The architecture of a musical composition system based on constraint resolution and graph rewriting

M. DESAINT-CATHERINE
R. STRANDH

LaBRI
Univ. Bordeaux 1
351, cours de la Liberation
33405, Talence Cedex
France
myriam@geocub.greco-prog.fr
strandh@geocub.greco-prog.fr

Abstract We propose an interface for the composer to describe his musical problem. We introduce two different levels in the relation between the composer and his piece: a global level, in which he considers his piece in full, without reference to time, and a sequential level where he focuses on some component of the piece, this time considering time and sequence. We propose a system design starting from an outline of a piece at the global level, which is represented by a constraint graph, and a thread at the sequential level, represented by a graph grammar, indicating how to make the graph grow. The constraint graph is specified as a set of equations on sequences of notes.

1 Introduction

The composer is not a problem solver, but a problem designer [4]. We are interested in providing the composer with a tool for specifying his musical problems. Of the two steps:

* specify the problem,
* solve the specification,

the first one is the more difficult, because it concerns the interface between the user and the system. We spent considerable time working on the second step with little or no progress because the first one was not well defined. This paper shows a solution to the first problem by proposing a suitable user interface to our system, as well as a software architecture for automatically solving the specification.

2 Choice of interface between composer and system

An interface is a language defined by its syntax and semantics. The computer scientist usually provides the user with syntactic tokens such as menus, boxes, buttons, windows,
curves, etc... From a syntactic point of view, such an interface is entirely satisfactory. The most important and difficult part of the interface problem is the semantics, i.e. the form and the meaning of the syntactic tokens and their operations. First of all, the composer, who is not necessarily a computer scientist, must be able to design his problem

1. in a natural way,
2. without having to explicitly tell the machine how to proceed in order to solve it,
3. without having to know how to solve it himself.

Items 2 and 3 indicate that a declarative language may be the best choice. In such a language, he can describe the solution he is expecting, and the system will build it for him. A description of a solution will be called a plan, and the declarative language for describing the solution will be called a plan language. Now, the first aim refers to the syntax and the semantic of the plan language. We mean by natural that the plan language must fit the state of mind of the composer. More precisely, the composer should not lose his initial idea while describing it in the plan language. After many fruitless language designs, we finally realized that a single language is insufficient for expressing the different ways the composer thinks about his creation. Thus, the composer senses his piece at different levels:

- the global level, where he represents in his mind the piece in full, abstracting out the time aspect.
- the sequential level, where he focuses on some component of the piece, concentrating on temporal relations within the component.

Components of the composition should be possible to relate to each other on the global level, be specified on the sequential level, or be recursively divided into subcomponents with their own global relations or sequential specifications, etc... Thus, intermediate levels could then be introduced infinitely. The composer may switch many times between those levels while composing. The plan language must allow for descriptions of the piece at different levels, and for easy switching from one level to another in a smooth way.

3 The two levels of the plan

Given the discussion in the previous section, we have decided to separate the global and the sequential level in our interface. The intermediate levels will be obtained by splitting the piece and consider the parts from either of the two levels. Corresponding to each of the levels, we introduce two sublanguages of the plan language.

3.1 The global level: the outline

The global level is described by an outline of the piece. A declarative language is provided for this purpose. In fact, for the composer the description of the outline is a description of the solution he is expecting. For this reason, the outline must also fit the solution. Thus, we chose to impose that it remains fixed during the sequential resolution.

The outline is described by an equational language [9]. Symbols of the language represent sequences of notes or superposition of sequences. Functional constraints (such as superposition, concatenation, length, duration, equality, addition, interval, etc...) apply to symbols.
The outline is represented by a constraint graph [7] with two types of nodes: the symbols and functional constraints connecting them (see a simple example displayed in figure 1).

Example: The outline specified in the following example is displayed in figure 1.

\[ L_1 = V_1 / V_2 \quad ; \; L_1 \text{ is the superposition of } V_1 \text{ and } V_2 \]
\[ V_1 = A + X + A \quad ; \; V_1 \text{ is } A \text{ appended to } X \text{ appended to } A \]
\[ \text{first}(V_1) = \text{do} \quad ; \; V_1 \text{ begins with the constant 'do'} \]
\[ \text{length}(L_1) = 10 \quad ; \; \text{length of } L_1 \text{ is 10} \]

Figure 1: An outline example

3.2 The sequential level: the thread

The sequential level is more complicated. It is a description of how the piece and all its components are organized in detail, specifically regarding sequence and timing. We have chosen to specify this level as a set of properties of the piece and its components. We call such a set a thread. The properties are of a different nature from those of the functional constraints of the outline because they concern data which are alterable. In fact, they also concern the "holes" in the outline, i.e. the parts that have to be computed automatically. Thus, those properties concern all data that don't appear in the outline and they will be used to manage the outline in order to complete it in the smallest details.

The thread is specified as a conjunction of boolean expressions on predicates bound to symbols of the outline. The composer may use predefined predicates or specify his own. The resolution is performed by rewriting the constraint graph, in a way determined by the predicates, until completion i.e. no remaining holes exist.

Example (from [2]): The principal conjunction of the specification of the thread is implicit.

\[ \text{mr-cjx}(V_1) \; ; \; \text{two successive notes of } V_1 \text{ are joint} \]
\[ \text{mr-cjt}(V_2) \]
\[ \text{no-5th}(L_1) \; ; \; \text{no successive fifths in } L_1 \]

ICMC 323c
4 Conclusion

We have defined the architecture of a compositional system that uses two different levels of specification, corresponding to two different ways of thinking about the piece. A system based on the architecture described in this paper is under development. The prototype, written in C++, will perform

- the outline interpretation giving a constraint graph,
- constraint propagation,
- the thread interpretation associating predicates with the graph, i.e. cursors and graph rewriting rules [6],
- the resolution, i.e. the generation of the outline holes.

Future work includes:

1. the specification of the predicates, by means of a language like LEIA [3] [1],
2. resolution under the composer's control.

Acknowledgement: We thank Pierre-Henri Vulliard for his collaboration on this work.

References


ICMC 323d