A music programming tool for learning object-oriented concepts

Hyosook Jung and Seongbin Park*

Department of Computer Science Education Korea University
{est0718, psb}@comedu.korea.ac.kr

Abstract

Object-oriented concepts are difficult to learn. In this paper, we propose a music programming language that involves music in object-oriented programming. We also present a client-server based system for individual learning and a P2P-based system for collaborative learning in a JXTA network.

1. Introduction

Algorithmic thinking and programming are an essential element for understanding information technology [1]. Especially object-oriented concepts have become an important element in modern computing curriculum [2]. However, most teaching approaches have not overcome the traditional way such as syntax-driven, starting with the popular 'Hello World' program [3].

In this paper, we present two music programming learning systems for both individual learning and collaborative learning. Our pedagogical approach is based on Constructivism, in particular Cognitive flexibility theory where learning should not begin with massive complexity because it leads to confusion that will overwhelm and discourage learners [4].

2. Music and programming

Composition is analogous to programming. Generally, music starts and finishes in the same key. Once the right key signature has been chosen, we do not need to write sharp or flat signs. Notes that do not lie in the scale of the key are called accidentals and these are marked by using the signs such as the natural, sharp and flat temporarily. When used, the accidental applies only to the end of that particular bar. In addition, it admits the change of key from one key to another. Key is to accidental what global variable is to local variable. Table 1 shows the grammar of the music programming language. In this grammar, all uppercase letters (i.e., MELODY, FORM, etc) are terminals and all lowercase letters (i.e., program, program_heading, etc) are nonterminals.

Table 1. Music programming grammar

<table>
<thead>
<tr>
<th>Grammar rule</th>
<th>Grammar representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>program</td>
<td>→ program_heading declarations body</td>
</tr>
<tr>
<td>program_heading</td>
<td>→ MELODY identifier</td>
</tr>
</tbody>
</table>

3. Music programming system

Figure 1 and figure 2 depict the proposed systems for programming using the proposed language described in section 2. In a client-server environment, students can learn programming as follows.

1. A student logs in the Web server.
2. The content manager sends the student the examples and exercises for programming music.

* To whom correspondence should be addressed.
3. The student writes a music program.
4. The student sends the music source program to the Web server and requests to compile it.
5. The music compiler compiles the music source program.
6. The music compiler sends a message which is the result of the compiling. If the program has errors, the student debugs the errors and tries to compile until there are no errors in the program.
7. The music compiler creates a MIDI file if the compiling is successful.
8. The MIDI player plays the MIDI file if the student requests to listen to the music.

4. Conclusions

In this paper, we presented a music programming language and proposed systems that can be used to learn object-oriented concepts. We taught 20 6th-grade students once a week for a month using a pilot program. We taught basic music theory about composition and how to do programming using the music programming language. They were interested in programming and wanted to know more about it. Currently, we are implementing both systems using SableCC [5] and JXTA technology [6].

5. References