ARTIST
An AI-based tool for the design of intelligent assistants for sound synthesis

Eduardo Reck Miranda,
AI/Music Group,
Faculty of Music and Dept. of Artificial Intelligence,
University of Edinburgh,
12, Nicolson Square,
Edinburgh, EH8 9DF,
Scotland, UK.
E-mail: miranda@music.ed.ac.uk

Abstract

In this paper we introduce the fundamentals of ARTIST (an acronym for Artificial Intelligence-aided Synthesis Tool). ARTIST is a tool for the design of intelligent assistants for sound synthesis that allow composition of sounds thought of in terms of qualitative descriptions (e.g. words in English) and intuitive operations rather than low level computer programming. Our research work is looking for (a) plausible strategies to map the composer’s intuitive notion of sounds to the parametric control of electronic sound synthesis and (b) how to provide artificial intelligence (AI) to a synthesiser. In this paper we introduce how we attempted to approach the problem by means of a compilation of a few well known expert systems design techniques used in AI research. ARTIST is a prototype system which embodies the results of our investigation so far.

Keywords: AI-based synthesiser, knowledge-based systems, machine learning

Introduction

In the final quarter of the 20th century the invention of sound recording followed by sound processing and then sound synthesis have changed our view of what constitutes music. These recent developments have vastly expanded our knowledge of the nature of sounds. Nowadays, computer technology offers composers the most detailed control of the internal parameters of sound synthesis and signal processing.

Working the effective use of the new technology, composers become more ambitious, but the complexity also increases. The scale and nature of the compositional task changes, technically and aesthetically. Theoretically the computer can be programmed to generate any sound one can imagine. But, on the other hand, this can get composers into trouble. Quoting Barrière (1989, pp. 116), “it is too easy to fail to take various consequences into account, to get technology side-tracked by a tool whose fascination complexity can become a disastrous mirage”.

Even if the composer knows the role played by each single parameter for synthesising a sound, the traditional way of working with computer synthesis, tediously entering exact data at the terminal, is not particularly stimulating. We are convinced that higher processes of inventive creativity and musical abstraction are often prejudiced in such a situation. In this case we think that the computer is being used as a kind of word processor combined with player piano, and not as a creative tool. We have come to believe that this can be improved by means of an appropriate coupling between human imagination and artificial intelligence (AI).

In this paper, we introduce the fundamentals of ARTIST (an acronym for Artificial Intelligence-aided Synthesis Tool). ARTIST is a tool for the design of intelligent assistants for sound synthesis that allow composition of sounds thought of in terms of intuitive qualitative descriptions (e.g. words in English) rather than low level computer programming (Miranda et al., 1993a; 1993b; Miranda, 1994a; 1994b; 1994c). By an intelligent assistant we mean a system which works cooperatively with the user by providing useful levels of automated reasoning in order to support laborious and tedious tasks (such as working out an appropriate stream of synthesis parameters for
each desired single sound), and to aid the user to explore possible alternatives when designing a sound. The desirable capabilities of such a system can be summarised as follows:

(a) The ability to operate the system by means of an intuitive vocabulary instead of sound synthesis numerical values,

(b) The ability to customise the system according to the user's particular needs, ranging from defining which synthesis technique(s) will be used to defining the vocabulary for communication,

(c) The encouragement of the use of the computer as a collaborator in the process of exploring ideas,

(d) The ability to aid the user in concept formation, such as the generalisation of common characteristics among sounds and their classification according to prominent attributes, and

(e) The ability of creating contexts which augments the chances of something unexpected and interesting happening, such as an unimagined sound out of an ill-defined requirement.

Apart from graphic workstations (such as the UPIC system (Xenakis, 1992; Marino et al., 1993; Lohner, 1980) and medium level programming languages (see Pennycook, 1985) for a survey), little research has been done towards a system for sound synthesis that responds to higher levels of sound description. An early attempt at the definition of a grammar for sound synthesis was made by Holtzman (1978) at Edinburgh University. Also, Lawson (1985) has proposed - not implemented on a machine though - a kind of vocabulary for sound composition based on his theory of sound colour which, he believes, he made up from Helmholtz's theory of vowel qualities of tones (Helmholtz, 1885). Lerdahl (1987) too has done some sketches towards a hierarchical perceptually-oriented description of timbre. Apart from these, it is worth mentioning that there have been a few attempts towards signal processing systems that understand natural language. The most successful ones are interfaces developed to function as a front end for systems which perform tasks to do with audio recording studio techniques such as, mixing, equalisation, and multitracking (e.g. CDM (Schmidt, 1987) and Eiltar (Garton, 1989)). More recently, Ethington and Punch (1994) proposed a software called SeaWave. SeaWave is an additive synthesiser (Dodge and Jersie, 1985) in which sounds can be produced by means of a vocabulary of descriptive terms. Although of a limited scope, SeaWave profits an excellent insight and it seems to work well. Verteget and Bonis (1994) have also been working towards a cognitive-oriented interface for synthesizers.

We begin the paper by introducing the problem from a musician's point of view. Then we introduce the signal processing of the synthesiser which will be used as an example study. After this, we indicate some methods for describing sounds by means of their attributes and suggest a technique for mapping those attributes onto the parameters of a synthesiser. Then we study how this technique works and present some examples. Here, we also study the utility and the functioning of machine learning in this kind of system. Finally, we propose a system architecture which embodies all the concepts discussed so far and introduce its functioning through examples. We end the paper with some final remarks and ongoing work.

An example study synthesiser

Assume that we wish a synthesiser which is able to produce human voice-like sounds. It is worth mentioning that to produce a perfect simulation of the human vocal tract is out of the scope of this paper. Thus, rather than making a description of the fundamental aspects of the phenomenon by means of a set of equations (e.g. (Woodhouse, 1992; Keele, 1992)), we opted for observing it by means of a more traditional formant modelling technique which uses subtractive synthesis (Flanagan, 1984; Klatt, 1990; Sundberg, 1991; Miranda, 1992; 1993). We come to believe that this level of description (see also (Fiedler, 1992) for a brief discussion about this business) suffices at this moment. The signal processing diagram of our example study synthesiser is shown in Figure 1.

Each block of the diagram (except the Envelope) is composed of several signal processing units (SPU). A composition of SPU's form sub-blocks within a block. Sub-blocks in turn may constitute sub-sub-blocks, and so forth. The Voicing Source, for example, has two sub-blocks: one, the Vibe (vibrator), contains an oscillator unit, and the other, the Pulse Generator sub-block, contains a pulse generator unit (Figure 2). Each SPU needs parameter values for functioning. We say that, in order to produce a certain

Describing sound by means of their attributes

There have been several studies defining a framework to systematically describe sounds by means of their attributes (Schafer, 1966; von Bismark, 1971; 1974a; 1974b; Cogan, 1984; Giomi & Ligabue, 1992; Carpenter, 1990; Terhardt, 1974) to cite but a few. They are derived mainly from work in the fields of both psychometrics and musical analysis. We classify these studies in two approaches: on the one hand, the psychoacoustically-based approach; and on the other hand, the perceptually-oriented approach. As it is not our aim to survey all these, we have selected one example of a formalised psychoacoustically-based approach to sound description, the PEASER approach (Perception Exemplarry and Sound Exemplar).
The source-filter model: a device-oriented approach

The source-filter model formulates that the characteristic of a sound is determined by its spectrum envelope's pattern. This pattern is composed of multiple hills called formants. Each formant has a centre frequency peak and a bandwidth. According to this model, the lowest two formants are the most significant determinants of sound quality.

The pattern of the spectrum envelope of formant frequencies is thought of as the result of a complex filter through which a source sound passes.

We can define here a two-dimensional space whose axes are the first (f1) and the second (f2) centre formant frequencies respectively. Then, four perceptual attributes, namely openness, acuteness, smallness, and laxness (after Lashley, 1985; 1987), can be specified as categories of equal-values contours in this space. The attribute openness varies with f1, acuteness with f2, smallness with the sum of f1 + f2, and laxness varies towards a neutral position in the middle of the space (Figure 3).

Figure 3: Two-dimensional sound space.

Implementing a sound event by means of the schema

We have seen before (Figure 1) that the synthesiser is composed of several connected blocks (Voicing Source, Noise Source, etc.), one of each responsible for a certain sound attribute. We can now define a compound sound event by means of the ASS scheme. Each component of the sound event is responsible for a certain aspect of the sound quality.

The leaves of the sound event are slots corresponding to the several sound synthesis parameters. Slots are grouped into nodes of a higher level layer, which in turn are grouped into nodes of a higher level, and so forth, up to the root of the tree (the sound event).

Figure 5 shows a partial definition of a sound event of the synthesiser shown in Figure 1. Although not shown in this figure, the links among the sound event's components are labelled has_component. They represent the offspring relation among nodes.

The partial sound event definition shown in Figure 5 can be implemented in Prolog as shown below. Each clause represents a has_component relationship between two atoms. The first clause, for example, is read: 'A sound event has a component called voicing source'. A interpretation of the whole layer 1, for example, is: 'the sound event has two components named voicing source and formant resonators'.

% % % layer 1
% % % has_component(sound_event, voicing_source).
% % % has_component(sound_event, formant_resonators).
% % % layer 2
% % % has_component(voicing_source, vibrato).
% % % has_component(voicing_source, pulse_generator).
% % % has_component(formant_resonators, formant1).
% % % has_component(formant_resonators, formant2).
% % % layer 3
% % % has_component(vibrato, rate).
% % % has_component(vibrato, width).
% % % has_component(pulse_generator, frequency).
% % % has_component(formant1, f1).
% % % has_component(formant1, bw1).
% % % has_component(formant2, f2).
% % % has_component(formant2, bw2).
% % % has_component(formant3, f3).
% % % has_component(formant3, bw3).
% % % vibrato_rate
% % % vibrato_width
% % % fundamental_frequency
% % % 1st_formant_frequency
% % % 1st_formant_bandwidth
% % % 2nd_formant_frequency
% % % 2nd_formant_bandwidth
% % % 3rd_formant_frequency
% % % 3rd_formant_bandwidth
Figure 5: Partial sound event definition.

Sound event

voicing source

formant resonators

formant

rate

width

f0

f1

bw1

f2

bw2

f3

bw3

vibrato pulse generator

All the slots of the ASS must be filled in order to completely specify a sound. We say that a completely specified sound is an *assemblage*. For each different sound there is a particular assemblage. Thinking of this synthesizer as a (rough) model of the vocal tract mechanism, an assemblage would correspond to a certain position of the vocal tract in order to produce a sound.

Sound hierarchy and the inheritance mechanism

Recapitulating, we have defined a general abstract scheme for representing a sound. Then we defined and implemented the notion of the *sound event* by means of this scheme. We also introduced the idea of assemblage: It was explained that an assemblage occurs when all the slots of the scheme are properly filled. In this case, each assemblage corresponds to a particular sound.

In practice, sounds are represented in a knowledge base as a collection of slot values. In other words, the knowledge for the assemblage of a particular sound is clustered around a collection of slot values. An assemblage engine is then responsible for taking the appropriate slot values and ‘assembling’ the desired sound.

The following Prolog facts correspond to an example knowledge base which contains slot values for the (partial) *sound event* definition shown in Figure 5. Each clause represents a *slot*. It has two atoms: the first is a reference name and the second is a tuple. The reference name is an atom which identifies the affiliation of the slot, i.e. which cluster it belongs to. The first element of the tuple is the name of the slot and the second element is the value of the slot. This value can be either a number, a word, or a formula for calculating its value (those will be dealt later). This example knowledge base contains information about three sounds, namely *back vowel*, *front vowel*, and *vowel /a*/.

```
% % % back vowel
% % %
slot( vowel(back), [ rate, 5.2 ] ).
slot( vowel(back), [ width, 0.06 ] ).
slot( vowel(back), [ f0, 155.56 ] ).
slot( vowel(back), [ f1, 622.25 ] ).
slot( vowel(back), [ f2, 1244.5 ] ).
slot( vowel(back), [ f3, 2687 ] ).
slot( vowel(back), [ bw1, 74.85 ] ).
slot( vowel(back), [ bw2, 56 ] ).
slot( vowel(back), [ bw3, 131.85 ] ).
% % %
% % %
% % %
```

The notion of partial assemblage

We ought to make the assembler engine flexible so that it also may assemble single internal nodes of the schema. In other words, besides the assemblage of the whole scheme there might be
Let us observe again the example shown in Figure 5. It has a brunch of filters which constitute three formant resonators. Taking as an example only the node formant(1), we say that it needs only its affiliated slots (namely f(1) and bw(1)) for assemblage.

The advantage of being able to think in terms of assemblages of single nodes, as an alternative to the solely ASS root assemblage, is that now ones can attach non-numerical attribute values (i.e., words in English) to partial assemblages too. For instance, one could refer to the node formant(1) as low and wide if it has f(1) = 250 Hz and bw(1) = 200 Hz. This is also represented in the knowledge base as a cluster of slots. Example:

\[
\begin{align*}
\text{slot}(\text{formant}(1), \text{low and wide}, [f(1), 250]), \\
\text{slot}(\text{formant}(1), \text{low and wide}, [bw(1), 200]).
\end{align*}
\]

Now, for each node of the schema one can define a set of possible non-numerical attribute values. Back to figure 5, the slots rate, and width constitute a node called vibrato which in turn, with the node pulse generator, forms the higher level node voice source. One could establish here that the possible attribute values for vibrato are none, uniform, and too slow. Each of these attributes will then correspond to either a numerical value or to a range of values within a certain interval. For example, one could say that vibrato is none if rate \(\leq 0\) Hz, and width \(\leq 0\).%

The node voice source could be similarly defined: one could establish that voice source is steady low if vibrato = none and pulse generator = 55 Hz, for example.

Hypothetically considering only this left part of the example schema shown in Figure 4, a sound, say sound(a), could be described as having steady low voice source and none vibrato. See example below:

\[
\begin{align*}
\text{slot}(\text{vibrato}, \text{none}, [\text{rate}, 0]), \\
\text{slot}(\text{vibrato}, \text{none}, [\text{width}, 0]), \\
\text{slot}(\text{voice source}, \text{steady low}, [\text{vibrato}, \text{none}]), \\
\text{slot}(\text{voice source}, \text{steady low}, [\text{pulse generator}, 55]), \\
\text{slot}(\text{sound(a)}, [\text{voice source}, \text{steady low}]), \\
\text{slot}(\text{sound(a)}, [\text{vibrato}, \text{none}]).
\end{align*}
\]

The role of machine learning

In this section we will study the role played by two machine learning techniques in our proposed system, namely inductive learning and supervised deductive learning. Both well known techniques which have been satisfactorily used in expert systems (see Dietterich and Michalski, 1981; Quinlan, 1982; Winston, 1984; Bratko, 1990; Carbonell, 1990) for a survey.

The field of inductive learning here is to induce general concept descriptions of sounds from a set of examples. A further aim is to allow the computer to use automatically induced concept descriptions in order to identify unknown sounds or possibly suggest missing attributive or incomplete sound description. Our main reason for inducing rules about sounds is that the computer can then aid the user to explore among possible alternatives during the design a certain sound. Here the user would be able to ask the system to 'play something that sounds similar to a bell or even play a kind of dull sound', for example. In these cases the system will consult induced rules in order to work out which attributes are relevant for synthesizing a bell-like sound or a sound with dull colourful attribute (Small et al. 1993).

An example rule, when looking for a description for, say sound(c), on the basis of some examples, could be as follows:

\[
\text{sound(c) = \{[\text{vibrato = fast}, [\text{openness = high}]\}}
\]

The interpretation of the above rule is as follows:

A sound is sound(c) if:

- it has fast vibrato and
- high openness.

No matter how many attributes sound(c) had in the training set, according to the above rule, the most relevant attribute this sound are vibrato = normal and openness = high. Most relevant here means what is most important for distinguishing sound(c) form other sounds of the input training set. In this case, if the system is asked to synthesize a sound with fast vibrato and high openness,
An example functioning

Let us study an example functioning of the architecture explained above. Assume that the following information can be used in order to assemble the scheme of Figure 6.

**Knowledge base module:**

<table>
<thead>
<tr>
<th>slot: sound_event</th>
<th>value: {rate, fast}</th>
</tr>
</thead>
<tbody>
<tr>
<td>slot: sound_event</td>
<td>value: {with, default}</td>
</tr>
<tr>
<td>slot: sound_event</td>
<td>value: {{F0}, low}</td>
</tr>
<tr>
<td>slot: sound_event</td>
<td>value: {opposition, high}</td>
</tr>
<tr>
<td>slot: sound_event</td>
<td>value: {{F2}, 2677}</td>
</tr>
<tr>
<td>slot: sound_event</td>
<td>value: {{bw(3), 131.85}}</td>
</tr>
<tr>
<td>slot: attribute</td>
<td>value: {opposition, low}</td>
</tr>
<tr>
<td>slot: attribute</td>
<td>value: {{F1}, low}</td>
</tr>
<tr>
<td>slot: attribute</td>
<td>value: {opposition, low}</td>
</tr>
<tr>
<td>slot: attribute</td>
<td>value: {{F2}, low}</td>
</tr>
<tr>
<td>slot: attribute</td>
<td>value: {opposition, low}</td>
</tr>
<tr>
<td>slot: attribute</td>
<td>value: {{bw(2), 110.8}}</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>etc.</td>
<td></td>
</tr>
</tbody>
</table>

**Dictionary module:**

<table>
<thead>
<tr>
<th>dict: slot</th>
<th>value: low</th>
</tr>
</thead>
<tbody>
<tr>
<td>value: medium</td>
<td></td>
</tr>
<tr>
<td>value: high</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>dict: slot</th>
<th>value: low</th>
</tr>
</thead>
<tbody>
<tr>
<td>value: medium</td>
<td></td>
</tr>
<tr>
<td>value: high</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>dict: slot</th>
<th>value: low</th>
</tr>
</thead>
<tbody>
<tr>
<td>value: medium</td>
<td></td>
</tr>
<tr>
<td>value: high</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>dict: slot</th>
<th>value: low</th>
</tr>
</thead>
<tbody>
<tr>
<td>value: medium</td>
<td></td>
</tr>
<tr>
<td>value: high</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>dict: slot</th>
<th>value: low</th>
</tr>
</thead>
<tbody>
<tr>
<td>value: medium</td>
<td></td>
</tr>
<tr>
<td>value: high</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>dict: slot</th>
<th>value: low</th>
</tr>
</thead>
<tbody>
<tr>
<td>value: medium</td>
<td></td>
</tr>
<tr>
<td>value: high</td>
<td></td>
</tr>
</tbody>
</table>

... etc.

**Theory module:**

<table>
<thead>
<tr>
<th>instrument_theory</th>
<th>value: {{F0}, medium}</th>
</tr>
</thead>
<tbody>
<tr>
<td>get: value</td>
<td>{F0}, low, V</td>
</tr>
<tr>
<td>F0 is V + 2.</td>
<td></td>
</tr>
</tbody>
</table>

... etc.

Suppose that a training set has been input and the system has already induced some rules, such as:

- sound(a) = \{ opposition = low \}
- sound(b) = \{ F0 = medium \}
- sound(c) = \{ rate = fast, F0 = low \}

... etc.

Now, let us suppose two hypothetical queries and examine what ARTIST would do in order to compute them.
Example query 1:
Produce a sound with fast vibrato rate and low pitch.

**ARTIST functioning 1:**
Firstly, the system consults the induced rules in order to find out if it knows any sound whose most prominent features are rate = fast and f0 = low. In this case, there is a rule which tells that sound(1) satisfies that requirement. Thus, sound(1) will be produced.

Before assembling the schema, the system consults the dictionary in order to compute the slots whose values are represented by a word (e.g., f0 = low actually means 20 Hz).

Example query 2:
Produce a sound with medium pitch and high openness.

**ARTIST functioning 2:**
In this case the system has no matching induced rules. Thus, this sound will be created from scratch. The system consults the dictionary in order to compute the value of f0 = medium and S = high, and automatically completes the missing slot data with default values. Note that instead of a value for f0 = medium, the dictionary points to a role. In this case, the system consults the theory modules in order to calculate it. The theory says that this value corresponds to the double value of f0 = low. Therefore, f0 = medium here means 400 Hz. The sound is then produced, the user is asked to name it, and a novel cluster of slot values is automatically created in the knowledge base for representing it.

**Conclusion and further work:**

In this paper we introduced the fundamentals of ARTIST: a tool for the design of intelligent assistants for sound synthesis.

ARTIST is provided with some degree of automated reasoning which supports the laborious and tedious task of writing down number sequences for generating a single sound on a computer. A 'synthesizer' implemented by means of ARTIST is provided with a certain knowledge about sound synthesis and it is able to infer the necessary parameters values for a sound from a quasi-natural language sound description.

Although the user has to specify the information of the knowledge base (i.e., the synthesis algorithm) and the vocabulary for sound description beforehand, this does not necessarily need to be exhaustive. The system is designed to begin with a minimum amount of information about certain sounds and attributes, but it is able to automatically expand the scope of its knowledge by acquiring new information through user interaction.

At the moment we are developing a higher level interface for the user specified modules. We wish to enable the user to specify these modules by means of natural language-like statements, instead of Prolog. We also plan to develop a visual interface for the specification of such attributes, such as envelopes, for example.

ARTIST is being tested using synthesis by physical modelling techniques (Roads, 1993). It seems that this technique matches many of the concepts developed in this paper, such as the representation of sound attributes and the manipulation of synthesis parameters. We are aware that ARTIST is still in its infancy. For the moment we regard it as a suggestive and plausible starting point only.

**References**


Bratka, I. (1990), Prolog programming for Artificial Intelligence, Addison-Wesley Publishers.


Carpenter, R. H. S. (1990), Neurophysiology, Psychological Principles and Disorders of Medicine Series, Edward Arnold.


Shawson, W. (1985), *Sound Color, University of California Press*


Representing Musicians' Actions for Simulating Improvisation in Jazz

Geber Ramalho
LA FORIA-IBP-CNRS
Université Paris VI
4, Place Jussieu
75252 Paris Cedex 05 - FRANCE
Tel. (33-1) 44.27.37.27
Fax. (33-1) 44.27.70.00
e-mail: ramalho@laforia.ibp.fr

Abstract

This paper considers the problem of simulating Jazz improvisation and accompaniments. Unlike most current approaches, we try to model the musicians' behavior by taking into account their experience and how they use it with respect to the evolving contexts of live performance. To represent this experience we introduce the notion of Musical Memory, which exploits the principles of Case-Based Reasoning (Schank & Abelson 1977). To produce live music using this Musical Memory we propose a problem solving method based on the notion of PACTs (Potential Actions) (Ramalho & Ganascia 1994b). These PACTs are a generic framework for representing the musical actions that are activated according to the context and then combined in order to produce notes.

1 - Introduction

This paper considers the problem of simulating the behavior of a bass player in the context of Jazz live performance. We have chosen to work on Jazz improvisation and accompaniment because of their spontaneity, in contrast to the formal aesthetic of contemporary classical music composition. From an AI point of view, modeling Jazz performance raises interesting problems since performance requires both theoretical knowledge and great skill. In addition, Jazz musicians are encouraged to develop their musical abilities by listening and practicing rather than studying in conservatories (Baker 1980).

In Section 2 we present briefly the problems of modeling musical creativity in Jazz performance. We show the relevance of taking into account the fact that musicians integrate rules and memories dynamically according to the context. In Section 3 we introduce the notion of PACTs, the basic element of our model. In Section 4, we give a general description of our model and show particularly how the composition module integrates the two above-mentioned notions to create music. In the last section we discuss our current work and directions for further developments.

2 - Modeling Musical Creativity

2.1 - The Problem and the Current Approaches

The tasks of improvisation and accompaniment consist in playing notes (melodies and/or chords) according to guidelines laid down in a given chord chart (sequence of chords underlying the song). Musicians cannot satisfy all the local choices they make (typically at note-level) even if they have consciously applied some strategies in the performance. This is the greatest problem of modeling the knowledge used to fill the large gap referred to above (Ramalho & Pachet 1994). To face this problem, the first approach is to make random-oriented choices from a library of musical patterns weighted according to their frequency of use (Auci & Domino 1992). The second approach focuses on very detailed descriptions so as to obtain a complete explanation of musical choices in terms of rules or grammars (Steedman 1984). Regardless of its musical results, the random-based approach