Bend Message

The Pitch Bend message includes two data bytes to specify the pitch bend value.The pitch bend information is used to modify the pitch of sounds being played on a given Channel.The two bytes of the pitch bend message form a 14

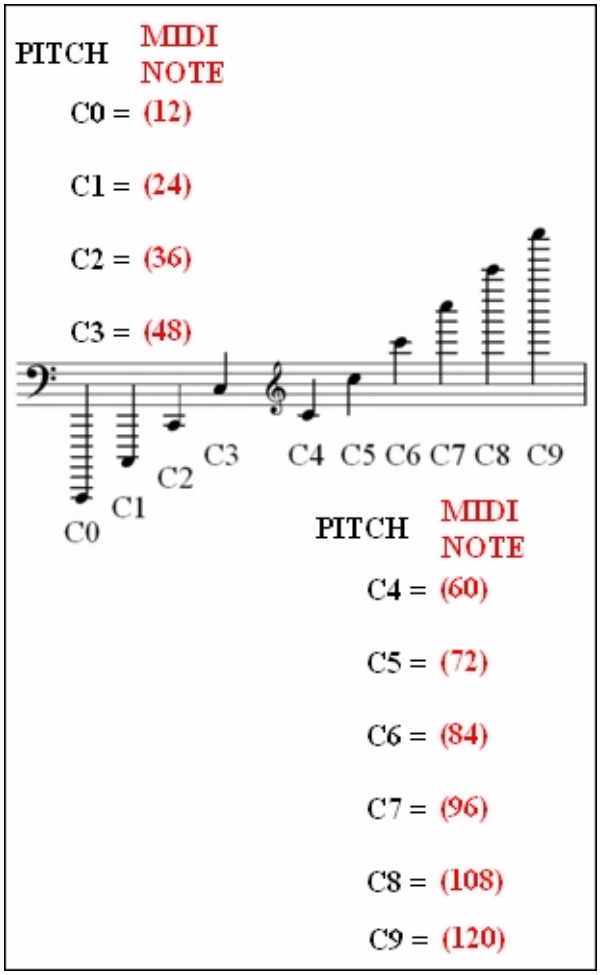

Figure 6: MIDI notes corresponding to different octave values

bit number,0 to 16383. The value 8192 (sent, LSB first, as $0x00, 0x40$), is centered, or "no pitch bend." The value $0(0x00, 0x00)$ means, "bend as low as possible," and, similarly, $16383(0x7F, 0x7F)$ is to "bend as high as possible." A Single Pitch Bend event is used to "smoothly" change the pitch of a note. Unlike playing two separate notes, the pitch bend will make the transition between the notes blend. This is analogous to a guitarist removing his finger from the fret but not striking the string which causes the note's pitch to suddenly but smoothly change to a lower level.

Varying the severity of Pitch Bend By default, the maximum pitch bend value is very slight only about one full step up or down. This is not enough for most tunes, especially for midis with guitarists, who frequently bend their strings

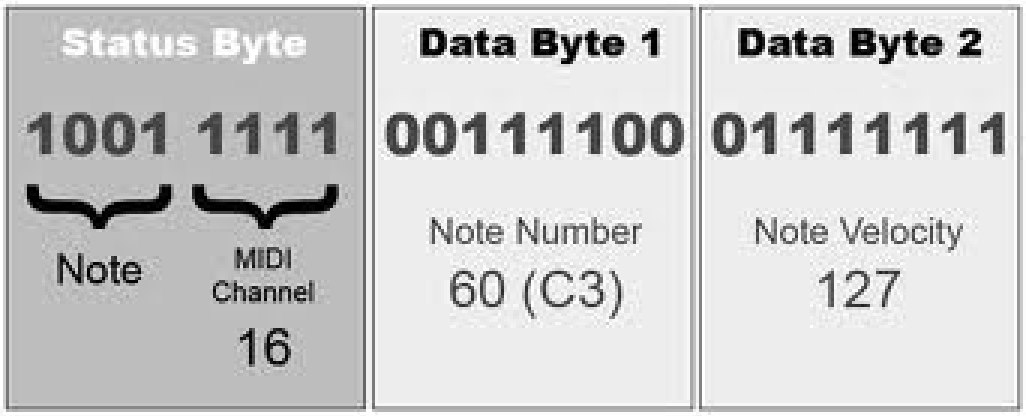

Figure 7: Structure of a MIDI message

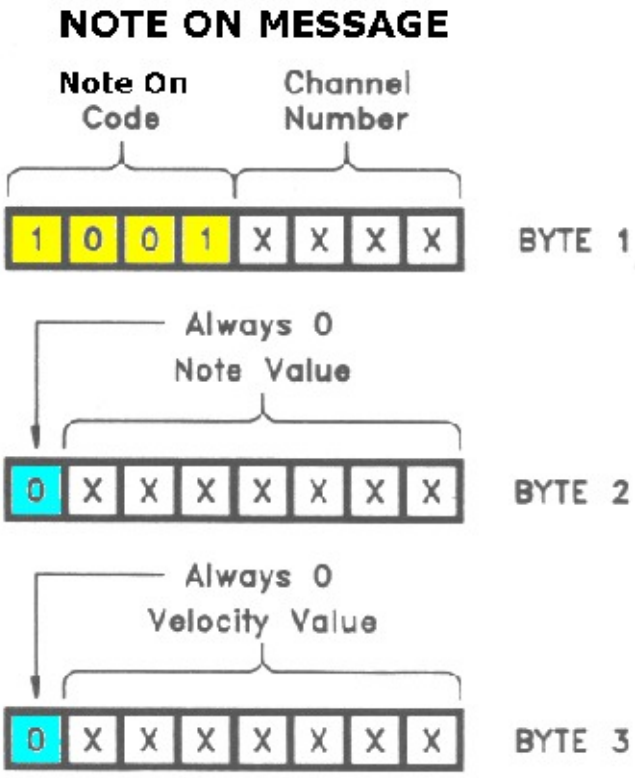

Figure 8: Structure of a Note ON message

and use a bar for more extreme bends. The amount of "severity" of the pitch bend is set by events that set the value of three midi controllers and it must be done in this exact order:

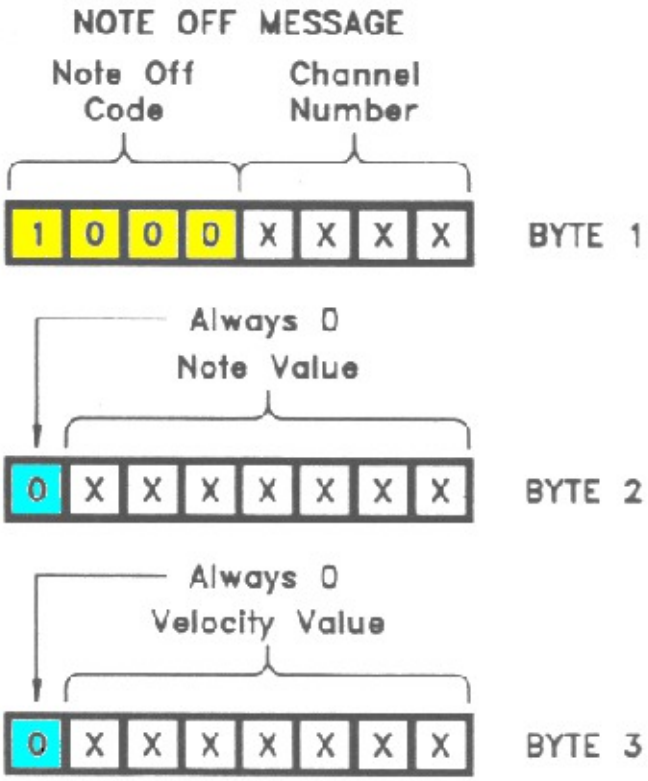

Figure 9: Structure of a Note OFF message

1) Registered Param LSB (set to zero) - enter this once at the beginning of the song.
2) Registered Param MSB (set to zero) - enter this once at the beginning of the song, right after the Registered Param LSB entry.
3) Data Entry MSB (set between 0 and 127) - but in reality 0 to 24 is plenty, and the most common value is between 0 and 6. Enter this once at the beginning of the song, right after the Registered Param MSB entry. Then whenever you want to change the severity of the pitch bend, make another entry just before that section.
So, to sum it up, Pitch Bend is an event - it tells the midi engine to change the pitch of a note whereas Data Entry MSB is a controller - it controls the severity of Pitch Bend events and we need to issue a "Controller Event" to set the value of the Data Entry MSB.

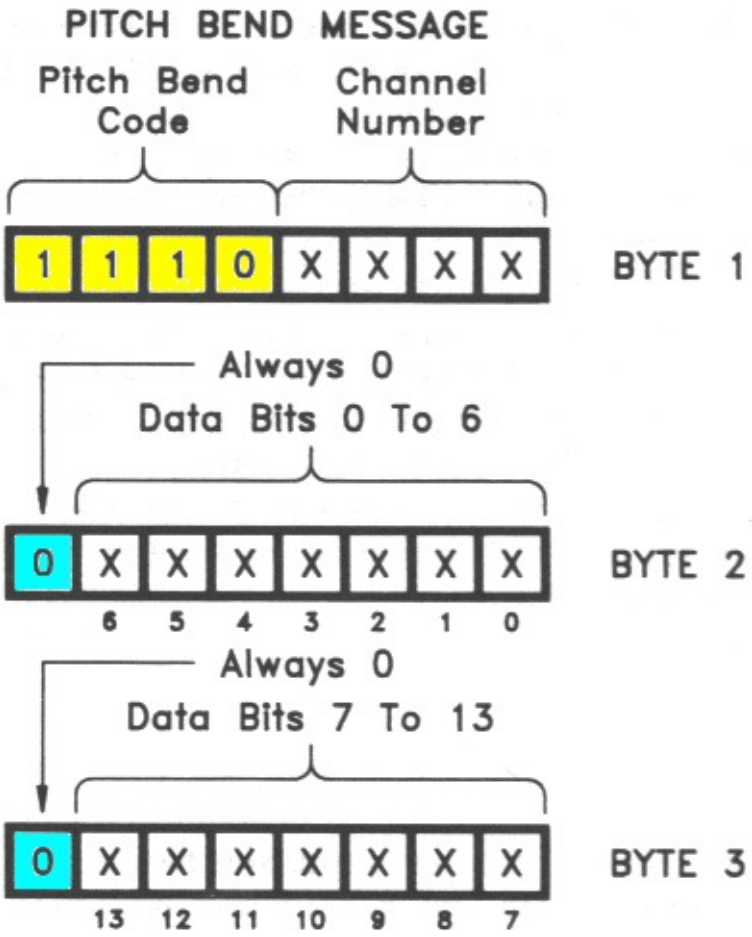

Figure 10: Structure of a Pitch Bend message

4.2.3 MIDI Vs Other Formats(MP3,WAV)

In MIDI we can store the messages generated by many instruments in one file and yet be able to easily pull apart messages on a per instrument basis because each instruments MIDI messages are on a different MIDI channel.Hence we can easily examine and edit the data corresponding to every instrument. Contrast this with digitizing the audio output of all the instruments. The audio signals from different instruments are mixed before digitizing them. These signals are analog and once we mix these signals together it takes massive computational power to seperate this signals. So, we lose control over each instruments output and only have the cumulative result available to us.

Size Difference in MIDI and other formats It typically takes much more storage to digitize the audio output of an instrument than it does to record an instruments MIDI messages. For Example-To record a whole note. With MIDI, there are only 2 messages involves. There is a Note On message when you sound the note and then the next message doesnt happened until we finally release the note. We could hold down that note for an hour and still we would still have only 2 messages. Contrast this to digitizing the audio signal, we would have to be recording all of that time when that note is sounding.So the resultant output will have a file size much much larger than the corresponding MIDI output.

WAV format is the most detailed and rich of the available formats. The details are recorded at the chosen bit rate and sampling speed, and it's all done without compression schemes.It takes up huge amounts of memory in the process. Four or five minutes of WAV sound can consume about 50MB of memory.The MP3 format is just the same as the WAV format as in it also contains the digitized audio data with the only difference being that MP3 uses compression to squeeze the data down in size, resulting in a typically much smaller file size.A MIDI file contains hundreds of instructions that tell the sound modules how to reproduce every single, individual note and nuance of a musical performance but its size is tiny as compared to the other formats.

5 Data Structures Used

We use C++ classes for reading/writing standard MIDI files. The main classes we have used are:

MidiFile-Primary class, responsible for reading and writing MIDI files.
MidiEventList-Responsible for managing MidiEvent lists for tracks within the MidiFile class.
MidiEvent-MidiMessage plus a timestamp and other bookkeeping variables.
MidiMessage-Raw MIDI protocol message. Inherited by the MidiEvent class.

5.1 MidiFile class

The MidiFile class is an interface for reading and writing Standard MIDI files. MIDI file tracks are stored as a list of MidiEventList objects, which in turn are lists of MidiEvents The MidiFile class can be considered to be a two-dimensional array of events that are accessible with the [] operator. The first dimension is the track index, and the second dimension is the event index for the given track. Thus, midifile[2][15]would return the 16th event in the third track.

MidiEvents consist of a list of MIDI message bytes along with timing and other variables. For example, the construct midifile[2][15][0] would return the MIDI command byte for the 16th message in the third track, and midifile[2][15].tick would return the timestamp for the event (either in delta or absolute tick values, see the tick-related functions described further below).

5.1.1 Reading/writing functions

The two main member functions for reading and writing Standard MIDI Files are read and write. The argument to these functions can be either a filename or an input/output stream object. The statusfunction can be called after reading or writing to determine if the action was successful. Two additional functions, called writeHex and writeBinasc can be used to print ASCII hex codes that determinstically represent a binary Standard MIDI File. The read function will transparently parse either binary MIDI files, or their ASCII representations in one of these two formats.

read-Read a Standard MIDI File in binary or ASCII formats.
write- Write a Standard MIDI file from the MidiFile contents.

status-Returns true if last read/write was successful.
writeHex - Print MidiFile as a list of ASCII hex bytes.
writeBinasc - Print ASCII version of MIDI file in binasc format.

5.1.2 Track-related functions

The MidiFile class contains a list tracks, each stored as a MidiEventList object. The [] operator accesses each list, and the getTrackCount function will report the current number of tracks in a MidiFile object.

The tracks in a MidiFile can reversibly be merged into a single event list by calling the joinTracksfunction. This will cause the getTrackCount function to report that there is one track in the file, and if the file is written in this state, it will be saved as a type-0 MIDI file (from which the multi-track state will not be recoverable). Before tracks are joined, the events in each track must be in correct time sequence, so the sort function may need to be called before joining the tracks. The hasJoinedTracks and hasSplitTracks functions can be used to detect the current state of the tracks in a MidiFile. By default, a MidiFile will be in the split state. The list of functions are:
operator[] -Returns the list of events for a track.
getTrackCount- Return the number of tracks in the MidiFile.

joinTracks -Merge all tracks into a single stream of events.
splitTracks-Separate events into their original track configuration.
hasJoinedTracks - Return true if MidiFile is in the joined-track state.
hasSplitTracks-Return true if MidiFile is in the split-track state.

When a MidiFile is in the joined state, the original track is stored in the track variable of each MidiEvent. The getSplitTrack function returns the track index for when a MidiFile is in the split state.Here are functions which relate to adding, deleting and merging tracks:

getSplitTrack- Get the track number of an event when MidiFile is in split state.
addTrack-Added one or more empty tracks to the MidiFile.
deleteTicks -Remove the specified track from the MidiFile.
mergeTracks-Combine two tracks into a single track.

When a MidiFile is in absolute-tick mode (see further below), the tick variable

of MidiEvent objects in the MidiFile are the total number of ticks since the start time for the file. In the absolute-tick mode, MidiEvents can be added to tracks in non-sequential order. To re-arrange the events into proper time order (such as before writing the file), use the sortTracks function, which will sort all tracks in the MIDI file, or sortTrack function, which will sort a particular track by its index number.

5.1.3 Time-related functions

MidiEvents stored in a MidiFile structure contain two public variables related to time:

int MidiEvent::tick- Quanta time-units describing durations in Standard MIDI Files.
double MidiEvent::seconds- Interpreted physical time units in seconds calculated bydoTimeInSecondsAnalysis() from .tick data and tempo meta-messages stored in the MidiFile.

5.1.4 Event tick interpretation

Tick values on MidiEvents can be set to two types of states: (1) delta time, which indicate the number of ticks to wait from the last event in the track before performing the current event, and (2) absoute time, where the tick value represents the cumulative tick time since the start of the MidiFile until the performance time of the current event. Standard MIDI Files store tick times as delta ticks, but it is often more useful to manipulate event data with absolute ticks. The absoluteTicks and deltaTicksfunctions switch the event tick values within the MidiFile between these two modes.The isDeltaTime and isAbsoluteTime functions can be used to check which mode in which the event ticks are currently given.

absoluteTicks - Convert event delta tick values into absolute ticks.
deltaTicks-Convert timestamps into delta times.
isDeltaTime- Returns true if event ticks are in delta-time mode.
isAbsoluteTime- Returns true if event ticks are in absolute-time mode.

Tick values are symbolic time units, such as rhythms in music notation. For example quarter notes do not have a specific duration in seconds until a tempo is applied to the rhythm. Within Standard MIDI Files, there is a field which specifies how many ticks represent a quarter note. This convserion value can be

read or set with the following two MidiFile member functions:

getTicksPerQuarterNote Return the delta time ticks units which represent a quarter note.
setTicksPerQuarterNote Set the ticks-per-quarter note value.

5.1.5 Physical time of events

doTimeAnalysis-Calculate the time in seconds for each event in the MidiFile.
getTimeInSeconds-Return the time in seconds for the specified message.
setMillisecondTicks-Set the ticks per quarter note value in the MIDI file header to milliseconds.
getAbsoluteTickTime-Convert a time in seconds into an absolute tick time.
sortTrack-Sort a track in terms of timestamps of the events.
sortTracks-Sort all tracks in the MidiFile by event timestamps.

Other functions
addEvent-Add a new MidiEvents to the end of the specified track.
addMetaEvent- Insert a meta message event.
addPitchBend- Insert a pitch bend message into the given track.
allocateEvents-Allocate extra space for a track event list.
clear- Clear all tracks, leaving one track with no data in it.
extractVlvTime- Extract a VLV value from the input stream.
getEvent- Return the event at the given index in the specified track.
getEventCount-Return the number of events in the specified track.
getFilename- Return the filename for the MidiFile.
makeVLV- Convert an integer into a list of variable length value bytes.
writeHex- Print MidiFile as a list of ASCII hex bytes.
setFilename-Set the filename of the MidiFile.

5.2 MidiEventsList

The MidiEventList class manages MidiEvent objects for a MidiFile track.
Public functions **append** - Add a event to the end of the list.
clear -Clear all events from the list.
getSize - Return the number of events in the list.
reserve-Pre-allocate space for events in the list.
operator[] -Access a MidiEvent in the event list.
clearLinks-Remove all note-on/note-off links.

detach -Clear event list without deallocating events.
inkNotePairs-Match note-ons and note-offs.

5.3 MidiEvents Class

The MidiEvent class is a MidiMessage plus an added timestamp public variable, .time. The MidiEvent class also adds other support variables such as .track which store the track number in the MIDI file that the MidiEvent occurs in.

Public functions **getDurationInSeconds** -Get the duration of a note in seconds.
getLinkedEvent-Returns a linked event (such as for note-on/note-off pair).
getTickDuration -Get the duration of a note in terms of MIDI ticks.
isLinked- Returns true of the message is linked to another event.
linkEvent-Links one event with another (such as note-ons/note-offs).
operatorEQUALS-Copy the contents of another MidiEvent or MidiMessage.
unlinkEvent - Disassociates any MidiEvents linked to this one.

5.4 MidiMessage Class

The MIDI message class contains the raw MIDI bytes which are stored in a Standard MIDI File. The class is inherited from vector, so all STL vector class functions can be used in this class. The MidiEvent class inherits these function from the MidiMessage class.

The first byte in the MIDI message is expected to be a command byte, which is a byte in the range from 0x80 to 0xff. Running status is not allowed in MidiMessage class data, and any missing running status byte will be inserted into the message when reading from a MIDI file. Currently MIDI files cannot be written in running status mode by the MidiFile class, but if that is needed, then the MidiFile class will be responsible for removing the running status command byte as necessary from the MidiMessage data as it is being written to a file.

The command byte of a MIDI message is in the range from 0x80 to 0xff hexadecimal (128-255 decimal). Each command converts 16 bytes, such as 0x80 to 0x8f for the note-off command. The "8" for note-offs is the *command nibble", which are the top four bits of the byte. The bottom nibble indicates the MIDI channel of the command, and ranges from 0x0 to 0xf hex (0 to 15 decimal). Thus, 0x80 is a note-off command on the first channel, 0x81 is the note-off command on the second channel, ..., and 0x8f is the note-off command on the sixteenth channel.

There are seven MIDI commands from 0x80 to 0xe0 command nibbles, plus 16 miscellaneous commands starting with a 0xf0 nibble that don't refer to MIDI channels in the bottom nibble. Each command has an expected number of parameter bytes after it, which are in the range from 0x00 to 0x7f hexadecimal (0 to 127 decimal). Here is a table summarizing the seven main MIDI commands and their required parameter count.

Command nibble , Command name, Parameter count ,Parameter meanings

0x80 Note off 2 key, off velocity
0x90 Note on 2 key, on velocity
0xA0 Aftertouch 2 key, pressure
0xB0 Continuous controller 2 controller number, controller value
0xC0 Patch change 1 instrument number
0xD0 Channel pressure 2 key, off velocity
0xE0 Pitch-bend 2 LSB, MSB

6 Results Obtained

(Please also see Page 37,38,39,40)
We were able to parse the input in the Indian notation (of the form Sa, Re, Ga etc.) and map it into its equivalent Western Notation (of the form C, C#, D etc) which is the backend used by the tool. We were also able to specify the base note (which is C by default) for our input and adjust the notes accordingly to get the correct mapping. We could also specify the tempo and beat of our audio file as well as model the octave, the volume and the duration of each individual note which make up the final sequence. Fianlly we can generate the MIDI file based on the notation provided.

In the reverse direction, given a MIDI file we were interested in renedering in our notation the notes and other information about the music. We were able to do that using the fact that MIDI is designed in a structured way so we could cover all the notes at various channels , their volume, pitch, key, timbre etc and present the information as the output.

We were also interested in inquiring whether the MIDI format is sufficient to cover all the varied aspects and the many ornamentations that are inherent in Hindustani Classical Music.As we were able to successfully generate the pitch bends in MIDI format and also the other ornamentations, wit gives us confidence to conclude that ornamentations like Meend and Gamak etc could be modelled accurately in the MIDI format.

7 Conclusion and Future Work

we were able to generate the pitch bends we had been looking for. We created several .mid files corresponding to different level of pitch bends. The controller for Data Entry MSB allows us a range of 0-127 for pitch bending but we found we could generate the required bends in the range of 0-24 easily. The larger values served as an example for the extreme pitch bends which are also supported by MIDI. We generated the bends by sending a series of pitch bend events between the time a note was turned on and off. We experimented with several durations of the bend messages and generated the corresponding MIDI files. Pitch bends of shorter duration resulted in a faster bend as compared to those which took longer time to bend by the same amount. We could concatenate several pitch bends events to get different resulting sounds. So finally, we were able to successfully generate the pitch bends in MIDI format which gives us confidence to conclude that ornamentations like Meend and Gamak etc could be modelled accurately in the MIDI format.

We have been able to develop our tool which is able to take annotated notes and generate the MIDI file for it. We are able to handle pitch bending which is essential in many movements of Hindustani Classical Music. In the future we would like to extend the tool so as to incorporate even more features and alankars (ornamentations) of the Hindustani Music such as 'gamak', 'aandolan', 'khatka' etc. Once we are able to handle that, we would like to generate the complete MIDI file for a given rendition and play it on a synthesizer and be able to identify it and score it based on how closely it follows the grammar of the Raga. . Further we would like to use machine learning techniques to learn feature based classifiers that explain the classifications of the ragas on the various parallel classification schemes (such as season, time of day, mood).

References

[1] The official MIDI 1.0 detailed specification.

[2] Colin de la Higuera. "Grammatical Inference: Learning Automata and Grammars"

[3] ITC Sangeet Research Academy. http://www.itcsra.org/

[4] Dipanjan Das and Monujit Chaudary. "Finite state models for generation of hindustani classical music."

[5] H..V. Sahasrabuddhe. "Analysis and synthesis of hindustani classical music."

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%
%% DATA LEGEND                          %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%
%%Filename: test.mid
%%Ticks per quarter note: 120
%%Time units used in column 1: beat
% 1 opcodes are present in the data:
%   opcode 1000 = note
%     column 1 = start time of note
%     column 2 = opcode for note
%     column 3 = duration of note
%     column 4 = MIDI key number
%     column 5 = MIDI attack velocity
%     column 6 = MIDI channel
%     column 7 = MIDI-file track number
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%
```

Figure 11: Input file:test.mid

```
NOTE = 1000;
data = [
0.00,   NOTE,  1.00,   60.00,  64.00,  0.00,
1.00;
1.00,   NOTE,  1.00,   0.00,   64.00,  0.00,
1.00;
2.00,   NOTE,  0.50,   62.00,  64.00,  0.00,
1.00;
2.50,   NOTE,  0.50,   64.00,  64.00,  0.00,
1.00;
3.00,   NOTE,  1.00,   0.00,   64.00,  0.00,
1.00;
4.00,   NOTE,  0.33,   66.00,  64.00,  0.00,
1.00;
4.33,   NOTE,  0.33,   68.00,  64.00,  0.00,
1.00;
4.67,   NOTE,  0.33,   70.00,  64.00,  0.00,
1.00;
5.00,   NOTE,  1.00,   0.00,   64.00,  0.00,
1.00;
6.00,   NOTE,  1.00,   68.00,  64.00,  0.00,
1.00;
7.00,   NOTE,  1.00,   66.00,  64.00,  0.00,
1.00;
8.00,   NOTE,  0.25,   70.00,  64.00,  0.00,
1.00;
8.25,   NOTE,  0.75,   68.00,  64.00,  0.00,
1.00;
9.00,   NOTE,  1.00,   0.00,   64.00,  0.00,
1.00;
10.00,  NOTE,  0.25,   71.00,  64.00,  0.00,
1.00;
10.25,  NOTE,  0.75,   70.00,  64.00,  0.00,
1.00;
11.00,  NOTE,  1.00,   0.00,   64.00,  0.00,
1.00;
12.00,  NOTE,  1.00,   71.00,  64.00,  0.00,
1.00;
13.00,  NOTE,  1.00,   70.00,  64.00,  0.00,
1.00;
14.00,  NOTE,  1.00,   68.00,  64.00,  0.00,
1.00;
15.00,  NOTE,  1.00,   72.00,  64.00,  0.00,
1.00;
];
```

Figure 12: Notes extracted from MIDI file

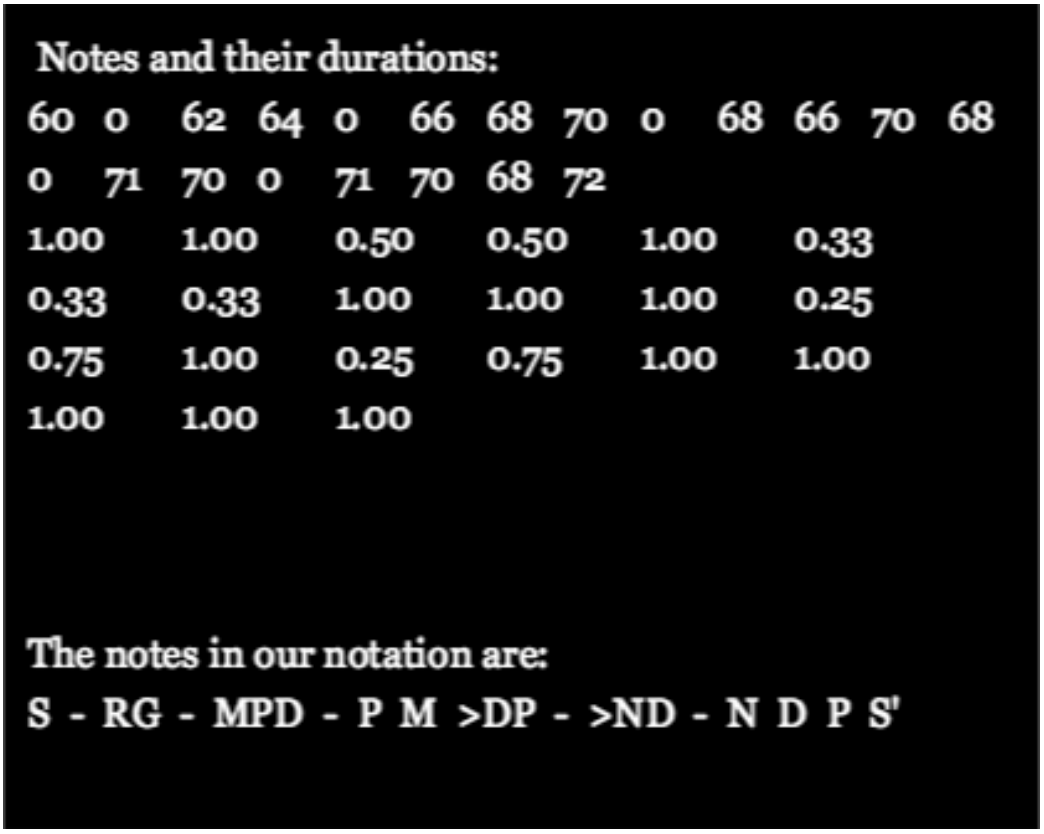

Figure 13: Final Output in our annotation.

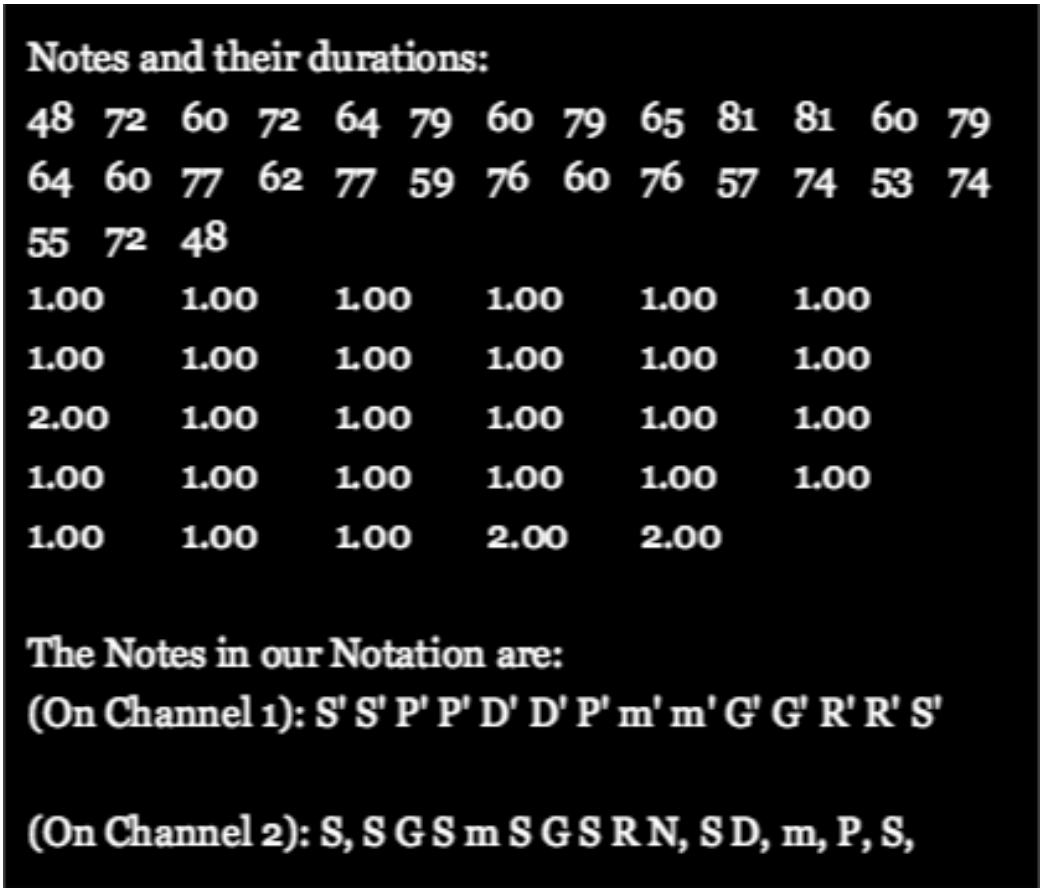

Figure 14: Multiple Channels Case