VISCO — Visual SALIERI Components

Jürgen Kilian[*]
Department of Computer Science,
Darmstadt University of Technology,
Wilhelminenstr. 7
64283 Darmstadt, Germany
kilian@iti.informatik.tu-darmstadt.de

Holger H. Hoos[*†]
Department of Computer Science
University of British Columbia
2366 Main Mall
Vancouver, BC, V6T 1Z4, Canada
hoos@cs.ubc.ca

Abstract

VISCO is a novel visual extension to the texbased SALIERI Language. VISCO makes it possible to use the SALIERI System for creating nearly any kind of score level or real-time music applications with integrated graphical user interfaces. The visual control elements and environments can be created and controlled with the integrated, interactive, GUI based VISCO Browser, from within running SALIERI programms or from the SALIERI System Dialog (command line interface).

Introduction

The SALIERI System is an interactive software environment for structure oriented composition, manipulation, and analysis of music (Hoos et al, 1998b). The system is built on the SALIERI Language, a universal programming language combining features of traditional functional and procedural programming languages with powerful concepts for manipulating musical material. The SALIERI System has been used in different application areas, such as score level analysis, editing and creation of music.

While the SALIERI System provides a graphical user-interface, the SALIERI Language is inherently a textual programming language. Although this is fully adequate for many applications, in certain situations it is desirable to allow SALIERI programs to have their own, application specific graphical user-interfaces (GUIs). For instance, this is the case for educational applications, where often fairly limited functionality has to be presented in a way which is easily accessible to relatively inexperienced users. Furthermore, complex SALIERI applications often benefit from carefully designed grapical user-interfaces. Finally, SALIERI programs for processing real-time MIDI data are often easier and more adequately controlled through graphical control elements.

Based on this motivation, we developed VISCO (Visual SALIERI Components), an extension of the SALIERI System for creating graphical user-interfaces for SALIERI programs. This approach combines the advantages of the text-based SALIERI Language, which is adequate for a wide range of musical applications and at the same time easy to learn and to use, with the well-known benefits realised by graphical user-interfaces. Because of the advantages of the SALIERI Language over conventional programming languages (see (Hoos et al, 1998b) for a more detailed discussion), we see significant advantages in creating musical applications with SALIERI and VISCO instead of using conventional programming languages, such as C++ or BASIC, and their respective extensions for building GUIs. To optimally integrate the GUI support into the SALIERI Language and System, to overcome some disadvantages of other packages, and to enhance VISCO by some special features that make it easy and intuitive to use, we decided to build VISCO “from scratch” (i.e., directly on top of the operating system programming interface) instead of using or adapting existing visual packages like TCL/TK (Ousterhout, 1994).

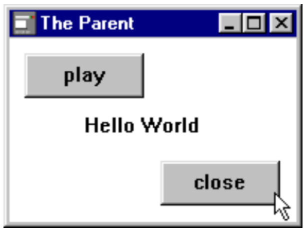


Figure 1: Simple example for a VISCO GUI.

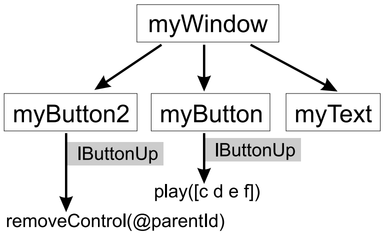


Figure 2: CE Hierarchy for the simple VISCO GUI shown in Fig. 1.

VISCO Architecture

The VISCO design is based on a reactive metaphor typical for many graphical user-interface designs, which allows grapical control elements (such as a button) to trigger actions (e.g., execution of a SALIERI function). There are three main types of objects in VISCO: windows, control elements, and actions. The windows and control elements (CEs) are organised in a hierarchical model, where windows can have lower-level windows and/or CEs as children. A typical example would be a window containing two buttons and a textfield (see Figures 1 and 2). In this model, windows can be regarded as special CEs which are typically used for logically grouping together and graphically placing other CEs.

Each control element is associated with a unique integer

---

[*] http://www.informatik.tu-darmstadt.de/AFS/SALIERI

[†] Also affiliated with the Department of Computer Science, Darmstadt University of Technology, Wilhelminenstr. 7, 64283 Darmstadt, Germany.

identifier (handle) which is generated when the CE is created and is then used for referencing it. Control elements have various properties, such as their location on the screen or within a parent window, their dimension, etc. These properties can be specified when the respective window or CE is created, but can also be modified later on.

Actions are SALIERI command strings which are associated with CEs by means of triggers. Typical triggers include pressing or releasing a button or selecting an entry from a list of items. In a certain sense, the actions which can be triggered in various ways by a CE can be regarded as properties of this CE. However, in the VISCO model, actions and their triggers play a more prominent role than other properties, and are therefore handled slightly differently. For example, like CEs, actions have their own, unique handles.

Because VISCO is realised by extending the SALIERI Language with a small number of new functions and commands, VISCO CEs cannot only be used to control textual SALIERI programs, but textual SALIERI programs can also be used to generate or modify graphical user-interfaces built with VISCO.

VISCO **Implementation**

VISCO is implemented as a SALIERI Integrated Application Module (SIAM), which allows the VISCO Module to be loaded into or removed from the SALIERI System at runtime. Like other SIAMs, such as the SALIERI Workspace Browser, or the FERMATA Module for importing MIDI files (Kilian, 1997), the VISCO Module communicates with other components of the SALIERI System via the text-based ORCA (Open Reactive Components Architecture) Protocol, which was developed by us along with the SALIERI System. From a software-engineering perspective, this design allows the VISCO Module to be fairly independent from the rest of the SALIERI System, while from a user viewpoint the integration is seamless. All VISCO functionality has been made accessible from the text-based SALIERI Language by extending it with a few new functions, which are used to create and delete CEs and to control their properties. Table 1 lists all SALIERI functions provided by VISCO.

addWindow	Creates a new VISCO window and returns a unique handle for the newly created CE.
removeWindow	Removes a VISCO window from the hierarchy
addControl	Creates a CE as child of a previous created VISCO window and returns the handle of the CE (see Table 2 for control element types)
removeControl	Removes a CE including all its descendant CEs. The handle of the CE to be removed is passed as a parameter.
setProperty	Sets a property value of a CE. Some properties are only available for certain CE types (e.g. caption or hScroll)
getProperty	Returns a single property value of a CE.
addAction	Defines an action string which will be executed if the specified trigger event occurs for the selected CE. Returns a unique handle for the action.
triggerAction	Simulates a trigger-event and executes the coresponding action of a CE.
removeAction	Removes a specific action from a given CE, where the handles of the action and the CE are passed as parameters.

Table 1: SALIERI language extensions

window	Window (in particular, top-level window)
button	Simple button including a caption string
edit	Multi-line edit box with scroll bars
text	Static text
lineEdit	Single-line edit box with accept and dismiss button
listBox	Listbox
selector	Combobox
hSlider	Horizontal slider
vSlider	Vertical slider
noteViewer	Noteview window for score display of musical objects in GUIDO Music Notation (Hoos et al, 1998a)

Table 2: VISCO control elements

For example, a new top-level VISCO window (caption=The Parent, x=1, y=1, width=200, height=150) can be created with a single SALIERI command line:

```
myWindow := addWindow(”The Parent”,1,1,200,150);
```

VISCO supports the most common control element types (buttons, editboxes, listboxes, etc.) as well as some specialized control element types (such as noteviewers or line-edits fields) which are useful for creating adequate GUIs for musical applications. A complete list of CE types currently supported by VISCO can be found in Table 2. For example, a textfield as child of ”myWindow” is created by:

```
myText := addControl(myWindow,”text”,
                     ”Hello World”,50,50,100,30);
```

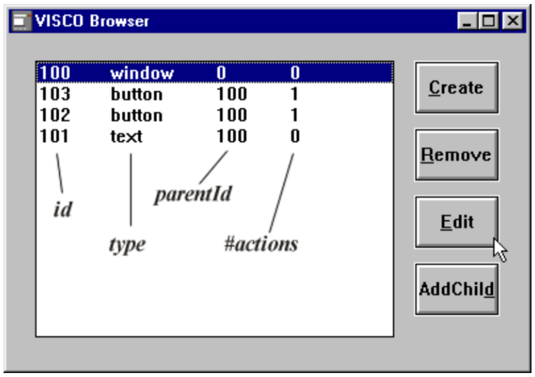
With two more simple commands we can add two buttons to our example GUI (see also Fig. 1):

```
myButton := addControl(myWindow,”button”,
                       ”play”,10,10,80,30);
myButton2 := addControl(myWindow,”button”,
                       ”close”,-90,-40,80,30);
```

VISCO allows the user create CEs and frontends in three different ways:

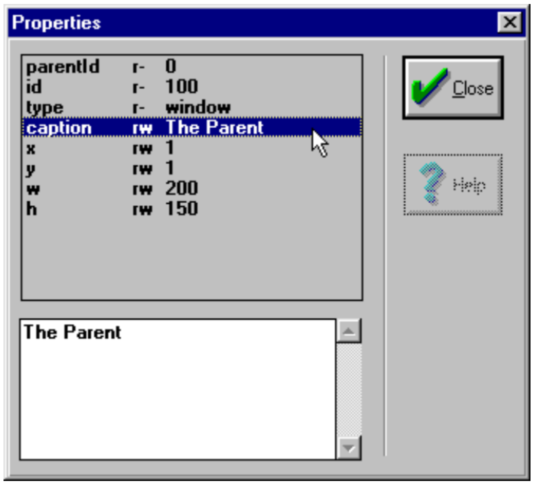
1. from the SALIERI command-line interface;
2. by SALIERI programs, these can also be called (triggered) from VISCO control elements.;
3. interactively by using the integrated VISCO Browser. The browser displays the current control element hierarchy and gives access to several dialog boxes for creating control elements, changing their properties and controling their trigger and action settings (see Fig. 3,4, and 5). The source code of a complete control element hierarchy can be exported as SALIERI language code. By executing the exported SALIERI program the hierarchy can be recreated again.

All control elements have user-defineable properties. Some of these (such as id, parentId, type, x, y) are uniformly applicable to all types of control elements; others only exist for some CE types (e.g. min/max for sliders). The basic properties (id, parentId and type) are read-only and cannot be changed after a control element has been created. For the x/y-coordinate and the width/height properties we implemented a special, very useful semantics: positiv values



100	window	0	0
103	button	100	1
102	button	100	1
101	text	100	0

Figure 3: VISCO Browser, showing the CEs of the example given in Fig. 1.



parentId	r-	0
id	r-	100
type	r-	window
caption	rw	The Parent
x	rw	1
y	rw	1
w	rw	200
h	rw	150

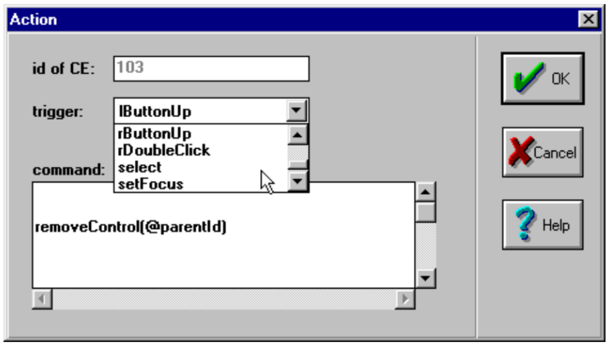
The Parent

Figure 4: Property dialogbox, showing the properties of the main window from the example in Figure 1. Some properties can be modified (marked rw), others can be inspected only (marked r-).

are relative to the left/upper corner of the parent window and negativ values are relativ to the right/lower corner of the parent window. Some control elements provide the special connectId property which allows two control elements (e.g. a hSlider and a edit) to be connected and synchronised. For example, in case of connected hSlider and edit the editbox will always display the current value of the slider and vice versa.

Each control element can also be associated with a set of actions which will be executed when selectable trigger events occur. The actions can be any kind of SALIERI commands or function calls to built-in and user-defined functions. All property values (e.g. handle or slider position) can referenced within SALIERI code of the action using the shortcut notation @prop, where prop is the name of a property of the CE associated with the action. For example, the following two commands extend the example from Fig. 1 with actions associated with the two buttons:

```
myAction := addAction(myButton,”lButtonUp”,
                                        ”play([c d e f ])”);
myAction2 := addAction(myButton2,”lButtonUp”,
                                ”removeControl(@parentId)”);
```

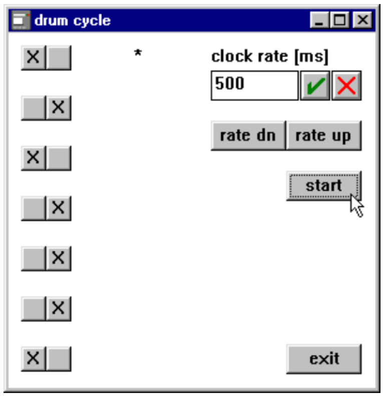


id of CE: 103

trigger: lButtonUp

command: removeControl(@parentId)

Figure 5: Action dialogbox, allows to inspect and edit the actions and triggers associated with a CE.



clock rate [ms]: 500

Figure 6: Example application “drum cycle” (see text).

The resulting GUI will execute the SALIERI command-line ”play[c d e f]” upon the left-mouse-button-up event at button ”myButton”, i.e., when the button is released after a click; upon a left-mouse-button-up event at ”myButton2”, the parent window of ”myButton2”, which is the top-level application window, will be closed. Note that at runtime, ”@parentId” will be textually replaced by the current handle of the button’s parent.

Application Example

In the meanwhile, we have used VISCO to add GUIs to many existing SALIERI applications. These applications range from educational software (e.g., an ear-trainer) and real-time software (such as a step sequencer) to musicological software for generating variations and analysing harmonic progressions. Due to space limitations, here we describe a SALIERI real-time application called “Drum Cycle”. This application is a kind of step sequencer, where at each step one of two selectable drum instruments are triggered by a real-time MIDI message. For each step, the drum instrument to be played can be selected via buttons of the VISCO user interface. Furthermore, the user-interface allows to control the trigger speed (step delay) via a line edit box as well as by a rate up/down button combination. Figure 6 shows the user-interface of this simple application.

The VISCO main window for the drum cycle application is created by:

```
hMain:=addWindow(”drum cycle”,10,10,300,310);
```

Next, we create the rate up button with upper/right corner 10/70 pixel relativ to upper/right corner of ”hMain”:

```
hBrup := addControl(hMain,”button”,
                         ”rate up”,-10,+70,-60,+24);
```

The rate down button is created analogously. Upon left-mouse-button-down at the button, a rate up function should be called:

```
hArup := addAction(hBrup,”lButtonDown”,”rup()”);
```

The “rup” function decreases the global variable “period” and changes the “text” property of “hBrate”:

```
rup := ‘if (‘period>50’,‘period := period - 50’);
          setProperty(hBrate,”text”,STR(period));’;
```

Now, we create the lineEdit CE for the trigger speed control and place its upper/right corner 10/30 pixel relativ the upper/right corner of the main window:

```
hBrate := addControl(hMain,”lineEdit”,
                              ”500”,-10,+30,-120);
```

If the accept button is pressed, “period” is set to the current input value of the lineEdit:

```
hArate := addAction(hBrate,”accept”,
                             ”period:=INT(@text)”);
```

In this action string, “@text” will be replaced by the current input value of line-edit field “hBrate” at runtime. The exit button of the window is created analogously to the close button in the example of Fig 1.

Next, we define a macro which will allow us to generically create two instrument select buttons for each sequencer step:

```
initDrum := MACRO(1,
‘n:=$1; (*seq step*)
hBdrum_a@n := addControl(hMain,”button”,
                              ”X”,10,hBdry@n,20,20);
hAdrum_a@n := addAction(hBdrum_a@n,
                 ”lButtonDown”,”slcDrum_a(”+STR(n)+”)”);
hBdrum_b@n := addControl(hMain,”button”,
                              ” ”,30,hBdry@n,20,20);
hAdrum_b@n := addAction(hBdrum_b@n,
                 ”lButtonDown”,”slcDrum_b(”+STR(n)+”)”);
```

The buttons created by this macro are connected to two other macros, ”slcDrum_a/b”, which perform the actual instrument selection for the step specified by the parameter. Additionally, the label (“caption”) of the respective button is changed to an “X” to indicate that the corresponding drum instrument has been selected. This exemplifies how actions triggered by the GUI can be used to modify the GUI itself.

```
slcDrum_a := MACRO(1,
‘n := $1;
setProperty(hBdrum_a@n, ”caption”,”X”);
setProperty(hBdrum_b@n, ”caption”,” ”);
lde@n := kickEv;’);
```

Finally, we have to implement the clock which periodically triggers the drums and animates the GUI, where an asterisk ($*$) marks the current step in the drum sequence. For this, we define a real-time event handler function “hClocks” which is triggered by an incoming clock event. The handler function updates the display by modifying the ”y” property of “hBbeat” (a text field containting the asterisk mark), sends the MIDI event to trigger the drum instrument, and finally sends a delayed message to trigger itself for the next step:

```
hClocks:=HANDLER(‘
if(‘state=1’,‘
   setProperty(hBbeat, ”y”, lbiv@drumPos);
   $t := now();
   sendEvent(lde@drumPos,0,$t);
   sendEvent (EVENT(2,0,0),1,$t+period);
   drumPos := (drumPos mod numDrums)+1;’);
’);
```

This example shows how CE properties can be controlled by Visco actions, by real-time events, and from within functions of a running Salieri program. The GUI of the “drum cycle” application is shown in Figure 6; with a few more lines of code, the application can be easily extended to handle more drum instruments at each sequencer step. The complete source code of this example and other Visco demo applications can be found at the SALIERI website (http://www.informatik.tu-darmstadt.de/AFS/SALIERI.).

Conclusions and Future Work

We introduced Visco, an extension of the Salieri Language and System which allows the realisation of graphical user-interfaces for textual Salieri programs. Visco was designed and implemented in such a way that it is simple and intuitive to use while providing the flexibility and power to design adequate GUIs for a broad range of musical applications. One of the challenges was the seamless integration of the desired functionality with other aspects and components of the Salieri Language and System, such as its interactive nature or realtime functionality. Visco provides multiple mechanisms for specifying and editing GUIs which support different approaches to realising GUIs. In its present state, we found Visco very useful for implementing simple GUIs for a variety of Salieri applications.

In the future, we plan to extend Visco with additional control elements and functionality. One direction we are pursuing in this context is to make Visco powerful enough to support a visual programming environment for the Salieri language. Somewhat different from other visual programming systems for music, such as MAX (Puckette, 1991), PatchWork (Assayag, 1995), OpenMusic (Assayag et al, 1999), we are aiming for combining the advantages of a text-based musical programming language and a visual programming interface. We believe that this can be achieved by building the visual programming environment on top of the existing text-based Salieri Language and supporting automatic conversions between graphical and textual program representations.

References

G. Assayag; Visual Programming in Music; Proceedings of ICMC’95, ICMA, San Francisco, 1995

G. Assayag , C. Rueda, M. Laurson, C. Agon, O. Delerue; Computer Assisted Composition at Ircam: PatchWork and OpenMusic; Computer Music Journal 23:3, 1999

H.H. Hoos, K.A. Hamel, K. Renz, J. Kilian; The GUIDO Music Notation Format - A Novel Approach for Adequately Representing Score-level Music; Proceedings of ICMC’98, p.451-454, ICMA, San Francisco, 1998

H.H. Hoos, J. Kilian, K. Renz, T. Helbich; Salieri - A General, Interactive Computer Music System; Proceedings of ICMC’98, p.385-392, ICMA, San Francisco, 1998

J. Kilian; FERMATA-Flexible Tempo Detection for MIDI-File Quantization; Darmstadt University of Technology, Tech. Report TI-32/97, 1997

J.K. Ousterhout; Tcl and the Tk Toolkit; Addison Wesley, 1994

M. Puckette; Combining Event and Signal Processing in the Max Graphical Programming Environment; Computer Music Journal 15(3);p. 68-77; MIT 1991