UnitInstruments: 楽器の機能要素を再構築可能なユニット型電子楽器の設計と実装

丸山, 裕太郎
竹川, 佳成
寺田, 努
塚本, 昌彦

(Citation)
コンピュータ ソフトウェア,28(2):193-201

(Issue Date)
2011

(Resource Type)
journal article

(Version)
Version of Record

(URL)
https://hdl.handle.net/20.500.14094/0100492851

UnitInstruments: 楽器の機能要素を再構築可能なユニット型電子楽器の設計と実装

丸山 裕太郎　竹川 佳成　寺田 努　塚本 昌彦

人は音楽を奏でるために古くからさまざまな楽器を開発してきた．西洋楽器を例に挙げると，バイオリンとチェロのように共通する形状・構造・奏法をもち，音域の異なる楽器がある．また，2 段の鍵盤をもつ電子オルガンと 1 段の鍵盤しかもたないピアノのようにミクロの構造は同じでも組み合わせ方が異なる楽器も存在する．一方，電気・電子技術の発展に伴い，アコースティック楽器と同様の見た目や演奏方法をもち，電子的に音を生成する電子楽器が多数開発されてきた．しかし，従来の電子楽器は既存楽器の形状をそのまま模写することが主な目的であった．本研究では，楽器を発音や音程決定などの機能要素 (ユニット) の集合であると捉え，それらのユニットを自由に組み合わせることで，音域や演奏スタイルの変化に柔軟に対応できるユニット楽器の開発を目指す．ユニットを組み合わせて楽器を再構築することで，楽器の音域増減などのカスタマイズや，既存楽器の特徴を組み合わせた新たな楽器の創造が行える．ユニットの設定は，本研究で提案するスクリプト言語によって柔軟に記述できる．また，本研究ではユニット楽器のプロトタイプを実装し，さまざまなイベントステージで実運用を行った．

Musical instruments have a long history, and many types of musical instruments have been created to attain ideal sound production. At the same time, various types of electronic musical instruments have been developed. Since the main purpose of conventional electronic instruments is to duplicate the shape of acoustic instruments with no change in their hardware configurations, the diapason and the performance style of each instrument is inflexible. Therefore, the goal of our study is to construct the *UnitInstrument* that consists of various types of musical units. A unit is constructed by simulating functional elements of conventional musical instruments, such as output timing of sound and pitch decision. Each unit has connectors for connecting other units to create various types of musical instruments. Additionally, we propose a language for easily and flexibly describing the settings of units. We evaluated the effectiveness of our proposed system by using it in actual performances.

1　はじめに

音楽は人間の日常生活を豊かにするために，なくてはならないものである．そのため，人は音楽を奏でるために古くからさまざまな楽器を開発してきた．例えば，西洋楽器は演奏方法により管楽器，弦楽器，打楽器，鍵盤楽器の 4 つに大きく分類され，これらの中には，弦楽器に属するバイオリン，チェロのように共通する形状・構造・奏法をもつ一方，音域が異なる楽器がある．また，2 段の鍵盤をもつ電子オルガンと 1 段の鍵盤しかもたないピアノのようにミクロの構造は同じでも組み合わせ方が異なる楽器も存在する．

一方，電気・電子技術の発展に伴い，電子ギターやキーボードなどアコースティック楽器と同様の見た目や演奏方法をもち，電子的に音を生成する電子楽器が多数開発されてきた．これらは，音量や音色の変更，音域のスライドなどの多彩な機能をもつ．しかし，従来の電子楽器は既存楽器の形状をそのまま模写することが主な目的であったため，対象とする楽器がもつ音域や演奏方法などの制約をそのまま継承している．そこで，本研究では図 1 に示すように，楽器を発音や音高決定などの機能要素 (ユニット) の集合である

UnitInstruments: Easy Configurable Musical Instruments.

Yutaro Maruyama, Tsutomu Terada and Masahiko Tsukamoto, 神戸大学大学院工学研究科, Graduate School of Engineering, Kobe University.

Yoshinari Takegawa, 神戸大学自然科学系先端融合研究環, Organization of Advanced Science and Technology, Kobe University.

コンピュータソフトウェア, Vol.28, No.2 (2011), pp.193–201.
[研究論文] 2010 年 5 月 31 日受付.

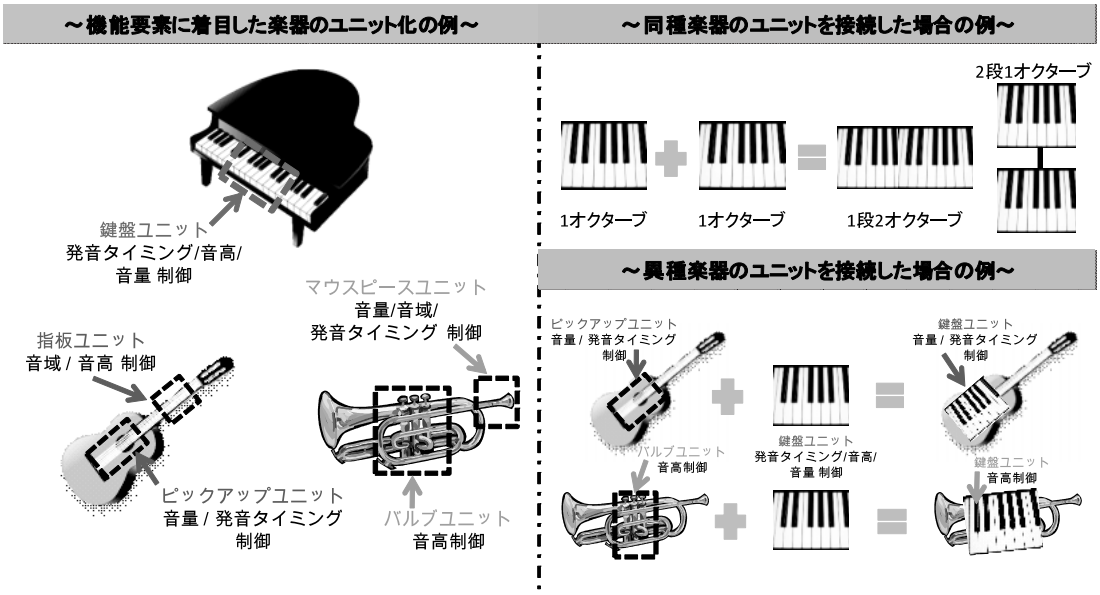

図 1　ユニット楽器のコンセプト

と捉え，それらのユニットを自由に組み合わせることで，音域や演奏スタイルの変化に柔軟に対応できるユニット楽器の開発を目指す．

本研究では具体的に 1 オクターブ分の鍵盤や 4 フレット分の指板 (弦の振動を制御して音高を変える板) というように楽器の機能要素を分割したものをユニットとする．これらを組み合わせて再構築することで，楽器の音域増減などのカスタマイズや，既存楽器の特徴を組み合わせた新たな楽器の創造が行える．同種のユニットをつないで音域や大きさをコントロールするだけでなく，例えばギターのピックアップ部 (弦の振動を検知し音を増幅する部分) を鍵盤に置き換えるというように異なる楽器のユニットを組み合わせることで今までにない新しい楽器が構築できる．また，センサやアクチュエータなど入出力機器を搭載した拡張ユニットを用いることでユーザの既存楽器の演奏技術を活かし，演奏操作を補助する．ユニットの設定は本研究で提案するスクリプト言語によって柔軟に記述できる．

本研究ではユニット楽器のプロトタイプを実装し，さまざまなイベントステージで実運用を行った．

以下，2 章で関連研究について述べ，3 章でシステムの設計，4 章で提案したスクリプト記述言語，5 章でシステムの実装，6 章で実運用について述べ，最後にまとめを述べる．

2　関連研究

単機能のユニットを組み合わせて高機能化を図る試みは多く行われている．例えば，LEGO 型ブロックを犬や飛行機などの形に組み立ててその形状をゲーム内で表示させて利用することのできるシステム [1] や，三角形の板を組み合わせてさまざまな形状を作成すると，形状に応じて Web ページを開いたり，物語を展開するシステム [2] が提案されている．これらはコンピュータとの直観的なインタラクションを目的としているため，新たな楽器の構築を目指す本研究とは異なる．一方，機能を組み合わせることで音楽を生成する取組みとして，音の強弱，短調，音色などを調節できるカードを並べ音楽を生成するもの [3] や，エフェクトを割り当てたマーカ付きブロックの位置をリアルタイムで認識し，各ブロックの位置関係によりさまざまな電子音楽を作り出すシステム [4] などがある．これらはいずれも音楽生成を目的とし，楽器演奏を支援する本研究とは異なる．従来の楽器演奏スタイルを守りつつ楽器のユニット化をコンセプトとする事例としては，1 オクターブを基本単位とする鍵盤を

組み合わせることで様々な鍵盤構造に適応できる鍵盤楽器[5]がある．これは同一の構造をもつ楽器の再構築を目的とし，異種楽器を組み合わせることで新たな楽器を構築する本研究にそのまま適応できない．また，本研究はユニット接続時の設定を柔軟に行うための枠組みをもっている点で異なる．

3 設計

3.1 設計方針

ユニット楽器は，楽器の機能要素を自由に組み合わせることによって新たな楽器を創り出すシステムである．ここで，楽器の機能要素とは発音タイミング，音高および音量など音を出力するための個々の要素を指す．例えば，ピアノは鍵一つ一つで発音タイミング，音高および音量の全てを決定している．一方で弦楽器に属するギターは指板部で弦の音高を決定し，ピックアップやサウンドホール (音を共鳴させるための穴) 上の弦を弾くことで発音タイミングと音量を決定している．このように機能要素は鍵や指板など，楽器の部位単位に 1 つまたは複数割り当てられている．本研究ではこれらの機能要素に着目し，以下の方針でシステムを設計する．

楽器の機能要素の最適なユニット化

ユニットは各楽器の部位ごとで分割し，従来の楽器の発音処理と演奏スタイルを踏襲している．各楽器の部位ごとに分割することで他のユニットとの組み合わせを柔軟に行え，既存楽器を踏襲することでユーザはこれまでに身につけた楽器演奏技術をそのまま転用できる．なお，ユニットは単体でも使用できるよう各楽器の部位において最低限必要な大きさに設計する．また，異なる楽器のユニットを組み合わせることで，今までにない新しい楽器を構築できるよう設計する．例えば，図 1 に示すように，ギターのピックアップ部分を鍵盤に置き換えることでヴィブラート奏法 (音を震わせる奏法) ができるキーボードや，鍵盤に管楽器のマウスピース (息を吹き込む部分) を接続することで，打鍵中でも音量や音域を変えられるキーボードなど，異なる楽器のユニットを組み合わせることで従来の楽器では実現できなかった演奏表現を可能にする楽器を構築できる．

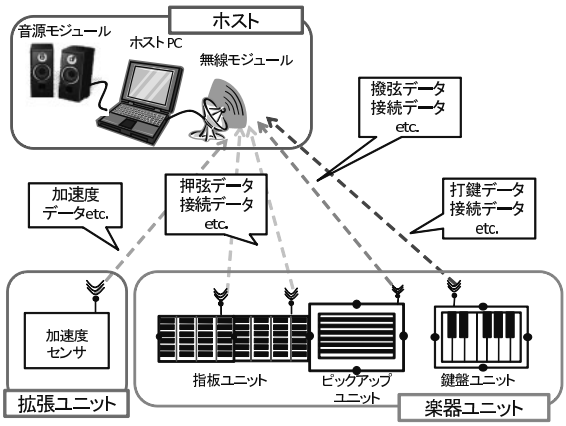

図 2 システムの構成例

自由に再構成できる柔軟性

単純に異なる楽器のユニットを組み合わせた場合，機能要素の割り当て方が複数存在したり，機能要素同士が重なりあってしまう問題が起こる．そのため，システムは各ユニットの機能要素を再設定できなければならない．そこで，本研究ではユニットの機能要素を自由に設定できるスクリプト言語を提案する．ユーザはスクリプト言語を使用することで，さまざまなユニット同士の組合せにおいて各機能要素を自由に割り当てることができる．

3.2 システム構成

図 2 にユニット楽器のシステム構成例を示す．システムはホストと楽器ユニット，拡張ユニットから構成される．

ホスト: 各ユニットの設定情報やユーザの操作などを一括して管理する．各ユニットとは無線通信で接続されている．

楽器ユニット: 楽器ユニットはさまざまな楽器の機能要素をユニット化したものを指す．ユニットの発音および音高決定，他ユニットとの接続イベントはホストに送信される．

拡張ユニット: 楽器の機能要素以外を用いてユーザの既存楽器の演奏技術を活かし，演奏操作を補助する．例えば，ギターにおけるヴィブラートやチョーキングといった動きを利用して音色にエフェクトをかける場合，加速度センサを搭載した拡張ユニットを使用

し，得られた加速度データがホストに送信される．

3.3 ユニットの設計

3.3.1 楽器ユニット

ユニット楽器は楽器の機能要素を分解，再構築することで楽器の音域増減やさまざまな楽器構造に適応できるようにすることが目的である．3.1 で述べたように，楽器の種類によって機能要素は独立および従属している．そのため，各楽器の機能要素と構造の観点からユニット化における分解能を明確にし，設計する必要がある．ここでは複数の楽器ユニットの設計例を挙げ説明する．

鍵盤ユニット

鍵盤楽器は 1 つの鍵を打鍵することで発音タイミング，音量および音高の全てを決定する．また，鍵盤楽器は C の音階を基準とした 12 個の鍵が 1 オクターブごとに連続して並んだ構造をもつ．そのため，鍵盤ユニットは C を基準とした 1 オクターブ分の鍵盤を基本単位とする．さらに，鍵盤ユニットの上下左右側面に接続コネクタを搭載し，他のユニットのコネクタと接続できるようにする．

ギターユニット

弦楽器に属するギターは指板上で弦を指や爪で押さえることで音高を決定し，サウンドホールやピックアップ付近の弦を弾くことで音量，および発音タイミングを決定する．また，一般的にギターは 4 フレットあれば単音弾きおよびコード弾きに対応できるため，指板ユニットのフレット数を 4 フレットとする．さらに，従来のギターの演奏スタイルを踏襲するため，指板ユニットの持ち幅は従来のギターのネックと同程度の太さとする．ピックアップユニットにおいても，これと同じ理由から従来のギターのピックアップ部と同程度の大きさとする．加えて，指板ユニットの高音側及び低音側の側面，ピックアップユニットの上下左右側面にコネクタを搭載し，他のユニットと接続できるようにする．

トランペットユニット

管楽器に属するピストンバルブ式トランペットはマウスピース部分で音域，音量および発音タイミングを決定し，バルブ部分で音高を決定する．そのため，トランペットユニットは息を吹き込むマウスピースユニット，指で音高を制御するバルブユニットを基本単位とする．また，マウスピースユニットの先端，バルブユニットの上下左右側面にコネクタを搭載し，他のユニットのコネクタと接続できるようにする．

3.3.2 拡張ユニット

拡張ユニットは楽器の機能要素以外を用いてユーザの既存楽器の演奏技術を活かし，ユニット楽器の演奏操作を補助できるよう設計する．例えば，ギターの機能要素である指板ユニットとピックアップユニットを切り離して使用する場合，本来ギターのボディに固定されている指板部の上で弦を指で押し上げたり，大きくスライドさせるチョーキングやグリッサンドといった演奏表現を行えない．そこで，加速度センサを用いて既存の演奏テクニックを踏襲する動きをユーザが行うことにより発音した音色にヴィブラートなどの効果を与えられる．

4 スクリプト記述言語

ユニット楽器はユーザの要求に従い，各ユニットの役割を設定できるようにする．そこで，本研究ではユニット楽器のための独自のスクリプト言語を提案し，ユーザがユニットの個別設定を自由に変更できるようにする．

4.1 スクリプト言語の設計方針

提案するスクリプト言語の設計方針を以下に記述する．

様々な種類のユニットに対応できる柔軟性

楽器ユニットは楽器の種類によって，弦，鍵およびフレットといった発音するためのトリガとなる部位の数や構造が大きく異なる．提案するスクリプト言語ではこれらを抽象化し，幅広い楽器に対応できる枠組みを構築する．

機能要素の再設定

異なる楽器ユニットを組み合わせた場合，音高やタイミングを決定する機能要素の割り当て方が複数存在したり重なりあってしまう問題が起こる．そこで，提案するスクリプト言語では各ユニットの機能要素を再設定できるよう設計する．例えば，鍵盤ユニットの

機能要素を音高決定のみに限定した場合，鍵盤ユニットを操作しても音を鳴らさず，他のユニットで発音タイミングが決定される音の音高を鍵盤ユニットの鍵で制御するというような使い方もできるようになる．

発音処理における優先度の設定

発音処理における音高やタイミングなどを決定する優先度は楽器により異なる．ギターを例に挙げると，同一弦で高音側と低音側のフレットを同時に押弦した場合，高音側の音高が採択される．一方，ピアノの場合は複数の鍵を同時に打鍵すると，打鍵した数だけの音高をもつ音が同時に発音される．したがって，提案するスクリプト言語では発音処理における優先度を定義し，ユーザの記述によって発音処理の優先度を再設定できるよう設計する．

ユニットの個別設定

スクリプト言語は音色や音階といった設定をユニットおよびトリガの両方から柔軟に記述できるよう設計する．これによりユニット単位で行った設定を一部のトリガだけ変更して再設定するといったこともできる．例えば，鍵盤ユニットの鍵全体の音色をピアノとし第 5 鍵のみ音色をストリングスに再設定したり，ピックアップユニットのチューニングを通常のチューニングから第 6 弦のみの音階を下げたドロップチューニングといった設定を簡単に行える．また，スクリプトで記述した設定をユニット構成が変更されたときに自動的に再実行できるようにする．これによりステージ中でのユニット組替えなど動的な構成変更にも追随できる．

4.2 スクリプト言語の仕様

各ユニットは鍵，弦およびフレットといった音高や発音を決定する要素を複数もつ．それらをスイッチと定義し，各スイッチはそれらが属するユニットと親子関係をもつ．また，ユニットおよび各スイッチはそれぞれ ID が割り当てられ，音色，基準音，ターゲット (音高制御などを反映させるユニットおよびスイッチ) の ID といった個別設定のパラメータをもたせる．

スクリプト言語は Unit オブジェクト，Switch オブジェクト，定数からなり，Unit オブジェクトおよび Switch オブジェクトはそれぞれメンバ変数をもつ．なお，提案したスクリプト言語では一般的な C 言語などで用いられる演算子 (代入，比較，加算，減算など) が利用できる．図 3 に Unit オブジェクトと Switch オブジェクトの相関関係を示す．また，表 1 にスクリプト言語中に使用されているデータ型，表 2 に Unit オブジェクトのメンバ変数，表 3 に Switch オブジェクトのメンバ変数を示す．例として，鍵盤ユニットの音色をストリングス，基準音を C3，第 5 鍵のみ音色をオーケストラヒットに設定したい場合，ユーザは *key_u* という Unit オブジェクトと *key_s[12]* という Switch オブジェクトの配列を定義し，*key_u.settone = STRINGS;*，*key_u.basepitch = C3;*，*key_s[4].tone = ORCHESTRA_HIT;* というようにメンバ変数に値を代入し各ユニットの設定を行う．

4.3 スクリプト記述例

マルチネックギター

図 4 にピックアップユニットと指板ユニットを組み合わせたマルチネックギター構造の例とスクリプトの記述例を示す．記述例では始めに変数，Unit および Switch オブジェクトを定義し，6 行目から 37 行目で各ユニットの初期の振る舞いを設定している．Unit オブジェクトのタイプを if 文で評価し，各タイプにより音色や音域を設定している．38 行目から 62 行目で水平方向のネック (接続された指板ユニット群) の振る舞いを設定し，62 行目から 87 行目で垂直方向のネックの振る舞いを設定している．それぞれの方向のネックに指板ユニットが接続されると，if 文で比較が行われ 46 行目から 48 行目，および 71 行目から 73 行目でターゲットとなるピックアップユニット (ID: 60) とそれに属する弦スイッチ (ID: 1〜6) を指定し音域や音色の再設定が行われる．この設定を用いると，水平方向のネックを使用して演奏する場合は，12 フレット分の音域をもち，オーバードライブギターの音色が割り当てられる．一方，垂直方向に接続されたネックを使用して演奏する場合は，8 フレット分の音域をもち，クリーンギターの音色で，オープン G チューニングが割り当てられる．

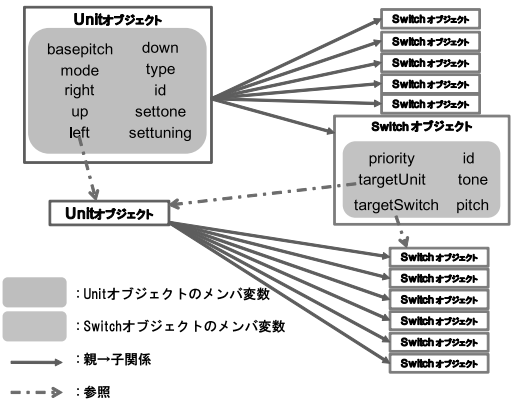

図 3　Unit オブジェクトと Switch オブジェクトの相関関係

表 1　データ型

データ型	説明
int	整数
tone	音色
pitch	基準音
mode	モード
type	ユニットのタイプ
tuning	ユニットのチューニング

表 2　Unit オブジェクトのメンバ変数

型 変数名	説明
pitch *basepitch*	ユニットの基準音
mode *mode*	ユニットの発音/音高決定モード
tone *settone*	ユニットの音色の設定
tuning *settuning*	ユニットのチューニングの設定
int *right*	右側コネクタに接続されている ユニットの ID
int *up*	上側コネクタに接続されている ユニットの ID
int *left*	左側コネクタに接続されている ユニットの ID
int *down*	下側コネクタに接続されている ユニットの ID
type *type*	ユニットのタイプ
int *id*	ユニットの ID 番号

表 3　Switch オブジェクトのメンバ変数

型 変数名	説明
int *priority*	スイッチの発音タイミング・ 音高制御の優先度
int *pitch*	スイッチの制御する音高
tone *tone*	スイッチに割り当てられた音色
int *targetUnit*	スイッチの発音・音高制御の トリガとなるユニットの ID
int *targetSwitch*	スイッチの発音・音高制御の トリガとなるスイッチの ID
int *id*	スイッチの ID 番号

スクリプト記述例

```
1.  unit u[5] = {54, 56, 60, 62, 64};
2.  switch s[5][24];
3.  int i, j, k;
4.  int left_basepitch, left_id;
5.  bool flag = true;
6.  for(i=0; i<5; i++){
7.    if(u[i].type == PICK_UP){
8.      u[i].mode = SOUND_DEC;
9.      u[i].setTuning(NORMAL);
10.     u[i].setTone(STEEL_GUITAR);
      }
11.   if(u[i].type == FINGER_BOARD){
12.     u[i].mode = KEY_DEC
13.     for(j=0; j<24; j++){
14.       s[i][j].priority = j;
15.       s[i][j].pitch = j+1;

16.     if(j>3){
17.       s[i][j].pitch += 1;
18.       s[i][j].priority -= 4;
19.     }
20.     if(j>7){
21.       s[i][j].pitch += 1;
22.       s[i][j].priority -= 4;
23.     }
24.     if(j>11){
25.       s[i][j].pitch += 1;
26.       s[i][j].priority -= 4;
27.     }
28.   if(j>15){
29.       s[i][j].priority -= 4;
30.     }
31.   if(j>19){
32.       s[i][j].pitch += 1;
33.       s[i][j].priority -= 4;
34.     }
35.   }
36. }
37. }
38. for(i=0; i<5; i++){
39.   if(u[i].type == PICK_UP){
40.     left_id = u[i].id;
41.     if(u[i].right != 0){
42.       for(k=0; k<5; k++){
43.         if((u[k].left == left_id)&&(u[k].type
        ==FINGER_BOARD)){
44.           u[k].basepitch = E3;
45.           for(j=0; j<24 ;j++){
46.             s[k][j].targetUnit = 60;
47.             s[k][j].tone = OVERDRIVE_GUITAR;
48.             s[k][j].targetSwitch = s[k][i][j] /
          4;
49.           }
50.       if(u[k].right != 0){
51.         u[k].basepitch = left_basepitch - 4;
52.         left_id = u[k].id;
53.         left_basepitch = u[k].basepitch;
54.       }else{
55.         u[k].basepitch = left_basepitch - 4;
56.         flag = false;
57.       }
58.     }
59.   }
60. }
61. }
62. }
63. for(i=0; i<5; i++){
64.   if(u[i].type == PICK_UP){
65.     left_id = u[i].id;
66.     if(u[i].up != 0){
67.       for(k=0; k<5; k++){
68.         if((u[k].left == left_id) &&(u[i].up
          == u[k].id) &&(u[k].type ==
          FINGER_BOARD) ){
69.           u[k].basepitch = E3;
70.           for(j=0; j<24 ;j++){
71.             s[k][j].targetUnit = 60;
72.             s[k][j].tone = JAZZ_GUITAR;
73.             s[k][j].targetSwitch =
            s[k][i].id / 4;
74.           }

75.       if(u[k].right != 0){
76.         u[k].basepitch = left_basepitch -
            4;
77.         left_id = u[k].id;
78.         left_basepitch = u[k].basepitch;
79.       }else{
80.         u[k].basepitch = left_basepitch -
            4;
81.         flag = false;
82.       }
83.     }
84.   }
85. }
86. }
87. }
```

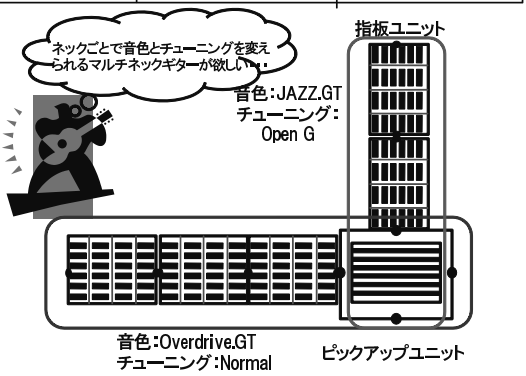

図 4　マルチネックギターの記述例

ギターのように押弦し鍵盤で打鍵して演奏する楽器

図 5 に鍵盤ユニットと指板ユニットを組み合わせた構造の例とスクリプトの記述例の例を示す．記述例では始めに変数，Unit および Switch オブジェクトを定義し 4 行目から 42 行目で各ユニットの初期の振る舞いを設定している．Unit オブジェクトのタイプを if 文で評価し，各タイプにより音色や音域を設定している．43 行目から 80 行目で指板ユニットの押弦スイッチと鍵盤ユニットの鍵と対応させる設定を行っている．48 行目の if 文で鍵盤ユニットの左側コネクタに指板ユニットが設定されたかどうかを評価し，真ならば，指板ユニットの各スイッチに対応するターゲットの鍵盤ユニット (ID:54) や鍵スイッチの振る舞いの再設定を行う．この設定を用いると，鍵盤ユニットの 6 つの白鍵に 1 弦ずつ割り当て，指板で音高決定をし，

スクリプト記述例

```
1.  unit u[5] = {54, 58, 61, 63, 64}
2.  switch s[5][24];
3.  int i, j, k;
4.  for (i=0; i<5; i++){
5.    if(u[i].type == KEYBOARD){
6.      u[i].tone = PIANO;
7.      u[i].mode = SOUND_DEC;
8.      u[i].basepitch = 0;
9.      for(j=0; j<12; j++){
10.       s[i][j].pitch = j;
11.       s[i][j]. priority = 0;
12.     }
13.   }

14.   if(u[i].type == FINGER_BOARD){
15.     u[i].mode = KEY_DEC
16.     for(j=0; j<23; j++){
17.       s[i][j].priority = j;
18.       s[i][j]. pitch = j+1;

21.   if(j>3){
22.     s[i][j]. pitch += 1;
23.     s[i][j]. priority -= 4;
24.   }
25.   if(j>7){
26.     s[i][j]. pitch += 1;
27.     s[i][j]. priority -= 4;
28.   }
29.   if(j>11){
30.     s[i][j]. pitch += 1;
31.     s[i][j]. priority -= 4;
32.   }

33.   if(j>15){
34.     s[i][j]. priority -= 4;
35.   }
36.   if(j>19){
37.     s[i][j]. pitch += 1;
38.     s[i][j]. priority -= 4;
39.   }
40.  }
41.  }
42.  }
43.  for(i=0; i<5; i++){
44.  if(u[i].type == KEYBOARD){
45.    left_id = u[i].id;
46.    if(u[i].left != 0){
47.  for(k=0;k<5;k++){
48.    if((u[k].left == left_id)
       &&(u[k].type== FINGER_BOARD) ){
49.      u[k].basepitch = E3;
50.      for(j=0; j<12 ;j++){
51.        s[k][j].targetUnit = 54;
52.        s[k][j].tone =
     OVERDRIVE_GUITAR;
53.      if(s[k][j].id<4)
54.        s[k][j].targetSwitch = 0;
55.      else if (s[k][j].id <8)
56.        s[k][j].targetSwitch = 2;
57.      else if (s[k][j].id <12)
58.        s[k][j].targetSwitch = 4;
59.      else if (s[k][j].id <16)
60.        s[k][j].targetSwitch = 5;

61.    else if (s[k][j].id <20)
62.      s[k][j].targetSwitch = 7;
63.    else if (s[k][j].id <24)
64.      s[k][j].targetSwitch = 9;
65.    }

66.    if(u[k].right != 0){
67.      u[k].basepitch =
     left_basepitch - 4;
68.      left_id = u[k].id;
69.      left_basepitch =
     u[k].basepitch;
70.    }else{
71.      u[k].basepitch =
     left_basepitch - 4;
72.      flag = false;
73.    }
74.  }
75.  }
76.  }
77.  }
78.  }
```

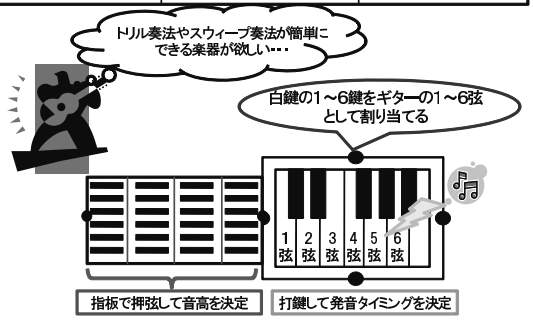

図 5 ピックアップを鍵盤に置き換えたギター

打鍵して発音を行う楽器が構築できる．この楽器は音を交互に出すトリル奏法や全ての弦を掃くように演奏するスウィープ奏法などが容易にできる．

5 実装

ユニット楽器のプロトタイプの構成を図 6 に示す．プロトタイプはホスト，鍵盤ユニット，ギターユニットである指板ユニットとピックアップユニット，および拡張ユニットで構成される．ギターユニットは YAMAHA 社の EZ-AG を 4 フレットごとに切断し，ネック部分の押弦スイッチやピックアップ部分の弦および振動センサを制御する回路を独自に製作して使用した．また，無線モジュールはアローセブン社の UM-100 を使用した．指板ユニットおよびピックアップユニットの制御には MICROCHIP 社の PIC16F877A を用いて行った．PIC のプログラミングは MICROCHIP 社の MPLAB 上で CCS 社の PIC C コンパイラを用いた．ホストは Panasonic 社

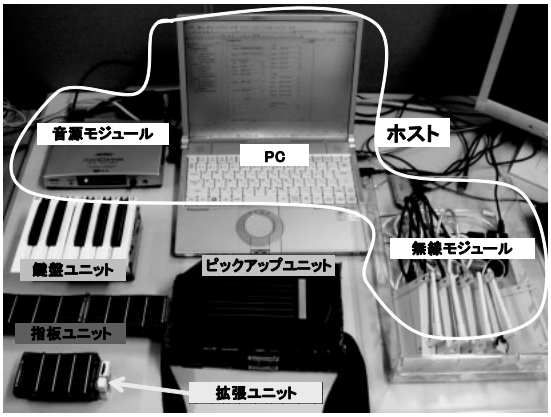

図 6 ハードウェア構成

のノートパソコン (CF-Y7, 2.1GHz, 1.5GB) を使用し，PC 上のソフトウェアの開発は Windows XP 上で Microsoft Visual C++ 2005 を用いて行った．拡張ユニットに搭載した加速度センサはワイヤレステクノロジー社の WAA-001 を使用した．ホストは，各ユニットのコンフィギュレーションデータの管理などを行うための CPU やメモリ，スクリプトを記述するためのアプリケーション，各ユニットと通信するための無線モジュール，発音を行うための音源モジュールをもつ．ホストの機能は，各ユニットのデバイス情報の管理，各ユニットからのイベント情報をもとにした音源モジュールからの発音処理，各ユニットの役割や音色，およびキー設定や接続先のユニットの役割に対するスクリプト設定の 3 つある．なお，プロトタイプにおけるスクリプト処理は，アプリケーションに記述したスクリプトコードをソースコードに追加し，開発環境のコンパイラを通して実行する．鍵盤ユニットのハードウェアに関しては文献 [5] で実装したものを用いた． 指板ユニットは，マイコン，24 個の押弦スイッチ，左右側面に接続関係を構築するためのコネクタ，無線モジュールで構成される．また，鍵盤ユニットと同様に，コネクタはマグネット式コネクタを自作した．なお，軽量化のため指板ユニットには電源を搭載せず，別途接続する電源ユニットから電源を供給する．ピックアップユニットは，マイコン，振動を検出する 6 つの振動センサおよび弦，上下左右側面に接続関係を構築するためのコネクタ，無線モジュールで

構成される．また，コネクタはマグネット式コネクタをもつ．拡張ユニットは加速度センサを用いて，スクリプトにより発音した音色に動きや姿勢でエフェクトをかける機能を実現した．

6 実運用

ユニット楽器の有効性を検証するために，さまざまなイベントステージでプロトタイプシステムの実運用を兼ねたステージパフォーマンスを行った．

2008 年度神戸ルミナリエ市民ステージ

2008 年 12 月 13 日および 14 日に行われた，神戸ルミナリエのイベントステージにて実運用を行った．図 7 に実運用の様子を示す．ステージでは，ユニット楽器を用いて，歌と演奏によるショーパフォーマンスを行った．プロトタイプは LED ユニットを搭載した指板ユニット 1 個およびピックアップユニットおよび鍵盤ユニットで構成された．なお，異種ユニットの組合せは実装には至っておらず，2 人のパフォーマーがそれぞれを個別に用いて演奏を行った．ピックアップユニットと指板ユニットが独立しているという特性を活かすことで，左手を動かしながら演奏するというこれまでにないパフォーマンスを行えた．一方で，指板ユニットとピックアップユニットを切り離して使用した場合，左手が固定されていないためコード弾きと単音弾きの切り替えがスムーズにできないことがあった．この問題は指板ユニットの 1 つの押弦スイッチにギターコードを割り当てるというように，提案したスクリプト言語を用いて各ユニットに最適な設定をすることで解決できると考えられる．

図 7　2008 年度神戸ルミナリエ市民イベントステージ

図 8　塚本研究室 5 周年記念イベントステージ

神戸大学塚本研究室 5 周年記念イベント

2009 年 10 月 30 日に行われた神戸大学塚本研究室 5 周年記念イベントにてステージパフォーマンスを行った．図 8 にステージでの実運用の様子を示す．ステージでは 2 つの鍵盤ユニットおよび指板ユニットとピックアップユニットのそれぞれ 1 つずつから構成されるギターユニットを用いて，BGM に合わせて演奏を行った．2 人のパフォーマーは鍵盤ユニットを 1 つずつ持ち，それらを楽曲に合わせて脱着させることで音色や音域を変化させながら演奏を行った．また，他の 1 人のパフォーマーも同様に指板ユニットとピックアップユニットを楽曲に合わせて脱着させ，音色と音域を変化させながら演奏を行った．一方で，自作のハードウェア回路の強度が不十分だったため途中で電源が落ちるトラブルが起こった．

2009 年度神戸ルミナリエ市民ステージ

2009 年 12 月 12 日に行われた神戸ルミナリエのイベントステージにて，再びステージパフォーマンスを行った．図 9 にステージでの実運用の様子を示す．ステージでは 1 つの鍵盤ユニット，指板ユニットおよびピックアップユニットのそれぞれ 1 つずつから構成されるギターユニットを使用し，BGM に合わせて演奏を行うパフォーマンスを行った．ステージの序盤から中盤にかけては 2 人のパフォーマーが鍵盤ユニット

図 9 2009 年度神戸ルミナリエ市民ステージ

とギターユニットをそれぞれ別々に演奏し，ギターユニットに関しては指板ユニットとピックアップユニットを脱着させ音域や音色を変化させながら演奏を行った．ステージ終盤ではギターユニットのピックアップユニットを鍵盤ユニットに取り替え，スウィープ奏法やトリル奏法を用いた演奏を行った．異なる楽器の要素を組み合わせることで難度の高い奏法を容易に行えることを確認できた．

情報処理学会創立 50 周年記念全国大会

2010 年 3 月 8 日から 12 日に行われた情報処理学会創立 50 周年記念全国大会にて，ステージパフォーマンスを行った．ステージでは 1 人のパフォーマーが鍵盤ユニット，指板ユニットおよびピックアップユニットをそれぞれ 1 つずつ使用し，BGM に合わせて演奏を行うパフォーマンスを行った．ステージ序盤から中盤にかけては鍵盤ユニットとギターユニットをそれぞれ別々に演奏した．ステージ終盤ではギターユニットのピックアップユニットを鍵盤ユニットに交互に取替え，音色や演奏スタイルを動的に変化させるパフォーマンスを行えた．一方で自作コネクタの強度が不十分だったためユニットの脱着が認識されないことが起こった．この問題を解決するには安定したハードウェアの実装が課題であると考えられる．

7 おわりに

本研究では楽器を発音や音程決定などの機能要素 (ユニット) の集合であると捉え，それらのユニットを自由に組み合わせることで，音域や演奏スタイルの変化に柔軟に対応できるユニット楽器を構築した．また，ユニット楽器の設定を自由に行えるスクリプト記述言語を提案した．ユニット楽器を組み合わせて再構築することで，楽器の音域増減やさまざまな楽器構造への適応を実現できる．今後の課題としては，打楽器や管楽器などの他の構成要素をもつ楽器ユニットやさまざまな楽器の演奏技法を再現できる拡張ユニットの実装があげられる．また，本研究で提案したスクリプト言語は数十の楽器構成のパターンを考え，それらを自然に記述できるものとして作成した．そのため，今後の新たなユニットの提案に伴い，設定や役割の割り当てアルゴリズムを改良するといったさらなる拡張が必要である．加えて，各ユニットのハードウェア特性の評価実験および使用評価実験を行う予定である．

謝辞

本研究の一部は，中山隼雄科学技術文化財団研究助成の支援によるものである．ここに記して謝意を表す．

参 考 文 献

[1] Anderson, D., Frankel, J., Marks, J., Agarwala, A., Beardsley, P., Hodgins, J., Leigh, D., Ryall, K., Sullivan, E. and Yedida J.: Tangible Interaction Graphical Interpretation: A New Approach to 3D Modeling, in *Proceedings of Special Interest Group on Computer GRAPHics* (*SIGGRAPH2000*), 2000, pp. 393–402.

[2] Gorbet, M., Orth, M. and Ishii, H.: Triangles: Tangible Interface for Manipulation and Exploration of Digital Information Topography, in *Proceedings of Computer-Human Interaction* (*CHI1998*), 1998, pp. 49–56.

[3] Henry, N. D., Nakano, H. and Gibson, J.: Block Jam, in *Proceedings of Special Interest Group on Computer GRAPHics* (*SIGGRAPH2002*), 2002, p. 67.

[4] Kaltenbrunner, M., Jorda, S., Geiger, G. and Alonso, M.: The reacTable: A Collaborative Musical Instrument, in *Proceedings of Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise* (*WETICE2006*), 2006, pp. 406–411.

[5] 竹川佳成, 寺田努, 西尾章治郎: さまざまな演奏スタイルに適応可能な電子鍵盤楽器 UnitKeyboard の設計と実装, 日本ソフトウェア科学会論文誌, インタラクティブソフトウェア特集, Vol. 26, No. 1 (2009), pp. 38–50.