# Audio Applications of the Sound Description Interchange Format Standard

Matthew Wright, Amar Chaudhary, Adrian Freed, Sami Khoury, David Wessel
CNMAT, UC Berkeley
1750 Arch St., Berkeley, CA 94709
{matt,amar,adrian,khoury,wessel}@cnmat.berkeley.edu
(510) 643-9990

## Abstract

The Sound Description Interchange Format (SDIF) is a recently established standard for the interchange of a variety of sound descriptions including spectral, time-domain, and higher-level models. SDIF consists of a specified data format framework and an extensible set of standard sound descriptions and their official representations. We begin with the history and goals of SDIF, followed by the specification of the data format and the standard sound descriptions. Then we describe some current applications of SDIF and conclude with a look at the future of SDIF.

## 1. Introduction

Now that digital representation of sound recordings have approached perceptual transparency and are close in efficiency to optimal codings, it is time to focus on new representations which exploit higher-level abstract models. Although these models may offer coding efficiencies, their real value lies in the mutability they afford. Well-known applications of mutable representations include morphing, pitch shifting and time stretching. Interesting new applications include scalable coding and transmission for the Internet and location-based spatialization [1].

Interchange formats for sound files, the time-domain samples used as the input to analysis software, are plentiful [2] and will not be discussed further. File formats for representing analyzed sound are also plentiful but existing formats are specific to particular models, making it impossible to translate among formats. This frustrates ongoing analysis/synthesis research and hampers commercial development of these ideas. SDIF is designed to represent a diverse range of models.

Sections 2 and 3 review existing analysis/synthesis models and the various file formats associated with them. Section 4 gives the history of SDIF, and section 5 lists the goals that guided SDIF's design and development. Section 6 specifies SDIF's extensible framework format and section 7 defines the library of standard sound descriptions in terms of this framework. Section 8 describes some of the applications of SDIF, and we conclude with a look at SDIF's future.

## 2. Review of Primary Analysis/Synthesis Models

The mid-1960s witnessed the birth of a variety of approaches to the analysis and synthesis of music and speech using computers. Jean-Claude Risset in association with Max Mathews at Bell Labs performed pitch-synchronous harmonic analysis of trumpet tones [3-5]. This analysis technique assumed that the sound was quasi-harmonic and produced output data formatted in such a way that the trumpet tones could be accurately resynthesized using an additive synthesis technique. The initial output of their analysis program consisted of sampled functions of time for the amplitudes and frequencies of a number of sinusoidal components. These curved functions were then approximated with linear segments and formatted in such a way that the sequence of amplitude and time break points controlled linear ramp generators for amplitudes of the sinusoidal components. Moorer and Grey followed a similar strategy [6-9].

Analog vocoders were widely used for speech modeling in the 1960's, but the development of the digital phase vocoder [10] was the key to high quality analysis/resynthesis. This development depended on the availability of faster digital computers and the rediscovery in 1965 of the FFT [11]. Moorer was the first to adapt these ideas for use with music [12].

Linear predictive coding of speech was also developed at the end of the 1960's [13, 14] and later applied to the sung voice and other musical sounds [15-17].

Sinusoidal representations of speech [18] have widely influenced both the speech and computer music research communities.

Researchers have started to explore alternate time-frequency representations [19] and wavelets [20] for audio and music applications.

All-pole or damped sinusoidal models [21] arise in many fields from Nuclear Magnetic Resonance (NMR) imaging [22], through modal analysis [23, 24] to earthquake modeling [25]. An efficient resonance analysis technique was developed for musical sounds in 1986 [26].

## 3. Review of Tools and File Formats

### 3.1 James Beauchamp's SNDAN

The publicly available SNDAN analysis/synthesis package [27, 28] from James Beauchamp's group at the University of Illinois Urbana/Champaign (UIUC) can perform either a pitch-synchronous short-time Fourier (Phase Vocoder) analysis or analysis based on the McAulay-Quatieri algorithm, each with an associated output file format. The two file formats share a common file header with 20 fixed fields that include a characterization of the analyzed sound (sample rate, length, maxiumum amplitude), parameters of the analysis stage (presumed fundamental frequency, time between analysis frames, FFT length), and ASCII text "musicological" information such as performer's name, instrument, pitch and dynamic played, and the date of the recording.

SNDAN's phase-vocoder analysis produces ".pv.an" files, representing a fixed number of harmonic partials of a fixed fundamental frequency. The fundamental is given as a parameter to the analysis; the frequencies of the analysis bins are integer multiples of this given fundamental because of band-limited interpolation of the input signal. The file header contains this fundamental. The body of the file consists of initial phases (in radians) for all partials, followed by a series of frames containing the frequency deviation and amplitude for each partial. These data are represented as 32-bit floating point numbers, or in an optional "compact" 16-bit format.

SNDAN's McAulay-Quatieri-style analysis produces ".mq" files, representing a time-varying number of sinusoidal tracks with frequencies of any relationship. The data consists of a series of frames, each containing a collection of partials with 32-bit floating point amplitude, frequency, and phase, and a 16-bit integer "link" field to associate partials with each other across frames.

The SNDAN package includes a program that estimates fundamental frequency over time from a ".mq" file [29], and then reduces it to a series of harmonic partials. The result uses the same format as ".pv.an" files, but has the suffix ".mq.an" by convention.

### 3.2 Lemur (Kelly Fitz, et al)

Another group at UIUC under Dr. Lippold Haken has released a sinusoidal analysis/synthesis package called Lemur [30, 31]. Lemur is based on Maher's and Beauchamp's extension of the McAulay-Quatieri algorithm [32]. Lemur uses its own data file format to represent analysis results [33]. The file header gives the length and sampling rate of the analyzed file, the FFT size, parameters of the analysis window, and other analysis parameters. The bulk of the data is represented with a frame structure containing amplitude and frequency, but not phase, for each peak. There are also some additional fields, e.g., to support morphing [34] and spectral line-broadened sinusoids [35]. Other research in the Lemur group has in the past resulted in special file formats on almost a per-project basis, although today only Kyma's .spc files and the standard Lemur file format are in use.

Lemur also has a format for "Synthesis Control Files" to impose time-varying parameters for time scaling, frequency scaling, frequency shifting, and magnitude scaling of all partials or selected partials. Time values given in Lemur Synthesis Control Files are always relative, since the length of the control function is always scaled to match the length of the file it modifies.

### 3.3 IRCAM

The Analysis/Synthesis team at IRCAM under Xavier Rodet and Phillippe Depalle has been active in developing sound analysis/synthesis software for many years [17, 36-42]. Here is a partial list of the many file formats used internally by IRCAM:

- Fundamental frequency estimates use ".f0" files, an ASCII format with two columns of numerals: time (in seconds) and fundamental frequency (in Hertz).

- A variant of ".f0" files called ".F0" with a capital "F" had two additional columns: an amplitude estimate (linear), and a confidence factor for the fundamental frequency estimate (0 to 1).
- FFT results were stored in ".i" files, in a binary format.
- Detected spectral peaks were stored in ".pics" files, an ASCII format organized in frames. Each frame applied to a given instant of time and gave the parameters of all the spectral peaks detected at that time, including frequency (in Hertz), amplitude (linear between 0 and 1), and phase (in radians).
- Additive synthesis data, i.e., parameters for a collection of sinusoids over time, were stored in ".format" files, another ASCII text format similar to ".pics" files. Each sinusoid in a ".format" file had an index number that was the same in each frame, allowing parameter values to be matched over time to form a "track."
- A variant of "format" files called ".fmt" files had the same data in a binary format.
- Another variant of "format" files, called ".fmte," had an additional field for "voicing amount" from 0 to 1.

IRCAM's AudioSculpt program [43] incorporated the "Super Vocodeur de Phase" phase vocoder [44] into a graphical time/frequency editor for the Macintosh. AudioSculpt can do FFT-based and LPC-based analysis, each with an associated output format. A unique feature of AudioSculpt is the "output format" dialog box, which allows users to select which fields will appear in the text file representing the analysis result.

IRCAM's Lisp-based PatchWork program from the Musical Representation team can be used to transform and synthesize analysis results [45]. PatchWork reads text files containing nested list structure delimited by parentheses. AudioSculpt can output certain analyses to PatchWork's file format.

### 3.4 CNMAT

CNMAT inherited a version of IRCAM's sinusoidal analysis tools in the early 1990s, along with this collection of file formats. Research on using neural networks to represent additive synthesis data [46, 47] resulted in a custom format for neural network architectures and weights called ".wt" files.

Michael Goodwin and Adrian Freed developed analysis and synthesis respectively for noise modeled using spectral envelopes sampled inside equal rectangular frequency bands (erb) [48]. This used a file format called "erb" files that gave the amplitudes inside each band over time.

### 3.5 Resonance Models at IRCAM and CNMAT

IRCAM's work with resonance models grew from singing voice research [17, 49] based initially

on tracked formant analyses. A (frequency, amplitude, formant bandwidth) triplet was used to represent each formant. At IRCAM and CNMAT a plethora of variations on these triplets and associated file formats have grown from this work including representations as lisp lists, in binary MacMix files [50] and Max/MSP collections [51]. The frequency parameter is usually specified in Hz, but radians/second is common. The initial amplitude is usually dimensionless but may be represented relative to a reference amplitude, i.e., in dB. The third parameter is variously represented as a bandwidth, damping factor ("Q"), or decay rate.

### 3.6 SMS

The Spectral Modelling Synthesis (SMS) system [52], originally developed from Xavier Serra's dissertation work at Stanford [53], and now a project of his group in Barcelona, used its own custom file format. An "SMS" file contained both the analyzed sinusoidal partials and a parametric model of the residual stochastic component. Each frame had the same number of sinusoids, and the frames were equally-spaced in time, with the frame rate, in Hertz, given in the header. The header also contained the sampling rate of the original sound, the magnitude threshold used in the analysis, the percentage of the residual with respect to the original, the average amplitude and frequency of the analyzed sound, pointers to the frames at the boundaries of the steady state portion of the sound, the spectral envelope of the partials, and an arbitrary ASCII text field for descriptive information. Each frame contained the frequency, amplitude, and phase for each partial, as well as gains and filter coefficients for the stochastic part.

### 3.7 Jean Laroche

Jean Laroche's analysis/synthesis research [42, 54-57] covered damped sinusoidal and sinusoid plus noise models and used custom file formats whose details have been forgotten [58].

### 3.8 Time-Frequency Images

Greg Wakefield's group at The University of Michigan has used various file formats based on a sequence of 2D matrices representing time-frequency "images" [59].

### 3.9 Csound and SoundHack Spectral Data Files

The Csound programming language [60, 61] includes an FFT-based phase vocoder named "pvoc" that reads its input from a file. This file begins with a header that contains the original sampling rate, FFT size, analysis frame rate and overlap factor. The body of the file is a sequence of short-time Fourier transform frames sampled regularly in time. Each frame contains magnitude and an instantaneous frequency estimate (phase increment) for "the first N/2 + 1 bins output by the transform."

Tom Erbe's SoundHack program [62] does FFT-based spectral analysis and resynthesis. SoundHack's spectral data file format is a variant of Csound's, except that SoundHack stores the phase in each bin rather than the instantaneous frequency.

## 4. The History of SDIF

SDIF began as a collaboration between Xavier Rodet of IRCAM and Adrian Freed of CNMAT. Discussions with scholars and vendors at the 1995 International Computer Music Conference (ICMC) in Banff confirmed the need for this standard in the academic and commercial communities. Adrian Freed coined the name and invited ICMC delegates to participate in the development work. Discussions over the months led to new requirements for the standard particularly in the area of Internet applications.

The SDIF specification was developed and refined during 1996 and 1997 and published informally via WWW pages at CNMAT and IRCAM. During this time existing analysis/synthesis tools at each institution began to be converted to use SDIF. Also during this time SDIF was added to the Spectral Modeling Synthesis (SMS) package from Pompeu Fabra University in Barcelona, as described in a paper presented at ICMC 97 [63].

We presented SDIF informally to the analysis/synthesis community at ICMC 1997 in Thessaloniki and found widespread support and approval. A number of improvements were suggested at that meeting, including support for 64-bit data, removing the limitation of a global size count, and the addition of ASCII name/value tables. Before this meeting SDIF was a subtype of Electronic Art's Interchange File Format (IFF) [64], like the AIFF and RIFF sound file formats. At this meeting we realized that guaranteeing 64-bit alignment of 64-bit data would be extremely awkward within the IFF framework, and soon thereafter decided not to make SDIF IFF-compatible.

We presented SDIF again in a similar situation at ICMC 1998 in Ann Arbor, Michigan, again receiving widespread support and approval in addition to helpful feedback and suggestions. A paper by authors from all three institutions using SDIF (CNMAT, IRCAM, and The Pompeu Fabra University) was also presented at this conference.

Recent improvements to SDIF have made the syntax more uniform, removing the need for various special cases by subsuming those features under more general mechanisms.

## 5. Goals of SDIF

In the spirit of the XML 1.0 specification [65] we have used the following goals to guide the SDIF standardization process:

- SDIF shall be straightforwardly usable as a storage method for interprocess applications and for streaming over the Internet.
- SDIF shall support a wide variety of applications.
- It shall be easy to write programs which process SDIF data.

- The number of optional features in SDIF is to be kept to the absolute minimum, ideally zero.
- SDIF documents should be directly translatable into human-legible form.
- The design of SDIF shall be formal and concise.
- SDIF documents shall be easy to create and parse.
- The syntax of SDIF shall be uniform and consistent, with a minimum of special cases.
- Terseness is of minimal importance. (SDIF is not a compression format.)
- The SDIF specification may be distributed freely.
- SDIF data can be validated easily both structurally and semantically.
- SDIF data types shall encompass existing art.
- The SDIF format should be extensible.
- SDIF programs shall be able to parse unrecognized data types correctly, and to ignore unrecognized data while processing other data.
- Keep simple things simple.

The motivating idea behind SDIF was that a new, common data format would promote interoperability between many interesting sound analysis and synthesis tools thereby resulting in improvements to those tools and in new research avenues and creative applications. This interoperability requirement leads to easy decisions in the SDIF design concerning data types: ASCII, and big-endian IEEE754 floating point numbers, two's complement integers, and 8-bit bytes. SDIF encodes text using UTF-8, which encompasses most of the world's writing systems and is backwards-compatible with ASCII [66].

The SDIF specification uses SI units (Système International d'Unités, e.g., meters, kilograms, and seconds) [67] to describe the units of parameters. One important consequence of this is that time values are never measured by counting samples or frames; this avoids imposing a particular sampling method, i.e., isochronous sampling.

Originally the first letter of the SDIF acronym stood for “spectral.” It was soon realized that a good format for spectral representations of sound would easily encompass time-domain representations as well. The key concept is that all these representations are discrete time samples of multivalued functions. The SDIF mechanisms to support this concept are frames that encapsulate sampled values and high-resolution time tags for each frame to associate a time to those values. This supports non-uniform sampling. The requirement to stream SDIF data resulted in the imposition of the natural temporal ordering constraint: frames are ordered earliest first.

Remaining design issues concern the contents of the frames themselves. Each frame is a small collection of typed, two-dimensional data matrices. It is easy to translate data from extant file formats into SDIF frames. This is very important as data formatting, file and stream I/O are among the least glamorous of tasks tool developers face. We had to minimize the extra work required to add support for SDIF data.

Fortunately, SDIF has more to offer developers than compatibility with a common format: extensibility. One straightforward form of extensibility is that additional fields may be added to standard matrix types in a backwards-compatible way. Most digital sounds and sound analysis data are represented in a form that frustrates the addition of new interpretations or new parameters. Extensible designs are not sufficient to guarantee that extensibility will be exploited. The AIFF format is an interesting case. AIFF offers a workable extensibility mechanism that Apple Computer successfully exploited, principally with the addition of compressed sound formats [68]. Several good proposals for extensions to AIFF were never adopted mainly through lack of an organization with responsibility to consider these proposals.

We are exploiting the Internet, XML, DTD and XSL to provide the community of SDIF users an efficient way to share and add to a pool of SDIF matrix and data types.

The first component of this system is the ability to obtain machine- and human-readable descriptions of SDIF matrices and frames from a central database via a URL. This URL is constructed by appending unique type designation strings to locations such as http://www.sdif.org/info/matrix/. The result of a query to these locations is an XML file describing the data in the given object. The associated DTD and XSL files give further information to enable a parser to validate and display the data.

The second component is an automated submission procedure that adds new data matrix and frame types to the database by simply submitting HTML forms. One form allows direct upload of XML. Another form builds correct XML files. These features allow for rapid dissemination of new experimental extensions, an important requirement of this community. We have put considerable effort into the base SDIF types so that most applications can already be served from the existing types and supersets thereof.

## 6. Specification of the SDIF Structure

SDIF is designed in two parts. The first is a fixed framework defining the byte format of an SDIF file. (An SDIF “file” may be a physical file on a disk drive or data streamed via a network. In both cases the byte format is the same, so in this section “SDIF file” should be taken to apply to either case.) The second is an extensible library of standard sound descriptions, explaining how each fits into the SDIF framework and giving the semantics. The library of standard sound descriptions will be described in the next section.

### 6.1 SDIF Data Types

SDIF uses the following data types:

- byte: eight bits without semantics
- UTF-8: eight-bit bytes encoding UTF-8 characters
- int32 and int64: 32-bit and 64-bit big-endian two's complement integer
- float32 and float64: 32-bit and 64-bit big-endian IEEE 754 floating point

### 6.2 SDIF Frames

An SDIF file consists of a series of “frames” similar to and inspired by the “chunks” in the IFF/RIFF/AIFF formats [64], as shown in Figure 1. Each frame consists of a four-byte “Frame Type ID,” a four-byte integer “Frame Size,” and the rest of the frame data. The Frame Size is in bytes and does not include the eight bytes taken by the Frame Type ID and the Frame Size itself. The size of a frame is always a multiple of 64 bits; this ensures 64-bit alignment of 64-bit data elements, facilitating reading on 64-bit machines.

The Frame Type ID provides a namespace of 4,294,967,296 (i.e., $2^{32}$) possible frame types. For the near future, we choose Frame Type IDs whose four bytes are each alphanumeric ASCII characters, similar to the way the “type” and “creator” fields for Macintosh files work [69]. Frame Type IDs for the standard sound descriptions begin with the numeral “1,” indicating the first version of the standard frame type library. We encourage developers of experimental frame types to name them starting with the letter “X.” When a program reading an SDIF file from disk sees an unrecognized Frame Type ID it can use the Frame Size to skip forward to the beginning of the next frame.

All SDIF files must begin with an “opening frame” that identifies the file as SDIF. The Frame Type ID for this frame is “SDIF,” so every SDIF file’s first four bytes are the ASCII characters “S,” “D,” “I,” and “F.” The opening frame’s data is typically just eight zero bytes. Figure 2 depicts a typical opening frame.

All SDIF frames after the opening frame have the following structure:

- Frame Type ID: 4 bytes, described above
- Frame Size: int32, described above
- Time Tag: float64, the time to which the given frame applies, in seconds, as measured from a “time zero.”
- Stream ID: int32, allowing multiple logical streams of frames within a single file
- Matrix Count: int32, giving the number of matrices in this frame
- The Matrices themselves, described in the next section

The portion of a frame from the Frame Type ID through the Matrix Count is called the “frame header”; note that its size is 32+32+64+32+32 = 3×64 bits.

Frames in the body of an SDIF file must appear in ascending order of Time Tag. Most frame types are applicable to a single instant of time, e.g., the center point of a STFT window. This avoids issues of overlapping frames. It also implies that synthesizers for these frame types will have some sort of interpolation model to change data smoothly between frames. A few frame types, e.g., time-domain samples and envelope breakpoint functions, represent data that span a given interval of time; in these cases the frame's time tag indicates the *beginning* of the interval.

Frames with a time tag of minus infinity are guaranteed to appear immediately after the opening chunk. This mechanism should be used for any required configuration or initialization data. For example, an SDIF file containing synthesis parameters over time for a custom synthesis method could have a description of the synthesis algorithm in a frame with a time tag of minus infinity. In general, however, each SDIF frame must be “self-contained,” interpretable without any information from other frames, to facilitate streaming.

Stream IDs associate a set of frames at different times together into a single “stream,” a continuous sequence of data that represents a single sonic object over time. Frames with the same Stream ID belong to the same stream. Thus, a single SDIF file can have multiple interleaved series of frame data. These 4-byte IDs do not carry any semantic information; they serve only to differentiate streams from each other. All of the frames with a given Stream ID must be of the same frame type. It is illegal for two frames to have the same Time Tag and the same Stream ID.

### 6.3 SDIF Matrices

A “matrix” is a two-dimensional table of data. Each column corresponds to a parameter such as frequency or amplitude and each row represents an object such as a filter, sinusoid, or noise band. SDIF matrices have the following structure:

- Matrix Type ID: 4 bytes, similar to a Frame Type ID
- Matrix Data Type: a 4-byte code indicating the size and data type of the matrix elements
- Row Count: int32, giving the number of rows in the matrix
- Column Count: int32, giving the number of columns in the matrix
- Matrix Data, in row-major order
- Padding Bytes: enough zero bytes to make the total size of the matrix a multiple of 64 bits

The portion of a matrix from the Matrix Type ID through the Column Count is called the “matrix

header"; note that its size is 32+32+32+32 = 2×64 bits.

The Matrix Type ID is equivalent to the Frame Type ID, providing a namespace of possible matrix types. There is one global namespace of matrix types, not a separate matrix namespace per frame type. A frame type is defined by the matrix types that must appear in it. Most frame types consist of a single matrix; in these cases the Matrix Type ID is the same as the Frame Type ID. Some frame types will consist of a main matrix of data plus a few extra fields in a secondary one-row matrix, e.g., STFT frames must include the frequency sampling interval and the frame size as well as the actual STFT bin values.

A frame may not contain two or more matrices of the same type. Frames may contain additional matrices besides the required matrices, but for ease of parsing these must come after the required matrices.

The Matrix Data Type indicates the type and size of each data element in the matrix, according to the table in Figure 3, e.g., int32 or float64. All data in a given matrix must have the same type. UTF-8 characters are a special case because they do not all have the same number of bytes. In this case, each UTF-8 byte is a separate matrix element, and there may be fewer characters than matrix elements.

The Row Count and Column Count give the size and shape of the matrix; their product is the number of elements in the matrix. To find the total size of the Matrix Data, multiply the number of elements by the size of each element as given by the Matrix Data Type. Programs must do this to skip over matrices with unrecognized types. One-dimensional matrices, for example, matrices containing a string of UTF-8 text, have one row.

If the Matrix Data's size is not a multiple of 64 bits, there must be enough padding bytes to make the total length a multiple of 64 bits. Because the frame header and the matrix header are a multiple of 64 bits, ensuring that the combined size of the Matrix Data and Padding Bytes is a multiple of 64 bits guarantees that matrix data will always begin on a 64-bit boundary.

The definition of a standard matrix type includes the name of the kind of entity that appears in each row, e.g., "sinusoid," and the list of standard columns. Each column is defined by the following:

- Name, e.g., "frequency"
- Units, in SI [67]
- Minimum and maximum values

There are many cases where it makes sense to add extra columns to a standard matrix, e.g., a measure of the "importance" or perceptual salience of each of a set of partials. An SDIF matrix of a standard matrix type can have more than the required number of columns; in this case the

first columns must have the standard meanings and the rest of the columns can have custom meanings. The definition of these extra columns can appear in the Stream Information Matrix, described in the next section.

### 6.4 The Stream Information Matrix

We provide an optional mechanism to associate a textual name/value table with a stream. This provides a mechanism to give meta-information describing the information in the stream, including the following:

- Technical information about a recorded sound: date, location, microphone types and placement, performer's name, instrument's serial number, etc.
- The analysis program that created this stream, values of parameters used, date it was run, user name, name of the file it analyzed, machine it ran on, etc.
- Structural information about how this particular stream relates to other streams in the file and how to interpret it in a given synthesis environment.

This matrix typically appears in a frame at time minus infinity for the stream being described. Since the frame type must be the same for all frames in a stream, the "declaration" of the stream's frame type comes from the Frame Type ID. It is legal for a frame at time minus infinity not to contain the required matrix or matrices that must appear in all other frames.

Although this name/value table is in use at IRCAM with a preliminary format, work is still underway to standardize the final format for this table. The draft Xlink "extended link" facility from the World Wide Web Consortium provides an elegant mechanism for parts of documents to refer to parts of other documents [70]. We would like to incorporate these ideas into SDIF's stream information matrix to allow it to refer to other streams, temporal segments of other streams, and even data in other files.

Recall that Stream IDs do not carry any semantic information; they serve only to differentiate streams from each other. A name/value table can have a name "StructuralInfo" whose value provides a way to document the meaning of the stream and indicate how it is grouped with other streams into higher-level structures. In particular, the StructuralInfo field can indicate the object (e.g., filter, oscillator, etc.) within the structure of a synthesis architecture to which a given stream of data applies.

We are inspired by URL syntax [71]. We use the URL protocol specification as a specification for the synthesis program and the URL path as a means for multi-level tree structuring. The OpenSoundControl protocol [72, 73] also uses a URL-inspired syntax for hierarchical organization of entities in a synthesis architecture.

Examples:

```
csound:/instr1/oscA/inpB
chant:/synth3/bank2/filter4
synthadd:/bank1
```

The semantics of the path structure is left to the synthesis program and is not specified by SDIF. In this sense we try to present an open and flexible architecture. For example, the path can be used to specify a certain input in a certain unit-generator in a certain instrument definition.

Note that we use a single slash rather than a double slash after the colon, since in a URL a double slash indicates an Internet host or address.

### 6.5 Support for Alternately-Structured Data

Although SDIF's structure of two-dimensional matrices within frames within streams seems to adequately encompass the features of pre-SDIF sound description formats, there are occasional cases where one would like to include data in an SDIF file that does not fit this structure. Examples include the embedding of sound files, graphical images, or other files.

All such data should go into a matrix of type "1BYT," short for "bytes." A 1BYT matrix should have a Matrix Data Type of 5, indicating "arbitrary byte," one row, and as many columns as the number of bytes of embedded data. (If the size of the embedded data is not a multiple of 8 bytes there should also be some padding bytes.) The embedded data can be placed at a particular time and within a particular stream by virtue of the time tag and Stream ID of the frame containing the 1BYT matrix.

By keeping non-matrix data within SDIF's matrix structure, there does not have to be a special case for a program that parses SDIF.

## 7. The Library of Standard Sound Description Types

All SDIF matrices and frames are typed using a compact 4-byte type ID. This allows information on the interpretation and documentation of matrix and frame data to be stored separately from SDIF streams. We use the recent XML standard to represent this additional data.

We will give the XML definitions of two standard frame types for illustrative purposes. Then we will document the remaining standard frame types in a more concise prose form. The prose form of the documentation was created by hand, but as XSL tools become available, this kind of document can be created automatically from the XML files.

### 7.1 Fundamental Frequency Estimates

Here is an example of a data matrix in XML form:

```
<matrix type= "1FQ0">
        <description>Fundamental frequency estimates</description>
```

```
        <row name="Candidate fundamental suggested by the estimator"></row>
        <column name="Fundamental frequency" units="Hertz" min="0.0"></column>
        <column name="Confidence" units="Unitless" min="0.0" max="1.0"></column>
</matrix>
```

This tells us that we are defining a matrix of type “1FQ0” to hold fundamental frequency estimates. Each row of a 1FQ0 matrix contains a candidate fundamental suggested by the estimator. (In other words, a fundamental frequency estimator can put multiple candidate frequencies in the different rows.) The first column of a 1FQ0 matrix holds the fundamental frequencies for each estimate, in Hertz, and the second holds the confidence factor, a unitless value between zero and one.

Note that we use the term “fundamental frequency” or “f0” rather than “pitch”; we might invent a new SDIF frame and matrix type for pitch to represent the result of a true pitch estimator that applied a model based on human perception [74].

We also use XML to define frame types in terms of the matrices they must contain. In this case, as with most standard SDIF frame types, the frame has a single required matrix whose matrix type is the same as the frame type.

```
<frame type="1FQ0">
        <description>Fundamental frequency estimates. Not all sounds have a definite
                fundamental frequency; some have multiple possible fundamental frequencies.
        </description>
        <required-matrix>1FQ0</required-matrix>
</frame>
```

### 7.2 Discrete Short-Term Fourier Transform

1STF frames represent the data that come out of a discrete time-domain to frequency-domain transform such as an FFT. 1STF frames have two required matrices: a 1STF matrix that contains the bin data, and a 1STI “info” matrix to record overall information about the transform that says how to interpret the bin data. Because of SDIF’s goal of consistent and uniform syntax, there is no such thing as a per-frame header that could hold this information; instead it appears in its own small matrix.

```
<frame type="1STF">
        <description>Result of a discrete time-domain to frequency-domain transform such
                as an FFT. The time tag in a 1STF frame is the time of the center of the window,
                not the beginning.
        </description>
        <required-matrix>1STI</required-matrix>
        <required-matrix>1STF</required-matrix>
```

```
</frame>

<matrix type="1STI">
        <description>Short- Term Fourier Transform Information</description>
        <row name="Info"><description>Always exactly one row</description></row>
        <column name="Frequency Sampling Interval" units ="Hertz" min="0.0">
                <description>frequency precision in the STFT, generally equal to the Nyquist
                frequency divided by the number of bins.</description>
        </column>
        <column name="Frame size" units="Seconds" min ="0.0">
                <description>length of the signal input to the transform.</description>
        </column>
</matrix>

<matrix type="1STF">
        <description>Short- Term Fourier Transform Bin Data. Convert these complex
                numbers into polar form to get magnitude and phase.</description>
        <row name="frequency bin"/>
        <column name="Real part" units ="unitless"></column>
        <column name="Imaginary part" units ="unitless"></column>
</matrix>
```

Note that the information matrix describes the interpretation of the bin data not in terms of the original sampling rate of the data that was input to the STFT, but in terms of the frequency sampling interval of the bins, in the SI unit Hertz.

### 7.3 Picked Spectral Peaks

Picked spectral peaks represent peaks (local maxima) in a spectrum at a given time. Peak pickers typically fit some kind of curve to 1STF data, estimating the frequency, amplitude, and phase based on a small number of bins around the maximum.

1PIC frames consist of a single 1PIC matrix. Each row represents a spectral peak. The columns are:

- Amplitude (linear)
- Frequency (Hertz, > 0)
- Phase (Radians, from 0 to $2\pi$)
- Confidence (Unitless, from 0 to 1)

The confidence factor might be used to indicate how much of the energy around this peak was from a sinusoid or how well the energy around this peak matches a sinusoid.

### 7.4 Sinusoidal Tracks

Sinusoidal tracks represent sinusoids that maintain their continuity over time as their frequencies, amplitudes, and phases evolve. Sinusoidal tracks are the standard data format used as the input to classical additive synthesis.

1TRC frames consist of a single 1TRC matrix. Each row represents the current state of a sinusoidal track. The columns are:

- Index (a unique integer >= 1 identifying this track and allowing it to be matched with 1TRC data in other frames. This is similar in concept to a Stream ID.)
- Amplitude (linear)
- Frequency (Hertz, > 0)
- Phase (Radians, from 0 to $2\pi$)

Synthesizers of 1TRC frames are expected to match the data for each sinusoid from frame to frame using the index numbers. Values for amplitude and frequency should somehow be interpolated so that they change smoothly between frames.

There is no guarantee that a partial appearing in one frame will also appear in the next frame. The situation where a partial appears in one frame but not the next is called a “death,” and when a partial does not appear in one frame but does appear in the next it is called a “birth.” These cases are challenging when writing a synthesizer. It is recommended that partials appearing for the first or last time in a series of frames should have amplitudes of zero, so that the semantics of fading in and out are explicitly in the SDIF data rather than needing to be added on by the synthesizer.

### 7.5 Pseudo-harmonic Sinusoidal Tracks

Pseudo-harmonic sinusoidal track frames are exactly like sinusoidal track frames except that the partials are understood to lie on or close to a harmonic series. Thus, the first column of the matrix represents harmonic number, rather than an arbitrary index. The fundamental is defined to be harmonic number zero. These frames use “1HRM” for the frame and matrix ID.

### 7.6 Exponentially Decaying Sinusoids/Resonances

Resonance data can describe the characteristics of a resonant system like a group of tuned filters, or can specify parameters for a model of sinusoids with fixed frequencies and exponentially decaying amplitudes. (These are equivalent because the impulse response of such a filter bank is that collection of decaying sinusoids.)

Unlike most other frame types, this data is generally specified once for all time rather than appearing in multiple frames with different time tags.

1RES frames consist of a single 1RES matrix whose rows represent individual resonances. The columns are:

- Initial amplitude ("$A_0$," linear)
- Frequency ("F," Hertz)
- Decay rate ("ρ," Hertz: see below)
- Initial phase ("ϕ," Radians, from 0 to 2π)

Each resonance is computed as follows:

$$s(t) = A_0 \sin(\phi + 2\pi F t) e^{-\rho t}$$

### 7.7 Time-Domain Samples

We represent multichannel time-domain samples in SDIF with 1TDS frames, which consist of a 1TDS matrix. Each row represents a sample frame; the columns represent different audio channels. Because SDIF matrices are in row-major order, this representation correctly interleaves samples from different channels. Floating point sample values should be in the nominal range [-1, 1].

## 8. Applications of SDIF

The main application of SDIF, as the name implies, is interchange of sound descriptions. SDIF allows interoperability of, e.g., results of the SMS analysis package synthesized on CNMAT's synthesizer.

### 8.1 The SDIF Library

At IRCAM and CNMAT, the analysis/synthesis teams have decided to use SDIF for all new software developments and gradually to upgrade existing software to use SDIF. For this purpose, we have written an SDIF library in C for reading and writing SDIF files. This library is currently used on multiple platforms, including SGI IRIX, DEC Alpha OSF, Apple MacOS, Windows, and Linux.

The SDIF library contains procedures for reading, writing, and skipping over frame headers, matrix headers, name/value tables, and matrices. These procedures hide the details of the SDIF byte format and automatically handle issues such as eight byte alignment and byte swapping on little-endian machines. The library also includes abstract numeric types of the required sizes for SDIF data that work on each architecture. These procedures can be used either to read data from a physical file on disk, or to parse streamed SDIF data.

The library defines an efficient, linked-list-based data structure for the representation of SDIF

data in memory, as opposed to the buffered array of bytes that were read from disk or from the network. This makes for more efficient traversal and manipulation, particularly when adding and removing frames and matrices. The library contains utility procedures to go between file descriptors and this in-memory representation.

### 8.2 Analysis/Synthesis Software Using SDIF

Analysis/synthesis systems at IRCAM, CNMAT, and Pompeu Fabra University now use SDIF as their primary file format.

IRCAM's analysis software, "additive," now writes its result as SDIF files. IRCAM's "HMM" program, which uses hidden Markov models [41] to match spectral peaks from frame to frame into tracks, also outputs SDIF.

CNMAT's real-time software synthesizer "softcast" reads tracks of partials, fundamental frequency estimates, noise data [75] and resonance data from SDIF files. It synthesizes from these data types offering extensive control over the progression through the time axis of the SDIF file with respect to real time and numerous real-time timbral modifications including inharmonicity, filtering, and timbral interpolation.

The SMS sinusoidal analysis/synthesis package was converted from using its own data file format to an early version of SDIF in 1997 [63]. Many special-purpose frame and matrix types were defined for this system, including the following:

- Generic: Container frame for one or several tracks.
- Generic Track: Container frame with some global information for the track and pointing to other frames.
- Region: Frame that describes a segment of a track with common characteristics.
- Envelope: Frame used to store High Level Attributes of a Generic Track, a Region, or another frame.

Maintainers of other analysis software, including SNDAN and Lemur, have expressed an interest in adding SDIF support in the near future.

### 8.3 Spectral Envelopes

A project at IRCAM uses SDIF to store the time-varying spectral envelope of a sound [76]. There is an individual frame type for each of the different parameterizations of spectral envelopes obtained by various estimation methods used in the project: cepstrum and discrete cepstrum coefficients, autoregressive coefficients, reflection coefficients, formant, "fuzzy formants," discretized spectral envelopes, etc. The spectral envelope of the sinusoidal component and of the residual noise component are stored in separate streams of a single file.

D. Schwarz has written programs which apply SDIF-encoded spectral envelopes to several sound

representations, including time-domain sample files, spectral peaks, sinusoidal tracks, resonances, and noise bands.

### 8.4 Chant

IRCAM's new CHANT synthesizer [77] uses SDIF via a new library written by D. Virolle. This synthesizer reads its configuration from an SDIF file. Here is an example “StructuralInfo” value for a stream that defines it to address a particular formant waveform (FOF) in a given FOF bank:

```
Chant:Patch0/1/FOB/1/5/0./2.5;
```

This library was ported to the Macintosh by Adrien Lefèvre and is included as a plug-in component in the Diphone software [78], which uses SDIF files to store and read Chant filters and (FOF) data.

### 8.5 Open Sound Edit: 3D Visualization and Editing of SDIF data

OpenSoundEdit (OSE) [79] is a tool for visualizing and editing SDIF data. OSE currently supports several standard SDIF sound representations, including time-domain samples, sinusoidal tracks, amplitude and fundamental frequency estimates, and spectral peaks.

The visual appearance of the OSE display consists of an earth-tone ground plane and blue sky, establishing a familiar and unambiguous horizontal plane. The provided navigational controls favor a familiar orientation perpendicular to the ground plane, i.e., walking. As sound objects are brought from SDIF files into the editor world, they are assigned an available region on the ground plane and a specific visualization object is instantiated to operate on them.

Figure 4 shows an example of sinusoidal tracks (1TRC) as visualized in OSE. The three dimensions drawn are time, frequency and amplitude. In this example, time is increasing from left to right, frequency increases with depth from the viewer and amplitude is vertical. Amplitude and frequency are measured on a logarithmic scale, and phase is not displayed. Each partial is drawn as a piecewise linear function connecting the amplitude and frequency values of the partial in successive frames of the SDIF stream.

A moveable time-selection frame (i.e., the “pipe” frame perpendicular to the ground plane) allows the user to “scrub” through the data along the time axis. “Shadows” on the time-selection frame represent the interpolated amplitude and frequency of each partial at the position of the frame.

Real-time audio feedback is achieved using a *synthesis server*, an application that produces sound in response to OpenSound Control messages. For example, moving time-selection frame in OSE sends messages to synthesis server indicating the selected time. The server then synthesizes sound based on the interpolated values of the two frames before and after the selected time. Thus, a user can scrub through the SDIF file both visually and aurally.

Editing operations are performed via direct manipulation in the 3D display or using external

controls. The behaviors of these editing controls are specific to the type of sound representation. For example, the sinusoidal track model allows users to select a range of partials and scale their amplitude and frequency components. A partial can be scaled uniformly along its entire length (i.e., along the time axis), or over only a small region near the time-selection frame.

Changes made to the SDIF representation in the editor can also be communicated to the synthesis server by sending the frames that have changed. The server then replaces the modified frames in its copy of the representation.

### 8.6 Open Sound World

OSW, or "Open Sound World," is a scaleable, extensible, object-oriented language that allows sound designers and musicians to process sound in response to expressive real-time control. In OSW, users connect components, called *transforms*, to form larger structures, called *patches*, which model the flow of signals and events.

OSW includes transforms and data types for manipulating SDIF streams and frames, including separate data types for the standard SDIF frame types, translating between SDIF frames and OSW vector and list types, and reading and writing of SDIF streams on local disks or networks.

Because OSW data types can be any valid C++ type (i.e., classes or scalar primitives), it can be easily extended to handle new SDIF frame types. Users can create new transforms and data types without deep knowledge of C++ using the "Externalizer," a graphical tool that allows users to create new transforms and data types as high-level specifications. The documentation for a new SDIF frame type could include Externalizer specifications of transforms that manipulate the frame type or synthesize sounds from streams of the new representation.

OSW is described in detail in another paper presented at this AES convention [80].

### 8.7 SDIF and MPEG-4's SAOL

The Motion Pictures Experts Group (MPEG) completed the MPEG-4 standard, formally ISO 14496 (for the International Standardization Organisation), in October 1998; MPEG-4 has now been published and designated as International Standard. Unlike the previous MPEG-1 (ISO 11172) and MPEG-2 (ISO 13818) standards, which were primarily for coding and compression of audiovisual data, MPEG-4 includes capabilities for the compressed transmission of synthetic sound and computer graphics, and for the juxtaposition of synthetic and "natural" (compressed audio/video) material.

Within the MPEG-4 standard, there is a set of tools called Structured Audio [81] that allows synthetic sound to be transmitted as a set of instructions in a unit-generator-based language, and then synthesized at the receiving terminal. The synthesis language used in MPEG-4 for this purpose is a newly devised one called SAOL (pronounced "sail") , for Structured Audio Orchestra Language [82], similar to Csound [60, 61] and the family of "Music-N" languages [83]

from the history of computer music. Timing and control comes in the form of a "score" written in SASL, the Structured Audio Score Language. All compliant decoders of the full MPEG-4 standard must include an implementation of the SAOL language and be able to synthesize sound from a given SAOL program and SASL score.

We have developed tools to encode SDIF data into SAOL programs and SASL scores, which effectively make an MPEG-4 decoder into an SDIF synthesizer. If MPEG-4 has the same success as MPEG-1 and MPEG-2, then SAOL/SASL will be an attractive platform for SDIF users.

The basic idea is to write SAOL programs to synthesize each kind of sound description supported by SDIF. For example, a sum-of-sinusoids additive synthesizer is required to convert 1TRC frames into sound. Note that there is not always a single correct mapping from SDIF data into sound, for example, a fundamental frequency envelope could usefully be "rendered" as sound by applying it to a synthetic timbre such as a sawtooth wave, but this particular mapping is arbitrary. As another example, the interpolation of parameter values between each pair of frames is not specified by the SDIF standard.

Then the data in a given SDIF file can be converted automatically into a SASL score to provide time-varying parameters to the synthesizer. SAOL/SASL has a feature called "dynamic wavetables" that makes the SDIF-to-SASL conversion much more space efficient. A SAOL "wavetable" is typically used to store a portion of an audio waveform, but can be used as an array of arbitrary numerical data like the arrays in traditional programming languages. A SASL score can contain a new set of data for a given dynamic wavetable, to take effect at a given time. At that time, the contents of the wavetable automatically change to the new values in the score. This can appear any number of times in a SASL score. SDIF matrices are translated into data for dynamic wavetables, and thus are encoded into SASL in an efficient manner.

This technique is described in depth in [84].

### 8.8 SDIF Integration in the Max/MSP Environment

Opcode's "Max" is a programming language and environment for reactive real-time control and scheduling of MIDI, graphics, and other multimedia for the Macintosh. Cycling 74's "MSP" extension to Max adds digital signal processing on the PowerPC, with no special hardware. Max/MSP has many attractive features for developing real-time computer music applications, including active support and development, a programming model based on patching that can be used by non-programmers, real-time interactivity even while developing a program, a large library of primitive computational objects, the ability to create new objects (called "externals") in C, and a rich history and repertoire.

Many of the types of sound descriptions supported by SDIF afford expressive and musically

interesting high-level manipulation and control. For example, it is possible to stretch and expand the timing of additive synthesis tracks without changing pitch. Timbral interpolation, filtering, and many other transformations are often much easier to implement in the domains of SDIF sound descriptions than for time-domain samples. Max/MSP is a good environment for exploring and prototyping these kinds of applications.

To this end we have created a collection of Max externals for storing, reading, writing, and manipulating SDIF data. The centerpiece is an object called "SDIF-buffer" that stores a portion (or all) of an SDIF stream in Max's memory and associates a symbolic name with it, analogous to MSP's "buffer~" object that represents audio samples in memory.

Max/MSP has objects that provide various control structures to read data from a "buffer~" and output signals or events usable by other Max/MSP objects. Similarly, we have created a variety of "SDIF selector" objects that select a piece of SDIF data from an SDIF-buffer and output it as a standard Max/MSP data type. The simplest SDIF selector outputs the main matrix from the frame whose time tag is closest to a given input time. More sophisticated SDIF selectors hide the discrete time sampling of SDIF frames, using interpolation along the time axis to synthesize SDIF data. This provides the abstraction of continuous time, with a virtual SDIF frame corresponding to any point along the time axis. We provide linear and a variety of polynomial interpolators.

Another set of objects, "SDIF mutators," modify the data in an SDIF buffer. These can be used to write new SDIF data from within a Max patch. For example, real-time sound analysis software can generate an SDIF model of an audio signal.

This technique is described in depth in [85]. We look forward to developing more new SDIF applications in the Max/MSP environment.

### 8.9 Streaming SDIF

We have started to pursue Internet streaming applications of SDIF. Early research suggests that existing streaming architectures such as the RTP and RTSP protocols provide an adequate framework for delivery of SDIF data, and support multicasting. The use of existing, widely adopted standards promises smooth integration of streaming SDIF work with existing servers, decoders, and environments.

The atomic unit of data transfer is an entire SDIF frame. SDIF frames are semantically self-contained, meaning that each frame can be interpreted without reference to any other frame. Any subset of the frames in an SDIF file can be synthesized in a meaningful way, so streaming SDIF is naturally tolerant of lost frames. We can use this feature of SDIF to drop frames on purpose, first transmitting a subset of frames coarsely sampled in time for a quick preview of an SDIF file. We can then fill in the detail by transmitting the intervening frames.

Our architecture for a streaming client includes a buffer of SDIF frames that grows dynamically as new data arrives from the network. We have implemented this in Max/MSP using the SDIF-buffer object described above.

The protocol by which the client requests data from the server presents a hierarchical database model. At the highest level are the various SDIF files served by the server. Within each SDIF file is a collection of SDIF streams, and within each stream is a sequence of frames. Clients can request a particular stream or all the streams of a given frame type from a file or set of files, and can request frames within a given time interval.

### 8.10 Spatial Audio Applications

Most SDIF frame data are the result of an analysis of a sound. We are exploring sound descriptions derived from user gestures, e.g., the desired spatial location for reproduced sound streams. A new data matrix and frame type has been developed to represent the desired location and perceived size of sound sources. By combining these frames with synthesizable sound descriptions in a single SDIF stream we have created a new “discrete” channel format for multi-channel audio. This supports our research into a scalable spatial audio distribution system and medium that avoids encoding audio for a particular speaker/room contribution by deferring spatial rendering to a real-time processing step that integrates knowledge of the listener’s actual listening environment and exploits the available signal processing and reproduction resources [86]. Because SDIF is both a storage and streaming format and can describe both abstract and concrete (recorded) representations of sound, it offers a coherent model for spatial audio that encompasses VR, interactive games, home theatre, kiosks, live performance, and theme parks.

## 9. The Future of SDIF

We will make the SDIF library described in section 6.1 available to the public at no charge.

Work is underway to use SDIF for video frames. It is straightforward to embed motion JPEG frames as SDIF frames. SDIF's time-tagging of frames provides the framework needed for audio/video synchronization.

We are developing guidelines to steer contributors of SDIF representations for new kinds of sound descriptions. Frame and matrix types for these SDIF representations will initially be used only for a particular project, and should have type IDs beginning with the letter “X” to indicate their experimental nature. Once the representation is stable and useful, the inventors will submit XML definitions of the new frame and matrix types via the Internet to a central repository, as described above in section 5. At the same time, inventors should publish their work so that it can be reviewed by the SDIF user community. Once there is widespread consensus that the new sound representation is of general use and that the proposed matrix and frame types utilize SDIF in an effective way, the maintainers of the SDIF standard will add them to the library of standard

sound description types.

We would also like to require that developers of new SDIF types supply one or more synthesizer programs for the new type; not every SDIF type must have an unambiguous mapping to sound, but an example synthesizer helps clarify the semantics and utility of the new type. We are exploring several candidate languages for this, including SAOL, C++ extended with our classes to support discrete functions of time [87], and MathML [88]. The language should also be capable of expressing the analysis algorithm.

## 10. Acknowledgements

Richard Andrews, James Beauchamp, Richard Dudas, Lippold Haken, Jean Laroche, Xavier Rodet, Xavier Serra, Eric Scheirer, Diemo Schwartz, Ron Smith, Greg Wakefield, Raymond Wang, Jim Wright.

## References

[1] S. Khoury, A. Freed, and D. Wessel, "Volumetric Visualization of Acoustic Fields in CNMAT's Sound Spatialization Theatre," presented at IEEE Visualization 98, Research Triangle Park, NC, 1998.

[2] S. T. Pope and G. van Rossum, "Machine tongues XVIII: a child's garden of sound file formats," *Computer Music Journal*, vol. 19, num. 1, pp. 25-63, 1995.

[3] J. C. Risset, "Computer Study of Trumpet Tones," *Journal of the Acoustical Society of America*, vol. 33, pp. 912, 1965.

[4] J. C. Risset, "Computer Study of Trumpet Tones," Bell Laboratories, Murray Hill, NJ 1966.

[5] J. C. Risset and M. V. Mathews, "Analysis of musical-instrument tones," *Physics Today*, vol. 22, num. 2, pp. 23-32, 1969.

[6] J. M. Grey and J. A. Moorer, "Perceptual evaluations of synthesized musical instrument tones," *Journal of the Acoustical Society of America*, vol. 62, num. 2, pp. 454-62, 1977.

[7] J. A. Moorer and J. Grey, "Lexicon of analyzed tones: Part I: A violin tone," *Computer Music Journal*, vol. 1, num. 2, pp. 39-45, 1977.

[8] J. A. Moorer and J. Grey, "Lexicon of analyzed tones: Part II: Clarinet and Oboe Tones," *Computer Music Journal*, vol. 1, num. 3, pp. 12-29, 1977.

[9] J. A. Moorer and J. Grey, "Lexicon of analyzed tones: Part III: The Trumpet," *Computer Music Journal*, vol. 2, num. 2, pp. 23-31, 1978.

[10] M. R. Portnoff, "Implementation of the digital phase vocoder using the fast Fourier transform," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. ASSP-24, num.

3, pp. 243-8, 1976.

[11] J. W. Cooley and J. W. Tukey, “An algorithm for the machine computation of complex Fourier Series,” *Mathematics of Computation*, vol. 19, pp. 297-301, 1965.

[12] J. A. Moorer, “The use of the phase vocoder in computer music applications,” *Journal of the Audio Engineering Society*, vol. 26, num. 1-2, pp. 42-5, 1978.

[13] B. S. Atal, “Speech analysis and synthesis by linear prediction of the speech wave,” , 1969.

[14] J. Makhoul, “Linear prediction: a tutorial review,” *Proceedings of the IEEE*, vol. 63, num. 4, pp. 561-80, 1975.

[15] P. Lansky and K. Steiglitz, “Synthesis of timbral families by warped linear prediction,” *Computer Music Journal*, vol. 5, num. 3, pp. 45-9, 1981.

[16] J. A. Moorer, “The use of linear prediction of speech in computer music applications,” *Journal of the Audio Engineering Society*, vol. 27, num. 3, pp. 134-40, 1979.

[17] X. Rodet and P. Depalle, “Synthesis by rule: LPC diphones and calculation of formant trajectories,” presented at ICASSP 85. Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (Cat. No. 85CH2118-8), Tampa, FL, USA, 1985.

[18] R. J. McAulay and T. F. Quatieri, “Mid-rate coding based on a sinusoidal representation of speech,” presented at IEEE International Conference on Acoustics, Speech, and Signal Processing, Tampa, FL, USA, 1985.

[19] W. J. Pielemeier and G. H. Wakefield, “A high resolution time-frequency representation for musical instrument signals,” *Journal of the Acoustical Society of America*, vol. 99, num. 4, pp. 2382-2396, 1996.

[20] P. De Gersem, B. De Moor, and M. Moonen, “Applications of the continuous wavelet transform in the processing of musical signals,” presented at 13th International Conference on Digital Signal Processing, Santorini, Greece, 1997.

[21] R. Kumaresan and D. W. Tufts, “Estimating the parameters of exponentially damped sinusoids and pole-zero modeling in noise,” *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. ASSP-30, num. 6, pp. 833-40, 1982.

[22] S. Umesh and D. W. Tufts, “Estimation of parameters of exponentially damped sinusoids using fast maximum likelihood estimation with application to NMR spectroscopy data,” *IEEE Transactions on Signal Processing*, vol. 44, num. 9, pp. 2245-59, 1996.

[23] K. D. Marshall, “Modal analysis of a violin,” *Journal of the Acoustical Society of America*, vol. 77, num. 2, pp. 695-709, 1985.

[24] I. Bork, “Modal analysis of acoustic fields,” *Acustica*, vol. 75, num. 2, pp. 154-67, 1991.

[25] S. Kinoshita, “Lattice filtering applications to earthquake observation,” *Zisin. Journal of the Seismological Society of Japan*, vol. 39, num. 1, pp. 1-14, 1986.

[26] Y. Potard, P.-F. Baisnée, and J.-B. Barriere, “Experimenting with Models of Resonance Produced by a New Technique for the Analysis of Impulsive Sounds,” presented at International Computer Music Conference, La Haye, 1986.

[27] J. W. Beauchamp and A. Horner, “Spectral modelling and timbre hybridisation programs for computer music,” *Organised Sound*, vol. 2, num. 3, pp. 253-8, 1997.

[28] J. W. Beauchamp, “Unix Workstation Software for Analysis, Graphics, Modification, and Synthesis of Musical Sounds,” presented at Audio Engineering Society, 1993.

[29] R. C. Maher and J. W. Beauchamp, “Fundamental Frequency Estimation of Musical Signals Using a Two-Way Mismatch Procedure,” *Journal of the Acoustical Society of America*, vol. 95, num. 4, pp. 2254-63, 1994.

[30] K. Fitz, L. Haken, and B. Holloway, “Lemur - A Tool for Timbre Manipulation,” presented at International Computer Music Conference, Banff, Canada, 1995.

[31] K. Fitz and L. Haken, “Sinusoidal modeling and manipulation using Lemur,” *Computer Music Journal*, vol. 20, num. 4, pp. 44-59, 1996.

[32] R. C. Maher, “An Approach for the Separation of Voices in Composite Musical Signals,” Ph.D, Dept. of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, 1989.

[33] K. Fitz, L. Haken, B. Holloway, E. Tellman, and B. Walker, “The Lemur File Format,” http://datura.cerl.uiuc.edu/Lemur/LemurDocFormat.html

[34] E. Tellman, L. Haken, and B. Holloway, “Timbre morphing of sounds with unequal numbers of features,” *Journal of the Audio Engineering Society*, vol. 43, num. 9, pp. 678-89, 1995.

[35] K. Fitz and L. Haken, “Bandwidth Enhanced Sinusoidal Modeling in Lemur,” presented at International Computer Music Conference, Banff, Canada, 1995.

[36] X. Rodet, “Musical sound signal analysis/synthesis: sinusoidal-plus-residual and elementary waveform models,” *Applied Signal Processing*, vol. 4, num. 3, pp. 131-41, 1997.

[37] P. Depalle, G. Garcia, and X. Rodet, “The recreation of a castrato voice, Farinelli's voice,” presented at 1995 Workshop on Applications of Single Processing to Audio and Accoustics, New Paltz, NY, USA, 1995.

[38] X. Rodet, “Recent developments in computer sound analysis and synthesis,” *Computer Music Journal*, vol. 20, num. 1, pp. 57-61, 1996.

[39] R. Gribonval, E. Bacry, S. Mallat, P. Depalle, and X. Rodet, “Analysis of sound signals

with high resolution matching pursuit," presented at IEEE-SP Third International Symposium on Time-Frequency and Time-Scale Analysis, Paris, France, 1996.

[40] T. Galas and X. Rodet, "An improved cepstral method for deconvolution of source-filter systems with discrete spectra : application to musical sound signals," presented at International Computer Music Conference, Glasgow, UK, 1990.

[41] P. Depalle, G. Garcia, and X. Rodet, "Tracking of partials for additive sound synthesis using hidden Markov models," presented at 1993 IEEE International Conference on Acoustics, Speech, and Signal Processing, Minneapolis, MN, USA, 1993.

[42] J. Laroche, "A new analysis/synthesis system of musical signals using Prony's method. Application to heavily damped percussive sounds.," presented at IEEE ICASSP-89, 1989.

[43] C. Rogers, P. Depalle, P. Hanappe, G. Eckel, E. Flety, and J. Carrive, "AudioSculpt home page," http://www.ircam.fr/produits/logiciels/log-forum/AudioSculpt-e.html

[44] P. Depalle, G. Poirot, C. Rogers, E. Fléty, and J. Carrive, "Super Phase Vocoder home page," http://www.ircam.fr/produits/logiciels/log-forum/superVP-e.html

[45] C. Agon, G. Assayag, M. Laurson, and C. Rueda, "Computer Assisted Composition at Ircam : PatchWork & OpenMusic," *Computer Music Journal*, 1999.

[46] A. Freed, M. Goldstein, M. Goodwin, M. Lee, K. McMillen, X. Rodet, D. Wessel, and M. Wright, "Real-Time Additive Synthesis Controlled by a Mixture of Neural-Networks and Direct Manipulation of Physical and Perceptual Attributes," presented at International Computer Music Conference, Aarhus, Denmark, 1994.

[47] D. Wessel, C. Drame, and M. Wright, "Removing the Time Axis from Spectral Model Analysis-Based Additive Synthesis: Neural Networks versus Memory-Based Machine Learning," presented at International Computer Music Conference, Ann Arbor, Michigan, 1998.

[48] M. M. Goodwin, "Adaptive signal models : theory, algorithms, and audio applications," Ph. D. dissertation, Memorandum no. UCB/ERL M97/91, Electronics Research Laboratory, College of Engineering, University of California, Berkeley, 1997.

[49] X. Rodet and G. Bennett, *Synthesis of the singing voice*. Cambridge, MA, USA: MIT Press, 1989.

[50] J.-B. Barriere, P.-F. Baisnee, A. Freed, and M.-D. Baudot, "A Digital Signal Multiprocessor and its Musical Application," presented at 15th International Computer Music Conference, Ohio State University, 1989.

[51] D. Zicarelli, "An Extensible Real-Time Signal Processing Environment for Max," presented at International Computer Music Conference, Ann Arbor, Michigan, 1998.

[52] X. Serra and J. Smith, III, "Spectral modeling synthesis: a sound analysis/synthesis system based on a deterministic plus stochastic decomposition," *Computer Music Journal*, vol.

14, num. 4, pp. 12-24, 1990.

[53] X. Serra, “A System for Sound Analysis/Transformation/Synthesis Based on a Deterministic Plus Stochastic Decomposition,” PhD dissertation, STAN-M-58, Music, CCRMA, Stanford University, 1989.

[54] J. Laroche, Y. Stylianou, and E. Moulines, “HNS: Speech modification based on a harmonic+noise model,” presented at IEEE International Conference on Acoustics, Speech, and Signal Processing (Cat. No.92CH3252-4), Minneapolis, MN, USA, 1993.

[55] J. Laroche, Y. Stylianou, and E. Moulines, “HNM: a simple, efficient harmonic+noise model for speech,” presented at IEEE Workshop on Applications of Signal Processing to Audio and Acoustics, New York, NY, 1993.

[56] J. Laroche, “The use of the matrix pencil method for the spectrum analysis of musical signals,” *Journal of the Acoustical Society of America*, vol. 94, num. 4, pp. 1958-65, 1993.

[57] J. Laroche and J. L. Meillier, “Multichannel excitation/filter modeling of percussive sounds with application to the piano,” *IEEE Transactions on Speech and Audio Processing*, vol. 2, num. 2, pp. 329-44, 1994.

[58] J. Laroche, personal communication, 1999.

[59] G. H. Wakefield, “Time-Pitch Representations: Acoustic Signal Processing and Auditory Representations,” presented at IEEE International Symposium on Time-Frequency/Time-Scale, Pittsburgh, PA, 1998.

[60] R. C. Boulanger, “The Csound Book : Perspectives in Software Synthesis, Sound Design, Signal Processing, and Programming,” . Cambridge, MA: MIT Press, 1999.

[61] B. Vercoe and others, “The Csound Manual (version 3.47): A Manual for the Audio Processing System and Supporting Programs with Tutorials,” 1997. http://mitpress.mit.edu/e-books/csound/csoundmanual/TITLE.html

[62] T. Erbe, “SoundHack user's manual version 0.880,” http://shoko.calarts.edu/~tre/SndHckDoc

[63] X. Serra, J. Bonada, P. Herrera, and R. Loureiro, “Integrating complementary spectral models in the design of a musical synthesizer,” presented at International Computer Music Conference, Thessaloniki, Hellas, 1997.

[64] J. Morrison, “"EA IFF 85" Standard for Interchange Format Files,” 1985. http://www.textfiles.com/programming/FORMATS/ea.iff

[65] B. DuCharme, *XML: The Annotated Specification*. Upper Saddle River, NJ: Prentice-Hall, 1999.

[66] F. Yergeau, “RFC 2279: UTF-8, a transformation format of ISO 10646,” 1998.

http://www.faqs.org/rfcs/rfc2279.html

[67] NIST, “The NIST Reference on Constants, Units, and Uncertainty: International System of Units (SI),” 1999. http://physics.nist.gov/cuu/Units/index.html

[68] T. Erbe, “aiff-c information,” 1999. http://shoko.calarts.edu/~tre/AIFFC/

[69] Apple Computer, “Finder Interface,” in *Files (Inside MacIntosh)*. Reading, MA: Addison-Wesley, 1992. http://developer.apple.com/techpubs/mac/Toolbox/Toolbox-447.html

[70] World Wide Web Consortium, “XML Linking Language (XLink): Working Draft,” 1998. http://www.w3.org/TR/WD-xlink

[71] T. Berners-Lee, L. Masinter, and M. McCahill, “RFC1738: Uniform Resource Locators (URL),” 1994. http://www.faqs.org/rfcs/rfc1738.html

[72] M. Wright and A. Freed, “Open Sound Control: A New Protocol for Communicating with Sound Synthesizers,” presented at International Computer Music Conference, Thessaloniki, Greece, 1997.

[73] M. Wright, “Implementation and Performance Issues with OpenSound Control,” presented at International Computer Music Conference, Ann Arbor, Michigan, 1998.

[74] E. Terhardt, “Psychoacoustic evaluation of musical sounds,” *Perception & Psychophysics*, vol. 23, pp. 483-492, 1978.

[75] A. Freed, “Real-Time Inverse Transform Additive Synthesis for Additive and Pitch Synchronous Noise and Sound Spatialization,” presented at AES 104th Convention, San Francisco, CA, 1998.

[76] D. Schwarz, “Spectral Envelopes in Sound Analysis and Synthesis,” Diplomarbeit Nr. 1622, Fakultät Informatik, Universität Stuttgart, Stuttgart, Germany, 1998.

[77] X. Rodet, Y. Potard, F. Iovino, G. Eckel, P.-F. Baisnée, Y. Potard, J.-B. Barrière, A. Lefèvre, and D. Virolle, “Chant home page,” 1999. http://www.ircam.fr/produits/logiciels/log-forum/chant-e.html

[78] X. Rodet, “The Diphone program: New features, new synthesis methods, and experience of musical use,” presented at International Computer Music Conference, Thessaloniki, Hellas, 1997.

[79] A. Chaudhary, A. Freed, and L. A. Rowe, “OpenSoundEdit: An Interactive Visualization and Editing Framework for Timbral Resources,” presented at International Computer Music Conference, Ann, Arbor, Michigan, 1998.

[80] A. Chaudhary, A. Freed, and M. Wright, “An Open Architecture for Real-Time Audio Processing Software,” presented at Audio Engineering Society 107th Convention, 1999.

[81] E. D. Scheirer, “Structured Audio and Effects Processing in the MPEG-4 Multimedia

Standard," *Multimedia Systems*, vol. 9, num. 1, pp. 11-22, 1999.

[82] E. D. Scheirer and B. L. Vercoe, "SAOL: The MPEG-4 Structured Audio Orchestra Language," *Computer Music Journal*, vol. 23, num. 2, 1999.

[83] M. Mathews, *The Technology of Computer Music*. Cambridge, Massachusetts: MIT Press, 1969.

[84] M. Wright and E. Scheirer, "Cross-Coding SDIF into MPEG-4 Structured Audio," presented at International Computer Music Conference, Beijing, China, 1999.

[85] M. Wright, S. Khoury, R. Wang, and D. Zicarelli, "Supporting the Sound Description Interchange Format in the Max/MSP Environment," presented at International Computer Music Conference, Beijing, China, 1999.

[86] A. Chaudhary and A. Freed, "A Framework for Editing Timbral Resources and Sound Spatialization," presented at Audio Engineering Society 107th Convention, 1999.

[87] A. Freed and A. Chaudhary, "Music Programming with the new Features of Standard C++," presented at International Computer Music Conference, Ann, Arbor, Michigan, 1998.

[88] P. Ion, R. Miner, S. Buswell, S. Devitt, A. Diaz, N. Poppelier, B. Smith, N. Soiffer, R. Sutor, and S. Watt, "Mathematical Markup Language (MathML) 1.0 Specification," 1998. http://www.w3.org/TR/REC-MathML/