CSIT 691D Independent Project

Sonification of the Document Object Model

Student: XIE Ning
E-mail: nxie@ust.hk
Supervisor: Prof. David Rossiter

Contents

1 Introduction 3
2 The Background 3
2.1 The Document Object Model (DOM) 3
2.2 The Musical Instrument Digital Interface (MIDI) 4
3 The Sonification Representation 5
3.1 Overview 5
3.2 With Gaps 5
3.2.1 Using the Blank 5
3.2.2 Using One Specific Sound 5
3.3 Without Gaps 6
3.4 Filter Some Specific HTML Node 7
4 Implementation 7
4.1 Overview 7
4.2 The Specific Details 9
4.2.1 'Options' 9
4.2.2 'The Tree Structure' 10
5 Conclusions 13
6 References 13

1 Introduction

Given any web page, we can obtain the Document Object Model (DOM), which is represented by a tree structure. However, most users could not understand it directly unless they have related backgrounds. Some people develop a system to show the DOM using graphics method. In contrast, we want to represent the DOM of any web page through combining Musical Instrument Digital Interface (MIDI) and some music models. The system we implement here can offer users a useful tool to hear the structure of any web page, and guarantee that it is possible to generate the piece of music automatically.

We introduce some backgrounds directly related to the topic of the work we done in chapter 2. In chapter 3, we discuss the main methods we utilize to represent the DOM of any given web page, including three methods. In chapter 4, we provide the details of how to implement the system and how to manipulate the system as users. In chapter 5, we describe the differences among the three methods conclude the facts we observe, and introduce possible future work we could do.

2 The Background

2.1 The Document Object Model (DOM)

The Document Object Model (DOM) [1] is a cross-platform and language-independent convention for representing and interacting with objects in HTML, XHTML and XML documents. Aspects of the DOM (such as its "Elements") may be addressed and manipulated within the syntax of the programming language in use.

In the system, we extract the DOM from the web page a user provides and store it in a tree structure. An illustration is shown in figure 1.

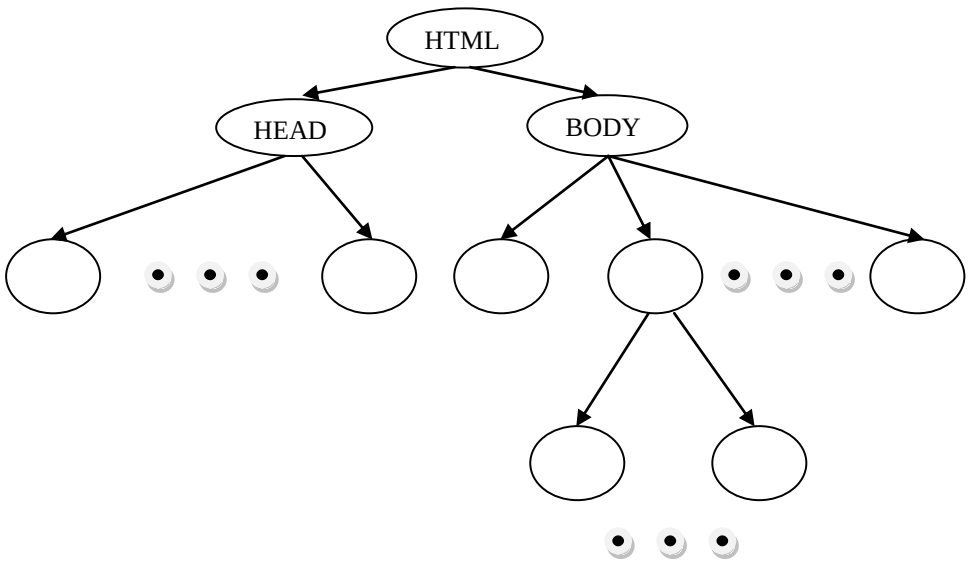

Figure 1 A DOM shown as a tree structure

We use two methods to represent the DOM tree in the system, shown in figure 2 and figure 3. In addition, the details of the meaning of each representation will be discussed in chapter 4. However, each method is based on the tree structure. The example used in this paper is Google’s web page,

which is www.google.com.hk.

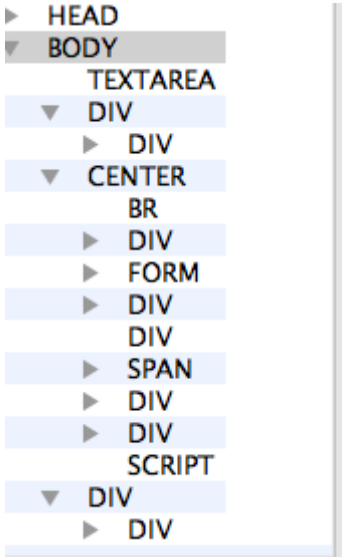

Figure 2 One way to show the DOM structure

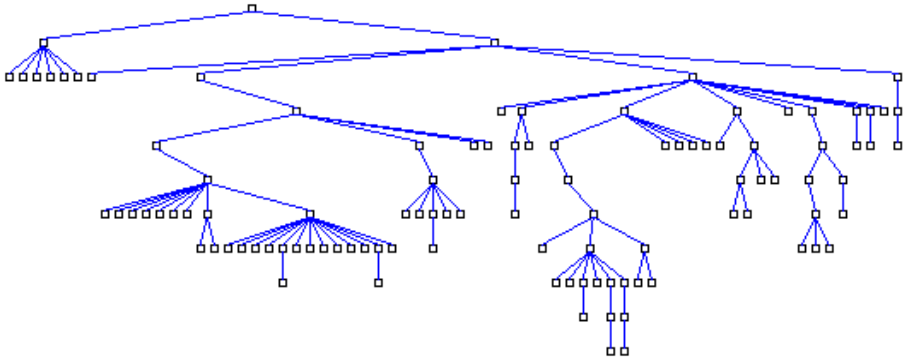

Figure 3 Another way to show the DOM structure

2.2 The Musical Instrument Digital Interface (MIDI)

Musical Instrument Digital Interface (MIDI) [2] is an industry-standard protocol that enables electronic musical instruments (synthesizers, drum machines), computers and other electronic equipment (MIDI controllers, sound cards, and samplers) to communicate and synchronize with each other. In addition, General MIDI or GM [3] is a standardized specification for music synthesizers that respond to the MIDI messages.

Table 1 Some options for users

	Details	Default
Sets of Instruments	Piano, Chromatic Percussion, Organ, Guitar, Bass, Strings, Ensemble, Brass, Reed, Pipe, Synth Lead, Synth Pad, Synth Effects, Ethnic, and Percussive	Piano
Volume	0 ~ 127	127
Period (seconds)	0.00 ~ 1.00	0.20

The system provides several parameters for users to select, so that users can choose, based on their preferences, different types of instruments, the volume of sound, and the period that each note plays. In table 1, it lists all the parameters related to MIDI devices that users can change. Note that

there are 16 sets of instruments offered by MIDI. However, the system only provides 15 sets for users to select, because the last set of instruments is reserved for the sonification of some specific HTML node users want to focus on.

3 The Sonification Representation

3.1 Overview

We implement two kinds of methods to represent the DOM structure, which is with gaps and without gaps. Note that the gap means the period of time used during sonification between two levels of the tree structure. Even though the method with gaps can guarantee users hear the whole structure clearly, it normally takes a long time to follow the complete structure, because not only does the node take one note to play, but also the gap does.

3.2 With Gaps

To concentrate on the structure of the DOM, we use a gap to split different levels. There are two possible positions we need to add the gap. If a node is a father, the system will wait for a while or play a specific note to notify users that it is going to the next level of the tree. If the node is the last child of its father, there still is another gap used to help users to understand the system is going back to the up level. To evaluate this method, we assume users set a as the interval to play one note, the tree structure with n nodes and m of n nodes having children. Then the system will need a*n seconds to play each HTML node, and an extra time of 2*m*a seconds to play the gaps, including once entering to the next level and once going back to the up level.

3.2.1 Using the Blank

When the tree structure of DOM goes down to the next level or goes back to the parent's level, the system sleeps for some seconds, which can be defined by users. The pseudo-code is described in table 2. Note that a blank means using no note playing for the gap. In another word, the system just sleeps for a period.

Here, we use a simple music model to generate a piece of music in order to guarantee the quality of music. The music model is +2, +2, +1, +2, +2, +2, +1 which means that if the note for the first node is 0, then the followings are 2, 4, 5, 7, 9, 11, 12, 14 and so on.

3.2.2 Using One Specific Sound

There is only one small difference between this method using one specific sound and that using no sound. In this method, we use the first note in the 'Applause' of the last channel to represent the focus is going down to the next level, and the first note in the 'Gunshot' of the last channel to represent the focus is going up to the parent's level. Then, the codes, the two lines of Sleep(interval), shown in table 2 will be replaced by the following respectively: playMIDIitem(the first note in the 'Applause', last channel, volume, interval) and playMIDIitem(the first note in the 'Gunshot', last channel, volume, interval). However, when we describe this method, for simplicity, we announce that we use one note to represent the gap. In fact, there are two notes in the system for entering the level and going back to former level.

Table 2 The pseudo-code for the method using blank

```
void examineChildElements(Element parentElement)
{
    Element element = parentElement.firstChild();    // element is used to represent the tree structure
    //  interval is defined by users to play a single note and sleep
    // volume and channel are defined by uses. The note is set as default from 0
    playMIDIitem(note, channel, volume, interval);
    if (element.firstChild() is not NULL)
    {
        Sleep(interval);    // Is going to next level, the system are going to sleep for a while
        Push(note);         // Push this level's note to a stack so that when it comes back to this level,
                            // the system can continue to play
    }
    if (element is parentElement.lastChild())
    {
        Pop(note);          // Pop the parent's note to continue to play following the model
        Sleep(interval);
    }
    Model(note);        // Get next note according to the model
    examineChildElements(element);
    element = element.nextSibling();    // nextSiling() is used to get the node's brother
}
```

3.3 Without Gaps

Even though the method without gaps may lead to the result that users possibly could not follow the structure easily, this method focuses on decreasing the time to play a piece of music generated automatically. Based on this reason, the system utilize this method, only taking a*n seconds without gaps, to provide another option, where a means the interval, n means there are n nodes in the DOM.

Because of no gaps to represent the level, it may be a little hard for users to understand. However, we use the same note to represent the HTML nodes at the same level. For different levels, the system takes advantage of the same music model as described in section 3.2.1 but multiplying by 2 in order to make it easy to hear the differences between two levels. For instance, if the note for the first level is 0, the notes for the following levels are 4 for the second level, and 8 for the third level, 10 for the fourth level, and so on. The pseudo-code is described in table 3.

Table 3 The pseudo-code for the method without gaps

```
void examineChildElements(Element parentElement)
{
    Element element = parentElement.firstChild();    // element is used to represent the tree structure
    // interval is defined by users to play a single note
    // volume and channel are defined by uses. The note is set as default from 0
    playMIDIitem(note, channel, volume, interval);
    if (element.firstChild() is not NULL)
    {
        Push(note);         // Push this level's note to a stack so that when it comes back to this level,
                            // the system can continue to play
        note = nextLevelNote;
    }
    if (element is parentElement.lastChild())
        Pop(note);          // Pop the parent's note to continue to play
    examineChildElements(element);
    element = element.nextSibling();    // nextSiling() is used to get the node's brother
}
```

3.4 Filter Some Specific HTML Node

In some situations, users want to focus on one specific HTML node, such as ‘A’ or ‘DIV’. This system provides a simple mechanism to represent such node specially. There are six HTML nodes which are users are possibly interested in, including ‘A’, ‘DIV’, ‘FRAME’, ‘IMG’, ‘INPUT’, and ‘STYLE’. We use the first note of ‘Guitar Fret Noise’, ‘Breath Noise’, ‘Seashore’, ‘Bird Tweet’, ‘Telephone Ring’, and ‘Helicopter’ in the last channel to represent each HTML node mentioned above.

4 Implementation

4.1 Overview

The environment is set on the Macintosh, and utilizes the several special frameworks, including QtWebKit, QtNetwork, QtCore, AudioUnit, AudioToolbox, CoreServices, and so on, to combine of the networking function and MIDI playing function. There are three areas in our design, which are ‘Options’, ‘Web Page Display’, and ‘The Tree Stucture’. The main window is shown in figure 4.

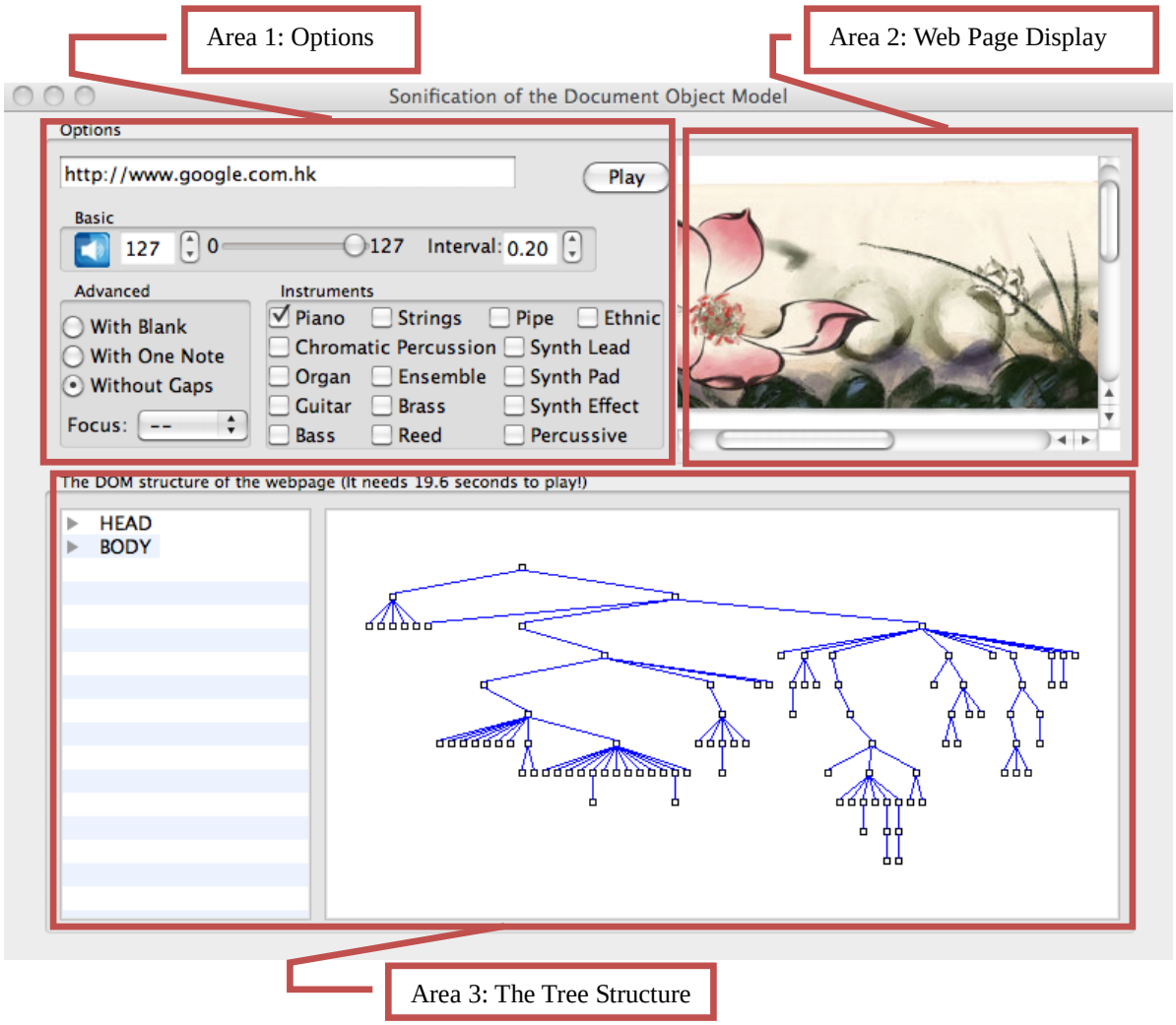

Figure 4 The overview of the system

The second area, which is ‘Web Page Display’, is used to show the web page that users input in the first area, ‘Options’. Once users input the URL of a web page, the system will show the web page in

area 2 and the tree structure of the DOM in area 3. In the first area, ‘Options’, most choices are set as default, which we mention in the previous chapters. After users click the ‘Play’ button, the system will generate a piece of music based on the tree structure of the DOM and the options users set. The flow of processing is shown in figure 5. Users can follow the steps to use the system.

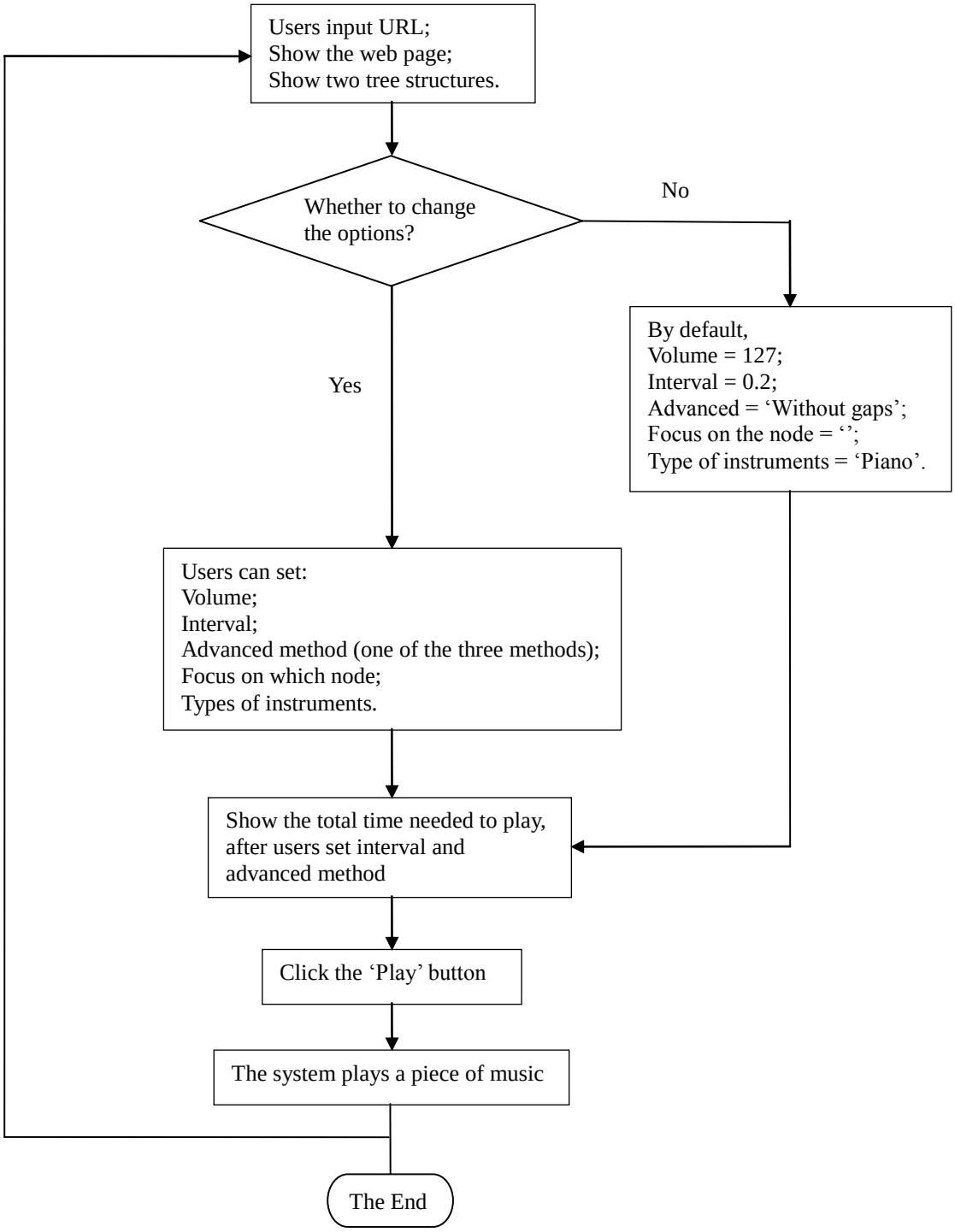

Figure 5 The flow of processing

4.2 The Specific Details

4.2.1 'Options'

In this area, there is one line editor for users to input the URL, one button to play a piece of music, and three different sub-areas for helping to define which kind of music to generate. Figure 6 shows all the options that a user can choose.

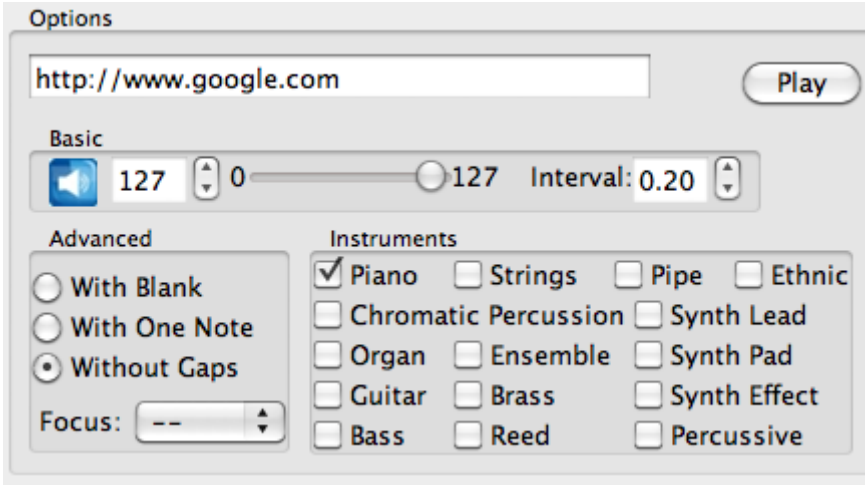

Figure 6 The overview of area 1

The first sub-area is the basic functions. Users can change the volume of the sound and set the interval for each note to play, including the gap. It is shown in figure 7.

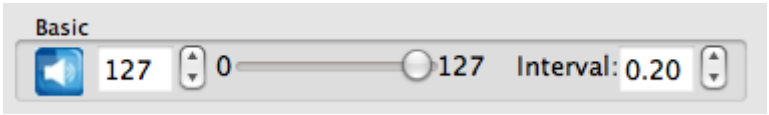

Figure 7 The basic sub-area of 'Options'

The second sub-area is the advanced functions. The theory is described in chapter 3. For convenience, when users choose the radio box, 'With One Note', or the combo box to filter one specific node, the system will play the node once in order to notify users the sound for the node needed to hear carefully. It is shown in figure 8. Note that 'With Blank' is described in section 3.2.1, 'With One Note' is explained in section 3.2.2, and 'Without Gaps' is introduced in section 3.3. The combo box for the 'Focus' is the function to filter one node described in section 3.4.

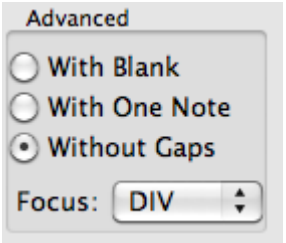

Figure 8 The advanced sub-area of 'Options'

The last sub-area is the instruments functions, which provides fifteen different sets of instruments for users to choose. If users only pick one set instruments, the whole level of the tree structure is played by this one set instruments. If users choose more than one set instruments, such as 'Piano', 'Guitar', 'Brass', and 'Pipe', the system will play the first level of the DOM using 'Piano', the

second level using ‘Guitar’, the third level using ‘Brass’, and the forth level using ‘Pipe’. After the forth level, we loop to use the sequences of the sets for the following levels, which means the system will play ‘Piano’ for the fifth level, and so on. The window area for choosing the sets of instruments is shown in figure 9.

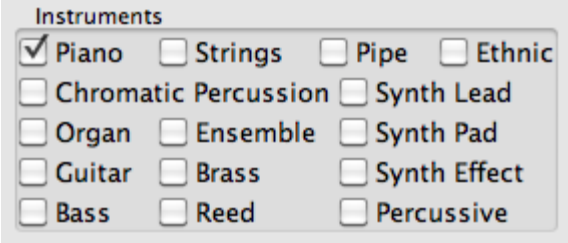

Figure 9 The instruments to choose

4.2.2 ‘The Tree Structure’

There are three sub-areas shown in area 3, ‘The Tree Structure’. The first sub-area is that the system will calculate the total time cost playing the piece of music. For example, in figure 4, the system needs 19.6 seconds to play the DOM of Google’s web page. With the same input choices, here is another example, which is YouTube’s web page (www.youtube.com). It is easily found that it needs more time to play because of the complexity of the DOM. This example is shown in figure 10. The system will play 138.8 seconds to finish the piece of music.

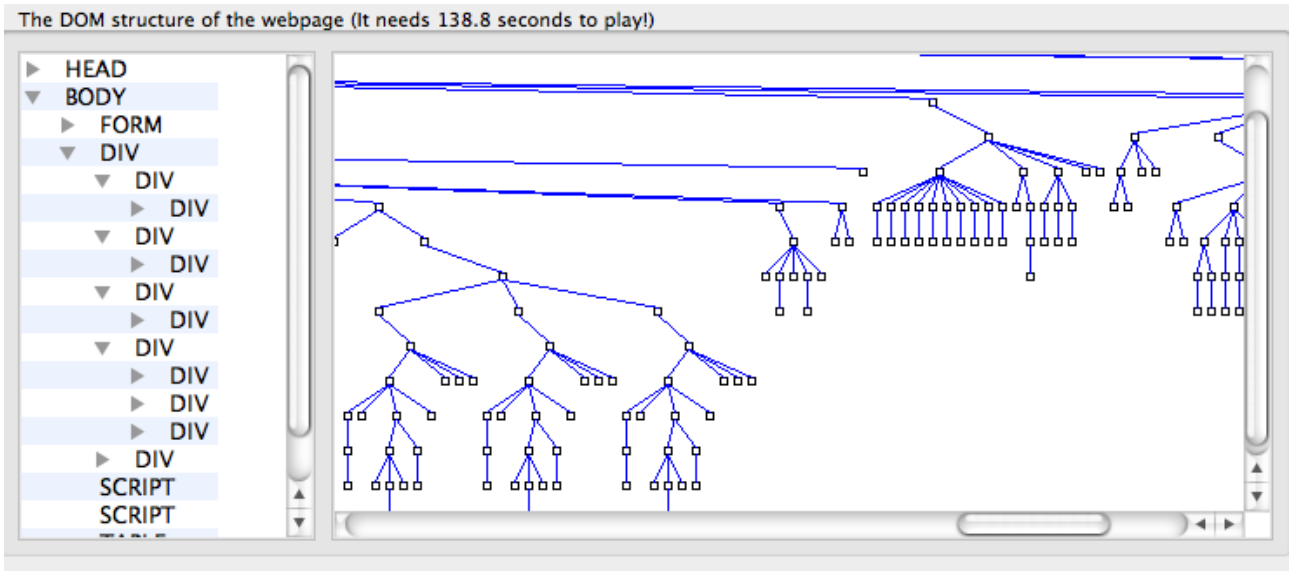

Figure 10 The example of YouTube’s web page

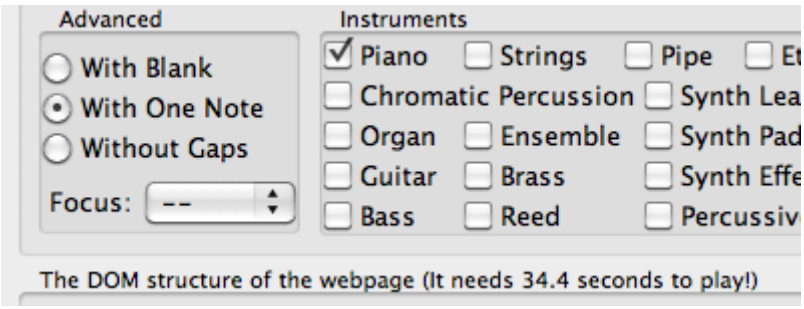

Figure 11 Another method to play Google’s web page

However, if we change to the method ‘With One Note’, using Google’s web page, the system will spend 34.4 seconds to play. The result is shown in figure 11. Here we can do the calculation: (34.4 seconds – 19.6 seconds) / 0.2 seconds = 74 gaps occur in the tree structure of figure 4, where 0.2 means the period for one note or gap. These gaps include going down to next level and going back the up level of the tree structure, which means there are 74 / 2 = 37 first children in the tree structure. The number can be easily counted in figure 4.

Table 4 The pseudo-code for drawing the tree structure

```
// First Round: Remember the basic positions of all    nodes
while (each node in the tree)
{
      // element is used to represent the tree structure
      element = element.firstChild();
      // xPos and yPos are used to store the x position and y position of each    node
      if (element.firstChild is not NULL)
            the depth of tree ++;
      else
            xPos += LENGTH;
      yPos = current depth * LENGTH;
      if (element is parent.lastChild())
      {
            if (element.firstChild() is not NULL)
                  Push(current depth of the tree);
            else
            {
                  if (parent.nextSibling is NULL AND Stack is empty)
                  {
                        int disTmp = Stack.back();       // disTmp is used to calculate how many elements needed to pop
                        disTmp -= Stack[Stack.size() - 2];
                        while (disTmp)
                        {
                                    Pop(current depth of the tree);
                                    disTmp--;
                        }
                  }
                  else
                        current depth --;
            }
      }
}
// Second Round: Adjust the new positions of some nodes
while (each position in the previous step)
{
      if (next x position is current x position)
            Push(i+1);
      if (next y position < current y position)
      {
            // nTmp is used to calculate how many element needed to adjust the new x position when it is the last child of
its parent
            int nTmp = (current y position – next y position) / LENGTH;
            while (nTmp)
           {
                  Pop(iTmp);
                  Position[iTmp] += (current x position - Position[iTmp]) / 2;      // Adjust the new position for x
                  nTmp--;
            }
      }
}
```

In the left area, we use Qt's tree structure, which is that the structure of the data takes the form of a tree built from TreeItem objects. Each TreeItem represents an item in a tree view, and contains several columns of data.

The data is stored internally in the model using TreeItem objects that are linked together in a pointer-based tree structure. Generally, each TreeItem has a parent item, and can have a number of child items. However, the root item in the tree structure has no parent item and it is never referenced outside the model. Since each item in a tree view usually contains several columns of data, it is natural to store this information in each item. The basic idea is shown in figure 12.

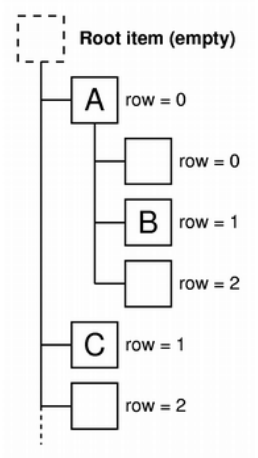

Figure 12 Qt's tree structure

Since Qt's simple tree structure is easy to construct and shows all necessary information in the TreeItem, the simple tree structure is difficult for users to understand. It is hard for users to imagine exactly what the tree structure looks like. Hence, we implement another tree structure in the right area of area 3, 'The Tree Structure'. The pseudo-code is described in table 4.

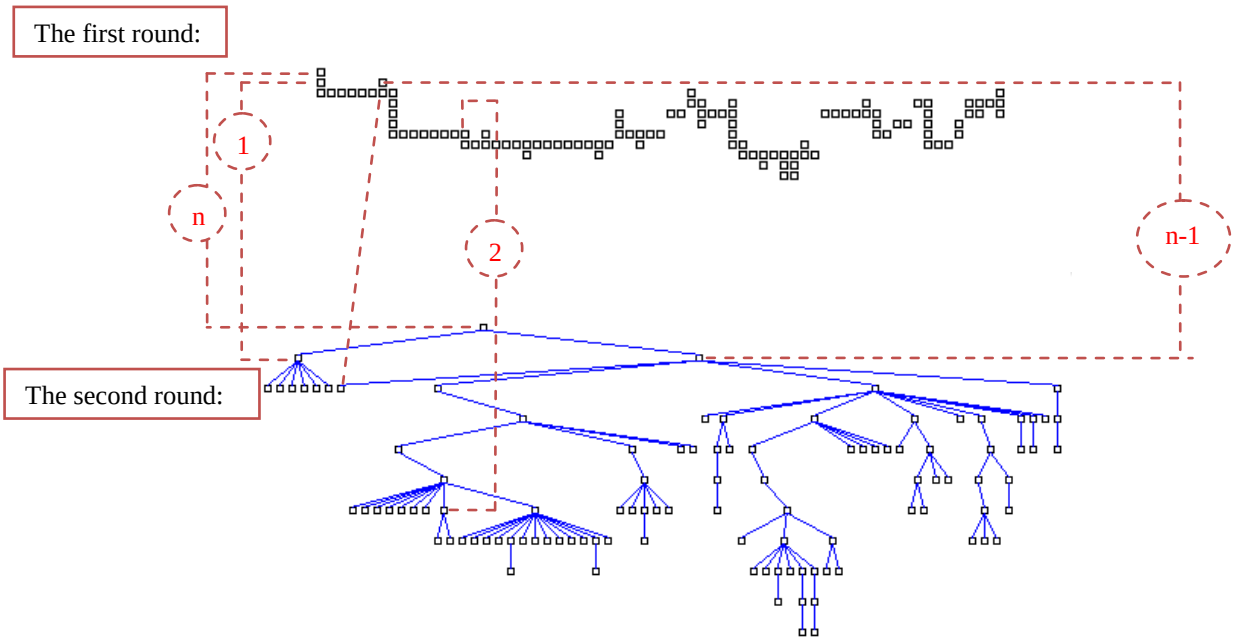

Figure 13 Two rounds to draw the tree structure

In the algorithm, there are two rounds to draw the tree structure. We first calculate the x position and the y position of each node roughly, and then adjust the x position to more appropriate position. In the second round, we not only re-calculate the new x position for each node, but also add branches in blue color between parent node and child node. The new result offers a directly understanding for users to obtain the tree structure of the DOM.

Using Google's web page as an example, figure 13 shows the process of these two rounds. In figure 13, it labels the first node, the second node, the node before the last one, and the last node, which are re-calculated for the new positions. The sequence to draw each node is based on whether all its children are finished or not. If the node's children are empty, then the system will draw the node directly. However, in figure 13, we only label the nodes with children in the sequence, ignoring those without children.

5 Conclusions

We develop a system to represent the DOM of any web page through three methods, with blank, with one note, and without gaps. To give users more control on the system, we provide the interface with volume control, interval control and instruments control for users. With different kinds of options, users can obtain diverse pieces of music for one web page.

Compared with these three methods, the one with one specific node could generate an interesting piece of music, but taking a longer time to play than the one without gaps. There are some sounds recorded, with different kinds of examples, to show the differences among each method.

For possible future work, we plan to add more options for users. For instance, now, users cannot choose which note to represent the special node to focus on. The system determines the note for users. In addition, there is no animation effect during playing the piece of music. It is difficult for users to understand exactly which node the system is playing.

6 References

[1] WIKIPEDIA. http://en.wikipedia.org/wiki/Document_Object_Model. Retrieved on 3 May 2011
[2] WIKIPEDIA. http://en.wikipedia.org/wiki/MIDI. Retrieved on 3 May 2011
[3] WIKIPEDIA. http://en.wikipedia.org/wiki/General_midi. Retrieved on 3 May 2011
[4] Alan Marsden and Anthony Pople. Computer representations and models in music. Academic Press. 1992
[5] F. Richard Moore. Elements of computer music. Prentice Hall. 1990
[6] Qt Class Reference. http://doc.trolltech.com/3.3/index.html. Retrieved on 3 May 2011