Pictures that change themselves: graphics programming languages to facilitate GUI building

Todd Cochrane
School of Information Technology
Wellington Institute of Technology
Petone, NZ
Todd.Cochrane@weltec.ac.nz

Paul Lyons
Institute of Information Sciences and Technology
College of Science, Massey University
Palmerson North, NZ

Reflection provides facilities in a computer programming language that specify inspection of the program as it is running. The "computing has an accurate representation of itself and the computation is consistent with its representation"(Lorenz, 2003). Reflection can change the execution of a part of the program by possibly modifying components of the program to match a condition as the program run. "Reification is the process in which information is 'raised' to a level at which reflection can occur" (Danvy, 1988).

Purpose built graphics programming languages produced from the late 1960s, for example "General Purpose Graphics Language" (Kulsrud, 1968), to the present day, for example "Flash Action Script" (Macromedia, 2004) while not explicit about the use of reflection, provide for reflection by some aspect of the computing undertaken by the running programs. The degree to which reflection is available determines the level to which the graphics programming language can be used to describe the sophisticated changes required by graphical user interfaces. A graphic language can be described in terms of a set of drawing objects "drawbles" representing commands that draw pictures. Instances of drawbles can be composed to represent the drawing of a picture. Adding values, expressions, variables and drawbles, that facilitate reflection allows the graphical language to describe pictures that change themselves. When these pictures are "hooked" into input event handlers they then provide a consistent and succinct representation of displays in the GUI.

Keywords

Pictures, Graphical User Interface, Hypermedia, Programming Language, Reflection, Object Oriented Programming

1. INTRODUCTION

An understanding of requirements of graphical user interface (GUI) programming or development tools becomes more significant as expression or development of "interactive" or "interactivity" in digital form (media) becomes more prevalent. Modelling of behaviour at the GUI can be expressed in the UML (Object Management Group , 2003) using State Charts , Activity Diagrams, and swim lane style timing diagrams. A key feature of the models produced with these notations is that "behaviour" is reactive, that is a response to some event that arises, or a response to a condition that occurs as a result of system or user actions. The behaviour, while consistent with the expected behaviour for a given action, for example dragging and dropping an icon onto a trash can in the MacIntosh desktop, will depend on the state of the running computer program when the action is taken. The behaviour as a result of an action taken in a set of states is modelled in an abstract way by UML notations, allowing consideration of alternative solutions prior to detailed modelling and coding. When taken to a very high level of refinement the models describe detailed changes in the graphical part of the GUI, for example the highlighting of the trash can as an item is dragged onto it. The behaviour indicates a change in the display, or precisely the "graphical display" of the graphical user interface. The state of the running program prior to this change is also indicated by the "graphical display". Interaction is indicated by a series of changes in the graphical display.

The development of a tool that assists with the production of this series of changes in the graphical display, has lead to the graphical display being treated as a picture that draws itself; the picture revealing and hiding some aspects, while adding deleting and updating other aspects, depending on the behaviour of the computer program. The part of picture drawn for a given state can also be thought of as a render of a part of a hypermedia document, depicting the content and the set of actions appropriate when in that area of the document. A picture that draws itself can also consider the situation in which it is to be rendered, for example the picture may be asked to depict a state in one case on a cell phone and another case a wall display. In these situations the picture identifies the context and modifies the render to suit.

Pictures that draw themselves as interaction in the GUI proceeds must inspect program state and themselves in order to determine appropriate adjustments for the next render. This process of self-inspection and inspection of the running computer program and the

system context is a "reflective" process. Lorenz (2003) describes reflective computing as when "computing has an accurate representation of itself and the computation is consistent with its representation". The graphical display as a picture that draws itself , requires both a representation of itself and that the computing that occurs is consistent with the representation. Inspection of itself implies that the picture is represented in a way that it can raise information to a level from which it can be inspected. Inclusion of facilities or operators in the self drawing picture which, either prior to render or during render allow information to be raised, that is facilitate "reification" supports the description of self drawing pictures as "reflective". "Reification is the process in which information is 'raised' to a level at which reflection can occur"(Danvy 1988).

The following section examines some graphics and interaction programming languages and development tools, identifying operators or structure that assist with "reification" and "reflection". The third section present a brief overview of the composable self-drawing pictures under discussion. Introducing the term "drawble" as an object that is composable and self-drawing. The fourth section draws some conclusions.

2. REFLECTION AND REIFICATION

Development and description of graphical display for interactive or static systems have been undertaken by many authors and researchers. Here reflection is described for a sample of these. Categories of reflection are not described; the sample is presented in chronological order. The systems in which "interaction" is a concern of the system are highlighted.

Sutherland (1963) describes SketchPAD in which graphical constraint macros can be specified. These macros inspect points on a diagram to determine subsequent adjustments in the their positions. The macros in SketchPAD set up a level of reflection in the diagram that allows the system to adjust itself.

Kulsrud (1968) describes categories of commands or statements required in a "General purpose graphic language". These include "description", "manipulation", "analysis" , "validity checking" and "other". The "other" includes facilities for construction sub-pictures. In this system a graphics program can inspect pictures for topological and regional attributes. Pictures and parts of pictures can be loaded, copied, erased and rotated. Picture manipulation can occur as a consequence of analysis. Analysis changes a value in a variable that can then be use to determine manipulation of the picture. The analysis statements inspect pictures with potential subsequent manipulation of the final render. Analysis statements combined with manipulation statement provide a level of reflection by the program which is useful for layout by determining appropriate distance relationships and for rendering by determining for example if gaps between lines should be snapped to a point.

O'Brien (1975) introduces "Image" a programming system that includes structured programming constructs as well as introducing purpose built graphics commands such as the currently familiar "lineto". "Image" also introduces specification of graphical objects that have "display generating" sections and "action" sections. The "display generating" sections produce graphics that depend on variables representing graphical parameters such as the X or Y of a location. The "display" generated is based on inspection of parameters used in the display. The "actions" section provides for handling of interupts caused by an "identifier strike in the preceding object". The graphics generating section sets up parts of the generated display that can be handled by the action. "Image" reflects on its identifiers for both the generation of graphic display and the handling of "interaction".

Bergeron *et al* (1978) describe the "Core Graphics System" which focuses on adding purpose built commands, for example "MOVE_ABS_TO(X,Y)" and providing a means of rendering into "Windows", "viewports" and "viewvolumes" to a standard programming system. The Core system includes a graphics "segment", into which a sequence of "primitives" (commands) is composed. Parameters can be set for a segment causing adjustments in the display. Interaction in the Core system is acheived using a queue of events generated from "virtual devices", for example "pick" or "locator". Message queues with event handlers are intrinsic to the Windows and other current operating systems. Core does not provide any features that assist with reflection; instead it relies on the standard data types and structures to allow reflection of the state in the display.

"PICTUREBALM" by Goates *et al* (1981) recognises analogous requirements for LISP and graphics languages. PICTUREBALM provides composition of graphics into Models and "clusters". These are based on points from which lines, polygons and higher graphical objects are composed. The features of most interest are described in the following:

```
765 anOrigin =  new objTargetPt(0,0,aShape);
766 curveEnd =  new objTargetPt(100,0,aShape);
767 extraDown = new objTargetPt(200,200,aShape);
768 pointX = new objPointX(curveEnd);
769 aGT       =    new objGT([pointX,300]);
770 aShape.addDrawble( new objBeginFill(aShape,0x00FF00,100,
771                    new objLineStyle(aShape,1,0x000000,100,
772                    new objMoveTo(aShape,anOrigin,
773                    new objTextAt(aShape,anOrigin ,"hello",
774                    new objCurveTo(aShape,new objTargetPt(50,50,aShape),curveEnd,
775                      new objCond(
776                                          aShape,
777                                 aGT ,
778                                          new objLineStyle(aShape,1,0xFF00FF,100,
779                                          new objLineTo(aShape,extraDown,
780                                                                          null)
781                                                                        ),
782                    new objCurveTo(aShape,new objTargetPt(50,100,aShape),anOrigin,
783                   // new objLineTo(aShape,new objTargetPt(0,100,aShape),
784                    new objEndFill(aShape,null)
785
786                       ) )/* CurveTo */ )/* LineTo*/))/* moveTo*/)// lineStyle
787                       )// beginFill
788                   );
789 aShape.draw(_root.Display1);
790
```

Figure 1: A Shape Drawble

1. "Allowing models to contain references to other models facilitates dynamic displays",

2. "Allowing procedure calls to be imbedded within Models provides the user with a mechanism which can easily effect arbitrary displays, transformations, parameterized models or other functions that may be required by a specific application". PICTUREBALM presents a perspective similar to the current paper. Pictures that draw themselves are an outcome of thought into requirements of dynamic graphics. Interactive systems are inherently dynamic.

Van de Bos (1988) presents "Abstract Interation Tools" (AIT) . AIT are components of a User Interface Management System that are intended to be added to an exisiting system. These tools relate user actions to a set of manipulations of the display. They have an action area and in a manner similar to "Image" the actions cause changes in the display. AITs are specified in terms of relations on virtual devices, for example a "pick" device. These can be reflective in the sense that the actions are determined by inspection of an object of interest. They are abstract because the do not need to be specific, an example AIT given is "DRAG" which acts on a "viewport" rather than a specific object.

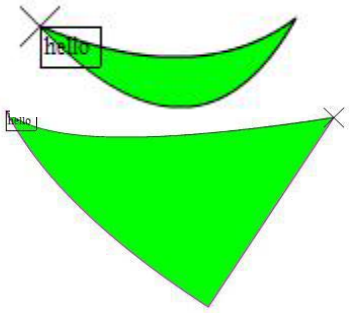
Bottoni (2002) introduces "Virtual Interaction Machines" that provide "multilevel" modelling and design of interactive systems. The "Virtual Interaction Machine" is derived based on a "3-D Interaction Modelling Space", with "activity language", "pictorial language" and "programming language" dimensions. "Virtual Interaction Machines" are derived for different levels of abstraction. This approach is instrinsically reflective, with higher level machines having to inspect lower level machines to determine appropriate changes in the system.

Macromedia (2004) presents in its product "Flash" a programing system that allows event handlers to be attached to "Movie Clips". The Flash editor "compiles" to a file format called Flash "shockwave"(swf) file, which is a "tag" based binary file. Tags in a "shockwave" file are composable, through reference to previously defined tags. Tags can be chunks of executable code which create and manipulate "MovieClips" and their content. The "shockwave" file is interpreted by the "flash" player. This system provides many opportunities for reflection. MovieClips inspect themselves as a result of events , values are changed in variables as a consequence of changes in text fields. These changes cause events that can be handled. The handlers can then be used to adjust the display as a consequence of reflection.

The World Wide Web Consortium (W3C, 2004) defines standards for web based markup and information exchange. The W3C Document Object Model (DOM) provides an object model that facilitates the construction of self modifying documents. The document object model provides a representation of the document which, when modified, affects the computing undertaken by the document as it is rendered. Rendering is called when a change is made to the DOM. Tags of a markup language used to describe the layout and content of the web document are represented in the DOM as instances of object classes called "nodes". Each node can contain a list of "child-nodes", and can access "sibling" and parent nodes. Nodes are also thought of as "elements" of the document. Each "element" can also be accessed using an identifier. The DOM is available to scripts that are themselves part of the document's elements. A "script tag" contains code that can inspect and modify elements of the document. Provision the DOM and its manipulation is an example of reification and reflection in this system. "Reflection" has been provided to meet interactive GUI requirements, for example display on different platforms and responding to user input.

The World Wide Web Consortium also provides a "Scalable Vector Graphics" (SVG) language specification for graphics to be used in the web. SVG includes the facilities for drawing complex shapes to be displayed in a web browser. An SVG with DOM like scripts provides a very powerful mechanism for delivering sophisticated graphics on the web. This solution also uses "reflection" as a means to meet these requirements.

Reflection has been inherent or in the background of a number of graphical programming systems. It has not been made explicit. In the following a system is described that uses composable graphic primitive objects making pictures that change themselves.



Figures 2 and 3: A Picture that changes itself depending on the value of its right most handle

3. DRAWBLES

Pictures that draw themselves are expressed as a composition of instances of "draw-able" objects, called "Drawbles". Drawbles have been defined in Macromedia's "Flash Action Script" and in Borland's Delphi. See Figure 1 for an example drawble shape written in Flash Action Script . Drawbles were first defined as wrappers around available available drawing commands, for example "lineto".

Sequences of commands were stored so that adjustments to the shapes or figures they produced could be adjusted in the most flexible manner. The system under construction requires very smooth and fine interactive adjustment of shapes, clean fading in and out, aswell as adjustments to scale and rotation. Storing graphics commands with references to their parameters allows fine adjustment to be made to a shape and allows for editing and updating of part of a shape. Each object manages itself and can have an object that follows it.

To facilitate rotation and scaling, transformational objects were added to the sequence. All points that are parameters of subsequent drawbles in the sequence are scaled or rotated by a transformation drawble. A "conditional" drawble was introduced to allow for parts of the sequence to be hidden and revealed under a condition. This might also affect scaling and and rotation of the rest of the display.

The addition of a conditional drawble moved the drawble from a static display to a dynamic display. At this point drawbles become reflective. The conditional drawble also introduces a "scope" problem. The drawbles that are introduced by the condition are thought of being in a block. Transformations introduced in a block affect the rest of the drawbles while the transformation drawble is present.

Additional drawbles are to be added that manipulate drawbles in the sequence:

- a CreateInsert drawble - creates and inserts a drawble after itself,
- a DeleteNext drawble - deletes the next drawble in the sequence,
- a DeleteAll drawble - deletes all the drawbles that follow
- a CreateReplace drawble - creates a drawble and replaces the next one.

Adding a drawble that assigns a value to a variable provides the facility for the drawble sequence (picture) to manipulate values on which it is based. A drawble that provides iteration could also be added to generate a sequence of drawing or patterns.

5. CONCLUSION

By storing a picture as a sequence of composable drawing command object instances, that provide self-inspection and adjustment, "reflection", the opportunity for expressing dynamic displays is increased. GUIs are inherently dynamic. Treating the graphic display as " a picture that draws itself" provides an approach that meets GUI development requirements.

REFERENCES

Bergeron, R. D., Bono, P. R.,Foley , J. D. 1978 "Graphics Programming Using the CORE System" in ACM Computing Surveys 1978, p 389.

Bottoni, P., Costabile, M. F., Fogli,D., Mussio, P., 2001 , "Multilevel Visual Interactive Systems", IEEE 2001 Symposium on Human Centric Computing Languages and Environments(HCC'01), September 05-07.Stresa Italy, 256.

Danvy, O., MALMKJAER,K., 1988 "Intensions and Extensions in an Reflective Tower",Proceedings of the 1988 ACM conference on LISP and functional programming ,Snowbird, Utah, United States , ACM, pp 327-341.

Goates, G B., Gris, M. L., Herron , G J. 1980, "PICTUREBALM: A LISP-BASED GRAPHICS LANGUAGE WITH FLEXIBLE SYNTAX AND HIERARCHICAL DATA STRUCTURE", ACM SIGGRAPH 1980, p 93.

Kulsrud, H. E., 1968, "A General Purpose Graphic Langauge", Comms of the ACM 11(4):247-254

Lorenz, D. H., Vlissides, J., 2003, "Pluggable Reflection: Decoupling Meta-Interface and Implementation", Proceedings of the 25th international conference on Software engineering, Portland, Oregon, IEEE 2003:3-4.

Macromedia , 2004, "Macromedia Flash MX 2004 - ActionScript".

O'Brien, C.D. , Brown , H. G 1975 "IMAGE : a language for the Interactive Manipulation of a Graphics Environment", in ACM SigGraph 1975, p53.

Object Management Group, 2003 "OMG Unified Modelling Language Specification - March 2003 Version 1.5 formal/ 03-03-01", Object Management Group, http:// www.omg.org/docs/formal/03-03-04.pdf (18 June 2004.)

Sutherland , I. E. , 1963, "SKETCHPAD - A MAN-MACHINE GRAPHICAL COMMUNICATON SYSTEM", PhD Thesis, MIT, Cambridge, USA.

Van Den Bos, J. , 1988 "Abstract Interaction Tools: A Language for User Interface Management Systems", ACM Trans on Programming Languages and Systems , 10(2):215

W3C - World Wide Web Consortium , 2004, www.w3c.org accessed June 2004.