A METHOD FOR OBJECT-ORIENTED USER-INTERFACE DESIGN BASED ON USE-CASES AND MULTIAGENT MODELS.*

J. Antoneti, F. Losavio, A. Matteo
Centro de Ingenieria de Software y Sistemas ISYS
Facultad de Ciencias, Universidad Central de Venezuela,
Apdo. 47567, Los Chaguaramos 1041-A,
Caracas, Venezuela
flosavio @conicit.ve / flosavio@anubis.ciens.ucv.ve
amatteo@conicit.ve / amatteo@anubis.ciens.ucv.ve

Abstract

The goal of this work is to present an object-oriented method for user interface design based on the use-case approach and multiagent models. The use-case approach is useful for the analysis step of object-oriented software development enhancing an early separation between entity control and interface objects, and multiagent models are widely employed to describe the arquitecture of interactive systems. Moreover, the method enhances adaptability of the resulting systems.

Keywords: object-oriented analysis, object-oriented design, user-interface design, PAC model, framework, use-case, object-oriented methods, graphical user interface.

1. INTRODUCTION.

The use-case approach is widley used to define the requirements from the user´s needs. Furthermore, in recent years, the graphical user interface (GUI) has been considered as fundamental part of the application in order to support human activities. Nevertheless, not much can be said about the design of the graphical user interface component, for which traditional development techniques are applied, now enriched with a wide use of toolkit libraries [HH 89]. The main goal of this paper is to propose an object-oriented (OO) method for GUI development based on the use-case approach and multiagent models. In particular, the OOSE[1] method [Jac&Al 92] is used as a front-end for the requirements and analysis phase of our method, which makes also use of GUI frameworks. A framework is defined as a reusable semifinished architecture for various application domains [Pre 95]. It is constituted by a set of classes, that may constitute several design patterns, [Gam&Al 95] conceived for working together and represents a generic subsystem that can be instantiated. The instantiation is acheived developing subclasses from the public abstract classes of the framework, called "hot spots" [Pre 95], which are seen as predefined places where application specific parts are composed. 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. The MVC (Model-View-Controller) model [Gol 84] for building GUI is considered one of the first known frameworks [KP 88]. Besides this introduction and the conclusions, this paper is divided into four sections. Section 2, gives a brief summary of the OOSE method followed by section 3 with an analogy between OOSE analysis objects

* This research is sopported by the New Technology Program of the BID-CONICIT, the CONICIT MOODE Project and the ISYS Center OOMGRIN Project.
[1] Object Oriented Software Engineering

and GUI interface agents.Section 4 introduces a schema for structuring OOSE analysis objects. Finally, section 5 presents the OOMGRIN[2] method, goal of this work.

2. THE OOSE METHOD.

The OOSE method [Jac&Al 92] covers three phases called *analysis*, *construction* and *testing*. The outputs of the *analysis* step are the requirements model and the analysis model, which specifies and defines the system that is going to be built. During this phase the requirements are defined, and all efforts are concentrated into marking the limits of the system and in defining its functionalities. This phase is composed of three elements: the *use-case model*, the *object model* and the *specific descriptions of the system interfaces*. The use-cases model is the key element in OOSE's method, whereby the functionality of the system to be developed is defined (See Figure 1). The task scripts of interfaces is an essential part of the use-case descriptions, showing also how these are to be presented to the users. The object model is proposed by OOSE for developing a logical vision of the system using objects from the problem domain. The purpose of this model is to developed a robust and extensible structure that can act as the basis for system construction. In this work, we will be mostly interested in the analysis step. The *design* model is the refinement and formalization of the analysis model, and takes into account the implementation details. Its goal is to refine the analysis model and to adapt it to the implementation environment, so that its results can be easily passed to the code. The *implementation* model consists of the source code. The information space of this model corresponds to the computer language that is used.

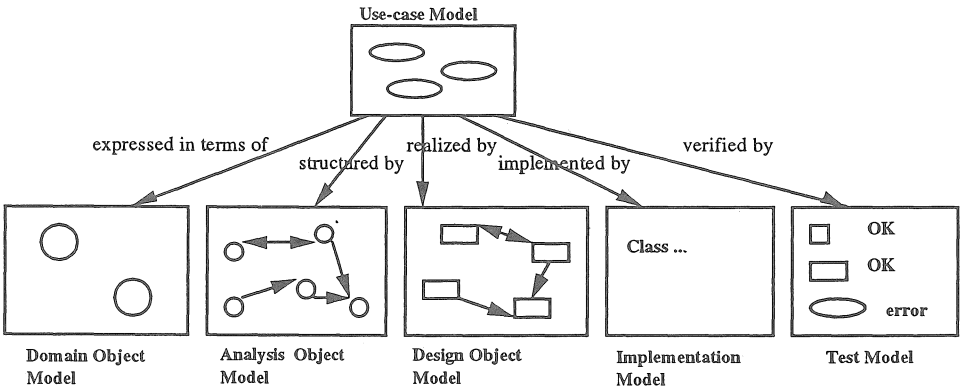


Figure 1. The use-case model encompassing the OOSE system life cycle

The use-case approach expresses the requirements in terms of actors, representing the people, systems and devices that will exchange information with the application. Use-cases represent events en terms of "what happens when" scenarios for each actor. Use-cases provide a common ground for developers and their

[2] Object Oriented Methodology for multiagents model based design of GRaphical INterfaces.

customers to achieve a shared vision of the application, early in the development process, when inconsistencies are easy and inexpensive to correct. The use-cases identified during requirements analysis continue through the whole OOSE life cycle, driving each subsequent phase (see Figure 1). In the Analysis Model, three types of objects for each use-case are identified: *interface*, *entity* and *control objects*. Interface objects (for example graphical user-interface widgets or complex windows), represent objetcs that depend on and change with the application's interface environment. Entity objects represent persistent data of the application, expressing its semantic and control objects contain rules not directly dependent on data or interface. For our approach, we will be interested on the control and interface objects.

3. ANALOGY BETWEEN OOSE ANALYSIS OBJECTS AND AGENTS FOR GRAPHICAL USER-INTERFACE DEVELOPMENT.

In order to construct an interactive system, a textual or graphical description of the user-interface must be provided. These specifications or task scripts express the user requirements for the user-interface of the application to be constructed. The architecture of an interactive system is expressed here by an agent based model [Cic 84], [Gol 84], [LVC 89] (an *agent* is seen as a processor, reacting at external events and generating events). In particular, the PAC (Presentation-Abstraction-Control) model [Cou 90], is constituted by agents, with three perspectives abstraction, control and presentation. PAC maintains the paradigm of the separation of the abstraction or semantic aspects of a problem from the presentation or visual aspects , differring from the well known MVC (Model-View-Controller) model [Gol 84], in the sense that the PAC presentation encapsulates the view-controller MVC pair for capturing and treating external events, and the control notion is introduced for communicating the abstraction with the presentation perspectives. Moreover, communication among the agents (and their perspectives) is only allowed through their respective controls and a hierarchy of agents is established, considering an agent constituted by subagents (as for example a window with menu and palette areas). According to the PAC model, the system architecture is constituted by a top level agent (representing the complete application) situated in the so called *Application level* and the hierachy of GUI agents, in the so called *Interface level.* We could think at first of a semantic analogy between all the objects defined in the OOSE analysis model and the different perspectives of the PAC agents. Nevertheless, since the PAC model is used for building GUI of interactive systems, it can clearly be established that a PAC agent corresponds to an OOSE interface object. Moreover, the three perspectives of a PAC agent correspond to a decomposition of an OOSE interface object: its image is the presentation, the image representation corresponds to the abstraction and the sequence of actions is the control. This correspondence can be established for every PAC agent, excepting the (root) agent on the highest level of the PAC hierarchy, representing the whole application that is being modelled. The abstraction, presentation and control of this agent have a different meaning. The abstraction of the root agent corresponds to all the objects of the application domain (OOSE entity objects). The control perspective of the root agent manages the set of all the system functionalities, particularly those of the user-interface (OOSE control objects). Finally, the presentation perspective of the root agent is in general an icon or window, representing the whole application, from where its functionalities may be accessed. These are represented by icons or menus windows, which are modelled by the agents of the lower levels, corresponding to the OOSE interface objects. The OOSE analysis model is used to derive the PAC model. This new model (to be discussed in Section 5) is called *Use-case-PAC*

Model and represents an architecture of the system's user-interface, where the use-case approach is included. Therefore, the model obtained reflects the involved actors interacting with the system through a GUI [LM 95].

4. A SCHEMA FOR STRUCTURING OOSE ANALYSIS OBJECTS.

A schema for organizing analysis objects in OOSE is presented [LM 95], showing the control and interface objects which have to be defined, considering the different use-cases related with a certain actor and establishing communications among these objects. The balanced control strategy, where the control object is an intermediary between the entity and interface objects, is taken into account in order to derive the schema, which is presented in Figure 2. The lines between the objects represent static links. For the definition of this schema, the following strategies are considered:

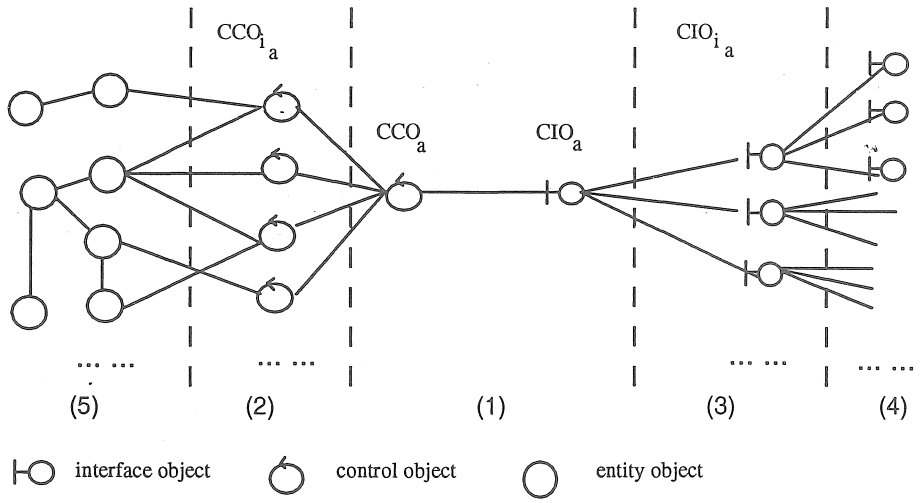
i. Define a central interface object, denoted by CIO_a, associated to the actor, denoted by *a*, corresponding to a window from which an actor can access the corresponding functionality (Level 1 in the schema). Define central interface objects CIO_{i_a}, 1_i_n, where *n* is the number of use-cases associated to *a*, each one corresponding to a window, from which one or more functionality or use-case can be executed (Level 3 in the schema). Then we have at most n CIO_{i_a} .

ii. Define a *central control object*, denoted by CCO_a(Level 1 in the schema). For each functionality or use-case, define a control object, denoted by CCO_{i_a}, 1_i_n, where *n* is the number of use-cases associated to a (Level 2 in the schema). The CCO_a expresses the control requirements of the actor. It manages the distribution of controls among the actor's functionalities, on the basis of the balanced control strategy, allowing also a clear separation between the abstraction and the user-interface aspects. The control object CCO_{i_a} associated to an actor facilitates the possible modifications relative to each functionaliy.

iii. Create independent or completely separated entity objects (Level 5 in the schema) from interface objects. This separation means that they do not "know" each other, in the sense that no links (static or dynamic) are established among them. Notice that the balanced control strategy (see 3.2) is also followed in this case.

iv. Locate toolkit widgets or widgets composition, to construct the windows corresponding to the CIO_{i_a}, 1_i_n (Level 4 in the schema). These objects are non central interface objects.

The above strategies clearly favor the separation between entity and interface objects, easing the system's decomposition into interface and problem domain components. Notice that the OOSE approach does not treat explicitly this separation, allowing links between entity and interface objects [Jac&Al 92]. As a consequence of the proposed schema, easy location of extensions or modifications, and the programmers' teamwork during the implementation phase are facilitated [Los&Al 94a], [Los&Al 94b].



(1) Central control and interface objects associated to the actor (CCO_a, CIO_a respectively)
(2) Control objects, one for each use-case associated to the actor (CCO_{i_a}, 1_i_n)
(3) Central interface objects, associated to the different use-cases of the actor. Each interface object in this level corresponds to a window, allowing the access to one or more functionalities (use-cases) associated to the actor (CIO_{i_a}, 1_i_n)
(4) Interface objects that are not central, such as widgets or objects composed from widgets
(5) Entity objects implied in the solution of the different functionalities (use-cases) associated to the actor

Figure 2. Schema of objects definition with respect to a given "human" actor.

5. AN OO METHOD FOR DEVELOPING INTERACTIVE SYSTEMS.

In [Los&Al 94a], [Los&Al 94b], [LG 95] and [Los 95] some guidelines have been proposed for developing interactive applications, whose architecture is based on the PAC model. They did not impose any particular design method; however, the use of the use-case approach is suggested. Then, in [LM 95] the schema presented in Section 4, has been developed to structure and communicate the objects of OOSE analysis model. From this schema the Use-case-PAC model is defined. The PAC agents will be implemented according to an agent framework defined in [LM 96]. Here, we will describe the OOMGRIN method, using the mentioned results from previous works.

The OOMGRIN Method.

I. *Formulation of the requirements for the application following the OOSE guidelines*. The Requirements model is developed, and concerns the following aspects: the use-case model, where the actors are identified with their functionalities (use-cases) with respect to the system; the description of the system´s interfaces; a description and /or a specification of the use-cases; and finally, an optional object model of the application.

II. *Creation of the Analysis Model, on the basis of the requirements formulated in step I*. This step develops a first logical system structure in which three dimensions are focused: the system behavior, the information contained in the system and the presentation of the system to the exterior world. Three types of objects are identified during this step: *control*, *entity* and *interface objects*. The schema (Figure 2) for organizing these objects developed in Section 4 is used for every "human" actor of the system.

III. *Obtention of the Use-case-PAC model*. The PAC model allows the definition of a system architecture where user-interface aspects are focused. Based on the schema defined in step II, the Use-case-PAC model is developed showing the different levels corresponding to the system´s actors, functionalities requiring user-interface facilities.

IV. *Design of the Use-case-PAC agents using frameworks.*. Two frameworks or semifinished generic architecture are used for developing each interface agent of the Use-case-PAC model. The approach to software design with patterns or templates that can be applied in many situations, captures the experience involved in designing object oriented (OO) software. A framework describing the implementation of a PAC agent has been presented in [LM 96].

V. *Implementation of the classes corresponding to the agents.* Reuse of the available toolkit classes and communication schemas preserving the PAC principles [Los&Al 94a], [Los&Al 94b] are used to implement the graphical (low level) objects.

In what follows we will be discussing in details each one of these steps.

5.1 Formulation of the requirements for the application following the OOSE guidelines.

A requirements model [Jac&Al 92] is created from the requirement specification in which we delimit the system and define all the functionality of the system. Two main elements: *actors* and *use-cases* help in developing this first transformation from the requirement specification. These concepts are simply an aid to defining what exists outside the system (actors) and what should be performed by the system (use cases). The actors represent what interacts with the system and also show everything that needs to exchange information with the system. A use case is a description in which identifies its transactions what are initiated by an actor. After we have defined what is outside our system we can define the functionality inside it. We do this by specifying use cases. A use case is a specific way of using the system by performing some part of the functionality.

The use case model [Jac&Al 92] uses actors and use cases for developing its complete structure. This use case model is part of the requirements model and will show all of the functionality of the system. The use case model will control the formation of all other models (Figure 1) It is developed in cooperation with the domain objet model. It is necesary to describe the interfaces in more detail in order to support the above use case and also the communication between them. Depending on the complexity of the user interface it could be interesting to be careful in describing interface descriptions.

5.2 Creation of the Analysis Model, on the basis of the requirements formulated in step I.

The Analysis Model [Jac&Al 92] aims to structure the system independently of the actual implementation environment. This means that we focus on the logical structure of the system. It is here that we define the stable, robust and maintainable structure that is also extensible. According to the OOSE approach, entity and interface objects should have been identified before the control objects. Control objects are non-persistent, in the sense that they only exist during the use-case execution. Control objects allow also communication among the other objects, connecting different scenarios. A control object can be assigned to each concrete or abstract use-case, where the interface and entity objects express the behaviour that has not been captured in the other objects. When a control object is related to different actors, its behaviour depends on the actor and it should be decomposed into different control objects, one for each actor. This descomposition favors extension and maintenance because modifications are induced by the actors. In general, the kind of funcionalities expressed by a control object are transaction oriented or particular control sequences of one or several use-cases. All the funcionalities specified in the use-case descriptions depending directly from the environment exterior to the system, are expressed by the interface objects. The actors communicate with the system through these objects. The role of an interface object is to translate the actor actions on the system into system events involving the actor and its queries. Interface objects describe bidirectional communications between the system and its users. The functionalities must be distributed into entity, interface and control objects. The schema defined in Figure 2 is used to define and distribute these objects.

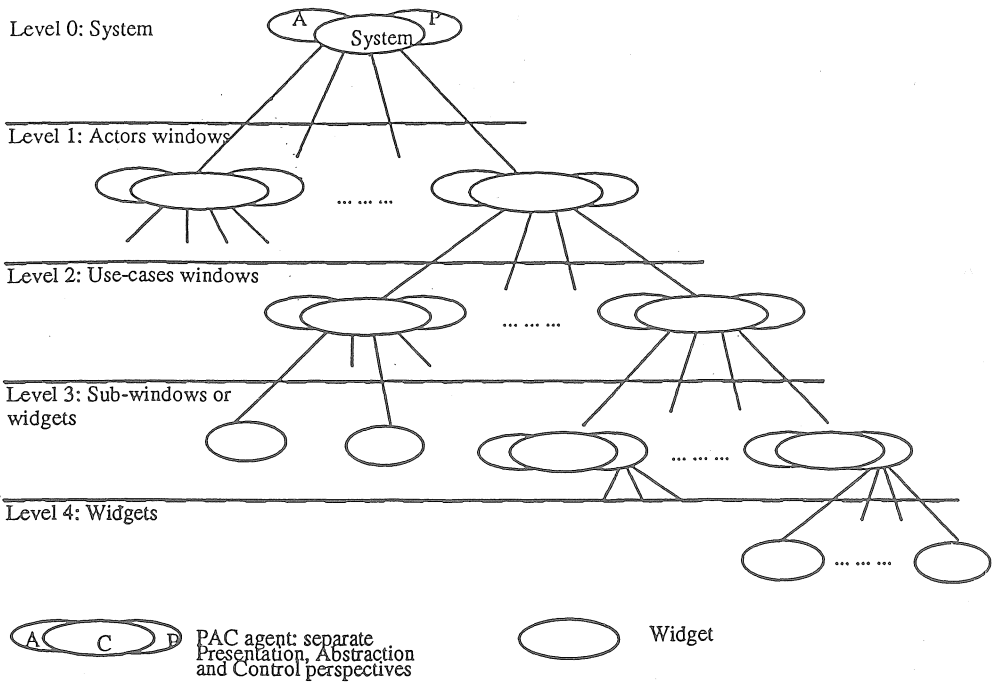
5.3 Obtention of the Use-case-PAC model.

The Use-case-PAC model is obtained through a schema defining the objects in the OOSE analysis model, with respect to a given "human" actor.The PAC model allows the definition of a system architecture where user-interface aspects are focused. In what follows, the PAC model derived from the use-case approach presented in the schema seen in 4, is considered. In the schema, the objects are defined with respect to a given actor of the system. The PAC diagram in Figure 3 shows levels corresponding to the system's actors, having functionalities requiring user-interface facilities.

In the diagram, Levels from 1 to 4 correspond to the different interface objects used for interface design, based on the use-case approach, according to the OOSE method. The schema defined in 4 allows the separation of the application or domain problem component from the interface component, since the only communication is through control objects. At Level 0, the control perspective of the agent called *System*, corresponds to the set of control objects of the OOSE analysis model. The strategy for defining their functionalities is to use balanced control objects. These will communicate with the PAC agents of Level 1 only through their control perspectives. The abstraction perspective of the system agent (*A* in the diagram), corresponds to the problem domain, that is to say to the set of entity objects of the OOSE analysis model. As we have already mentioned, the presentation perspective of this agent (*P* in the diagram) correspond to just an icon to start the application, or a window representing the whole application, from where its functionalities may be accessed.

Notice that in the Use-case-PAC model, the agents are equivalent to the interface objects identified in the OOSE analysis model; this correspondence is possible according to the schema presented in 4. The advantages of the Use-case-PAC model are:

- Structure hierachically the interface objects.
- Decompose each interface object into three perspective (abstraction, presentation, control), with all the advanteges of the PAC agents.
- Provide an object-oriented design for the user-interface component.
- Have a clear separation between the problem domain and the user-interface component in the system architecture.
- Facilitate modifications and the implementation step [Los&Al 94a], [Los&Al 94b].



Level 0: the agent corresponds to the application that is being developed
Level 1: each agent corresponds to a CIO_a, called actor window, from which the actor's use-cases are accessed
Level 2: each agent corresponds to a CIO_{i_a}, 1_i_n, from which the actor can execute functionalities associated to one or more use cases. For each agent (actor window) of Level 1, we have at most n windows.
Level 3: each agent corresponds to non central interface objects, which are windows contained in the actor windows
Level 4: each object corresponds to a widget.

Figure 3. The Use-case-PAC Model

5.4 Design of the Use-case-PAC agents using frameworks..

According to [Pre 95] a framework is defined as a generic semifinished architecture for various application domains. One of the first application of frameworks was in the domain of GUI. Actually, MVC is one of the first known frameworks [KP 88]. A framework is constituted by one or several design patterns [Gam&Al 95], that are used for rapid development of adaptable (reusable and extensible) GUI. A framework represents a generic subsystem that can be instantiated through public abstract classes of the framework (hot spots).

The approach to software design with patterns or templates that can be applied in many different situations, capture the experience involved in designing object-oriented (OO) software. Each design pattern systematically names, explains and evaluates an important and recurring design OO systems. Recurring patterns of classes and communicating objects are found in many object-oriented system. These patterns solve specific design problems and make OO designs more flexible, elegant and reusable. They help designer reuse successful designs by pasing new designs on prior experience. A designer, familiar

with such patterns can apply them inmediatly to design problems without having to rediscover them. 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 common design structure that make it useful for creating a reusable OO design.

Considering these ideas, a framework for a PAC agent has been defined in [LM 96]. Three main patterns characterize the PAC model: *Mediator*, *Strategy* and *Factory Method*. The control, which introduces the notion of communication between abstraction and presentation, mediating for the coherence between the two perspectives, is modelled by the *Mediator* pattern. It is important to point out that, since a PAC agent may be decomposed in subagents, due to the PAC agents hierarchy, each subagent has its own control, and the nested views are treated as subagents. The *Strategy* pattern attaches a view to a controller allowing to change the way a view responds to user inputs and finally the *Factory Method* relates a view with the different incapsulated treatements of the user inputs, also like MVC.

5.5 Implementation of the classes corresponding to the agents.

Depending on the multiagent model, we could select an appropiate toolkit available in the market. When the toolkit has tools at a high abstraction level, the tools have more control on the agent than the user or implementor; this mean that the presentation and the control are kept together and the two perspectives cannot be separated as for PAC agents [LG 95]. This is the case of MVC like agents, where the controller and the view are together in the presentation perspective, and the abstraction is separated from the rest of the agent and the case of agents with non separated perspectives. This combination of models seems well adapted to toolkits with tool at lower abstraction level, like X,Xt[3]/Motif[4]. The implementor has more control on the agents and three separate perspectives can be always considered. An important issue is that at application level, an abstraction perspective is maintained in both models, and at interface level, the lower level the specific application component objects have also a separate abstraction. This fact guarantees that the PAC principles are maintained in the sense that this architecture favors the distribution of semantical and syntactical actions, enhancing the system flexibility to changes. The communications among the PAC agents follows a schema defined in [Los&Al 94a], [Los&Al 94b] which preserves the PAC principles, to implement directly the communications in C++ [Str 93]. The three PAC perspectives are builted as separate classes.

6. CONCLUSIONS.

This work describes an OO method for developing interactive systems. Our method enhances the early separation of GUI aspects from the semantic application features, favoring also a scenarios-oriented design. Notice that steps I, II, III correspond to the analysis phase, that it is performed using the Use-case-PAC model. Steps IV and V correspond to the detailed design phase. Particular care has to be taken for the development of the abstraction corresponding to the application level agent, which represents the

[3] X-Window is registered trademark of the Massachusetts Institute of Technology.
[4] Motif is a registered trademark of Open Software Foundation.

problem domain component and can be modelled accordingly, using any of the existing OO methods, or a combination of them, within the spirit of recent trends in software development methodology. We claim [LG 95] that the structure of the PAC agents may be greatly influenced, according to the choices of the plataform and the graphical toolkits. In the sense of that, according to the toolkit tool, the control perspective of an agent cannot be always separated from the presentation and abstraction perspective. Nevertheless, the PAC communication principles can still be preserved. The guidelines for interactive software development presented in this paper have been used successfully in several software projects at the ISYS[5] Research Center. The benefits from this experience with respect to groupware are multiple, in the sense that the work can be easily distributed among the programmers' teams, assigning to different teams the separate development of the presentation and the abstraction perspectives of the agents constiting the GUI of the application. Finally, another different team can be in charge to implement the predefined communication schema [Los&Al 94a], [Los&Al 94b] between agents and their perspectives. Moreover, the problem domain component can be developed in parallel with the GUI component, favoring an early prototyping of the application. The formalization of the concepts underlying the OOMGRIN method and the experimentation with the system's adaptability to changes in the GUI, as for example the incorporation of different interaction media such as voice and videos are undergoing research topics.

7. REFERENCES.

[Car 95] CARROL J., M., "Scenario-Based Design", Jhon Wiley &Sons Inc., 1995.

[Cic 84] CICCARELLI E. C., "Presentation Based User-Interfaces", Technical Report 794, Artificial Intelligence Laboratory, Massachussets Intelligence Laboratory, August 1984.

[Col 95] COLLINS D., "Designing Object-Oriented User Interfaces", Benjamin/Cummings Publishing Company, Inc., 1995.

[Cou 90] COUTAZ J., "Interfaces homme-ordinateur", Dunod 1990.

[Del 92] DELMAS S. "Tcl/TK User's Manual", University of California, Berkeley, 1992.

[Jac&Al 92] JACOBSON I., CHRISTERSON M, JONSSON P., ÖVERGAARD G., "Object-Oriented Software Engineering, a Use Case Driven Approach", Addison Wesley, 1992.

[F&D 84] FOLEY D.J., VAN DAM, "Fundamentals of Interactive Computer Graphics", Addison Wesley, 1984.

[Gar 93] GARFINKEL S.L., MICHEL K.M. "The NeXSTEP Programming: Step One, Object-Oriented Applications" A Tutorial for developers using the Objective-C Language and the NeXT Interface Builder. Santa Clara, CA:Telos/Springer-Verlag, 1993.

[Gol 84] GOLDBERG A., "Smalltalk-80 - The Interactive Programming Environment", Addison-Wesley 1984.

[HH 89] HARTSON R., HIX, D., "Human-Computer Interface Development Concepts and Systems for its Management", ACM Computing Surveys, Vol.21, No.1, March 1989.

[Lan 86] LANTZ K. A., "On User-Interface Reference Models", ACM SIGCHI, Bulletin, vol. 18, 2 (1986), 36-44.

[LG 95] LOSAVIO F., GALVEZ C.,"The platform influence on object-oriented development based on the multiagent model", Proceedings of the XXI Conferencia Latinoamericana de Informática, Panel'95, Canela RS, Brasil, August 1995.

[5] Centro de Ingeniería de Software y Sistemas.

[Los 95] LOSAVIO F. "An Object-Oriented Methodology for User-Interface Design Based on the Multiagent Model", Proceedings of the Information Systems Analysis Synthesis, ISAS'95, 89-92, Baden-Baden, Germany, August 1995.

[Los&Al 94a] LOSAVIO F., MATTEO A., ORDAZ O., MEZA O., GONTIER W. "An implementation of the PAC architecture using object-oriented techniques" Proceedings IFIP'94, 13th World Computer Congress 94, Volume 2, 149-155. Hamburgo-Alemania, Agosto 1994, K. Brunnstein and E. Raubold (Editors) Elsevier Science B.V. (North-Holland).

[Los&Al 94b] LOSAVIO F., MATTEO A., ORDAZ O., MEZA O., GONTIER W. "Object-oriented approach and PAC model: design of the GReAt (Graph Researcher Assistant) environment" Proceedings XX Conferencia Latinoamericana de Informática, Panel'94, 433-443, Monterrey, Estado de México, Septiembre 1994.

[LM 95] LOSAVIO F, MATTEO A., " Use-case and Multiagent Models for Object-Oriented Design of User-Interfaces", L.R.I., Orsay, France, Rapport de Recherche No. 992, Sept. 95. To appear in Journal of Object- Oriented Programming.

[LM 96] LOSAVIO F, MATTEO A., "Object-Oriented User-Interface Design Based On Agents Frameworks", Centro ISYS, Reporte de Investigación, ISYS No. 1, RI/01/96, Enero 1996.

[LVC 89] LINTON M.A., VLISSIDES J.M., CLADER V., "Composing User-interfaces with Interviews", IEEE, Computer, 8-22, Feb. 1989.

[Mey 88] MEYER B., "Object-Oriented Software Construction", Prentice Hall 1988.

[Ste 87] STERN H. L. , "Comparison of Window Systems" BYTE 12, 11, Nov. 1987.

[Str 93] STROUSTRUP B., "The C++ Programming Language", Second Edition, Addison-Wesley Publishing Company, 1993.

[KP 88] 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.

[Pre 95] PREE W. "Design Patterns for Object-Oriented Software Development", Addison Wesley, 1994.

[Gam &Al 95] GAMMA E., HELM R., JOHNSON R.,VLISSIDES J. "Design Patterns. Elements of Reusable Object-Oriented Software" Addison Wesley Publishing Co., 1994.