

Cognition, 4(1):99–122.

Taube, H. 1993. “Stella: Persistent score representation and score editing in common music.” *Computer Music Journal*, 17(4):38–50.

Widmer, G. 1992. “A knowledge intensive approach to machine learning in tonal music.” In M. Balaban, K. Ebcioglu, and O. Laske, editors, *Understanding Music with AI: Perspectives on Music Cognition*, pages 490–507. MIT–AAAI Press.

Winograd, T. 1968. “Linguistics and the computer analysis of tonal harmony.” *Journal of Music Theory*, 12:2–49.

interchange format.” In *International Computer Music Conference*, pages 106–109. San Francisco: International Computer Music Association.

Rodet, A. X., and P. Cointe. 1984. “Formes: Composition and scheduling of processes.” *Computer Music Journal*, 8(3):32–50.

Roads, C. 1979. “Grammars as representations for music.” *Computer Music Journal*, 3(1):48–56.

Roads, C. 1985. “Research in music and artificial intelligence.” *ACM Computing Surveys*, 17(2):163–190.

Rothgeb, J. E. 1968. *Harmonizing the Unfigured Bass: A Computational Study*. PhD thesis, Yale University.

Schottstaedt, B. 1983. “Pla: A composer’s idea of a language.” *Computer Music Journal*, 7(1):11–20.

Shieber, S. M. 1986. *An Introduction to Unification-Based Approaches to Grammar*. CSLI Lecture Notes Series, Chicago University Press.

Shieber, S. M. 1992. S.M. Shieber. *Constraint-Based Grammar Formalisms*. The MIT Press, Cambridge MA, London England.

Sloan, D. 1993. “Aspects of music representation in hytime/smdl.” *Computer Music Journal*, 17(4):51–59.

Smaill, A., and G. Wiggins. 1994. “Hierarchical music representation for analysis and composition.” *Computers and the Humanities*. Forthcoming.

Smith, L. C. 1973. “Editing and printing music by computer.” *Journal of Music Theory*, 17(2).

Smoliar, S. W. 1976. “An approach to music theory through computational linguistics.” *Journal of Music Theory*, pages 105–131.

Smoliar, S. W. 1980. “A computer aid for schenkerian analysis.” *Computer Music Journal*, pages 41–59.

Sundberg, J., and B. Lindblom. 1976. “Generative theories in language and music descriptions.”

- Lerdahl, F., and R. Jackendoff. 1983. *A Generative Theory of Tonal Music*. The MIT Press, Cambridge, Mass.
- Lidov, D., and J. Gabura. 1973. "A melody writing algorithm using a formal language model." *Computer Studies in the Humanities and Verbal Behavior*, 4(3/4):138–148.
- Loy, G. 1985. "Musicians make a standard: The midi phenomenon." *Computer Music Journal*, 9(4):8–26.
- Minsky, M. 1980. "K-lines: A theory of memory." *Cognitive Science*, 4:117–133.
- Minsky, M. 1986. *The Society of Mind*. Simon and Schuster, New York, NY.
- Narmour, E. 1977. *Beyond Schenkerism: The Need for Alternatives in Music Analysis*. The University of Chicago Press, Chicago, IL.
- Oppenheim, D. V. 1987. "The p-g-g environment for music composition." In *International Computer Music Conference*, pages 40–48, Urbana, IL. San Francisco: International Computer Music Association.
- Oppenheim, D. V. 1989. "Dmix: An environment for composition." In *International Computer Music Conference*, pages 1–8, Columbus, Ohio. San Francisco: International Computer Music Association.
- Oppenheim, D. V. 1992. "Compositional tools for adding expression to music." In *International Computer Music Conference*, San-Jose, CA. San Francisco: International Computer Music Association.
- Orlarey, Y., D. Fober, S. Letz, and M. Bilton. 1994. "Lambda calculus and music calculi." In *International Computer Music Conference*. San Francisco: International Computer Music Association.
- Pope, S. T. 1992. "The interim dynapiano: An integrated computer tool and instrument for composers." *Computer Music Journal*, 16(3):73–91.
- Pope, S. T. 1992. "The smallmusic object hernel: A music representation, description language, and

- schenkerian analysis.” *Computers and the Humanities*, 10:21–32.
- Frankel, R. E., S.J. Rosenschein, and S.W. Smoliar. 1978. “Schenker’s theory of tonal music - its explication through computational processes.” *Int. J. Man-Machine Studies*, 10:121–138.
- Gomberg, D. 1977. “A computer-oriented system for music printing.” *Computers and the Humanities*, 11:63–80.
- Gourlay, J. 1986. “A language for music printing.” *Communications of the ACM*, 29:388–401.
- Hamel, K. 1987. “Issues in the design of music notation systems.” In *International Computer Music Conference*, pages 325–332. San Francisco: International Computer Music Association.
- Harel, D. 1988. “On visual formalisms.” *Communications of the ACM*, 31(5):514–529.
- Hacken, L., D. Blostein, and W. Walker. 1991. “Lime music notation software for the macintosh.” In *International Computer Music Conference*, pages 582–585. San Francisco: International Computer Music Association.
- Hacken, L., and D. Blostein. 1993. “The tilia music representation: Extensibility, abstraction, and notation contexts for the lime music editor.” *Computer Music Journal*, 17(3):43–58.
- Harris, M., A. Smaill, and G. Wiggins. 1991. “Representing music symbolically systems.” In *IX Colloquio di Informatica Musicalw*, pages 55–69, Venice, Associazione di Informatica Musical Italiana.
- Haus, G., and A. Sametti. 1994. “Modelling and generating musical scores by petri nets.” *Languages of Design*, 2(1):39–58.
- Johnson, M. 1988. *Attribute-Value Logic and the Theory of Grammar*. CSLI Lecture Notes Series, CSLI International, 1988.
- Kassler, M. 1977. “Explication of the middleground of schenker’s theory of tonality.” *Miscellanea Musicologica*, 9:72–81, 1977.
- Kirsh, D. 1991. “Today the earwig, tomorrow man?” *AIJ*, 47(1-3):161–184.
- Laske, O. 1975. “Introduction to a generative theory of music.” *Sonological Reports*, 16.

- Byrd, D. 1974. "A system for music printing by computer." *Computers and the Humanities*, 8.
- Byrd, D. 1977. "An integrated computer music software system." *Computer Music Journal*, 1(2):55–60.
- Camurri, A., M. Frixione, C. Innocenti, and R. Zaccaria. 1992. "A model of representation and communication of music and multimedia knowledge." In *ECAI-92*, pages 164–168.
- Carpenter, B. 1992. *The Logic of Typed Feature Structures*. Cambridge University Press.
- Cope, D. 1989. "Experiments in musical intelligence (emi): Non-linear linguistic-based composition." *INTERFACE, Special Issue on Models of Musical Communication and Cognition*, Leman, M., ed., 18(1-2):117–139.
- Cope, D. 1992. *Computers and Musical Style*. A-R Editions, Madison, WI.
- Courtot, F. 1992. "Logical representation and induction for computer assisted composition." In M. Balaban, K. Ebcioglu, and O. Laske, editors, *Understanding Music with AI: Perspectives on Music Cognition*, pages 156–181. AAAI-MIT Press.
- Dannenberg, R. B. 1989. "The canon score language." *Computer Music Journal*, 13(1):47–56.
- Dannenberg, R. B., P. McAvinney, and D. Rubine. 1986. "Arctic: A functional language for real time systems." *Computer Music Journal*, 10(4):67–78.
- Diener, G. 1989. "Ttrees: Tool for the compositional environment." *Computer Music Journal*, 13(2):77–85.
- Ebcioglu, K. 1986. "An expert system for chorale harmonization." In *AAAI-86*, pages 784–788, Philadelphia, PA.
- Erickson, R. F. 1975. "The darms project: A status report." *Computing and the Humanities*, 9(6):291–298.
- Field-Richards, H. S. 1993. "Cadenza: A music description language." *Computer Music Journal*, 17(4):60–72.
- Frankel, R. E., S.J. Rosenschein, and S.W. Smoliar. 1976. "A lisp-based system for the study of

Music Cognition, pages 11–13. MIT–AAAI Press.

Balaban, M., and M. Elhadad. 1995. “On the need for visual formalisms for music processing.” Technical Report TR 94-14, Department of Mathematics and Computer Science, Ben-Gurion University, Israel. Submitted.

Balaban, M., and N.V. Murray. 1993. “Interleaving time and structure.” Technical Report TR 93-14, Department of Mathematics and Computer Science, Ben-Gurion University, Israel, and Department of Computer Science, SUNYA.

Balaban, M., and Samoun C. 1993. “Hierarchy, time and inheritance in music modelling.” *Languages of Design*, 1(2):147–172.

Balaban, M., and S.E. Shimony. 1994. “Structured plans with sharing and repetition: Model and specification.” Technical report TR 94-11, Department of Mathematics and Computer Science, Ben-Gurion University, Beer Sheva, Israel. Submitted.

Barbar, K., M. Desainte-Catherine, and A. Miniussi. 1993. “The semantics of musical hierarchies.” *Computer Music Journal*, 17(4):30–37.

Bel, B. 1992. “Modelling improvisational and compositional processes.” *Languages of Design*, 1(1):11–26.

Bel, B. 1992. “Symbolic and sonic representations of time-object structures.” In M. Balaban, K. Ebcioglu, and O. Laske, editors, *Understanding Music with AI: Perspectives on Music Cognition*, pages 64–109. AAAI-MIT Press.

Bel, B., and J. Kippen. 1992. “Bol processor grammars.” In M. Balaban, K. Ebcioglu, and O. Laske, editors, *Understanding Music with AI: Perspectives on Music Cognition*, pages 366–401. AAAI-MIT Press.

Brooks, R. A. 1991. “Intelligence without representation.” *AIJ*, 47(1-3):139–159.

Buxton, W., W. Reeves, R. Baecker, and L. Mezei. 1978. “The use of hierarchy and instance in a data structure for computer music.” *Computer Music Journal*, 2(4):10–20.

We believe that our approach will lead to the design of well founded knowledge representation tools for music. Compositional and instructional environments built on this ground will be supported by a solid theory, that will account for their development. Systems developed under this methodology are no more black boxes. They can be incrementally developed, and their properties can be investigated. In particular, desirable features, such as *modularity*, the ability to describe *incomplete knowledge*, and *uniformity* of design, may be studied.

The relevance of our approach goes beyond music applications per se. Computer-music systems that are built on the basis of a solid theory, can be coherently embedded into multimedia environments. The richness and specialty of the music domain are likely to initiate new thinking and ideas, that will have impact on areas such as knowledge representation and planning in Artificial Intelligence, and on the design of visual formalisms, and man-machine interfaces in general.

Acknowledgements

The author is deeply indebted to Danny Oppenheim for numerous discussions about compositional environments, and to Michael El-Hadad for discussions concerning Constraint-Based Grammars, and visualization in computer systems. The continuous efforts of Eli Barzilay to produce a clean, correct, and open implementation, are invaluable. Many vague issues about music structures were clarified as a result.

References

- Anderson, D.P., and R. Kuivila. 1989. "Continuous abstractions for discrete event languages." *Computer Music Journal*, 13(3):11–23.
- Balaban, M. 1989. "Music structures: A temporal-hierarchical representation for music." *Musikometrika*, 2:1–50.
- Balaban, M. 1992. "Music structures: Interleaving the temporal and hierarchical aspects in music." In M. Balaban, K. Ebcioğlu, and O. Laske, editors, *Understanding Music with AI: Perspectives on*

Implementation

An implementation of a music processing tool that is based on the music structures approach is on the way. The system, called *BOOMS*, is designed and implemented by Eli Barzilay in CLOS (Common Lisp Object System). *BOOMS* manipulates objects that are associated with values, that are either music structures, or music structure operators. *BOOMS* is faithful to the intensional character of music structures, due to the separation between objects and their values, the manipulation of operators as values, and the delayed evaluation policy. A memoization technique is used to prevent repeated computations. The *class* and *method* data structuring facilities of CLOS are used as an ad-hoc account for additional information about music structures.

The implemented graphical interface is still rather simplistic: *Operator-argument* edges are used as a single structuring visualization. For example, there is, yet, no visual distinction between explicit VMSs, where the **inside** topological relation can explicitly convey the temporal structure, to implicit VMSs, where operator-argument edges seem to be a good choice.

We envision a continuous, incremental implementation of the system, along with further developments of the representation of *OOMPs* and the visual formalism. A detailed description of the system, its architecture, and application will be described in a follow up paper.

Conclusion

In this paper we described the systematic development of the music structures approach, which consists of the ontological level of structured music pieces and object-oriented music pieces, the representation level of symbolic and visual music structures, and the implementation level in the *BOOMS* system. The major goal of our approach is to lay a basis for a methodology of engineering computer-music systems. We claim that this approach is essential in order to keep systems reliable and manageable, and in order to deepen our understanding of the capabilities and limitations of a computational account for music.

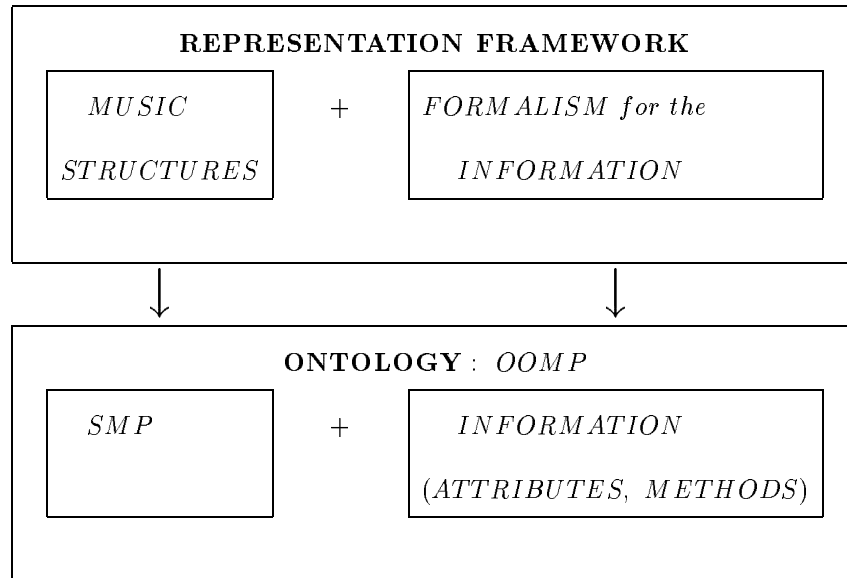


Figure 15: The Structure of the representation Framework for the *OOMP* Ontology

Beyond Music Structures – A First Step

Music Structures is a framework for representing and processing structured music pieces. In order to account for *OOMP*s we need a framework for describing the information associated with structured music pieces. Graphically, the relationship of the missing part in the representation framework, to the existing formalism and ontology, is described in Figure 15.

A representation approach that comes to mind for the missing slot in Figure 15 is the *Constraint-Based Grammar Formalisms* that derive from the *Functional Unification Grammars* approach (Shieber 1986, 1992; Johnson 1988; Carpenter 1992). This representation school is designed for describing information about linguistic strings. The information is assumed to be *modular, partial, hierarchical*, and enable *unification (equationality)* among its components. The information is termed *feature structures*, and the formalisms are *logics of feature structures*. Constraint-based formalisms seem appropriate for describing the *OOMP* ontology, since *OOMP*s remind feature structures (see, for example, Figure 9). The development of a formalism for describing *OOMP*s is a subject for future research.

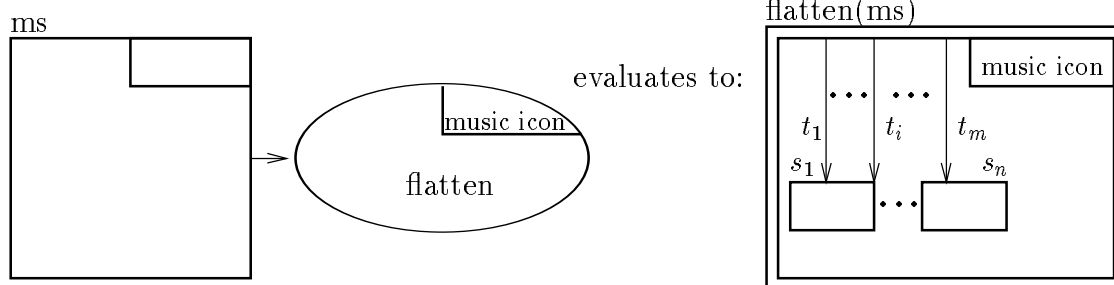
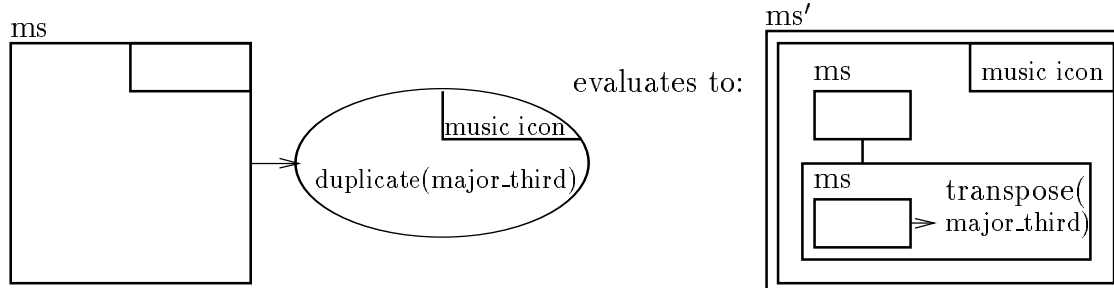
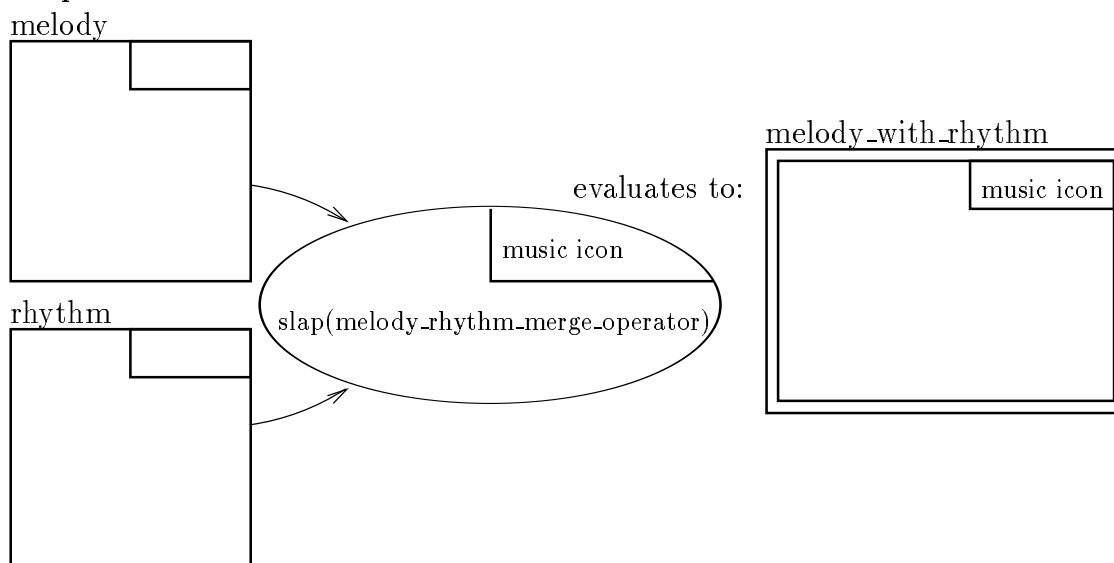
a. **flatten**b. **duplicate**c. **slap**

Figure 14: Implicit visual music structures

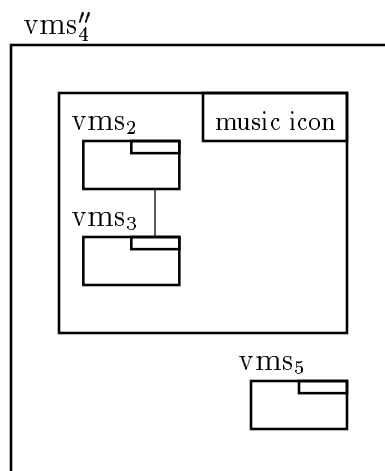


Figure 13: An incomplete visual music structure with temporal relationships between components **flatten**, **duplicate**, and **slap**. The left side in each of parts (a), (b), (c) describes the implicit *VMS*, and the right side describes its evaluation into an explicit *VMS*.

Towards a Visual Music Structures Formalism

Harel, in (1988), suggests higraphs as a general visual formalism that uses the complex structure of nodes as a visual means for generating complex structures. Observing the *VMS*s described above, we see that all are higraphs, with different kinds (sorts) of nodes and edges. We believe that higraphs can account for the *VMS*s intuition, laid above, since they are simple, formal, general, and Graphics independent. In (Balaban and Elhadad 1995) we discuss the need for a visual representation level in music systems, that is separated from the KR level, and independent from the actual graphics. In (Orlarey, Fober, Letz, and Bilton 1994), the operation of *visual abstraction* is introduced as an essential operation in music systems. It seems that achieving an adequate account for human aspects of visualization is a major issue.

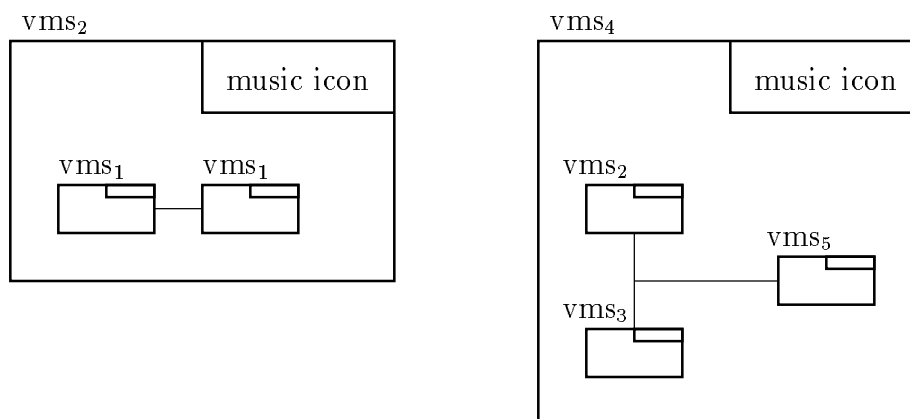


Figure 11: Visual representations for ms_2 and ms_4

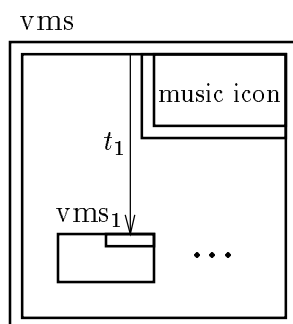


Figure 12: A complete visual music structure

displacement. Figure 13 is a suggestion for a visual expression for ms_4'' . Note that unlike in vms_4 from Figure 11, the simultaneous concatenation of vms_2 and vms_3 becomes a nameless VMS of its own.

Implicit visual music structures, and music structures' abstraction: Implicit music structures can be viewed as concrete applications of music structure operators to music structures and to other arguments. Eli Barzilay (who implements the BOOMS system, see below), suggests to visualize an operator application by a special visual form, and use directed edges as a visual expression for operator-argument relation. Figure 14 describes this suggestion with three operators:

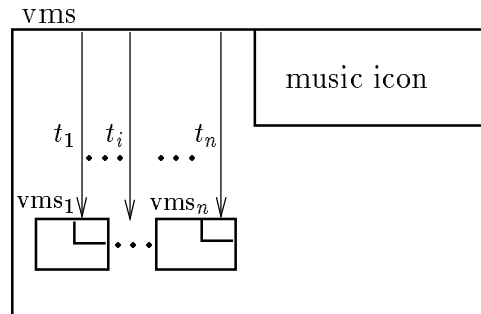


Figure 10: An explicit visual music structure

language we use DMIX. As a source of inspiration for a desired visual language we use (Harel 1988). We now present, in an intuitive way, our ideas concerning *VMS*s. We classify the presentation by the different kinds of music structures described above, since our aim is to provide a visual account for all kinds.

Explicit visual music structures: In explicit music structures the intended temporal-hierarchical structure is explicitly specified. One option for a visual expression for a temporal hierarchy is described in Figure 10. Suggestions for visual ms_2 and ms_4 are described in Figure 11. This visualization of ms_2 applies only to its explicit, extensional definition. Its intensional definition as $\mathbf{seq_repeat}(ms_1)$ is implicit, and cannot exploit the horizontal time axis as a visualization of the temporal structure.

Partial specification in visual music structures: The visual expression for a null value can be some visual “emptiness”, like \blacksquare . For variables we will need an identification means (a name) next to the “black hole”.

Complete/incomplete visual music structures: Completeness of a music structure can be visualized by a double frame line, as in Figure 12. Incomplete (explicit) music structures, that result from specification of temporal relations between components, can be visualized by using the horizontal axis as a time axis, and placing the components in a way that conveys their temporal

The definition $ms_2 := ms_1 - ms_1$ can be replaced by $ms_2 := \mathbf{seq_repeat}(ms_1)$, which happens to be the correct description for mp_2 from Figure 5. The new definition abstracts the *intension* of sequential repetition as the major characteristic of ms_2 . Once ms_2 is defined using the **seq_repeat** operator, its structure is defined as a sequential repetition of the same music piece. Any change in ms_1 must be reflected in both components of ms_2 . With the original *extensional* definition, ms_2 is a sequence of two pieces that just happen to be equal. It may be possible to apply independent changes to its components, and get $ms_2 := ms_3 - ms_1$. Note, however, that under the new intensional definition, ms_2 turns from an explicit music structure into an implicit one. This change affects its visualization.

The structure abstracted in an operator can be applied to different arguments. For example, the structure of sequential repetition, embedded in the **seq_repeat** operator, can be used to generate complex sequential repetitions, as in:

seq_repeat($(ms_2 | ms_2)$), *that evaluates into* $(ms_2 | ms_2) - (ms_2 | ms_2)$

Examples of possible operators:

round(ms, t) := $(ms@0 \ ms@t)$.

stretch($ms, time-interval$).

slap($ms_1, ms_2, slap-operator$).

Music Structures – a Plan for a Visual Version

A visual representation of structured music pieces must account for their essential hierarchical structure, and its interleaving with temporal relativization. We need visual primitives, and visual ways for combining these primitives into Visual Music Structures (*VMSs*). We would like to have visual expressions for the full symbolic music structures language laid above, including relationships, transformations and abstractions. As a source of inspiration for a musically desired visual

Complete/incomplete music structures: A complete music structure is one, all of whose components are “known” (the number of arguments of the specification operator is known). All previous music structures are complete. Even in the partially specified ms_3'' , it was known that there are two identical components, occurring at partially specified times T_1 and $\mathbf{duration}(Ms)/2$. Implicit music structures are complete. An incomplete music structure is needed to allow for unknown components or to specify incomplete temporal relationship between components.

Unknown components: Suppose that in ms_3'' , above we wish to allow for extra components.

That is, in addition to the two repeated occurrences of Ms , there may possibly be other components:

$$ms_3''' := (Ms@T_1 \quad Ms@mathbf{duration}(Ms)/2 \quad \bullet \quad Rest)$$

$Rest$ stands for all unknown components. “ \bullet ” is the operator that “combines” $Rest$ with the known part of ms_3''' . This kind of incompleteness applies to any recursively defined operator.

Incomplete temporal relationships between components: Suppose that in ms_4 it is just known that the $(ms_2|ms_3)$ component starts before the ms_5 component. It is not known when either component starts, or whether there are other components at all:

$$ms_4'' := \mathbf{before}((ms_2|ms_3), ms_5)$$

This kind was not included in the already published Music Structures language (Balaban 1989, 1992). It is currently being developed for the purpose of plans representation in AI (Balaban and Shimony 1994).

Abstraction in music structures – Music structure operators

Music structures can be abstracted to yield operators. For example, the music structure $ms_1 - ms_1$ can be abstracted into an operator **seq_repeat**, for sequential repetition:

$$\mathbf{seq_repeat}(ms) := ms - ms$$

stretch(ms , time-interval-expression).

duplicate(ms , music-interval-expression).

slap(ms , $slap_ms$, $slap_operator$) – A special case is:

slap($rhythmic_ms$, $slap_rhythm$, $rhythm_slap_operator$), as in:

slap($\downarrow \downarrow \circ$, $\downarrow \downarrow$, $rhythm_divide_operator$)
which yields $\downarrow \downarrow \downarrow \downarrow \downarrow \downarrow$

or in: **slap**($rhythmless_melody$, $slap_rhythm$, $melody_rhythm_merge_operator$),

which merges the *melody* with the given *rhythm*. Slappability as an essential composition operation was first suggested by D. Oppenheim, and introduced into his DMIX system, as an operation on the visual representation of music pieces.

Full/partial specification (ground/non-ground): A fully specified (ground) music structure is one that all of its parts are fully specified, as in all of the above examples. A partially specified (non-ground) music structure is one that information is missing about some of its components. For example:

Suppose that in ms_3 we do not know when the first occurrence of ms_1 starts. That is, instead of the timing 0 we have some “null value”, or a variable T_1 (names starting with upper case letters are variables):

$$ms'_3 := (ms_1 @ T_1 \quad ms_1 @ \mathbf{duration}(ms_1) / 2)$$

Suppose that in ms_4 the identity of the piece following ($ms_2 | ms_3$) is not known:

$$ms'_4 := ((ms_2 | ms_3) - Ms)$$

Suppose that in ms_3 also the identity of the subparts of ms_3 is not known, but it is still known that it is the same subpart that is repeated:

$$ms''_3 := (Ms @ T_1 \quad Ms @ \mathbf{duration}(Ms) / 2)$$

Explicit music structures: An explicit music structure is in place when a musician explicitly specifies the timing of sub pieces within a piece. For example, explicit specifications of the structured music pieces mp_2 , mp_3 , and mp_4 from Figures 5, 6, and 7, are:

$$\begin{aligned}
 ms_2 &:= ms_1 - ms_1 && \text{"}ms_1 \text{ plays twice, sequentially.} \text{"} \\
 ms_3 &:= (ms_1 @ 0 \quad ms_1 @ \text{duration}(ms_1)/2) && \text{"}ms_1 \text{ plays twice; first at time 0 and second at the mid} \\
 &&& \text{point of the first occurrence.} \text{"} \\
 ms_4 &:= ((ms_2 | ms_3) - ms_5) && \text{"}ms_2, ms_3 \text{ are played simultaneously, while } ms_5 \\
 &&& \text{follows sequentially.} \text{"}
 \end{aligned}$$

Explicit specification of some known music forms:

$$Chord : \quad \prod_{i=1}^n sound_i$$

$$Harmonic \ sentence : \quad \prod_{i=1}^n chord_i$$

$$Melody : \quad \prod_{i=1}^n sound_i$$

$$Four \ voice \ choir : \quad \prod_{i=1}^4 melody_i$$

For simplicity we ignore, in the verbal explanations below, the distinction between the syntactic entities, i.e., music structures, to the semantic entities, i.e., structured music pieces. We refer to music structures as if they are also semantic objects.

Implicit music structures: An implicit music structure is one that describes a structured music piece via a relationship with other structured music pieces, or via a transformation applied to other structured music pieces. For example:

flatten(ms), that describes the flat form of ms .

transpose(ms , music-interval-expression).

on a given one (similar to attribute grammars, and less typical in object oriented environments). We also noted that some properties have default behavior.

The investigation of OOMPs is still in its beginning. We believe that the information extending SMPs should be added, gradually, possibly classified, and be structured by aspects.

Representation Framework: Music Structures

The representation framework is the dominant part in a music processing system. The representation is just all that the implemented system has in: It is the source of power and weakness of a system. A good representation for structured music pieces need to capture temporal hierarchies, and be sensitive to available information and desired medium of expression.

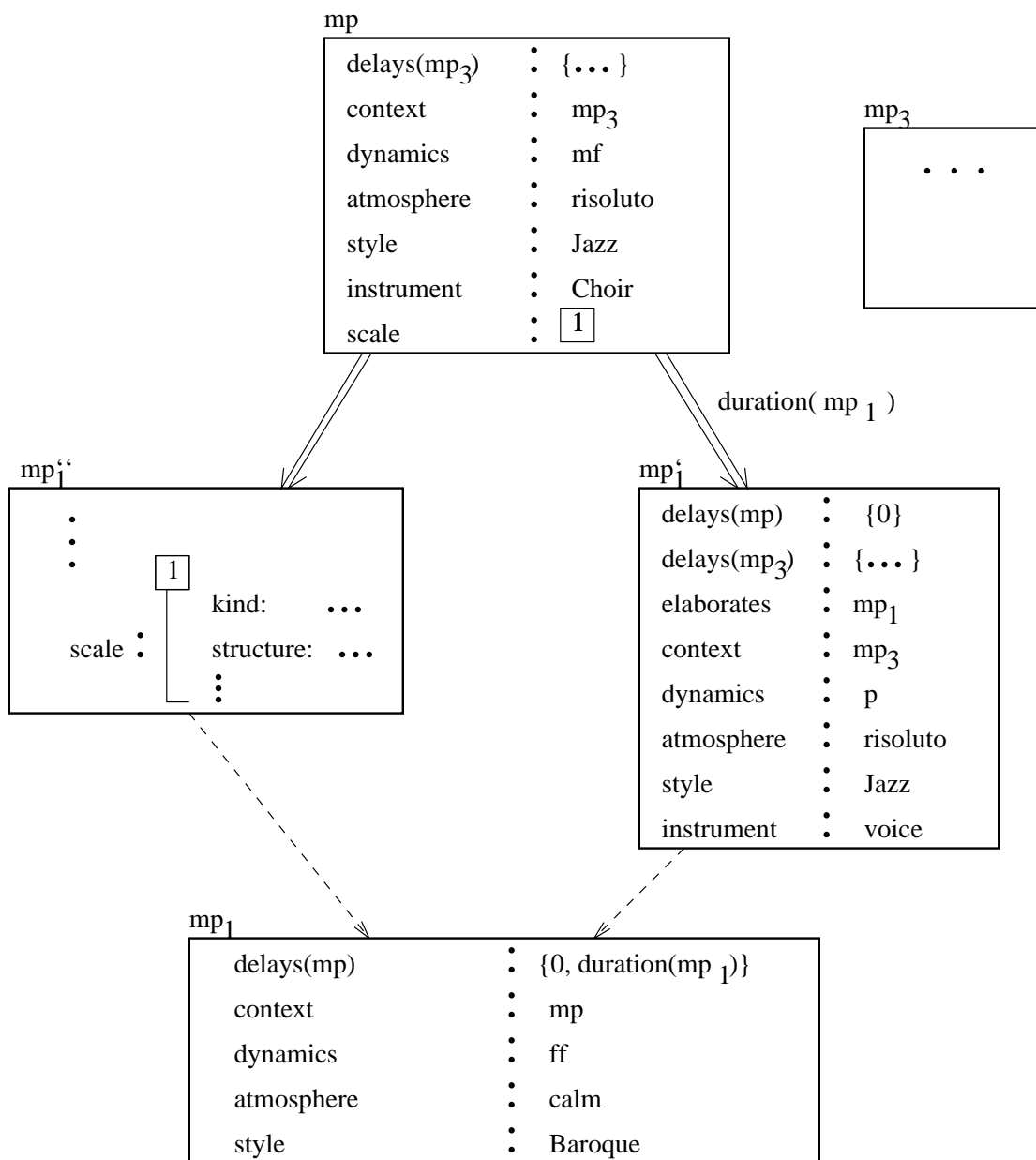
The available information about the ontology is determined by the application. We set our criteria to match the needs of composition environments, as we think that these are most demanding, and are provided with least information. We distinguish six kinds of information that are elaborated below: *Explicit* or *implicit*, *fully-specified (ground)* or *partially-specified (non-ground)*, and *complete* or *incomplete*.

For musicians, a linguistic/symbolic medium seems out of question. It is no coincidence that traditional music notation is graphic, and has two dimensions: Horizontal for time, and vertical for pitch. A visual/graphical medium seems like a better choice, and the success of DMIX is an indication in that direction. The music structures framework indeed started as a symbolic representation, but is now being developed into a visual one.

In the following we elaborate the various kinds of music structures, based on the amount of information they carry. The symbolic version is presented first, followed by a tentative plan for a visual one.

Music Structures – a Symbolic Version

There are six kinds of expressions, that capture the six kinds of information about SMPs.



- A double arrow denotes the *component* relation (mp_1'' is a component, at time 0, of mp_2).
- A dashed arrow denotes the *elaboration* relation (mp_1'' is an elaboration of mp_1).
- A direct reference to an object is captured by using its name. For example, the value of the *context* attribute of mp_1 is mp ; the value of the *scale* attribute of mp is (intensionally) the value of that attribute of mp_1'' (the latter is marked $\boxed{1}$, for the purpose of this reference).

Figure 9: An OOMP named mp

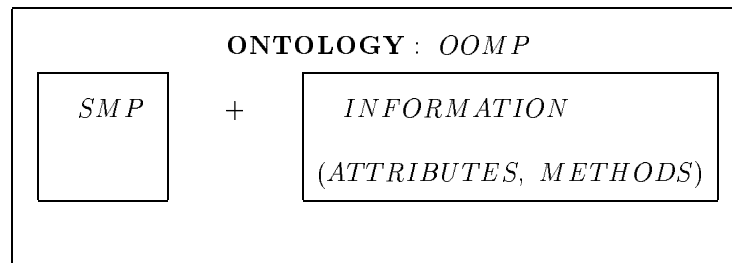


Figure 8: The Structure of the OOMPs Ontology

In order to respect the modular hierarchical nature of structured music pieces, we assumed that information is associated, in an independent way, with structured music pieces, and is accessed via labels, called *attributes*. In other words, we assumed the existence of partial functions, called *attributes*, that assign “information” to structured music pieces. The structure of the information being assigned was left unspecified. Following the object-oriented approach in knowledge representation, databases and programming, attributes were allowed to take also extra parameters. Such attributes were called *methods*. The resulting ontology was called *Object Oriented Music Pieces (OOMP)*. It is described in Figure 8.

Figure 9 provides a visual description for an OOMP called *mp*. The components of *mp* are two elaborations of mp_1 , called mp'_1 and mp''_1 , that extend mp_1 with properties regarding other music aspects. The OOMPs mp'_1 and mp''_1 inherit some properties of mp_1 but change others, and have new properties. The new *mp* is a sequential playing of mp'_1 and mp''_1 . Note that *mp* is not an elaboration of mp_2 , since it is not a sequential repetition of a single music piece, but is a sequence of two pieces that inherit the structure of mp_1 .

We investigated forms of information flow between entities (SMPs and their elaborations). The overall conclusion was that information flow among music entities is far more complex than standard inheritance relations in object oriented environments, or than attributes’ computation in attribute grammars. There is no fixed direction of flow, and (unlike in attribute grammars) it is not a self property of the information. Information flow may involve computation of new information based

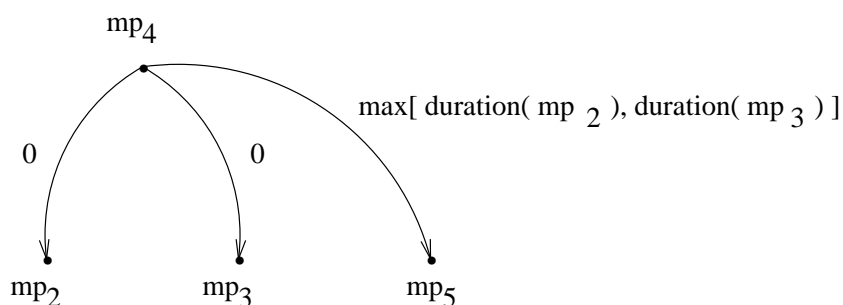


Figure 7: The structured music piece mp_4

graphically described in Figure 7. The SMP mp_4 has three components, mp_2 , mp_3 , and mp_5 . The piece starts simultaneously with mp_2 and mp_3 , to be followed, when both mp_2 and mp_3 end (time point $\max[\text{duration}(mp_2), \text{duration}(mp_3)]$), by mp_5 .

It is important to understand that the SMPs are the hierarchical objects described above. Each SMP can be *flattened*, yielding a stream of sounds that can be actually performed. Clearly we may have many different SMPs sharing the same flattened form. In this sense, SMPs capture hierarchy and time in an essential way: Different hierarchical views of the “same” music give rise to different SMPs. In particular, the so called “ambiguity” phenomenon, which in traditional formal grammar theory is considered problematic, becomes the standard situation. SMPs are more powerful than plain streams since they capture hierarchy and time, and can produce the intended streams of sounds.

SMPs carry an intensional flavor. The SMP mp_2 , for example, has the structure of a sequential repetition of mp_1 , and this structure is invariant under changes to mp_1 . That is, any change to mp_1 affects the two components of mp_2 , and possibly the point of repetition as well.

Beyond Structured Music Pieces – A First Step

In (Balaban and Samoun 1983) a first step towards extending the domain of SMPs was introduced. Our purpose was to associate structured music pieces with information of any kind, and investigate inheritance relations inspired by this information, between an SMP and its components.

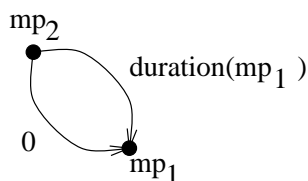


Figure 5: The structured music piece mp_2

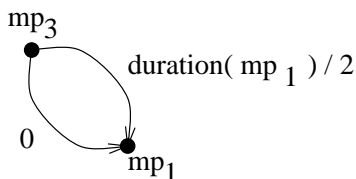


Figure 6: The structured music piece mp_3

precise subsets of music, e.g., all Mozart style pieces, all tonal music pieces, etc., we may be able to characterize larger sets, that are more loosely defined. We now describe this world in general terms, since we just wish to give the flavour of our approach, without getting into a detailed account.

Structured music pieces are hierarchical objects built along a time line. Suppose that a composer has a piece mp_1 , and s/he wishes to compose a new one, that will be a sequential repetition of mp_1 . Then, the new piece, mp_2 , consists of two occurrences of mp_1 , played at times 0 and **duration**(mp_1). The composite piece mp_2 can be drawn as the Directed Acyclic Graph (DAG) in Figure 5. The nodes of the graph stand for music pieces. An edge $mp_2 \xrightarrow{t} mp_1$ states that mp_1 is a component of mp_2 , and it plays at time t relative to the beginning of mp_2 . Figure 5 shows that mp_2 has two components, both identical to mp_1 . The first occurrence of mp_1 plays at the beginning (time 0), and the second occurrence is an immediate repetition (time point **duration**(mp_1), i.e., when the first occurrence of mp_1 is finished). Suppose now that the repetition is not sequential, but the second occurrence of mp_1 starts at the middle time point of the first occurrence. Then the new composite piece mp_3 still has two components, identical to mp_1 , but they play at times 0 and **duration**(mp_1)/2. The graphical description is given in Figure 6. A new piece, mp_4 , that consists of simultaneous occurrences of mp_2 and mp_3 , followed, let us say sequentially, by a piece mp_5 , is

The Music Structures Approach

An implemented music system can either directly manipulate streams of physical sounds, or manipulate a music ontology that singles out concepts and structures in the music. Systems of the first kind act in a low level, where they manipulate direct entities of the real world problem. It is not clear how they can be extended to higher conceptual levels (Brooks 1991; Kirsh 1991). Systems of the second kind manipulate representations of conceptual music abstractions. Interestingly, even such systems are grounded within the real world, since music is performed in real time (this property might be found attractive for high level planners). Hence, the representation level is not detached from its real source problem.

The music structures approach follows the conceptual representation school. In this section we explain its methodology. There are three stages: Definition of the ontology, development of representation frameworks, and implementation, using high level software engineering tools.

Ontology: Structured Music Pieces

In music, a complete formal account for the semantical phenomenon is not attainable. On one side, we cannot adopt the suggestion of mental dispositions, since we don't know what they are, and how to manipulate them. On the other side, we cannot assume that the real world music we process is just streams of music events/sounds. We look for objects that include more music characteristics than low level streams of sounds, but are not as vague as mental dispositions. The music world described by a system is always partial. Hence, the ontology must be *extensible*. The ontology reflects a deliberate decision as to which parts of the real phenomenon are captured.

Structured Music Pieces (SMPs) are music objects restricted to capture *Time* and *hierarchy* alone. It is important to emphasize that the intent is to characterize the notion of a *music piece*, restricted to these aspects. The assumption is that while we are, probably, unable to characterize

category we count, for example, the system of (Camurri, Frixione, Innocenti, and Zaccaria 1992), that is based on the KL-1 representation model, formal grammar based systems like (Bel and Kippen 1992), attribute grammar based systems like (Barbar, Desainte-Catherine, and Miniussi 1993), and learning applications like (Widmer 1992). Systems in this category have to cope with the limitations of the formalisms they use, as these formalisms were not designed for music processing. For example, KL-1 is a model of classification, that accounts for analytic definitions of concepts and binary relations. Certainly, there are music operations, transformations, and characterizations that do not fit into this framework. Formal grammars were developed for the purpose of distinguishing well formed strings, on the basis of unambiguous derivations. But in music, the existence of multiple views of the same music is a source of richness, and not a problem to cope with. Attribute grammars assume that the attributes attached to different non-terminals used in the parsing of a string, have a fixed direction of computation. In composition, as in most design tasks, this assumption is not realistic. Moreover, all traditional grammar models assume complete global knowledge of the task, an assumption that does not fit the partial, modular nature of music making.

Among systems that build their own computational basis we count software engineering tools for music, like CHARM (Harris, Smaill, and Wiggins 1991; Smaill and Wiggins 1994) and (Diener 1989), and formal models for music processing. Examples for the later kind is the system of music types and operations of (Courtot 1992), and the *music structures* representation of (Balaban 1992; Balaban and Samoun 1993), that suggests an account for the aspects of *Time*, *Hierarchy*, and *Inheritance*. Systems in this category are not theories of music, but develop computer tools for music processing. That is, they provide frameworks within which a composer, for example, can define her/his music operations and characterizations, at different levels of abstraction.

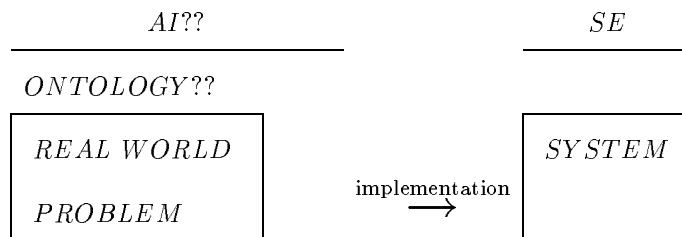


Figure 4: Stages in the Development of Existing Music Processing Systems

of computer music tools.

Systems that implement specific theories of music, like the generative grammar based systems of (Lerdahl and Jackendoff 1983; Laske 1975; Lidov and Gabura 1973; Sundberg and Lindblom 1976; Rothgeb68), or the Schenkerian theory based systems of (Kassler 1977; Narmour 1977; Frankel, Rosenschein, Smoliar 1976, 1978; Smoliar 1980), or the distinguished systemic grammar based system of (Winograd 1968), are not open ended. Such systems are based on tight models, and did not develop into music processing tools.

Computer music tools intended to support a variety of music tasks, have opposite nature and goals. Such systems are evaluated by the amount of support they provide users with, and by their reliability and open endedness. Under this category we count systems with impressive music processing performance, such as (Buxton, Reeves, Baecker, and Mezei 1978; Rodet and Cointe 1984; Dannenberg, McAvinney, and Rubine 1986; Dannenberg 1989; Pope 1992a; Ebcioğlu 1986; Cope 1989, 1992; Oppenheim 1989, 1992; Haus and Sametti 1994). All of these systems have deep musical insights embedded in them. The assumptions underlying several systems can be abstracted away, and may improve the engineering of computer music environments. For example, DMIX emphasizes the need for visualization of high order operations. However, the essential formal representation level required by AI systems is missing. These systems do not provide a stable basis for the design of computer music tools. The visual picture is given in Figure 4.

Among systems that use representation or software engineering tools we distinguish those that use “off-the-shelf” tools from those that build their own custom tailored frameworks. In the first

tool poses a major problem. First, let us say what the real-world music is certainly not: It is not a graphical representation. Hence, languages like DARMS (Erickson 1975), and (Smith 1973; Byrd 1974, 1977; Gomberg 1977), and more recent descriptions of music scores (Gourlay 1986; Hamel 1987; Hacken, Blostein, and Walker 1991; Field-Richards 1993; Sloan 1993) are not of our interest here. Clearly, a music processing system does not denote a picture of music.

Most existing systems do not explicitly characterize their music ontology. Grammar based systems, such as (Smoliar 1976; Laske 1975; Lerdahl and Jackendoff 1983; Bel 1992a, 1992b; Roads 1979, 1985; Dannenberg, McAvinney, and D. Rubine 1986; Dannenberg 1989; Anderson and Kuivila 1989; Sloan 1993), manipulate strings of *sounds/events*. Other systems, like (Loy 1985; Schottstaedt 1983; Hacken and Blostein 1993), process multi-dimensional arrays of *sounds/events*, where the most popular dimensions are *pitch* and *time*. Object based systems, such as (Rodet and Cointe 1984; Pope 1992a, 1992b; Taube 1993; Oppenheim 1987, 1989, 1992), view their music ontology via the *object-oriented* approach glasses. Hierarchy based systems, like (Buxton, Reeves, Baecker, and Mezei 1978; Balaban 1992; Balaban and Samoun 1993; Barbar, Desainte-Catherine, and Miniussi 1993; Smaill and Wiggins 1994), process structures of music entities. Minsky, in (Minsky 1980, 1986), suggested that music systems should process *mental dispositions*. This suggestion is, probably, the most accurate, but is difficult from the ontology-representation viewpoint.

An essential property of the real world problem in music is that it is not conceptualizable in its totality. The ontology underlying a music processing system is always partial. Moreover, ontologies vary by the viewpoint of the application. Hence, music tools should admit incremental development.

Existing Systems, Approaches, Theories, and Models in Music Processing

Computer music systems can be classified by their goals, demonstrated performance, and underlying computer tools. The classification is neither exhaustive nor mutually exclusive, but it provides an idea about the relevance of deep insight, in music and in design of computer systems, for the success

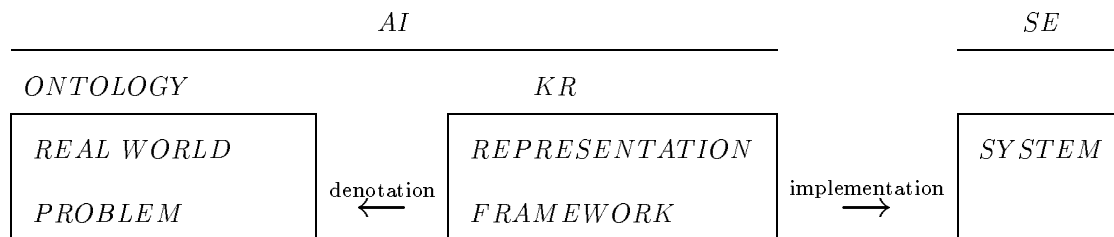


Figure 2: The AI stages in System Development

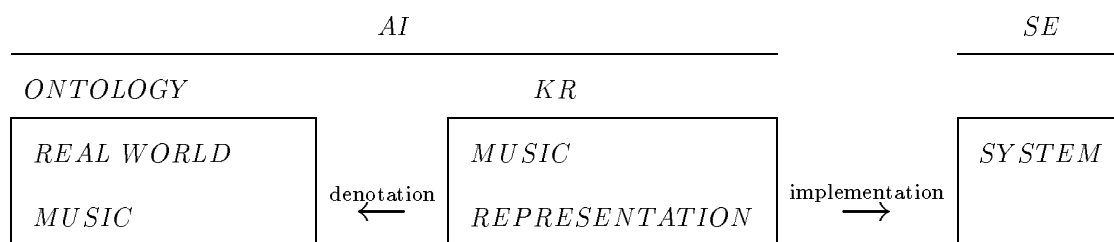


Figure 3: Stages in the Development of Music Processing Tools

A good KR framework should respect the requirements of the real world problem, should respect the characteristics of the available information about the problem, and should take into account users demands, such as convenient media of expression. For example, conventional parsing tools, used in syntax directed compilers, assume the existence of a grammar, that provides a complete characterization of a set of strings, and strives for a unique parsing (acceptance decision) for each string in the set. Such grammars are, for example, *context free grammars* or *attribute grammars* used in compiler construction. Hence, traditional formal grammars may not be a good choice for representing a world of music pieces, a major property of which is the existence of multiple views for a piece, and partiality of available information. The combined visual picture of the AI and the SE approaches is described in Figure 2.

Implications for Music Processing – What is a Music Ontology

Music processing tools that respect the software engineering and the artificial intelligence experience can be visualized as in Figure 3. Yet in music the nature of “real-world-music” denoted by a music

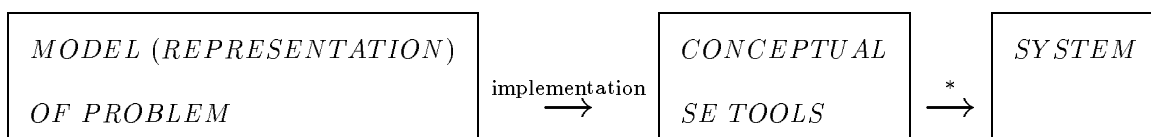


Figure 1: The Software Engineering Process for System Development

output effects. Consequently, tools built on formal grounds have the prospect of becoming general, objective tools, used by those users that accept the underlying assumptions. Their expected life cycle is long since they lend themselves to incremental formulation and construction. We hope that with such tools we can get a better understanding of music problems, and in particular, of the limits of analytic processing of music.

The Software Engineering (SE) and the Artificial Intelligence (AI) Views

Software engineering provides criteria and tools for designing systems in a way that allows for *incremental evolution* of systems design, leads to manageable systems, and extends their life time. Major criteria for high quality in software engineering are *abstraction*, and *modularity*. The development process of a system can be viewed as a sequence of stages, starting from a conceptual model of the problem, and ending in a concrete computer system. In each stage, the abstract tools of the preceding stage are implemented in terms of the tools of the next stage. This process can be visualized as in Figure 1.

AI augments the software engineering view with the connection to the real world. Problems attacked by AI systems are rooted in the real world, and AI systems are descriptions of such problems. The real problem handled by the system is singled out as the *ontology* of the problem. The ontology circumscribes the entities, operations, relations, and structures, from which the problem consists. The ontology provides for the *detachment* of the problem from the rest of the world it is part of. The essence of an AI system is a (possibly hybrid) Knowledge Representation (KR) framework, that denotes the ontology. The KR framework, its components, their interaction, and their processing procedures, are evaluated and justified by their direct reference to the ontology.

Introduction

Music processing tools are systems built for purposes such as composition, instruction, research in cognitive music activities, music analysis, and commercial needs. The latter can be viewed as engineering tools, built following clearly specified input-output requirements. The scope of application of such tools is, usually, deliberately restricted, so to allow for efficient performance, and the life cycle is relatively short. The non-commercial trends in music processing have opposite characterization: Broad scope, efficiency is less important, but long life cycle is a must.

In this paper we discuss the nature, importance, feasibility, and need for formal development of research tools in music. We claim that a systematic development of knowledge representation frameworks for music is essential for obtaining manageable, reliable, generally accepted music processing tools, such as composition systems. Such systems must be incrementally developed, as their specifications are always partial. We demonstrate our claim using the *music structures* approach, starting with an *ontology of music objects*, and ending with symbolic and visual representation frameworks, and their implementations.

In the following section we discuss the nature of knowledge representation, and the need for it. In the core section of this article we lay the principles for knowledge representation in music processing, and demonstrate its feasibility using the music structures framework (Balaban 1992; Balaban and Murray 1993). In the concluding section we discuss the expected benefits of our approach.

Knowledge Representation in Music Processing

Knowledge representation in music is concerned with the design and realization of music tools, based on well defined formalisms of representation and implementation. The main advantage of such tools is that they can be *observed*, their algorithms can be verified, and their mode of operation can be studied, evaluated, modified, and extended. They are not *black boxes*, used only for their input-

The Music Structures Approach in Knowledge Representation for Music Processing*

Mira Balaban

Dept. of Mathematics and Computer Science

Ben-Gurion University of the Negev

P.O.B. 653, Beer-Sheva 84105, Israel

mira@cs.bgu.ac.il

(972)-7-461622

Abstract

A framework for formal design and construction of music systems is introduced. It is demonstrated using the *music structures* approach, starting with an *ontology of music objects*, and ending with symbolic and visual representation frameworks, and their implementations. A visual formalism based on the music structures approach is introduced. We claim that a systematic development of knowledge representation frameworks for music is essential for obtaining manageable, reliable, generally accepted music processing tools, such as composition systems. It is also essential for deepening our understanding of the capabilities and limitations of a computational account for music.

keywords: Music ontology, representation framework, time, hierarchy, inheritance, music structures, visual formalisms, abstraction operations, object-oriented representation.

*This research was supported, in part, by the Israeli Ministry of Science and Arts, and by the Paul Ivanir Center for Robotics and Production Management at Ben-Gurion University of the Negev.