SPECIFICATION OF GUI FRAMEWORKS*

F. Losavio, F. Marchena, A. Matteo

Centro de Ingeniería de Software Y Sistemas ISYS,
Facultad de Ciencias, Universidad Central de Venezuela,
Apdo. 47567, Los Chaguaramos 1041-A, Caracas, Venezuela
flosavio @conicit.ve / @anubis.ciens.ucv.ve, fmarchen @anubis.ciens.ucv.ve
amatteo@conicit.ve / @anubis.ciens.ucv.ve

Abstract

Interactive systems enhance the usability of the application, in the sense of providing a convenient access to their services, allowing the user to spend less time learning the application and to produce results quickly. The graphical user-interface is the vehicle for achieving this usability. Frameworks, or semifinished generic architectures, have been successfully used in the development of GUI (Graphical User-Interfaces). Besides, multiagent models are used to describe the architecture of interactive systems. Moreover, this architecture must reflect the paradigm of the separation between the abstract or semantic aspects of the system and its presentation to the final user. Our main goal is to present a specification of the PAC (Presentation, Abstraction, Control) GUI framework. This specification, written in a pseudoformal language, is directly used to implement the GUI agents in any object-oriented target language. In this paper, an example of a small application, a simplified graph editor, is developed in C++, applying the specification.

Keywords: framework, graphical user-interface, multiagent model, reusable software design, design patterns, interactive system, usability, user-interface design, object-oriented programming, software engineering

1. INTRODUCTION

Our main goal is to illustrate an experience of interactive software development [7], [8], [9], [10] using the PAC (Presentation, Abstraction and Control) model approach [4], presenting a framework for a PAC agent [8] and the specification of its public abstract classes, which have been practically used for implementing object-oriented (OO) applications. The specification is written in a pseudoformal language, and its purpose is to serve as a general guideline for coding the concrete classes customized to the particular application, obtained by subclassing the public abstract classes of the framework. A simplified graphical editor is constructed for workstations under the Unix[1]/X-Window[2]/OSF/Motif[3] platform, according to the PAC framework specification. Part of the code of a particular interface agent (the Editor agent) is presented as an example, to show the paractical use of the framework. The complete application code may be found at the ISYS Center's home page: http://anubis.ciens.ucv.ve/SPANISH/publicaciones97.html, and the interested readers may retrieve and run the whole application. On the basis of our experience, we will be mostly concerned with the PAC model, nevertheless the main differences with the well known MVC (Model-View-Controller) model [5] are pointed out.

Beside the first introductory section, this work is structured into Section 2 concerned with the description and specification of the PAC framework and Section 3 focused on an example to custumize the PAC framework specification. Finally the Conclusion and an Appendix, showing the C++ code [12] corresponding to the Presentation perspective of the Edition agent, are presented.

1.1 Patterns and frameworks

The approach to software design with patterns or templates [4] that can be applied in many different situations, captures the experience involved in designing OO software. Patterns, at a given level of abstraction, are descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context. A design pattern names, abstracts and identifies the key aspects of a

* This research is supported by the New Technology Program of the BID-CONICIT I-09 Project, the CONICIT MOODe S1-9500512 Project and the CDCH OOMGRIN No. 03-13-3483-95 Project

[1] Unix is a registered trademark of Unix System Laboratories.

[2] X-Window is a registered trademark of the Massachusetts Institute of Technology.

[3] Motif is a registered trademark of Open Software Foundation.

common design structure that make it useful for creating a reusable OO design. The design pattern identifies the participating classes and instances, their roles and collaborations and the distribution of responsibilities, providing also a sample code to illustrate an implementation. In [4], C++ [12] and/or Smalltalk [5] codes are given, and these languages have been selected on the basis of experience in the language, increase of its popularity and the language facility of expression. Actually, some patterns can be expressed more easily in one language than in another. A framework is defined as a reusable semifinished architecture for various application domains [11]. It is constituted by a set of classes, that may conform several design patterns, conceived for working together and it represents a generic subsystem that can be instantiated. The instantiation is achieved developing subclasses from the public abstract classes of the framework, called sometimes "hot spots". A developer using a framework must know the hot spots for a given problem and know how to adapt them to the application's needs.

1.2 Architecture of Interactive Systems

Interactive systems allow a high degree of user interaction through their graphical user-interface, enhancing the usability of the application, in the sense of providing a convenient access to their services, allowing the user to spend less time learning the application, to produce results more quickly [1]. The architecture of interactive systems must reflect the paradigm of the separation between the abstract or semantic aspects (functional core) of the system and its presentation. Usually the core does not change much in time, remaining relatively stable. User-interfaces, however, are often subject to change and adaptation. For example, systems may have to support different user-interface standards, customer-specific "look and feel" metaphors, or interfaces that must be adjusted to fit into a customer's business processes. The system's architecture must support the adaptation of the user-interface without causing major effects to the application specific functionality or the underlying data model.

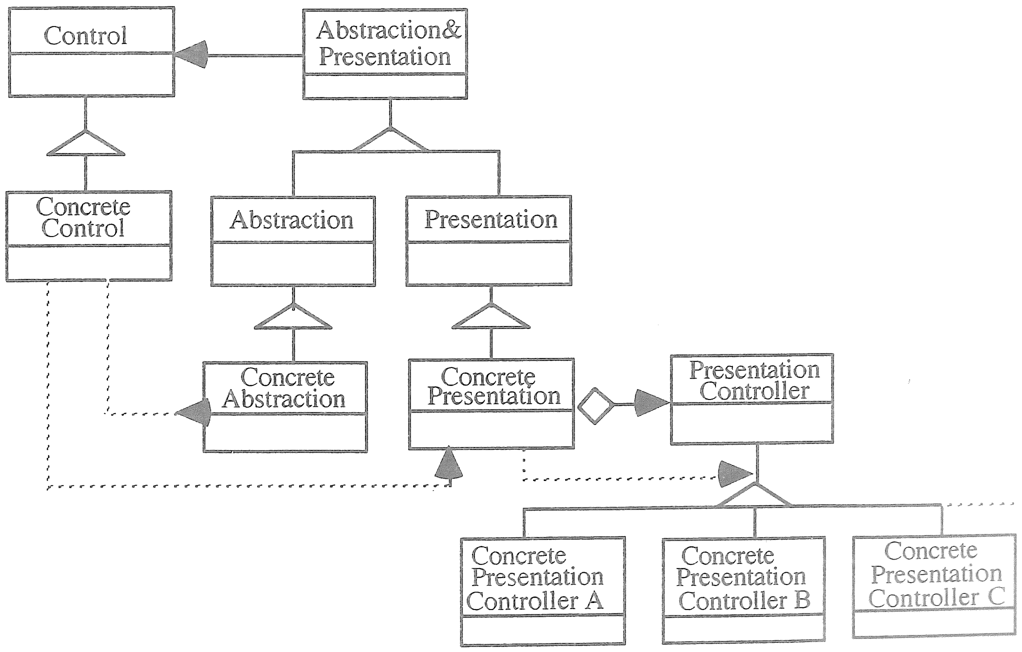
1.3 Multiagent Model

The multiagent model [3], [5] is used to structure the architecture of interactive systems. It is inspired from the stimuli-answers systems, which are organized as a set of cooperating *agents*, reacting at external events (stimuli) and generating events (answers). An *event* or *stimulus* is of a certain kind, it holds some information depending on its kind, it is produced by an *emissary* and received by a *receptor*. An agent may be seen as a *processor*, with receptors and emissaries for capturing and producing events. It is constituted by a two level memory, one to register detected events, the other to memorize a state, and it is characterized by a modular organization, parallel execution of processes and event-driven communications. This model adds another dimension, the parallelism, to the traditional models used in the construction of graphical user-interfaces for interactive systems. Notice that the basic OO notions are present in the agent concept: a class and its *instances* may define a category of agents; the *operations* are the instructions of the processor, the *attributes* constitute the memory elements, modeling the agent's state. The *constraints* (e.g. preconditions reacting to the activation of an operator) specify the semantics of the processor's instructions. Moreover, agents may be connected by *inheritance* and/or *association* relations. An important issue of the multiagent model, as for the OO paradigm, is that an agent defines the granularity and modularity of the system. It is then possible to modify a behaviour without compromising the whole system. The agents we are interested in are called *interface agents*, since they are responsible of capturing man-machine interaction.
Frameworks have been successfully used in the development of graphical user-interfaces. The MVC (Model-View-Controller) model [5] for building graphical user-interfaces (GUI) is actually considered one of the first known frameworks [7]. Two basic frameworks are used for implementing multiagent models, providing a fundamental structural organization for the architecture of interactive software systems: - The *Model-View-Controller* (MVC), which models an interactive application as three separate components or perspectives, the *model*, containing the core functionality and data; the *view*, displaying or presenting information to the user and the *controller* for handling user inputs. Views and controllers together constitute the user-interface. Consistency between the view-controller pair and the model is ensured by a change-propagation mechanism.
- The *Presentation-Abstraction-Control* (PAC) [3], which structures the system in a hierarchy of cooperating interface agent. Each agent is responsible for a specific aspect of the application's functionality and consists also of three perspectives, the *presentation* corresponding to the user-interface or MVC view-controller pair, the *abstraction* or MVC model and the *control*, which is responsible for the consistency

between the presentation and the abstraction and for communicating the PAC agents. Moreover, PAC agents model the whole interactive system, because the higher level in the hierarchy of agents is constituted by a special top agent representing the whole application. Its abstraction corresponds to the functional core and data model of the system. The intermediate levels are constituted by the agents representing the windows and subwindows of the system GUI. The inferior level corresponds to elementary PAC agents, which can be implemented directly reusing particular toolkit libraries. The PAC framework is presented in the next section.

2. THE PAC FRAMEWORK

In what follows, the framework corresponding to a PAC agent [8] is shown in Figure 1, according to the OMT [12] based notation used in [4]. Notice that the Control, Abstraction, Presentation and Presentation Controller classes are the hot spots or public abstract classes of the framework, from which the subclasses or concrete classes customizing a particular application are derived. The Abstraction&Presentation abstract class establishes the communication relationship of the Abstraction and Presentation classes with the Control class, according to the PAC model principles. Three main patterns characterize the PAC model: *Mediator, Strategy and Factory Method.* [1], [4], [8]. Mediator is the most relevant pattern involved, because it reflects the role of the PAC control, in the sense of mediating for the coherence between the presentation and the abstraction perspectives. Notice that, since a PAC agent may be decomposed in subagents, due to the PAC agents hierarchy, each subagent has its own control, and nested views are treated as subagents. The Strategy pattern is used in the same way as for MVC, attaching a view to a controller, allowing to change the way a view responds to user inputs. Notice that Strategy defines a family of algorithms, encapsulating each one and making them interchangeable. It lets the algorithm vary independently from clients that use it. Finally, the Factory Method pattern is used to relate a particular situation in the presentation, with the different encapsulated treatments of the user inputs. In Figure 1, we use a dashed arrow pointing to the inheritance relation, instead of pointing to a particular subclass, because, at framework level, we don't know which one of the algorithm has to be related to the particular presentation instance. Actually, this will be known only at the moment of instantiating the framework with a particular application. Notice also that we have added the dashed line, continuing the inheritance relation line, meaning that there may by any number of algorithms, depending on the application. We have adopted these slightly different notations, because the OMT [12] based notation used in [4] does not model these situations.



Control
Abstraction& Presentation
Concrete Control
Abstraction
Presentation
Concrete Abstraction
Concrete Presentation
Presentation Controller
Concrete Presentation Controller A
Concrete Presentation Controller B
Concrete Presentation Controller C

Figure 1. The PAC framework

As we have seen, the graphical notation of [4] lacks semantic power, presenting ambiguities and situations that cannot be easily modeled, since they are not expressed in a more formal language. In the next section, we present a semiformal description of the abstract classes (hot spots) constituting the framework, written in a pseudoformal language. We have applied in practice this description [9], [10] and we have noticed that it is quite easy to use as a reusable design guideline for implementing the corresponding concrete classes.

2.1 THE PAC FRAMEWORK SPECIFICATION

Let A be an interface object or agent in the PAC model. Taking the definition of the PAC frameworks, whose diagram showing the relations among the diffferent classes is shown in Figure 1, we denote:

C, the abstract class CONTROL
AP, the abstract class ABSTRACTION&PRESENTACION
A, the abstract class ABSTRACTION
P, the abstract class PRESENTATION
PC, the abstract class PRESENTATION CONTROLLER
CC, the class CONCRETE CONTROL, c an object of CC
CA, the class CONCRETE ABSTRACTION, a an object of CA
CP, the class CONCRETE PRESENTATION, p an object of CP
CPC, the class CONCRETE PRESENTATION CONTROLLER (we can have several classes of this kind, according to the PAC framework definition)

Let A_f be the parent agent of A in the model corresponding to the application, then we denote by CC_f., the class implementing the control perspective of agent A_f. Let c_f be an object of CC_f.
Let A_{si} be an agent son i of the agent A in the PAC model corresponding to the application; then we denote by CC_{si} the class implementing the control perspective of agent A_{si}. Let c_i be an object of CC_{si}.
Let Ω be the set of all the classes defined by the toolkit used for developing the GUI component of the application. W will denote a particular class belonging to Ω and w an object of W.
Let us consider a disjoint partition of Ω, $\Omega = \Omega1 \cup \Omega2$, where: $\Omega1$ represents the set of all the classes whose objects may be composite interface objects, such as menu, window etc.; $\Omega2$ represents the set of all the classes whose objects are atomics or elemental interface obects, such as button, field, etc.
The symbol $\uparrow$, indicates that the attribute with this symbol is of reference type. Besides, we will use the standard elements: if, and, or, not, $\Rightarrow$, $\in$, $\exists$, not $\exists$ (meaning doesn't exist)

When the PAC framework is used to implement an interface object, we assume a specification and/or a description of the different events that may happen on the interface object, that is to say the all the possible interactions with the interface object. This description is assumed written in natural language as a list of all the possible events.

SPEC ABSTRACTION&PRESENTATION (SPEC AP)

- **Description:** Abstract class AP, superclass of classes A and P
- **Attributes:**

  Control of type $\uparrow$. The reference is the object $c \in CC$.
- **Methods:**

  Put_reference

  precondition: None

  postcondition: If $a \in CA$, $p \in CP$, $c \in CC \Rightarrow$
  ($a \uparrow c$ or $p \uparrow c$)

SPEC PRESENTATION (SPEC P)

- **Description:** Abstract class P (hot spot), superclass of the concrete class CP.
- **Attributes:**

  Graphicomponent _top the type is $W \in \Omega1$. This type will be indicated by the user of the framework in the class CP.

Eventually other attributes are defined, corresponding to toolkit interface objects, required by the interface object to be constructed. Suppose r toolkit objects are required, the following objects are defined:

Graphicomponent_1, Graphicomponent_2, ...Graphicomponent_r, where the type of each attribute is respectively W1,W2,...Wr, $\forall$ i, (i=1..r), $Wi \in \Omega$

- **Methods:**

  Create_presentation

precondition: $\exists\, c \in CC$ and not $\exists\, p \in CP$

postcondition: $(\exists\, p \in CP)$ and $(\exists\, w \in W.\ W \in \Omega 1)$ and if $(r \geq 1 \Rightarrow (\forall\, i, (i=1..r), \exists\, wi \in Wi))$

Destroy_presentation

precondition: $\exists\, p \in CP$

postcondition: (not $\exists\, p \in CP$) and (not $\exists\, w \in W$) and if $(r \geq 1 \Rightarrow (\forall\, i, (i=1..r),$ not $\exists\, wi \in Wi))$

The set of operations given below must be defined for each one of the events identified in for the corresponding interface object to be implemented.

Detect_event_i

precondition: Event i has occurred

postcondition: The method notify_event_i has been called

Notify_event_i

precondition: Event i has been detected

postcondición: A message to the object $c \in CC$ has been sent notifying the occurrence of event i

We also have a number q of updating operations, whose effect is an update of the presentation of the interface objet. $\forall$ i=1..,q we have

Update_i

precondition: None

postcondición: the corresponding update algorithm encapsulated into a Presentation Controller Class has been called

SPEC CONTROL (SPEC C)

- **Description:** Abstract class C, superclass of class CC
- **Attributes:**

Cabstraction	of type $\uparrow$. The reference to the object $a \in CA$.
Cpresentation	of type $\uparrow$. The reference to the object $p \in CP$.
Ccontrol_father_name	of type $\uparrow$. The reference to the object $c_f \in CC_f$.

If agent A has sons, assume a certain number j, the following attributes have to be defined:

Ccontrol_son_name1, Ccontrol_son_name2,...... Ccontrol_son_name j, each of type $\uparrow$, the reference to object $c_i \in CC_{si}$, $\forall$ i, (i=1...j)

- **Methods:**

Create_control

precondition: $(\exists\, c_f \in CC_f)$ and (not $\exists\, c \in CC$)

postcondition: $(\exists\, c \in CC)$ $(\exists\, a \in CA)$ and $(\exists\, p \in CP)$ and (if $j \geq 1 \Rightarrow \forall\, i, (i=1...j)\ \exists\, c_i \in CC_{si}$)

Destroy_control

precondition: $\exists\, c \in CC$

postcondition: not $\exists\, c \in CC$ and not $\exists$ ($a \in CA$ and $p \in CP$ and if $j \geq 1 \Rightarrow \forall i, (i=1..j), c_i \in CC_{si}$)

The set of methods given below have to be defined for each one of the events identified for the corresponding interface object to be implemented (we suppose n events, i=1..n).

Notify_abstraction_event_i

precondition: event i has occured and a message belonging to object $c \in CC_f$ is called to verify conditions on the state of the interface objects, on which A is dependent

postcondition: The message has been sent to object $a \in CA$, in order to perform a query or update its state

The PAC model establishes a hierachical architecture, where the communication among different agents, located in different hierarchical levels, is achieved through the control perspective (CC classes corresponding to these agents). Then we distinguish communications toward the father or sons agents. The following methods are defined with respect to these communications:

Com_father_update_application_abstraction

precondition: None

postcondition: a message has been sent to $c_f \in CC_f$ to update the abstraction of the application

Com_father_information_state_other_agents

precondition: None

postcondition: a message has been sent to $c_f \in CC_f$ to ask about the state of other agents

Com_father_update_other_agents

precondition: None

postcondition: a message has been sent to $c_f \in CC_f$ to provide information to update the state of other agents

Com_son_name_state

precondition: None

postcondition: a message has been sent to $c_i \in CC_s$ to ask about the state of the A_{si} son agent state

Com_son_name_update

precondition: None

postcondition: a message has been sent to $c_i \in CC_s$ to update the A_{si} son agent

SPEC ABSTRACTION (SPEC A)

- **Description:** Abstract class A, superclass of class CA
- **Attributes:**

Structure its type is a complex structure defined to represent the state of the presentation perspective at each instant.

- **Methods:**

Create_presentation

precondition: $\exists c \in CC$ **and not** $\exists a \in CA$

postcondition: $\exists a \in CA$

Destroy_presentation

precondition: $\exists a \in CA$

postcondition: not $\exists a \in CA$

A set of methods is available to observe the state of an object of this class. That is to say, at a given instant the information represented in *structure* can be observed. Let us suppose that a certain number p of these methods is required.

Observer i

precondition: None

postcondition: The required value is determined

A set of methods allowing to update the state of an object of this class must be defined. That is to say, *structure* must be updated. Let us suppose that a certain number k of these methods is required.

Update i

precondition: None

postcondition: The state of the object $a \in CA$ has been updated

SPEC PRESENTATION CONTROLLER (SPEC PC)

- **Description:** Abstract class PC, superclass of a set of classes named CPC. This class defines the name of the method with its parameters, which will be implemented differently in each of the CPC classes defined for the interface object that has to be implemented.
- **Methods:**

Algorithm_name

3. A SIMPLE EXAMPLE ILLUSTRATING THE INSTANTIATION OF THE PAC FRAMEWORK

Let us consider a small application, a simplified graph editor for assisting researchers in graph theory. The main goal of the system is to be used by students and researchers, for drawing graphs and computing graph properties. Figure 2, below shows a picture of the window, with two zones: *Menu,* showing from left to right, the edition facilities for graph drawing, *node* (circle), *arc* (line) and finally the *exit* option (square). The drawing area, *Edition*, is shown with the picture of a three node graph, one already selected (node shown in black).

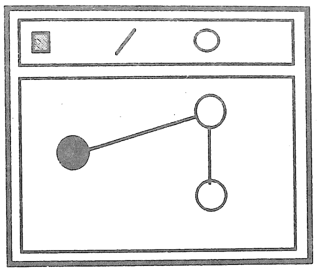
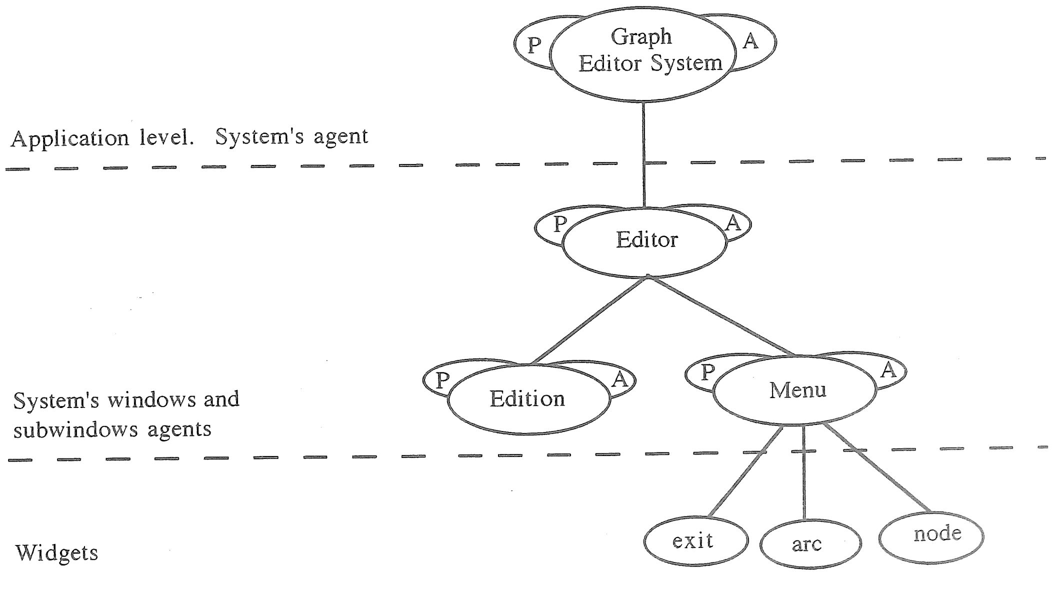


Figure 2. Picture of the window of the Editor tool

Suppose that a three buttons mouse is available as the user-interface pointing divice, and that we want to draw a node in the drawing area. The figure will be displayed as a consequence of the following actions, supposing the node shape selected on the menu bar (*Menu* interface object) and the cursor moved to the drawing area (*Edition* interface object): (A) the third mouse button is clicked and released and the figure is displayed on the drawing area. The display of the node as a consequence of the click is implemented by an algorithm that will be called A. Moreover, in the drawing area, the following situations may occur when the first mouse button is clicked: (B) If the cursor is on the node, the figure will be selected; the actions corresponding to this situation are treated by algorithm B. (C) If we are in situation (A) and the cursor is outside the node, the figure will be deselected; algorithm C handles this situation. (D) If the cursor is outside the node, and situation (A) has not occurred, nothing happens. Notice then that four different situations are possible in the drawing area, which is under control of the *Edition* interface object: the figure is drawn, the figure is selected, the figure is deselected, no action is taken, and three different algorithms handle the relevant situations. The architecture of the system, using the PAC model, is shown in Figure 3. The big oval represent the control perspective of the agent, with the agent's name in it, and the small ovals correspond to the Abstraction and Presentation perspectives, respectively. The straight lines represent communication between the agents. Notice the hierarchy of the system. The superior level is constituted by the Graph Editor System agent, representing the whole system. The presentation of this agents is usually constituted by an icon, to allow the access to the whole application. The intermediate level is constituted by agents representing the main window (Editor agent), containing the system main functionalities, the Edition and Mnue agents, respectively, sub windows of the main window. The inferior level is contituted by the elementary agents which cannot be decomposed further, the exit, arc and node buttons, respectively.



P
Graph Editor System
A
Application level. System's agent
P
Editor
A
P
Edition
A
P
Menu
A
System's windows and subwindows agents
exit
arc
node
Widgets

Figure 3. Architecture of the Graph Editor System

The C++ implementation of the Presentation perspective of the the Edition agent, according to the PAC framework specification given in 2.1, is shown in the Appendix. Notice that the list of events required in the specification is present in the Presentation code.
The complete C++ implementation of the Edition Agent may be retrieved at the ISYS home pages: http://anubis.ciens.ucv.ve/SPANISH/publicaciones97.html.

4. CONCLUSION

The specification of the PAC framework's abstract classes, described through this paper, aims to standardize the elements of the PAC model, appearing in the definition of GUI agents, independently from the application to be developed and from the object-oriented target language used to implement the application. Moreover, this specification has been practically used as a guideline for implementation in projects on multimethods-case environments developed at the ISYS Center [7], [8], [9], resulting very attractive for modifying and extending the existing code and for separating works among programmers teams, easing the overall control of the project. Finally, to experiment the flexibility of the framework with respect to robustness and portability, its implementation in Java [14] is an undergoing work [10]. Moreover, at present, we are studying the impact of the PAC approach on distributed applications.

5. REFERENCES

1 . BUSCHMANN F., MEUNIER R., RHONERT H., SOMMERLAD P., STAL M. "Pattern-Oriented Software Architecture. A System of Patterns", Jhon Wiley & Sons, 1996.
2 . COLLINS D., "Designing Object-Oriented User Interfaces", Benjamin/Cummings Publishing Company, Inc. 1995.
3 . COUTAZ J., "Developing Software for the User Interface", Bass L., Coutaz J. Addison Wesley Cummings Publishing Company (1991).
4 . GAMMA E., HELM R., JOHNSON R.. VLISSIDES J. "Design Patterns. Elements of Reusable Object-Oriented Software" Addison Wesley Publishing Co., 1994.
5. GOLDBERG A., "Smalltalk-80 The Interactive Programming Environment", Addison-Wesley 1984.
6 . KRASNER G. E., POPE S.T. "A Cookbook for using the Model-View-Controller user-interface paradigm in Smalltalk-80" Journal of Object-Oriented Programming 1(3), 1988.
7 . LOSAVIO F., MATTEO A. "A Method for User-Interface Development. To appear in Journal of Object Oriented Programming, Dec. 1997.
8 . LOSAVIO F., MATTEO A. "Object-Oriented User-Interface Design Based on Agents Frameworks", Proceedings of the International Conference on Technology of Object-Oriented Languages and Systemas, TOOLS USA '96, Santa Barbara, California, U.S.A., July-August 1996.
9 . LOSAVIO F., MARCHENA F., MATTEO A. " Frameworks for Interactive Systems Development", Research Report, ISYS RI/01/97, No. 23, Caracas, January 1997.
10 . BENCOMO N., LOSAVIO F., MARCHENA F., MATTEO A. "Java Implementation of User-Interface Frameworks", To appear in Proceedings of the International Conference on Technology of Object-Oriented Languages and Systemas, TOOLS USA '97, Santa Barbara, California, U.S.A., July-August 1997.
11 . PREE W. "Design Patterns for Object-Oriented Software Development", Addison Wesley 1994.
12 . RUMBAUGH J., BLAHA M., PREMERLANI W., EDDY F., LORENSEN W. "Object-Oriented Modeling and Design" Prentice Hall International, Inc (1991).
13 . STROUSTRUP B., "The C++ Programming Language", Second Edition, Addison-Wesley Publishing Company, 1993.
14. SUN MICROSYSTEMS COMPUTER COMPANY , "The Java Language Specification" Release 1.0 Alpha 3. Sun Microsystems Computer Company, May 1995.

APPENDIX

```
//EDITION'S PRESENTATION
//pedition.h
#include "xt.h"
#include "utility_edition.h"
#include "apedition.h"
#include "dnode_controller.h"
#include "darc_controller.h"
#include "snode_controller.h"
#include "sarc_controller.h"
#include "carea_controller.h"
const int radius=17;
//Presentation perspective for Edition agent
//this class inherits from APEdition and XT
class PEdition:APEdition,Xt {
          Display *dpy;
          Window win;
          graphics_data data;
          Pixel black,white;
Widget Graphicomponent_top,
Graphicomponent_canvas;
//objects that encapsulate algorithms
DrawNodeController DNodeC;
DrawArcController DArcC;
SelectNodeController SNodeC;
SelectArcController SArcC;
ClearAreaController CAreaC;
//create graphic contexts
GC create_gc_image(Widget w);
GC xs_create_xor_gc(Widget w);
//initialization of graphic data
void init_data(graphics_data *data);
//detects when mouse button 1 is pushed
void   detect_event_button1_pushed(Widget
w, PEdition *_this,XEvent * event);
//detects when mouse button 3 is pushed
void
```

```
                detect_event_button3_pushed(Widget
                w,PEdition *_this,XEvent * event);
                //detects when mouse is moved holding a
                //button
                void detect_event_button_motion(Widget
                w,PEdition *_this,XEvent * event);
                //detects when a mouse's button is released
                void detect_event_button_released(Widget
                w,PEdition *_this,XEvent * event);
                Pixel get_pixel(Widget w,char *colorname);
                //builds the interface
                void draw_drawingarea(Widget form.
                Widget menu);
                //detects when expose events occur
                void detect_event_refresh(Widget w,PEdition
                *_this,XmAnyCallbackStruct *call_data);
public:
                //creates Presentation object
                PEdition(CEdition *c);
                //destroys Presentation object
                ~PEdition();
                void update_place_constructor(tposition
                p1,tposition p2,GC gc);
                void update_node_selection(tposition  pto, tposition
                pt1);
                void update_arc_selection(tposition pto,
                tposition pt1);
                void update_draw_node(tposition p,int  r,GC gc);
                void update_draw_arc(tposition s,  tposition d,GC
gc);
                void update_clear();
                //notifies Control of the refresh event
                void notify_event_refresh();
                //notifies Control of the mouse's button 1
                //pushed event
                void  notify_event_button1_pushed();
                //notifies Control of the mouse's button 3
                //pushed event
                void  notify_event_button3_pushed();
                //notifies Control of the mouse's button
                //released event
                void notify_event_button_released();
};
//pedition.cc
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <X11/Xutil.h>
#include <Xm/Xm.h>
#include <Xm/ScrolledW.h>
#include <Xm/ScrollBar.h>
#include <Xm/DrawingA.h>
#include "pedition.h"
//creates Presentation object
PEdition::PEdition(CEdition *c):APEdition(c)
{
//builds the interface
draw_drawingarea(control
->com_father_information_state_widget_form(),
control
->com_father_information_state_widget_menu());
}
//destroys Presentation object
PEdition::~PEdition()
{
//destroys interface
XtDestroyWidget(Graphicomponent_top);
}
//creates a graphic context
GC PEdition::create_gc_image(Widget w)
{ XGCValues values;
GC gc ;
Arg args[2];
XtSetArg(args[0], XtNforeground, &values.foreground);
XtSetArg(args[1], XtNbackground, &values.background);
XtGetValues(w, args, 2);
values.fill_style = FillSolid;
values.line_style = LineSolid;
values.line_width = 2;
values.function = GXcopy;
values.cap_style = CapButt;
values.join_style = JoinMiter;
gc = XCreateGC(XtDisplay(w), XtWindow(w), GCBackground |
GCForeground | GCFillStyle | GCLineStyle | GCLineWidth |
GCFunction | GCCapStyle | GCJoinStyle, &values);
return (gc);
}
//creates a graphic context
GC PEdition::xs_create_xor_gc(Widget w)
{ XGCValues values;
GC gc;
Arg args[2];
values.function = GXxor;
XtSetArg(args[0], XtNforeground, &values.foreground);
XtSetArg(args[1], XtNbackground, &values.background);
XtGetValues(w, args, 2);
values.foreground = values.foreground ^ values.background;
values.fill_style = FillSolid;
values.line_style = LineOnOffDash;
values.join_style = JoinRound;
values.line_width = 2;
values.cap_style = CapButt;
gc = XCreateGC(XtDisplay(w), XtWindow(w),
GCFunction|GCForeground|GCFillStyle|GCLineStyle|GCJoinStyle|
GCLineWidth|GCCapStyle, &values);
return (gc);
}
//initialization of graphic data
void PEdition::init_data(graphics_data *data)
{ Arg args[2];
data->gc =  create_gc_image(Graphicomponent_canvas);
        data->xorgc = xs_create_xor_gc(Graphicomponent_canvas);
       XtSetArg(args[0], XtNforeground,
&data->foreground);
        XtSetArg(args[1], XtNbackground,
&data->background);
        XtGetValues(Graphicomponent_canvas, args, 2);
}
void PEdition::update_draw_node(tposition p,int r,GC gc)
{ static gc_pos gp;
gp.gc=gc;
gp.value=r;
gp.pos1=p;
//call to encapsulated algorithm
DNodeC.Algorithm(Graphicomponent_canvas,&gp); }
void Pedition::update_draw_arc(
tposition s,tposition d,GC gc)
{ static gc_pos gp;
gp.gc=gc;
gp.pos1=s;
gp.pos2=d;
//call to encapsulated algorithm
DArcC.Algorithm(Graphicomponent_canvas,&gp); }
void PEdition::update_arc_selection(tposition pto, tposition pt1)
{ XGCValues values;
GC gc;
static gc_pos gp;
values.foreground =  data.foreground ^ data.background;
values.fill_style = FillSolid;
values.function = GXcopy;
gp.gc = XCreateGC(dpy,
RootWindowOfScreen(XtScreen(Graphicomponent_canvas)),
GCForeground | GCFillStyle | GCFunction , &values);
gp.pos1=pto;
gp.pos2=pt1;
//call to encapsulated algorithm
SArcC.Algorithm(Graphicomponent_canvas,&gp); }
void PEdition::update_node_selection(tposition pto, tposition
pt1)
{  XGCValues values;
GC gc;
static gc_pos gp;
values.foreground =  data.foreground ^ data.background;
values.fill_style = FillSolid;
values.function = GXcopy;
gp.gc = XCreateGC(dpy,
RootWindowOfScreen(XtScreen(Graphicomponent_canvas)),
GCForeground | GCFillStyle | GCFunction , &values);
gp.pos1=pto;
gp.pos2=pt1;
//call to encapsulated algorithm
SNodeC.Algorithm(Graphicomponent_canvas,&gp); }
void PEdition::update_clear()
{ static gc_pos gp;
gp.pos1.x=0;
gp.pos1.y=0;
gp.pos2.x=2000;
gp.pos2.y=2000;
//call to encapsulated algorithm
CAreaC.Algorithm(Graphicomponent_canvas,&gp); }
void PEdition::detect_event_refresh(Widget w,PEdition
*_this,XmAnyCallbackStruct *call_data)
{ _this->notify_event_refresh(); }
void PEdition::update_place_constructor(tposition p1,tposition
p2,GC gc)
{ if (control
->observer_local_current_constructor()==NODE)
update_draw_node(p2,radius,gc);
else update_draw_arc(p1,p2,gc); }
//detects when mouse button 1 is pushed
void PEdition::detect_event_button1_pushed(Widget w,PEdition
*_this,XEvent * event)
{ if (event->xbutton.button == Button1)
{              _this->data.ini.x = _this
->data.end.x=event->xbutton.x;
```

```
                _this->data.ini.y = _this
->data.end.y=event->xbutton.y;_this
->notify_event_button1_pushed();
                _this->control
->notify_abstraction_active_button(event
->xbutton.button);   }
}
//detects when mouse button 3 is pushed
void PEdition::detect_event_button3_pushed(Widget w,PEdition
*_this,XEvent * event)
{ if (event->xbutton.button == Button3)
{               _this->data.ini.x = _this
->data.end.x=event->xbutton.x;
                _this->data.ini.y = _this
->data.end.y=event->xbutton.y;
_this->notify_event_button3_pushed();
                _this->control
->notify_abstraction_active_button(event
->xbutton.button);   }
}
//detects when mouse is moved holding a button
void PEdition::detect_event_button_motion(Widget w,PEdition
*_this,XEvent * event)
{ tposition p1,p2,p3;
if (_this->control->observer_active_button() == Button3)
{   p1.x=_this->data.ini.x;
    p1.y=_this->data.ini.y;
    p2.x=_this->data.end.x;
    p2.y=_this->data.end.y;
    p3.x=event->xbutton.x;
    p3.y=event->xbutton.y;
    _this->data.end.x = event->xbutton.x;
    _this->data.end.y = event->xbutton.y;
    _this->update_place_constructor(p1,p2,_this
->data.xorgc);
    _this->update_place_constructor(p1,p3,_this
->data.xorgc);  }
}
//detects when a mouse's button is released
void PEdition::detect_event_button_released(Widget w,PEdition
*_this,XEvent * event)
{ _this->data.size.dx=_this->data.size.dy=34;
_this->notify_event_button_released(); }
Pixel PEdition::get_pixel(Widget w,char *colorname)
{ Display    *dpy = XtDisplay(w);
int scr = DefaultScreen(dpy);
Colormap   cmap = DefaultColormap(dpy, scr);
XColor color, ignore;
if (XAllocNamedColor(dpy, cmap, colorname, &color, &ignore)) {
      XAllocColor(dpy, cmap, &color);
     return(color.pixel);  }
 else
{     printf("!.. Advertencia: No se puede asignar el color %s ..!\n",
colorname);
     return(BlackPixel(dpy, scr));  }
}
//builds the interface
void PEdition::draw_drawingarea(Widget form,Widget menu)
{ Arg args[10];
int n;
Colormap cmap;
XColor unused, color;
Pixel bg_color, fg_ret, top_shadow, bottom_shadow,
select_color;
n=0;
XtSetArg(args[n],XmNtopAttachment,XmATTACH_WIDGET);n++;
XtSetArg(args[n],XmNtopWidget,menu);n++;
XtSetArg(args[n],XmNrightAttachment,XmATTACH_FORM);n++;
XtSetArg(args[n],XmNleftAttachment,Xm
ATTACH_FORM);n++;
XtSetArg(args[n],XmNbottomAttachment,Xm
ATTACH_FORM);n++;
XtSetArg(args[n],XmNscrollingPolicy,Xm
AUTOMATIC);n++;
XtSetArg(args[n],XmNscrollBarDisplayPolicy,
XmSTATIC);n++;
Graphicomponent_top =
XtCreateManagedWidget("sw",xmScrolledWindowWidgetClass,
form,args,n);
Graphicomponent_canvas=XtCreateManagedWidget("draw",
xmDrawingAreaWidgetClass,Graphicomponent_top,args,n);
dpy = XtDisplay(Graphicomponent_canvas);
XtVaGetValues(Graphicomponent_canvas, XmNcolormap, &cmap,
NULL);
XAllocNamedColor(dpy, cmap, "white", &color, &unused);
bg_color = color.pixel;
XmGetColors(XtScreen(Graphicomponent_canvas), cmap,
bg_color, &fg_ret, &top_shadow, &bottom_shadow,
&select_color);
n=0;
XtSetArg(args[n],XmNresizable,TRUE); n++;
XtSetArg(args[n],XtNwidth,2000);  n++;
XtSetArg(args[n],XtNheight,2000);  n++;
XtSetArg(args[n], XmNbackground, bg_color); n++;
XtSetArg(args[n], XmNtopShadowColor, top_shadow); n++;
XtSetArg(args[n], XmNbottomShadowColor, bottom_shadow);
n++;
XtSetArg(args[n], XmNselectColor, select_color); n++;
XtSetArg(args[n], XmNarmColor, select_color); n++;
XtSetArg(args[n], XmNborderColor, fg_ret); n++;
XtSetValues(Graphicomponent_canvas,args,n);
init_data(&data);
black = get_pixel(Graphicomponent_canvas, "black");
xt_add_event_handler(Graphicomponent_canvas,
ButtonPressMask,FALSE,(XtEventHandler)
&detect_event_button1_pushed,(XtPointer) this);
xt_add_event_handler(Graphicomponent_canvas,
ButtonPressMask,FALSE,(XtEventHandler)
&detect_event_button3_pushed,(XtPointer) this);
xt_add_event_handler(Graphicomponent_canvas,
ButtonMotionMask,FALSE,(XtEventHandler)
&detect_event_button_motion,(XtPointer) this);
xt_add_event_handler(Graphicomponent_canvas,
ButtonReleaseMask,FALSE,(XtEventHandler)
&detect_event_button_released,(XtPointer) this);
xt_add_callback(Graphicomponent_canvas,XmNexposeCallback,
(XtCallbackProc) &detect_event_refresh,(XtPointer) this);
XtPopup(control
->com_father_information_state_widget_shell(),XtGrabNone);
win=XtWindow(Graphicomponent_canvas);  }
void PEdition::notify_event_refresh()
{ control->notify_abstraction_refresh(&data); }
void PEdition::notify_event_button1_pushed()
{  control
->notify_abstraction_button1_pushed(&data);   }
void PEdition::notify_event_button3_pushed()
{  control
->notify_abstraction_button3_pushed(&data);   }
void PEdition::notify_event_button_released()
{  control
->notify_abstraction_button_released(&data);   }
```

Francis LOSAVIO received the Doctor degree in Computer Science in 1991 and a 3ème. Cycle Doctor degree in Computer Science in 1985 from the Univ. Paris-Sud, Orsay, France. She also received a MSc degree in Computer Science from the Univ. Simón Bolívar, Venezuela in 1983. At present, she is a Titular Professor at the School of Computer Science, Faculty of Science, Univ. Central de Venezuela and coordinates the ISYS Research Center. Her research includes software engineering environments, methodologies for software development, graphical user ionterfaces, software arhitectures, automatic program construction, formal specifications.
E-mail: flosavio@conicit.ve / @anubis.ciens.ucv.ve

Francisco MARCHENA received the Bachelor's degree in Computer Science in 1997 from the Universidad Central de Venezuela and is member of the ISYS Research Center, Faculty of Sciences, Universidad Central de Venezuela. At present he coordinates the development of the OODEST (Object Oriented DEsign Support environmenT) project at the ISYS Research Center. His research includes protocol analyzers for local area networks and implementation of multiagent models for interactive applications.
e-mail: fmarchen@anubis.ciens.ucv.ve / @strix.ciens.ucv.ve

Alfredo MATTEO received the Doctor degree in Computer Science from the Univ. Paul Sabatier, Toulouse, France in 1984. At present, he is an Aggregate Professor at the School of Computer Science, Faculty of Science, Univ. Central de Venezuela and is member of the ISYS Research Center. His research includes software engineering environments, methodologies for software development, formal specifications and algorithmic complexity.
E-mail: amatteo@conicit.ve / @anubis.ciens.ucv.ve