ESPECIFICACION DE LA INTERFAZ USUARIO EN UN PROCESO QUE COMBINA METODOS ORIENTADOS A OBJETOS[1]

F. Losavio, A. Matteo, Ch. Metzner, N. Niño, N. Urbina
Centro de Ingeniería de Software Y Sistemas ISYS,
Facultad de Ciencias, Universidad Central de Venezuela,
Apdo. 47567, Caracas 1041-A, Venezuela
e-mail: flosavio/amatteo/cmetzner/nnino/nurbina@anubis.ciens.ucv.ve

Resumen

Este trabajo describe la incorporación de un método para la especificación de interfaces gráficas de usuario como parte de un proceso de combinación de métodos orientado a objetos para el desarrollo de aplicaciones. Se muestra por una parte, la flexibilidad del proceso combinado y por otra parte, la insuficiencia de métodos orientados a objetos para la especificación de interfaces gráficas de usuario complejas.

Palabras Claves: Especificación de Interfaz de Usuario, Métodos para el desarrollo de Software Orientado a Objetos, Proceso de Desarrollo de Software, Interfaz Gráfica de Usuario, Ingeniería de Software.

1. INTRODUCCION

Para los desarrolladores de aplicaciones con una componente de interfaz gráfica de usuario (GUI) compleja, son claras las limitaciones de los métodos Orientados a Objetos (O-O) más populares, para la especificación de las interacciones usuario - sistema. Entendemos por complejidad de interfaces gráficas de usuario, aquellas en las que el número de diálogos es alto ó un diálogo involucra secuencias largas de interacción usuario - sistema.

El método propuesto por [Jaa 95] presenta un enfoque "top-down" para la especificación de GUI's, que permite al desarrollador concentrarse en los aspectos más importantes de la interfaz usuario y dividir la fase de diseño en partes manejables. El proceso combinado de métodos O-O presentado en [LMM 97] cubre todas las fases del ciclo de desarrollo de sistemas, pero los métodos combinados, OOSE[2] [Jac & al 93] y OMT[3] [Rum & al 91], son pobres en la especificación de interfaces usuarios; en particular, si se desea expresar la especificaciones de diálogos usuario - sistema. A fin de solventar ésta deficiencia, se propone integrar al proceso combinado un método para la especificación de GUI, método utilizado con éxito [Jaa 95] para el desarrollo de aplicaciones en Nokia Telecommunications, Finlandia y se experimenta la integración sobre un caso de estudio: el desarrollo de la herramienta DyMoT[4] [NU 97], para la construcción de modelos dinámicos en el ambiente OODEST[5] [LMM 94] el cual es un Case Multimétodo para el análisis y diseño de aplicaciones, que ofrece el uso de métodos O-O OMT [Rum & al 91], Booch [Boo 94] y Coad & Yourdon [CY 91].

Este trabajo está estructurado en dos secciones principales, aparte de la introducción y las conclusiones. En la Sección 2 se describe brevemente el proceso combinado y su adaptación para incorporar la especificación de GUI's [LMM 97]. En la Sección 3 se presenta la experiencia en la aplicación a un caso de estudio.

2. INCORPORACION DE ESPECIFICACIONES DE GUIS AL PROCESO COMBINADO

No existen en OMT o en OOSE mecanismos gráficos o textuales que permitan expresar, de manera sencilla y clara, situaciones como las que se ejemplifican a continuación: las transiciones entre ventanas, los diálogos con el usuario, cuales ventanas pueden duplicarse, dado un evento, cuales cambios de estado ocurren sobre las ventanas (aparecer o desaparecer). En OOSE pueden ser utilizados los diagramas de interacción de objetos (OID), pero para secuencias largas de interacción la complejidad de estos diagramas es notable. En OMT pueden ser utilizados el modelo dinámico para especificar actividades, acciones y eventos en transiciones.

Para la especificación de interfaces [Jaa 95] sugiere los pasos siguientes:

---

[1] Este trabajo está enmarcado en el contexto de los proyectos grupales CDCH AC++E/gen N° 03-13-3482-95, OOMGRIN N° 03-13-3483-95, y CONICIT MOODe N° S1-9500512

[2] Object-Oriented Software Engineering

[3] Object Modeling Technique

[4] Dynamic Model Tool

[5] Object Oriented Design Support environmenT

1. Análisis de las Operaciones: una operación es una actividad de la aplicación que debe ser implementada. Se identifica por las funcionalidades que debe proveer el sistema. Una operación está formada por cero ó más tareas, una tarea por cero ó más acciones y una acción es una actividad atómica que el usuario puede ejecutar con la interfaz usuario. En éste paso se identifican y analizan las operaciones que necesita el usuario y se crean tareas relacionadas con éstas. A partir de ellas se identifica una lista de tareas comunes y no comunes.
2. Especificación de la Estructura: permite identificar los diálogos necesarios para la aplicación, las tareas que se realizan dentro de cada diálogo y las relaciones entre diálogos a través de los diagramas de diálogo, como se ilustra en la figura 1.

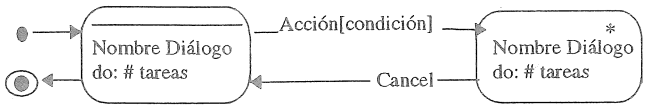

Figura 1. Diagrama de Diálogo

3. Especificación de Componentes: se identifican los tipos de componentes de interfaz usuario que el usuario necesita para ejecutar las tareas. El tipo de componentes puede ser "*feedback*", que informa al usuario sobre el estado de un objeto de la aplicación, de *manipulación* para la manipulación de objetos de la aplicación y *combinada* que posee ambos comportamientos, de manipulación y de "feedback".
4. Visualización: permite visualizar cada diálogo de la aplicación y definir el formato exacto y el estilo de la interfaz usuario desarrollando un prototipo.
5. Especificación de Tareas: detalla cómo la ejecución de las tareas se refleja en la interfaz usuario. Cada una de las tareas se representa con un diagrama de traza de eventos, que muestra la comunicación usuario - sistema y los elementos de la interfaz usuario.

De los pasos descritos se observa que el método se centra en la identificación de interacciones entre objetos, en la definición de los diálogos y utiliza la notación del método OMT

Una GUI está formada por cero ó más diálogos, cada diálogo está formado por cero ó más componentes y cada componente por una ó más herramientas. (Ver figura 2).

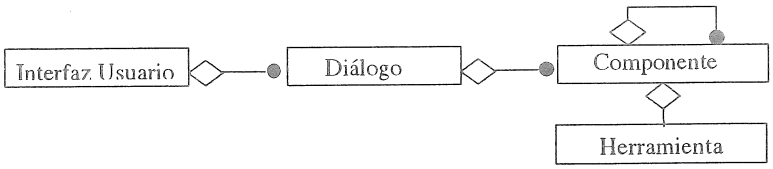

Figura 2. Elementos de la Interfaz de Usuario

2.1. Proceso Combinado de Métodos Orientado a Objetos

El proceso combinado tiene por objetivo utilizar las características resaltantes y mejores de dos de los métodos O-O mas populares como son OOSE [Jac & al 93], y OMT [Rum & al 91], incorporando el modelo Use Case - PAC [LM 97] para el diseño de la componente de interfaz.

El Proceso de Combinación para la Fase de Análisis [LMM 97], consiste en los siguientes pasos:

1. Desarrollar el Modelo de Requerimientos: se extraen y enumeran los "use case", esto es, una lista de todas las facilidades que el sistema debe proveer. Se continúa definiendo los actores, elaborando las descripciones de los "use case" (caso de uso normal), y construyendo el prototipo de interfaz gráfica. Luego, se describen "use case" excepcionales y de error, y opcionalmente se construye un modelo objeto inicial que describe el dominio del problema.
2. Desarrollar el Modelo de Análisis: se crea un modelo objeto parcial para cada "use case" elaborado, identificando objetos interfaz, control y entidad.
3. Validar el Modelo de Análisis con respecto a los requerimientos del usuario definidos en el Modelo de Requerimientos.
4. Estructurar los objetos interfaz de acuerdo al Modelo Use Case - PAC.
5. Desarrollar los Diagramas de Interacción de Objetos, los cuales definen las interacciones entre el sistema y los actores y, los eventos de comunicación entre los objetos.
6. Trasladar el Modelo de Análisis a la notación de OMT.

7. Modelar los cambios de estados de los objetos, usando los Diagramas de Transición de Estados (DTE), un DTE para cada clase de objetos que posea comportamiento dinámico.
8. Desarrollar el Modelo Funcional de alto nivel usando el esquema propuesto en [Rum 95].

El proceso descrito es iterativo e incremental, es decir, si un nuevo requerimiento es identificado o uno existente es cambiado durante cualquier etapa, se define un "use case" que satisfaga éstos requerimientos o se modifica uno existente, cambiando de manera acorde el modelo desarrollado. De ésta manera, todos los requerimientos no tienen que ser definidos inicialmente, sino que pueden ser agregados posteriormente o redefinidos. Una vez definidos los modelos usando la notación OMT, se organizan y se refinan las clases, sus relaciones y su implementación tal como se definen en el proceso de diseño de OMT.

2.2. Incorporación de la Especificación de GUI al Proceso Combinado

Un proceso no tiene que ejecutarse de una manera estricta, se pueden combinar varios métodos si con ello se obtienen mejores resultados, como es el caso del proceso combinado [LMM 97] en el cual se conserva la identidad y notación de cada método individual. Los elementos metodológicos de [Jaa 95] son un superconjunto de [LMM 97]: ambos se basan en OMT, ambos utilizan conceptos de OOSE, por lo tanto la integración basada en estos aspectos comunes parecía ser prometedora. Puesto que eran justamente los OID's los que perdían claridad y simplicidad cuando la secuencia de interacciones es compleja, el desarrollo de los OID's se realizará utilizando [Jaa 95]. Esto es, en el paso cinco de la sección 2.1.

La figura 3 resume el diagrama de pasos del proceso obtenido.

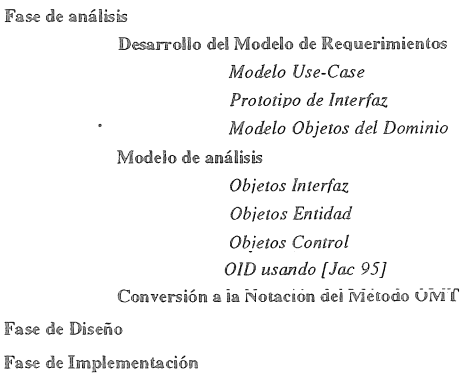

Figura 3. Resumen del Proceso Obtenido

3. CASO DE ESTUDIO: DESARROLLO DE DyMoT

DyMoT es una herramienta gráfica [NU 97] que permite el desarrollo de modelos dinámicos de aplicaciones construidas con el ambiente OODEST en el cual la persistencia se obtiene a través del Sistema Manejador de Base de Datos Orientado a Objetos O_2.[Her 97]. Un modelo dinámico presenta la visión del comportamiento del sistema y en OODEST se utiliza la notación OMT para estos modelos. El modelo dinámico manipula los siguientes conceptos: *Clase*: descripción de un grupo de objetos con propiedades similares, comportamiento, relaciones y semántica comunes, *Evento*: es un estímulo enviado de un objeto a otro y no tiene duración, *Transición*: es un cambio de estado causado por un evento, *Restricción*: es una relación funcional entre objetos, clases, atributos, enlaces y asociaciones; una declaración acerca de una condición o relación que debe ser mantenida como verdad y *Estado*: representa un período de tiempo durante el cual un objeto espera por la ocurrencia de un evento.

El modelo dinámico de una aplicación consiste de:

- Descripción de Escenarios: describe una secuencia de eventos durante la ejecución particular de un sistema. Puede ser representado en forma textual.
- Diagrama de Traza de Eventos: muestra el envío y recepción de una secuencia de eventos.
- Diagrama de Transición de Estados: es un grafo dirigido en el cual los nodos representan estados de un objeto y los arcos representan transiciones entre los estados.
- Diagrama de Flujo de Eventos: es un grafo dirigido en el cual los nodos representan objetos del sistema y los arcos representan eventos entre ellos.

3.1. Desarrollo del Modelo de Requerimientos

La figura 4 ilustra el modelo "use case" del sistema DyMoT. Los requerimientos funcionales son: Describir Escenarios, Generar Diagrama de Traza de Eventos, Generar Diagrama de Transición de Estados y Describir Diagrama de Flujo de Eventos, y los actores identificados son: Actor ***OODEST***: proporciona las clases de objetos y sus relaciones y las restricciones de una aplicación que intervendrán en la generación de Escenarios, Diagramas de Estados, Diagramas de Traza de Eventos y Diagramas de Flujo de Eventos, Actor ***Usuario_Desarrollador***: realiza solicitudes referentes a la creación de Escenarios, Diagramas de Traza de Eventos, Diagramas de Estados y Diagramas de Flujo de Eventos para una aplicación y Actor ***Smbdooo***$_2$.

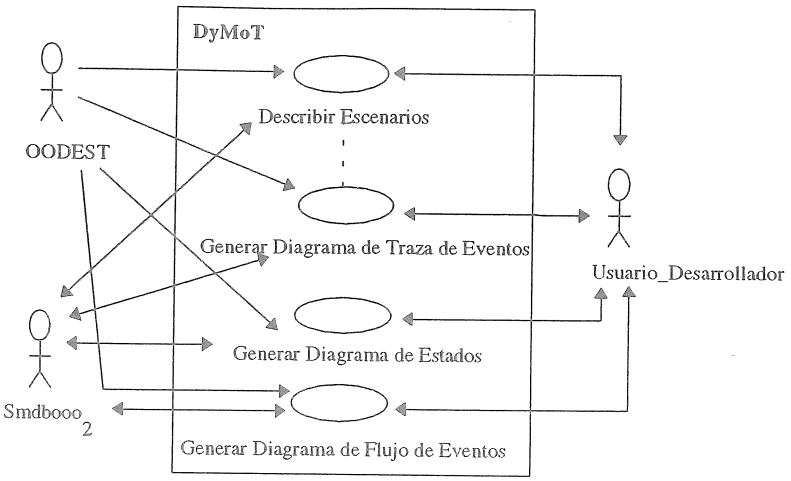

Figura 4. Modelo "Use Case" del Sistema DyMoT

Como parte del modelo de requerimientos se definen los prototipos de interfaz. En la figura 5 se ilustra el prototipo de ventana establecido en el ambiente OODEST, el cual está constituido por el *Toolbar*, que a su vez posee un área para el título de la ventana y un área de opciones, un *área de edición* y un *área de constructor* que a su vez está constituido por botones asociados a las paletas del editor. La opción *Dynamic_Mode*l al seleccionarse despliega un *Menu popup* cuyas opciones representan las funcionalidades de DyMoT. Cada opción desplegará una ventana que mantiene el prototipo establecido en el ambiente con opciones propias del "use case".

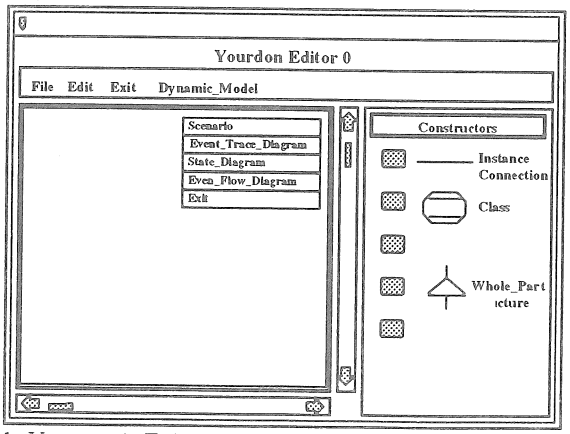

Figura 5. Prototipo de Ventana de Edición del Editor Coad & Yourdon en el Ambiente OODEST.

3.2. Desarrollo del Modelo de Análisis

A partir de los "use case" especificados en el modelo de requerimientos se identificaron los objetos interfaz, entidad y control. En particular a cada actor se le asocia un objeto interfaz a saber:

Smbdooo$_2$: permite realizar conversiones para estandarizar los protocolos de comunicación entre el sistema y el Sistema Manejador de Base de Datos O_2 [O_2T 95].

Pantalla: controla y manipula información de la interfaz.

La relación del objeto *Pantalla* se ilustra en la figura 6, el cual mantiene una comunicación directa con el *Objeto Control*. A su vez, *Objeto Control* puede accesar la información, involucrada con los diagramas de una aplicación de un "use case", a través del $Smbdooo_2$ y transferirla a los objetos entidad específicos para la realización de los "use case".

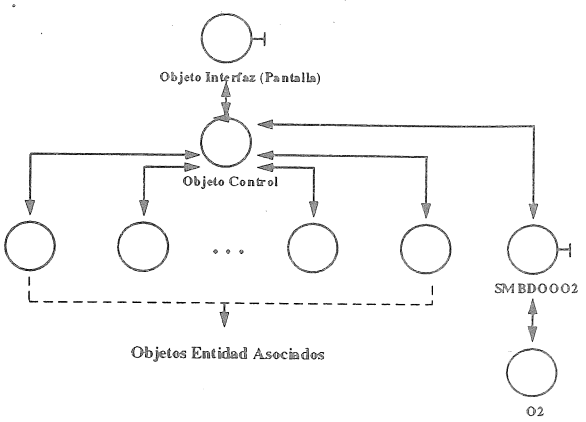

Figura 6. Arquitectura del Objeto Interfaz *Pantalla*

Además se definen los objetos asociados al objeto interfaz *Pantalla: Ventana_Interacción, Ventana_Mensaje* y *Menu_Popup*. Los objetos entidad identificados son: *Clase, Evento, Transición, Estado, Restricción, Escenario, Diagramatrazaevento, Diagramaestado, Diagramaflujoevento* y O_2. Las asociaciones entre estos objetos se ilustra en la figura 7.

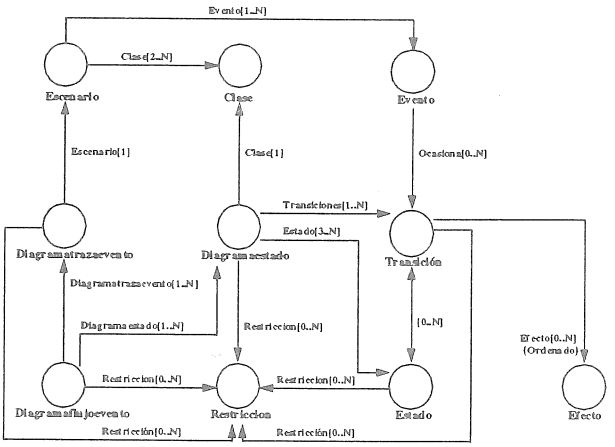

Figura 7. Relación de Agregación de los Objetos Entidad

3.5. Desarrollo de los Diagramas de Interacción de Objetos

1. Análisis de las Operaciones

Los "use case" descritos en el modelo de requerimientos, permiten identificar las operaciones principales que el usuario puede realizar con DyMoT. La figura 8 ilustra las principales operaciones y la lista de tareas para una de las operaciones. Posteriormente se genera una lista de tareas, enumerando las tareas comunes y no comunes asociadas a las operaciones definidas.

2. Especificación de la Estructura

En el desarrollo de DyMoT, el Prototipo de Interfaz se realizó en el Modelo de Requerimientos. El Diagrama de Diálogo que se muestra en la figura 9 representa los diálogos, las interacciones y las tareas necesarias para invocar DyMoT y sus funcionalidades.

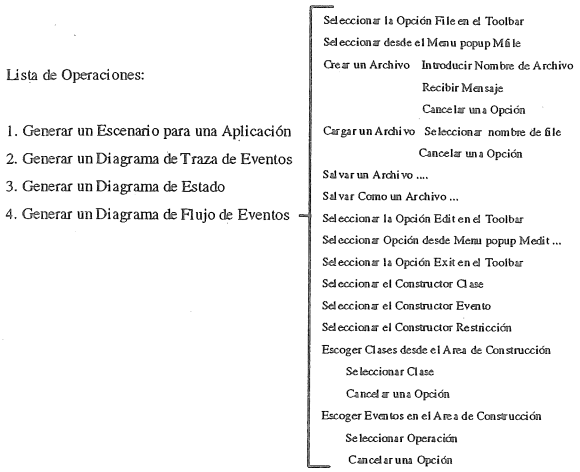

Figura 8. Operaciones y Lista de Tareas para la Operación Generar Diagrama de Flujo de Eventos

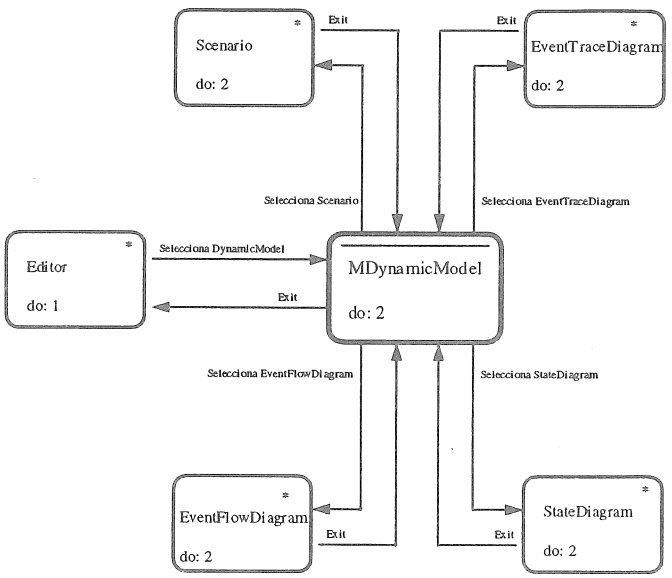

Figura 9. Diagrama de Diálogo: *MDynamicModel*

El diálogo principal *MDynamicModel* puede ser invocado desde cualquier Editor del ambiente OODEST con la tarea N° 1: Seleccionar la opción *Dynamic_Model* desde el *Toolbar* del diálogo *Editor*.

Desde el diálogo *MDynamicModel*, el usuario puede ejecutar la tarea N° 2: Seleccionar una opción (*Scenario*, *Event_Trace_Diagram*, *State_Diagram* o *Event_Flow_Diagram*) desde el *Menu popup*, en cuyo caso se despliega el diálogo asociado a la opción. En cualquier diálogo, el usuario puede seleccionar la opción *Exit* y retornar el control al diálogo invocador.

3. Especificación de Componentes

La figura 10 ilustra el Diálogo *Pantalla*, el cual está constituido por los componentes *Toolbar*, *Area_Edición* y *Area_Construcción*.

El componente *Toolbar* formado por una herramienta "feedback" (*Area_Título*) que permite identificar el título de la pantalla, y un componente *Area_Opción*, que contiene herramientas del tipo combinado (*Opciones*) que presenta las opciones que el *Usuario-Desarrollador* puede accesar para ejecutar las tareas y una herramienta de manipulación (*Exit*), el componente *Area_Edición* está formado por una herramienta del tipo combinada (*Area_Edición*) que le permite al usuario realizar manipulaciones en el área e interactuar con el sistema.

La última componente, del Diálogo *Pantalla*, es *Area_Constructor* y está constituida por herramientas de manipulación (*Botón_Constructor*) y herramientas combinadas (*Botón*).

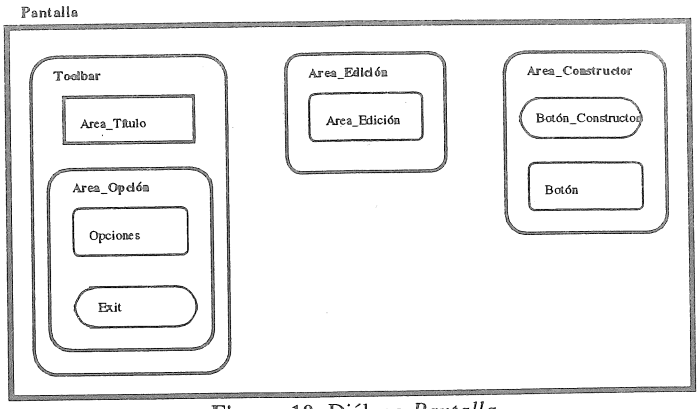

Figura 10. Diálogo *Pantalla*

4. **Visualización**

Como parte del Modelo de Requerimientos se establecieron los prototipos de interfaces gráficas de usuario y éstos mantienen la consistencia con los diálogos definidos.

5. **Especificación de Tareas**

Los diagramas de interacción en [Jaa 95], presentan un objeto sistema que encapsula el comportamiento de los objetos de control y los objetos entidad, impidiendo la identificación de los objetos y las interacciones que ocurren internamente. Se incorpora en la fase de especificación de tareas la descomposición del objeto sistema en objetos entidad y control mostrando las interacciones que ocurren entre ellos.

En la figura 11 se muestra el diagrama de interacción para la tarea *introducir operaciones*, donde el Usuario_Desarrollador puede ejecutar la tarea *introducir una operación* en el área de interacción de la ventana de interacción y presionar el botón *Save*. Luego se realizan una secuencia de acciones de transferencia de control e información entre los elementos del sistema. Si el usuario presiona el botón *Accept* en el área de botones de la ventana de interacción, se finaliza la ejecución de la tarea retornando el control al área de edición de *Pantalla*.

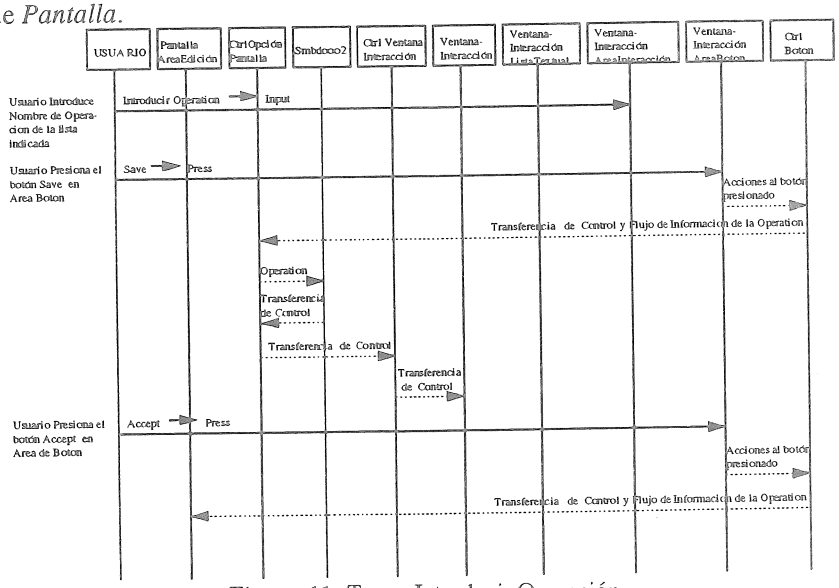

Figura 11. Tarea: Introducir Operación

3.6. Trasladar el Modelo de Análisis a la Notación de OMT

Los objetos entidad, interfaz y control, en este modelo, se representan como clases, por lo tanto los objetos interfaz *Pantalla*, *Ventana_Interacción*, *Ventana_Mensaje* y *Menupopup* constituyen clases identificadas con sus respectivos nombres, que mantienen consistencia con las definiciones realizadas en el Modelo de Análisis en OOSE. Estas clases conforman la interfaz usuario - sistema.

Los objetos entidad identificados se representan a través de clases del mismo nombre y la relación entre las clases se ilustra en el diagrama de clases de la figura 12.

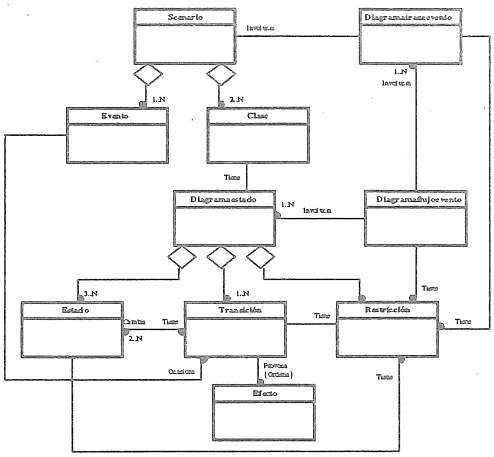

Figura 12. Diagrama de Clases de DyMoT

3.7. Modelar los Cambios de Estados

La figura 13 presenta el diagrama de estado para la clase *Pantalla*. El estado inicial se activa con el evento *display_window*. En el estado inicial se ejecutan las actividades mostradas a continuación de la palabra reservada *do*, que permiten dibujar la pantalla. Al completar estas actividades se cambia al estado *Activa*. Si ocurre el evento *activate_dialog* se cambia al estado *No Activa* ejecutándose la acción *display_dialog*. Al cerrar el dialogo, se genera un nuevo evento que ocasiona la transición al estado *Activa*. El estado final se dispara cuando ocurre el evento *close_window*.

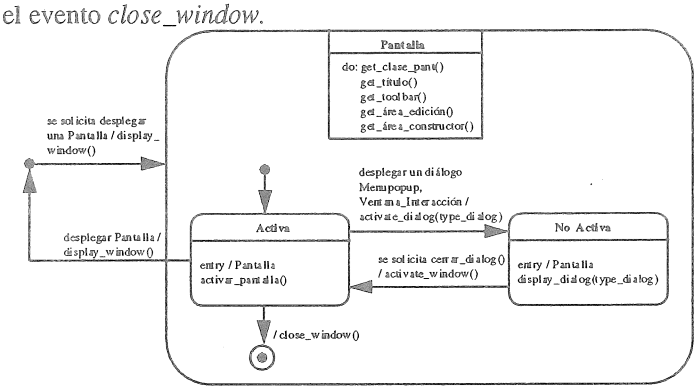

Figura 13. Diagrama de Estado para la Clase *Pantalla*

3.8. Desarrollo del Modelo Funcional

En este modelo se describen las operaciones de los objetos, en particular, se describe como la ejecución de una operación afecta los valores de objetos.

Las operaciones de los objetos se especifican de manera no procedural, siguiendo el esquema presentado en [Rum 95]. A titulo ilustrativo se muestra en la figura 14 la especificación de la operación *create*.

Operación: Create(Nombre Archivo: string)

Responsabilidades: Activa el área de edición con el nombre del archivo dado.

Entradas: El nombre del objeto de la clase: Escenario, Diagrama de Traza de Eventos, Diagrama de Estados y Diagrama de Flujo de Eventos que se desea crear.

Retorno:

Objetos Modificados: Modifica un objeto de la clase: Escenario, Diagrama de Traza de Eventos, Diagrama de Estados y Diagrama de Flujo de Eventos, el área de edición y área de título (toolbar) de la pantalla activa.

Precondición: El nombre de archivo no exista en la base de datos O2.

Postcondición: El área de edición de la pantalla está activada y totalmente en blanco. Y en el área de título aparece el nombre del archivo creado.

Figura 14. Especificación de la Operación *Create*

Con la definición de los tres modelos OMT finaliza la fase de análisis y se procede con la fase de diseño: de sistema y de objetos. La implementación de DyMoT se realizó en el lenguaje C++ [Str 93] sobre una plataforma Unix con OSF/Motif[6].

Algunas métricas recolectadas para el caso de estudio son las siguientes:

No de Use Case: 4
No. de Operaciones: 4
No. de tareas Principales:21
No. de Subtareas: 41
No de Diagramas de Interacción: 30
No. de clases de análisis (en el dominio del problema):
Clases Interfaz: 17
Clases Entidad: 22
Clases Control: 4

No de Clases de Diseño (en la solución)
Clases Interfaz: 14
Clases Entidad: 11
Clases Control: 6

No. promedio de atributos visibles / clases: 4
No. promedio de métodos públicos / clase: 10
Profundidad Máxima de la Jerarquía de clasificación (DIT): 2
Máximo Número de Hijos (NOC): 9

4. CONCLUSIONES

El desarrollo del caso de estudio utilizando la especificación y el proceso nos permite concluir que

- El proceso combinado de métodos O-O [LMM 97], es un proceso flexible que permite la incorporación de otros métodos, tal como el presentado en éste trabajo. La definición precisa de los pasos del proceso y sus entradas y resultados, sirven de ayuda para identificar en que paso se puede extender el proceso o en que paso se puede reducir el proceso.
- Se sugiere la incorporación de mecanismos de jerarquización para los diagramas de interacción. Si bien para sistemas con GUI's complejas, los diagramas desarrollados al ser mas pequeños, son mas sencillos y comprensibles que los OID de OOSE, la cantidad de diagramas generados puede exceder la capacidad de comprensión e interacción general de objetos sobre todo si son utilizados para sesiones de revisión con usuarios. A manera de ejemplo, las métricas del caso de estudio muestran que es una aplicación pequeña, cuando se considera el numero de clases. Pero se elaboraron 4 operaciones (y 4 "use case") 30 diagramas de interacción para las tareas identificadas. Esto es, el número de los Diagramas de Interacción puede incrementarse de tal manera que no se aprecia la interacción general entre los objetos.
- Se recomienda realizar un estimado de la complejidad de la GUI antes de utilizar el proceso [LMM 97] con la incorporación del método. Si la complejidad no es alta, es preferible usar los diagramas de interacción tal como se presentan en la sección 2.

5. BIBLIOGRAFIA

[Boo 94] BOOCH G.;
"Object-Oriented Analysis and Design with Applications ",
Benjamin/Cumminhgs Company, 1994.

[CY 90] COAD P.; YOURDON E.;
"Object-Oriented Analysis", Englewood Cliffs, NJ Prentice Hall, 1991.

[Her 97] HERNANDEZ Francisco;
"La Persistencia de Información en el Ambiente OODEST (Object Oriented Design Support environmenT). Una Aplicación del Sistema Manejador de Base de Datos Orientado a Objetos O2" , Trabajo Especial de Grado, Universidad Central de Venezuela, Facultad de Ciencias, Escuela de Computación, Caracas, Mayo 1997.

[6] Open Software Fundation

[Jaa 95] JAAKSI Ari;
"Object-Oriented Specification of User Interfaces", Software-Practice and Experience, Vol. 25(11), John Wiley & Sons, Ltd., November 1995.

[Jac & al 93] JACOBSON I.; CHRISTENSON M.; JONSSON P.; OVERGAARD G.;
"Object-Oriented Software Engineering: A Use-Case Driven Approach", Addison-Wesley, 1993.

[LM 97] LOSAVIO F.; MATTEO A.;
"Use Case and multiagent models object-oriented design of user interfaces", Journal of Object Oriented Programming, Procceding, Centro de Ingeniería de Software y Sistemas ISYS, Facultad de Ciencias, U.C.V., Caracas - Venezuela, Vol. 10, N° 2, Mayo 1997.

[LMM 94] LOSAVIO F.; MATTEO A.; METZNER Ch.;
"OODEST a PCTE-based IPSE for Object-OrientedApplications",
Centro de Ingeniería de Software y Sistemas ISYS, Escuela de Computación U.C.V., Caracas - Venezuela, Abril 1994.

[LMM 97] LOSAVIO F.; MATTEO A.; METZNER Ch.;
"Systems Development Combining Object-Oriented Methods", International Conference on System Analysis and Synthesis (ISAS) 97,
Caracas - Venezuela.

[NU 97] NIÑO Norelva; URBINA Nancy;
"Una Herramienta para la Contrucción de Modelos Dinámicos en el Ambiente OODEST (Proyecto DAAT)" , Trabajo Especial de Grado, Universidad Central de Venezuela, Facultad de Ciencias, Escuela de Computación, Caracas, Mayo 1997.

[O_2T 95] O_2TECHNOLOGY;
"O_2C Reference Manual", March 1995.

[Rum & al 91] RUMBAUGH J.; BLAHA M.; PREMERLANI W.; EDDY F.; LORENSEN W.;
"Object-Oriented Modelling and Design", Prentice-Hall, 1991.

[Rum 95] RUMBAUGH, James;
"OMT: The Functional Model", JOOPMarch-April 1995, Volumen 8, N° 1.

[Str 93] STROUSTRUP Bjarner;
"The C++ Programming Languaje", Second Edition, Addison Wesley, June 1993.