Chapter 4

Implementation of DAPHNE system

4.1 Development environment

The development environment to build and run DAPHNE is as follows:

- **Hardware**
  - PC: NEC NX, PENTIUM II 300, Mem 64MB, HD 4G
  - Monitor: NEC NX + Gateway EV700
  - Scanner: HP ScanJet 4c

- **Display driver (for 2 monitors)**: GA-VDB16/PCI

- **Operating system**: Windows98

- **Programming languages**: Visual Studio 6.0 (Visual C++, Visual Basic)

4.2 Composition of DAPHNE system

DAPHNE is an object-oriented system. Basically it is composed of two parts: the GUI and the Kernel. The GUI part is a standalone program developed with Visual Basic. The Kernel is developed with Visual C++ as a dynamic link library (DLL). Figure 21 depicts their relationship.

Since the Kernel is implemented as a DLL, it is independent with the GUI part. The executable program can keep a small size. Moreover, multiple programs can be executed with only one copy of the Kernel in the memory, which makes the system more efficient and faster.
Figure 21. Composition of DAPHNE

Figure 22. Modules diagram of the GUI
4.2.1 The **GUI**

The GUI part interprets the input commands by users, converts them into the system instructions and calls the Kernel with these instructions. Meanwhile, it displays the analysis results retrieved from the Kernel. In addition, it handles the visualization of performance analysis and replay.

Figure 22 depicts the components inside the GUI.

There are three kinds of windows related to analysis:

- **Main control window**
  It controls and coordinates all the other windows.

- **Analysis sub-windows**
  For each analysis item, there is at least one analysis sub-window, which performs the real analysis, retrieves users' input and displays the corresponding results.

- **Score image windows**
  A score image window displays one page of score and permits the user manipulating it.

All these windows can receive user's input commands, translate them to the system instructions and pass it to the Kernel. Meanwhile, the results from the Kernel will be displayed in the corresponding window in a proper format.

Besides, GUI also provides utilities like Performance Data Visualization and Performance Replay, which can display the analyzed UNI data in graphs and playback the analyzed performance respectively.

4.2.2 The **Kernel**

The analysis Kernel, which is triggered by the GUI, implements all the music analysis functions. The Kernel is depicted in Figure 23.

- **DaphneMain** communicates with GUI. It dispatches the system instructions to other modules in the Kernel and sends the results back to GUI.

- **EuropaManager** maintains the EUROPA files. It loads a EUROPA file, converts the EUROPA data to the internal data format, and does some calculations like number of notes in a measure and number of measures in a piece.
> **ScoreManager** takes charge of files related to scores. Its purpose is to prepare enough data for score images used in **GUI**. It reads the score description file and obtains the coordinates information of notes. This coordinates information can be interchanged with the position information stored in the **EUROPA** file and thus makes it easier to display the analysis results on the score images.

> **DaphneManager** maintains the DAPHNE data files, which store the analysis results.

![Diagram](image)

*Figure 23. Modules diagram of the Kernel*

> **UniManager** is the file manager concerning UNI data files. It contains the performance data described before. By now, part of the analysis results can also be stored in the UNI data files. To calculate the note's positions in a UNI file, the note's measure position information stored in the DAPHNE file is required.

> **Analyzer** is the heart of the whole system. It performs the music analysis, such as structure analysis, structure function analysis, similarity level analysis, etc. *Figure 24*
depicts the analysis modules in the Analyzer and how their results are shared with each other. Musical structure is the basis of all the other analyses and must be done in the first step. Harmony information is used by both Structure function analysis and Musical form analysis.

![Diagram](image)

Figure 24. Modules in the Analyzer

### 4.3 Analysis specification of DAPHNE system

#### 4.3.1 Analysis items

Some items are very important to be observed for music analysis. In DAPHNE, the following observation items are set.

##### 4.3.1.1 General

- Analysis range
  - Start position / End position
- Voice part with melody
  - Range / Voice part with melody
4.3.1.2 Structure analysis

- Unit
  - Unit range (Start position, End position) / Prev unit / Next unit / Unit level
- Labeling
  - Range / Name

4.3.1.3 Similarity level

- Similarity level of unit
  - Reference unit / Similarity level / Similarity type of each measure

4.3.1.4 Structure functions

- Upbeat
  - Range
- Initiative
  - Position
- Desinence
  - Range

4.3.1.5 Harmony

- Global tonality
- Local tonality
Range / Tonality

➤ Chord

Chord / Root omission / Inversion

4.3.1.6 Musical forms

➤ Paragraph

Level / Range / Reference paragraph / Similarity level of paragraph

➤ Musical form type

Name / Number of part

➤ Range of each part

Number / Range

4.3.1.7 Others

➤ Score

Number of page / Size of page / Note position

➤ Compare analysis results of two versions' scores

Position of difference / Information of unit

4.3.2 Input syntax of music analysis

The input syntax of music analysis is described below.

4.3.2.1 Basic settings

As the fundamentals of analysis, the segment range of analysis and the positions of notes are the premises of further observations.

4.3.2.1.1 Position

① Position information level

➤ Number of measure position
➤ Beat position
➤ Fractional beat position

   (beginning of note - beginning of beat) / pseudo-sound value

② Specifying method (example: measure 9, note 1)
   ➤ n (measure number)

    Example: 9
   ➤ n_m (measure number_beat position)

    Example: 9_1
   ➤ n_m_l (measure number_beat position_fractional beat position)

    Example: 9_1_0

4.3.2.1.2 Range

① Current range - Range on the score which the user is analyzing.

   If the range of the analysis process is not specified, the current range value will be the default range since usually users continue analyzing on the same range. The range is changed only when the input command contains positions definitely.

② Specifying method

   Example: from the start of the 9th measure to the 2nd 1/16 note of the 4th beat in the 10th measure.

   ➤ from <start position> to <end position>

    Example: from 9 to 10_4_1/4

   ➤ <unit-class> <measure position> -- unit range

    unit-class: motif, phrase, sentence, and paragraph

    Example: motif 9

4.3.2.2 Commands

   There are two types of commands, declaration and inquiry. Declaration means the user specifies the content of analysis manually. Inquiry means the user asks the system to enumerate the
candidates of the content of analysis commands based on the last analysis action. It can be used for automatic analysis.

Inside DAPHNE system, input commands are used as common interface. In fact, all the inputs from analysis sub-windows and score images must be converted to these commands first.

The command syntax is shown as follows. For each command,

"Syntax" stands for the format of the command.

"Example" stands a sample input by the user.

"Output" shows the result obtained from DAPHNE.

<range> stands for:

from <position> to <position> / at <position> / of <position>

for example: from motif5 to motif6

at 1_1_1/2

of motif9

1) Analysis range (change and inquiry)

➢ Declaration

Syntax: set current range <range>

Example: set range at motif15

➢ Inquiry

Syntax: show current range

Example: show current range

Output: from 15_2_3/4 to 15_4

2) Unit

➢ Declaration

Syntax: set <unit-class> <measure position> <range>

-- declaration of unit range

Example: set motif1 from 1_1_0 to 2_4_1/2

➢ Inquiry

49
3) Similarity level of unit

➢ Declaration

Syntax: set similarity level of <unit-range>:<unit-range, level>

Example: set similarity level of motif11: motif9, 2

➢ Inquiry

Syntax: show similarity level of <unit-range>

Example: show similarity level of motif11

Output: similarity level of motif11: motif9, 2

4) Labeling

➢ Declaration

Syntax: name <label> for <range>

Example: name g15 for motif15

➢ Inquiry

Syntax: show name for <range>

Example: show name for motif15

Output: name for motif15: g15

5) Reflected unit

➢ Declaration

Syntax: reference performance of

<unit-range(reference)> to <unit-range(reference)>

Example: reference performance of motif11, motif13 to motif9
6) Structure function (upbeat, initiative, desinence)

Upbeat

> Declaration

Syntax:  
Example:  
Output:  

> Inquiry

Syntax:  
Example:  
Output:  

Initiative

> Declaration

Syntax:  
Example:  

> Inquiry

Syntax:
Example:  
Output:  

Desinence

> Declaration

Syntax:  
Example:  

> Inquiry

Syntax:  

51
7) Slur
   ➤ Declaration
      Syntax:   set slur <range>
      Example: set slur from 0_3 to 1_1
   ➤ Inquiry
      Syntax:   show slur [<range>]
      Example: show slur
      Output:   slur from 0_3 to 1_1

This is a user-specified command, which means the analysis can not be done automatically.

8) Tonality
   ➤ Declaration
      Syntax:   set tonality to <tonality>[majorminor] [range]
      Example:  set tonality to E Minor
   ➤ Inquiry
      Syntax:   show tonality [range]
      Example:  show tonality from 16 to 20
      Output:   C

9) Chord Function
   ➤ Declaration
      Syntax:   set chord to <chord> <range>
      Example:  set chord to 1 at 9_1_0
   ➤ Inquiry
      Syntax:   show chord <range>
Example: show chord at 10_1_0
Output: 4

10) Musical form name

- Declaration
  Syntax: set musical form name as <formname>
  Example: set musical form name as sonata

- Inquiry
  Syntax: show musical form name
  Example: show music form name
  Output: music form name as sonata

11) Musical form range

- Declaration
  Syntax: set musical form range <range>
  Example: set musical form range from motif1 to motif24

- Inquiry
  Syntax: show musical form range
  Example: show musical form range
  Output: musical form range from motif1 to motif24,
           from motif25 to motif77,
           from motif77 to motif105
4.4 Data processing in DAPHNE

4.4.1 Data files

In DAPHNE, a user can analyze a music piece on score images, which are exactly the same as the published ones. In order to provide this graphical feature, several files must be prepared prior to the analysis session for each piece. Including bitmap files for score sheets, a score description file, which includes the X-Y positions of notes on the score sheets, and a score-coding file. To perform an analysis, the user need to specify the analysis range, define the analysis item and decide whether to do it automatically or manually on the score image. The obtained results will be displayed on the score images immediately and dumped into the music analysis data file (DAPHNE file) and the performance data file (UNI file) which can be shared among the other systems of PSYCHE. As shown in Figure 25, there are three types of data files in DAPHNE.

Figure 25. Data files in DAPHNE
4.4.1.1 Score data files

Score-coding file (EUROPA)

EUROPA (Exact Universal Representation Of Phrasing and Articulation) is a general-purpose music description language developed in my laboratory.

EUROPA describes scores and specify the methods of performance for automatic performance especially of pianos. When describing scores, the performance information like notes, pedals, and expression symbols can be recorded. When specifying the methods of performance, not only the performance method of each note, but also the way to change the tempo and volume in a specific section can be recorded. For instance, a part of Chopin Mazurka 7-3 can be expressed in EUROPA as shown in Figure 26.

```
!p1[maz7.eur]
!t 180
!r 3/4
!p9v1 f8 s16 g16 a@8 b@8 c+8 a@8
!p10v1 (g/a@g)4 f4 f+4
!p11v1 b@8 s16 c+16 d@+8 e@+8 f+8 d@+8
!p12v1 (d@+/e@+/d@+)4 c+4 a@4
!p13v1 f8 s16 g16 a@8 b@8 c+8 a@8
!p14v1 (g/a@g)4 f4 c+4
!p15v1 c8 s16 d16 e@8 f8 g8 b-8
!p16v1 c4 . c8 d@8 s16 c16
!p17v1 f8 s16 g16 a@8 b@8 c+8 a@8
!p18v1 (g/a@g)4 f4 f+4
!p19v1 b@8 s16 c+16 d@+8 e@+8 f+8 d@+8
!p20v1 (d@+/e@+/d@+)4 c+4 a@4
!p21v1 f8 s16 g16 a@8 b@8 c+8 a@8
!p22v1 (g/a@g)4 f4 c+4
!p23v1 c8 s16 d16 e@8 f8 g8 b-8
!p24v1 c2 c+4
```

Figure 26. EUROPA data for Chopin Mazurka 7-3

Score image file

In order to display scores as real as possible, scanned score images are used as user analysis interface. The files are in BMP format. Each file stores one page of score. A whole piece is
composed by a series of score image files.

**Score description file**

In order to recognize the range the user specified on the score image and show the analysis results on the right place, score description files are used to store miscellaneous information of scores. Including total score pages, dimension of each page, positions of notes, etc. Score description files, along with score image files, are created to assist analyzing on the score interface, and must be used together with EUROPA files.

### 4.4.1.2 Performance data files

UNI is an extendable note events file format based on MIDI messages. In the file, each line represents a note event containing both the performance information, which is derived from MIDI file, and the score analysis information obtained from DAPHNE. The file structure is shown in Figure 27.

![UNI file structure](image)

**Figure 27. UNI file structure**

Figure 28 shows a part of Chopin's Mazurka 7-3. Fields of each note event are labeled for readability.

*In table 4, the fields of UNI file are described. Fields K, V, L, and I are information from MIDI data; fields A and W are from scores. Besides readability, the most important characteristic of UNI format is the extensibility. More fields can be easily added if new analysis items are appended. In table, fields from MOT to SS are for structure analysis results (such as motif, reference motif, and similarity level, etc.). Fields ACC, UPB and DES are for structure functions. Other fields are for musical forms, etc.*
Table 4. UNI file field description

<table>
<thead>
<tr>
<th>Field</th>
<th>Data type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>Int</td>
<td>Interval</td>
</tr>
<tr>
<td>K</td>
<td>Int</td>
<td>Key</td>
</tr>
<tr>
<td>V</td>
<td>Int 0-127</td>
<td>Velocity</td>
</tr>
<tr>
<td>L</td>
<td>Int</td>
<td>Length</td>
</tr>
<tr>
<td>A</td>
<td>Int</td>
<td>Note value</td>
</tr>
<tr>
<td>W</td>
<td>Int</td>
<td>Voice part</td>
</tr>
<tr>
<td>M</td>
<td>Int</td>
<td>Measure</td>
</tr>
<tr>
<td>MOT</td>
<td>Int</td>
<td>Motif</td>
</tr>
<tr>
<td>PHR</td>
<td>Int</td>
<td>Phrase</td>
</tr>
<tr>
<td>STN</td>
<td>Int</td>
<td>Sentence</td>
</tr>
<tr>
<td>MRF</td>
<td>Int</td>
<td>Reference motif</td>
</tr>
<tr>
<td>PRF</td>
<td>Int</td>
<td>Reference phrase</td>
</tr>
<tr>
<td>SRF</td>
<td>Int</td>
<td>Reference sentence</td>
</tr>
<tr>
<td>MSL</td>
<td>Int</td>
<td>Similarity level of motif</td>
</tr>
<tr>
<td>PSL</td>
<td>Int</td>
<td>Similarity level of phrase</td>
</tr>
<tr>
<td>SSL</td>
<td>Int</td>
<td>Similarity level of sentence</td>
</tr>
<tr>
<td>ACC</td>
<td>Int 0,1</td>
<td>Initiative</td>
</tr>
<tr>
<td>UPB</td>
<td>Int 0,1</td>
<td>Upbeat</td>
</tr>
<tr>
<td>DES</td>
<td>Int 0,1</td>
<td>Desenanse</td>
</tr>
<tr>
<td>FOM</td>
<td>Int</td>
<td>Musical form</td>
</tr>
</tbody>
</table>
Figure 29 is an example of extended UNI file, which stores the analysis results of Chopin’s Mazurka 7-3.

---

Figure 29. UNI data for Chopin Mazurka 7-3 (after analysis)
4.4.1.3 Music analysis data files

In order to distinguish various information, DAPHNE saves music analysis results in four types of files: .STR, .MET, .HAR, and .FOM. These files’ descriptions are depicted in Table 5.

<table>
<thead>
<tr>
<th>File type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.STR</td>
<td>Musical structure data file. Preserve the results of musical structure and similarity level analysis. Format: <img src="binary/octet-stream" alt="Format diagram" /> Examples: motif9, 9_1_1, 10_3_1, 9, 0;</td>
</tr>
<tr>
<td>.MET</td>
<td>Structure function data file. Preserve the results of structure function analysis. Format: <img src="binary/octet-stream" alt="Format diagram" /> Examples: ACC, 10_3_0; INT, 12_2_1/3; ANA, 9_1_0, 12_2_0; DES, 12_2_2/3, 12_3_0;</td>
</tr>
<tr>
<td>.HAR</td>
<td>Harmony data file. Preserve the results of harmony analysis. Format: <img src="binary/octet-stream" alt="Format diagram" /></td>
</tr>
<tr>
<td>.FOM</td>
<td>Musical form data file. Preserve the results of musical form analysis. Format: <img src="binary/octet-stream" alt="Format diagram" /></td>
</tr>
</tbody>
</table>

4.4.2 Data structures

As an object-oriented system, DAPHNE is built upon a series of classes. Each class describes an object in music processing like a file manager, a music unit, or an analysis engine. In this section, some important classes are introduced.
4.4.2.1 Data structure of the EuropaManager

<table>
<thead>
<tr>
<th>Class</th>
<th>Data</th>
<th>Functions</th>
</tr>
</thead>
</table>
| DNote | int Pitch; | void SetNotes();
|       | int Beat; | void *GetNotes();
|       | int StartTime; | int GetNumOfNotes();
|       | ...... | void ProcessLine();

<table>
<thead>
<tr>
<th>Class</th>
<th>Data</th>
<th>Functions</th>
</tr>
</thead>
</table>
| DMeasure | DNote Note; | DMeasure *GetMeasure();
|        | int NumOfNotes; | DNote *GetNote();
|        | ...... | void GetLine();

<table>
<thead>
<tr>
<th>Class</th>
<th>Data</th>
<th>Functions</th>
</tr>
</thead>
</table>
| DEuropa | int NumOfMeasures; | int NormalPitch();
|        | int NumOfW; | int Beat_Per_Measure;
|        | int NormalPitch(); | DMeasure Measure[];
|        | ...... | void ProcessLine();

Figure 30. Classes in the EuropaManager

The EUROPA data contain score information, such as pitch, length. They are stored in the EUROPA file. For the analysis process, the DAPHNE system encapsulates the EUROPA data into a DEuropa object. As shown in Figure 30, there are three classes in the EuropaManager, which are DNote, DMeasure and DEuropa. DEuropa is a class to handle score data, such as number of measures (NumOfMeasures), number of voice part (NumOfW), key signature (NormalPitch), and rhythm (Beat_Per_Beat). It reads information of a note through member function GetNote(), and obtains information on a measure with function GetMeasure(). It loads a score data when a DAPHNE session starts, then creates DNote and DMeasure objects. When a voice part is specified, a note on a score is uniquely specified with the number of the measure it resides and the offset from the start position of the measure. Class DMeasure corresponds to a measure in a score, which has a list of DNote objects as an attribute. Class DUnit represent occurrences of various musical structures. It has attributes for the start and end positions of an occurrence in a score presented by measures and the offset in the measure, the level of the hierarchy (whether the occurrence is a motif, phrase, or something else), and attributive analysis information involved in the occurrence.
4.4.2.2 Data structure of the DaphneManager

The results of music analysis are stored in the DAPHNE file. The DAPHNE files contain the information of the start and end positions of the structure, the number of reference structures, similarity level, and the structure functions, etc. For processing, the number of unit which is either previous or next to a structure is specified. As shown in Figure 31, there are two class in module DaphneManager, which are DUnit and DDaphne. In DDa,hne, the data of structural unit like motif, phrase, and sentence can be taken out of an existing DAPHNE file with function Load(), and the analysis results are stored into the DAPHNE file with function Save().

![Figure 31. Classes in the DaphneManager](image)

61
### 4.4.2.3 Data structure of the UniManager

![Class DUni Diagram](image)

**Class DUni**

<table>
<thead>
<tr>
<th>Functions</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL Save();</td>
<td>Europa *EuropaPtr;</td>
</tr>
<tr>
<td>BOOL SaveMid();</td>
<td>Daphne *DaphnePtr;</td>
</tr>
<tr>
<td>BOOL Compare();</td>
<td></td>
</tr>
<tr>
<td>void MakeUni();</td>
<td></td>
</tr>
<tr>
<td>int GetNoteNo();</td>
<td></td>
</tr>
<tr>
<td>void GetMNNo();</td>
<td></td>
</tr>
<tr>
<td>void GetUni();</td>
<td></td>
</tr>
<tr>
<td>void ModifyUni();</td>
<td></td>
</tr>
<tr>
<td>void Getline();</td>
<td></td>
</tr>
<tr>
<td>BOOL ProcessLine();</td>
<td></td>
</tr>
<tr>
<td>int GetNote();</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 32. Class in the UniManager**

The results of music analysis can be saved into the UNI data files directly. Inside DAPHNE, a note is expressed as "measure number, beat position". The UNI data files which obtained from performances do not contain measure information. Such information, like measure number and fractional beat position, can only be obtained by collating each note’s position in the whole score with the note information stored in the EUROPA file. Figure 32 depicts the members of class DUni. Function GetNoteNo() calculates the note’s position in the whole score according to the "measure number, beat position" structure. Functions Load() and Save() can fetch the performance data from UNI file and save the analysis results back respectively. Function Compare() finds the difference between two UNI files. And function MakeUni() can save this difference into another UNI file.
### 4.4.2.4 Data structure of the Analyzer

<table>
<thead>
<tr>
<th>Class</th>
<th>DAnalyzer</th>
</tr>
</thead>
</table>
|         | BOOL FInputEurope();  
|         | BOOL FInputDaphne();  
|         | BOOL FSaveDaphne();   
|         | BOOL FCut();          
|         | BOOL FSimi();         
|         | BOOL FReadSEuropa();  
|         | BOOL FSetCurRange();  
|         | BOOL FImqCurRange();  
|         | BOOL FMetrIk();       
|         | ......                |
| Data    | DEuropa *Europa;      
|         | DDaphne *Daphne;      
|         | TStructType CurStruType;  
|         | ......                |

Figure 33. Class in the Analyzer

Figure 33 shows the composition of class DAnalyzer, which is responsible for analysis. It implements all the analysis algorithms described in Chapter 3.2.