Co-Algebraic Modeling as an Adequate Means for Representing, Transforming and Discussing Transcendentally Defined Æsthetic Objects
(extended abstract)

Markus Lepper, Baltasar Trancón y Widemann

Technische Universität Berlin, Fakultät IV,
Institut für Softwaretechnik und Theoretische Informatik,
Fachgruppe ÜBB, Sekr. FR 5–13,
Franklinstr. 28/29, D–10587 Berlin,
E-mail: {lepper,bt}@cs.tu-berlin.de

1 Basic Aspects of Computer-Based Modeling of Music

1.1 Informatical Analysis and Synthesis as Device for General Cognition

Under certain circumstances the usage of digital information processing can be of some benefit even in humanities („Geiſteswiſſenſchaften“).

An outstanding example is *music*, because the structure of its *material* is already quite similar to mathematics.

One of the basic axioms underlying the work of the authors' research group is, that applying „computer science“ to a given problem in a first step always has to explore, extract and explain the *mathematical contents* of that problem. (This is often called „applying Formal Methods“.)

If, in a second step, computer *technology* shall be applied, the results of this analysis *necessarily* form the basis for every efficient implementation (since even the computer cannot solve problems, which are mathematically unsolvable ;-)

But also without any technological context, the result of applying analytical tools of computing science can enrich our insight into the structures of the problem: A mathematical model can be very helpful for more preciseness even in discourses on a pure conceptual level, for clarifying the underlying structures and material „boundary conditions“ of a certain weltanschauung [1].

A specialization of this principle occurs in language design, — a central topic in the authors' research group: As soon as a complex process, e.g. in public administration, can be mapped to or modeled by a specially defined (or adopted) language, then this process can be specified with the appropriate accuracy, and potentially brought into clearer focus.

[1] Cf. the model of the compositional process in [2].

1.2 Models, Meta-Models and Executability

In the following, each data object that is meant to describe one musical corpus is called a *model*.

Then any *meta-model* defines a family of possible models, — any meta-model is a description, how to encode certain semantics into a corresponding model, and, inversely, how to interpret a given model. Further a meta-model may impose consistency constraints on all of its models.

A *meta-meta-model* is a high-level language for describing (or „specifying“) meta-models.

A meta-model is called *executable*, if its models are executable. A model is executable[2], if it is not only a static representation of fixed information, but shows a dynamic behaviour: At least some „new“ pieces of information, which have not been put explicitly into the model during its construction, must be derivable by applying computational rules to the stored information.

Moreover, an executable model is able to perform local updates, ie. local changes of its information state, and automatically re-establish all integrity conditions and recalculate all derived values which are affected by this alteration.

Computer-aided modeling is of course really beneficial only if the model is executable. Practically, this can only be achieved by enriching the corresponding meta-model with rules for execution (or compilation), which can be applied to *all* of its instance models.

The approach described in this paper goes one step further: It presents the outlines of one certain *meta*-meta-model, called m_0^2, which contains evaluation rules sufficient for the executability of any model of any meta-model constructed in m_0^2.

[2] or „deductive“ in the nomenclature of data base engineering

Figure 1 Type Construction Language

$$
\begin{array}{lll}
\mathit{type} & ::= & \mathit{ID} \\
 & | & \mathit{Nat} \mid \mathit{Num} \mid \mathit{Rat} \mid \mathit{Real} \mid \mathit{Text} \mid \ldots \\
 & | & \mathit{ident}\ (\ \underline{\texttt{[}}\ (\mathit{type} \cup \mathit{expr})^{+}\ \underline{\texttt{]}})^{?} \\
 & | & \mathit{type}\ \texttt{ONLY}\ \mathit{constraint} \\
 & | & (\mathit{ident}\,\underline{\texttt{:}}\,)^{?}\ \mathit{type}\ (\ \underline{\texttt{*}}\ (\mathit{ident}\,\underline{\texttt{:}}\,)^{?}\ \mathit{type}\ )^{+} \\
 & | & \mathit{ident}\,(\,\underline{\texttt{:}}\ \mathit{type}\ )^{?}\ (\ \underline{\texttt{+}}\ \mathit{ident}\,(\ \underline{\texttt{:}}\ \mathit{type}\,)^{?}\ )^{+} \\
 & | & (\ \texttt{OPT} \mid \texttt{SET} \mid \texttt{BAG} \mid \texttt{SEQ}\ )\ \mathit{type} \\
 & | & (\ \texttt{MAP} \mid \texttt{REL} \mid \texttt{FUN}\ )\ \mathit{type}\ \underline{\texttt{=>}}\ \mathit{type}
\end{array}
$$

Figure 2 Defining Algebraic Types for Metric Specifications

```
IN PACKAGE cwn  {   // Common Western Notation
  TYPE meter = num : std.nat ONLY {2 3 4 5 6 7 8 9}
                  * den : std.nat ONLY {2 4 8 16 }
}
IN PACKAGE special_for_composer_ml {
  TYPE meter = sum : SEQ fract * sub : OPT fract
  TYPE fract = num:std.nat * den:std.nat
}
IN PACKAGE special_bulgarian {
  TYPE meter = nums : SEQ std.nat
             * den  : std.nat ONLY {2 4 8 16 }
}
```

1.3 The Rôle of Models and Meta-Models in Practice

In traditional programming the usage of models is separated into two disjoint worlds, called „file formats" (or „transmission encodings") *vs.* „internal representation". When describing file formats, the meta-models are used in a „passive" way, because evaluation is only needed on a conceptual level, for describing the intended semantics. Nevertheless the model is somehow „active".

The modern standpoint is substantially different: How much of the semantics of the meta-model will indeed be „implemented", and how much will only be present on the conceptual level, is a purely technical question of secondary range. In any case the semantics of the meta-model should be meta-meta-modeled consistently and most completely.

The difference between a „disk file", i.e. some encoding of the models for the sake of data interchange, and the „active" model maintained „in" the computer is indeed just a technical one.

1.4 Existing Meta-Models from Academic and Industrial R&D

While the following discourse tries to extract the central deficiencies of nearly all existing systems, it does in no case imply any neglection of the important and good work done in all these approaches. On the contrary: the basic propositions for the design on meta-meta-level contained in this paper could not be described without the intensive cultural discussion, achieved by the exchange of these meta-models[3].

Their first major deficiency is simply that they *are* meta-models, i.e., they constrict modeling, since they are not extensible w.r.t. the core semantics of the *algebraic* types used to encode the different attributes.

But the decisions in favour of certain basic attribute types are indeed *ideological*, they pre-determine the set of possible models by supporting different „flavours" of composing more or less conveniently, and are often inadequate for specific æsthetic purposes.

Figure 2 shows definitions, i.e. meta-models, for different algebraic types to represent *metrical specifications*, which respectively allow denotations like...

$$\frac{3}{4} \quad (or) \quad \frac{3+2+3}{8} \quad (or) \quad \frac{3}{2} + \frac{1}{4} - \frac{1}{12}$$

In contrast, figure 1 shows a language which serves as our meta-meta-model, suitable for defining the meta-models in figure 2, as well as the co-algebraic types introduced below.

1.5 Case Study: Attaching Arbitrary Additional Information to Some Given *Corpus*

To show the necessity of co-algebraic information modeling, let us simply follow closely the way of operation when assigning some additional information to some given object:

What we have to do is ...

1. to provide (a computer-readable encoding of) the information to be attached,
2. to identify the event (or set of events) to which the information is attached, and
3. to choose some identifier for later retrieval/usage of the stored information.

The first step must be provided by the „user", but should not impose severe difficulties, since the propositions do aim at attaching *any* kind of information. Of course this implies that the user provides the information in some computer-readable meta-model, and that there is an *encoding*, which allows to realize the information physically within the overall framework. The latter is merely a technical problem, and the lately standardized „XML" meta-encoding system seems an adequate basis for defining these required encodings.

[3]A comprehensive survey of the most important models used for information exchange („file formats") is given by the webpage [1]. Due to lack of space please refer e.g. to [3] for further bibliographic data.

Figure 3 Defining the Co-Free Type of „bar“ Objects

```
IN PACKAGE cwn  // Common Western Notation
  COFREE TYPE timedEvent
    =   tstart * tend * tdura * tscale
      * tsimul * tpred * tsucc ;
// In a co-free type definition, all factors
// of a product are implicitly optional!

  COFREE TYPE bar = timedEvent**(metricSpec:[T]);
  DEF metricSpec =  ?metricSpec
                 || ?tpred >> ?metricSpec
      ON EVERY bar ;
  DEF tdura:std.msec
    = ?metricSpec:std.rat * (60000.0*4/(?tempo:MM))
      ON EVERY bar ;
  LAW (?tstart:std.rat + ?metricSpec:std.rat
         = ?tend:std.rat)
      OR (NOT ?tpred)  OR (NOT ?tsucc)
      ON EVERY bar ;
```

The third step simply requires some house-keeping or generation rules to keep identifiers unique.

1.6 Crucial Point: The Identity of Events

The crucial step is the second one: How can an identity of e.g. an *musical event* be established ?

First we see that all above-mentioned meta-models use some kind of *container* objects for organizing event data. These containers may be called „voices“, „tracks“, „chunks“, etc.

This construction has two major disadvantages: Firstly the containers are not only used for „additionally organizing“ certain aspects of the event data, but indeed they even *constitute* the identity of the events, because every single „event“ — whatever an „event“ may be ! — is represented on the next lower level of data hierarchy as an *algebraic* term[4].

But in an algebraic setting, identically formed terms represent identical objects. That means that each data object of some given structure, e.g.

```
metric ( sum=(num=3,den=2)::(num=1,den=4),
         sub=(num=1,den=12) )
```

represents one and the same „entity“ of the „mathematical world“. So, if the container is supposed to have algebraic semantics, any event does only *exist* in the context of a

[4]The second disadvantage is the missing correspondence between these technically defined and required container objects, and the diverse and mostly non-disjoint groupings existing on the conceptual level of musical analysis.

E.g. if I just want to place events somewhere in time, in a Cageian manner, why do I have to put them into some „voice“ object, which does not correspond to any semantics of neither the model nor of its psycho-internal counterpart ?

Figure 4 Construction of Some „bar“ Objects

```
    ASPECT tempo=cwn.MM(72)
    ASPECT metrSpec=cwn.meter(3,4) ;
      ASPECT tpred=NULL, num=1  ;
      tpred = NEW bar ;
      ASPECT num= [(tpred>>num)+1] ; // quoted expr
      tpred = NEW bar ;
      tpred = NEW bar ;
    ASPECT metrSpec=cwn.meter(2,4) ;
      tpred = NEW bar ;
    ASPECT tempo=cwn.MM(80)
      tpred = NEW bar ;
      tpred = NEW bar ;
```

„container expression“, and cannot be referred to, identified or processed *per se*, i.e. out of this context.

Therefore some traditional meta-models introduce a *global* definition space for attaching „labels“ to certain events. But at this junction the basic semantic paradigm of algebraic data terms is broken, and the totally different world of co-algebraic semantics is introduced into the system, — a fact rarely noticed by the programmers of the meta-model[5].

1.7 Co-Algebraic Foundation of Models of Music

All these observations give reason to take the exact opposite viewpoint by making the *identity* the „first-class-resident“ in our architecture. So there is a kind of co-algebraic foundation of the semantics of each model: Initially every „event“ is given a single „identity“, which exists and is distinguishable independently from any „attribute“ the event subsequently assumes[6].

[5]This effect is comparable with that of giving „object“-semantics to an XML-document, as it is done in W3C-DOM: Then every paragraph does not carry only the information of its „contents“, the character data it consists of (i.e. its algebraic semantics, where two „identically looking“ paragraph objects are the *same* value object), but also the „knowledge“, that it is the first paragraph in the second chapter of the XY-book. This may seem quite harmless at a first glance, but it implies immediately, that even the smallest „colon“-character appearing somewhere in the text carries as its semantics the complete contents of the *whole document*, and additionally the locator information corresponding to this colon!

It seems clear that the same tiny „colon“ has to be treated in both cases quite differently!

[6]This principle also coincides with meta-meta-models from the area of Erkenntnißtheorie, where any transcendentally defined object is constituted by an *axiomatic* act of proposing some self-identity, cf. [3].

Figure 5 Mapping Rules from Special to Standard Types

```
DEF metricSpec:std.rat <== cwn.meter
 = std.rat(num, den)
ON ALL bar

DEF metricSpec:std.rat <== special_bulgarian
 =  std.rat (REDUCE (nums,_+_,0), den)
ON ALL bar

DEF metricSpec:std.rat <== special_for_composer_ml
 =  std.rat ( REDUCE ( MAP(sum, std.rat(num,den)),
                       _+_, 0),
              den)
    - std.rat (sub.num, sub.den)
```

2 Practical Design and Possible Architecture

While the considerations so far are on a somewhat abstact level of applied theory, they nevertheless aim at a very practical goal, namely the realization of some concrete editing and processing system. The fundamental outlines of its architecture are fixed by these analytical results.

The research activities of the authors' group concentrate currently on defining and designing a general-purpose meta-meta-language („`OPAL-2`"), onto which the execution of some m_0^2-model could be mapped. The results and efforts presented in this paper will serve as a case study, — a rather challenging case study, since the meta-meta-modeling of musical structures is among the most complex themes of data-modeling at all.

2.1 Architecture and Language Design

2.1.1 Corpus Definition Language

Let A be a finite, but dynamically growing set of aspect names, I be a set of pre-defined identities, X be a set of „tracks", S is the set of all algebraic signatures known to the system which are used to represent „simple values" (including some wrapper encoding for „identities"), and let V be the disjoint union of all free algebras induced by all signatures currently known to the system, ie. $V = \bigcup \mathcal{A}\,(\!|S|\!)$,

Then each „physical" model of a musical corpus can roughly be described as ...

$$K = (\,X \times I \times A\,) \nrightarrow (S \times V)$$
$$\textbf{where } \pi_2\, k(i,a,s) \in \mathcal{A}_i(\,\pi_1\, k(i,a,s)\,)$$

So for every combination of track object, event identity and aspect maximally one value can be stored in some arbitrarily chosen algebraic signature. It should be pointed out that the „track" is a pure practical concept.

As front-end representation we choose a scope-and-context-based style, where the „current settings" of aspect values are inherited from outer by inner scopes. The call of an object constructor „freezes" the current settings into the value fields of the newly created co-algebraic object, like in figure 4, which shows the creation of a sequence of bars.

2.1.2 Evaluation, Query Language

But the information contents visible to the user of the model, e.g. some tool application, is different, namely ...

$$K_R = (I \times A \times S \nrightarrow V) \cup (I \times A \nrightarrow S \times V)$$

Consequently queries can be executed for the current value of an aspect of a given identity, either by prescribing a certain signature into which the value is converted, or by accepting any signature, e.g. for further processing in a transparent/polymorphic way.

The tracks are totally invisible to the user operating on a model, but when opening the „view", a certain stack of tracks is configured, and the shadowing among these tracks rules the evaluation. Only the top-level track is writable.

2.1.3 Functional Closed-Ness

Every modern computer language provides some means to represent „programs" as values, e.g. by allowing function-valued expressions, parameters, results and variable contents. This principle enables us to *integrate* the „interpretation rules" of the meta-model *into* the model data.

Such interpretation rules can be associated with a certain meta-model, a certain model, a certain „use-case" of interpreting a model, or even with one single event, — all these very different applications are achieved using the same language means.

Operational semantics are always defined by recurring to existing processing algorithms, e.g. from a run-time library, and by calling, parameterizing and combining them to realize the calculation according to the intended semantics.

In our example, figure 5 defines one mapping for each new introduced algebraic type for metrical specification into the standard type of (positive) rational numbers. By adding these translation rules, all „library" functions which request a value for `.metricSpec` in this standard format also work on our newly defined and freshly invented formats. E.g. the „`LAW`" established in figure 3, — whereby each bar has to be „complete", except the very first and the very last, — *automatically* works for *all* types of metrical specification the user wants to add to the meta-model.

References

[1] Gerd Castan. (survey website on musical notation etc.). 2002, `http://www.music-notation.info`.

[2] Markus Lepper. *Komposition als Disposition von Datentransformation und Sprachdesign.* Number 14 in Folkwang-Texte. Essen, 1999.

[3] Markus Lepper. Modeling Music Using XML — Some Basic Considerations. In *Proceedings of the MAX 2002, First International Conference on Musical Application using XML*, pages 30–37. IEEE CS / TC on Computer Generated Music, 2002.