A GENERIC ARCHITECTURE FOR USE-CASE BASED OBJECT-ORIENTED DESIGN [1]

F. Losavio, A. Matteo
Centro de Ingeniería de Software Y Sistemas ISYS,
Facultad de Ciencias, Universidad Central de Venezuela,
Apdo. 47567, Caracas, Venezuela
flosavio@conicit.ve / @anubis.ciens.ucv.ve
amatteo@conicit.ve / @anubis.ciens.ucv.ve

Abstract

A generic framework is defined for describing a reusable architecture of a system, according to a use-case based design and it can be used for any system developed according to Jacobson's OOSE method. The layered architecture obtained is constituted by three main levels: problem domain, control and interface level, where the interface and control levels are structured as a function of the system's actors. Particularly, for the interface level, interface agents are also modelled with frameworks. Notice that, with respect to the interface level, the general framework provides a greater level of genericity, in the sense that it can be adapted to the framework implementing the interface agents, according to different multiagent models, such as MVC or PAC. Moreover, different frameworks are used for implementing each level and then plugged into the generic framework.

Keywords: use-case, object-oriented design, user-interface design, framework, design patterns, multiagent model, interface agent, PAC model

1. INTRODUCTION

The goal of this work is to derive a generic framework for object-oriented (O-O) design from the system's actors functionalities, captured by the use-case concept, according to a scenario-based [Car 95] point of view. A framework is defined as a reusable semifinished architecture for various application domains [Pre 94]. It is constituted by a set of classes, that may constitute several design patterns, conceived for working together, representing 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 94], 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 graphical user-interfaces (GUI) is considered one of the first known frameworks [KP 88]. Use-cases are identified in the OOSE (Object Oriented Software Engineering) requirements model [Jac&Al 92] and the framework is established on the basis of a general schema [LM 95] for organizing entity, control and interface objects, considering a given "human" actor. This framework integrates other frameworks, considering three layers or levels, for achieving a complete system design: *problem domain level*, *control level* and *graphical user-interface (GUI) level*. The problem domain level can be modelled using any existing O-O methods or a combination of them, appearing now as an incouraging trend for developing and using object-oriented methods. The interface between this level and the control level is established through the Facade design pattern [Gam&Al 95]. The framework for the control level is built under the basis of generating control classes for each actor and their respective GUI interface classes. The GUI level is derived from the interface classes applying also a generic framework for GUI design based on multiagent models [LM 96].

This paper is structured into three main sections, besides this introduction and the conclusion. Section 2 is dedicated to a brief presentation of the OOSE method, focusing on the heuristics and strategies followed to

[1]This research is supported by the New Technology Program of the BID-CONICIT, the CONICIT MOODe Project and the ISYS center OOMGRIN Project.

characterize interface and control objects in the analysis model. Section 3 presents the schema obtained from the interface objects of the OOSE analysis model. Section 4 is focused on the generic framework for system's design, goal of this work.

2. THE OOSE (OBJECT ORIENTED SOFTWARE ENGINEERING) METHOD

The OOSE method [Jac&Al 92] encompasses the application development in terms of three main processes: Analysis, Construction and Testing, each maintaining their own models. Figure 1 below shows the OOSE method through the use-case modelling approach. The analysis step, concerns the obtention of the Requirements and Analysis Model. The use-case driven approach is important during this step, considering the requirements expressed in terms of actors, representing the people, systems and devices that will exchange information with the application. Use-cases represent events in terms of "what happens when" scenarios for each actor. They provide a common ground for developers and their customers to acheive a shared vision of the application, early in the development process, when inconsistencies are easy and inexpensive to correct. The terms use-case and functionality will be used indistinctly through this work. The use-cases identified during requirements analysis continue through the whole OOSE life cycle, driving each subsequent phase. In the Analysis Model, three types of objects for each use-case are identified: *interface*, *entity* and *control*. Interface objects (for example graphical user-interface widgets or complex windows), represent objects that depend on and change with the application's interface environment. Entity objects represent persistent data of the application, expressing its semantics and control objects contain rules not directly dependent on data or interface. We will be interested here in control and interface objects.

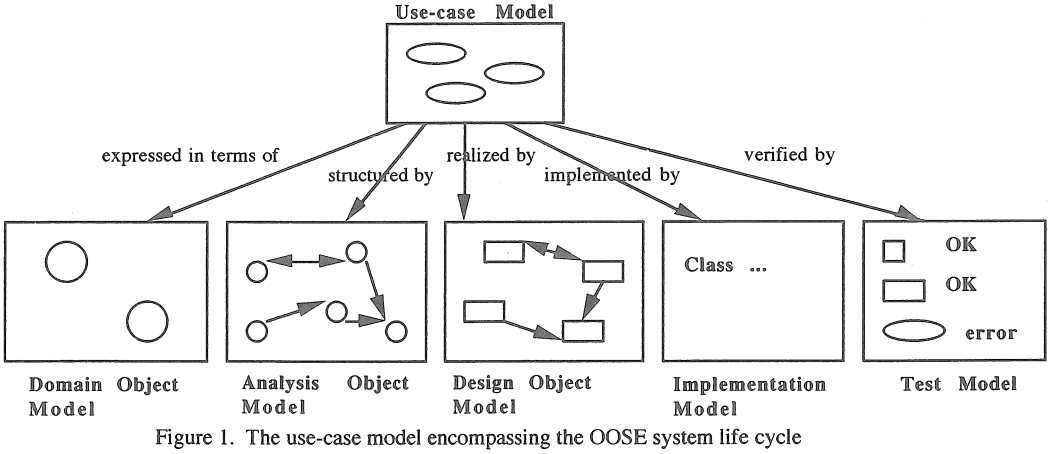


Figure 1. The use-case model encompassing the OOSE system life cycle

2.1 Control objects of the Analysis Model.

According to the OOSE approach, entity and interface objects should have been identified before the control objects. These are then used as a "cement" to relate or link the other objects, in order to constitute a complete use-case. Control objects are non-persistent, in the sense that they only exist during the use-case execution. In a use-case, the sequence of actions may follow different alternatives, constituting a *scenario of events*. Control objects allow also communication among the other objects, connecting different scenarios.

Some heuristics for identifying control objects are [Jac&Al 92]:

- They are determined from the use-cases. At first, a control object can be assigned to each concrete or abstract use-case, where the interface and entity objects are implicit and have already been identified. Then,

control objects express the behaviour that has not been captured in the other objects. There is no need to model control objects, when there are no more features to be captured through them. If, on the contrary, a complex behaviour has to be modelled, the functionalities can be subdivided into several simple control objects, with specific and limited tasks, easy to describe.
- 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 decomposition favors extension and maintenance because modifications are induced by the actors. Hence, system modifications may be easily isolated if a control aspect is associated to a unique actor. In general, the kind of functionalities expressed by a control object are transaction oriented or particular control sequences of one or several use-cases.

2.2 Interface objects of the Analysis Model.

All the functionalities 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.
Interface objects are identified according to [Jac&al 94]:

- Through the interface descriptions of the Requirements Model.
- Through the actor role.
- Through the use-case description, extracting those functionalities specific to the interface

Changes on the system interface are in general frequent and expansive and it is important to have criteria to manage them accordingly. Interface objects facilitate extension of the system interface, since the impact of modifications are concentrated only on these objects.

According to [Jac&Al 92], two kinds of interfaces may be modelled for a problem solution: interfaces with other systems and human user interfaces. For the first kind of interfaces, the communication is described in terms of *protocols*. For the second kind, GUI are used for communication between the user and the system. Several systems such as X-Window[2], News [Ste 87], NexTStep [Gar 93] and Windows[3], among many others, and class libraries written in different languages such as C++ [Str 93] and Smalltalk [Gol 84], are widley used tools, constituting also standards for developing systems with large GUI components. Different levels of granularity characterize the modelling of the interface objects. In general, a window is considered an interface object. In order to model each window structure, the gadgets or widgets of the tooolkit corresponding to the particular windows manager system are also considered interface objects, constituting the fine grain levels. A *central interface object* corresponds to a whole window and the other interface objects model the multiwindows toolkit widgets. Interface objects focus on the presentation, but can also process information and hold a certain behaviour. The amount of information and behaviour for each interface object is decided accordingly to each particular case. The same occurs with the other objects. All the use-case functionalities must be distributed into entity, interface and control objects. There are several strategies for assigning these functionalities [HH 89]:

i. *Integrated control*: when the control functionalities are placed into control and entity objects. In this case, the interface objects are not relevant, because they have no functionalities. This structure may be efficient

---

[2] X-Window is a registered trademark of the Massachusetts Institute of Technology.
[3] Windows are registered trademarks of Microsoft

with respect to execution time, but is not convenient for prototyping because the interface objects have very few functionalities.

ii. *Dialog oriented control*: when most of the dialog control functionalities are placed into interface objects and these model most of the system functionalities. In this case the control objects are not relevant, because they have few functionalities. This structure facilitates prototyping, but increases the interface complexity, due to the fact that events are detected and processed in the same object.

iii. *Mixed control*: when control functionalities are divided into entities and interface objects, allowing, for example, to invoke the dialog from an entity object, where some computations are performed, and reciprocally. In this case, communication facilities are not concentrated on particular objects, not easing modifications.

iv. *Balanced control*: when the control is completely separated form the computing aspects of the entity objects and from the dialog aspects of the interface objects. Then in this case, control objects manage communication between entity and interface objects.

3. A SCHEMA FOR DEFINING ANALYSIS OBJECTS IN OOSE

This schema [LM 95] defines the objects in the OOSE analysis model, with respect to a given "human" actor and the use-cases corresponding the GUI functionalities. It shows 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 described in 2.2, 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:

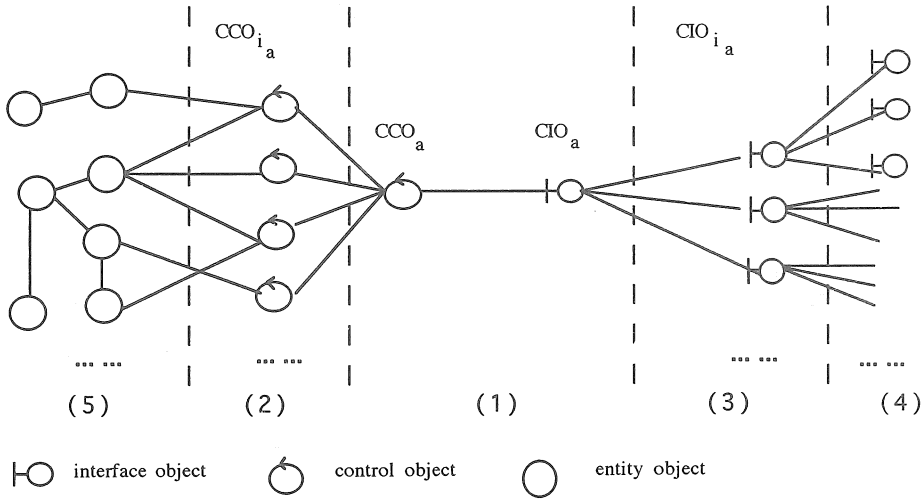
i. Define a central interface object, denoted by CIO_a, associated to the actor, denoted by *a*, corresponding to a window from which an actor can access the corresponding functionality (Level 1 in the schema). Define central interface objects CIO_{i_a}, $1 \le i \le n$, where *n* is the number of use-cases 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}, $1 \le i \le n$, 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 possble 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}, $1 \le i \le n$ (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}, $1 \le i \le n$)

(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}, $1 \le i \le n$)

(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

4. A FRAMEWORK FOR USE-CASE BASED DESIGN

A generic framework for object-oriented system design is presented in this section (Figure 7). From the above schema (Figure 2), the traceability of the control and interface objects with respect to a given actor can be followed through the actor's control objects framework (Figure 3) and the actor's window with related use-case windows framework (Figure 6). The entity objects are shown as part of the outside world in Figure 3, as the problem domain component. Figure 4 and 5 show two frameworks for GUI interface agents, corresponding to different multiagent models. One of these framework can be used for designing the GUI component, particularly to construct the framework shown in Figure 6, which can be "plugged" into the control objects framework (figure 3) for obtaining a generic framework for the general architecture of the whole system, shown in Figure 7. In what follows, these frameworks will be discussed in details, to illustrate the complete system design obtained with this approach.

4.1 Framework for control objects associated to an actor

From the above schema, the Central Control Object CCO_a, has been defined, for distributing the user requirements with respect to the different use-cases of a given actor. This object communicates with the GUI level, through the CIO_a , corresponding to an actor window, from where the actor's different GUI functionalities are accessed, inplemented by the Agent framework of Actor *a*. The ControlCalss Actor $_a$ will be activated or created by the SystemTop-Level Control class (see Figure 3), representing the central control of the whole system.

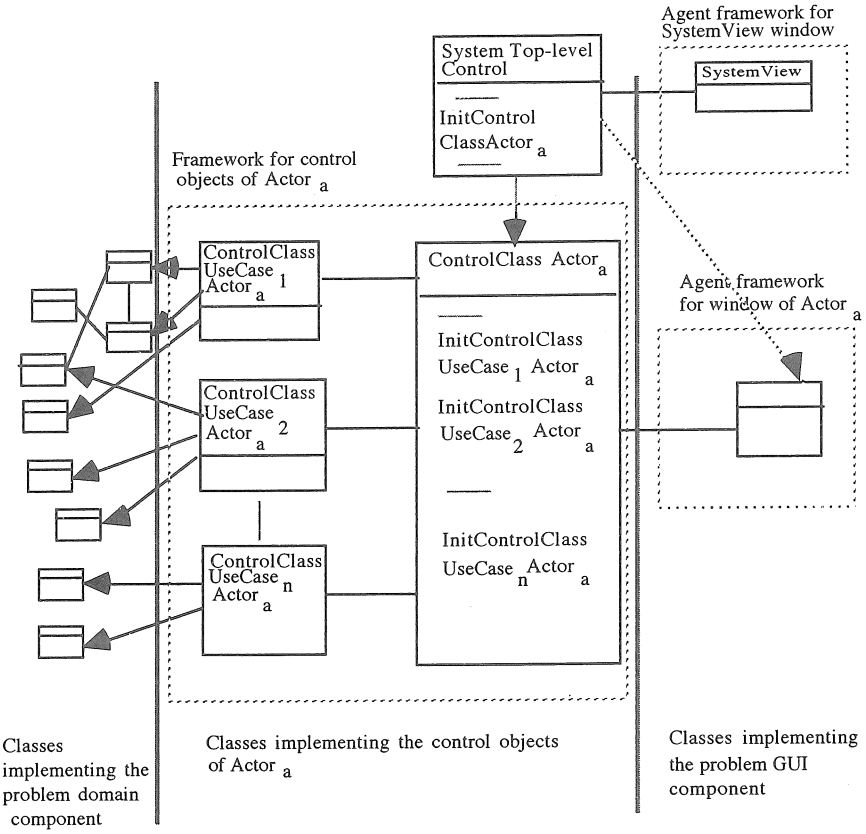


Figure 3. Framework for the control objects of an actor

Moreover, control objects CCO_{i_a}, for each use-case of the actor, interact with CCO_a, who discriminates, at a certain moment, the control objects responsible of the particular functionality required by the actor. Besides, each one of these control objects connects the entity objects required for the solution of the respective functionality. The framework corresponding to the actor's control objects is defined, considering exactly one class for each control object; the relations among these classes are justified by the above discussion. Moreover, the Control Objects framework, denoted by the dashed rectangle in Figure 3, shows also the connections with the exterior world, represented by the classes implementing the entity objects, belonging to the problem domain component, and by the classes implementing the GUI agents of the problem interface component. The ControlClassUseCase$_i$, implement the CCO_{i_a}, $1 \le i \le n$, and corresponds to the FACADE design pattern [Gam&Al 95]. This pattern defines a higher interface level that makes the

subsystem easier to use, structuring a system into subsytems reduces complexity. A common design goal is to minimize the communications and dependences between subsytems. The FACADE object helps to acheive this goal providing a single simplified interface to the more general facilities of a subsystem. The classes implementing the entity objects and corresponding to some functionality, may be seen as subsytems. The FACADE pattern is used when there are many dependences between clients and the implementation classes of an abstraction. The subsystem is decoupled from clients and other subsystems, promoting subsystem independence and portability. Another important usage of FACADE is to define an entry point to each subsystem level. If subsystems are dependent, then the dependencies can be simplified by making them communicate with each other only through their facades. FACADE knows which subsystem classes are responsible for a request and delegates client requests to appropriate subsystem objects. Subsystem classes implement subsystem functionality, handle work assigned by the FACADE object and have no knowledge of the FACADE, keeping no references to it. So, a convenient interface is defined, for solving the functionality related to use-case *i* of actor *a*. In other words, $ControlClassUseCase_i$ $1 \le i \le n$, sends messages to the classes corresponding to the entity objects required for the use-case solution. This class is also connected and interacts with the ControlClass $Actor_a$, which is responsible of interacting with the GUI classes related to the actor, discriminating which on of the ControlClass $UseCase_i$ must appear at a given moment, to satisfy user requirements.

4.2 Framework for interface objects associated to an actor

Interface objects can be structured and defined according to each "human" actor, as we have seen in Section 3. These interface objects correspond to the GUI component of the system. In order to define this architecture, the well known MVC or PAC multiagent models can be used, favoring separation between semantic and presentation aspects of the system, enhancing its adaptability. As a consequence, we can establish a correspondence between the main interface objects identified in the analysis model and the interface agents of the GUI models. So, a framework is used [LM 96] to implement each agent (see Figure 4 and 5), according to the particular model used.

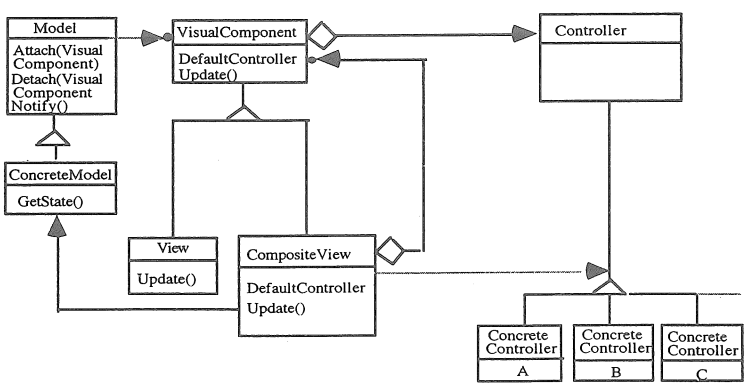


Figure 4. MVC agent framework

In particular, we are interested on the actor window agents and on the use-case windows of this actor, according to the schema defined in Section 3. These are the interface objects in the first level of the interface component relative to an actor. Hence, these interface objects are included in the framework shown in Figure 6.

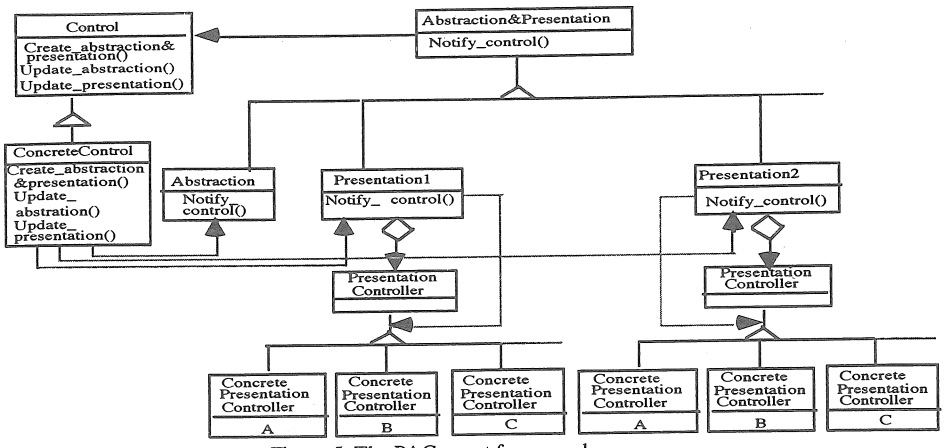


Figure 5. The PAC agent framework

So, this framework is pugged into the Agent framework for window of $Actor_a$, shown in Figure 3. Assuming one of these solutions, ControlClass $Actor_a$ interacts with the GUI agent relative to actor *a*, through the particular ConcreteModel $Actor_a$ window class, if the MVC model is selected, or with ConcreteControl $Actor_a$ window class, if the PAC model is selected.

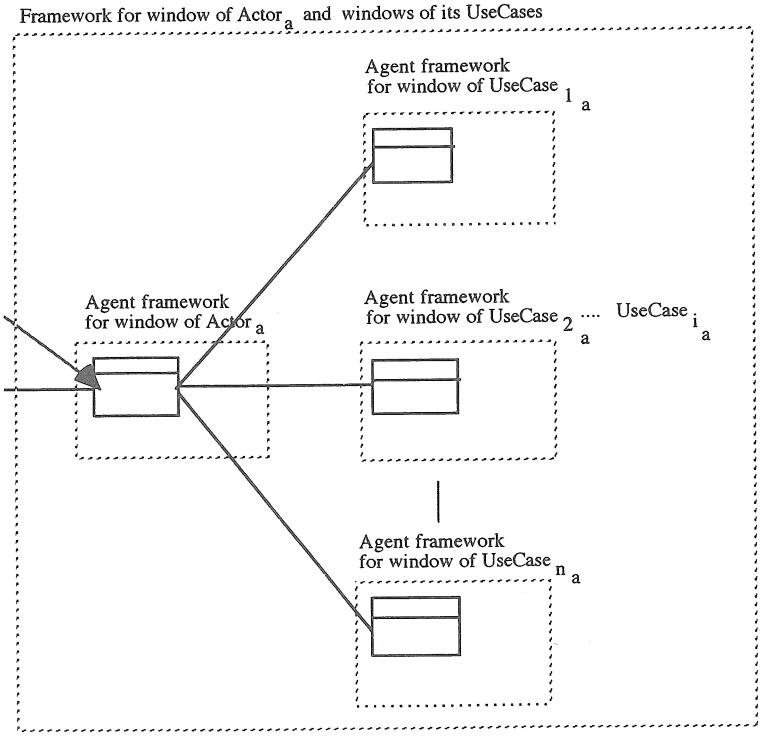


Figure 6. Framework for an actor's window and its use-case windows

4.3 Generic framework for an object-oriented system architecture based on use-cases. In this section a generic framework is defined for expressing the system architecture. We have seen that the frameworks defined in Sections 4.1 (Figure 3) and 4.2 (Figure 6) can "plugged" in, providing the design of the control and interface objects associated with a system's actor. On these basis, considering the human actors and their respective use-cases expressing the GUI functionalities, we can define the genric framework shown in Figure 7. In this framework the SystemTop-LevelControl class is responsible for the activation of the subsytem corresponding to an actor, which requires to interacts with the system, at a given moment. The requirement is captured through the interface, offering the whole system presentation, corresponding to the framewark called Agent framework for SystemView Window.

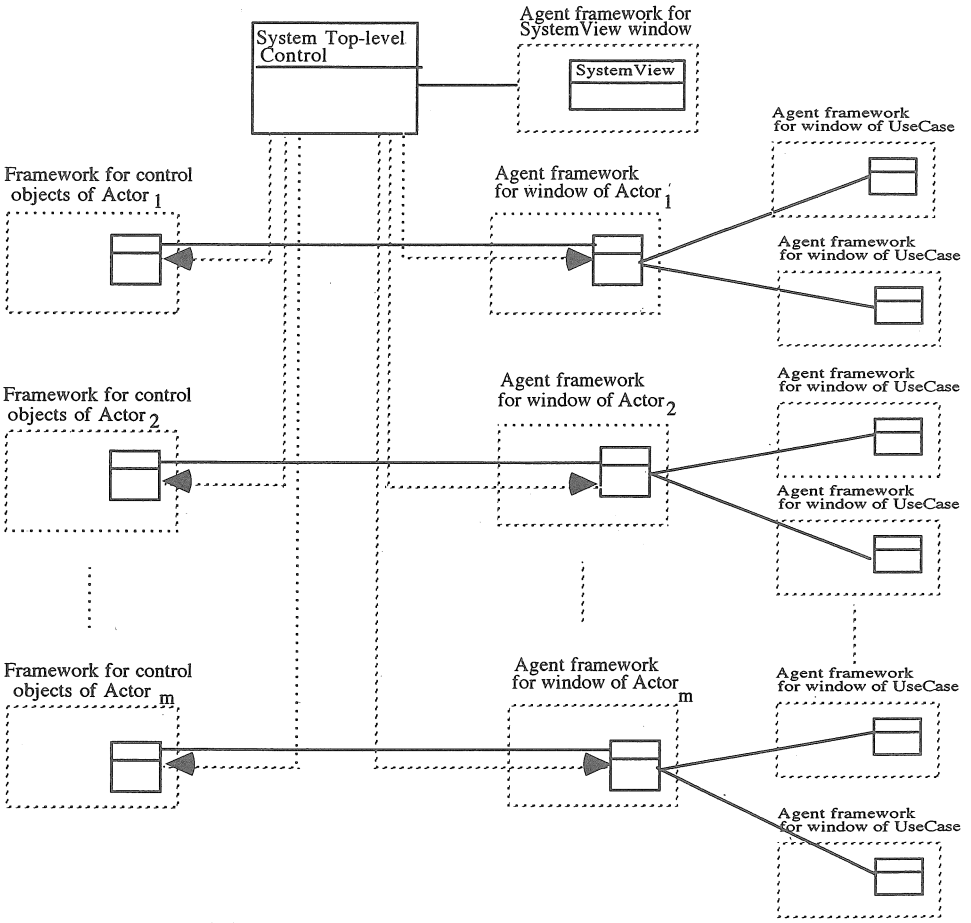


Figure 7. Generic framework for an object-oriented system architecture, according to a use-case based design

5. CONCLUSION

This work has presented a generic framework for a system architecture, according to a use-case based design. This framework can be reused for any system in the design step, for establishing the respective model, according to Jacobson's OOSE method. It also establishes a clear traceability among the control and interface objects identified in the analysis model and the respective classes implementing them. Moreover, the system architecture is completely determined as a function of the identified human actors and their corresponding use-cases. On the other hand, the architecutre obtained through this framework allows a clear differentiation of three layers, organizing the classes implementing the OOSE objects, corresponding respectively, to the problem domain, the control and interface components. This organization enhances a total separation between the problem domain and the interface component, which communicate indirectly trhough the control component. From these considerations, it is evident that a structure easing reuse, maintenace and portability of the system is obtained. Teamwork is also facilitated during the development phase. The system may be seen as a set of sybsystems, one for each actor. The interface agent notion is present in multiagent models such as MVC and PAC and is designed by the respective framework implementing the agent. According to the selected multiagent model, the architecture of the generic framework is adapted to the framework for the particular model. In consequence, we can conclude that the proposed framework provides a grater level of genericity, in the sense that it is adaptable to the specific framework implementing the interface agent.

6. 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.

[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.

[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.

[LM 95] LOSAVIO F., MATTEO A. "Use-Case and Multiagent Models for Object-Oriented Design of User-Interfaces" LRI, Orsay, France, Research Report No. 992, September 1995. To appear in Journal of Object Oriented Programming

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

[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

[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.

Flavio Araujo de Mattos *é Mestre pelo Programa de Engenharia de Sistemas e Computação da COPPE/UFRJ (Universidade Federal do Rio de Janeiro) e Bacharel em Matemática (modalidade informática) pela UFRJ. Trabalha na área de informática desde 1984, tendo ministrado cursos e palestras para empresas. Suas áreas de interesse são: Engenharia de Software, Banco de Dados, Reutilização de Software e Orientação a Objetos. Atualmente, é analista de sistemas do BNDES.*

Cláudia Maria Lima Werner *é Professora Adjunto do Programa de Engenharia de Sistemas e Computação da COPPE/UFRJ (Universidade Federal do Rio de Janeiro). Doutora em Eng. de Sistemas e Computação pela COPPE/UFRJ e Bacharel em Matemática (modalidade informática) pela UFRJ. Trabalha na área de informática desde 1983, tendo ministrado diversos cursos de pós-graduação na COPPE e cursos/palestras para empresas. Suas áreas de interesse são: Engenharia de Software, Reutilização de Software, Orientação a Objetos e Ambientes de Desenvolvimento de Software. Já publicou diversos artigos técnicos em congressos nacionais e internacionais.*