EV: Multilevel Music Knowledge Representation and Programming

Jesus L. Alvaro[1], Eduardo R. Miranda[2], and Beatriz Barros[1]

[1] Departamento de Lenguajes y Sistemas Informáticos, UNED, Spain
JesusLAlvaro@gmail.com
bbarros@lsi.uned.es
[2] Computer Music Research, University of Plymouth, UK
eduardo.miranda@plymouth.ac.uk

Abstract. *This paper introduces EV Meta-Model, a new system for representing musical knowledge for computer-aided composition. It starts with a brief historical discussion on the fields of composition and sound synthesis. Then, the practice of musical composition is presented as a communication process from composers to listeners, where musical messages go through different representations: from the complex abstractions of composers and their compositional tools, to the performers and the perceptual representation of listeners.*

EV Meta-Model is proposed as a generic tool for representing any kind of time-based events that manifest themselves as coherent representations at different levels, including high-levels of musical abstractions. At the same time, it is intended to be a dynamic representation system, capable of handling each element as a "living" variable, and transmitting such dynamic character to the music that it represents. As examples of its applicability, the paper presents the Evscore Ontology, the implementation of EVcsound, a tool for the creation of detailed compositions with flexible temporal representations, and finally an example of algorithmic composition upon the EV Model.

1. The Representation of Musical Knowledge

A suitable knowledge representation tool (KR) is fundamental for a successful Artificial Intelligence system development. Brachman explains the KR concept with these words [Brachman, Levesque, 1985]: “It simply has to do with writing down, in some language or communicative medium, descriptions or pictures that correspond in some salient way to the world or a state of the world". The effectiveness of the intelligent system will depend, to a great extent, on how that KR fits the problem domain.

It can therefore be established that the first step for the design of an effective musical composition system is the definition of a musical KR that is appropriated to the domain and to the creative manipulations within the domain.

Conventional music notation allows for communication between the composer and the interpreter. But in this case, the compositional knowledge corresponding to the

mental abstractions of the composer is not represented explicitly. The compositional techniques, their experience, their creative processes and their intentions are not clearly represented by conventional music notation. In order to study what this compositional knowledge consists of, an analysis of the composition and the creative processes involved should be developed.

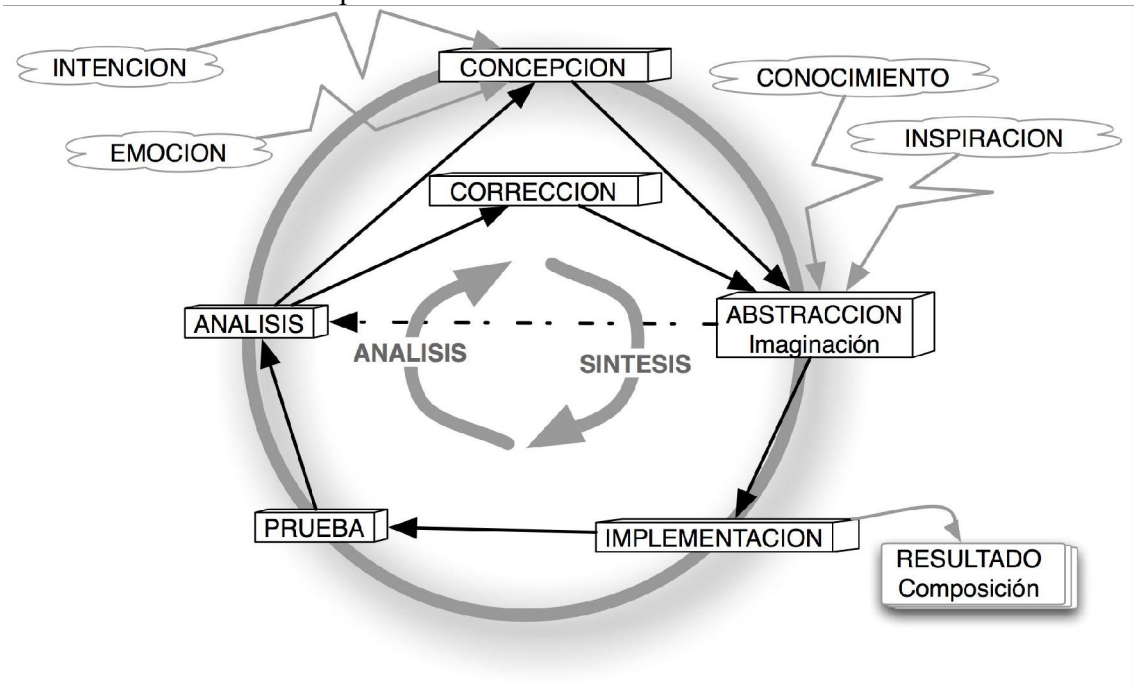

Figure 1. Subprocesses Cycle in Composition

Composition is comprised of subprocesses in the composer abstraction, such as conception, abstraction, imagination, implementation, analysis and correction. Figure 1 is a representation of an analysis model of composition processes. It is shown as a cyclic process of subprocesses. Starting from a voluntary intention or from an emotion, an element is conceived. Following it is imagined, and abstracted in the context from both experience and musical knowledge, and also, why not, from inspiration. Once imagined, the element is concretized, implemented, and tested. Often, the implementation is not explicit, but a mental visualization of the result is enough. Sometimes drawings and sketches are used for testing and analysis. Once evaluated, the affections to the rest of the composition are analyzed. As a consequence, new corrections are conceived starting a new cycle again. Every musical element or structure in the composition could be seen as a product of this kind of synthesis-analysis cycle.

In this analytic search of musical knowledge, several components have been identified. They include entities, relationships, procedures, strategies, and metaknowledge. They are explained in table 1. We can conclude that the musical knowledge comprises aspects, elements, procedures and strategies brought into play by the composer during the music creation process.

Table 1. Components of Musical Knowledge

Elements and entities,	Conceptualization and identification of elements and abstract objects arranged into ontological classes. These elements might be considered at different levels, from atomic elements such as notes, to more and more complex structures constructed by combinations of simpler elements. Objects and relationships
Rules, patterns, constraints	Relationships between the entities above, creating a definition of musical language and style.
Intention driven procedures	Procedures to develop the musical discourse, the temporal evolution, the narrative line.
Rules breaks, originality	Exceptions to the rules, patterns, creative innovations at all levels of relationships and structures. Providing liveliness, originality and freshness to the discourse.
Strategies	Based on experience, they constitute a whole heuristic of composition as a solution search
General Criteria, Global Intentions, Finality	Criteria above the composition process, as motivation, aesthetics and intention for composing. Some criteria as attention attraction, surprise, equilibrium. Meta-knowledge.

2. Paradigms in Music Representation

2.1.The score paradigm

The traditional musical score can be considered as a set of symbols arranged in a temporal succession, representing musical entities such as notes, durations and pitches. All annotations in the score are symbolic representations of instructions indicating what the interpreters should do in order to play the music. In this sense, the score is regarded as a good representation for the performance of music. One could hardly consider musical scores as a comprehensive KR method because information about the composition itself has to be deduced by means of an objective analysis of the score or obtained through subjective interpretation that is part of the listening process.

Being the performance instructions specific to different types of musical instrument, the composition is not completely defined without the specification of the instruments and the contribution of the interpreters. Therefore, attaching the corresponding orchestra completes the definition of a composition. Considered from the point of view of its function, an orchestra could be defined as a sound generator that has some knowledge about the interpretation of the score and its translation into acoustic waves. This *Orchestra-Score metaphor* is implemented in numerous sound synthesis systems, such as MusicV and its descendants Csound [Vercoe, 1994], CLM [Schottstaedt, 1994] and a few others. In Csound, for example, the composition is split into the score file *(.sco*) and the orchestra file *(.orc*).

2.2.The unit generator (UG) paradigm

The architecture of the first analog synthesizers has greatly influenced the development of sound synthesis systems. Those machines required the physical interconnection of small operating units called unit generators (UG). By means of the combination of UGs, multiple architectures with different performance could be assembled.

This philosophy has been traditionally applied in numerous systems for musical composition and sound synthesis. For example, in Csound instruments are programmed using variables to connect basic op-codes building instruments as more complex operating elements. In CLM and Nyquist [Dannenberg, 1997] the interconnection of UGs is defined using the LISP language, and in some other systems, such as PD [Puckette, 1997], these connections are done by means of a graphical interface.

From a point of view of KR, those structures could be represented as elements of the type 'operating-unit-in the-time-domain', defined as an interconnection of simpler units. It is interesting to note, however, that the UG paradigm allows for the definition of high-level structures starting from similar structures in a lower level.

2.3.The MIDI paradigm

MIDI standardization has unquestionably greatly influenced the development of computer music. In music representation, it considers that music is performed with a keyboard. This simplification provides the possibility of representing every musical note just with a number associated with the key being pressed, the pressing velocity, and when and for how long it is pressed. MIDI has been useful as a music representation for the last decades, but it cannot be considered as a complete representation system, as it does not include any information concerning the orchestra. Although some attempts have been made to standardize some orchestras (e.g., General MIDI), the impossibility of unifying MIDI players and it low resolution, prevent MIDI from being an efficient tool for representing musical sound. However, it provides the possibility of developing systems with low computational costs. Within algorithmic composition, systems such as Common Music (CM) [Taube, 2004] or Symbolic Composer (SCOM) [Stone, 1997] are examples of MIDI-based systems implemented in LISP.

3. From Composer To Listener

We consider the musical phenomenon as a communication process starting from the abstractions and emotions of the composer towards the ear of the listeners. Figure 2 shows the different representations that musical messages go through from the abstract world level, down to the acoustic level. The acoustic level can be reached through two possible ways.

In the traditional way, a human interpreter plays the conventional score found in the notation level. Composers write the score from mental abstractions. Usually, compilation does not take place directly, but sketches, scripts, plan drawings, drafts and many other methods are used. This is a long elaboration process where composers repeatedly go back to higher levels to perform adjustments and make finer conceptualizations of their abstractions. It is an iterative and feedback debugging

process, where many different paths are tried out and their results evaluated. From the point of view of Artificial Intelligence, this process could be considered as heuristic searches for the satisfaction of self-imposed constraints.

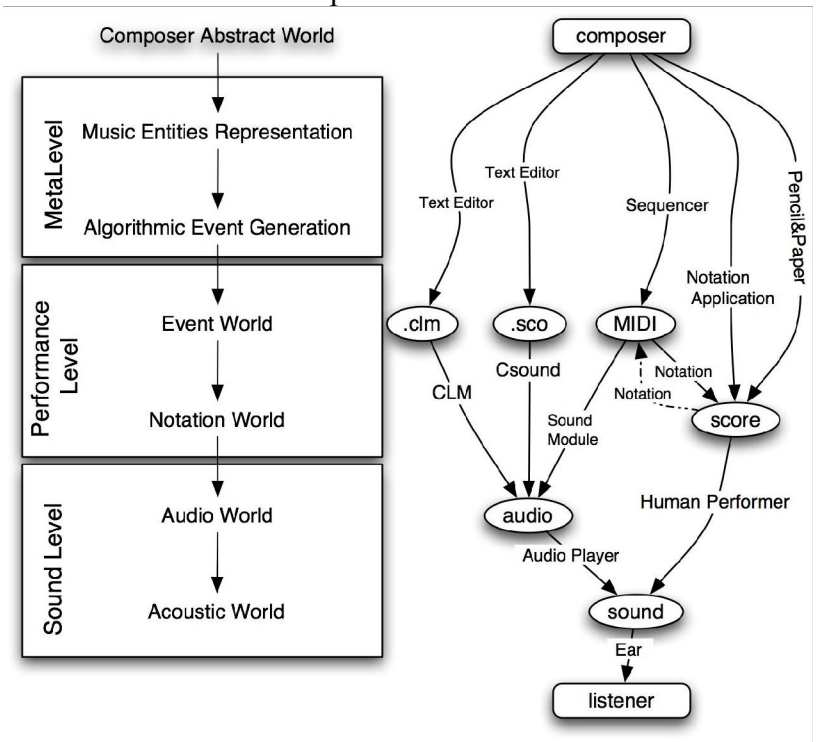

Figure 2. Levels of representation in musical communication

In conclusion, it is a time consuming work developed at levels above the level of the interpretation languages.

The other way of producing sound comes from a stratum immediately above, the audio world. A speaker, reproducing audio samples, now emits sound. Sound files are created based on performance events at the interpretation level like MIDI, *Csound* . *sco* or any other event list created by the composer.

3.1.The metalevel paradigm

The interactive composition process described for the creation of the score is similarly applicable in this case. The metalevel is not seen here just as a layer in the levels of structure, but as a stratified zone capable of supporting different architectures. In that area, we could place the following:

- Newell's "*knowledge level*" [Newell-1982], corresponding to the symbolic level of the score, where the musical entities of score are situated.
- Conceptualizations of musical elements in the dimensions of time and form, not directly represented in the score, such as motives, sequences, form structures and their relationships.
- Extra-score conceptualizations in the spectral or pitch dimension, such as intervals, chords, harmonic elements, tonalities and so on.
- New *"musical meta-objects"* at closer to composer levels.
- Elements and procedures of algorithmic musical generation.

The metalevel is an attractive area for the development of compositional tools. It is, however, necessary to use an efficient KR that is flexible and able to cope with new entities at several levels, and which can integrate new musical elements and structures created by the composer.

Music created with computers is often regarded by the general public as mechanical and lifeless. Note that in conventional music communication, both the composer and the interpreter contribute their human dimension to the music, In this case, the final sound carries creative contributions and "beautiful imperfections", which manifest the richness and liveliness of its human origin. This is a target not to be forgotten when designing a representation system for musical composition.

4. EV Meta-Model: a Multilevel Music Representation

EV Meta-Model is proposed as a KR for elements in time at different levels. One of the key points of this approach is its simplicity. Figure 3 shows the core of our ontology, consisting on three main classes: the *event*, the *parameter* and the *dynamic object*. The definition of the classes is given as follows (in LISP notation):

```
(define-Class EVENT :is-a evclass
 :slots ((start-position :type real)
         (length :type real)
         (parameters :multiple list :type parameter)
         (events :multiple list :type event)
         (position-function :type function)))

(define-Class PARAMETER :is-a evclass
 :slots ((name :type string)
         (value :type dynamic-object)))

(define-Class DYNAMIC-OBJECT :is-a evclass
  :slots ((type  :doc "Type of the out value")
          (out :type t)
          (dyn-function :type function)
          (dyn-arguments :multiple list :type DYNAMIC-OBJECT)
          (status-memory :type t :doc "State memory")
          ))
```

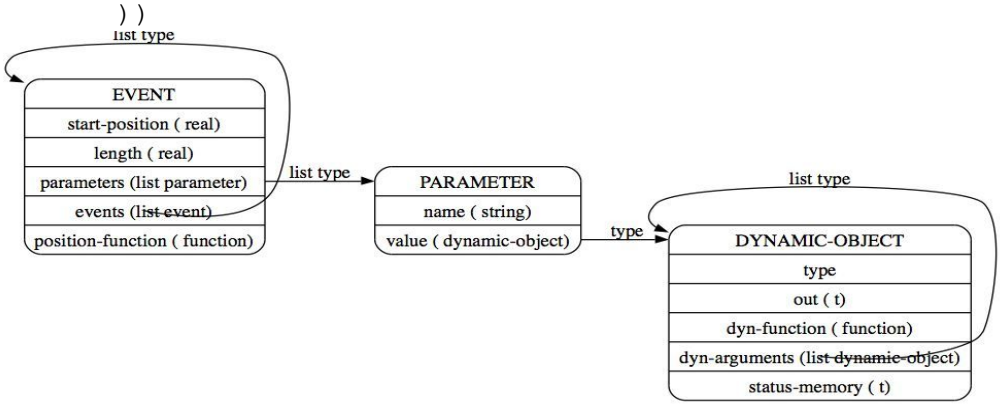

Figure 3. Ontological Core of EV Meta-Model.

4.1.The *event* class

This class comprises any kind of element that can be positioned in time. It is the central class of the ontology and it includes five main slots:

- *Start-position* is the time position of the event, the time position where it starts.
- *Length* indicates the active time, or the duration of the event.
- *Parameters* is a list of the properties of an object. Those properties (or parameters of a musical event) are instances of the class "parameter". Each parameter includes a parameter name and a dynamic value of type *"Dynamic-Object"*, which is described in next paragraph.
- *Events* is the most important slot. It makes recursivity possible. Therefore it provides a multiple level representation capability. It contains a list of new events sharing the same object structure; it is a list of instances of its same class *event*. A "child-event" could itself contain new "child-events" and so on, thus creating a rich recurrent data tree from a single class. This concept drastically simplifies the definition of time structures at different levels and their relationships. A general behaviour could also be defined by generic functions. It is, therefore, a flexible ontological skeleton capable of representing almost any musical element as an instance of the class *event*. It keeps coherence from the notes level up to higher abstraction levels.
- *Position-Function* is another interesting feature that makes this approach unique. Each event could have its own time magnitude and scale with a dedicated handling. This means that there is an internal time inside the event, defined by its *Position-Function*, and a different time perception from the outside, which corresponds to the time organization of its "mother-event". That timing flexibility multiplies the creative possibilities and, at the same time, solves several practical problems. Let's see some examples:

1. Let us consider a practical example in film scoring. A film would be seen as an instance of the class *event*, using *SMPTE timecode* as time organization. A musical cue for the score is an *event* that begins at a specific timecode and ends at another, but it uses bars and meter from traditional notation for its internal time organization. All internal events in this cue, such as notes, are defined within this bar organization. In this example, the musical cue owns a *Position-Function* or time function called "tempo", which translates those time magnitudes. In live performance, the conductor will take the control of that Position-Function. The following code shows a possible implementation by subclass definitions:

```
(defIne-Class FILM :is-a event
  :slots ((position-function :default #'smpte2real)
          (events :multiple list :type MUSIC-CUE)
          ))
(defIne-Class MUSIC-CUE :is-a event
  :slots ((master-track :type master-track )
          (position-function :default #'meter2smpte)
          (events :multiple list :type MUSIC-SEGMENT)))
(defIne-Class MUSIC-SEGMENT :is-a event)
```

2. A previously defined musical motive could be used at several moments of the piece at different speeds, by augmentation or diminution: a generalization of the traditional counterpoint procedure. For example, in order to decrease the speed by two, it is enough to provide the function *#'double* as position-function. If we use the function *#'half* we would get the counterpoint diminution.
3. Once the piece is completed, its timing could be reshaped, enlarging and shortening some sections by modifying the main position-function. In film scoring, it could be useful in many situations. Let us consider the last-minute director's change in the edition of an already orchestrated scene; some small adjustments in the position function could be enough to fit the new timing.

4.2.Dynamic Objects

The parameter value of the *event* is not static. It is considered to be a *dynamic object* with an evolving value. Data is dynamic during the event, so a parameter could be considered as "living" entities. Several possibilities are available for defining the evolution of the dynamic object. In order to simplify definitions, a set of elementary *dynamic objects* and an intuitive syntax to describe combinations was developed inspired by CYBIL [Piché, Burton, 1995]. Complex objects built from simpler ones will keep the dynamic character. The set of elementary units includes *lines* between an initial and a final value (the beginning and ending of the event), *logarithm* curves, *random* values within a range, *sequences* of values, *functions*, memory provided *generators*, etc. Recursivity is again present here because arguments included in the definition are also considered as instances of *dynamic objects*. In addition, dynamic objects are conceived with memory capability.

Both dynamic arguments and memory, provides the possibility of creating very complex dynamic objects, such as evolving systems, cellular automata and so forth.

4.3.Generator events

EV Meta-Model also provides the events with *behavior*. A subclass can be defined from the main class *event* with a specific behaviour. Generators are a special subclass type, which are useful for the abstraction of musical forms. A generator is an *event* that develops itself into new events of a particular subtype. From a musical point of view, generators provide the possibility of compiling a "musical piece" *event* instance into a score of interpretation events iteratively.

5. EVScore Ontology

Figure 4 shows the *EVScore Ontology,* an example based upon our proposal. It is a traditional notation compatible representation, which can integrate higher level music elements. Every class in this ontology is a descendant of the core class *event,* so they inherit all the properties of the Meta-Model.

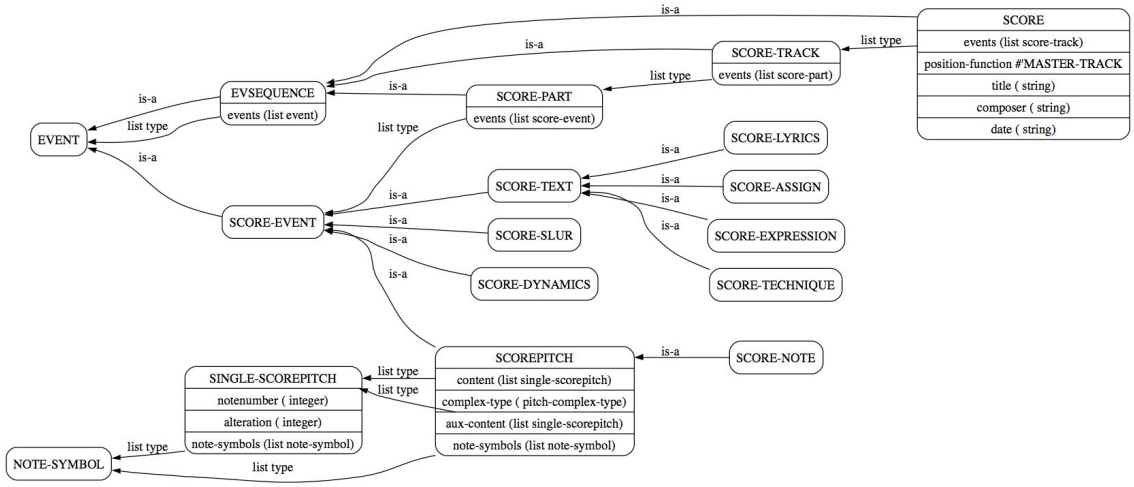

Figure. 4. EVScore Ontology

6. A Test Implementation: EVCsound

In this section we present a practical application using the EV Meta-Model: EVcsound. Less than 100 lines of code were necessary for this implementation, as 95% of the requirements were already available in the EV Meta-Model.

EVcsound compiles a composition-event into a Csound *.sco* score. Every *csound-event* in the *.sco* file is associated with an instrument of the *.orc* file by an identifier and it consists of a list of numeric figures corresponding to parameters *p4* to *pn* for that instrument (p1 is the instrument ID, p2 the start position and p3 the duration of the event). The key point of the system is the *sampler-generator* type event, which samples the dynamic-objects along its development process and builds up every *csound-event* with such figures.

The code bellow is a simple example with two events of type *sampler-generator*, which develops itself into *csound-events* for the instrument referenced by its name.

```
(evcsound '(0 20 superfm
                ( period (ran f .2 2)
                  amplitud-db (ran f (li f 50 70)
                                     (li f 60 80))
                  duration (op #'/  300 (pa amplitud-db))
                  pan (ran f  (lo f 0  .5) (lo f 0 .99))
                  pitch (op #'* 55 (ran i 6 16))
                  reverb-send (lo f .07  .7)
           0 25 reverb-st
                 ( reverb-time 7))))
```

The first generator, lasting for 20 seconds, develops into multiple *csound-events* for an instrument named "*superfm*", with a random sampling period between 0.2 and 2 seconds, a random amplitude in the evolving range between 50-60 dB and 70-80 dB, a random pan evolving from *closed L* to a range between C and R, a duration inversely

proportional to amplitude between 3 and 8 seconds and a randomly chosen harmonic pitch between the 6th and the 16th harmonics of A1 (55Hz). The second event, lasting for 25 seconds, activates the reverb instrument. This example presents just one step down in the hierarchy; a compilation from such a high level can be easily achieved by defining recursive events.

7. An Algorithmic Composition Example

As a last example of applicability of the EV framework, a simple example of algorithmic composition of a melody is presented. In this example we also introduce three new elements of our system: tables, zones and maps.

Tables are used as a new kind of dynamic object. They can store the output of sequential objects and use them as a linear function. In our example, a random *brownian* shape is stored in a table to be used repeatedly.

Zone is a particular subclass of *generator event* inside which, a detailed time structure is defined. Categories for every zone in the structure are defined at the same time so it is easy to compare them and set a desired evolution for that category across the zones. All category definitions are interpreted within the zone structure, so it is easy to address the desired zone by writing according structures in the definition. If just one dynamic object is provided, it is applied for the entire event. If a nested list is provided, each object is associated with the "same level and order" zone element. The zone-event generator can be developed into child events according with the defined parameters in zones.

Maps are dynamic objects that allow you to use symbol lists for representing recurrent structures of data in a simpler manner. Symbols are easily combined and manipulated to define a sequence of elements generating a musical form. This concept can be extended by the use of grammars to generate complex and rich forms. In the example below, they are defined by the dynamic function *mp*.

```
(deftable t1 256 (br 0 1 :seed .42723))

(define-zone-event my-melody
    :zone '((a b b c) ( a b c b c))
    :ryth (mp zone '("+-++" "+-+-" "+---"))
    :shape (mp '(a b) (list t1 (retrograde t1)))
    :tessitura (tab t1 :min 0 :max 12)
    :ambitus (li i 5 8)
    :pitchclass 'glydian
    :step0pitch  'g4
    )
```

The example code above defines a melody as a zone-event. In this subclass, the melody generation is developed by the algorithm represented in figure 5: A melody shape is obtained by placing a shapes structure within a defined tessitura and ambitus. That melody shape is then sampled by a rhythm and step quantified in some intelligent way. The resulting step values are then converted to pitches by mapping them into a given pitchclass and register.

In our example, rhythm is defined as the expansion of the zone structure, by using a figure map. The combination of zone and rhythm also define the time map or the duration of every segment in the musical form. Pitch is obtained by mapping the sampled and quantified shape, into the lydian scale on G and using the reference of pitch G4 as step number 0.

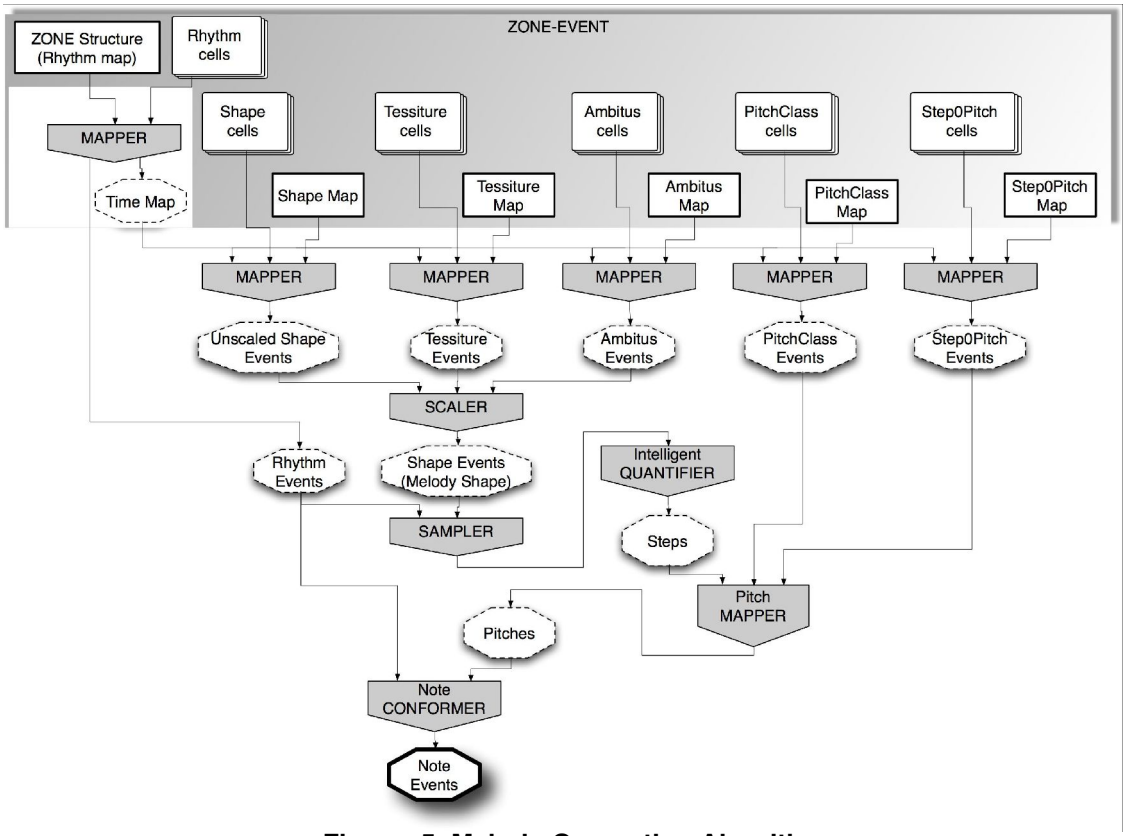

Figure. 5. Melody Generation Algorithm

One of the important features of zone-events is the coherence of the music form and it applicability to several categories independently. That is, all categories can share time structure but they manage their time relationships in an independent way.

Figure. 6. Generated Melody

Figure 6 represents the melody generated by the code above. The presented code is just an example of the combination of the algorithm over the EV Model. It is not the purpose of this section to give a detailed approach to algorithm composition inside EV, but just to provide a simple musical example of its successful support for algorithmic composition.

8. Conclusions

The creative possibilities of a musical system are determined by the underlying knowledge representation and the paradigm in which it emerges.

Although the traditional score is a good representation for performance, actual musical knowledge extends far beyond the traditional musical score. Musical knowledge includes elements and entities, rules and constrains, intention driven procedures and creative breaks of patterns and rules. Musical representation could be stratified into levels spanning from the composer abstraction to the acoustic representation. We are interested in addressing the zone above performance or metalevel. A successful knowledge representation for composition should be flexible and simple, but still providing coherent representation at different levels.

EV Meta-Model emerges in this research enquire as an intuitive multilevel music representation system. Its design is based on both structure and function recursivity, making it appropriate in multiple levels and magnitudes. EV Meta-Model provides a dynamic data structure, being able to successfully implement musical ideas efficiently. This approach introduces innovative aspects to music computing research. One of the key points in its design is the unification of all musical entities in a single data type, designed specially for music.

9. References

Brachman, R.J. and Levesque, H.J. (Editor) (1985). "*Readings in Knowledge Representation*", San Mateo, CA: Morgan Kaufmann.

Dannenberg, R.B. (1997). "*Machine tongues XIX: Nyquist, a language for composition and sound synthesis*", *Computer Music Journal*, 21(3):50-60.

Newell, A. (1982). "*The knowledge level*", *Artificial Intelligence*, 18(1):82-127.

Piché, J. and Burton, A. (1995) "CYBIL Language" Université de Montréal http://emu.music.ufl.edu/cecilia/cybil.html

Puckette, M. (1997). "*Pure Data*", *Proceedings of ICMC*, Thessaloniki, Greece, pp 224-227.

Schottstaedt, B. (1994). "*CLM: Music V Meets Common Lisp*", *Computer Music Journal*, 18(2): pp 30-37.

Stone, P. (1997) *"Symbolic Composer"* http://www.symboliccomposer.com

Taube, H. K. (2004) *"Notes from the Metalevel",* Andover UK: Swets & Zeitlinger.

Vercoe, B.L. (1994). *Csound: A Manual for the Audio-Processing System*, Boston, MA: MIT Media Lab.