UNIVERSIDAD DEL ACONCAGUA

FACULTAD DE CIENCIAS SOCIALES Y ADMINISTRATIVAS

INGENIERIA EN SOFTWARE

PRACTICA PROFESIONAL

AUTOR: EMILIANO NICOLÁS MARTIN

DOCENTE: LIC. ALEJANDRO VAZQUEZ

LEGAJO: 15356

TEMA: MARCO DE MODELADO CON UML PARA APLICACIONES DESARROLLADAS CON ABL

LUGAR: MENDOZA – ARGENTINA

FECHA: SEPTIEMBRE 2010

CALIFICACIÓN

ÍNDICE

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

RESUMEN TÉCNICO

El desarrollo de la presente tesina está dirigido a demostrar mi aporte profesional en la obtención de documentación mediante UML (Unified Modeling Language - Lenguaje Unificado de Modelado), de aquellas aplicaciones informáticas que se han programado con un lenguaje de programación denominado ABL (Advanced Business Language - Lenguaje Avanzado de Negocio).

ABL surgió a principios de la década de 1980, para permitir la programación de aplicaciones informáticas que necesiten procesar una gran cantidad de datos, en ámbitos de negocios tales como el financiero, manufacturero y gubernamental.

Muchas de las aplicaciones informáticas que se han programado con ABL, no cuentan con algún tipo de soporte documental que muestre cómo fueron diseñadas previamente. La solución a esta problemática viene de la mano de UML.

UML es un lenguaje gráfico que ofrece una simbología estándar, la cual permite documentar en diagramas los diversos aspectos que constituyen a una aplicación informática, desde las interfases de usuario (pantallas) hasta las bases de datos, entre otros.

Por lo tanto, en este proyecto se ha tomado como ejemplo una aplicación informática programada con ABL denominada “Gestión de Expedientes”, la cual es utilizada en Instituciones Públicas. Dicha aplicación no posee documentación acerca de su diseño; para obtenerla se especificará qué componentes de UML se necesitan utilizar y cómo estos serán utilizados. A este desarrollo se lo ha denominado “Marco de modelado con UML para aplicaciones desarrolladas con ABL”.

CAPITULO I:

INTRODUCCIÓN

1. 1 Introducción

En el marco de la Carrera Ingeniería en Software de la Facultad de Ciencias Sociales y Administrativas, Universidad del Aconcagua, se presenta la tesina denominada "Marco de modelado con UML para aplicaciones desarrolladas con ABL", con el fin de obtener el Título de Grado de Ingeniero en Software.

El Lenguaje Avanzado de Negocio, o ABL (Advanced Business Language), es un lenguaje de programación que permite desarrollar aplicaciones informáticas, las cuales estén destinadas a resolver necesidades en ámbitos de negocio tan diversos entre sí, como lo son el financiero, manufacturero y gubernamental, entre otros. Es por ello que estas aplicaciones reciben el nombre de aplicaciones de negocio.

ABL ha sido específicamente creado para construir todos los aspectos de una aplicación de negocio, incluyendo las interfases de usuario, la lógica de negocio y la gestión compleja de datos.

La mayoría de las aplicaciones que se han desarrollado con ABL suelen tener 10 o incluso 20 años de antigüedad. El problema que presentan estas aplicaciones es que se han construido sin pasar por algún proceso de análisis y diseño conceptual, por lo cual no cuentan con documentación relacionada.

Desde hace un tiempo a la actualidad, el modelado visual de software ha ido creciendo dentro de la comunidad informática, como la técnica para representar gráficamente los diferentes aspectos que dan forma a las aplicaciones, desde la especificación de requerimientos hasta la implantación en los clientes, entre otros. El principal exponente en la materia es el Lenguaje Unificado de Modelado, o mejor conocido como UML por sus siglas en inglés de Unified Modeling Language.

UML es un lenguaje gráfico que define reglas y notaciones para visualizar, especificar, construir y documentar sistemas de software.

Por lo expresado hasta aquí, la hipótesis que se aborda en el ámbito de la presente tesina consiste en demostrar que, con el uso de UML se puede hacer ingeniería inversa sobre aplicaciones desarrolladas con ABL, con el objeto de generar representaciones gráficas que documenten sus interfases de usuario, módulos de código y acceso a bases de datos, y faciliten su comprensión.

1. 2 Objetivos Generales

Los objetivos que se pretenden cumplir en este proyecto son los siguientes:

- Analizar conceptualmente las interfases de usuario, módulos de código y acceso a bases de datos de toda aplicación desarrollada con ABL.

- Exponer una propuesta estándar de modelado con UML para dichas aplicaciones.

- Desarrollar un marco de modelado experimental en base a la propuesta anterior, para aplicarlo a una opción particular de una aplicación de negocio real denominada Gestión de Expedientes.

CAPITULO II:

MARCO TEÓRICO

2. 1 ABL

2. 1. 1 Origen

En el año 1981, un grupo de profesionales graduados del prestigioso Instituto de Tecnología de Massachusetts, en Estados Unidos, decidieron juntarse para fundar una compañía de software bajo el nombre de Data Language Corporation. Unos años más tarde, en 1987, la compañía fue renombrada a Progress Software Corporation (PSC).

Desde sus comienzos hasta nuestros días, PSC viene ofreciendo una completa infraestructura de productos y servicios de software para el desarrollo, despliegue, integración y gestión de aplicaciones informáticas de negocio.

El producto más conocido de la compañía fue bautizado inicialmente como PROGRESS. Consiste básicamente en un lenguaje de programación de cuarta generación (4GL, de Fourth-Generation Programming Language) y en un sistema de administración de bases de datos relacionales (RDBMS, de Relational Data Base Management System). Juntos, el lenguaje y la base de datos, han sido llamados PROGRESS 4GL/RDBMS. Su primer lanzamiento comercial fue en el año 1984 con la versión 2.1.

Desde un principio, PSC se ha caracterizado en que cada una de las versiones de PROGRESS cumpla con una política constante de compatibilidad hacia atrás, es decir, haciendo que los desarrollos realizados en versiones anteriores mantengan totalmente su funcionalidad al implementarlos bajo las nuevas versiones.

En el año 1993 fue publicada la versión 7 de PROGRESS, convirtiéndose en la primera versión compatible con el sistema operativo Windows.

A mediados de la década de 1990, PSC comenzó a establecer un perfil más amplio dentro de la industria del software con la adquisición de algunas empresas pequeñas del medio, entre las cuales se encontraban Apptivity, Sonic Software Corporation y NuSphere Corporation. Desde entonces, la compañía ha continuado utilizando este tipo de práctica comercial, la cual le ha permitido expandir su oferta de productos y servicios hacia nuevos horizontes tecnológicos, y a la vez dividir su negocio en unidades operativas. Un claro ejemplo de tal escenario se dio en el año 1999, cuando creó la plataforma de negocio denominada Progress OpenEdge (en adelante OpenEdge) con el objeto de mejorar su tradicional sistema de desarrollo de aplicaciones.

La introducción en el mercado de una nueva plataforma de desarrollo llevó a que el tradicional lenguaje de programación Progress 4GL pasara a llamarse OpenEdge ABL. El motivo de este cambio era superar una supuesta percepción de la industria acerca de que los lenguajes de cuarta generación eran menos competentes que otros lenguajes.

2. 1.2 Conceptos básicos

John Sadd lo define en el capítulo 1 del libro ABL Handbook como "*un lenguaje de programación de procedimientos de alto nivel, desarrollado para permitir la construcción de casi todos los aspectos de una aplicación de negocio empresarial, desde la interfaz de usuario hasta el acceso a las bases de datos y la lógica de negocio*". Para comprender un poco más este concepto, vamos a analizar la expresión "*lenguaje de programación de procedimientos de alto nivel*" desde los puntos de vista que ofrecen sobre el lenguaje los siguientes términos claves:

- Procedimientos: Este término hace referencia al paradigma de programación por procedimientos en que se basa ABL. Siguiendo la filosofía de construcción de software que propone este paradigma, ABL permite organizar el código fuente de una aplicación en módulos llamados procedimientos, los cuales se encargan de realizar todas aquellas tareas simples y complejas que dan funcionalidad a la aplicación.

- Alto nivel: Este otro término hace referencia al elevado nivel de abstracción que ofrece ABL a través de sus sentencias de programación y palabras claves. Esto quiere decir que, una simple instrucción en un 4GL como ABL, puede hacer el trabajo de decenas o posiblemente cientos de líneas de código en un lenguaje de tercera generación (3GL, de Third-Generation Programming Language) estándar como Visual Basic, Java o C++.

ABL es un lenguaje específicamente propuesto para expresar de manera concisa y eficiente procesos de negocio. Estas operaciones o entidades de negocio suelen tener un importante componente de procesamiento de datos. Para ello, ABL incluye poderosas sentencias y palabras claves que se especializan en la construcción de aplicaciones de negocio. Estas expresiones del lenguaje utilizan una sintaxis breve, de fácil lectura y muy cercana al idioma Inglés que simplifica el desarrollo de software.

ABL es el único lenguaje de programación actual que proporciona capacidades integradas para acceder, manipular y almacenar datos de diferentes fuentes de datos y formatos, incluyendo las bases de datos relacionales de OpenEdge y las de otros gestores populares como Oracle y Microsoft SQL Server, XML (eXtensible Markup Language), archivos estructurados y no estructurados, formatos definidos por el usuario, etc. Combinar estas capacidades con una sofisticada lógica de negocio, hace que ABL sea particularmente adecuado para desarrollar grandes aplicaciones de negocio.

En el libro denominado Progress, PSC establece las principales filosofías por las cuales ABL deposita todos sus esfuerzos:

- Hacer al desarrollador de aplicaciones más productivo, minimizando la cantidad de código necesario para crear aplicaciones completas y de gran alcance.

- Aislar al desarrollador de posibles cuestiones como la portabilidad, dejando que el lenguaje maneje esas cuestiones específicas de la plataforma durante el tiempo de compilación o el tiempo de ejecución.

- Proporcionar características flexibles e inteligentes, que permitan al usuario final trabajar de manera intuitiva y productiva con aplicaciones de gran capacidad de respuesta.

2. 1. 3 Evolución

En sus primeras versiones, ABL permitía a los desarrolladores construir aplicaciones de interfaz de caracteres que podían ejecutarse sobre una amplia variedad de plataformas de hardware, incluyendo UNIX, DOS, y algunos otros sistemas operativos que ya no se usan.

Las primeras aplicaciones desarrolladas sobre OpenEdge eran completamente ejecutables entre plataformas, de modo que un desarrollador podía simplemente mover una aplicación de un tipo de máquina o un tipo de pantalla de terminal a otro con la confianza que funcionaría correctamente en todas partes.

Con la creciente presencia de Windows como una plataforma para interfases gráficas, ABL evolucionó para dar soporte a este tipo de interfases, con toda su diversidad de controles visuales, además de las construcciones de programación dirigidas por eventos necesarias para una aplicación basada en una de barra de menú de opciones y controlada con un mouse.

Hoy en día el lenguaje continúa creciendo, con más nuevas extensiones para proporcionar cada vez más definición dinámica de componentes de aplicación, así como el acceso a tecnologías abiertas como XML, y una serie de otras construcciones para dar soporte a un entorno abierto de desarrollo y despliegue de aplicaciones.

En todo momento, las aplicaciones basadas en ABL se pueden mover de una versión a la siguiente en gran parte sin cambios. Esta ventaja se debe a que el lenguaje proporciona un grado de compatibilidad y migración ascendente no comparable a ningún otro lenguaje de programación de alto nivel.

Una de las mejoras más recientes añadidas a ABL, a partir de la versión 10.1A de OpenEdge, fue la incorporación de los principios de la programación orientada a objetos.

2. 1. 4 Análisis conceptual de aplicaciones desarrolladas con ABL

A partir de aquí, en las próximas secciones se analizan desde un punto de vista conceptual las interfases de usuario, módulos de código y acceso a bases de datos de toda aplicación desarrollada con ABL.

2. 1. 4. 1 Interfases de usuario

La interfaz de usuario es el medio de comunicación entre un usuario y una computadora. Desde el punto de vista del hardware, está representada por aquellos dispositivos periféricos que son utilizados para la entrada y visualización de datos, como el teclado, mouse, monitor o pantalla, etc. En cambio desde el punto de vista del software, la interfaz está representada por aquellos elementos de una aplicación, como ventanas, controles, menús, etc., que el usuario ve y con los cuales interactúa.

Es importante distinguir entre dos tipos de interfases de usuario, la interfaz de usuario gráfica o GUI (Graphical User Interface) y la interfaz de usuario de caracteres o CHUI (Character User Interface). En las siguientes figuras se pueden apreciar estas interfases.

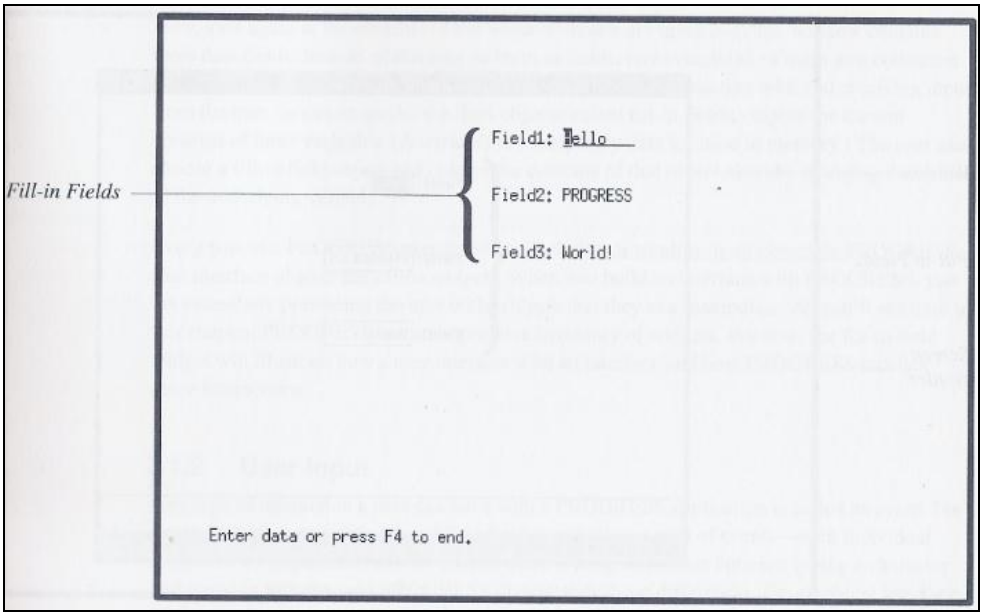

Figura 1: Interfaz de usuario basada en caracteres
Fuente: Progress Language

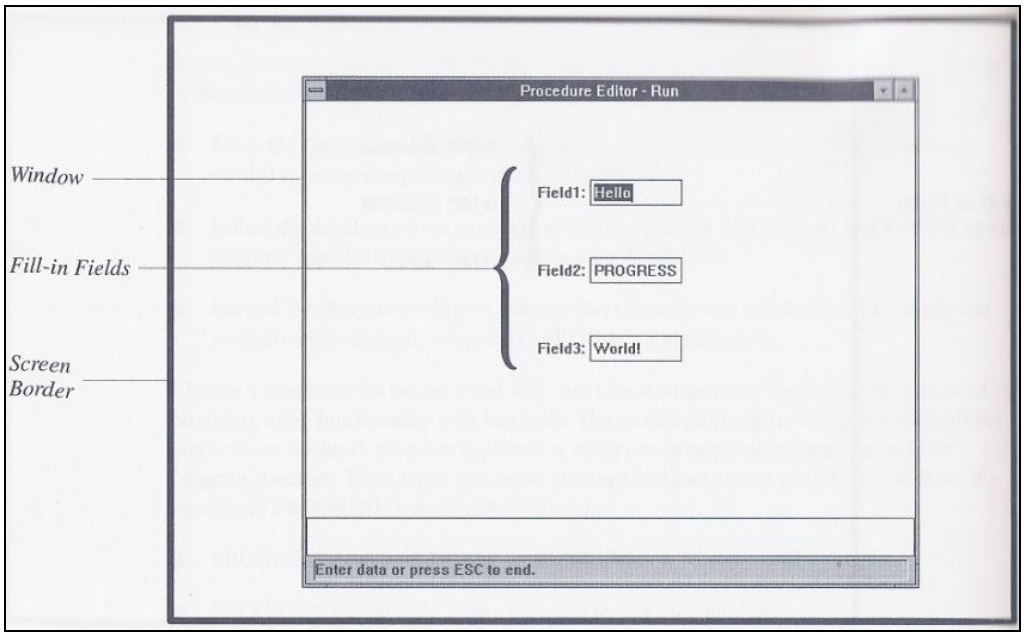

Figura 2: Interfaz de usuario basada en gráficos
Fuente: Progress Language

Una de las diferencias entre estos dos tipos de interfases está en que, una CHUI ofrece gráficos de baja resolución basados en caracteres y permite solamente la entrada de datos a través de un teclado, en cambio, una GUI ofrece gráficos de alta resolución y permite adicionalmente la entrada de datos a través de un mouse.

Otra de las diferencias entre estas interfases se basa en que, mientras una CHUI ocupa toda la pantalla, una GUI está dentro de un objeto que ocupa solamente una parte de la pantalla. Este objeto se llama ventana. A pesar de que no se puede ver un borde gráfico, una CHUI también está contenida en una ventana. Las interfases de caracteres utilizan una sola ventana que no tiene borde gráfico y abarca toda la pantalla. Las interfases gráficas pueden utilizar múltiples ventanas superpuestas, o pueden utilizar una sola ventana que ocupe toda la pantalla; en cualquier caso, las ventanas GUI siempre tienen un borde gráfico.

Volviendo a las figuras mostradas anteriormente, se puede observar que cada una de las ventanas contiene tres campos de datos. En lugar de pensar en ellos solo como campos, no obstante, también se los puede considerar como una colección de objetos individuales. Cada objeto tiene la capacidad para comunicarse y recibir entradas del usuario. En los ejemplos, los objetos de campos de datos muestran el contenido actual de tres variables. El usuario puede elegir uno de ellos y cambiar el contenido de ese objeto, cambiando así el valor de la variable subyacente.

Para terminar de expresar la idea planteada en el párrafo anterior podemos agregar que, cada parte de una interfaz de usuario en ABL, incluyendo una ventana, es un objeto. En ABL, los objetos de interfaz de usuario se llaman widgets. Por lo tanto, cuando se construye una interfaz con ABL, esencialmente se están presentando al usuario los widgets que este puede manipular.

Uno de los widgets más utilizados en las aplicaciones ABL son los menú. Estos se describen a continuación.

2. 1. 4. 1. 1 Menú

Los menú añaden funcionalidad a una interfaz. A través de éstos, los usuarios pueden acceder a las diferentes funciones de una aplicación en el orden que mejor les convenga.

Los menú son una parte importante de un modelo de programación dirigido por eventos, como el que ofrece ABL, ya que permiten a los usuarios controlar el flujo de una aplicación. Para ello, un menú ofrece diferentes opciones a los usuarios, las cuales les dan más control sobre cómo utilizar una aplicación.

ABL define un menú como un widget que contiene una lista de comandos u opciones disponibles para los usuarios. En ABL, un menú está siempre asociado con otro widget. El tipo más común de menú es la barra de menú. Una barra de menú es una barra horizontal presentada en la parte superior de una ventana. La barra de menú está siempre asociada con un widget de ventana.

Una barra de menú consiste de submenú. Un submenú es una lista vertical de comandos y opciones disponibles para un usuario. De este modo, la barra de menú es una colección de listas de comandos y opciones. Los títulos de los submenú son listados de un lado a otro de la barra de menú. Para un usuario, una barra de menú es una colección de menú desplegables. Para un programador en ABL, una barra de menú es una colección de submenú. Entonces, los términos submenú y menú desplegable son análogos, y, menú y barra de menú también lo son.

Un submenú consiste de ítems de menú. Un ítem de menú es un comando individual u opción, o incluso otro submenú. Cuando un submenú es un ítem de menú de otro submenú, se lo conoce como submenú anidado. Los submenú anidados aparecen al lado del submenú principal cuando se los escoge.

Cuando se ha completado una barra de menú, lo que se presenta al usuario es una estructura jerárquica que consiste de:

- Un widget de menú.
- Uno o más widgets de submenú.
- Uno o más widgets de ítems de menú por cada submenú.

Los widgets que componen una barra de menú completa están relacionados entre sí. Por ejemplo, los submenú son los hijos de la barra de menú. Cada uno de los submenú son hermanos entre sí. De la misma manera, los ítems de menú son hijos de un submenú padre y así sucesivamente. Cuando se programa una barra de menú completa, se la construye desde abajo hacia arriba. Los widgets de nivel inferior deben existir antes de poder relacionarlos con los widgets de nivel superior.

El widget de menú en sí no tiene un padre, ya que un menú siempre es propiedad de un widget. En el caso de la barra de menú, el widget de ventana lo posee. Cada ventana en ABL puede poseer una barra de menú.

La siguiente figura muestra una barra de menú con dos títulos de menú, “File” y “Edit”. Cuando se elige “Edit”, un menú desplegable con tres ítems de menú aparece debajo del mismo. Cuando se elige el ítem de menú “Add”, un menú desplegable con cuatro ítems de menú aparece al lado de éste. La flecha a la derecha de “Add” indica que hay un submenú anidado.

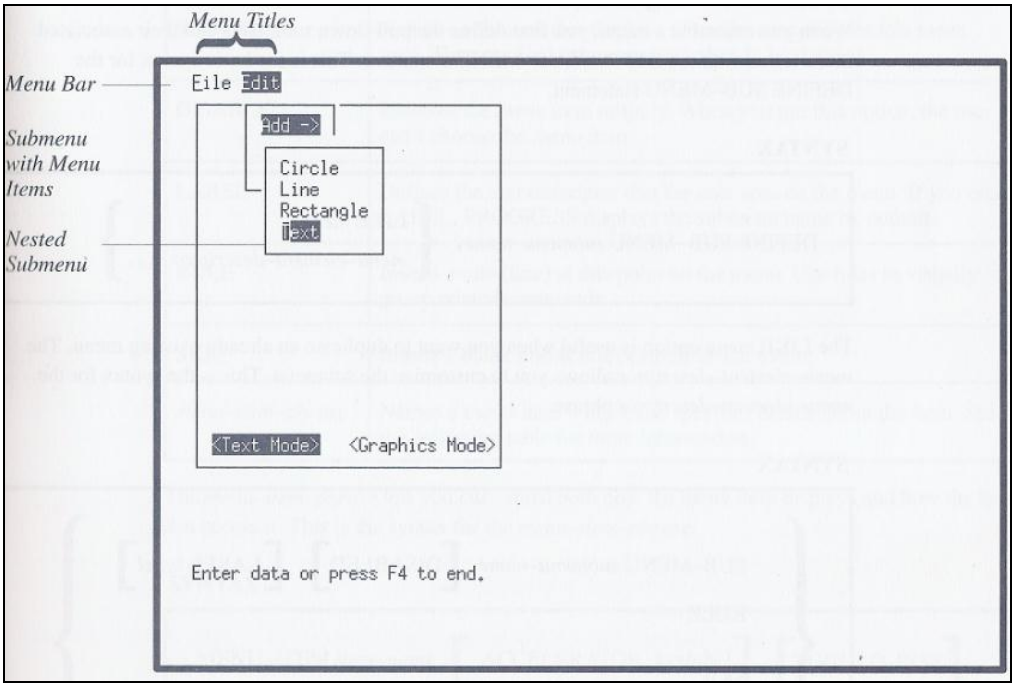

Figura 3: Barra de menú
Fuente: Progress Language

2. 1. 4. 2 Módulos de código

Cuando se aborda una aplicación, la primera inclinación podría ser escribir un número de procedimientos individuales potencialmente largos, donde cada uno realice una completa y quizá compleja tarea. Se puede ahorrar tiempo y hacer la aplicación más fácil de mantener mediante el uso de código modular. En ABL, se pueden utilizar diferentes tipos de procedimiento para crear módulos de código. El tipo de procedimiento que se use depende de las necesidades de la aplicación.

2. 1. 4. 2. 1 Procedimientos

Un procedimiento es un módulo funcional de código fuente escrito en ABL. Un archivo de procedimiento es un archivo de sistema operativo en formato de texto que contiene uno o más procedimientos. Por convención, estos archivos tienen una extensión de tipo “.p”.

Dado que un archivo de procedimiento puede contener muchos procedimientos, se podría ajustar una aplicación entera en uno solo. Sin embargo, este archivo extenso rápidamente se volvería difícil de leer y mantener. En su lugar, se pueden crear muchos archivos de procedimiento, cada uno con uno o más procedimientos. Uno de estos archivos contiene el procedimiento principal o inicial, el cual llama a otros procedimientos según sea necesario por la aplicación. Los procedimientos llamados por el procedimiento principal se denominan subprocedimientos.

Los subprocedimientos permiten:

- Minimizar el tiempo de desarrollo de la aplicación: No se tiene que volver a escribir una secuencia de instrucciones cada vez que se las necesite, sino se llama al subprocedimiento tantas veces como se quiera.

- Lograr un consistente comportamiento de la aplicación: Las acciones realizadas por un conjunto de sentencias se comportan de manera predecible cuando son llamadas por o incluidas dentro de otro procedimiento.

- Mantener la aplicación fácilmente: Si se quiere cambiar el código, se puede cambiar el único subprocedimiento que lo contiene, en lugar de corregir el código en muchas áreas diferentes del procedimiento principal.

Para ejecutar un procedimiento desde otro procedimiento, se utiliza la sentencia “RUN”.

Para demostrar este proceso, supongamos que un procedimiento principal crea una interfaz que muestra cuatro botones. Cuando un usuario escoge un botón, una tarea distinta se ejecuta. Cada una de estas tareas o módulos se almacenan en un archivo de procedimiento separado. Entonces cada módulo se convierte en un subprocedimiento del procedimiento principal.

La siguiente figura ilustra este concepto.

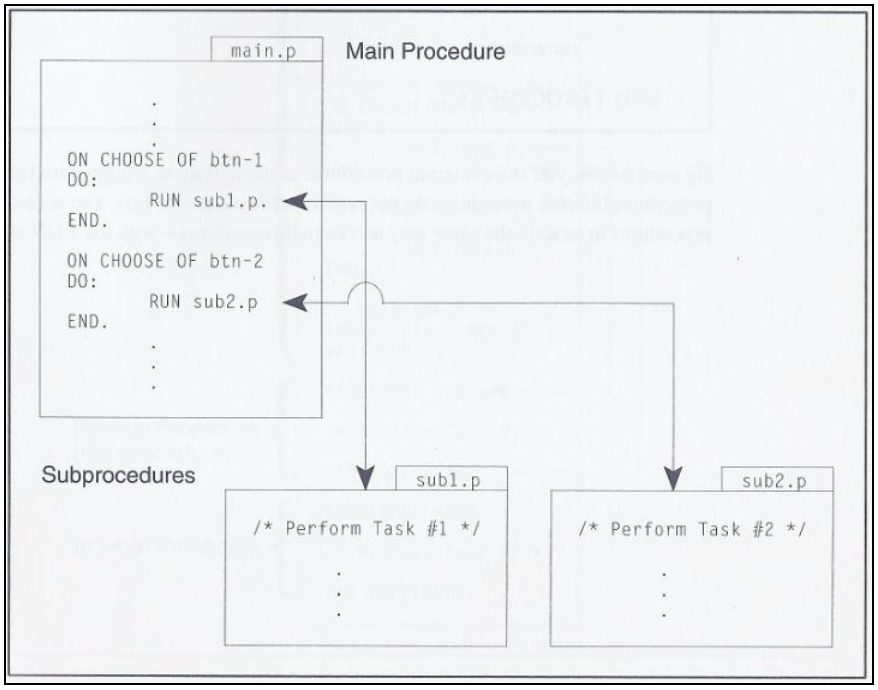

Figura 4: Procedimiento principal y subprocedimientos
Fuente: Progress Language

En el ejemplo mostrado en la figura anterior, cada archivo de procedimiento contiene un procedimiento. La sentencia "RUN" hace referencia al nombre del archivo para acceder al procedimiento que contiene. Cuando un subprocedimiento se almacena en un archivo de procedimiento separado, se lo denomina procedimiento externo.

ABL también permite almacenar subprocedimientos con el procedimiento principal en un archivo de procedimiento. Estos subprocedimientos se llaman procedimientos internos. Se pueden tener muchos procedimientos internos en un archivo de procedimiento.

Por convención, los procedimientos internos se almacenan en la parte inferior de un archivo de procedimiento, después del procedimiento principal. Los procedimientos internos no requieren una extensión de archivo (".p"). Estos procedimientos se acceden exactamente de la misma manera que los externos, es decir, con la sentencia "RUN".

La siguiente figura muestra un ejemplo simple de un archivo de procedimiento que contiene dos procedimientos internos.

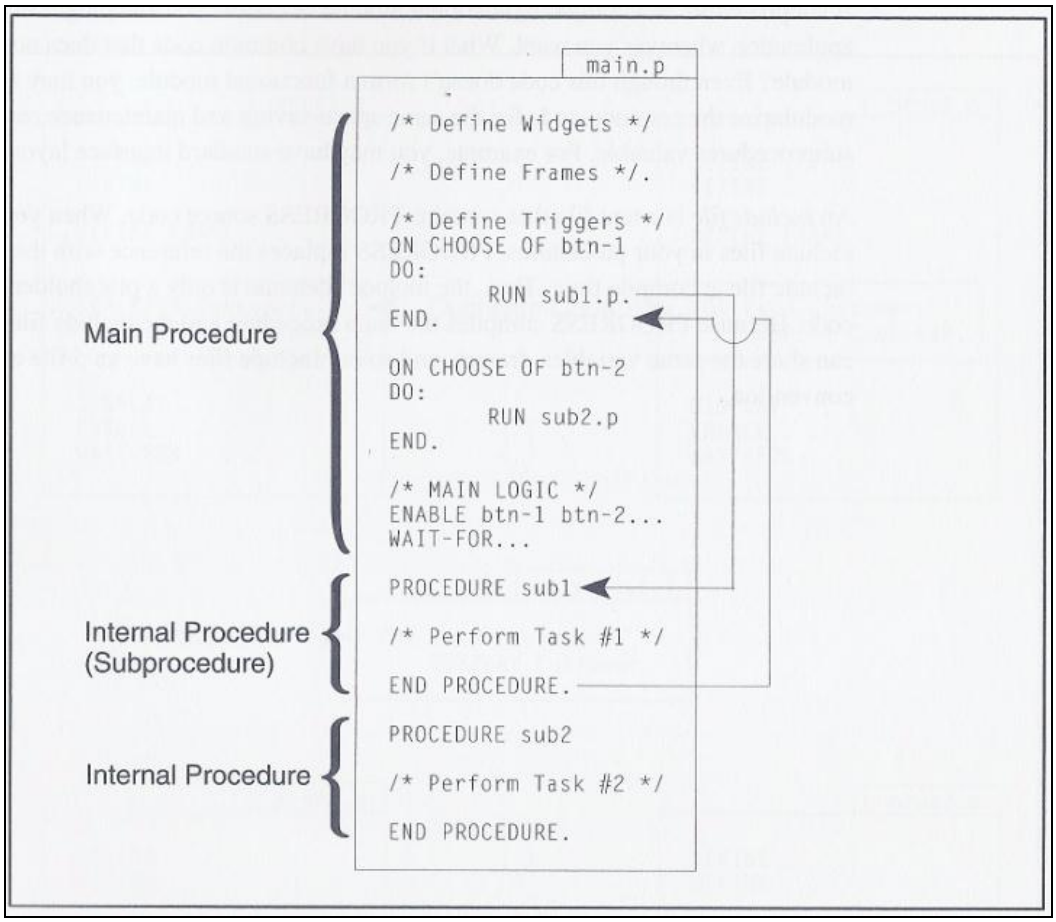

Figura 5: Procedimiento principal y procedimientos internos
Fuente: Progress Language

Todos los procedimientos definen recursos como variables, widgets, frames y triggers. Por defecto, el procedimiento principal no puede utilizar los recursos definidos en un subprocedimiento interno o externo. Tampoco un procedimiento externo puede utilizar los recursos definidos en el procedimiento principal. En cambio, un procedimiento interno puede utilizar los recursos definidos en el procedimiento principal. Esta característica es la principal ventaja que los procedimientos internos tienen sobre los procedimientos externos.

2. 1. 4. 2. 2 Archivos de inclusión

Un subprocedimiento es un módulo funcional y completo de código. Se lo puede conectar a una aplicación más grande donde sea posible. Pero si se tiene un código común que no constituye un módulo completo, es posible que se lo desee modularizar para el mismo ahorro de espacio y las razones de mantenimiento que hacen valioso al subprocedimiento. Por ejemplo, se pueden tener diseños de interfaz estándar.

Un archivo de inclusión es un archivo de texto que contiene código fuente de ABL. Cuando se referencian archivos de inclusión en los procedimientos, ABL reemplaza la referencia con el contenido del archivo en tiempo de compilación. Así, el nombre del archivo de inclusión es sólo un marcador de posición para el código que lo referencia. Debido a que ABL compila en conjunto al procedimiento principal y al archivo de inclusión, estos pueden compartir las mismas variables, frames, etc. Los archivos de inclusión tienen una extensión de tipo ".i" por convención.

Para referenciar a un archivo de inclusión, se agrega el nombre del archivo con su extensión al procedimiento y se lo cierra entre llaves.

La siguiente figura ilustra cómo trabajan los archivos de inclusión.

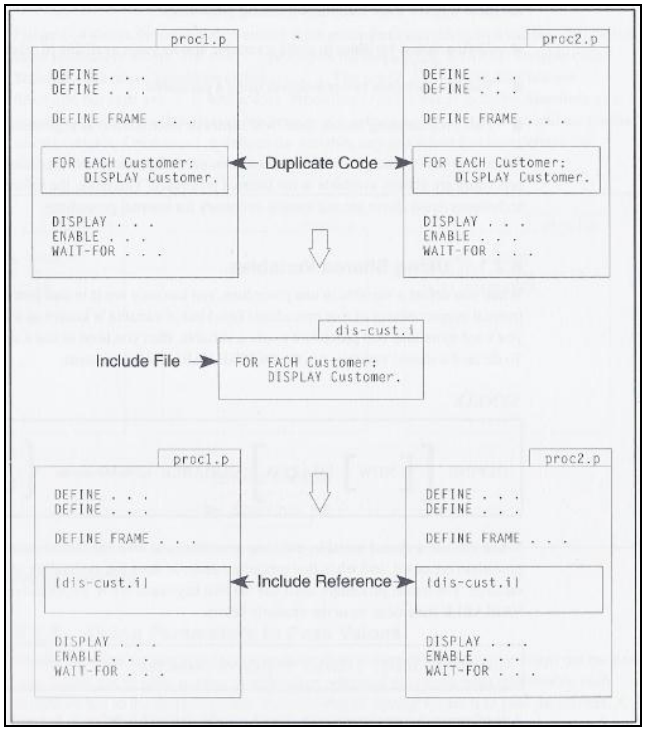

Figura 6: Usando un archivo de inclusión
Fuente: Progress Language

2. 1. 4. 3 Acceso a bases de datos

Para manipular datos con las sentencias de ABL, es útil conocer las principales ubicaciones donde éste almacena datos. Una vez que se entiende dónde están los datos, es más fácil visualizar cómo las sentencias del lenguaje cambian o mueven esos datos.

Para comenzar a trabajar con datos, primero se los tiene que copiar desde la base de datos. La base de datos es la primera ubicación de datos.

Cuando un procedimiento copia datos desde una base de datos, ABL los almacena en memoria. Un área de memoria utilizada para almacenar datos se denomina buffer. Cuando se solicita algún dato de un registro de la base de datos, ABL coloca una copia de dicho registro en un buffer. Estos buffers se denominan buffer de registro.

Para cada tabla con la que se trabaje, ABL crea y mantiene un buffer de registro separado. En general, no se tendrá que trabajar directamente con los buffers de registro. ABL entiende qué hacer con los buffers basados en el código.

Antes que los usuarios finales puedan ver o trabajar con datos, esos datos tienen que ser visibles en la pantalla. Los datos visibles en pantalla son también almacenados en memoria. La memoria reservada para los datos que aparecen en pantalla se denomina buffer de pantalla. Al igual que con los buffers de registro, ABL mantiene los buffers de pantalla por el programador.

Estas ubicaciones de datos son importantes de entender porque se refieren a tres reglas fundamentales del movimiento de datos en ABL. Éstas son:

1. Un procedimiento no puede interactuar con datos hasta que los copie de la base de datos a un buffer de registro. Una vez en el buffer de registro, el procedimiento puede manipular esos datos.

2. Los usuarios finales no pueden interactuar con datos hasta que un procedimiento los copie del buffer de registro al buffer de pantalla. Una vez en el buffer de pantalla, los datos son visibles en pantalla.

3. ABL no copia automáticamente los cambios hechos en un buffer de pantalla o un buffer de registro al buffer relacionado. Por ejemplo, si una copia de un campo particular existe en ambos buffers, cambiando los datos en el buffer de pantalla no cambia los datos en el buffer de registro. Un procedimiento programado controla el movimiento de datos entre buffers.

Pensemos en las ubicaciones de datos como paradas a lo largo de una calle de dos vías. Para mostrar un registro de una base de datos, ABL primero debe moverlo a un buffer de registro y luego a un buffer de pantalla. Del mismo modo, ABL no puede escribir nuevos datos del buffer de pantalla a una base de datos sin primero escribirlos a un buffer de registro. La siguiente figura muestra cómo se mueven los datos en ABL.

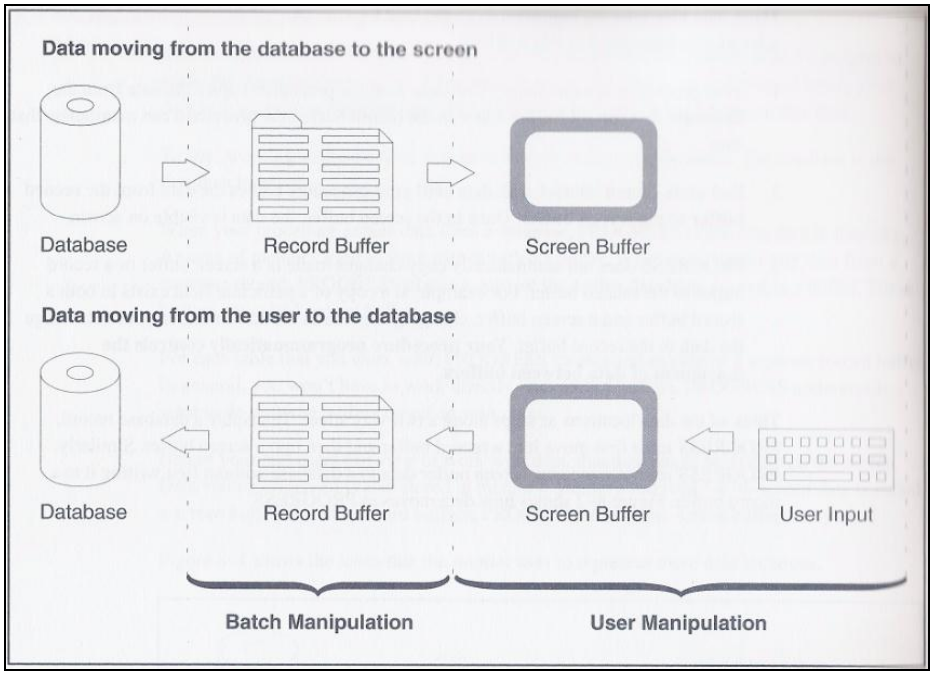

Figura 7: Movimiento de datos en ABL
Fuente: Progress Language

En la figura se observa la diferencia entre la manipulación por lote (batch manipulation) y la de usuario (user manipulation). Si se escribe un procedimiento que procese registros en segundo plano, entonces el procedimiento trabaja exclusivamente con buffers de registro. Si se escribe un procedimiento que permita a los usuarios interactuar con los datos, el procedimiento además trabaja con buffers de pantalla. En algún momento, este procedimiento interactivo copia datos entre los buffers de pantalla y los de registro.

2. 2 UML

2. 2. 1 Conceptos básicos

UML es un lenguaje gráfico estándar que permite visualizar, especificar, construir y documentar "planos" de software, incluyendo aspectos conceptuales tales como procesos de negocio y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes reutilizables.

Para comprender mejor cuál es el significado de UML, basta con analizar cada una de las palabras que lo componen, por separado.

- Lenguaje: UML es, precisamente, un lenguaje. Lo que implica que éste cuenta con una sintaxis y una semántica. Por lo tanto, al modelar un concepto en UML, existen reglas sobre cómo deben agruparse los elementos del lenguaje y el significado de esta agrupación.

- Unificado: Unifica varias técnicas de modelado en una única técnica.

- Modelado: UML es visual. Mediante su sintaxis se modelan distintos aspectos del mundo real, que permiten una mejor interpretación y entendimiento de éste.

2. 2. 2 Un modelo unificado y estándar

UML fue creado para unificar tres métodos diferentes de modelado de software, cada uno de los cuales tenía sus puntos fuertes: el Método Booch (Booch Method) de Grady Booch, la Técnica de Modelado de Objetos (OMT, de Object Modeling Technique) de James Rumbaugh y la Ingeniería de Software Orientada a Objetos (OOSE, de Object-Oriented Software Engineering) de Ivar Jacobson.

En el año 1994 los tres precursores de UML dieron forma a la primera versión del lenguaje, y en el año 1997 comenzó a contar con el respaldo del consorcio OMG (Object Management Group), fecha en la que fue lanzada la versión 1.1. Desde entonces, UML atravesó varias revisiones y refinamientos hasta llegar a la versión actual, la 2.2. En el año 2005 recibió la aprobación de la ISO (International Organization for Standardization) para convertirse en estándar.

Desde su introducción y amplia aceptación, ha crecido enormemente en alcance y capacidad. Si bien su principal uso está dirigido al diseño y desarrollo orientado a objetos de nuevas aplicaciones, y a la ingeniería inversa de aplicaciones existentes orientadas a objetos, hay un pequeño pero creciente cuerpo de trabajo y literatura sobre su uso en antiguas aplicaciones no orientadas a objetos, cuyo pionero en este campo es el Dr. Thomas Mercer-Hursh (VP Technology de Computing Integrity, Inc.).

2. 2. 3 Propuesta de modelado con UML para aplicaciones desarrolladas con ABL

Para poder modelar con UML aquellas antiguas aplicaciones de negocio, aún en funcionamiento, que hayan sido desarrolladas con ABL, ya sea sólo mediante procedimientos como también en combinación con clases, el Dr. Thomas Mercer-Hursh propone hacer ingeniería inversa sobre este tipo de aplicaciones, utilizando tres categorías de modelos de UML:

- Modelo de Interfases de Usuario: Puede utilizarse para representar una barra de menú y las unidades funcionales asociadas.

- Modelo de Componentes: Puede utilizarse para representar unidades de programa (archivos con extensión ".p" (procedimientos externos), ".w" (procedimientos externos de construcción de ventanas), y ".cls" (clases), más los procedimientos internos y funciones), unidades de inclusión (archivos con extensión ".i") y unidades de compilación (archivos con extensión ".r").

- Modelo de Datos: Puede utilizarse para representar todos aquellos aspectos de las bases de datos de OpenEdge, como tablas, columnas, índices, triggers, etc., junto con las propiedades de esos elementos.

Para llevar adelante esta propuesta de modelado, se requiere del uso de un perfil de UML.

2. 2. 4 Perfil de UML

UML introdujo a partir de su segunda versión el concepto de “perfil”, el cual es un mecanismo de extensión del lenguaje que permite proporcionar un vocabulario adecuado para facilitar la construcción de modelos en un dominio particular.

Un perfil de UML consiste de estereotipos, valores etiquetados y restricciones, que se aplican a elementos, atributos, métodos, vínculos y más, con el propósito de describir conjuntamente un problema de modelado en particular.

2. 2. 5 Ejemplo simple de estereotipo

Para introducirnos brevemente en el contenido de un perfil de UML, consideremos como ejemplo el estereotipo “oeProgram”, el cual se usaría para representar procedimientos externos de ABL. Este estereotipo es asignado a un elemento del lenguaje llamado “componente”. Éstos pueden tener asociados valores etiquetados, de los cuales “oeProgram” puede tener los tres siguientes:

- PathName: La ruta completa que identifica el archivo fuente desde el cual se ha extraído la información.

- HasUI: Una bandera que indica si esta unidad de programa tiene alguna declaración de interfaz de usuario.

- HasDA: Una bandera que indica si esta unidad de programa tiene alguna declaración de acceso a base de datos.

2. 2. 6 El Perfil de UML para ABL

Generalmente, los perfiles de UML no son definidos para un lenguaje de programación en particular, como es el caso de aquellos que estén basados en el paradigma de programación orientado a objetos, ya que tienden a ser representados de una manera bastante sencilla por UML. Pero en el caso de ABL, especialmente sus primeras versiones, solo se puede lograr tal representación si es a través del uso de un perfil. Es por ello que, el Dr. Thomas Mercer-Hursh propuso la creación de un perfil de UML para que sea adoptado como estándar en el modelado de aplicaciones desarrolladas con ABL. Este perfil está dividido en cinco secciones, cada una de las cuales se describe brevemente a continuación.

2. 2. 6. 1 Esquema de base de datos de OpenEdge

Se cubre gran parte de las propiedades de este tipo de estructura de datos, incluyendo el diseño físico de sus bases de datos, los índices y la estructura lógica de claves foráneas, aunque todavía no se ha implementado todo aquello relacionado al servidor de datos.

En la siguiente figura se ilustra un elemento de modelado llamado “clase” con estereotipo “table” (estereotipo estándar de UML), que representa una tabla típica de una base de datos de OpenEdge, que incluye su nombre, los nombres y atributos (tipos de datos y valores iniciales) de columnas, clave primaria e índices.

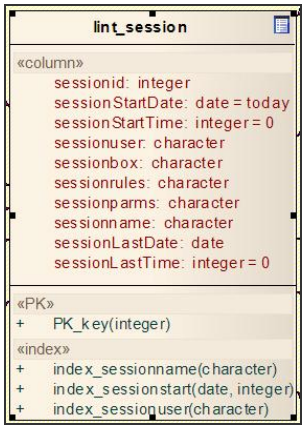

Figura 8: Tabla de base de datos de OpenEdge con UML
Fuente: Modeling Existing ABL Systems with UML

2. 2. 6. 2 Código fuente de ABL

Se hace hincapié en aquellas unidades estructurales de código fuente desde y hacia las cuales fluye el control de datos. Por lo tanto, hay estereotipos para archivos de procedimiento, archivos de inclusión, clases, procedimientos internos, funciones, métodos y unidades de compilación.

En el análisis para la construcción de un modelo, los procedimientos internos y las funciones pueden ser clasificados como públicos o privados, según si se los referencia desde fuera de una unidad de compilación o no.

Todas aquellas nuevas definiciones compartidas y referencias a variables compartidas también son incluidas en el perfil.

En la siguiente figura se ilustra el elemento de modelado “componente” con el estereotipo “oeProgram” ejemplificado anteriormente, que representa un archivo de procedimiento típico de ABL, el cual incluye su nombre y las referencias a variables compartidas.

Figura 9: Archivo de procedimiento de ABL con UML
Fuente: Modeling Existing ABL Systems with UML

2. 2. 6. 3 Enlaces entre unidades de código fuente

Los estereotipos para las conexiones de control de flujo entre componentes de código cubren todos los tipos de relaciones posibles, incluyendo declaraciones simples de ejecución de procedimientos externos o procedimientos internos, declaración de nuevas clases, invocación de métodos, llamada de funciones y adición de superprocedimientos locales o de sesión. Para todos estos tipos de relaciones, por un lado existe un vínculo detallado entre las unidades actuales de código en las cuales el flujo comienza y termina, y por otro lado existe un vínculo resumido entre los procedimientos externos o clases adjuntos. Un segundo vínculo resumido es creado entre las unidades de compilación. Esto facilita la diagramación en múltiples niveles.

En la siguiente figura se ilustra la sección de un Diagrama de Componentes de UML, donde se representan las relaciones entre procedimientos externos, procedimientos internos y funciones.

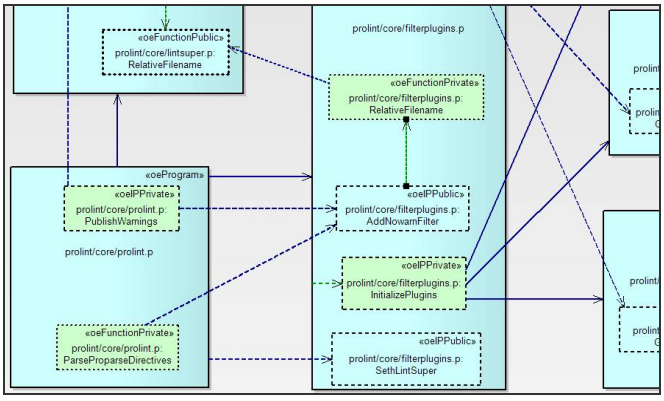

Figura 10: Enlaces entre unidades de código fuente de ABL con UML
Fuente: Modeling Existing ABL Systems with UML

2. 2. 6. 4 Enlaces entre unidades de código y datos

Las conexiones entre código y datos deben establecerse en medio de la unidad más chica que contenga código fuente, como por ejemplo procedimientos externos, procedimientos internos, clases, funciones o métodos, y deben resumirse a nivel de unidad de programa y unidad de compilación.

Los estereotipos para estas relaciones distinguen entre los que son de solo lectura y aquellos donde ocurre una operación de creación, eliminación o actualización de datos, donde este tipo de operación se proporciona en una etiqueta.

La cláusula WHERE de cualquier consulta SQL (Structured Query Language) también es capturada, por lo cual cada campo que sea leído o escrito explícitamente debe ser anotado en una etiqueta junto con la acción que ocurre.

La siguiente figura ilustra un Diagrama Simple de UML, donde se observa una tabla en el centro y el acceso a sus datos desde procedimientos internos y unidades de programa.

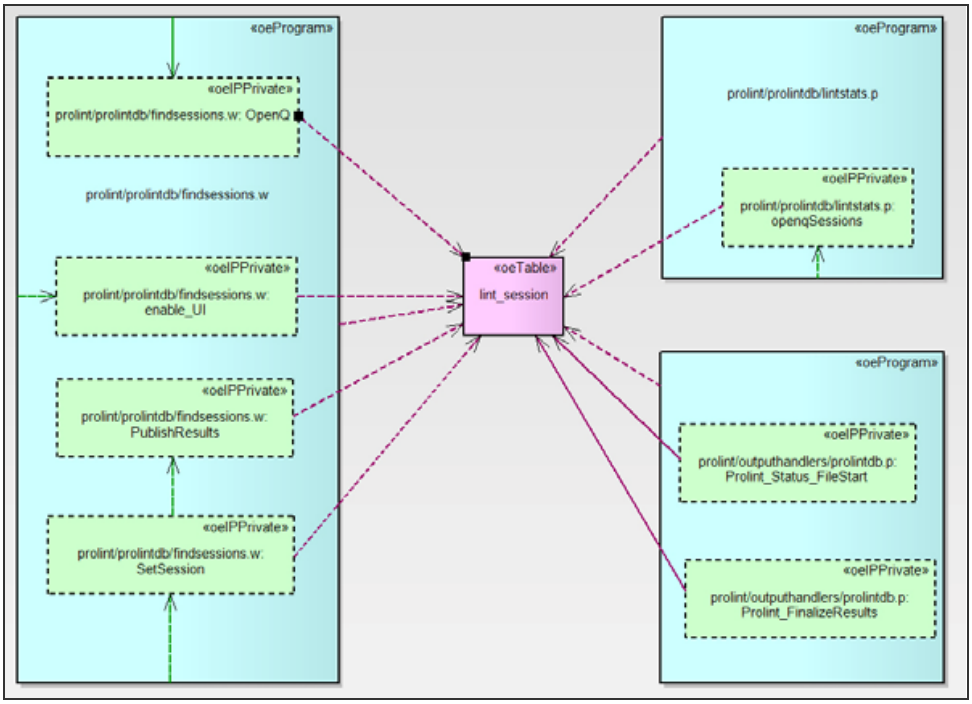

Figura 11: Enlaces entre unidades de código y datos con UML
Fuente: Modeling Existing ABL Systems with UML

2. 2. 6. 5 Estructura lógica

La estructura lógica hace referencia a la estructura de menú de este tipo de aplicaciones, la cual proporciona acceso a funciones individuales del sistema. Para modelar esta estructura existe un conjunto general de estereotipos que permite representar el modo de uso de estas aplicaciones.

Toda selección de menú que cause la ejecución de algo más que un procedimiento de menú debe ser vinculado a una unidad funcional. Esta unidad contiene todo aquél código que podría alcanzarse durante su ejecución. A su vez, esa unidad funcional está conectada a una unidad de compilación, siendo ésta el primer procedimiento que ejecuta.

En la siguiente figura se ilustra el fragmento de un sistema de menú con las selecciones correspondientes que conducen a unidades funcionales.

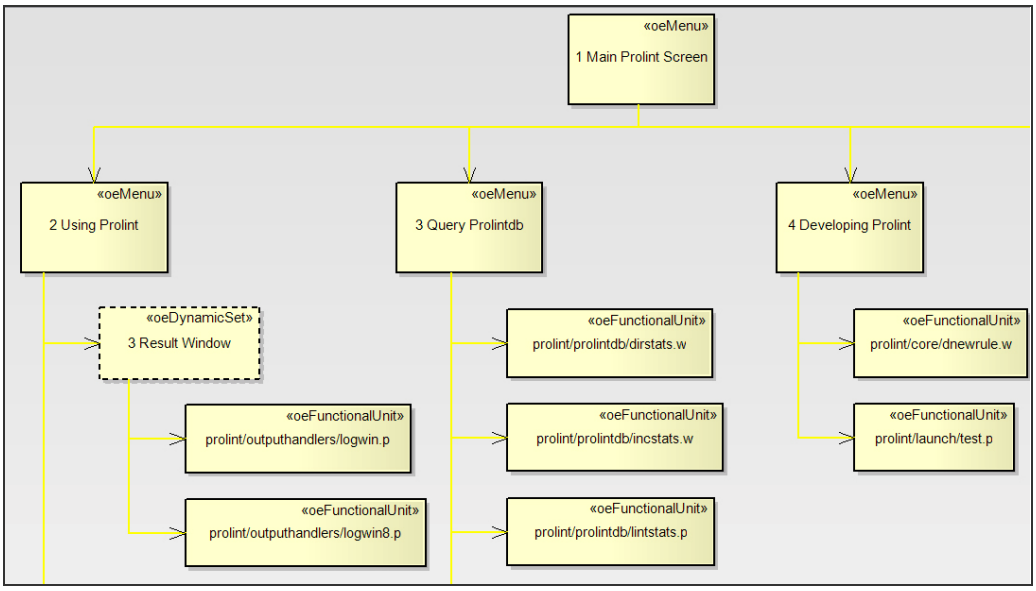

Figura 12: Estructura lógica de una barra de menú de ABL con UML
Fuente: Modeling Existing ABL Systems with UML

CAPÍTULO III:

MARCO DE MODELADO

3. 1 Introducción

La aplicación de negocio desarrollada con ABL que se ha considerado como caso práctico experimental en el ámbito de la presente tesina, denominada Gestión de Expedientes, si bien por un lado no puede ser modelada directamente con UML, por otro lado tampoco lo puede ser haciendo uso del perfil de UML para ABL que se expuso en el capítulo anterior. Por lo que, en este capítulo se presenta el marco de referencia que se ha desarrollado con Enterprise Architect para demostrar el modelado de dicha aplicación. A éste se lo ha denominado Marco de Modelado (en adelante MM).

3. 2 Componentes de UML

Los principales componentes de UML que constituyen el MM son los siguientes:

- Modelos: Un modelo es la abstracción de un sistema físico con un cierto propósito.
- Vistas: Una vista es la proyección de un modelo desde una perspectiva dada.
- Elementos: Un elemento es un objeto de modelado de cualquier tipo (clase, componente, nodo, objeto, etc.).
- Diagramas: Un diagrama es la presentación gráfica de una colección de elementos de modelado.
- Conectores: Un conector es un vínculo lógico entre elementos de modelado.
- Estereotipos: Un estereotipo es un nuevo tipo de elemento de modelado que extiende la semántica de la notación UML original, pero no su estructura.

3. 3 Modelos y Vistas

El MM se divide en tres modelos, cada uno de los cuales se divide a su vez en vistas.

En la siguiente figura se ilustran los modelos y vistas del MM.

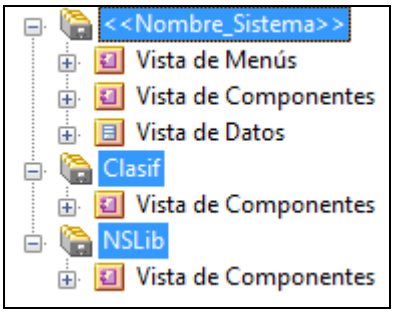

Figura 13: Modelos y vistas del marco de modelado

Cada uno de los modelos representa los siguientes componentes de la aplicación:

- Modelo principal: Representa la estructura principal de la aplicación desde tres perspectivas diferentes y asociadas a la vez:
    - Barra de menú.
    - Archivos locales de procedimiento e inclusión.
    - Tablas de bases de datos.
- Modelo "NSLib": Representa aquellos procedimientos de una librería de funciones denominada NSLib, que la aplicación debe utilizar para realizar sus tareas de administración de datos.

- Modelo “Clasif”: Representa aquellos procedimientos utilizados por la aplicación que se encuentran contenidos en un directorio denominado “clasif”.

A partir de aquí, en las próximas secciones se describen en detalle cada una de las vistas que forman parte de los modelos recién descriptos.

3. 4 Modelo principal

El modelo principal está formado por las siguientes vistas.

3. 4. 1 Vista de Menú

Representa todo el contenido de la barra de menú de opciones de la aplicación, incluyendo además las referencias a las unidades funcionales que conforman la Vista de Componentes, tanto del modelo actual como de los modelos NSLib y Clasif.

Los elementos de modelado que se utilizan en esta vista son:

- Paquetes: Representan los submenú o menú desplegables (en adelante menú principales) y los submenú anidados (en adelante submenú).

- Componentes: Representan la barra de menú y los ítems de menú.

Estos elementos se almacenan lógicamente dentro de la siguiente estructura de paquetes:

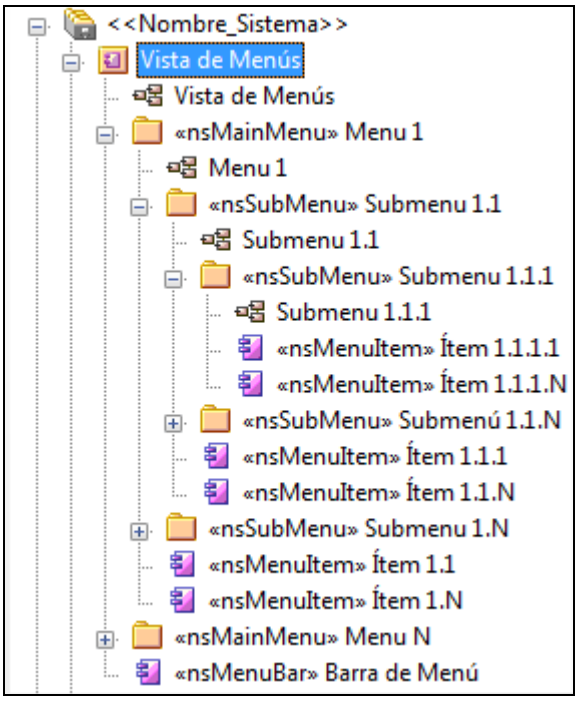

Figura 14: Organización de elementos en la Vista de Menús

- Los menú principales (en adelante menú) y la barra de menú se almacenan dentro del paquete “Vista de Menú”.

- Los submenú se almacenan dentro de los menú y otros submenú.

- Los ítems de menú se almacenan dentro de los menú y submenú.

El paquete “Vista de Menú” junto con cada uno de sus menú y submenú contiene un Diagrama de Componentes de igual nombre que su contenedor, donde se representan gráficamente sus propios elementos y las referencias a elementos externos. Todos estos elementos se relacionan entre sí mediante el conector de Dependencia, de la siguiente manera:

- La barra de menú se relaciona con los menú y una unidad funcional.

- Un menú se relaciona con uno o más submenú y uno o más ítems de menú.

- Un submenú se relaciona con uno o más submenú, uno o más ítems de menú y una unidad funcional.

- Un ítem de menú se relaciona con una unidad funcional.

Los estereotipos que se aplican a los elementos y el conector de modelado son los siguientes:

Para Paquetes:

- nsMainMenu: Se aplica a los menú.

- nsSubMenu: Se aplica a los submenú.

Para Componentes:

- nsMenuBar: Se aplica a la barra de menú.

- nsMenuItem: Se aplica a los ítems de menú.

Para Dependencias:

- nsLinkFunctionalUnit: Se aplica a la relación entre una unidad funcional y la barra de menú y entre una unidad funcional y un submenú.

- nsLinkMenuBar: Se aplica a la relación entre la barra de menú y un menú.

- nsLinkMainMenu: Se aplica a la relación entre un menú y un submenú y entre un menú y un ítem de menú.

- nsLinkSubMenu: Se aplica a la relación entre un submenú y un submenú y entre un submenú y un ítem de menú.

- nsLinkMenuItem: Se aplica a la relación entre un ítem de menú y una unidad funcional.

Todos los elementos que forman parte de esta vista se denominan igual que las opciones que representan de la barra de menú de la aplicación.

A modo de ejemplo, en las siguientes figuras se ilustran los diagramas de "Vista de Menú", un menú y un submenú.

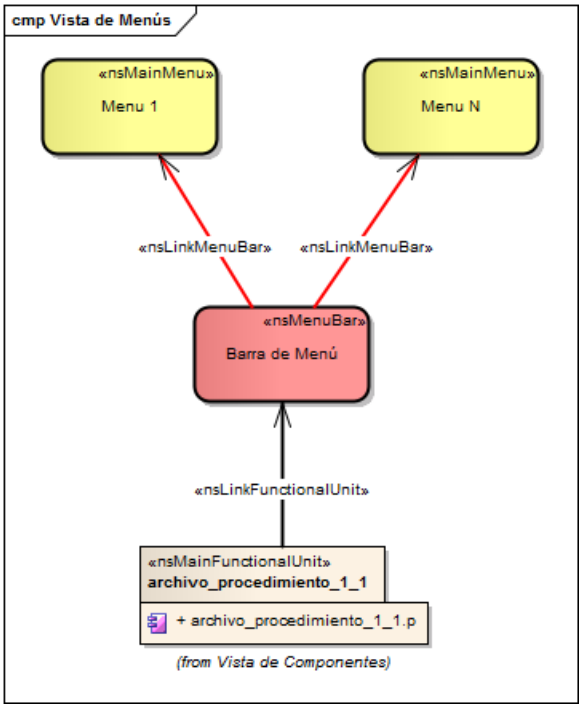

Figura 15: Diagrama principal de la Vista de Menú

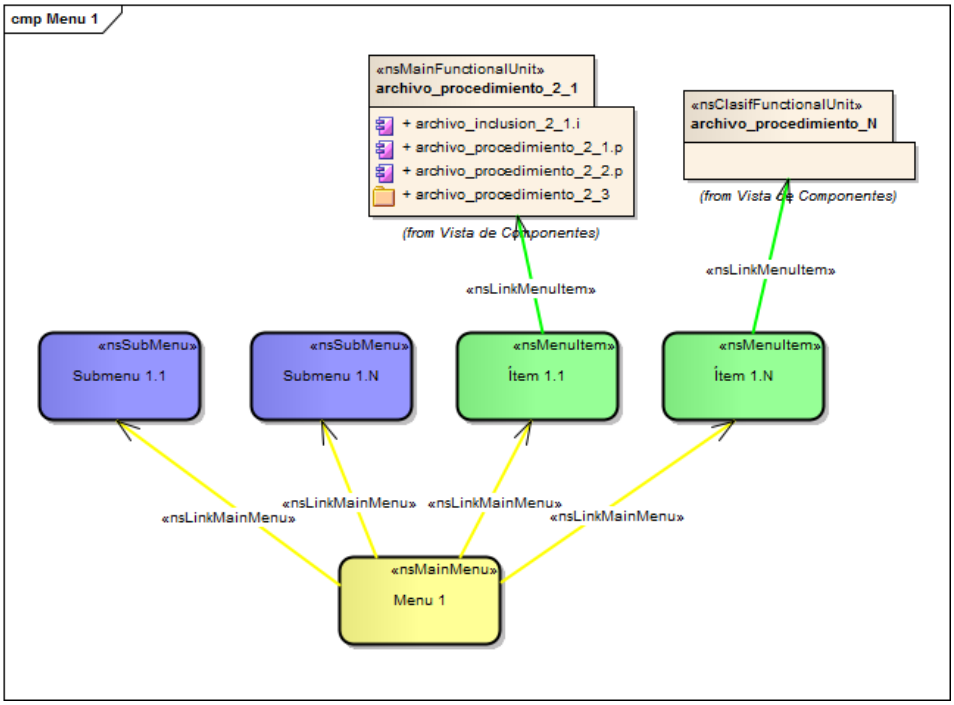

Figura 16: Diagrama de un menú principal

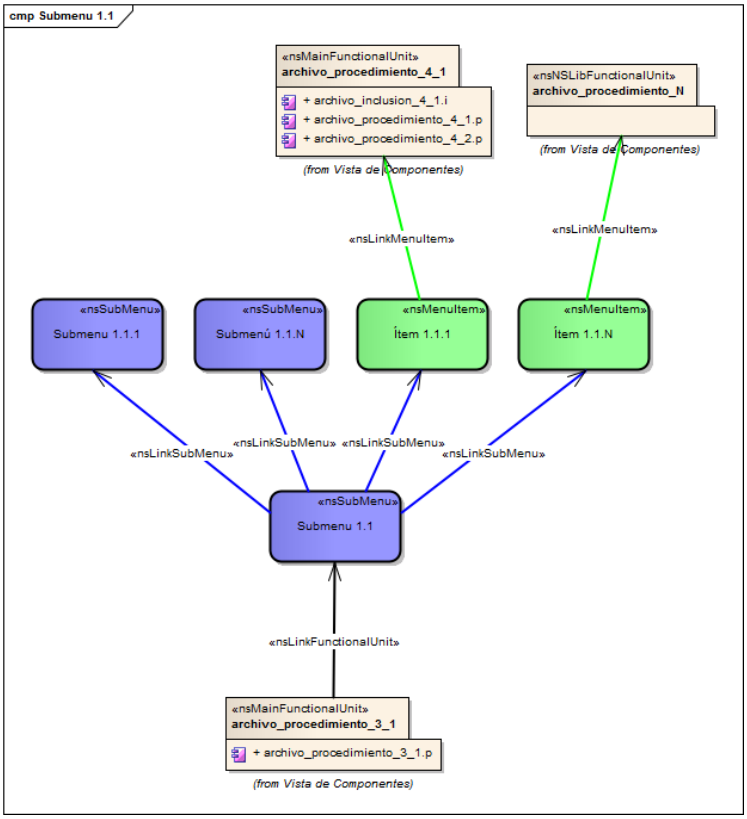

Figura 17: Diagrama de un submenú

3. 4. 2 Vista de Componentes

Representa los componentes que forman parte de la estructura de cada unidad funcional, incluyendo además las referencias a archivos de procedimiento y/o inclusión de los modelos NSLib y Clasif, y las referencias a tablas de la Vista de Datos.

Una unidad funcional cumple con algunas de las siguientes características:

- Si se invoca desde alguno de los elementos de la Vista de Menú o desde más de una unidad funcional, se denomina unidad funcional principal.

- Si se invoca desde una unidad funcional principal se denomina subunidad funcional, al igual que si se invoca desde otra subunidad.

- Su nombre hace referencia al nombre de aquel archivo de procedimiento que primero se ejecuta tras su invocación.

- Utiliza un procedimiento de la librería para la administración de datos.

A diferencia de lo que es una unidad funcional en sí misma, un componente es aquel archivo individual de procedimiento o inclusión que conforma su estructura.

Los elementos de modelado que se utilizan en esta vista son:

- Paquetes: Representan las unidades funcionales principales y las subunidades funcionales.

- Componentes: Representan los archivos de procedimiento y los archivos de inclusión.

Estos elementos se almacenan lógicamente dentro de la siguiente estructura de paquetes:

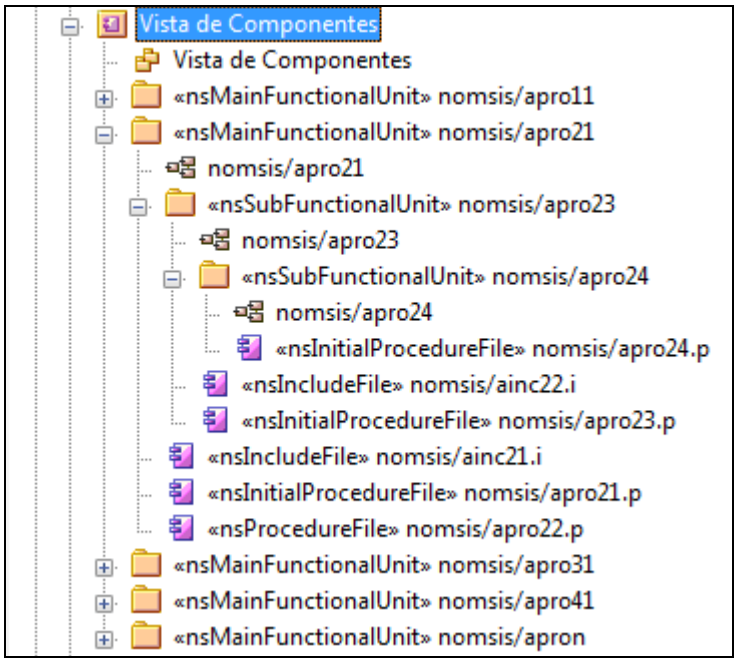

Figura 18: Organización de elementos en la Vista de Componentes

- Las unidades funcionales principales se almacenan dentro del paquete "Vista de Componentes".

- Las subunidades funcionales se almacenan dentro de las unidades funcionales principales y otras subunidades.

- Los archivos de procedimiento e inclusión se almacenan dentro de las unidades y subunidades funcionales.

El paquete "Vista de Componentes" contiene un Diagrama de Paquetes de igual nombre que su contenedor, donde se representan gráficamente todas las unidades funcionales principales.

La siguiente figura ilustra este diagrama.

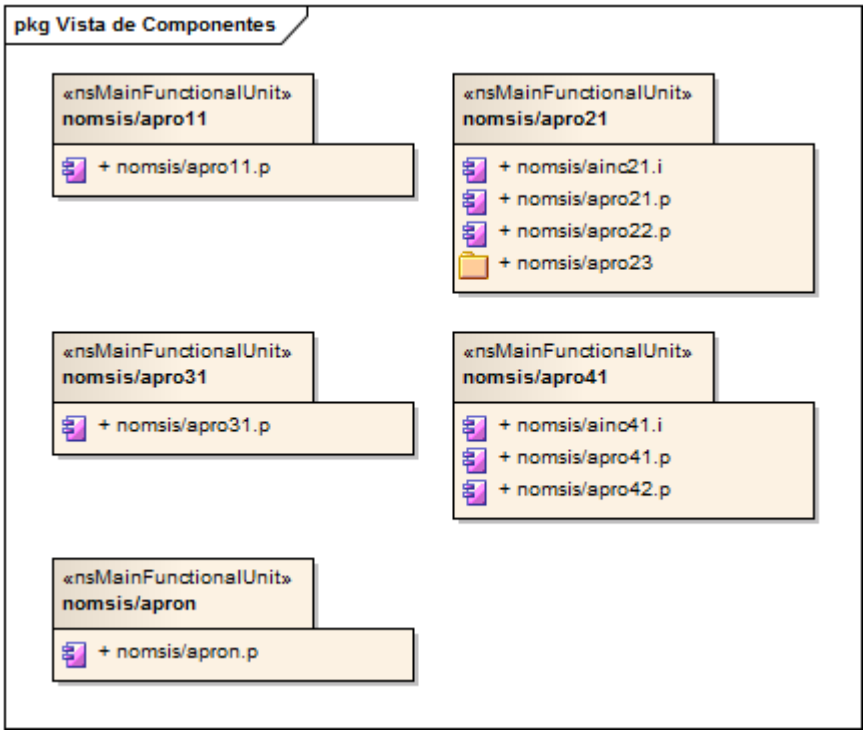

Figura 19: Diagrama principal de la Vista de Componentes

Todas las unidades funcionales contienen un Diagrama de Componentes de igual nombre que su contenedor, donde se representan gráficamente sus propios elementos y las referencias a elementos externos. Todos estos elementos se relacionan entre sí mediante el conector de Dependencia, de la siguiente manera:

- Un archivo de procedimiento y/o uno de inclusión de una unidad funcional principal o una subunidad funcional se relaciona con uno o más archivos de procedimiento y/o inclusión de esa misma unidad funcional, uno o más archivos procedimiento y/o inclusión de los modelos NSLib y Clasif, una o más unidades funcionales principales, una o más subunidades funcionales y/o una o más tablas.

- Se ocultan todas aquellas relaciones entre elementos que hayan sido definidas en otras unidades funcionales.

- No se ocultan las relaciones entre tablas definidas en la Vista de Datos.

Los estereotipos que se aplican a los elementos y el conector de modelado son los siguientes:

Para Paquetes:

- nsMainFunctionalUnit: Se aplica a las unidades funcionales principales.

- nsSubFunctionalUnit: Se aplica a las subunidades funcionales.

Para Componentes:

- nsInitialProcedureFile: Se aplica a aquellos archivos de procedimiento que primero se ejecutan en las unidades funcionales.

- nsProcedureFile: Se aplica a los archivos de procedimiento.

- nsIncludeFile: Se aplica a los archivos de inclusión.

Para Dependencias:

- nsLinkRun: Se aplica a la ejecución de un archivo de procedimiento desde uno de inclusión u otro de procedimiento.

- nsLinkInclude: Se aplica cuando se referencie un archivo de inclusión en uno de procedimiento u otro de inclusión.

- nsLinkReadData: Se aplica a la lectura de datos de una tabla desde un archivo de procedimiento o uno de inclusión.

- nsLinkWriteData: Se aplica a la escritura de datos en una tabla desde un archivo de procedimiento o uno de inclusión.

Todos los elementos que forman parte de esta vista se denominan igual que los nombres de los archivos de procedimiento e inclusión que representan, precedidos por el nombre del directorio que los contiene y una barra invertida hacia la derecha (/). Si el elemento es una unidad funcional, no se coloca el tipo de extensión de archivo al que hace referencia (.p). En cambio si es un archivo individual, sí se coloca su extensión (.p o .i).

A modo de ejemplo, en las siguientes figuras se ilustran los diagramas de una unidad funcional principal y una subunidad funcional.

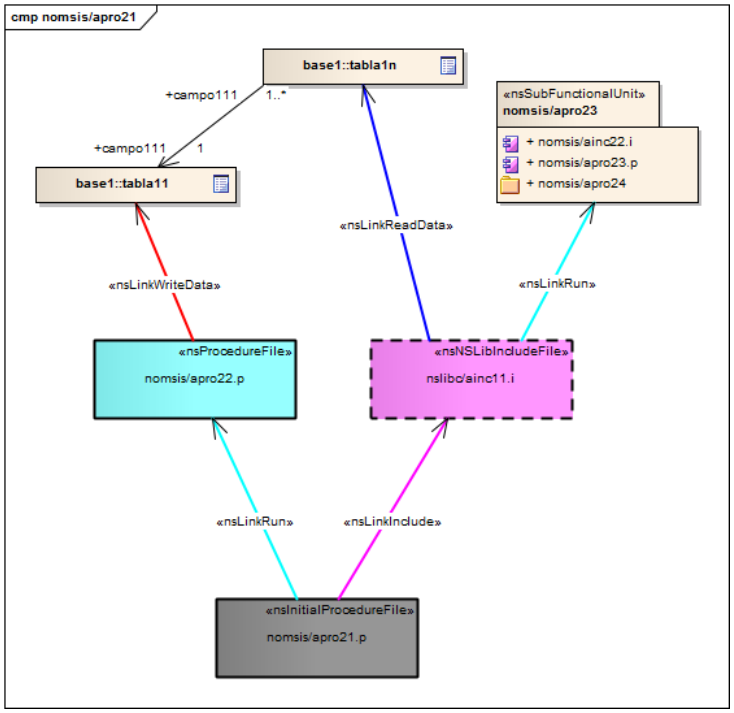

Figura 20: Diagrama de una unidad funcional principal

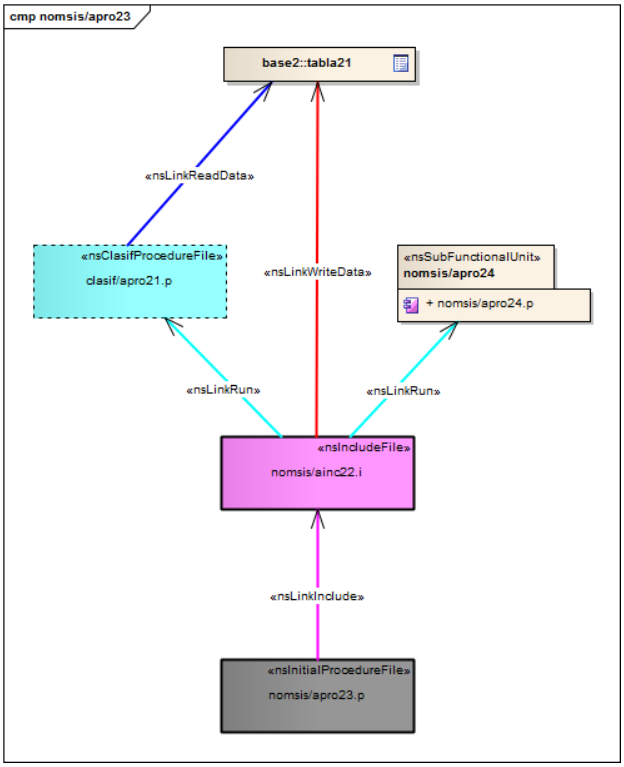

Figura 21: Diagrama de una subunidad funcional

3. 4. 3 Vista de Datos

Representa todas aquellas tablas de bases de datos que se utilizan en la aplicación y las relaciones establecidas entre sí.

Los elementos de modelado que se utilizan en esta vista son:

- Paquetes: Representan las bases de datos.

- Clases: Representan las tablas.

Estos elementos se almacenan lógicamente dentro de la siguiente estructura de paquetes:

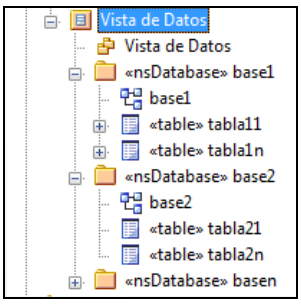

Figura 22: Organización de elementos en la Vista de Datos

- Las bases de datos se almacenan dentro del paquete "Vista de Datos".

- Las tablas se almacenan dentro de las bases de datos a las que pertenecen.

El paquete "Vista de Datos" contiene un Diagrama de Paquetes de igual nombre que su contenedor, donde se representan gráficamente todas las bases de datos que se utilizan.

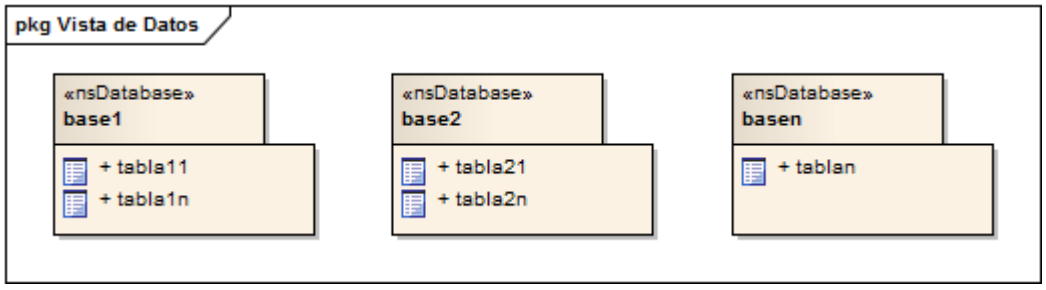

Figura 23: Diagrama principal de la Vista de Datos

Todas las bases de datos contienen un Diagrama de Clases de igual nombre que su contenedor, donde se representan gráficamente sus propias tablas y las referencias a tablas externas. Todas estas tablas se relacionan entre sí mediante el conector de Asociación, de la siguiente manera:

- Una tabla de una base de datos se relaciona o no con una o más tablas de esa misma base de datos y/o una o más tablas de otras bases de datos, a través de una o más referencias a campos de datos externos.

Los estereotipos que se aplican a los elementos de modelado son los siguientes:

Para Paquetes:

- nsDatabase: Se aplica a las bases de datos.

Para Clases:

- table (estereotipo estándar): Se aplica a las tablas.
- nsColumn: Se aplica a las columnas o campos de datos.
- nsPk: Se aplica a las claves primarias.
- nsIndex: Se aplica a los índices.

Todos los elementos que forman parte de esta vista se denominan igual que las bases de datos y tablas que representan. Además en el caso de las tablas, las propiedades de sus campos, claves e índices coinciden con las definidas físicamente.

A modo de ejemplo, en la siguiente figura se ilustra el diagrama de una base de datos.

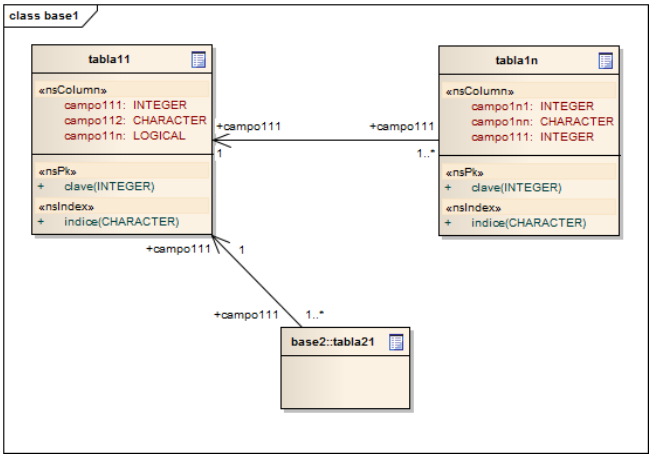

Figura 24: Diagrama de una base de datos

3. 5 Modelos NSLib y Clasif

Cada uno de estos modelos está formado por la siguiente vista.

3. 5. 1 Vista de Componentes

Representa solamente aquellos archivos de procedimiento, archivos de inclusión y unidades funcionales que se reutilizan y referencian respectivamente en la Vista de Componentes y la Vista de Menú del modelo principal. No se tienen en cuenta las estructuras internas de las unidades funcionales como se hace con las del modelo principal.

Los elementos de modelado que se utilizan en esta vista son:

- Paquetes: Representan las unidades funcionales.

- Componentes: Representan los archivos de procedimiento y los archivos de inclusión.

Estos elementos se almacenan lógicamente dentro de la siguiente estructura de paquetes:

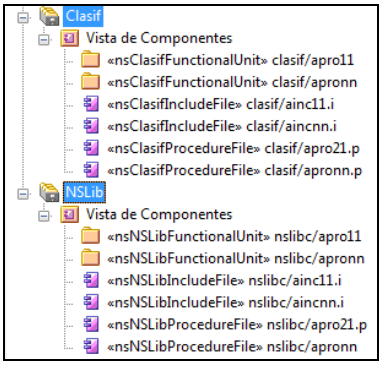

Figura 25: Organización de elementos en la Vista de Componentes de NSLib y Clasif

- Tanto las unidades funcionales como los archivos de procedimiento e inclusión de cada modelo se almacenan dentro del paquete “Vista de Componentes” correspondiente (no se organizan como la Vista de Componentes del modelo principal).

El paquete “Vista de Componentes” no contiene un diagrama para representar sus propios elementos. Menos aún las unidades funcionales.

Los estereotipos que se aplican a los elementos de NSLib son los siguientes:

Para Paquetes:

- nsNSLibFunctionalUnit: Se aplica a las unidades funcionales de la librería.

Para Componentes:

- nsNSLibProcedureFile: Se aplica a los archivos de procedimiento de la librería.

- nsNSLibIncludeFile: Se aplica a los archivos de inclusión de la librería.

Los estereotipos que se aplican a los elementos de Clasif son los siguientes:

Para Paquetes:

- nsClasifFunctionalUnit: Se aplica a las unidades funcionales del directorio “clasif”.

Para Componentes:

- nsClasifProcedureFile: Se aplica a los archivos de procedimiento del directorio “clasif”.

- nsClasifIncludeFile: Se aplica a los archivos de inclusión del directorio “clasif”.

Todos los elementos que forman parte de esta vista se denominan de la misma manera que aquellos que pertenecen a la Vista de Componentes del modelo principal, aunque hacen referencia a los directorios “nslibc” y “clasif” que los contienen respectivamente.

CAPITULO IV:

CASO EXPERIMENTAL

4. 1 Introducción

Con el objeto de demostrar la aplicación a un caso real del marco de modelado que se describió en el capítulo anterior, a continuación se presenta el modelado de la aplicación de negocio denominada Gestión de Expedientes (en adelante GE) para la cual éste se desarrolló.

La aplicación GE fue diseñada y desarrollada para satisfacer los requerimientos que se producen en el área de un organismo público, dedicado al manejo de expedientes, notas y resoluciones.

En el ámbito de la presente tesina, debido a la cantidad de funciones que ofrece la aplicación anterior, solo se van a representar aquellas funciones relacionadas con el manejo integral y exclusivo de expedientes internos de una organización.

En las próximas secciones de este capítulo, se describen cada una de las vistas del modelo principal aplicadas al caso experimental propuesto.

4. 2 Vista de Menú

Esta vista representa aquellas opciones de menú que permiten acceder a las funciones de GE.

La barra de menú de GE está formada por los siguientes menú principales:

1. Clasifica: Permite administrar cada una de las bases de información (también llamadas clasificadores o nomencladores) que sirven de sustento para realizar gran parte de las tareas que ofrece la aplicación.

2. Norma Legal: Provee las herramientas necesarias para administrar la información de todo tipo de expedientes, notas, resoluciones y reclamos.

3. Consultas: Brinda un conjunto de herramientas con las cuales se pueden generar una variada gama de reportes, basados en información suministrada en las opciones de la aplicación.

4. Utilitarios: Ofrece un conjunto de herramientas que permiten efectuar tareas de administración y mantenimiento de la aplicación.

5. Volantes: Permite confirmar los pases de expedientes y notas a distintas reparticiones u oficinas.

Los menú anteriores se representan en el diagrama principal del paquete “Vista de Menú” que se ilustra en la siguiente figura.

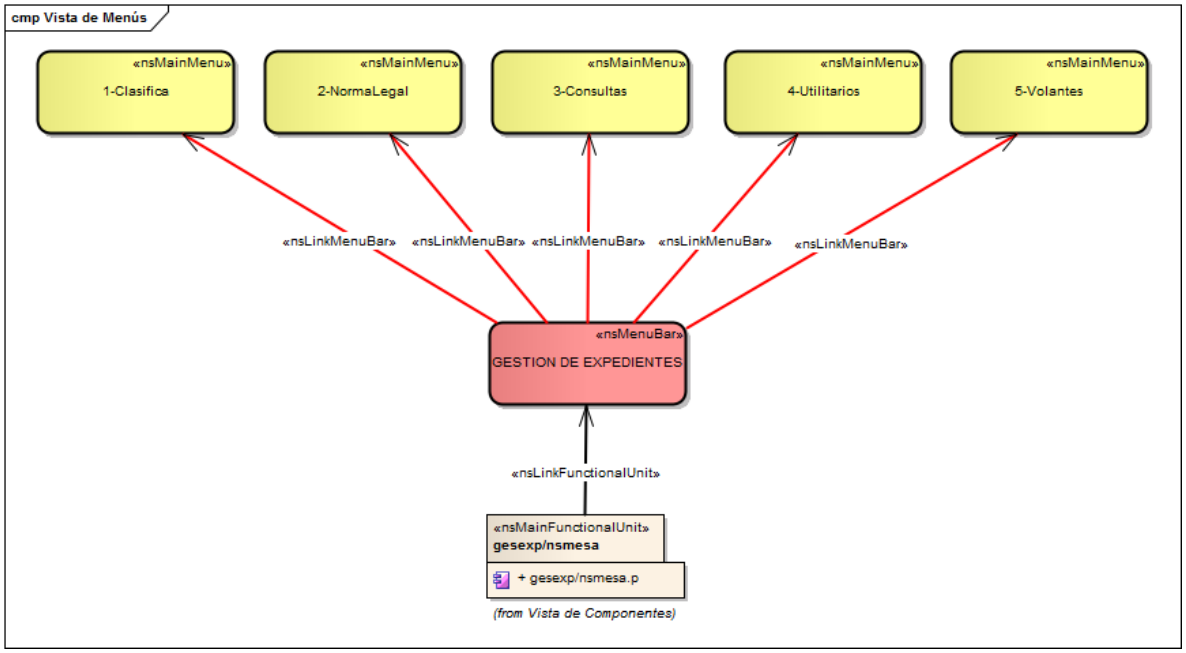

Figura 26: Diagrama de la Vista de Menú de Gestión de Expedientes

A partir de aquí, se comienza a focalizar la atención en aquel menú principal cuyo contenido se aproxima al ejemplo a tratar en cuestión. Para ello, se representa el menú “2-NormaLegal” que cuenta con las siguientes opciones:

a. Expedientes: Es un submenú que ofrece diferentes opciones para acceder a la administración de los datos de expedientes.

b. Notas: Es un ítem de menú que permite acceder a la administración de los datos de notas.

c. Resoluciones: Es otro ítem de menú que permite acceder a la administración de los datos de resoluciones.

d. Reclamos: Es otro submenú que ofrece diferentes opciones para acceder a la administración de los datos de reclamos.

En la siguiente figura se ilustra el diagrama correspondiente al menú anteriormente descripto.

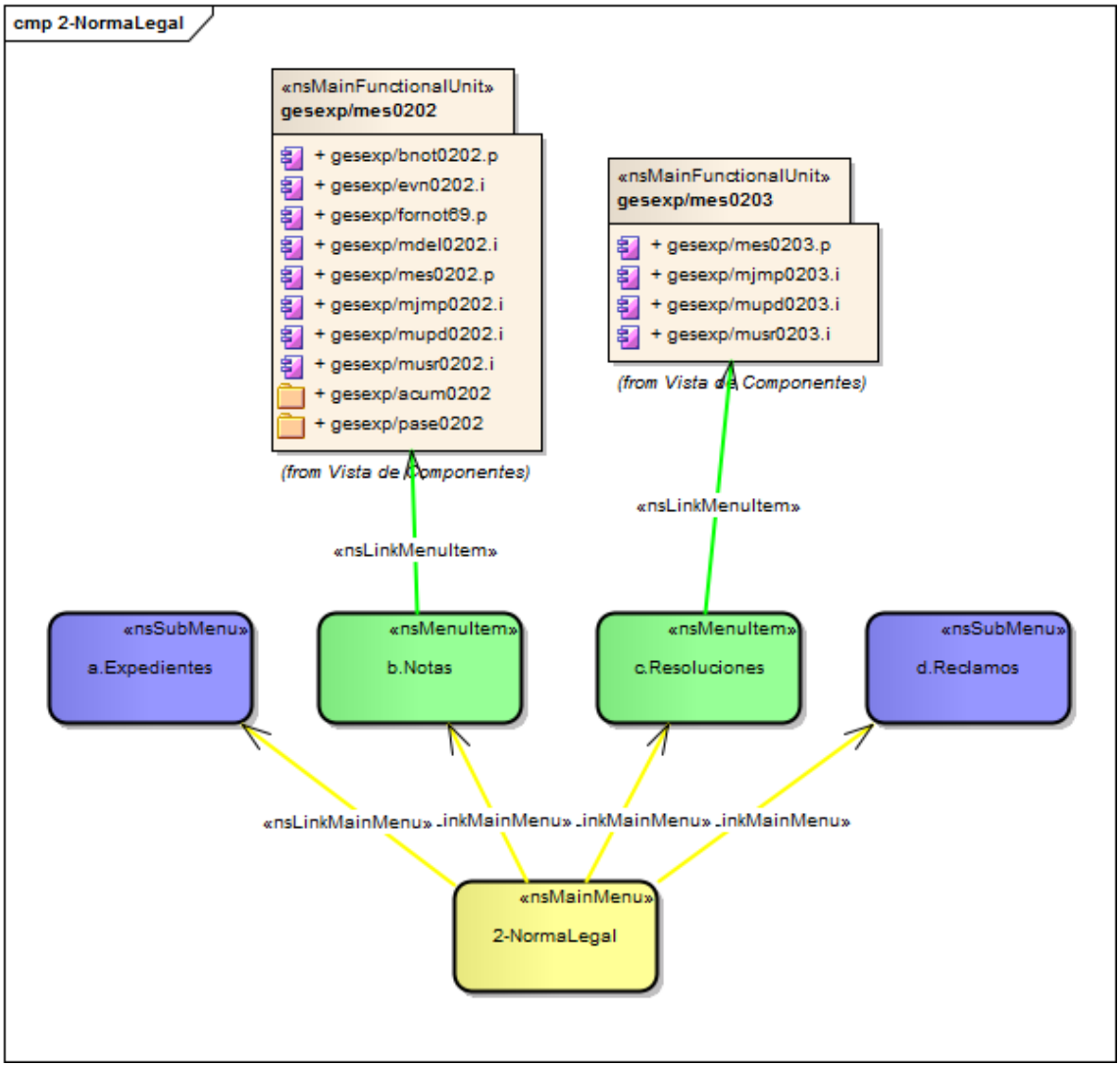

Figura 27: Diagrama del menú "2-NormaLegal"

Finalmente, para identificar dentro de la vista actual cuál es la opción de GE que se ha elegido como caso práctico de aplicación del marco de modelado, se representa el submenú "a.Expedientes" que cuenta con las siguientes opciones:

a. Expedientes Internos: Es el ítem de menú que permite acceder a la gestión de expedientes internos de una organización. Esta es la opción a la que se hace referencia y por ende la que se representa en la Vista de Componentes y la Vista de Datos respectivamente.

b. Expedientes Externos: Es aquel ítem de menú que permite acceder a la gestión de expedientes externos que provienen de otras organizaciones.

c. Carga manual: Es aquel otro ítem de menú que permite acceder a la gestión de archivos históricos de expedientes.

En la siguiente figura se ilustran las relaciones del submenú "a.Expedientes" con cada uno de los ítems de menú anteriores, y las relaciones entre éstos y las unidades funcionales correspondientes.

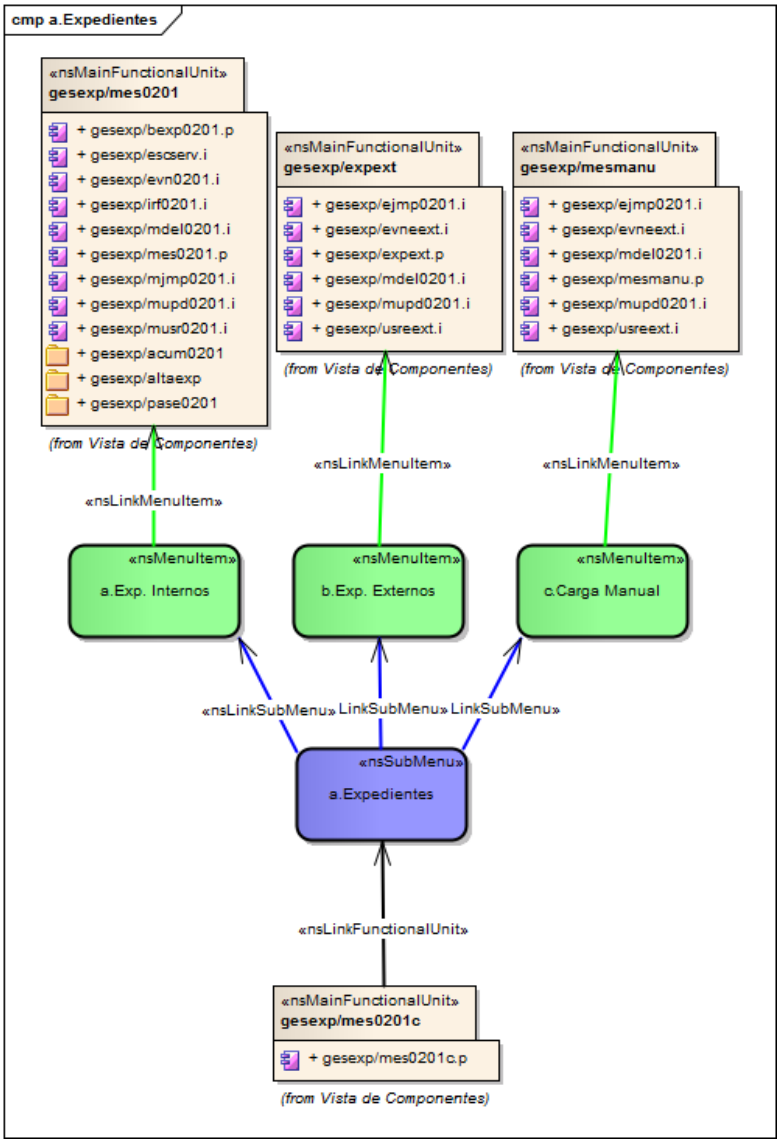

Figura 28: Diagrama del submenú "a.Expedientes"

Todos los elementos de modelado que se han utilizado en los diagramas anteriores, excepto las unidades funcionales, se almacenan dentro de la estructura de paquetes que se ilustra en la siguiente figura.

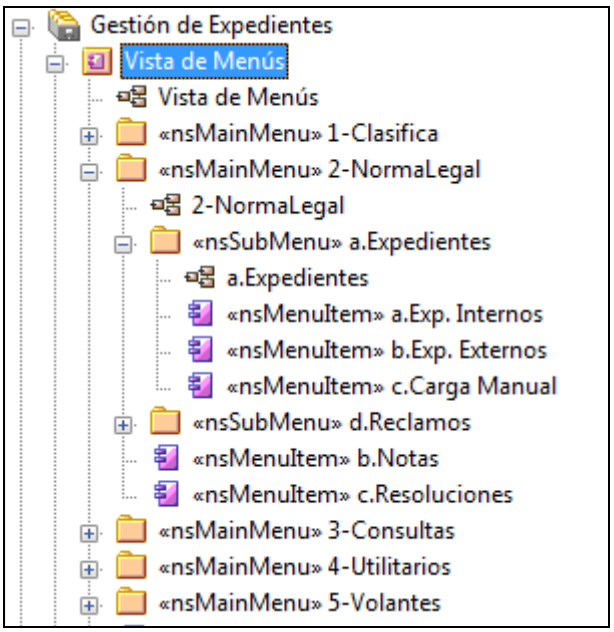

Figura 29: Organización de elementos en la Vista de Menú de Gestión de Expedientes

4. 3 Vista de Componentes

En esta otra vista se representan algunas de las unidades funcionales que se asocian al ejemplo que se analiza. Una de ellas es la unidad funcional principal denominada “gesexp/mes0201”. La otra es una de las subunidades funcionales que contiene la unidad funcional anterior, la cual se denomina “gesexp/altaexp”.

La unidad funcional principal “gesexp/mes0201” es aquella unidad funcional que se relaciona con el ítem de menú “a.Exp. Internos” de la figura 28. Su función es permitir que los usuarios lleven a cabo la gestión de expedientes internos (autonumerados) de una organización.

En la siguiente figura se ilustran algunos de los archivos de procedimiento, archivos de inclusión, tablas, unidades funcionales y relaciones definidas entre todos estos elementos que conforman la estructura de la unidad funcional principal.

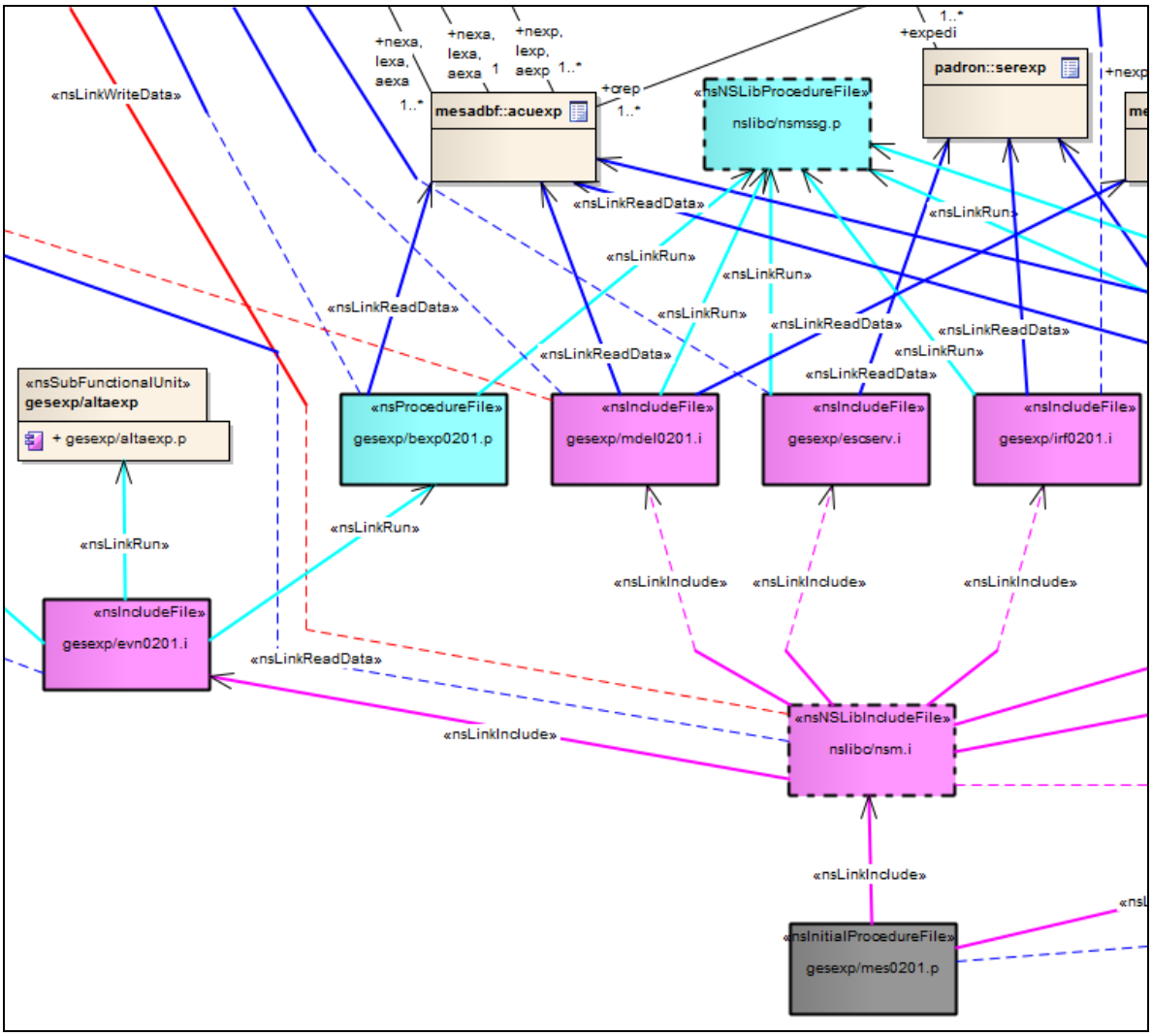

Figura 30: Recorte del diagrama de la unidad funcional "gesexp/mes0201"

La subunidad funcional "gesexp/altaexp" es aquella otra unidad funcional que se relaciona con el archivo de inclusión "gesexp/evn0201.i" de la figura anterior. Su función es permitir que los usuarios lleven a cabo la gestión de alta de expedientes internos autonumerados.

En la siguiente figura se ilustran todos aquellos elementos y sus relaciones que conforman la estructura de la subunidad funcional.

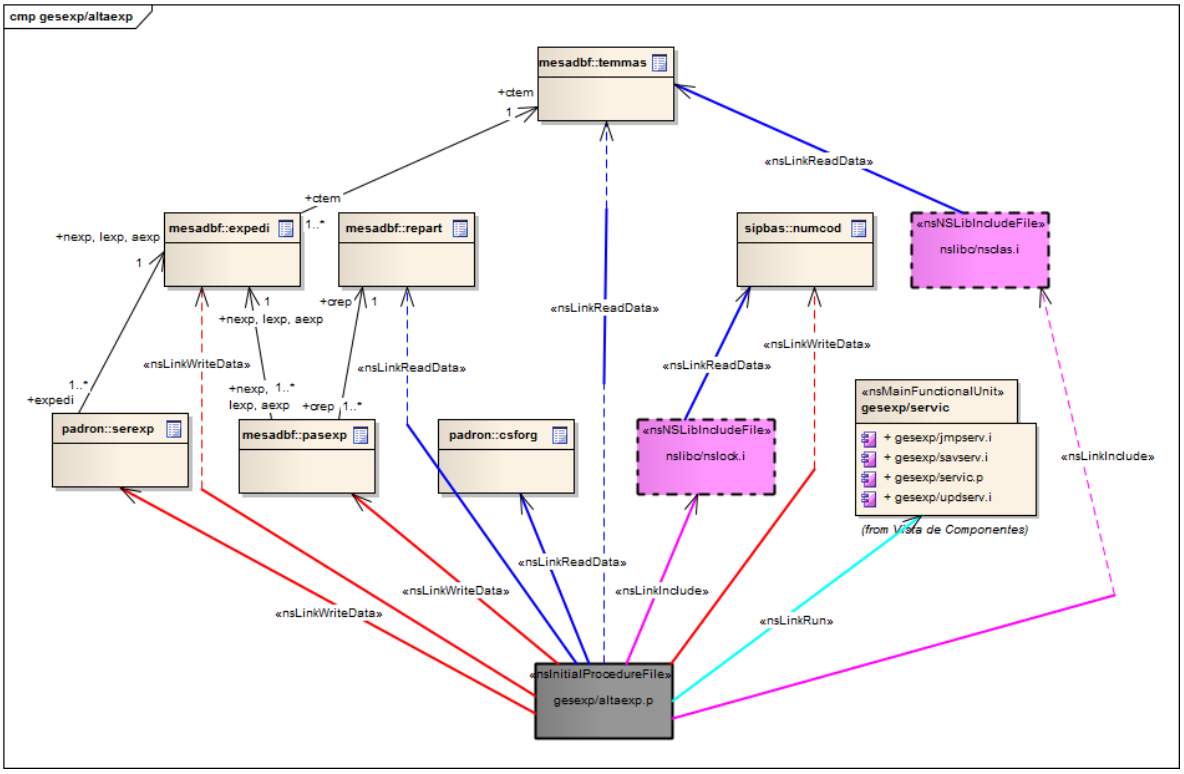

Figura 31: Diagrama de la subunidad funcional "gesexp/altaexp"

Todos los elementos de modelado que se han utilizado en los diagramas anteriores, excepto tablas, unidades funcionales principales y archivos de procedimiento e inclusión externos (de NSLib y/o Clasif), se almacenan dentro de la estructura de paquetes que se ilustra en la siguiente figura.

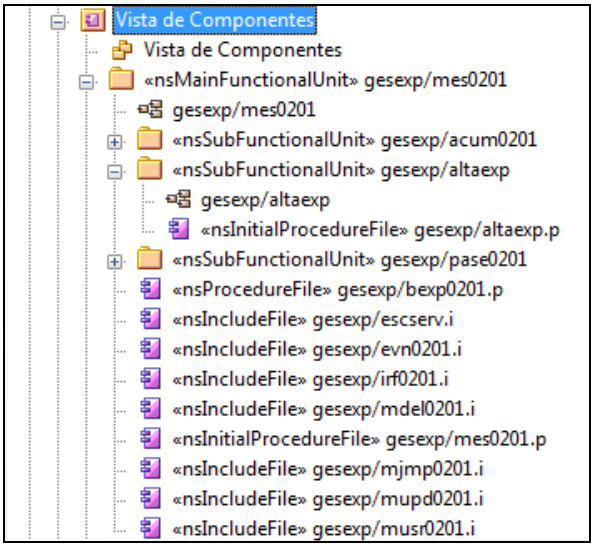

Figura 32: Organización de elementos en la Vista de Componentes de Gestión de Expedientes

4. 4 Vista de Datos

En esta última vista se representa solamente aquella base de datos cuyas tablas se utilizan en parte por las unidades funcionales anteriores. Dicha base de datos se denomina “mesadbf”.

En la siguiente figura se ilustran algunas de las relaciones que se han definido entre tablas de la misma base de datos, como también aquellas otras con tablas de otras bases.

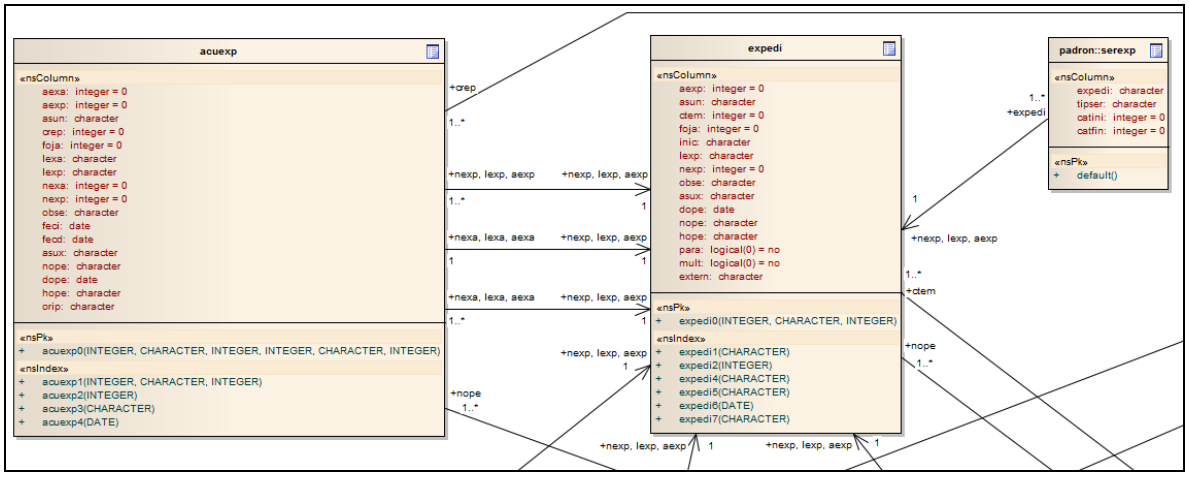

Figura 33: Recorte del diagrama de la base de datos “mesadbf”

Todas las tablas que se han utilizado en el diagrama anterior, excepto aquellas de otras bases de datos, se almacenan dentro de la estructura de paquetes que se ilustra en la siguiente figura.

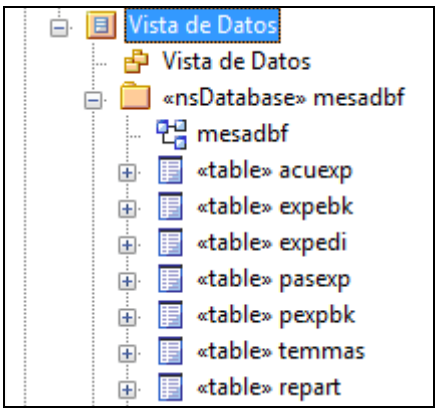

Figura 34: Organización de elementos en la Vista de Datos de Gestión de Expedientes

4. 5 Acerca de los resultados obtenidos

La aplicación experimental del marco de modelado a una de las funciones específicas de GE demostró ser una experiencia satisfactoria, debido a que permitió obtener representaciones gráficas de sus interfaces de usuario, módulos de código y acceso a bases de datos a través de la Vista de Menú, Vista de Componentes y Vista de Datos, respectivamente.

CAPITULO V:

CONCLUSIÓN

5. 1 CONCLUSIÓN

En este proyecto se abordó en forma clara y precisa el análisis conceptual de las interfases de usuario, módulos de código y acceso a bases de datos que caracterizan a las aplicaciones desarrolladas con ABL, y cómo éstas pueden ser modeladas con UML mediante el perfil de modelado desarrollado por el Dr. Thomas Mercer-Hursh.

Del análisis anterior surgió la necesidad de desarrollar un marco de modelado con UML, el cual permitiese modelar la aplicación de negocio considerada como ejemplo.

El conocimiento adquirido en el desarrollo de esta tesina ha demostrado que UML no solo permite modelar aplicaciones orientadas a objetos, sino también puede ser utilizado en el modelado de otros tipos de aplicaciones, gracias a su flexibilidad para extenderse a nuevos dominios.

Por lo tanto, dado los resultados positivos que se obtuvieron en el capítulo anterior, es muy importante aclarar que el marco de modelado desarrollado en el capítulo 3 no solo se aplica a Gestión de Expedientes, sino también a aquellas otras aplicaciones de negocio desarrolladas con ABL, las cuales en conjunto forman parte de un gran sistema informático de gestión pública denominado SIGA (Sistema Integral de Gestión Administrativa).

CAPITULO VI:

ANEXOS

6. 1 Glosario

3GL: Los Lenguajes de Programación de Tercera Generación son la gama de lenguajes de programación para ámbitos computacionales donde se logra un alto rendimiento con respecto a lenguajes de generaciones anteriores (primera y segunda generación). Ejemplos: C, Fortran, Smalltalk, Ada, C++, C#, Cobol, Delphi, Java, etc.

4GL: Los Lenguajes de Programación de Cuarta Generación son los lenguajes en los cuales en lugar de escribir cómo deben obtenerse los resultados, se especifica cuáles resultados son los que se quieren obtener. Por ejemplo, los lenguajes de consulta de bases de datos (como el SQL) son considerados lenguajes de cuarta generación.

ABL: Lenguaje de desarrollo de aplicaciones de negocio creado y mantenido por Progress Software Corporation. Forma parte de la familia de los 4GL.

Buffer: Área de memoria utilizada para almacenar datos.

CHUI: Interfaz de usuario basada en caracteres.

Enterprise Architect: Herramienta CASE (Computer Aided Software Engineering) desarrollada por Sparx Systems para el diseño y construcción de sistemas de software, para el modelado de procesos de negocios, y para objetivos de modelado más generalizados. Ofrece una plataforma de modelado basada en UML.

Estereotipo: Un nuevo tipo de elemento de modelado que extiende la semántica del metamodelo.

Frame: Área de presentación dentro de una ventana que ABL utiliza para mostrar widgets a nivel de campo.

GUI: Interfaz de usuario basada en gráficos.

Ingeniería inversa: Es un método que avanza en dirección opuesta a las tareas habituales de ingeniería, con el objetivo de obtener información a partir de un producto accesible al público, para determinar de qué está hecho, qué lo hace funcionar y cómo fue fabricado.

Lógica de negocio: Término no técnico generalmente utilizado para describir los algoritmos funcionales que se encargan del intercambio de información entre una base de datos y una interfaz de usuario.

Metamodelo: Un modelo que define el lenguaje para expresar un modelo, como por ejemplo el metamodelo estándar de UML.

OMG: El Grupo de Gestión de Objetos es un consorcio de compañías de software (HP, IBM, Apple, etc.) dedicado al cuidado y el establecimiento de diversos estándares de tecnologías orientadas a objetos, tales como UML, XMI, CORBA, entre otros.

OpenEdge: Plataforma de desarrollo de aplicaciones de negocio creada y mantenida por Progress Software.

Paquete: Un contenedor lógico de elementos de modelado. Agrupa elementos y también puede contener otros paquetes.

Perfil (de UML): Un perfil es un paquete estereotipado que contiene elementos de modelado que se personalizaron para un dominio o un propósito específicos, utilizando mecanismos de extensión tales como estereotipos, definiciones etiquetadas y restricciones.

Plataforma: En informática y tecnología, plataforma se refiere al sistema operativo o a sistemas complejos que a su vez sirven para crear programas, como las plataformas de desarrollo.

Portabilidad: Se define como la característica que posee un software para ejecutarse en diferentes plataformas.

Programación orientada a objetos: Es un paradigma de programación que utiliza objetos y sus interacciones, para diseñar aplicaciones y programas de ordenador.

Programación por procedimientos: Paradigma de programación que consiste en basarse de un número muy bajo de expresiones repetidas, englobarlas todas en un procedimiento o función y llamarlo cada vez que tenga que ejecutarse.

Progress Software Corporation: Compañía de software responsable de la plataforma de desarrollo OpenEdge y el lenguaje ABL, entre otros productos.

RDBMS: Un Sistema de Administración de Bases de Datos Relacionales es una colección de hardware y software que organiza y proporciona acceso a bases de datos relacionales.

Software: Es el conjunto de los programas de cómputo, procedimientos, reglas, documentación y datos asociados que forman parte de las operaciones de un sistema de computación.

SQL: El Lenguaje de Consulta Estructurado es un lenguaje formal declarativo, estandarizado por ISO, para manipular información en una base de datos.

Trigger: Procedimiento que se ejecuta cuando se cumple una condición establecida al realizar una operación de inserción, actualización, o eliminación de registros en una base de datos.

UML: El Lenguaje Unificado de Modelado es un lenguaje gráfico estándar respaldado por el OMG para visualizar, especificar, construir y documentar "planos" (modelos) de software, incluyendo aspectos conceptuales tales como procesos de negocio y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes reutilizables. Actualmente es el lenguaje de modelado de sistemas de software más conocido y utilizado.

Unidad funcional: Colección de archivos de procedimiento y/o archivos de inclusión que trabajan en conjunto para realizar alguna función particular del sistema.
Variable: Una variable es una ubicación temporal de datos en memoria.

Widget: Combinación de las palabras window-gadget, que se interpretaría como aparato, artilugio o dispositivo de ventana. Para una aplicación ABL, es un objeto que proporciona capacidades visuales e interactivas.

6. 2 Bibliografía

SADD, John (2006): ABL, Massachusetts: EE.UU., Progress Software Corporation.

PROGRESS SOFTWARE (edit.) (1994): Progress Language, Massachusetts: EE.UU., Progress Software Corporation.

MERCER-HURSH, Thomas: Modeling Existing ABL Systems with UML, www.oehive.org/ABL2UML

MERCER-HURSH, Thomas: UML Profile for ABL, www.oehive.org/UMLProfile

NÓMADESOFT S.R.L.: Manual de Usuario de Gestión de Expedientes, www.nomadesoft.com.ar/wiki/index.php/Archivo:MAUS_IRRI3006_GestiónExpedientes.pdf

NÓMADESOFT S.R.L.: Programación P4GL.

EPIDATA CONSULTING: Bloque de Conocimiento de UML 2.0, www.epidataconsulting.com/tikiwiki/tiki-index.php?page=UML+2.0

MILESTONE CONSULTING: Base de Conocimiento de UML y Arquitecturas, www.milestone.com.mx/BaseConocimientoUML.htm