Transactions on Engineering Sciences vol 15, © 1997 WIT Press, www.witpress.com, ISSN 1743-3533

Writing A Graphical User Interface for Mathematica using Mathematica and MathLink

Chikara MIYAJI and Hiroshi KIMURA *Institute of Health and Sport Sciences, University of Tsukuba, Tsukuba City, Ibaraki, 305 JAPAN*
Department of Engineering, Kyushu Institute of Technology, Tobata, Kitakyushu, 804 JAPAN

Abstract

Interactive graphics are a weak part of Mathematica. In this article, a MathLink program which provides interactive graphics in realtime will be introduced. The program can be used to detect a user interface event, and send it to the Mathematica Kernel as an Event Expression. This mechanism enables one to write Graphical User Interface(GUI) in Mathematica. To define a GUI object in Mathematica, an Object Oriented Programming Style (OOPS) is introduced. The combination of realtime interaction and kernel evaluation provides a dynamical interface building, execution, and debugging environment. In this article, theoverall design of the program will be discussed.

1 Introduction

For example, several example of the Game of Life have been implemented as Mathematica's Notebook animations. The animation itself, however, is a collection of graphical output that is essentially static. The Game of Life isn't animated in realtime. Additionally, the user can't edit the Life's Cell graphically using a user interaction such as a mouse click. These limitations come from the design of the Mathematica Notebook. While the notebook format is a very good tool to describe Mathematical expression and thought, it lacks realtime graphics capability and a mechanism to react to user events. Instead of using the Notebook interface, the MathLink program which is introduced in this article can handle realtime graphics and

Transactions on Engineering Sciences vol 15, © 1997 WIT Press, www.witpress.com, ISSN 1743-3533

realtime interaction with a user interface event. Figure 1 shows a sample interface window generated by this program.

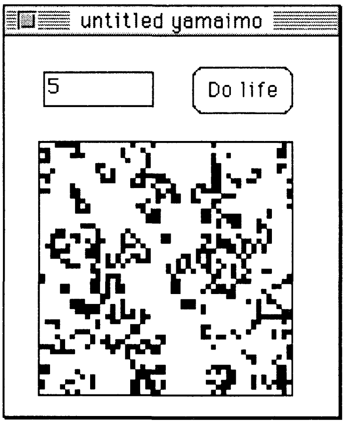


Figure 1: A sample window created by the MathLink Program.

Clicking on a Cell using the mouse inverts the cell shading. Setting the number in the text area sets the frequency of the system evolution. Selecting the Button in the figure will start the evolution of the Game of Life simulation. Users can then see the cell evolution in realtime. The user can modify the object's location and size by mouse dragging. It doesn't use more than 5 minutes to create this window.

2 The Program

The main advantage of this program, realtime graphics and event handling mechanism comes from the ability of MathLink. Object oriented programming style and Interactive programming and debugging environment of this program comes from the Mathematica's flexibility.The key ideas of this program can be summarized as follows:
1. A simple object oriented programming style for Mathematica
2. An event expression format for user events
3. An event passing mechanism via a MathLink connection
4. An evaluation mechanism for multiple MathLink programs

3 Object Oriented Programming Style

All of the element of the program, the window, visible objects, button, text area, menu and program itself are defined as Mathematica's object using this OOPS. To define the object as OOPS in Mathematica, a function closure is used. This is an definition of object class `foo`.

```
New[foo] :=  Module[{var1,self,super={New[p1],New[p2]}},
  self[dispose] := <<dispose method>>;
  self[setvar1,var_] := var1 = var; (* setter *)
  self[getvar1] := var1; (* getter *)
  self[sel_,args___] :=
    findmethod[foo[self,sel,args],
               findsuper[super,sel,arg]];  self]
foo[self_,m1] := << method ''m1'' definition >>
foo[self_,m2,a1_] := <<method ''m2'' definition >>
```

New[foo] will return a new function instance with a temporary name, such as self$123. Sending a message to this instance allows operations to be performed on it, for example, self$123[dispose]. If the method doesn't match local definitions, findmethod[...] will be used. If the method doesn't match to foo[self,sel,args] pattern, findsuper[super,sel,arg] will be searched as the next candidate.

```
findmethod[f_[m___],next_] := sub[f[m],f[m],next]
SetAttributes[findmethod,HoldAll]
sub[f1_,f2_,next_] :=
If[Hold[f1] === Hold[f2] || f1 === $noMethodfound, next,f1]
SetAttributes[sub,HoldRest]
findsuper[f_List,m___] :=
  findmethod[First[f][m],findsuper[Rest[f],m]]
findsuper[{},m___] := $noMethodfound
```

findsuper will invoke findmethod to search the pattern in the list of super, {New[p1],New[p2]}. This recursive search of the method will go through the object's hierarchy. This allows the implementation of Multiple Inheritance of the object. Figure 2 shows the object's class hierarchy of the program.

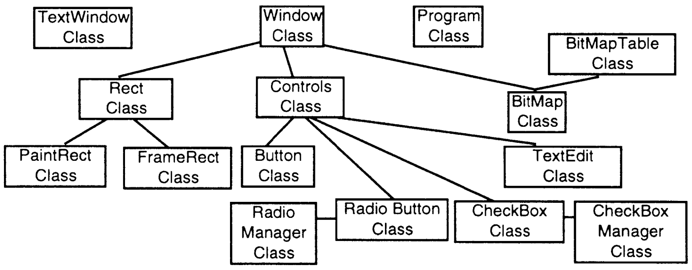


Figure 2: Class hierarchy of the program

4 Realtime Graphics

Below is a sample of the graphics functions of the program. It defines a MathLink template to draw a line in its window.

```
:Begin:
:Function:   lineto
:Pattern:   LineTo[{x_,y_}]
:Arguments:   {x, y}
:ArgumentTypes: {Integer, Integer}
:ReturnType:   Integer
:End:
```

When the Mathematica's Function `LineTo[]` is invoked, the C function `lineto()` which will be executed.

```
int lineto(int x, int y)
{
  GrafPtr cp;
  GetPort(&cp): // save current graphics port.
  SetPort(gWindowPtr); // set graphics output port to the windc
  LineTo(x, y);  // Toolbox call.
  SetPort(cp); //restore current port.
  return(0);
}
```

After establishing a connection between the program and kernel, the Mathematica function `LineTo[10,20]` will draw a line on the program's window immediately. In this way, MathLink template functions can be defined, for example to open an window, close the window, draw a rectangle etc. These enable the manipulation of graphics in a program from within Mathematica. These MathLink Function are encapsulated into each object's definition. For example, a Window object can use the MathLink Functions, `OpenWindow[]` and `CloseWindow[]` to create and dispose of windows. These function can be invoked as the message to an object.

```
New[Window, title_String:"untitled yamaimo",opts___Rule]:=
  Module[{wport,self},
    self[port] := wport;
    self[dispose] :=
     (CloseWindow[wport];Remove[wport,self]);
    self[selector_,args___] :=
       findmethod[Window[self,selector,args],
         $noMethodfound];
    wport=OpenWindow[self,title,opts,WindowColor->False];
  self]
```

This is shown schematically in Figure 3.

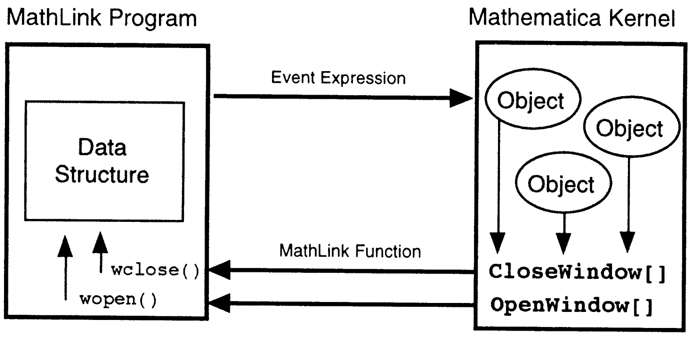


Figure 3: The relation between Object and MathLink Function.

5 Handling user events

If the MathLink program captures a user event, for example a mouse down event, it will be send to kernel. When the kernel receives that event, it is easy to interact with that event. For example the kernel can draw a line on that window in realtime. This brings realtime interaction to the program. It is not hard to handle user events via a MathLink program. When compiling a MathLink template, it is expanded into a C program by the "`mprep`" program. This C program has its own event loop inside it. To detect a user event is only necessary to add a few lines of code in the proper place. For example, a mouse click event can be send to the kernel by using the `sendclick()` function which was inserted in the `handle_user_event`'s branch.

All user events which are sent to kernel are defined in this way. For example, mouse clicks, dragging, menu selection and window closing events etc. are send to kernel by the function defined in the MathLink program. Some users event are not sent to the kernel, but are processed internally. For example, the window move event is processed inside the MathLink program. This makes the writing of the Mathematica based program simpler.

6 The event format

Usually, the events have several arguments, for example the click location, key modifier information etc. The event format should be flexible to match the wide variety of event types. For this purpose, this program uses a Mathematica expression to contain the event format information. When the user generates an event, the Mathematica expression will be send to the kernel. This gives great flexibility to the program. The Mathematica expression can express virtually anything. In this article, these expression are referred to as "Event Expressions". A User event will generate an Event Expression, and the format of the Event Expression is as a message to the object itself which the user selected. For example, if the user clicks an object which name is `self$123` in a window, the Event Expression would be the expression `Hold[self$123[click, 12, 34]]`. MathLink Library functions can be used to construct the Event Expression. For example, the Event Expression `Hold[self$123[click, 12, 34]]` can be constructed by this 7 line C fragment.

```
MLPutFunction(eventlink,"Hold",1);
MLPutFunction(eventlink,getSymbolName(),3);
MLPutSymbol(eventlink,"click");
MLPutInteger(eventlink, 12);
MLPutInteger(eventlink,34);
MLEndPacket(eventlink);
MLFlush(eventlink);
```

7 Event sending Mechanism

So as to not drop a asynchronously generated user event, a Queue mechanism is generally used. But, instead of creating a Queue for the event, a MathLink connection is used by the program. This link itself has queue characteristics. When the MathLink program is launched, a new MathLink connection, named eventlink is created. All Event Expression will be send on this link. And the Standard MathLink connection will be used for the MathLink Function. To get an Event Expression from the MathLink program, an infinite while loop is be used. After this while loop starts, all event generated from the MathLink program are handled.

```
While[True,
      EvalExpression[GetEventExpression[eventlink]]
     ]
GetEventExpression[link_LinkObject] :=
  If[LinkReadyQ[link],LinkRead[link]]


EvalExpression[exp_] :=
   If[expr =!= Null,
      Switch[flag,
             print, Print[expr]],
             trace,TracePrint[ReleaseHold[expr]],
             _, ReleaseHold[expr]
            ]
     ]
```

This is shown schematically in Figure 4.

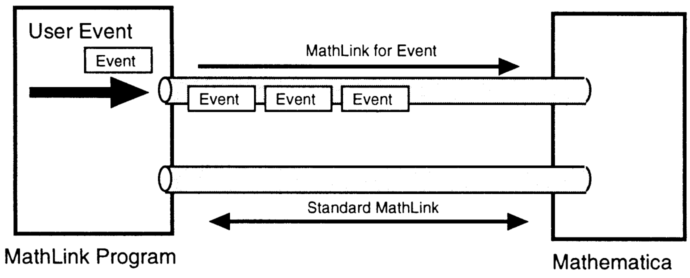


Figure 4: MathLink connection to send event.

8 Multiple MathLink Front end

When the kernel is connected to multiple MathLink programs with same event sending mechanism, it is possible to process all events from each of the MathLink programs. `EvalExpression` will be applied to each eventlink of `eventlinklist` by `Map[]`.

```
While[True, Map[EvalExpression[GetEventExpression[#]]&,
        eventlinklist]
```

This makes several MathLink programs work together as a Front end. For example, the GUI program will run with the TextEditor MathLink program and the Mathematical Formula Editor.

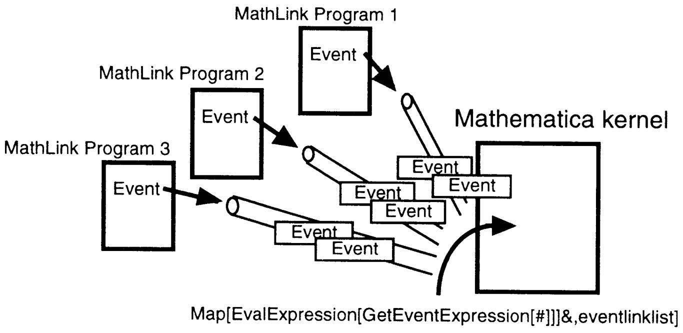
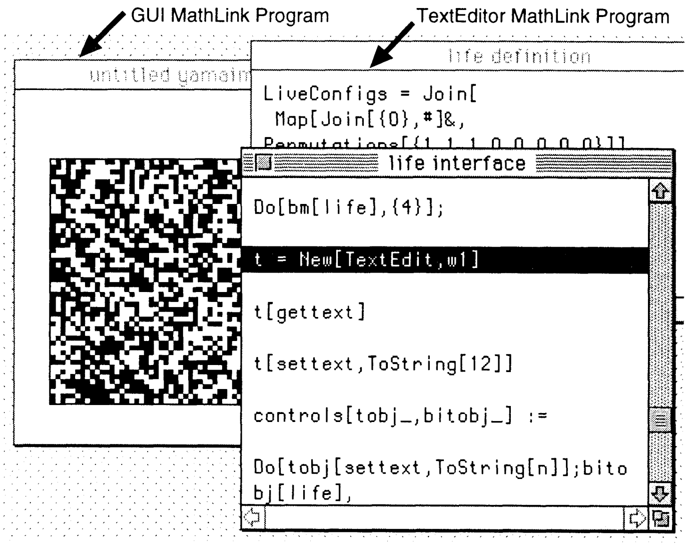


Figure 5: Multiple MathLink Programs

To examine this idea, a simple MathLink TextEditor program was constructed. Figure 6 shows how the GUI program and the TextEditor work together very well. The programming complexity is relatively small. Each program can focus on its own special features. The programs only need to share the same event sending mechanism to work together.



```
LiveConfigs = Join[
 Map[Join[{0},#]&,
Permutations[{1,1,1,0,0,0,0,0}]]
```

```
Do[bm[life],{4}];

t = New[TextEdit,w1]

t[gettext]

t[settext,ToString[12]]

controls[tobj_,bitobj_] :=

Do[tobj[settext,ToString[n]];bito
bj[life],
```

Figure 6: GUI MathLink program and TextEditor MathLink Program work together

The use of multiple MathLink program will reduce the complexity of the Front end program construction and at the same time allow for the construction of a wide variety of user Front end combinations to meet a variety of needs. The idea of using Multiple MathLink Front ends will match the concept of Compound Documents very well.

References

[1] Gaylord, R., Wellin, P.: Computer Simulations with Mathematica,Exploration in Complex physical and Biological Systems, Springer/TELOS, 1995.

[2] Gaylord,R. and Nishidate,K: Modeling Nature with Cellular Automata using Mathematica,Springer/TELOS, 1996.

[3] MIYAJI, C.: Network Programming by Mathematica, Iwanami, 1997 (Printing)