Algebraic Specification of User Interfaces

M. Cabrera, M. Gea, F. Gutierrez, J.C. Torres

Dpt. Lenguajes y Sistemas Informaticos
Universidad de Granada (Spain)

Address:
Dpt. de Lenguajes y Sistemas Informaticos
Escuela Superior de Informatica
18071 Granada
Spain

Telephone: 34-58 242 809
Fax: 34-58 243 179

email: (mcabrera, mgea, fgutierr, jctorres)@goliat.ugr.es

Algebraic Specification of User Interfaces

M. Cabrera, M. Gea, F. Gutierrez, J.C. Torres

Dpt. Lenguajes y Sistemas Informaticos
Universidad de Granada
Escuela Superior de Informatica
18071 Granada
SPAIN

Telephone: 34 58 242 809 Fax: 34 58 243 179
email: (mcabrera, mgea, fgutierr, jctorres)@goliat.ugr.es

Abstract. Formal methods have been successfully used to specify graphic [1,2] and interactive system [3,4]. This paper discuss the use of algebraic specification to User Interface. We use a language, GRALPLA [5], for the specification and a translator for developing a running prototype. We propose two level of description for the specification and validation of a User Interface system: formally using algebraic specification to describe properties and interactively using the prototype for checking its flexibility. The paper contains an example of a Simple User Interface, to explain the design process.

1. INTRODUCTION

One of the most important steps in the development of computer system is a correct specification of the system to be designed. Within this system we could study the User Interface requirements. Using formal specification we have the benefits of avoid ambiguity and facilities for correctness programs [6].

As formal method, we use algebraic specification, which is a property-oriented specification method, as opposed to a model-oriented method, since it doesn't contain a model of the system. An important advantage of algebraic specification is that the specifier does not fix a model of the system, which may be a constraint on the design and implementation process.

Most of the works in HCI studies has been based on using a model for constructing the specification of the interface. Most of this models covers partly of the system (usually focuses on presentation or dialogue layer[7]) and this usually generates a partial specification of the system.

We use an approach which focuse on the operations, instead of on fixing the model, for the specification of user interfaces. One benefit of this approach is that we could specify each level of the UI at different levels of abstraction, without loss of generality. Also is possible to specificate at different levels of abstraction.

2. ALGEBRAIC SPECIFICATION

In algebraic specification [8], we describe the system using a set of specification modules, each one specifies one component of the system. Formally these parts are abstract data types.

The specification of every module describes a set with a precise algebraic structure, this structure is expressed in terms of an interface, a collection of functions, and a set of conditions that hold for the module. The conditions are expressed as a set of axioms over the module functions, that is, the main components of the algebraic specification of a module are: its carrier set, the collection of defined functions (its signature), and the collection of axioms.

We could study the system using a two steps approach. Firstly, we could focus on the properties of the system (completness, consistency between modules, etc.) using formal methods. Secondly, we could study perceptual properties (easy of use, learning, style, etc), using an implementation of the specification. With this purpose, we use a translator from algebraic specification to a high level programming language, which derives C++ classes for every specification module.

Once we have validated the specification, we could use the prototype as a first version for the design and the implementation process.

3. DESCRIPTION OF THE *GRALPLA* LANGUAGE.

GRALPLA is an algebraic specification language developed at Granada University. From an GRALPLA specification it is possible to build a prototype automatically in a high level language.

The specification of a system using GRALPLA consists of a collection of modules which are written and translated independently. The specification of each module has five parts: header, dependencies, constructors, functions and axioms. A correct module is formed with a sequence of these components in the order described (see fig. 1). The remainder of this paragraph describes the syntax and semantics of these sections.

· **Header.** This section contains the name of the module (object_id), which is also the name of the carrier set, and an optional list of formal parameters, used to build generic modules. The parameter list is a sequence of identifiers, which are used as types for generic modules. The actual type for each identifer is fixed when using the module. The signature of these parameters is obtained from their specifications.

· **Dependencies.** This section gives a list of the modules which are used by the actual module. Object_id is a module identifier which is used in the specification of the actual module. That is, the actual module may use its carrier set and all its functions. The list of types must appear when the used module has parameters, they act as actual parameters for this instance of the generic module.

```
<Specification_Module> ::= Header [Dependencies] Constructor
                        Functions  Axioms [Synchronization]

<Header>::= [parametric] [graphic] object object_id
                        [ [parameter_id {,parameter_id}] ] ;

<Dependencies>::= import { object_id [[ type_id {,type_id}] ];}

<Constructors> ::= [Constructor]
                        {function_id : [type_id { , type_id}] -> object_id
                         [where function_id[var_id {, var_id}] = expression ]; }

<Functions> ::= Functions { function_id
                        [< selector_id, {selector_id} > ] :
                        type_id { , type_id} -> type_id { , type_id}; }

<Axioms> ::= Axioms [ var { var_id {, var_id} : type_id ;} ]
                        {function_id [Term {,Term}] [.selector_id] =
                        Expression; }

<Expression>::= Term ¦ if [Condition] Expression else
                        Expression Endif ;

<Condition>::= function_id [[ Term {, Term }] [.selector_id]]
                        (=¦>¦<¦<>) Term ¦ Boolean-Expression

<Term>::= function_id [ [Term {,Term}] [.selector_id] ] ¦
            var_id ¦ Constant ¦ Boolean-Expression ¦
                        Integer-Expression ¦ ERROR [ String]

<Synchronization> ::= Synchronization
                                {do function_id [(var_id {, var_id })] when condition; }
```

· **Constructors.** This section contains the constructor signature. Constructors are a special kind of function, every constructor returns only one result of the carrier set type. They are used to create objects of this type, so any module must have at least one constructor. Function_id is the constructor name. The type list is employed optionally to allow the use of parameters when creating an object (i.e. the dimension of an array).

· **Functions.** This section contains the signature of the module functions. Any function may return more than one result, in this case, selectors must been included to identify every result. Any function definition has four components: the function identifier, an optional list of selector identifiers, a list of argument types, and a list of result types. Any function must have at least one argument and one result.

· **Axioms.** This section contains the axioms that hold for this module. Every axiom is an equation, whose lefthand-side is a function. To write the equation it is possible to use auxiliary variables, whose only purpose is to establish relationships within the axiom. The variable declaration uses a pascal-like structure. Every axiom refers to one module function, whose arguments may be functions. Expressions may be conditional ones. Integer and boolean expressions are built up with integer or boolean terms and operators.

Axioms are interpreted as directional equations: what appears on the left hand side could be carried out as explained on the right hand side.

The error conditions are described, indicating that the corresponding function returns as result ERROR.

· **Synchronization.** The synchronization section may imposes execution restriction to the function defined in the module. A function appearing in the *do* part of a synchronization statement could be carried out only when the associated expression is evaluated as true.

4. EXAMPLE

This section shows the application of the approach to the specification of a simple User Interface.

We can specificate user interface using bottom-up methodology[9,10]. We start specifying the basic elements behaviour. At bottom level we could link application actions to user interface elements and to their graphical appearance. The inner levels define the behaviour of the user interface itself. At the top level we define the visible functionality for the user.

Each element encapsulates part of the global user interface behaviour. This allows us to do specification refinement in an easy way. Interaction is distributed through elements. For each elements we define its interaction and its appearance.

For this description, we specificate four elements: windows, buttons, menus and cascades.

Conectivity between elements may be shown in the following schema:

Now we are going to describe every element independently:

Buttons

A *button* is a labeled rectangle with an associated action belonging to the application layer. The system allows to manage its appareance (*ShowButton, HideButton*), geometry (*getLimits, move*), and interaction (*event, is_in*).

Menus

Conceptually, a *menu* is a list of elements. Those elements are: *button* and *cascade*.

Menu delegates its functions over the element list.

Cascades

Cascade is an object, defined by a *button* with an associated *menu*.

The menu is visible to the user if we activate it picking the button. The menu is desactivated if we pick outside the menu items.

Windows

Windows appearance may be defined with two points, the upper left corner and the lower right corner, that define its extension. It is possible to control the visibility of the window using two functions for manage it (*open_Window, close_Window*).

Over a window there are several functions defined to manage its geometry: *move, size, getLimits*.

For the construction of the interface there are functions that let us add valid elements into the window. The elements could be other window, a button and a menu.

The window has two functions (*event, is_in*), to control the user interaction. This functions let know the interface and trigger associated actions.

Once we have build the prototype of the system, we could use it build up concrete interface. For instance, to build up the interface showed in the fig. 2 it is necesary to carry out the following operations on the prototype:

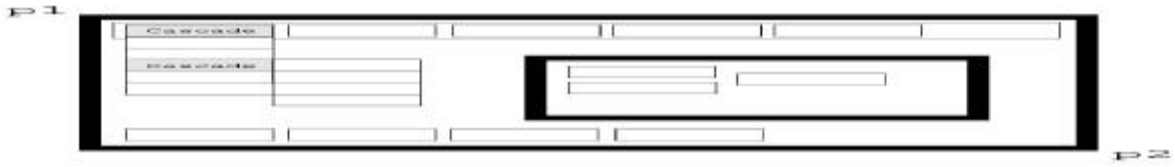
```
MainWindow = new Window(p0,p1);
Window1 = new Window(p2,p3);
MainWindow.additem(Window1);

Button1 = new Button(p4,p5);
Button2 = new Button(p6,p7);
Button3 = new Button(p8,p9);
Window1.additem(Button1);
Window1.additem(Button2);
Window1.additem(Button3);

Button4 = new Button(p10,p11);
Button7 = new Button(p16,p17);
Cascade2= new Cascade(Button7, Menu2);

Button5 = new Button(p12,p13);
Button6 = new Button(p14,p15);
Button8 = new Button(p18,p19);
Button9 = new Button(p20,p21);
Cascade1= new Cascade(Button4, Menu1);
Cascade1.additem(Button5);
```

```
Cascade1.additem(Button6);
Cascade1.additem(Button8);
Cascade1.additem(Button9);
Cascade1.additem(Cascade2);
Button10 = new Button(p22,p23);
Button11 = new Button(p24,p25);
Button12 = new Button(p26,p27);
Button13 = new Button(p28,p29);
MenuBar.additem(Cascade1);
```



User interaction generates a point (usually a mouse movement). This is an event managed for the interface, sending the point to the MainWindow (MainWindow.event(point))

5. CONCLUSIONS AND FUTURE WORKS

In this paper we have shown the specification of a simple User Interface. The benefits are the flexibility to define the properties of the system and its refinement.

Another benefits is the posibility to study the completness and consistency in the framework of algebraic specification to the user interface. As this could be an easier way to ensure correctness of the user interfaces.

In previous works we have studied specifications of concrete model of interactions, based on interactor.

REFERENCES

1 **Carson, G.S.**, *The specification of Computer Graphics Systems*, IEEE Computer Graphics & Applications. **3**(6), pp.27. Sept. 1983.

2 **D.A. Duce**, "Formal Specification of Graphics Software", pp.543/574 in *Theoretical Foundation of Computer Graphics and CAD*, R.A. Earnshaw (ed.), NATO ASI Series F40, Springer-Verlag, Berlin (1988).

3 **J.C. Torres; B. Clares**, *Using an abstract model for the formal specification of interactive graphic system.* Proceeding of the Eurographic Workshop on design specification and verification of interactive system. Italy 1994

4 **F.L. Gutierrez; M. Gea;** *Especificacion algebraica de sistemas interactivos basadas en interadores.* CEIG 95. Palma de Mallorca. Spain 1995

5 **M.Gea; J.C. Torres**, *Object Oriented Prototyping of Graphic Application from algebraic Specification*, Fourth Worshop on Object Oriented Graphics, Sintra (1994)

6 **W.R. Mallgren**, *Formal Specification of Interactive Graphics programming languaje*. MIT press Cambridge 1983

7 **Dream, M.;** *The University of Alberta User Interface Management System*. Proceeding on SIGGRAPH 85. ACM, New York, pp 205-213.

8 **Bergstra J.A.; Heering J.; Klint P**., *Algebraic Specification*, ACM Press Books. Addison-Wesley, 1989.

9 **H.R. Hartson; D. Hix**; *Human Computer Interface Development*. ACM Computer Surveys. Vol 21 no. 1. March 89.

10 **M. R. Frank; J. D. Foley**; *Model Based User Interface Design by Example and by Interview.* Proceedings of the ACM Symposium on User Interface Software and Technology. Atlanta, Georgia, November 1993.

APENDIX: USER INTERFACE SPECIFICATION

```
{ ************************************************
  ****    User Interface Specification: WINDOW
  ************************************************ }

graphic object Window;

        import Button;

        Window : point, point -> Window;

        private OpenWin : point, point -> Window
                where OpenWin(p1,p2) = rectangle(p1,p2);

Functions

        open_Window : Window -> Window;
        close_Window : Window -> Window;

        move : Window, point -> Window;
        size : Window, point -> Window;
        getLimits<top,down> : Window -> point, point;

        additem : Window, Window -> Window;
        additem : Window, Button  -> Window;
        additem : Window, Menu -> Window;

        addCorrectItem : Window, Window -> Window;
        addCorrectItem : Window, Button -> Window;
        addCorrectItem : Window, Menu -> Window;

        is_in : Window, point -> bool;
        event : Window, point -> bool;

Axioms
        var     B, B1 : Button
                M, M1 : Menu
                V1,V2,V3 : Window
                pt, p1, p2, p3, p4 : point;

open_Window (Window (p1, p2)) = OpenWin (p1, p2);
open_Window (OpenWin (p1, p2)) = OpenWin (p1, p2);
open_Window (addCorrectItem (V1, V2)) = addCorrectItem (open_Window (V1), V2);
open_Window (addCorrectItem (V1, M)) = addCorrectItem (open_Window (V1), M);
open_Window (addCorrectItem (V1, B)) = addCorrectItem (open_Window (V1), B);

close_Window (Window (p1, p2)) = Window (p1, p2);
close_Window (OpenWin (p1, p2)) = Window (p1, p2);
close_Window (addCorrectItem (V1, V2)) = addCorrectItem (close_Window (V1), V2);
close_Window (addCorrectItem (V1, M)) = addCorrectItem (close_Window (V1), M);
close_Window (addCorrectItem (V1, B)) = addCorrectItem (close_Window (V1), B);

is_in (Window (p1, p2), pt) = if ((pt >= p1) and (pt <= p2)) true else false endif;
is_in (OpenWin (p1, p2), pt) = is_in (Window (p1, p2), pt);
is_in (addCorrectItem (V1, V2), pt) = is_in (V1, pt);
is_in (addCorrectItem (V1, M), pt) = is_in (V1, pt);
is_in (addCorrectItem (V1, B), pt) = is_in (V1, pt);

event (Window (p1, p2), pt) = false;
event (OpenWin (p1, p2), pt) = if (is_in (OpenWin (p1, p2), pt) activeWin (OpenWin (p1, p2)) endif;
event (addCorrectItem (V1, V2), pt) = if ((is_in (V1, pt)) and(is_in (V2, pt))) event (V2, pt) else event (V1, pt) endif;
event (addCorrectItem (V1, M), pt) = if ((is_in (V1, pt)) and (is_in (M, pt))) event (M, pt) else event (V1, pt) endif;
event (addCorrectItem (V1, B), pt) = if ((is_in (V1, pt)) and (is_in (B, pt))) event (B, pt) else event (V1, pt) endif;

move (Window (p1, p2), pt) = Window (p1+pt, p2+pt);
move (OpenWin (p1, p2), pt) = OpenWin (p1+pt, p2+pt);
move (addCorrectItem (V1, V2), pt) = addCorrectItem (move (V1, pt), move (V2, pt));
move (addCorrectItem (V1, M), pt) =addCorrectItem (move (V1, pt), move (M, pt));
move (addCorrectItem (V1, B), pt) =addCorrectItem (move (V1, pt), move (B, pt));
```

```
size (Window (p1, p2), pt) = if ((get_x (p2+pt)>get_x (p1))and(get_y (p2+pt)>get_y (p1))) Window (p1, p2+pt)
                                                    else ERROR ("Cannot resize") endif;
size (OpenWin (p1, p2), pt) = if ((get_x (p2+pt)>get_x (p1))and(get_y (p2+pt)>get_y (p1))) OpenWin (p1, p2+pt)
                                                    else ERROR ("Cannot resize") endif;
size (addCorrectItem (V1, V2), pt) = addCorrectItem (size (V1, pt), V2);
size (addCorrectItem (V1, M), pt) = addCorrectItem (size (V1, pt), M);
size (addCorrectItem (V1, B), pt) = addCorrectItem (size (V1, pt), B);

getLimits (Window (p1, p2)).top = p1;
getLimits (Window (p1, p2)).down = p2;
getLimits (OpenWin (p1, p2)).top = p1;
getLimits (OpenWin (p1, p2)).down = p2;
getLimits (addCorrectItem (V1, V2)).top = getLimits (V1).top;
getLimits (addCorrectItem (V1, V2)).down = getLimits (V1).down;
getLimits (addCorrectItem (V1, M)).top = getLimits (V1).top;
getLimits (addCorrectItem (V1, M)).down = getLimits (V1).down;
getLimits (addCorrectItem (V1, B)).top = getLimits (V1).top;
getLimits (addCorrectItem (V1, B)).down = getLimits (V1).down;

additem (Window (p1, p2), V1) = if ((getLimits (V1).top >= p1) and (getLimits (V1).down <= p2))
                                                    addCorrectItem (Window (p1, p2), V1)
                                                else ERROR ("Window too large") endif;
additem (Window (p1, p2), M) = if ((getLimits (M).top >= p1) and (getLimits (M).down <= p2))
                                                    addCorrectItem (Window (p1, p2), M)
                                                else ERROR ("Menu too large") endif;
additem (Window (p1, p2), B) = if ((getLimits (B).top >= p1) and (getLimits (B).down <= p2))
                                                    addCorrectItem (Window (p1, p2), B)
                                                else ERROR ("Button too large") endif;

additem (OpenWin (p1, p2), V2) = if ((getLimits (V2).top >= p1) and (getLimits (V2).down <= p2))
                                                    addCorrectItem (OpenWin (p1, p2), V2)
                                                else ERROR ("Window too large") endif;
additem (OpenWin (p1, p2), M) = if ((getLimits (M).top >= p1) and (getLimits (M).down <= p2))
                                                    addCorrectItem (OpenWin (p1, p2), M)
                                                else ERROR ("Menu too large") endif;
additem (OpenWin (p1, p2), B) = if ((getLimits (B).top >= p1) and (getLimits (B).down <= p2))
                                                    addCorrectItem (OpenWin (p1, p2), B)
                                                else ERROR ("Boton too large") endif;

additem (addCorrectItem (V1, V2), V3) = if ((getLimits (V3).top >= getLimits (V1).top)
                                                    and (getLimits (V3).down <= getLimits (V1).down))
                                                        addCorrectItem (V1, V3)
                                                    else ERROR ("Window too large") endif;
additem (addCorrectItem (V1, V2), M) = if ((getLimits (M).top >= getLimits (V1).top)
                                                    and (getLimits (M).down <= getLimits (V1).down))
                                                        addCorrectItem (V1, M)
                                                    else ERROR ("Menu too large") endif;
additem (addCorrectItem (V1, V2), B) = if ((getLimits (B).top >= getLimits (V1).top)
                                                    and (getLimits (B).down <= getLimits (V1).down))
                                                        addCorrectItem (V1, B)
                                                    else ERROR ("Boton too large") endif;

additem (addCorrectItem (V1, B), V2) = if ((getLimits (V2).top >= getLimits (V1).top)
                                                    and (getLimits (V2).down <= getLimits (V1).down))
                                                        addCorrectItem (V1, V2)
                                                    else ERROR ("Window too large") endif;
additem (addCorrectItem (V1, B), M) = if ((getLimits (M).top >= getLimits (V1).top)
                                                    and (getLimits (M).down <= getLimits (V1).down))
                                                        addCorrectItem (V1, M)
                                                    else ERROR ("Menu too large") endif;
additem (addCorrectItem (V1, B), B1) = if ((getLimits (B1).top >= getLimits (V1).top)
                                                    and (getLimits (B1).down <= getLimits (V1).down))
                                                        addCorrectItem (V1, B1)
                                                    else ERROR ("Button too large") endif;
```

```
additem (addCorrectItem (V1, M), V2) = if ((getLimits (V2).top >= getLimits (V1).top)
                                        and (getLimits (V2).down <= getLimits (V1).down))
                                                addCorrectItem (V1, V2)
                                        else ERROR ("Window too large") endif;
additem (addCorrectItem (V1,M), M1) = if ((getLimits (M1).top >= getLimits (V1).top)
                                        and (getLimits (M1).down <= getLimits (V1).down))
                                                addCorrectItem (V1, M1)
                                        else ERROR ("Menu too large") endif;
additem (addCorrectItem (V1, M), B) = if ((getLimits (B).top >= getLimits (V1).top)
                                        and (getLimits (B).down <= getLimits (V1).down))
                                                addCorrectItem (V1, B)
                                        else ERROR ("Button too large") endif;


{ ***********************************************
  **        User Interface Specification: BUTTON
  *********************************************** }

graphic object Button;

        Button: point, point, string , func( -> bool) -> Button;

        VisibleButton: point, point, string, func ( -> bool) -> Button
                where VisibleButton (p1, p2, s, f) = rectangle(p1,p2)+text(p1,s);

Functions

        ShowButton: Button -> Button;
        HideButton: Button -> Button;
        getLimits<top,down>: Button -> point, point;
        move: Button, point -> Button;
        is_in: Button, point -> bool;
        event: Button, point -> bool;

Axioms
        var     p1,p2,pt: point
                s: string
                f: func ( -> bool);

ShowButton (Button (p1, p2, s, f)) = VisibleButton (p1, p2, s, f);
ShowButton (VisibleButton (p1, p2, s, f)) = VisibleButton (p1, p2, s, f);
HideButton (Button (p1, p2, s, f)) = Button (p1, p2, s, f);
HideButton (VisibleButton (p1, p2, s, f)) = Button (p1, p2, s, f);

getLimits (Button (p1, p2, s, f)).top = p1;
getLimits (Button (p1, p2, s, f)).down = p2;
getLimits (VisibleButton (p1, p2, s, f)).top = p1;
getLimits (VisibleButton (p1, p2, s, f)).down = p2;
move (Button (p1, p2, s, f), pt) = Button (p1+pt, p2+pt, s, f);
move (VisibleButton (p1, p2, s, f), pt) = VisibleButton (p1+pt, p2+pt, s, f);

is_in (Button (p1, p2, s, f), pt) = false;
is_in (VisibleButton (p1, p2, s, f), pt) = if ((pt>=p1) and (pt<=p2)) true else false endif;
event (Button (p1, p2, s, f), pt) = false;
event (VisibleButton (p1, p2, s, f), pt) = f;
```

```
{**********************************************
 ****    User Interface Specification: MENU
 ********************************************** }

graphic object Menu;

        import Cascade, Button;

        Menu: -> Menu;

Functions

        openMenu: Menu -> Menu;
        closeMenu: Menu -> Menu;
        additem: Menu, Cascade -> Menu;
        additem: Menu, Button -> Menu;
        getLimits<top,down>: Menu -> point, point;
        move: Menu, point -> Menu;
        event: Menu, point -> bool;
        is_in: Menu, point -> bool;

Axioms
        var     pt : point
                M : Menu
                B : Button
                C : Cascade;

openMenu (Menu ()) = Menu ();
openMenu (additem (M, C)) = additem (openMenu (M), ShowCascade (C));
openMenu (additem (M, B)) = additem (openMenu (M), ShowButton (B));

closeMenu (Menu ()) = Menu ();
closeMenu (additem (M, C)) = additem (closeMenu (M), HideCascade (C));
closeMenu (additem (M, B)) = additem (closeMenu (M), HideButton (B));

getLimits (Menu ()).top = pt;
getLimits (Menu ()).down = pt;
getLimits (additem (M, C)).top = if (getLimits (M).top > getLimits (C).top) getLimits (M).top
                                 else getLimits (C).top endif;
getLimits (additem (M, C)).down = if (getLimits (M).down < getLimits (C).down) getLimits (M).down
                                 else getLimits (C).down endif;
getLimits (additem (M, B)).top = if (getLimits (M).top > getLimits (B).top) getLimits (M).top
                                 else getLimits (B).top endif;
getLimits (additem (M, B)).down = if (getLimits (M).down < getLimits (B).down) getLimits (M).down
                                 else getLimits (B).down endif;

move (Menu (),  pt) = Menu ();
move (additem (M, C), pt) = additem (move (M, pt), move (C, pt));
move (additem (M, B), pt) = additem (move (M, pt), move (B, pt));

is_in (Menu (), pt) = false;
is_in (additem (M, C), pt) = if  (is_in (C, pt)) true else is_in (M, pt) endif;
is_in (additem (M, B), pt) = if  (is_in (B, pt)) true else is_in (M, pt) endif;

event (Menu (), pt) = false;
event (additem (M, B), pt) = event (M, pt) or event (B, pt);
event (additem (M, C), pt) = event (M, pt) or event (C, pt);
```

{
**** User Interface Specification: CASCADES
** }**

graphic object *Cascade*;

import Button, Menu;

Cascade: Button, Menu -> Cascade;

Functions

ShowCascade: Cascade -> Cascade;
HideCascade: Cascade -> Cascade;
move : Cascade, point -> Cascade;
getLimits<top,down>: Cascade -> point, point;
is_in: Cascade, point -> bool;
event: Cascade, point -> bool;
pick<s,b>: Cascade, point -> Cascade, bool;

Axioms

var C, C1: Cascade
p1, p2, pt: point
B : Button
M : Menu;

ShowCascade (Cascade (B, M)) = Cascade (ShowButton (B), openMenu (M));
HideCascade (Cascade (B, M)) = Cascade (HideButton (B), closeMenu (M));

move (Cascade (B, M), pt) = Cascade (move (B, pt), move (M, pt));

getLimits (Cascade (B, M)).top = getLimits (M).top;
getLimits (Cascade (B, M)).down = getLimits (M).down;

is_in (Cascade (B, M), pt) = is_in (B, pt);

event (Cascade (B, M), pt) = if (is_in (B, pt)) pick (Cascade (B, M)).b endif;

pick (Cascade (B, M)).s = ShowCascade (Cascade (B, M));
pick (Cascade (B, M)).b = event (M, pt);