Diseño de un ABM para Datos Personales

Maximiliano Cristiá
Ingeniería de Software
Facultad de Ciencias Exactas, Ingeniería y Agrimensura
Universidad Nacional de Rosario

2016

Resumen

Este documento contiene parte de la documentación de diseño de un sistema muy simple para administrar datos personales. El diseño es orientado a objetos y arquitectónicamente responde al estilo conocido como Cliente/Servidor de Tres Capas. Si bien el sistema es muy simple creemos que brinda suficiente detalle como para que a partir de este se puedan diseñar sistemas más complejos.

Se puso énfasis en mostrar cómo obtener una adecuada separación de las capas usando objetos. Además se aplicó la metodología de Parnas para descomponer un sistema en módulos.

Por otro lado, este diseño muestra cómo transformar el diseño estructurado mostrado en el apunte "Diseño de Software" para los mismos requerimientos.

1. Los requerimientos

Implementar un ABM de datos personales. Los datos personales que hay que registrar son nombre y apellido, DNI, fecha de nacimiento y edad. La primera versión debe implementar una interfaz de texto y los siguientes requerimientos.

Repositorio de datos. En este caso el repositorio de datos será un archivo de texto UNIX (tipo `/etc/passwd`). El formato de cada línea del archivo será:

`DNI:nombre_apellido:DDMMAA:EDAD`

donde `DNI` y `nombre_apellido` son cadenas de caracteres de longitud variable y `EDAD` son dos dígitos. Se almacenarán los datos de una persona por línea.

Alta. Se reciben los datos de una persona, se controla que la persona no exista en el archivo y que los datos tengan ciertas restricciones (por ejemplo el DNI no tiene más de 8 cifras) y se los almacena en el repositorio.

Baja. Recorrer el archivo mencionado más arriba y si hay una línea que comienza con el DNI que se recibe como parámetro se deberá eliminar la línea correspondiente.

Modificación. Recibir un DNI y nuevos datos de ese documento, buscar el DNI en el archivo y, si existe, reemplazar los datos.

Búsqueda por nombre. Recorrer el archivo en busca del nombre que se le suministra y si está, emitir por salida estándar la línea completa sin los "`:`" e imprimiendo la fecha con el formato "`DD de MM de AA`".

Búsqueda por DNI. Hacer lo mismo que en el caso anterior sólo que la búsqueda se realiza por DNI.

Listado de nombres. Emitir por salida estándar cada una de las líneas del archivo eliminando el separador e imprimiendo la fecha como en los casos anteriores.

2. Los ítem de cambio

Siguiendo la metodología de Parnas para diseñar un sistema de software se analizaron los posibles cambios que podrían aparecer y se concluyó que los más probables son los siguientes.

- La interfaz de texto puede ser reemplazada por una interfaz gráfica o una basada en mensajes de correo electrónico. Incluso pueden coexistir todas ellas.
- El formato de cada registro puede variar. Los campos podrían estar en otro orden, el separador podría ser otro, etc.
- El formato del archivo podría ser otro. Por ejemplo, los datos podrían guardarse en dos o más archivos o podría ser reemplazado por un DBMS, etc.
- Los usuarios o clientes podrían solicitar otras consultas.
- El formato del resultado de una consulta puede cambiarse a pedido de un usuario o cliente.
- El destino de los resultados de las consultas (actualmente la pantalla) puede cambiar.

3. Los módulos del diseño

En esta sección describimos en 2MIL las interfaces de los módulos que conforman el diseño, la Estructura de Módulos y la Estructura de Herencia.

3.1. Estructura de Módulos

Comenzamos enumerando los módulos lógicos del diseño.

MG DP

Module	**ABMDatosPersonales**
comprises	Presentacion
	LogicaNegocio
	AccesoADatos
	ManejoErrores

MG DP

Module	**Presentacion**
comprises	CoordinadorInterfaz
	Menus
	OpcionesMenus
	Ordenes
	PresentacionResultadosConsultas

MG DP

Module **Menus**
comprises
Menu
MenuPrincipal
MenuBusquedas
MenuListados

MG DP

Module **PresentacionResultadosConsultas**
comprises
PresentarRBusquedaNombre
PresentarRBusquedaDNI
PresentarRListadoNombres

MG DP

Module **OpcionesMenus**
comprises
OpcionMenu
OpcionesOperaciones
OpcionesBusquedas
OpcionesListados

MG DP

Module **OpcionesOperaciones**
comprises
OpcionAlta
OpcionBaja
OpcionModificacion

MG DP

Module **OpcionesBusquedas**
comprises
OpcionBusquedaNombre
OpcionesBusquedaDNI

MG DP

Module **OpcionesListados**
comprises
OpcionListadoNombres

		MG DP
Module	**Ordenes**	
comprises	Orden	
	OrdenAlta	
	OrdenBaja	
	OrdenModificacion	
	OrdenBusquedaNombre	
	OrdenBusquedaDNI	
	OrdenListadoNombres	

		MG DP
Module	**LogicaNegocio**	
comprises	Alta	
	Baja	
	Modificacion	
	DatosPersonales	
	BusquedasYListados	
	BaseDatosPersonales	

		MG DP
Module	**BusquedasYListados**	
comprises	BusquedaNombre	
	BusquedaDNI	
	ListadoNombres	

		MG DP
Module	**AccesoADatos**	
comprises	RecuperarDatos	
	GuardarDatos	

		MG DP
Module	**ManejoErrores**	
comprises	Errores	
	Mensajes	

3.2. Especificación de Interfaces y Estructura de Herencia

Ahora especificamos las interfaces de los módulos físicos. Algunos de los módulos físicos solo son interfaces que no deben ser implementadas y a partir de los cuales se definen herederos que sí serán implementados.

Todos los elementos que forman la interfaz deben comunicarse con una instancia global de CoordinadorInterfaz para que esta decida cómo y cuando presentar ese elemento. Inicialmente en main() se declara una instancia de MenuPrincipal y se llama a CoordinadorInterfaz::presentarMenu() pasándole como parámetro la instancia de MenuPrincipal; a su vez el coordinador invoca a dibujar() del

parámetro recibido.

	MG DP
Module	**CoordinadorInterfaz**
imports	Menu, OpcionMenu, Orden
exportsproc	presentarMenu(**i** Menu) presentarOpcionMenu(**i** OpcionMenu)

Tanto Menu como OpcionMenu son módulos que actúan como interfaces y no deben ser implementados (por lo que no tiene sentido que haya una cláusula **imports**). Cada menú y cada opción de menú son, respectivamente, sus herederos. Comenzamos con los menús.

	MG DP
Module	**Menu**
exportsproc	dibujar()

dibujar() imprime las opciones del menú; espera, muestra y gestiona la entrada relativa al menú concreto del cual se trate; y crea una instancia de la opción de menú o del menú seleccionado. En otras palabras, cada menú convierte una entrada en un objeto. Por el momento los herederos son:

	MG DP
Module	**MenuPrincipal inherits from** Menu
imports	MenuBusquedas, MenuListados, OpcionAlta, OpcionBaja, OpcionModificacion, OrdenAlta, OrdenBaja, OrdenModificacion, CoordinadorInterfaz

	MG DP
Module	**MenuBusquedas inherits from** Menu
imports	OpcionBusquedaDNI, OpcionBusquedaNombre, OrdenBusquedaDNI, OrdenBusquedaNombre, CoordinadorInterfaz

	MG DP
Module	**MenuListados inherits from** Menu
imports	OpcionListadoNombres, OrdenListadoNombres, CoordinadorInterfaz

Las opciones de menú deben dibujar el cuadro de diálogo que les permita recibir todos los parámetros necesarios y gestionar la entrada hasta que se considere correcta (todo esto lo implementa dibujar()). Además debe ser posible asociar una Orden a cada OpcionMenu que se debería ejecutar una vez que se ha capturado toda la entrada. Esta asociación se implementa en el menú al cual pertenece la opción, y la ejecución de la orden en CoordinadorInterfaz. Por todo esto la interfaz abstracta de las opciones de menú es la siguiente.

	MG DP
Module	**OpcionMenu**
exportsproc	inicializar(**i** Orden) dibujar() miOrden():Orden
comments	inicializar() puede implementarse como el constructor.

Como cada cuadro de diálogo captura datos diferentes que deben ser comunicados a la capa de negocio, el entorno debe poder acceder a dichos datos. En consecuencia cada opción de menú extiende la interfaz de OpcionMenu con subrutinas para acceder a sus datos específicos.

MG DP

Module	**OpcionAlta inherits from** OpcionMenu
imports	OrdenAlta, DatosPersonales, ManejoErrores
exportsproc	datos():DatosPersonales

MG DP

Module	**OpcionBaja inherits from** OpcionMenu
imports	OrdenBaja, DNI, ManejoErrores
exportsproc	dni():DNI

MG DP

Module	**OpcionModificacion inherits from** OpcionMenu
imports	OrdenModificacion, DatosPersonales, DNI, ManejoErrores
exportsproc	datos():DatosPersonales
	dni():DNI

MG DP

Module	**OpcionBusquedaNombre inherits from** OpcionMenu
imports	OrdenBusquedaNombre, ManejoErrores
exportsproc	nombre():String

MG DP

Module	**OpcionBusquedaDNI inherits from** OpcionMenu
imports	OrdenBusquedaDNI, DNI, ManejoErrores
exportsproc	dni():DNI

MG DP

Module	**OpcionListadoNombres inherits from** OpcionMenu
imports	OrdenListadoNombres, ManejoErrores

Notar que en algunos herederos se usan módulos de la lógica de negocios.

Las capa de presentación y la capa de negocios interactúan, entre otras formas, a través de los errores que se comunican. Usualmente la lógica de negocios detecta un error y lo debe comunicar a la lógica de presentación para que esta lo presente adecuadamente. El código correspondiente a la detección del error se debe incluir en los módulos de la lógica de negocios, en tanto que el código necesario para mostrar los errores debe incluirse en los módulos de la lógica de presentación. La forma en que estas dos capas interactúan es utilizando los módulos Errores y Mensajes. El primer módulo es una pila de los errores que se han ido detectando en tanto que el segundo relaciona el código de

un error con el mensaje que se debe mostrar.

	MG DP
Module	**Errores**
exportsproc	nuevoError(**i** Int) ultimoError(): Int hayMas():Bool

	MG DP
Module	**Mensajes**
exportsproc	inicializar() mensaje(**i** Int):String
comments	inicializar() obtiene los mensajes de algún archivo de configuración.

Se supone ambos módulos son únicos y globales. Los módulos de la lógica de negocios apilan errores (nuevoError()) en la instancia de Errores y los módulos de la lógica de presentación los desapilan (ultimoError()) hasta que la pila quede vacía (hayMas()), y muestran cada error consultando a la tabla de los mensajes.

A continuación especificamos la interfaz de las diversas órdenes. Una Orden implementa la relación entre una opción de menú y la o las operaciones de la Lógica de Negocios correspondientes; son el nexo entre la Lógica de Presentación y la Lógica de Negocios. El menú donde están cada una de las opciones compone cada opción con su orden, cuando las primeras son creadas. Más tarde, una vez que el método dibujar() de una opción de menú ha finalizado, el CoordinadorInterfaz ejecuta la orden de esa opción de menú, pasándole esa misma opción como parámetro. Orden es una interfaz que no debe ser implementada (por lo que que no se incluye la cláusula **imports**).

	MG DP
Module	**Orden**
exportsproc	inicializar(**i** OpcionMenu) ejecutar()
comments	inicializar() NO puede implementarse como el constructor porque OpcionMenu recibe una Orden durante su creación.

Los herederos corresponden a cada opción de menú solicitada actualmente por el cliente.

	MG DP
Module	**OrdenAlta inherits from** Orden
imports	OpcionAlta, Alta, DatosPersonales, Errores

	MG DP
Module	**OrdenBaja inherits from** Orden
imports	OpcionBaja, Baja, DNI, Errores

	MG DP
Module	**OrdenModificacion inherits from** Orden
imports	OpcionModificacion, Modificacion, DNI, DatosPersonales, Errores

	MG DP
Module	**OrdenBusquedaNombre inherits from** Orden
imports	OpcionBusquedaNombre, BusquedaNombre, PresentarRBusquedaNombre, BaseDatosPersonales, Errores

	MG DP
Module	**OrdenBusquedaDNI inherits from** Orden
imports	OpcionBusquedaDNI, BusquedaDNI, PresentarRBusquedaDNI, BaseDatosPersonales, Errores

	MG DP
Module	**OrdenListadoNombres inherits from** Orden
imports	OpcionListadoNombres, ListadoNombres, PresentarRListadoNombres, BaseDatosPersonales, Errores

Las órdenes que corresponden a opciones de consulta incluyen, además, las sentencias necesarias para que los resultados de las consultas se muestren, lo cual lo hacen usando convenientemente los módulos pertenecientes a PresentacionResultadosConsultas. Esto podría implementarse en CoordinadorInterfaz extendiendo su interfaz para que presente resultados y no solo elementos de la interfaz.

Con el objetivo de clarificar la relación entre OpcionMenu, CoordinadorInterfaz, Orden, etc. suministramos la siguiente Secuencia de Invocaciones, la cual muestra un ejemplo de dónde, cómo y cuándo se realizan las invocaciones entre objetos. En la parte final del ejemplo se muestra una especificación incompleta de OrdenBusquedaDNI.

Declarations

$presentarMenu == ci.presentarMenu$

$opcion == opbd : OpcionBusquedaDNI$

$orden == obd : OrdenBusquedaDNI$

$presentarOM == ci.presentarOpcionMenu$

$ejecutarBusqueda == OrdenBusquedaDNI :: ejecutar$

$buscar == BusquedaDNI :: buscar$

$presentar == PresentarRBusquedaDNI :: dibujar$

Description

$main = \textbf{new}\ ci : CoordinadorInterfaz \rightarrow presentarMenu!\textbf{new}\ m \rightarrow END$

$presentarMenu?m = m.dibujar \rightarrow END$

$dibujar = \textbf{new}\ opcion \rightarrow opbd.inicializar!\textbf{new}\ orden \rightarrow presentarOM!opbd \rightarrow END$

$presentarOM?op = op.dibujar \rightarrow op.miOrden?o \rightarrow o.inicializar!op \rightarrow o.ejecutar \rightarrow END$

$ejecutarBusqueda = op.dni?d \rightarrow buscar!d?resp \rightarrow presentar!resp \rightarrow END$

Los módulos que forman parte de PresentacionResultadosConsultas se encargan de presentar el resultado de una búsqueda o listado. En consecuencia el método dibujar() requiere uno o más

parámetros. Los herederos de Orden son quienes invocan los servicios de estos módulos.

	MG DP
Module	**PresentarRBusquedaNombre**
exportsproc	dibujar(**i** BaseDatosPersonales)

	MG DP
Module	**PresentarRBusquedaDNI**
exportsproc	dibujar(**i** BaseDatosPersonales)

	MG DP
Module	**PresentarRListadoNombres**
exportsproc	dibujar(**i** BaseDatosPersonales)

Con esto finaliza la especificación de la interfaz de los módulos pertenecientes a la Lógica de Presentación. Continuamos, entonces, con los módulos de la Lógica de Negocios. Estos módulos implementan las reglas de negocio de cada una de las operaciones requeridas. Obtienen los datos de un objeto global BaseDatosPersonales inicializado en main() usando los módulos de la Lógica de Procesamiento de Datos (módulo AccesoADatos); asimismo modifican ese objeto el cual será grabado en disco por main() justo antes de que el programa finalice. Es decir, ningún módulo de LogicaNegocio accede a los datos físicos por lo cual desconoce dónde y cómo son almacenados.

Comenzamos por las operaciones básicas que modifican datos.

	MG DP
Module	**Alta**
imports	DatosPersonales, BaseDatosPersonales
exportsproc	alta(**i** DatosPersonales)

	MG DP
Module	**Baja**
imports	DatosPersonales, BaseDatosPersonales, DNI
exportsproc	baja(**i** DNI)

	MG DP
Module	**Modificacion**
imports	DatosPersonales, BaseDatosPersonales, DNI
exportsproc	modificar(**i** DNI, **i** DatosPersonales)

Continuamos con las operaciones que consultan datos. Los resultados de las consultas se guardan en instancias temporales de BaseDatosPersonales; los módulos encargados de presentar los resultados

pueden utilizar los datos que necesiten.

MG DP

Module	**BusquedaNombre**
imports	DatosPersonales, BaseDatosPersonales
exportsproc	buscar(**i** String):BaseDatosPersonales

MG DP

Module	**BusquedaDNI**
imports	DatosPersonales, BaseDatosPersonales, DNI
exportsproc	buscar(**i** DNI):BaseDatosPersonales

MG DP

Module	**ListadoNombres**
imports	BaseDatosPersonales
exportsproc	listar():BaseDatosPersonales

Un diseño alternativo para todos los módulos de la Lógica de Negocios presentados hasta aquí po-dría ser fusionarlos en un único módulo que exporte una interfaz que incluya cada uno de los métodos. Sin embargo, con ese diseño ese módulo ocultaría varios secretos (cada una de las reglas de negocio).

Dentro de la Lógica de Negocios también tenemos los nexos entre esta capa y la capa de acceso a los datos. Estos módulos constituyen una suerte de base de datos no persistente pero orientada a objetos.

MG DP

Module	**DatosPersonales**
imports	DNI, Fecha
exportsproc	setNombreYApellido(**i** String, **i** String) setDNI(**i** DNI) setFechaNac(**i** Fecha) setEdad(**i** Int) getNombre():String getApellido(): String getDNI(): DNI getFechaNac():Fecha getEdad():Edad

MG DP

Module	**BaseDatosPersonales**
imports	DatosPersonales
exportsproc	add(**i** DatosPersonales) first() next() more():Bool get():DatosPersonales delete()

Finalmente es el turno de presentar los módulos de la capa de acceso a datos. Estos módulos son los únicos que conocen dónde y cómo se guardan persistentemente los datos. En esta primera versión los datos se leen de un archivo al iniciarse el programa y se guardan justo antes de que este finalice.

MG DP

Module	**RecuperarDatos**
imports	DatosPersonales, BaseDatosPersonales
exportsproc	recuperar():BaseDatosPersonales

MG DP

Module	**GuardarDatos**
imports	DatosPersonales, BaseDatosPersonales
exportsproc	guardar(**i** BaseDatosPersonales)

Es interesante resaltar que las cláusulas **imports** de los módulos muestran claramente la separación en capas que impone el diseño.

4. Ejercicios

Sugerimos resolver los siguientes ejercicios.

Ejercicio 1 *Completar la documentación de diseño con: Guía de Módulos, Estructura de Uso y Estructura de Objetos. Dibujar la Estructura de Módulos y la de Herencia.*

Ejercicio 2 *Determine informal pero objetivamente si el diseño es correcto o no. Justifique su respuesta.*

Ejercicio 3 *Implementar el diseño en un lenguaje orientado a objetos y en otro no orientado a objetos.*

Ejercicio 4 *Completar el diseño con los mensajes de error. Implementar esta funcionalidad.*

Ejercicio 5 *Actualizar la documentación de diseño y programar otra versión que incluya:*

- *Listado de nombres ordenado alfabéticamente.*
- *Búsqueda por rango de fecha de nacimiento.*
- *Consulta de los datos de un DNI por medio de mensajes de correo electrónico. Es decir, se recibe un mensaje con el DNI y se envía otro con el contenido de la consulta.*

Ejercicio 6 *Según se comentó informalmente en la descripción del diseño del sistema, en esta versión los datos se leen en el momento en que comienza a ejecutarse el programa y se guardan justo antes de que este finalice.*

Rediseñe el sistema de forma tal que sea posible implementar otras políticas de acceso a los datos. ¿Tuvo que cambiar el diseño? ¿Fue un cambio importante? ¿La implementación de cuántos módulos tuvo que modificar? ¿Cuántas interfaces? ¿Dónde estuvo el error? ¿En la metodología de Parnas, en la aplicación de la metodología o en otro lado? ¿Qué podría hacer para no volver a cometer un error similar?

Ejercicio 7 *¿Por qué no se incluyó en el método dibujar() de todos los módulos la ventana donde deberían dibujar? ¿Impide esto implementar una interfaz gráfica? Si es así, ¿dónde estuvo el error? ¿En la metodología de Parnas, en la aplicación de la metodología o en otro lado? ¿Qué podría hacer para no volver a cometer un error similar? ¿Qué costo tiene reparar ese error habiendo implementado ya tres versiones del sistema?*

Ejercicio 8 *Suponga que un nuevo cliente quiere guardar, además, la dirección de cada persona y tener alguna consulta sobre ese campo. ¿Cómo afecta este requerimiento al diseño y a su implementación? ¿Falló la metodología de Parnas o la aplicación de la metodología? ¿Sigue siendo correcto el diseño? ¿Qué podría hacer para no volver a cometer un error similar? ¿Qué costo tiene reparar ese error habiendo implementado ya tres versiones del sistema?*

5. Un ítem de cambio no considerado

En esta sección presentamos una modificación del diseño presentado en la sección 3. Esta nueva versión tiene en cuenta un nuevo ítem de cambio por lo que aprovechamos le hecho al comienzo pero el lector debe considerar como que el diseño se hace desde cero teniendo en cuenta el nuevo ítem de cambio (en otras palabras, no es un ejercicio para mostrar una modificación de un diseño, sino un ejercicio para mostrar cómo diseñar teniendo en cuenta la metodología de Parnas). Se tiene en cuenta el siguiente ítem de cambio.

- Es muy posible que el destinatario del sistema solicite se registren más datos de una persona.

Además, se trata de incorporar los cambios a través de nuevos módulos que se integran con los existentes, evitando así tener que modificar código en el caso en que se dé la situación prevista en el ítem de cambio.

Antes de redefinir las interfaces de los módulos notemos que al agregar un nuevo dato personal es *necesario* modificar:

- OpcionAlta y OpcionModificacion de manera tal que ambas dibujen en la pantalla el nombre del nuevo dato y reciban la entrada correspondiente.
- El tipo DatosPersonales para que permita almacenar y recuperar este nuevo dato.
- Los resultados de las consultas y listados para que muestren este nuevo dato.
- Los módulos de la Lógica de Procesamiento de Datos.

Por otro lado, es *posible* que haya que modificar:

- Alta y Modificacion porque puede ser necesario modificar las reglas de negocio para ejecutar estas operaciones.

Finalmente, es *posible* que el cliente requiera nuevas consultas y/o listados que involucren a este nuevo dato.

Teniendo en cuenta todas estas consideraciones y previendo que el cliente puede pedir agregar más de un dato, proponemos el siguiente diseño.

Modificamos la interfaz de DatosPersonales de forma tal que permita agregar tantos *atributos extendidos* como queramos.

MG DP

Module	**DatosPersonales**
imports	DNI, Fecha, AtributosExtendidos
exportsproc	setNombreYApellido(**i** String, **i** String)
	setDNI(**i** DNI)
	setFechaNac(**i** Fecha)
	setEdad(**i** Int)
	setAtributosExtendidos(**i** AtibutosExtendidos)
	getNombre():String
	getApellido(): String
	getDNI(): DNI
	getFechaNac():Fecha
	getEdad():Edad
	getAtributosExtendidos():AtibutosExtendidos

Los atributos extendidos permiten incorporar y consultar una cantidad arbitraria de pares de la forma (*atributo*, *valor*) de un tipo suficientemente general, como se muestra a continuación.

MG DP

Module	**AtributosExtendidos**
exportsproc	agregar(**i** String atr, **i** String val)
	first()
	next()
	more():Bool
	getAtr():String
	getVal():String
	delete()

El atributo es una cadena de caracteres que identifica al nuevo dato o campo solicitado por el cliente, por ejemplo `direccion`, en tanto que el valor es, precisamente, el valor de ese atributo para cada persona, por ejemplo `Tucuman 4142`.

Obviamente, usar los atributos extendidos implica una pérdida de desempeño porque, por ejemplo, al guardar un número entero como una cadena de caracteres ocupará más espacio y además requerirá que sea convertido idea y vuelta entre cadena de caracteres y entero cada vez que deba ser procesado. Por otro lado, si los atributos extendidos no son necesarios no ocupan lugar y si son necesarios ocupan un espacio proporcional a la cantidad de nuevos datos requeridos.

Ahora modificamos la interfaz de las opciones de menú para que tengan en cuenta nuevos datos

que deben solicitar al usuario.

	MG DP
Module	**OpcionMenu**
exportsproc	inicializar(**i** Orden) dibujar() miOrden():Orden ampliarForm(**i** InputElement)
comments	inicializar() puede implementarse como el constructor.

InputElement es una interfaz que nos permite heredar diferentes elementos que se dibujan a sí mismos y que se encargan de capturar la entrada del usuario. Cada InputElement se inicializa con un nombre que será el que se use como atributo en AtributosExtendidos. El método posicion() permite fijar las coordenadas para que el elemento se dibuje a sí mismo en un determinado lugar de la pantalla; en tanto que el método leer() retorna la entrada del usuario luego de aplicar algún filtro de entrada adecuado (es decir, cada leer() implementa un filtro de entrada, por ejemplo leer tres caracteres).

	MG DP
Module	**InputElement**
exportsproc	inicializar(**i** String) posicion(**i** Int, **i** Int) dibujar() leer():String getNombre():String
comments	inicializar() puede implementarse como el constructor.

De esta forma una opción de menú puede ser configurada con muchos InputElement, la OpcionMenu los puede distribuir en la pantalla y puede coordinarlos para que cada uno lea su dato en el momento adecuado. Cuando read() de un InputElement finaliza, la OpcionMenu que lo contiene recibe el dato leído y puede recuperar el atributo al cual corresponde llamando a getNombre(). Luego, almacena el par en el objeto AtributosExtendidos que previamente habrá creado. Notar que OpcionMenu debe crear dicho objeto solo si su lista de InputElement es no vacía, y si dicha lista es vacía, la opción solo dibuja el cuadro de diálogo original.

Algo semejante hacemos con los módulos que se encargan de mostrar los resultados de consultas. Por ejemplo PresentarRBusquedaNombre queda así.

	MG DP
Module	**PresentarRBusquedaNombre**
exportsproc	dibujar(**i** BaseDatosPersonales) ampliarForm(**i** OutputElement)

donde OutputElement tiene una interfaz muy parecida a InputElement.

Adaptamos los módulos a nivel de la Lógica de Negocios por medio de la técnica de *envoltura* o *wrapping*. Veamos el caso de Alta. Primero renombramos ese módulo a, por ejemplo, **Alta1** y luego definimos un nuevo módulo **Alta** que tiene exactamente la misma interfaz (de esta forma los componentes que dependían originalmente de **Alta** no se ven afectados por el cambio). La diferencia entre ambos está en la implementación: **Alta** usará los servicios de **Alta1** pero antes o después podrá implementar alguna regla de negocio que involucre a los nuevos datos.

Finalmente es el turno de los módulos de la capa de acceso a datos. Esta es la parte más compleja porque depende mucho de cómo se guarden los datos. Por ejemplo, si los nuevos datos se ponen en

un nuevo archivo relacionado con el primero por el campo DNI entonces se puede usar la técnica de envoltura usada para la Lógica de Negocios; lo mismo es válido si los nuevos datos se agregan al final de cada línea y recuperar() no busca el fin de línea. En cualquier otro caso no será posible resolver el problema agregando módulos sino que habrá que modificar los módulos existentes. En nuestra opinión siempre es mejor ensuciar el modelo de datos si esto evita tener que modificar código que funciona bien (a menos que esto provoque una caída de desempeño prohibitiva).

Ejercicio 9 *Si no hay nuevas reglas de negocio para dar de alta los datos personales que incluyen los nuevos campos solicitados por el cliente, ¿es necesario renombrar* **Alta** *y definir* **Alta1***? Justifique.*

Ejercicio 10 *Definir la interfaz del tipo* **OutputElement**.

Ejercicio 11 *Mostrar cómo y dónde se configura el sistema.*

Ejercicio 12 *Documentar la estrategia de cambio.*

Referencias