Striking the right note with ARTIST: an AI-based synthesiser

Eduardo Reck Miranda
SONY CSL - Paris
6 rue Amyot
75005 Paris – France
miranda@csl.sony.fr

Abstract

This chapter introduces the fundamentals of ARTIST (an acronym for Artificial Intelligence-aided Synthesis Tool). ARTIST is a prototype system for sound synthesis that allows composition of sounds thought of in terms of qualitative descriptions (e.g. words in English) and intuitive operations rather than low level computer programming. My 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) a means to provide artificial intelligence (AI) for synthesisers. This chapter discusses my approach to the problem using a compilation of a few well known design techniques of expert systems used in AI research. ARTIST is a prototype system which embodies the results of my investigation so far.

1. 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.

In search of a more effective use of new technology, composers have come more ambitious, but the complexity of the task of sound composition has also increased. The scale and nature of the compositional task changes, technically and aesthetically. Theoretically a computer can be programmed to generate any sound that one can imagine. But, on the other hand, this can get composers into trouble. Quoting Barriere (1989, p. 116), "it is too easy to fail to take various consequences into account, to get technology side-tracked by a tool whose fascinating 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 (as in synthesis systems such as Csound, for example), is not particularly stimulating. I am convinced that higher processes of inventive creativity and musical abstraction are often prejudiced in such a situation. In this case, I think that the computer is being used as a kind of word processor combined with a Pianola, and not as a creative tool. I have come to believe that this can be improved by means of an appropriate coupling between human imagination and artificial intelligence (AI) (Miranda 2000).

In this chapter, I introduce the fundamentals of ARTIST (an acronym for Artificial Intelligence-aided Synthesis Tool). ARTIST is a system for sound synthesis that allows 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., 1993; Miranda, 1994; Miranda, 1998). By an intelligent assistant I mean a system which works co-operatively 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 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:

- The ability to operate the system by means of an intuitive vocabulary instead of using sound synthesis numerical values.

- 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.

- The encouragement of the use of the computer as a collaborator in the process of exploring ideas.

- 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.

- The ability to create contexts which augment the chances of something unexpected and interesting happening, such as an unimagined sound from an ill-defined requirement.

Apart from graphic workstations (such as the UPIC system (Xenakis, 1992; Marino et al., 1993; Lohner, 1986)) 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, Slawson (1985) has proposed - although not yet implemented on a machine - a kind of vocabulary for sound composition based on his theory of sound colour which, I believe, he derived from Helmholtz's theory of vowel qualities of tones (Helmholtz, 1885). Lerdahl (1987) too has done some sketches towards a hierarchical perceptually-orientated description of timbres. 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. CIMS (Schmidt, 1987) and Elthar (Garton,

1989)). More recently, Ethington and Punch (1994) proposed a software called SeaWave. SeaWave is an additive synthesiser (Dodge and Jerse, 1985; Miranda 1998) in which sounds can be produced by means of a vocabulary of descriptive terms. Although of a limited scope, SeaWave proffers an excellent insight and it seems to work well. Vertegal and Bonis (1994) have also been working towards a cognitive-orientated interface for synthesisers.

The following section introduces the signal processing of the synthesis technique used to illustrate the system. Next, I indicate some methods for describing sounds by means of their attributes and suggest a technique for mapping these attributes onto the parameters of a synthesiser. Then I demonstrate how this technique works and present some examples. Here, I also contemplate the functioning and the usefulness of machine learning (ML) in ARTIST. Finally, I introduce the architecture of the system and illustrate its functioning through examples. This chapter ends with some final remarks and ongoing work.

2. The signal processing level of the example study

For the examples in this chapter, ARTIST is used in the context of a subtractive synthesiser that produces human voice-like sounds. It is worth mentioning that to produce a perfect simulation of the human vocal tract is beyond the scope of my research at this stage. Thus, rather than making a description of the fundamental aspects of the phenomenon by means of a set of physical modelling equations (e.g. (Woodhouse, 1992; Keefe, 1992)), I opted to observe it by means of a more traditional formant modelling technique which uses subtractive synthesis (Flanagan, 1984; Klatt, 1990; Sundberg, 1991; Miranda, 1998). I believe that this level of description is sufficient at this moment. The signal processing diagram of the example study synthesiser is shown in Figure 1.

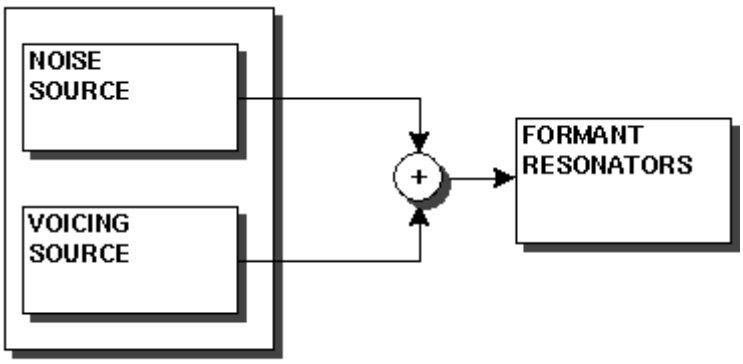

Figure 1. *The example study synthesiser.*

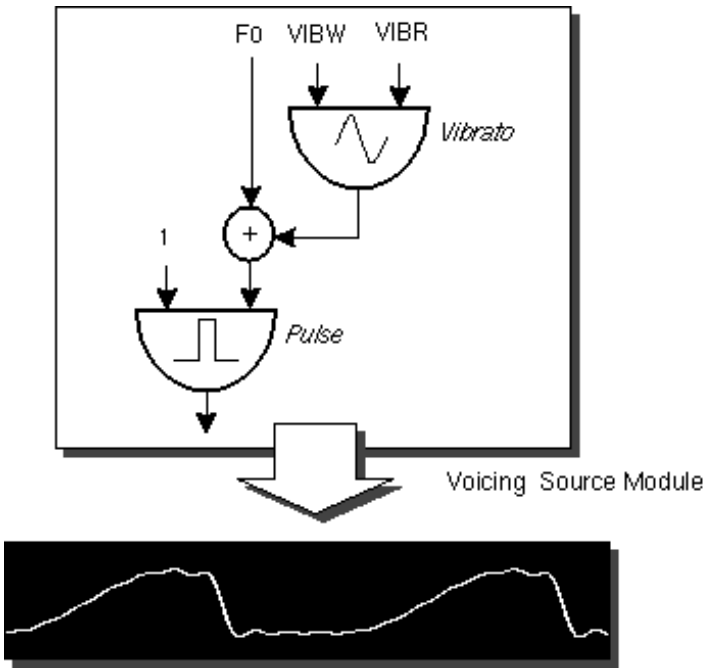

Figure 2. *The Voicing Source module is composed of two SPUs.*

Each block of the diagram is composed of several signal processing units (SPU). A composition of SPUs form sub-blocks within a block. Sub-blocks in turn may constitute sub-sub-blocks, and so forth. The *Voicing Source* module, for example, is composed of two SPUs: a *Vibrato* source and a *Pulse generator* (Figure 2).

Each SPU needs parameter values to function. In other words, in order to produce a certain sound, the synthesiser needs a stream of values to feed the SPUs.

3. Describing sounds by means of their attributes

There have been several studies defining a framework to systematically describe sounds by means of their attributes ((Schaeffer, 1966; von Bismark, 1971; 1974a; Cogan, 1984; Giomi and Ligabue, 1992; Carpenter, 1990; Terhardt, 1974) to cite but a few). They are derived mainly from work in the fields of both psychoacoustics and musical analysis. I classify these studies in two approaches: on the one hand, the *device-orientated* approach and, on the other hand, the *perceptually-orientated* approach. As it is not my aim to survey all these, I have selected one example of a device-orientated approach for illustration.

3.1. The source-filter model: a device-orientated approach

The source-filter model propounds that the characteristic of a sound is determined by the shape of its spectral envelope. The shape of the spectral envelope of vocal sounds 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 spectral envelope of formant frequencies is thought of as the result of a complex filter through which a source sound passes.

One 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*, can be specified in this space (after Slawson, 1985; 1987). 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).

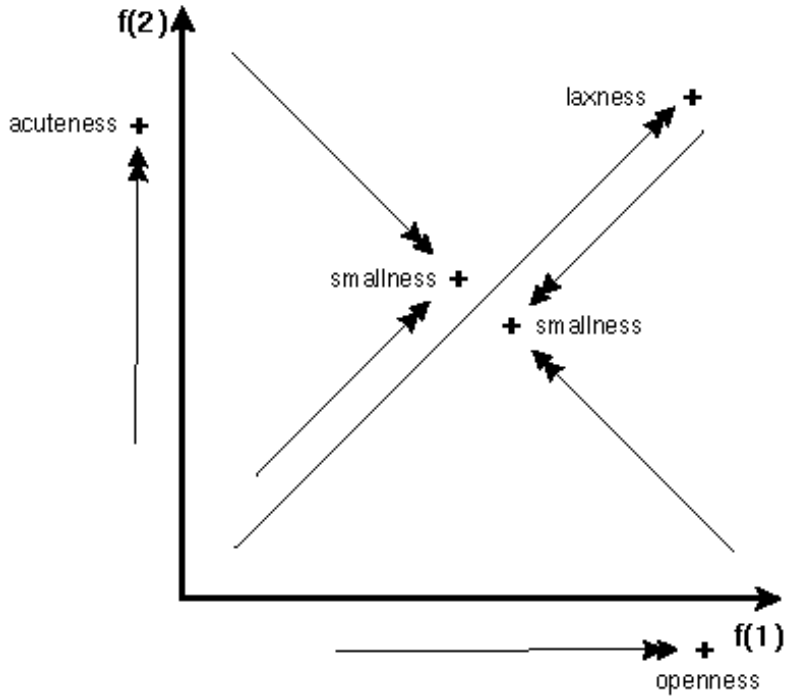

Figure 3. *Two-dimensional sound space.*

4. The notion of Abstract Sound Schema (ASS)

The Abstract Sound Schema (ASS) is the representation scheme devised to describe a sound in terms of its perceptual *components* and the *relations* between them. The ASS scheme is constituted of: *nodes, slots,* and *links.* Nodes and slots are the components,

and the links correspond to the relations between them. The links are labelled. The role of the ASS is twofold: it embodies a multi-levelled representation of the signal processing architecture of a synthesiser and provides an abstraction to represent sounds.

The ASS is, in fact, a tree-like abstract data structure whose ultimate nodes (the leaves) are slots. Each slot has a name and accommodates a sound synthesis datum. Slots are grouped bottom upwards into higher level nodes, which in turn are grouped into higher level nodes, and so forth, up to the top node (Figure 4).

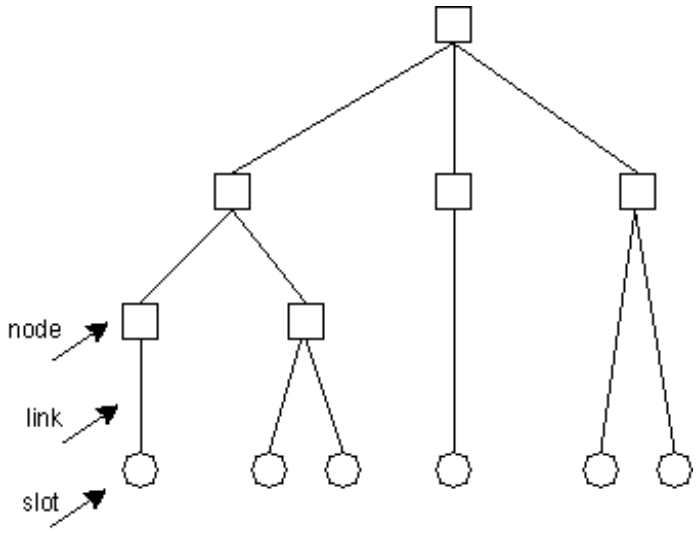

Figure 4. *The ASS representation scheme.*

The ASS enables the organisation of the knowledge of sounds, based upon the signal processing model that produces them. A sound event is represented here in terms of the various perceptual features which contribute to its identity. These features must however be tied to the signal processing model in some way. It is assumed that each descriptive attribute is caused by a certain component, or group of components of the synthesiser (e.g. blocks, sub-blocks and SPUs).

4.1. Implementing an abstract sound event

I have already demonstrated that the synthesiser is composed of several connected building blocks (Voicing Source, Noise Source, etc.), one of each is responsible for a certain sound attribute. Now I will 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, to the root of the tree (the *sound_event*).

Figure 5 portrays a partial definition of a *sound_event* for the synthesiser shown in Figure 1. Although not shown in Figure 5, the links among the components of the *sound_event* are labelled *has_component*; these links represent the offspring relation among nodes.

The *sound_event* definition shown in Figure 5 can be implemented in Prolog (Bratko, 1990) 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'*. The 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_source ).
has_component( voicing_source,  pulse_generator ).
has_component( formant_resonators,  first_formant ).
has_component( formant_resonators,  second_formant ).
%%%
%%% layer 3
%%%
has_component( vibrato_source,  vibr ). % vibrato rate
has_component( vibrato_source,  vibw ). % vibrato width
has_component( pulse_generator,  f0 ). % fundamental frequency
has_component( first_formant,  f1 ). % 1st formant frequency
has_component( first_formant,  bw1 ). % 1st formant bandwidth
has_component( second_formant,  f2 ). % 2nd formant frequency
has_component( second_formant,  bw2 ). % 2nd formant bandwidth
```

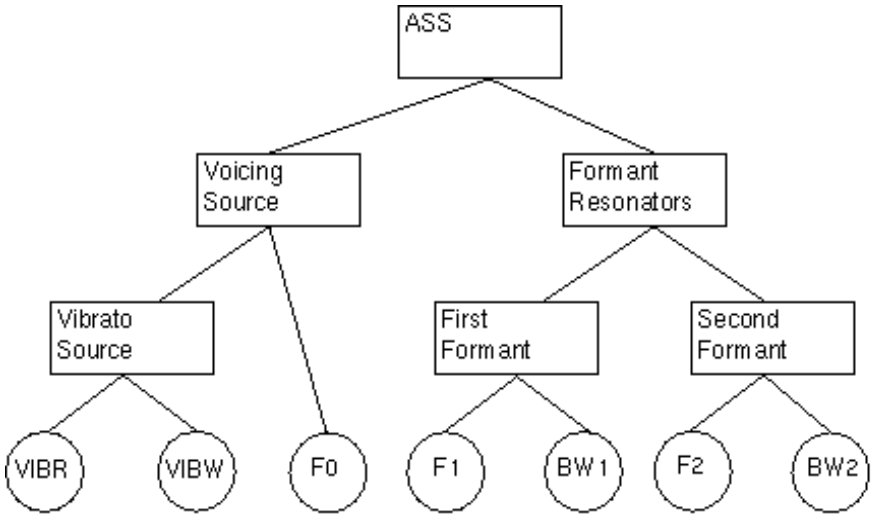

Figure 5. *Abstract sound event representation.*

All the slots of the ASS must be filled in order to completely instantiate a sound. In the context of ARTIST, a completely instantiated sound is an *assemblage* and for each different sound there is a particular assemblage. Thinking of this synthesiser 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.

4.2. Sound hierarchy and the inheritance mechanism

So far, I have defined a general abstract scheme for representing a sound. Then I defined and implemented an abstract *sound_event* by means of this scheme. I 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 *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. This example knowledge base contains information about three sounds, namely *back vowel, front vowel*, and *vowel /a/.*

```
% %% back vowel
%%%
slot( vowel(back), [ vibr, 5.2 ] ).
slot( vowel(back), [ vibw, 0.06 ] ).
slot( vowel(back), [ f0, 155.56 ] ).
slot( vowel(back), [ f1, 622.25 ] ).
slot( vowel(back), [ f2, 1244.5 ] ).
slot( vowel(back), [ bw1, 74.65 ] ).
slot( vowel(back), [ bw2, 56 ] ).
%%%
%%% front vowel
%%%
slot( vowel(front), [ vibr, 5.5 ] ) .
slot( vowel(front), [ vibw, 0.06 ] ).
slot( vowel(front), [ f0, 138.59 ] ).
slot( vowel(front), [ f1, 559.37 ] ).
slot( vowel(front), [ f2, 1108.7 ] ).
slot( vowel(front), [ bw1, 78.3 ] ).
slot( vowel(front), [ bw2, 110.8 ] ).
%%%
%%% vowel /a/
%%%
slot( vowel(a), [ a_kind_of, vowel(back) ).
slot( vowel(a), [ f0, 103.83 ] ).
```

Note that the representation of the sound **vowel(a)** is different from the other two: it is incomplete (i.e. there are no slot values for the **vibrato_source** and for the **formant_resonators**). On the other hand, there is new information in it. The new information, called *a_kind_of,* is not a simple *sound_event* slot, as it might appear to be, but it is a link (Figure 6). This is a link which associates one collection of slots with other collection of slots.

The *a_kind_of* link allows for the hierarchical organisation of the knowledge base. The ability to represent the relationship between slot collections hierarchically is useful for inheritance relation. Inheritance is a relation by which an individual assumes the properties of its class and by which properties of a class are passed on to its subclass. Thus, when a slot collection for a sound is attached to another slot collection at a higher

level, the former inherits properties of the latter. The first fact of the third cluster of slots listed above states that a **vowel(a)** is *a_kind_of* **vowel(back)**. This is to say that slots not defined for **vowel(a)** will be filled with slot values taken from **vowel(back)**. In practice, the assemblage engine 'knows' that the missing slots in one level are inherited from a higher level.

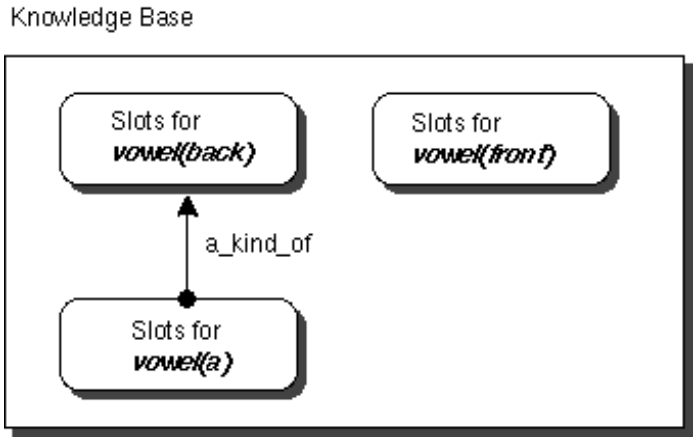

Figure 6. *This knowledge base has information about three sounds. Each sound is represented as a collection of slot values. Note that* ***vowel(a)*** *inherits slots from* ***vowel(back)****.*

4.3. The notion of partial assemblage

Note that the assemblage engine may also assemble single internal nodes of the ASS. In other words, besides the assemblage of the whole ASS there might be (partial) assemblages of only certain nodes. Let us consider again the example shown in Figure 5. It has a branch of filters which constitute the formant resonators. Taking as an example only the node **first_formant** , one could say that it needs only its affiliated slots (namely **f1** and **bw1**) 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 one can attach non-numerical attribute values (i.e. words in English) to partial assemblages too. For instance, one could refer to the node **first_formant** as **low_and_wide** if it has **f1 = 250 Hz** and **bw1 = 200 Hz.** This is also represented in the knowledge base as a cluster of slots. Example:

slot([first_formant, low_and_wide], [f1, 250]).
slot([first_formant, low_and_wide], [bw1, 200]).

Now, for each node of the schema one can define a set of possible non-numerical attribute values. Back to Figure 5, the slots **vibr**, and **vibw** constitute a node called **vibrato_source**

which in turn, with the node **pulse_generator,** forms the higher level node **voicing_source**. One could establish here that the possible attribute values for **vibrato_source** 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_source** is **none** if **vibr = 0 Hz**, and **vibw = 0 %**. The node **voicing_source** could be similarly defined: one could establish that **voicing_source** is **steady_low** if **vibrato = none** and **pulse generator = low_frequency**, for example.

Hypothetically considering only the left branch of the ASS portrayed in Figure 5, a sound, say **sound(example)**, could be described as *having steady low voicing source and none vibrato.* See example below:

slot([vibrato_source, none] , [vibr, 0]).
slot([vibrato_source, none] , [vibw, 0]).
slot([pulse_generator, low_frequency], [f0, 55]).
slot([voicing_source, steady_low], [vibrato_source, none]).
slot([voicing_source, steady_low], [pulse_generator, low_frequency]).
slot([sound(example), [voicing_source, steady_low]).

5. 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 are 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 target of inductive learning here is to induce general concept descriptions of sounds from a set of examples. A further aim is to allow ARTIST to use automatically induced concept descriptions in order to identify unknown sounds or possibly suggest missing attributes of an incomplete sound description. Our main reason for inducing rules about sounds is that ARTIST can then aid the user to explore among viable alternatives during the design of 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 synthesising a bell-like sound or a sound with dull colour attribute (Smaill et al., 1993).

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

sound(drill) = { [vibrato_source = fast], [openness = high] }

The interpretation of the above rule is as follows:

A sound is **sound(drill)** if:
it has **fast vibrato** and
high openness.

No matter how many attributes **sound(drill)** had in the training set, according to the above rule, the most relevant attributes for this sound are **vibrato_source = normal** and **openness = high**. *'Most relevant'* here means what is most important for distinguishing **sound(drill)** from other sounds of the input training set. In this case, if the system is asked to synthesise a sound with **fast vibrato** and **high openness**, then it will produce **sound(drill)**.

The target of supervised deductive learning in our system is to allow the computer to update its knowledge about attribute values throughout user interaction. I would like to draw your attention to the fact that the input requirement for producing a sound can contain either or both attribute values (e.g. **vibrato_source = none**) or slot values (e.g. **f0 = 55 Hz**). The aim of supervised deductive learning here is to allow ARTIST to infer whether or not input slot values (in a requirement) match with attribute values that ARTIST already knows. If there is no matching, then the system automatically adds this yet unknown information to the knowledge base and asks the user to give a name for this newly-deduced attribute value. As an example, suppose that ARTIST is aware of three values for the attribute **vibrato** (an attribute attached to the component **vibrato_source**):

vibrato = uniform if { **vibr = 5.2 Hz**, **vibw = 3 %** }
vibrato = too_slow if { **vibr = 3.6 Hz**, **vibw = 3 %** }
vibrato = none if { **vibr = 0 Hz**, **vibw = 0 %** }

If the user requires a sound with vibrato rate **vibr = 12 Hz**, then ARTIST will synthesise it and deduce that there is no attribute value for vibrato in the knowledge base whose **vibr** is equal **12 Hz**. In this case the system adds this new information to the knowledge base, works out the other slot values needed to create this new attribute value, and asks the user to name it. Let us say, for example, that the user wishes to call it **fast**. Eventually ARTIST will add the following information to its knowledge base:

vibrato = fast if { **vibr = 12 Hz**, **vibw = 3 %** }

6. The system architecture

User configuration is one of the desirable capabilities of this system. Therefore rather than providing a closed architecture which reflects both a particular synthesiser and a particular vocabulary for sound description, I have devised an architecture which provides open-ended modules (Figure 7). This system architecture provides a means to handle information about sound synthesis but it remains open-ended regarding the precise nature of the information.

6.1 Built-in modules: engines and services provided by the system

The role of the *Assemblage Engine* and the functioning of the *Knowledge Acquisition* module have already been introduced.

The *Knowledge Acquisition* module performs the two kinds of learning: *inductive learning* and *supervised deductive learning*, just discussed above. The training set for the inductive learning mechanism is given either by the user or it is automatically produced by the system by consulting its own knowledge base. The input for the supervised deductive learning mechanism is provided partly by the system and partly by the user.

The *User Interface* module provides a means to communicate with the system. Here the user can activate the *Assemblage Engine* in order to produce a sound, consult the status of the system (such as the content of the *Knowledge Base* module), and input any external information the system might need (such as the names for new sounds and attributes, and training sets).

6.2. Open-ended modules: user specified information

These modules define the domain of the system, that is, the sonic world governed by the system. Here the user implements the *Synthesis Algorithm(s)* and the *Knowledge Base* whose information is used to 'play' it. Default libraries of such modules are provided in case the user does not wish to start from scratch. However, as these modules are to be user-customised, it may not always be very useful to exchange highly customised libraries with other users.

Firstly, the user specifies the *Synthesis Algorithm.* This can be done by means of any suitable sound synthesis package, such as CLM or Cmusic (Miranda, 1998). Having specified the instrument(s), then the user implements the ASS. Secondly, the *Knowledge Base* module is specified. In this module the user creates clusters of slot values. As previously mentioned, each cluster corresponds to an instantiation of either a whole sound event or an internal node of the schema, i.e. a sound attribute. Here the user builds

a *dictionary* of terms used to describe the parameter values for slots. Each term of this vocabulary may mean either a numerical synthesis parameter or a pointer to a formula for calculating it. Also in the *Knowledge Base*, the user specifies a *theory* for the instrument. A theory is a set of formulae for calculating slot values. These formulas can calculate values either based on other slot values or by the random choice of a value within a certain interval.

As the system starts with a certain body of knowledge which will be expanded throughout user interaction, these specifications do not need to be exhaustive.

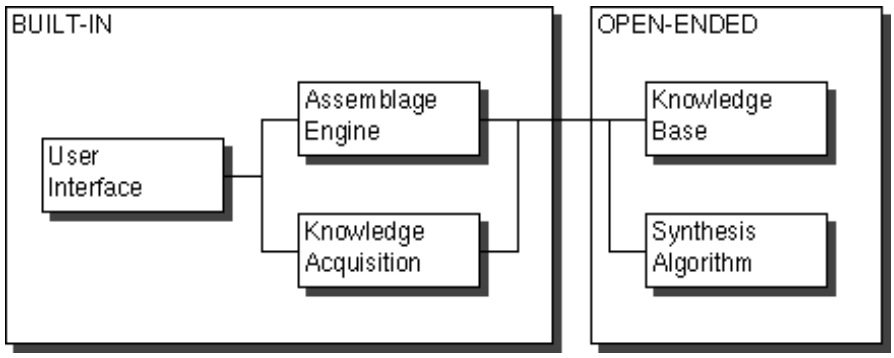

Figure 7. *The system architecture.*

7. An example functioning

Let us study an example functioning of ARTIST. Assume that a *Knowledge Base* accommodates the information to assemble the ASS in Figure 5 as follows:

Cluster of slot values:

slot(sound_event(sound(cheerful)), [vibr, fast]).
slot(sound_event(sound(cheerful)), [vibw, default]).
slot(sound_event(sound(cheerful)), [f0, low]).
slot(sound_event(sound(cheerful)), [openness, high]).
slot(sound_event(sound(cheerful)), [acuteness, low]).
...
slot(attribute([openness, low]), [f1, low]).
slot(attribute([openness, low]), [bw1, 74.65]).
...
slot(attribute([openness, high]), [f1, high]).
slot(attribute([openness, high]), [bw1, 78.3]).
...

```
slot( attribute( [ acuteness, low ] ), [ f2, low ] ).
slot( attribute( [ acuteness, low ] ), [ bw2, 110.8 ] ).
...
...
etc.
```

Dictionary:

```
dict( slot( f1), [ value( low, 290 ) ,
                   value( medium, 400 ),
                   value( high,  650 ) ] ).

dict( slot( f2 ), [value( low, 1028 ),
                   value( medium, 1700 ),
                   value( high, 1870 ) ] ).

dict( slot( f0 ), [value( low, 220 ),
                   value( medium, rule( f0, medium ) ),
                   value( high, rule( f0, high ) ) ] ).

...
etc.
```

Theory:

```
instrument_theory( rule( f0, medium ), F0 ):-
        get_value( f0, low, V ),
        F0 is V * 2.
...
etc.
```

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

```
sound(dull) =          { [openness = low ] }
sound(bang) =          { [ f0 = medium ] }
sound(cheerful) =      { [ rate = fast ], [ f0 = low ] }

...
etc
```

Now, let us take 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 of any sound whose most prominent features are **vibr = fast** and **f0 = low**. In this case, there is a rule which states that **sound(cheerful)** matches this requirement. Thus, **sound(cheerful)** 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 **220 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 values of **f0 = medium** and **f1 = 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 rule. In this case, the system consults the theory module in order to calculate it. The theory says that this value corresponds to the double value of **f0 = low**. Therefore, **f0 = medium** here means **440 Hz**. The sound is then produced, the user is asked to name it, and a new cluster of slot values is automatically created in the knowledge base to represent it.*

8. Conclusion and further work

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. The system holds knowledge about sound synthesis and it is able

to infer the necessary parameter 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(s) and the vocabulary for sound description) beforehand, this does not necessarily need to be exhaustive. The system begins with a minimum amount of information about certain sounds and attributes, but it is automatically able to expand the scope of its knowledge by acquiring new information through user interaction.

At present I am developing a higher level interface for the user specified modules. I wish to enable the user to specify these modules by means of natural language-like statements, instead of Prolog programming. I also plan to devise a graphic interface for the specification of some attributes, such as envelopes, for example.

ARTIST is currently being tested using physical modelling synthesis techniques (Roads, 1993; Miranda, 1998), which match many of the concepts I have developed to date.

ARTIST is still in its infancy but it is a plausible starting point towards intelligent synthesis systems.

References

Barriere, J-B.,"Computer music as cognitive approach: Simulation, timbre and formal processes", *Contemporary Music Review*, vol. 4, p. 117-130, 1989.

Bratko, I., *Prolog programming for Artificial Intelligence*, Addison-Wesley Publishers, 1989.

Carbonell, J., *Machine Learning: paradigms and methods*, The MIT Press, 1989.

Carpenter, R. H. S., *Neurophysiology*, Edward Arnold, 1990.

Cogan, R., *New Images of Musical Sound*, Harvard University Press, 1984.

Dodge, C., Jerse,T., *Computer Music*, Schirmer Books, 1985.

Dietterich, T., Michalski, R., "Inductive Learning of Structural Descriptions", *Artificial Intelligence*, vol. 16, 1981.

Ethington, J., Punch, D., "SeaWave: A system for Musical Timbre Description", *Computer Music Journal*, vol. 18, no. 1, 1994.

Flanagan, F., "Voices of Men and Machines", *Electronic Speech Synthesis*, (Bristow, G. - Editor), Granada, 1984.

Garton, B., "The Elthar Program", *Perspectives of New Music*, vol. 27, no. 1, 1989.

Giomi, F., Ligabue, M., Analisi Assistita al Calcolatore della Musica Contemporanea, Rapporto Interno C92-01 - CNUCE/CNR, Conservatorio di Musica L. Cherubini, 1992.

Helmholtz, H. L. F., *On the sensations of tone as a physiological basis for the theory of music*, Longmans, Green and Co, 1885.

Holtzman, S. R., A description of an automated digital sound synthesis instrument, DAI research report no. 59, Dept. of Artificial Intelligence, University of Edinburgh, 1978.

Klatt, D. H., "Software for a cascade/parallel formant synthesiser", *Journal of Acoustic*

Society of America, vol. 67, no. 3, 1980.
Keefe, D. H., "Physical Modeling of Wind Instruments", *Computer Music Journal*, vol. 16, no. 4, 1992.
Lerdhal, F., "Timbral Hierarchies", *Contemporary Music Review*, vol. 2, 1987.
Lohner, H., "The UPIC System: A User's Report", *Computer Music Journal,* vol. 10, no. 4, 1986.
Luger, G. F., Stubblefield, W. A., *Artificial Intelligence and the design of Expert Systems*, Benjamin/Cummings, 1989.
Marino, G., Serra, M-H., Raczinski, J-M., "The UPIC System: Origins and Innovations", *Perspectives of New Music*, vol. 31, no. 1, 1993.
Miranda, E. R., Smaill, A., Nelson, P., "A Symbolic Approach for the design of Intelligent Musical Synthesisers", Proceedings of the *X Reunion Nacional de Inteligencia Artificial in Mexico City*, Megabyte/Noriega Editores, 1993.
Miranda, E. R., Smaill, A., and Nelson, P., "A Knowledge-based approach for the design of Intelligent Musical Instruments", Proceedings of the *X Simposio Brasileiro de Inteligencia Artificial,* p. 181-196, 1993.
Miranda, E. R., "Modelagem do Aparelho Fonador e suas Aplica es na M sica", *Ac stica & Vibra es* (Journal of the Brazilian Acoustic Association), no. 12, 1993.
Miranda, E. R., "From Symbols to Sound: Artificial Intelligence Investigation of Sound Synthesis", *Contemporary Music Review*, vol. 10, 1994.
Miranda, E. R., "The Role of Artificial Intelligence in Computer-aided Sound Composition", *Journal of Electroacoustic Music*, vol. 8, 1994.
Miranda, E. R., *Computer Sound Synthesis for the Electronic Musician*, Focal Press, 1998.
Miranda, E. R., *Readings in Music and Artificial Intelligence*, Harwood Academic Publishers, 2000.
Pennycook, B. W., "Computer Music Interfaces: A Survey", *Computing Surveys,* vol. 17, no. 2, 1985.
Roads, C., "Initiation la synthese par modeles physiques", *La Synthese Sonore*, Les cahiers de l'Ircam, no. 2, 1993.
Quinlan, J. R., "Semi-autonomous Acquisition of Pattern-based Knowledge", *Introductory Reading in Expert Systems,* (Michie, D. - Editor), Gordon & Breach, 1982.
Schaeffer, P., *Trait des objets musicaux*, Ed. du Seuil, 1966.
Schmidt, B. L., "Natural Language Interface and their application to Music Systems", *Proceedings of the 5th Audio Engineering Society International Conference*, p. 198-206, 1987
Slawson, W., *Sound Color*, University of California Press, 1985.
Slawson, W., "Sound-color Dynamics", *Perspectives of New Music*, vol. 25, no. 1 and 2, 1987.
Smaill, A., Wiggins, G. A., Miranda, E. R., "Music Representation - between the Musician and the Computer", *Music Education: An Artificial Intelligence Approach*, (Smith, M. et al. - Editors), Workshops in Computing Series, Springer- Verlag, 1994.
Spender, N., "Psychology of music (I-III)", *The New Grove's Dictionary of Music and*

Musicians, (Sadie, S. - Editor), vol. 15, 1980.
Sundberg, J., "Synthesising Singing", *Representation of Musical Signals*, (De Poli et al. - Editors), The MIT Press, 1991.
Terhardt, B., "On the Perception of Periodic Sound Fluctuation (Roughness)", *Acustica*, vol. 30, 1974.
Vertegal, R., Bonis, E., "ISEE: An Intuitive Sound Editing Environment", *Computer Music Journal,* vol. 18, no. 2, 1994.
von Bismark, G., "Timbre of Steady Sounds: Scaling of Sharpness", *Proceedings of the 7th Internacional Congress on Acoustics*, vol. 3, p. 637-640, 1971.
von Bismark, G., "Timbre of Steady Sounds: A Factorial Investigation of its Verbal Attributes", *Ac stica*, vol. 30, 1974.
von Bismark, G., "Sharpness as an Attribute of the Timbre of Steady Sounds", *Acustica*, vol. 30, 1974.
Winston, P., *Artificial Intelligence*, (2nd ed.), Addison-Wesley, 1984.
Woodhouse, J., "Physical Modeling of Bowed Strings", *Computer Music Journal,* vol. 16, no. 4, 1992.
Xenakis, I., "Musiques Formelles", *La Revue Musicalle,* double issue no. 253 and 254, 1963.
Xenakis, I., *Formalized Music: Thought and Mathematics in Music*, Pendragon Press, 1992.