SALIERI

A General, Interactive Computer Music System

Holger H. Hoos, Jürgen Kilian, Kai Renz, and Thomas Helbich

Darmstadt University of Technology
Wilhelminenstr.7, D-64283 Darmstadt, Germany
email: {hoos,kilian,renz,helbich}@iti.informatik.tu-darmstadt.de

Abstract. In this paper we describe the SALIERI System, an interactive software environment for structure oriented composition, manipulation, and analysis of music. The system is built on the newly developed SALIERI Language, a universal programming language based on a hierarchical model of formal music representation, combining features of traditional functional and procedural programming languages with powerful concepts for manipulating musical material. The basic musical data types of SALIERI are built on GUIDO Music Notation, a novel approach for adequately representing score-level music. Extensions of the core SALIERI System realise real-time functionality and a graphic user interface builder (VISCO).

1 Introduction

Since the first digital computers became available in the 1950s, a broad range of musical applications have been proposed and realised. With the introduction of the Personal Computer in the 1970s this development has considerably gained momentum, giving rise to the field commonly known as *computer music*. Until today, a huge number of tools and systems for different application areas of computer music such as music composition, analysis, perception (and many more) have been created, and most of them vanished as quickly as they have shown up. Only a small number of systems, like CSound [Vercoe; 1993], Max [Puckette; 1991], or Patchwork [Laurson; 1996] are widely kwown and used. One reason for this situation might be the fact that most systems are very specialised and therefore have a very limited range of application. Furthermore, many systems were only prototypically realised as research tools; these are usually difficult to use for people other than their developers.

Our motivation in designing the SALIERI System was to overcome these drawbacks by creating a powerful, interactive environment for realising a broad range of musical applications in an adequate way. Thus, the SALIERI Language, on which the system is based, comprises all elements of a universal programming language as well as powerful concepts for representing and manipulating musical material. Of course, we do not claim that SALIERI can be adequately applied to problems from *all* areas of computer music. For example, in its present form the system is based on a score-level representation of music; therefore it is certainly not adequate for physical-level applications like the composition or analysis of electroacoustic music. Nevertheless, we feel that there is a wide field of application in music theory, composition, and education.

The remainder of this paper is structured in the following way: In Section 2, we give some theoretical background on music representation and manipulation in SALIERI and compare our approach with related work. Section 3 introduces the SALIERI Language and gives some very simple examples for its usage. In Section 4, we describe the SALIERI System and its various components; we also discuss the highly modular architecture of the system. Then, we give a brief overview of some extensions to the basic system, in particular the real-time module and the graphic user interface builder (VISCO). Finally in Section 6, we sum up the main characteristics of the SALIERI System and roughly sketch some future developments and applications.

2 Background and Related Work

To allow for manipulation by a computer program, musical material has to be represented in a formal language. We distinguish three fundamentally different approaches for designing such formal languages. The first, and probably most common one is based on using the data structures of an existing, universal programming language for representing musical data. Examples for this approach are OpenMusic [Assayag *et.al.*; 1997] and Common Music [Schottstaedt; 1997], which are both based on LISP. A second approach is to use a native music representation for realising musical data types which are embedded into a universal programming language that is used for manipulating these. The SALIERI System is based on a variant of this approach: Here, the music representation language is GUIDO Music Notation (GMN), a novel approach for adequately representing score-level music [Hoos, Hamel; 1997] [Hoos *et.al.*; 1998]. GUIDO is the basis for the musical data types of the newly designed SALIERI Language, a universal programming language specifically designed for manipulating score-level music.

One of the advantages of this approach is that, differently from the first one, the musical data types do not have to be mapped onto existing data structures provided by programming models which have not been developed with musical applications in mind. Thus, there is no hidden representation which usually has to be taken into account by advanced or expert users who start pushing the limits of a computer music language or system. The down-side of this approach is the fact that designing and implementing a new programming language is usually more difficult than extending or adapting an existing one to a specialised application area. Nevertheless, it is our belief that musical applications benefit a lot from providing carefully designed, native musical data types and language concepts. In particular, according to our experience so far, users which do not have an extensive background in programming usually have great difficulties when dealing with languages like LISP.

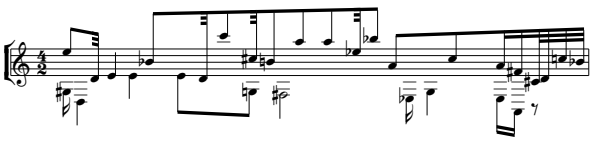
There is a third approach which is based on extending a music representation language into a full-fledged, specialised musical programming language by providing functional abstraction and application. However, the only system based on this approach which we are currently aware of, is the Elody system [Orlarey *et.al.*; 1997]. While this approach is formally very elegant and certainly interesting for certain applications, it is not clear whether this concept will be adequate for realising a broad range of musical applications. In contrast, the SALIERI approach is based on distinguishing between the music representation level and the music manipulation level. One reason for this is our belief that in the context of many applications treating these levels differently better reflects the intuitive ways of thinking about a musical problem.

Therefore, when designing the SALIERI System, one of the primary goals was *adequacy* in the sense of allowing a broad variety of musical applications to be realised in a straight-forward, intuitive way which enables the user to concentrate on musical aspects while avoiding implementation overhead as far as possible. Other important design principles like *consistency* and *minimality* in combination with a thouroughly designed graphical user interface help to ensure the usability of the system. Finally, *flexibility* and *extensibility* ensure that the system can be applied to a broad range of tasks from computer music.

3 The SALIERI Language

The SALIERI Language combines features of conventional procedural and functional programming languages (such as PASCAL or LISP) with powerful mechanisms for representing and manipulating musical data. SALIERI is based on a dynamic typing system which supports function overloading and polymorphism. There are two basic musical data types which are based on GUIDO Music Notation [Hoos *et.al.*; 1998]: *Sequences* are representations of one voice pieces, while *segments* represent multi-voice music. The basic operations on sequences and segments are *concatenation* and *polyphonic aggregation*; these realise horizontal and vertical composition of musical material and provide an essential basis for the adequate representation of important structural aspects. Fundamental musical operations on sequences and segments include primitive variations (such as transposition, retrograde, ...) and functions to extract

```
randomMusic := FUNC(2,
 `(* Creates series of notes with
     random duration/pitch *)

  numnotes := $1; (* first parameter *)
  seednote := $2; (* second parameter *)

  music := [];

  loopn(numnotes,
    `denominator := INT(exp(2, rndUniform( 1, 4 ) ));
     note := transp(seednote,  rndUniform( -12, 12));
     note := expand(note, 1, denominator);
     music := music + note;
    ');

  return(music)');

(* Creates a song with two voices *)

randomSong := poly(randomMusic( 20, [c2*1/2] ),
                   randomMusic( 10, [g0*1/1] ));
```



```
(* structural description of frere jacques *)

a := repeat(2, [ \slur( c1*1/4 d e c ) ] );
b := repeat(2, [ \slur( e*1/4 f g/2 ) ] );
c := repeat(2, [ g*1/8 a g f e/4 c ]);
d := repeat(2, [ c1*1/4 g0 c1/2 ]);

song := a + b + c + d;

voice1 := song + a;
voice2 := rest(a) + song ;

canon := poly( transp( voice1, -12) ,
               voice2 );
```

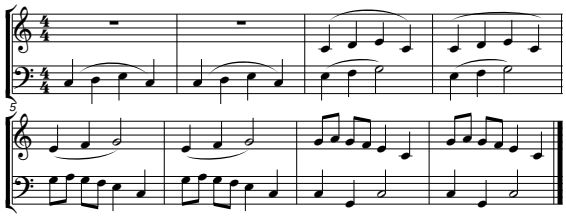


Figure 1: Simple examples of SALIERI programs

information from musical data (like ambitus, duration, ...).

Other object-types in SALIERI include the commonly used simple types `boolean`, `integer`, `real` and `string`. The complex data-type `list` implements heterogeneous lists of objects, thereby forming the basis for more sophisticated data-structures which are needed in individual, problem-specific contexts. Since part of the SALIERI Language is based on the functional programming paradigm, functions and macros are regarded as ordinary objects which can be stored in lists or passed as parameters, thus allowing the flexible realisation of higher-order concepts (as can be used for aleatoric music generation or variation).

One of the most important aspects of SALIERI is its set of thoroughly selected, powerful built-in functions. Among these, complex musical operations can be found, such as tonal transposition based on generalised scales or transcription, which allows the translation of music material between different keys and generalised scales. Of course, the SALIERI built-in functions also include common arithmetic, comparison-, set-, list-, and i/o-operations, as well as control flow functions like conditional and loop structures. By extending the system with new, user-defined functions, the expert user can easily adapt SALIERI to her specific needs. Using native musical data-types and built-in functions allows to significantly reduce programming overhead compared to conventional programming languages such as LISP or C. This is further faciliated by a number of libraries that can be dynamically linked to the system. These include more specific, complex musical functions for various application areas like chord handling, music analysis, or aleatoric and deterministic composition techniques.

Figure 1 shows two very basic examples on using the SALIERI Language. The first example (left side) demonstrates how a user-defined function for creating simple aleatoric melodies can be defined and used: `randomMusic` generates a sequence of random notes, its two parameters are the number of notes in the sequence and a seed note. The notes of the resulting sequence are generated by applying a random expansion factor (duration) and a random transposition level to the seednote. The example also illustrates the usage of `randomMusic`: a two-voice segment `randomSong` is calculated by calling `randomMusic` twice with different parameters and combining the resulting sequences using the constructor `poly`. One result of this call can be seen on the bottom left side of Figure 1.

The second example (right side) shows, how the SALIERI language can be used to represent musical pieces using structural information. The well-known french canon "Frère Jacques" is defined using four motifs,

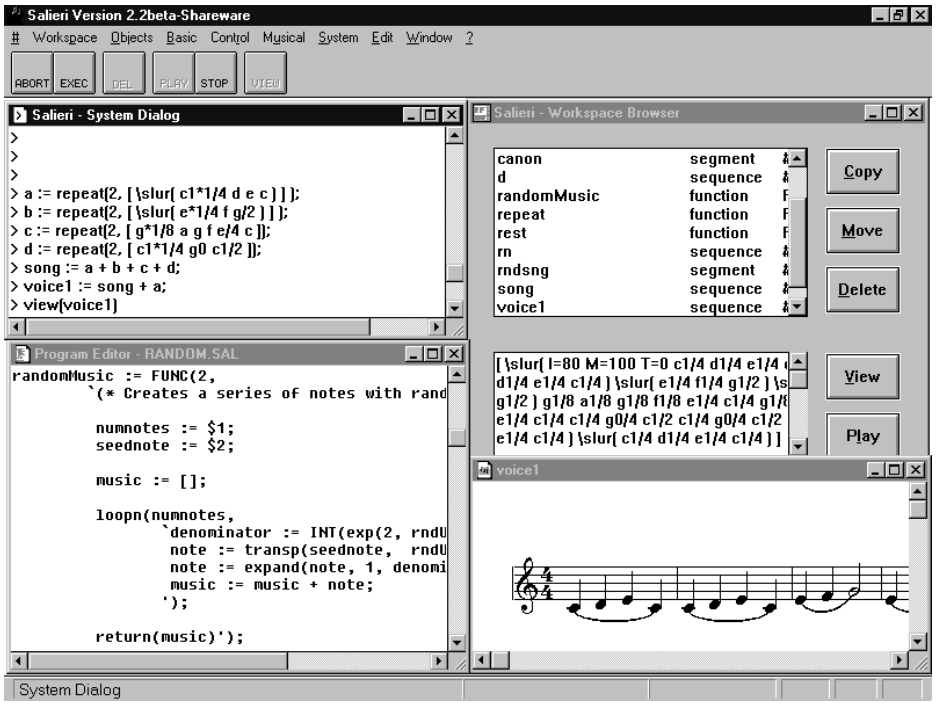


Figure 2: Screen-shot of the SALIERI System

each of which is repeated twice. In SALIERI, the musical motifs can be represented using GUIDO Music Notation; these building blocks can then be manipulated using built-in or user-defined functions. In the example, the variables `a,b,c,d, voice1,voice2`, and `song` are defined according to the structure of the song. Finally, a canon is calculated by combining two versions of the melody, one of which is transposed by an octave (first voice) while the other starts after a rest with the duration of fragment `a`. The equivalent score is shown on the bottom right side of Figure 1. More complex examples for using the SALIERI language can be found at the SALIERI Website[1] or directly obtained from the authors.

4 The SALIERI System

The SALIERI System offers a complete, interactive working environment for creating, manipulating, and analysing musical material. All objects are held in a persistent global workspace, which allows the user to restart a new session exactly where the previous one was quitted. To combine the interactive nature of the system with the powerful concepts and features of the language while achieving a reasonable performance, the SALIERI Language is compiled into an intermediate code (SALIERIIntermediate Code) and then executed on a virtual machine (SIC machine).[2] This compiler and the virtual machine are the core of the SALIERI System. Different from conventional compiled programming languages like C, in SALIERI the compilation process is transparent to the user, so that the system basically behaves like a very fast interpreter.

To illustrate the structure of the graphical user interface, Figure 2 shows a screen shot of the SALIERI System. Via the System Dialog (upper left window in Figure 2), the user can input commands and queries and read the system's answers. For an easy management of objects, Workspace Browser windows (upper

[1] `http://www.informatik.tu-darmstadt.de/AFS/SALIERI`
[2] In this respect, SALIERI is based on a similar concept like the programming language JAVA.

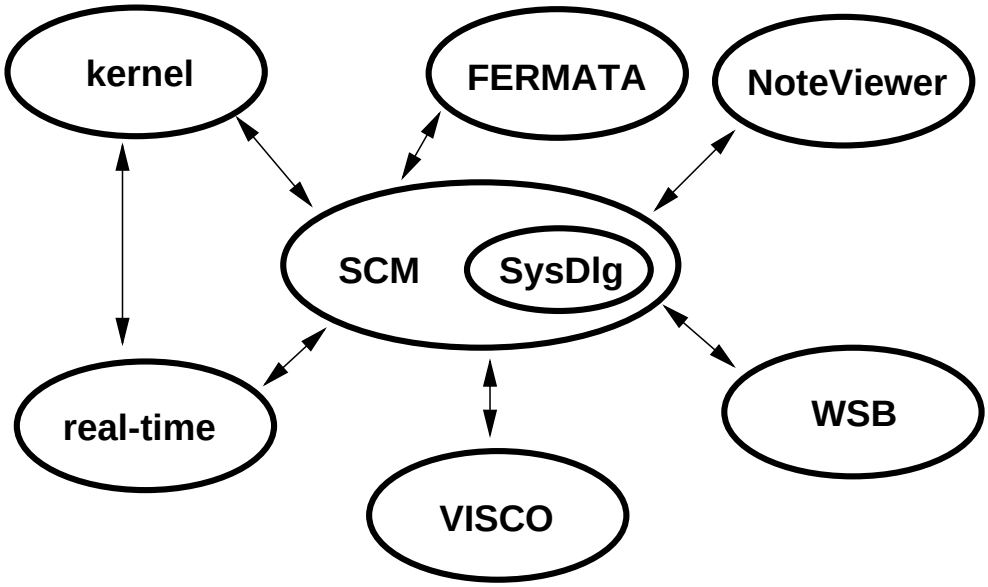


Figure 3: SALIERI System architecture, based on ORCA.

right window in Figure 2) are available. These display objects by their names, types and abbreviated values; for the currently selected object, the complete value is displayed. The Workspace Browser allows easy copying, renaming and deleting of selected objects, as well as playing and viewing musical material in conventional music notation. The latter is realised by NoteViewer windows (lower right window in Fig. 2), which do not only display note sequences and segments but also allow the manipulation of parameters like clefs, meter or metrical subdivision. When developing applications or additional libraries, Program Editor windows (lower left window in Figure 2) can be used for writing and editing SALIERI program code; Program Editors also support the direct execution of selected parts of SALIERI programs. In Program Editors, as well as in the System Dialog, all internal functions can be accessed by a menu based language support (see top of Figure 2). The SALIERI System and Language are documented by an extensive HTML-based help system, which can also be accessed via the WWW.

The implementation of the SALIERI System is based on ORCA (Open Reactive Components Architecture), a highly modular message-passing architecture developed by our group (see Figure 3). The message traffic between the different components of the system uses the text-based, synchronous ORCA protocol; all communication between modules are routed by the SCM (SALIERI Components Manager).[3] The SCM allows to dynamically load or remove other components at runtime without restarting the system (plugin mechanism). These components (like the Workspace Browser) are implemented as SIAM's (SALIERI Integrated Application Modules). Each SIAM supports a basic set of ORCA messages and provides an interface for communicating with the SCM. The SALIERI System can be easily extended by implementing new SIAM modules and loading them into the system. In future, we plan to support distributed realisations of the ORCA architecture; then, SIAMs could run on different machines and communicate via the internet.

[3]For peformance reasons, there is one exception: time-critical information is directly passed between the real-time module and the kernel.

5 Extensions

In this section we shortly describe two extensions of the basic SALIERI System which have been recently realised. The *Real-time Extension* supports real-time processing of music in the SALIERI System. Our basic approach is based on MidiShare [Orlarey, Lequay; 1989] and realised by a simple extension of the SALIERI Language. The extension provides new datatypes representing real-time events and handler functions which are triggered by these. Real-time events cover not only MIDI events, but also clock-, keyboard-, and other internal system events. MIDI events can be received from and sent to any MidiShare port or application, thus allowing to effectively connect the SALIERI System to other MidiShare applications. Real-time handler functions are user-definable functions of the SALIERI Language which are activated by real-time events. Thus the full expressive power of the SALIERI Language can be used to manipulate real-time events in a straight-forward, adequate way and within a consistent and easy-to-learn formal framework.

The real-time extension can be used to realise a variety of interesting applications, ranging from simple MIDI filters to complex real-time processing like meter and key detection, harmonic and motivic analysis, and variation of real-time input. We believe this approach to be complementary, in a sense, to the MAX approach [Puckette; 1991]: since SALIERI is based on a universal, textual programming language specialised for music processing, complex program structures which tend to be difficult to realise within the visual programming paradigm underlying MAX-like systems can be represented more adequately. At the same time, based on MidiShare, it is very easy to connect SALIERI to other MIDI-based real-time systems and thus to combine the advantages of different approaches.

VISCO (Visual SALIERI Components) is a SALIERI extension which allows the realisation of integrated graphical user interfaces for SALIERI applications. The VISCO extension is realised as a SALIERI Integrated Application Module (SIAM) which can be easily loaded into the system at runtime. It supports the usual visual control elements (windows, buttons, edit boxes, ...) and also some special controls (e.g., noteview boxes, ...), which are organised in a hierarchical model. Each control element can be connected to a set of SALIERI command strings (actions) which are triggered by different types of events, like mouse clicks. Further properties control the graphical appearance (like size and position) of the visual controls.

We fully integrated VISCO into the SALIERI System by extending the SALIERI Language with a few new functions. The visual control elements and environments can be created and controlled using the integrated, interactive, GUI based VISCO Browser, from within running SALIERI programs, or from the SALIERI System Dialog (command line interface). VISCO provides an easy way for realising customised graphical user interfaces for SALIERI applications. At the same time, user-defined VISCO controls can be used to extend or customise the SALIERI System GUI. VISCO also provides the basis for extending the SALIERI System with visual programming capabilities in the spirit of MAX or OpenMusic.

6 Conclusion and Future Work

In this work, we presented SALIERI, a newly developed, interactive software environment for structure oriented composition, manipulation, and analysis of music. We gave some background on our approach and compared it to related work, pointing out the benefits we see in basing our system on a native musical programming language combining features from procedural and functional programming paradigms. We then discussed the main characteristics and design objectives of the SALIERI musical programming language and the SALIERI System. We gave an overview over the highly modular ORCA system architecture which faciliates porting the core system to different platforms. At the same time, ORCA provides a convenient interface for extending the SALIERI System with integrated application modules (SIAMs). We also discussed two important extensions to the basic SALIERI System, the real-time module and the graphic user interface builder (VISCO).

Due to the limited space, we could only give a slight flavour of musical programming with SALIERI here. Nevertheless, SALIERI has been successfully applied to a number of problems from areas such as

- analysis of music in the style of Palestrina (see also [Lüttig; 1994]);
- algorithmic composition of music based on mathematical concepts (see also [Allouche, Johnson; 1996], [Moore; 1990]);
- automated generation of variations [Chico-Töpfer; 1998];
- using formal grammars to generate etudes for string instruments [Boivin, Möller; 1996], [Franke, Mendez; 1995] (see also [Laske; 1974], [Roads; 1979]);
- harmonic and rhythmic analysis based on formal theories of tonal music [Flade, Weber; 1995] (see also [Schenker; 1906], [Lerdahl, Jackendoff; 1993]).
- applications for musical education and training.

Other applications are currently being developed and will be presented elsewhere. Our experience so far shows that the SALIERI System in its present form can be regarded as a versatile and powerful tool for realising a wide variety of approaches and techniques from different areas of computer music.

Currently, we are developing SALOME (SALIERI Object Manipulation Environment), an advanced graphical user interface for the SALIERI System based on a simple, yet powerful and very consistent graphical metaphor for manipulating objects. In cooperation with researchers from the Dresden University of Technology, we are working on a SALIERI extension for handling microtonality and dynamic tuning (based on the existing MUTABOR II system [Abel *et.al.*; 1992]). Furthermore we have recently begun implementing a JAVA version of the SALIERI System which will allow to use the system with different operating systems and on different hardware platforms. This development will also affect the WWW Server version of SALIERI which is presented in a separate paper [Renz, Hoos; 1998]. Future plans include extending SALIERI to support physical level descriptions and digital signal processing.

Acknowledgements

In the first place, we wish to thank Prof. H.K.-G. Walter for his support of the project since its earliest stages. Thanks to him we have been able to build up the project group in Darmstadt and develop our ideas in a stimulating environment. Other members of our group who helped this project to become a success are Peter Lüttig, Stefan Franke, Michael Fuhlbrügge, Matthias Streicher, and Matthias Huber. Last not least, Holger H. Hoos wishes to express his gratitude to the Intellectics Group and Prof. W. Bibel for supporting his work on this project.

References

[Abel *et.al.*; 1992] Volker Abel, Peter Reiss, Rudolf Wille; MUTABOR II: Ein Computergesteuertes Musikinstrument zum Experimentieren mit Stimmungslogiken und Mikrotönen; Technische Hochschule Darmstadt, Fachbereich Mathematik, Preprint Nr. 1513; Darmstadt 1992

[Allouche, Johnson; 1996] Jenan-Paul Allouche, Tom Johnson; Narayana's Cows and Delayed Morphisms; Proc. JIM-96

[Assayag *et.al.*; 1997] Gerard Assayag, Carlos Agon, Joshua Fineberg, Peter Hanappe; An Object Oriented Visual Environment For Musical Composition; Proc. ICMC-97

[Boivin, Möller; 1996] Michèle Boivin, Clara Möller; Grammatikbasierte Musikerzeugung; Seminarausarbeitung zum Seminar Computermusik; TH Darmstadt 1996

[Chico-Töpfer; 1998] Wolfgang Chico-Töpfer; AVA – A Grammar/Case-based Composition System to Variate Music Automatically Through the Generation of Scheme Series; Proc. ICMC-98

[Flade, Weber; 1995] Kai Flade, Jens Weber; Generative Theorie Tonaler Musik; Seminarausarbeitung zum Seminar Computermusik; TH Darmstadt 1995

[Franke, Mendez; 1995] Stefan Franke, Martha Méndez; Grammatikbasierte Musikerzeugung und -beschreibung; Seminarbericht zum Seminar Computermusik; TH-Darmstadt 1995

[Hoos, Hamel; 1997] Holger H. Hoos, Keith Hamel; The GUIDO Music Notation Format – Specification Part 1; Technical Report TI 20/97; Darmstadt University of Technology; Darmstadt 1997; available from `http://www.informatik.tu-darmstadt.de/AFS/CM/GUIDO/docu/`

[Hoos *et.al.*; 1998] Holger H. Hoos, Keith A. Hamel, Kai Renz, and Jürgen Kilian; The GUIDO Notation Format – A Novel Approach for Adequately Representing Score-Level Music; Proc. ICMC-98

[Laske; 1974] Otto E. Laske; In search of a Generative Grammar for Music; Sonological Reports, No. 1, Institute of Sonology; Utrecht, Netherlands 1974; *in:* Machine Models of Music; Stephan M. Schwanauer, David A. Levitt; MIT Press; Cambridge, Massachusetts 1993

[Laurson; 1996] M. Laurson; Patchwork, A Visual Programming Language and Some Musical Applications; Sibelius Academy, Studia Musica No. 6, Helsinki 1996

[Lerdahl, Jackendoff; 1993] Fred Lerdahl, Ray Jackendoff; An Overview of Hierarchical Structure in Music; *in:* Machine Models of Music; Stephan M. Schwanauer, David A. Levitt; MIT Press; Cambridge, Massachusetts 1993

[Lüttig; 1994] Peter Lüttig; Der Palestrina-Stil als Satzideal in der Musiktheorie zwischen 1750 und 1900; Hans Schneider; Tutzing 1994

[Moore; 1990] F. Richard Moore; Elements of Computer Music; Chapter 5: Composing; Prentice Hall 1990

[Orlarey *et.al.*; 1997] Yann Orlarey, Dominique Fober, Stéphane Letz; L'environnement de composition musicale Elody; Proc. JIM-97

[Orlarey, Lequay; 1989] Yann Orlarey, H. Lequay; MidiShare: a real-time multi-tasks software module for MIDI applications; Proc. ICMC-89; ICMA Publishing, San Francisco

[Puckette; 1991] M. Puckette; Combining Event and Signal Processing in the Max Graphical Programming Environment; Computer Music Journal 15(3):68-77; MIT 1991

[Renz, Hoos; 1998] Kai Renz, Holger H. Hoos; A HTTP Interface to Salieri; Proc. ICMC-98

[Roads; 1979] Curtis Roads; Grammars as Representations for Music; Computer Music Journal Vol. III, Nr. 1; MIT 1979

[Schenker; 1906] Heinrich Schenker; Harmonielehre; fotomechanischer Nachdruck der Ausgabe von 1906; Universal Edition A. G.; Wien 1978

[Schottstaedt; 1997] Bill Schottstaedt; Common Music Notation; *in:* Beyond MIDI; The Handbook of Musical Codes; E. Selfridge-Field (ed.); pp. 217-221; The MIT Press; Cambridge Massachusetts, London England 1997

[Vercoe; 1993] Barry Vercoe; Csound: A Manual for the Audio Processing System and Support Programs with Tutorials; 2nd revised edition; Cambridge MA; Massachusetts Institute of Technology Media Lab 1993