A Visual Programming Environment for Constraint based Musical Composition

CAMILO RUEDA
Ingeniería de Sistemas, Universidad Javeriana de Cali
Cali, A.A 26239 Colombia

ABSTRACT

We describe a visual programming environment called *Niobé* in which the composer can easily construct and operate on *template* musical structures defined by a set of relations. *Niobé* provides primitives allowing the composer to graphically program arbitrary constraints on some musical domain (harmonic, rhythmic, etc) and also a mechanism for computing one or several instances of specific musical structures whose elements satisfy the given constraints. The composer can in this way construct a potentially large data base of different musical structures, each having the same precisely defined properties. Resulting structures can be visualized and hand modified in different supplied music notation editors. *Niobé* has been carefully optimized for computing sequences of harmonic or rhythmic elements. It uses the graphical interface of *PatchWork*, the visual music composition language developped at IRCAM in Paris. *Niobé* is implemented in Common Lisp-CLOS and is extensible.

I. Introduction

We present *Niobé* , a graphical environment for rule based music composition. *Niobé* is well suited to the incremental construction of musical structures obeying precisely defined properties. Structures are built in *Niobé* either by setting control parameters of built-in relations or by imposing a set of new user defined constraints. This way of regarding computation falls within the realm of what is called *Constraint Satisfaction* . *Niobé* is logically divided in two components: A computational engin adapted for solving constraint satisfaction problems in the musical domain and a graphical programming interface. We show how the composer can take advantage of the interaction between these two components to interactively construct and refine harmonic or rhythmic structures in an incremental process. That is, instances of roughly specified structures computed by the system can be visually represented in suitable music notation editors. The composer might then see the need to impose further melodic or harmonic constraints which are used by *niobé* to compute new refined instances. *Niobé* runs on top of *PatchWork* (Laurson, Duthen & Rueda, 1992), a graphical music composition language adapted to the representation of preocesses as a sequence of functional transformations. We describe how to exploit in *niobé* the two alternative ways of regarding programming, constraint-oriented and functional, to effectively compute complex musical structures. *Niobé* is entirely programmed in Common LISP-CLOS (Steele, 1990). Finally, we present some examples of the usage of *Niobé* in real musical applications.

II. Background

Several music composition languages such as Pla (Schottstaedt, 1983) have been defined in the past. These languages consider the compositional activity as divided in two basically independent processes, a score defining scheduling of different types of events and a functional part where structures are built by composition of transformations. In the latter, the composer is responsible for programming the appropriate transformations leading to the desired structure. This activity requires in general good programming skills. To alleviate somewhat the burden of the interaction of the composer with the computer, these systems provide a library of predefined transformations the composer can use as basic building blocks. In Common Music (taube, 1991), this idea is complemented with a set of built-in pattern structures that the user regards as templates for instanciating the desired structure. Though very powerful, this schemes leaves the problem of having to decide appropriate ways of combining patterns and functions. In a broad sense, all of these composition languages reflect the functional paradigm of the underlying implementation language in which it is necessary to describe a desired result by making explicit the way of achieving it. There are situations, however, in which it is not at all obvious to find appropriate algorithms to compute structures that nevertheless admit simple descriptions. Recently, the composition environment PatchWork (Laurson, Duthen & Rueda, 1992) has been proposed as a way of easing the programming task by redefining it as a visual activity.A program in this language is a graphical patch where boxes represent computations and links between boxes define functional composition of transformations (see figure 1). As in the above mentioned languages, a library of predefined boxes provide building blocks for programming. Although we believe that giving composers an entirely visual programming environment is a step in the right direction, this might not reduce significantly the programming effort in situations where the actual musical processes are not conceived algorithmically (i.e as a sequence of transformations) at the beginning but rather as evolving sets of precisely defined relations. What is needed in this case is the capability to propose *descriptions* of structures in a declarative manner, leaving to the system the task of actually computing them.

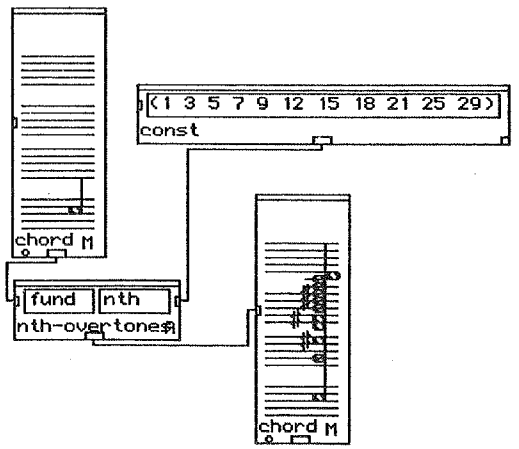


Figure 1. A PatchWork patch computing a chord from a set of harmonics of a base note

Recently, a system based on this idea, called *Echidna* (Ovans, 1990), has been proposed to support counterpoint generation.*Echidna* is not in itself a music composition language but rather a general purpose declarative programming tool based on the principle of constraint satisfaction. The rules of counterpoint are first stated as a set of constraints on (finite) values

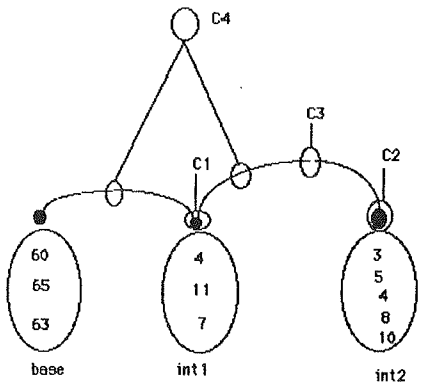
representing pitches and then *Echidna* is used to choose subsets of these satisfying the given harmonic and melodic constraints.

The two approaches, functional and declarative, are complementary in the sense that for certain type of material one or the other prooves to be more convenient. Our research concern has been to unify both ways of conceiving the process of constructing musical structures. Thew underlying notion sustaining this aim is that of a *partially instanciated musical structure* (PIMS). Loosely speaking a PIMS (Assayag & Rueda, 1993) is a generalization of a structure in the functional sense whose elements are sets and augmented with a collection of relations or constraints. In what follows we precise this notion and describe its implementation in *Niobé* .

III. The theory of PIMS

A PIMS is the basic building block for generating musical material. It is defined as the structure <D, R, C> where D is a finite collection of finite sets (called *Domains*), R is a binary relation on D and C is a set of constraints (relations) on D. Basically, R is a *structuring* relation on D whereas elements in C are *filtering* relations on elements od D. Constraints in C define subsets of the cartesian product of the sets in some subset of D. Any element in the cartesian product defined by a constraint c of C is said to *satisfy* c. If all constraints in C define non empty sets the PIMS is said to be (locally) *consistent* . A PIMS in which D contains only singleton sets is called a PIMS *instacnce* . A PIMS *exemplary* is a consistent PIMS instance.

A Partial order can be defined on PIMS as follows: Let P=<D1, R, C> and Q=<D2, R, C> be PIMS. If each cartesian product on D1 is contained in some product on D2 then P≤Q. Given a PIMS P, the *PIMS instanciation problem* consists in finding a PIMS exemplary E such that E≤P. Seen from this perspective a PIMS is a structure scheme representing the set of its exemplary structures. The graph in the figure below represents a PIMS for the set of all three note chords starting at any one of the notes in the set *base* (in MIDI), having consecutive intervals taken from the sets *int1* , *int2* (in semitones), not containing octaves and positioned within the register from 60 to 79 in MIDI.



C1: int1 ≠ 12; C2: int2 ≠ 12
C3: int1 + int2 ≠ 12; C4: base + int1 +int2 ≤ 79
R: base-> int1-> int2; D: {base, int1, int2}
Figure 2. A PIMS

3.1 Structure instanciation by arc consistency.

Building musical material can thus very genarally be seen as a two step process, first constructing a suitable PIMS and then solving the PIMS instanciation problem on it. For the latter we use in *Niobé* arc consistency techniques. These are well known algorithms in the constraint satisfaction field aiming at improving the efficiency of finding a solution by trying to reduce the given domains. Domains are reduced by insuring that constraints are locally consistent. A constraint can be represented by a graph (see figure 2) having domains as nodes and constraints as arcs linking those domains it constrains. An arc in this graph is said to be consistent if for any element in any of the linked domains there can always be found elements in the other linked domains such that all taken together satisfy the constraint. Values in the linked domains not obeying this property can be eleminated, thus reducing domain sizes. Algorithms for achieving arc consistency are described in (Mackworth, 1977). Recently, a more efficient arc consistency procedure called AC-5 has been proposed in (Deville & Van Hentenryck, 1991). AC-5 runs in time proportional to the square of the biggest domain size, but can easily be specialized to a linear time algorithm for useful categories of constraints. These are refered to in (Deville & Van Hentenryck, 1991) as functional and monotonic constraints. Briefly stated, these are constraints such that suitable representatives in each domain suffice to test the validity of the constraint for the whole domain. Constraint C4 in figure 2 above is of this type. AC-5 forms the core of the structure instanciation scheme in *Niobé* . Additional optimizations are considered by defining hierarchies on the PIMS domains reflecting frequently encountered musical constraints. One example is the problem of instanciating chord sequences structures where constraints imposing particular melodic movements on the upper and lower voices are frequently stated.Chord structures (PIMS) are thus supplied with an additional (hidden) domain comprising possible sums of consecutive intervals in the chord. Melodic constraints can thus be (automatically) redefined to act on the base note and sum-of-intervals domains avoiding the need to look into each particular interval composition of a chord. Domains in *Niobé* are thus *trees* allowing constraint impositions at any level. Figure 3 below shows an example this tree for a chord intervals domain. A similar structuring technique has been proposed in *Echidna* (Sidebotton & Havens, 1991) for representing constraints on real numbers.

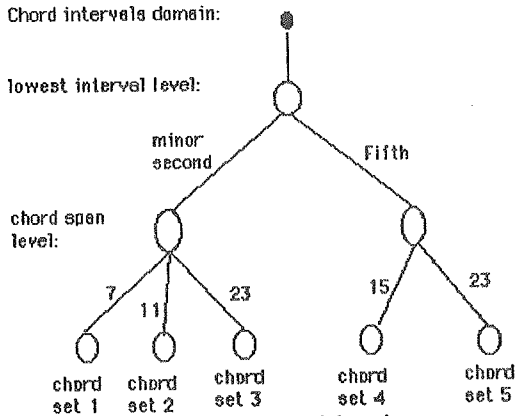


Figure 3. A structured domain

Musical constraints within a PIMS are in general conceived by the composer as having different degrees of importance. We describe next a mechanism implemented in *Niobé* for taking account of this fact.

3.2 Soft constraints in PIMS.

Constraints in a PIMS can be assigned a degree of importance. In *Niobé* this is simply a number between zero (useless constraint) and one (required constraint). A valuation function is defined on PIMS instances. The value of an instance P is equal to one minus the importance degree of the most important unsatisfied constraint c in P. *Niobé* computes highest valued instances of a PIMS by extending AC-5 with a process similar to the standard*alpha-beta* procedure used in several Artificial Intelligence applications such as game tree search. Formally, The PIMS instanciation problem is redefined as follows: Given a PIMS P, find a PIMS instance Q≤P such that, for any other PIMS instance R≤P, *Valuation* (R)≤*Valuation* (Q). A detailed account of this way of handling constraint preferences (or *soft* constraints) can be found in (Schiex, 1992). In *Niobé* , both the degree of importance of constraints and the minimum value requiered of a solution are user controlled parameters. We develop next the user interface of *Niobé* looking into the details of some examples.

3.3 Using *Niobé* .

Niobé is implemented in Common Lisp-CLOS. A PIMS, its domains and constraints are CLOS objects. A graphical interface in *PatchWork* is supplied for constructing and parameterizing these objects and for triggering the instanciation mechanism. PIMS instances can also be graphically interpreted, functionally transformed and displayed/edited in standard music notation by using suitable *PtachWork* editors. A *PatchWork* box (called *harmonic-constraints* in figure 4) representing *Niobé* defines entries for domains and constraints specifications. These entries consist of a set of parameters controlling*templates* of built-in constraints and domains. In the example of figure 4, domains are sets of chords and the PIMS to be computed is simply a sequence of chords obeying precise vertical (harmonic) and horizontal (melodic) constraints. The entry called *ambitus*

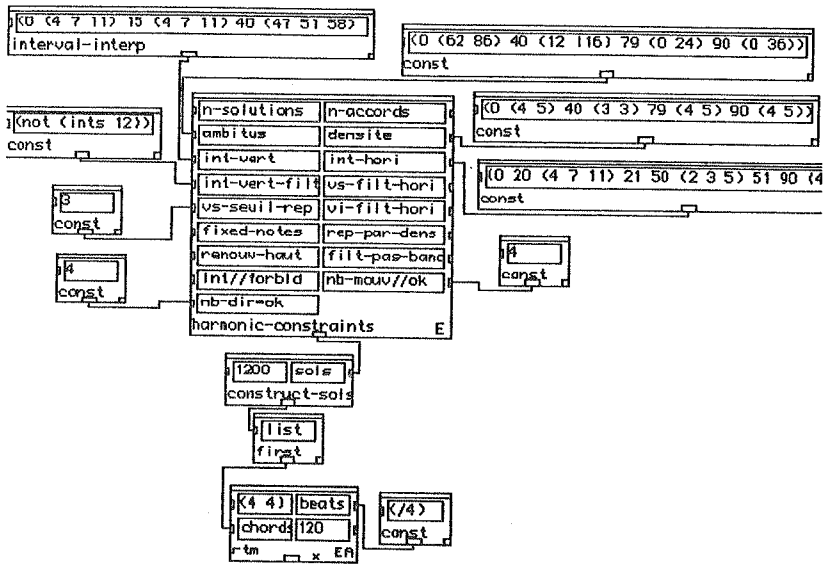


Figure 4. The graphical interface of *Niobé* .

is connected to a *const* box defininig the global vertical span of the sequence. It covers two octaves (62 to 86 in MIDI) in the first chord then goes to about 9 octaves (12 to 116) for the 40th chord and comes down to 3 octaves at the last chord (0 to 36). The span given for selected chord numbers represent points of a linear interpolation computing the global span for the rest of the chords. This thus gives the general form of the region where the chord sequence should fit. The box connected to the entry *int-vert* (vertical intervals) defines the domain of intervals (taken between consecutive notes) for the chords. For the first 40 chords only major third (4 in semitones), fifth (7 in semitones) and/or major seventh (11) are allowed. In the middle of the sequence very long intervals are demanded. The intended effect is to shift perception from harmonic to melodic. At the end of the sequence the original intervals return. Entry *int-vert-filt* defines any filtering relation on the interval contents of each chord. Here only the elimination of octaves is imposed (by the predicate *(not (ints 12))*).*Densite* is just the number of allowed notes in each chord. Here either 4 or 5 notes (as *Niobé* likes it) in each chord is established for all but the middle of the sequence where exactly 3 note chords are demanded, the reason being to precisely control melodic movement of each of the three voices in this part of the sequence. Melodic control is done at the *int-hori* (horizontal intervals) entry of the box. Here the same intervals of major third, fifth and major seventh (4, 7, 11) are imposed *horizontally* for the upper and lower voices in most of the sequence, except in the middle where intervals of a second, minor third and fourth (3, 5, 8) are actually requiered horizontally for each one of the three voices. Although not shown in this example, melodic movement can be further controlled by drawing break-point functions defining curves to be followed by the highest point of each chord. Other entries (not used in the example) can define parallel or contrary movement of voices (entry *nb-mouv//ok*), chord intervals contents as a function of register (*filt-pas-band*) or lower bounds on the total number of different notes in arbitrary sections of the sequence (*renouv-haut*). Figure 5 shows part of a sequence computed by *Niobé* from the constraints specification of figure 4. The rhythm has been

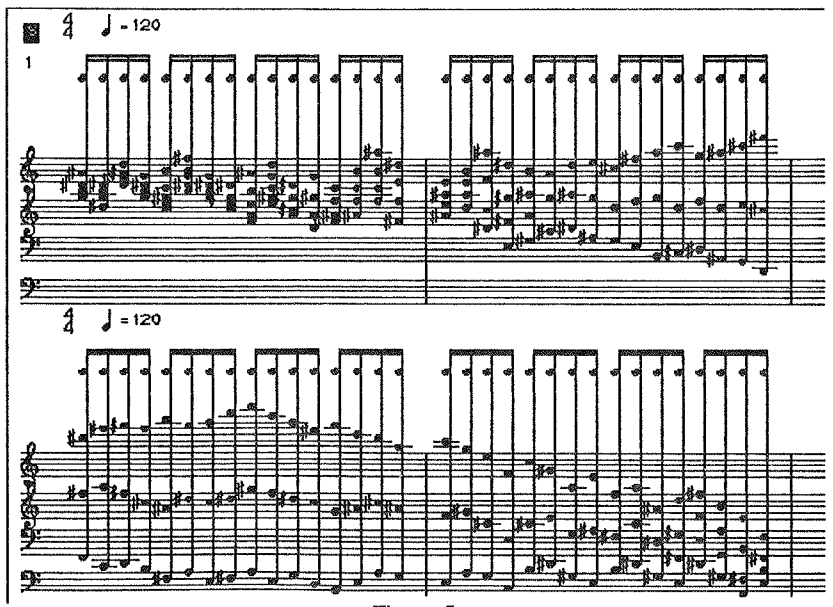


Figure 5.

arbitrarily set (in one of the entries to the *rtm* box) to eighth notes. The above example solves a chord sequence generation problem by a simple adjustment of parameters of built-in harmonic and melodic constraints. This is only one way of proceeding. The composer can draw any patch computing an arbitrary constraint and then connect it to one of the entries of the *harmonic-constraints* box. *niobé* will then compute instances of the sequence taking into account the new constraint. Figure 6 shows one example, used by the italian composer Marco Stroppa to compute a sequence of chords following a given progression of *homogeneity* , defined as the difference between the biggest and smallest chord intervals. The patch to the left of the figure takes in the *parameter* box the current chord proposed by *Niobé* and also its position in the sequence. The patch computes the homogeneity of the chord and tests it against the acceptable range desired for a chord in that position of the sequence (see box *interval-interp*), giving TRUE (acceptable chord) or NIL (non-acceptable) accordingly.

IV. Conclusions.

We have described *Niobé* , a music composition system integrating the relational and functional conceptions of programming through the use of a visual programming environment. *Niobé* has successfully been used by several composers to compute harmonic and rhythmic sequences. A refined

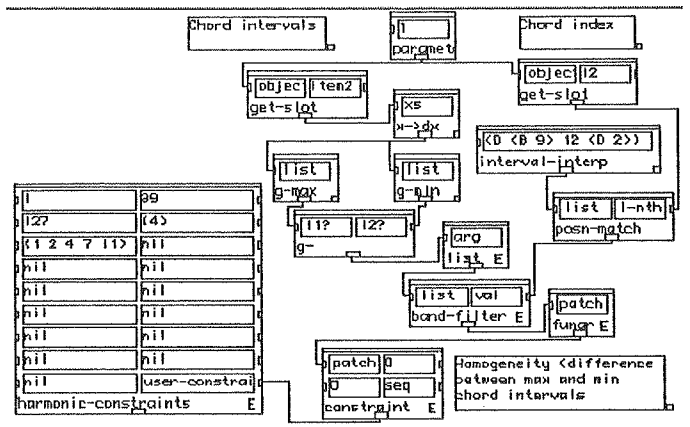


Figure 6.

version of the example in figure 4 was used by the french composer Antoine Bonnet in his piece *Epitaphe*. The german composer Michael Jarrell has used *Niobé* to generate melodic sequences for his piece *Rhyzomes -IV* . We are currently studying more uniform models of integrating the functional (and object-oriented) and relational aspects in *Niobé* . Although *Niobé* has proved to be reasonably efficient (a few minutes in a Mac Quadra 700 for computing sequences of less than a hundred chords) in several practical situations, we have accomplished that at the price of restricting the allowed types of *horizontal* constraints. We are exploring different optimizations of the base algorithm to allow the specification of more global vertical or horizontal relations.

References

Assayag, G. & Rueda, C. (1993). The Music Representation Project at IRCAM. *Proceedings of t he*

ICMC . Tokyo, 1993.
Deville, Y. & Van Hentenryck, P. (1991). An Efficient Arc Consistency Algorithm for a Class of CSP
Problems. *Proceedings of the IJCAI* . Sydney, 1991.
Lurson,M. Duthen, J & Rueda, C. (1992). The PatchWork Reference Manual. *IRCAM* , 1992.
Mackworth, A. (1977). Consistency in Networks of Relations. *Artificial Intelligence* . 99-118.
Ovans, R. (1990). Music Composition as a Constraint Satisfaction Problem. *Proceedings of the ICMC* .
Schiex,T. (1992). Possibilistic Constraint Satisfaction Problems or "How to Handle Soft Constraints".
CERT-ONERA . Personal e-mail communication.
Schottstaedt, B. (1983). Pla: A Composer's Idea of a Language. *Computer Music Journal* . 7(1).
Sidebotton, S & Havens, W. (1991). Hierarchical Arc Consistency Applied to Numeric Processing in
Constraint Logic Programming. *CSS-IS TR 91-06* . Simon Fraser University, Burnaby, Canada.
Steele, G. (1990). *Common Lisp: The Language* . Digital Press.
Taube, H. (1991). Common Music: A Music Composition Language in Common Lisp and CLOS.
Computer Music Journal . 15(2).

Incremental evaluation in a musical hierarchy

M. DESAINTE-CATHERINE
K. BARBAR and A. BEURIVÉ
LaBRI[1]
Université Bordeaux I
351, cours de la Libération
33405, Talence Cedex
France
myriam@labri.u-bordeaux.fr

Abstract

The work we present in this paper is a formalism of a dynamic computational model in a hierarchy. We are interested in representing musical hierarchies and bindings of characteristics (such as the mode, measure, tempo, duration, key, etc.) within them in order to provide the composer a means to verify the consistancy of the piece during the compositional process. The model transfers any modification from the composer to the representation in an incremental way, without computing again the whole hierarchy.

1 Introduction

The complexity of a musical piece can be organized in a hierarchical way based on its temporal structure. Musical characteristics (such as the mode, measure, tempo, duration, key, etc.) can be defined at any point of the hierarchy (that is any sub-piece). These characteristics are then bound together according to the temporal structure and the musical rules imposed by the composer. We are interested in representing musical hierarchies and bindings of characteristics within them in order to provide the composer a means to verify the consistancy of the piece during the compositional process.

Our work may be situated between constraints propagation techniques and hierarchical representations à la Balaban. We are interested in designing the representation and the computation model which is appropriate to it. From our point of view, a musical piece is an object that is composed of several dimensions. Classic dimensions are time, frequency, timbre and volume. The variations of the values in these dimensions are not independent from each other. The result of a musical analysis is exactly a set of correlations between variations within a single dimension and between different dimensions. In order to formalize those correlations, we define several relational operators which are dedicated to specific dimensions. The set of values in each dimension can then be structured in a hierarchical way using these operators. Hierarchical way means that the object representing the structure is not always a simple tree, but a directed acyclic graph (see the notions of shared occurrencies and repetitions of Mira Balaban (Balaban 1993)). The originality of this work relative to the others based on hierarchical representations is the addition of a semantics to the hierarchy. This semantics provides a very sound way to represent

[1] Laboratoire Bordelais de Recherche en Informatique – Unité de Recherche Associée au Centre National de la Recherche