54
24-

# UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

## OBJETOS PERSISTENTES. UNA APLICACION EN LA INDUSTRIA PETROLERA

**T E S I S**

QUE PARA OBTENER EL TITULO DE:

INGENIERO EN COMPUTACION

P R E S E N T A :

MARIA ELOISA HERNANDEZ HERNANDEZ

DIRECTOR: M.C. ALFONSO MIGUEL REYES

CODIRECTOR: DRA. ANA MARIA VAZQUEZ VARGAS

MEXICO, D.F. 1998.


**UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso**

# OBJETOS PERSISTENTES:

# UNA APLICACIÓN EN

# LA INDUSTRIA PETROLERA.

# DEDICATORIAS

A mi madre *Lourdes Hernández Angeles:*
Por todo el amor que me has brindado, porque sembraste en mi los deseos de superación, por toda una vida de sacrificios y esfuerzos, porque simplemente sin tu apoyo no hubiera empezado jamás. Este trabajo es para ti.

A mi padre *Juan Hernández Gómez:*
Porque a pesar de tu silencio se que me amas y estas conmigo en todo momento, gracias por confiar en mi y no dejar que mis sueños se esfumaran.

A mi hermana *Guadalupe Hernández Hdez. y* a mis *sobrinos*:
Por el apoyo y ánimo que siempre me han dado, llenándome de paz y alegría en los instantes que estamos juntos.

A mis *amigas y amigos*:
Por su amistad incondicional, por compartir ratos buenos y malos conmigo, porque son ustedes muy valiosos en mi vida.

A *Dios*:
Porque me has llenado de bendición al permitirme finalizar esta meta, porque todo lo que he logrado lo hecho gracias a ti.

Al *Instituto Mexicano del Petróleo:*
Por la oportunidad que me dieron para realizar este trabajo y en especial a mi director de tesis *Alfonso Miguel Reyes*, por su apoyo y a todas las personas que hicieron agradable mi instancia en el IMP.

A la *Universidad Nacional Autónoma de México:*
Por ser mi Alma Mater y a la Facultad de Ingeniería por abrirme las puertas al conocimiento. Agradezco a todos los profesores y en especial a mi codirectora de tesis *Dra. Ana María Vázquez Vargas* por su dedicación y ayuda.

## RESUMEN.

Este trabajo presenta la Tecnología Orientada a Objetos (TOO), como una mejor opción para los sistemas de información, enfocándose en el concepto de persistencia.

La persistencia es una característica de la TOO, que consiste en que los objetos puedan escribir y leer por ellos mismos, de manera transparente al usuario, sin preocuparse de saber, si el objeto está en almacenamiento permanente o transitorio o si está relacionado con otros objetos; en consecuencia los objetos tienen la habilidad de vivir más allá de la vida de los programas que los crearon.

En base a su complejidad existen tres niveles de persistencia: simple, isomórfica y polimórfica; siendo esta última el nivel más alto de persistencia.

En base a su arquitectura existen dos caminos para el almacenamiento de objetos persistentes: Base de Datos y Archivos. También hay distintas técnicas y enfoques para implementar un sistema con objetos persistentes. La mejor elección va depender de lo que se requiera.

Además de la persistencia, se muestra dos normas de la industria petrolera: Geoshare y el formato RP66 (DLIS). La primera determina la semántica de los datos petroleros y la segunda establece la sintaxis, es decir, la manera en que se va a acomodar la información en el archivo. Estas normas son necesarias para el manejo de la información petrolera.

Finalmente se culmina con el desarrollo de un sistema prototipo que integra datos voluminosos petroleros en objetos persistentes, para su desarrollo se usa la metodología denominada "Desarrollo de Software con Objetos Orientados a Calidad -DSOOC-"; usando como lenguaje de programación Builder C++ ver. 1.0 , bajo el sistema operativo Windows 95.

# 1. INTRODUCCIÓN.

El presente trabajo tiene como propósito mostrar la importancia de aplicar Tecnología Orientada a Objetos (TOO), en sistemas de información, en especial cuando se habla de grandes volúmenes de información.

Una de las tendencias actuales es la aplicación de técnicas con orientación a objetos, debido a que proporcionan mayores beneficios al aplicarse a sistemas complejos, contrastando con las técnicas tradicionales. Esto puede aparentar un crecimiento en complejidad; sin embargo, no implica que el desarrollador deba conocer la estructura interna de los objetos, si bien los objetos tuvieron una programación previa de cierta complejidad, un desarrollador práctico deberá limitarse a conocer su comportamiento y su forma de uso.

Este trabajo se enfoca especialmente en el concepto de objetos persistentes, dado que éste proporciona el beneficio de almacenar/recuperar los objetos de manera transparente, en forma integra independientemente si el programa que los creo existe o no, esto lleva a poder utilizarlos en distintos procesos.

Actualmente la industria del petróleo maneja grandes volúmenes de datos y trae consigo problemas al tratar de integrarlos, por lo que se busca alcanzar su integración aplicando TOO.

El trabajo está integrado por siete capítulos, a través de los cuales se refleja una amplia visión acerca de los objetos persistentes desde las características, los niveles que hay, los métodos para aplicarla hasta el desarrollo de un prototipo, el cual además busca establecer una estandarización por medio de la utilización del formato Geoshare y del RP66 (DLIS).

El capítulo 2 presenta las características de la información petrolera, con el fin de mostrar lo importante que resulta manejarlo a través de objetos persistentes, así como las características y normas que deben de seguir los datos petroleros en base a estándares establecidos en la industria petrolera, en especial los formatos RP66 (DLIS) y Geoshare. Con esto se pretende mostrar los conceptos fundamentales para comprender el contexto en el que se desarrollará el objeto persistente.

El capítulo 3 proporciona una visión general de lo que es la Tecnología Orientada a Objetos (TOO), en especial el concepto de persistencia. Muestra los diferentes ámbitos de la persistencia a través de archivos, Base de Datos, etc. también se presentan los niveles que existen de persistencia en base a la complejidad que se tiene para su implementación.

El capítulo 4 da a conocer varios métodos para implementar objetos persistentes en archivos utilizando un lenguaje no persistente, buscando utilizar la combinación de varios enfoques, que lleven al desarrollo del modelo más adecuado en base a las características de los datos y del ambiente de trabajo.

El capítulo 5 plantea un problema real que se tiene en el manejo de los datos petroleros, el cual se pretende solucionar utilizando persistencia sin omitir sus estándares. Para el desarrollo de este sistema se utiliza la metodología denominada Diseño de Software Orientado a Objetos con Calidad (DSOOC), el cual es considerado un estándar en la Línea de Negocios Adquisición y Procesamiento de Datos de Pozo del IMP (Instituto Mexicano del Petróleo). Las fases que se cubren con esta metodología en este capítulo son el de análisis y diseño, teniendo como resultado un modelo básico.

El capítulo 6 presenta la implementación del modelo del objeto persistente, el cual utiliza el lenguaje Builder C++ versión 1.0 bajo ambiente Windows 95. Además se muestra el diseño detallado del mismo y las pruebas que se llevaron a cabo para determinar el correcto funcionamiento del sistema en diferentes niveles.

Finalmente el capítulo 7 presenta las conclusiones que se obtuvieron a lo largo de la elaboración de la presente tesis.

# 2. ESTÁNDARES DE LA INDUSTRIA PETROLERA.

## *2.1 CARACTERÍSTICAS DE LA INFORMACIÓN PETROLERA.*

### 2.1.1 INTRODUCCIÓN.

Los registros geofísicos de pozo representan las mediciones de las propiedades petrofísicas más aproximadas de las formaciones petroleras[1], dicha información es indispensable junto con la sísmica para estudios de caracterización integral de yacimientos; ésta posee las siguientes características:

- Se tienen registros de longitud variable, vectores o matrices.
- La información es de grandes volúmenes.
- Existen diferentes tipos de datos, pocas instancias.
- La evolución de la información es de grandes periodos.
- Las transacciones son de gran duración.
- Es necesario guardar versiones viejas de procesamiento de información.

### 2.1.2 ADQUISICIÓN DE DATOS DE LOS REGISTROS.

Todos los modelos y ecuaciones comunes para la interpretación de registros pueden ejecutarse en la unidad CSU (Sistema Computarizado de Computación Digital). CSU es un sistema computarizado integral para la adquisición y el procesamiento de datos. Puede transmitir registros con un enlace de comunicación adecuado. Los medios pueden ser:

1. Vía satélite.
2. Por radio.
3. Vía telefónica.

Con ello la medición de registros básicos puede tener grandes cantidades de información.

### 2.1.3 PROCESAMIENTO DE DATOS.

Se realiza en por lo menos tres niveles:

1. En el pozo (en la herramienta).
2. A la boca del pozo (en el camión).
3. En un centro de cómputo central.

El lugar depende de:

1. Donde produce los resultados con mayor eficacia.
2. Donde se necesita primero la información.

3. Donde estén los expertos.
4. Donde lo exige las consideraciones tecnológicas.

Es conveniente llevar los datos primarios medidos a la superficie para su grabación y su procesamiento. CSU supera las limitaciones de los registros anteriores, de tener registros combinados (múltiples sensores de medición en una sola cadena de herramientas de registros), proporciona el potencial obvio para el procesamiento de datos en el lugar del pozo.

### 2.1.4 TRANSMISIÓN DE DATOS.

CSU puede transmitir datos en un sistema de comunicación adecuada. Los posibles receptores son:

1. Otro CSU.
2. Una terminal de transmisión.
3. Una central de cómputo.
4. Una máquina fax digital.
5. Una telecopiadora portátil.

### 2.1.5 FUNDAMENTOS DE LA INTERPRETACIÓN CUANTITATIVA DE REGISTROS.

Los parámetros petrofísicos para evaluar un depósito son:

- Porosidad.
- Saturación de hidrocarburos.
- Espesor.
- Area.
- Permeabilidad.

Con estos parámetros se puede determinar la terminación y producción de un yacimiento .

Del yacimiento se necesita su:

- Geometría.
- Temperatura.
- Presión.
- Litología.

Las formaciones productivas (yacimientos), se presenta en una cantidad casi ilimitada de formas, tamaños y orientaciones. Su productividad se puede ver afectada por la orientación y forma física del yacimiento. Existen yacimientos anchos y estrechos, espesos y delgados, grandes y pequeños. Con estos parámetros se puede determinar la terminación y producción de un yacimiento.

La temperatura y la presión también afectan la producción de hidrocarburos. En el yacimiento, la temperatura y la presión controlan las viscosidades y las solubilidades mutuas de los tres fluidos: petróleo, gas y $H_2O$.

La relación petróleo/gas se ve sometida a variaciones significativas en respuesta a cambios de temperatura y presión. Comúnmente, la temperatura de un yacimiento productivo no varia demasiado. Sin embargo, es indispensable una baja de presión entre yacimiento virgen y el pozo.

## 2.1.6 INTERPRETACIÓN DE REGISTROS.

De los parámetros antes mencionados, sólo pueden obtenerse algunos de manera directa y los restantes deben de inferirse u obtenerse de la medición de otros parámetros físicos. Una interpretación de registro es el proceso por el cual dichos parámetros mensurables, se traducen a los parámetros petrofísicos deseados de porosidad, saturación de hidrocarburos, permeabilidad, productividad, litología, etc.

Durante el proceso de perforación los fluidos dentro de los poros de la roca que rodea el agujero pueden verse desplazados o contaminados debido a la invasión por el líquido de perforación, lo cual complica la interpretación de los parámetros.

Históricamente, la interpretación de registros se ha realizado en un proceso secuencial de operaciones lógicas. El analista de registros determina primero un registro, después otro y así sucesivamente hasta que el problema, está resuelto. Este enfoque tiene ventaja de ser comprensible, puede repetirse y es lógicamente aceptable.

La mayoría de los programas de computadoras reproducen este proceso clásico secuencial de interpretación manual, lo cual resulta lógico ya que el proceso estaba bien documentado y era relativamente fácil de programar la computadora para duplicarlo.

Sin embargo, estas técnicas han sido superado por la evolución de la interpretación de registros en su esfuerzo por manejar la complejidad de las formaciones en los que en la actualidad se busca petróleo y gas, y por la introducción de nuevos sensores sofisticados para medir características adicionales de las rocas del yacimiento. Como resultado cada vez se vuelve más difícil encontrar un camino secuencial en toda esta confusión de mediciones petrofísicos y modelos de interpretación que permita el mejor uso de todos los datos y conocimientos disponibles.

## 2.1.7 TIPOS DE REGISTROS.

- Registro de resistividad.
- Registro de porosidad.
- Registro de potencial y espontáneo y rayos gamma naturales.
- Registro NGS.
- Registro de porosidad.
- Registro Sónico.

- Registro Densidad.
- Registro Neutrones.

## *2.2 EL ESTÁNDAR GEOSHARE.*

### 2.2.1 INTRODUCCIÓN.

Actualmente en la industria del petróleo y gas en donde se manejan grandes volúmenes de datos, se busca alcanzar la integración de los mismos , antes que preocuparse por la manera de modelarlos. Ya se cuenta con diferentes herramientas creadas por distintos vendedores, en donde cada una tiene su propio método para almacenar sus datos , aunque se compren productos de un mismo vendedor, no se puede garantizar que los datos serán almacenados bajo el mismo método de almacenamiento. Todo esto origina un grave problema , que es cómo mover datos entre una aplicación a otra, sin invertir mucho tiempo y esfuerzo y lo principal es que los datos se conserven integrados. Para ello existen varios métodos de integración[2].

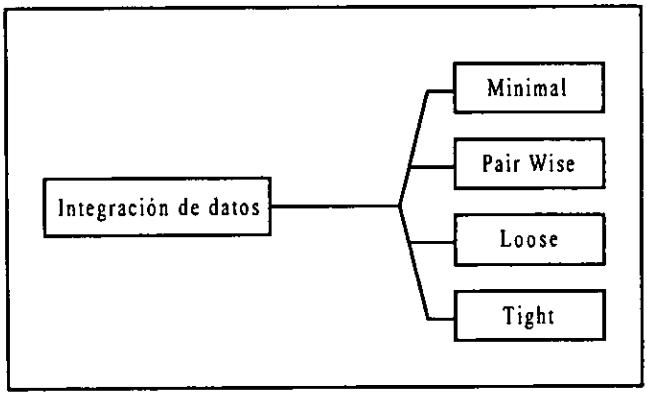



**Figura 2.1.- Métodos de integración.**

#### 2.2.1.1 Minimal.

Esta técnica esta muy próxima a no ser un método de integración, ya que maneja los datos de una manera un tanto insegura , al existir varios pasos en el que interviene el usuario y al tener un formato superficial casi estándar (SEG-Y, UKOOA, LIS), sin llegar hacerlo provocando que arroje como resultado formatos no adecuados que necesitan ser ajustados y/o extendidos en forma particular.

### 2.2.1.2 Pair-Wise.

El método trata de interactuar entre aplicaciones a través de ligas estrechas, que son creados en la mayoría de los casos por los usuarios con el apoyo de otras herramientas. Esta técnica ha alcanzado cierto éxito en aplicaciones pequeñas y nuevas en donde no existen grandes volúmenes de datos, ni muchas ligas, sin embargo no sucede así en aplicaciones muy grandes donde se maneja muchos datos , en donde además existe bastante interacción entre aplicaciones, provocando que pueda haber confusión y congestión, donde el problema principal consiste en que el usuario debe de aprender muchos términos técnicos, para poder mover sus datos a través de las aplicaciones existentes. Este método realiza tantas copias del dato como son requeridas por los usuarios.

### 2.2.1.3 Tight.

El método consiste en tener los datos guardados en un almacenamiento estándar, para que puedan ser accesadas inmediatamente por todas aquellas aplicaciones que entiendan y estén bajo esas normas. Esto nos lleva a tener solo una copia del dato, teniendo como beneficio que el dato es cargado solo una vez.

La desventaja de este método es que los datos deben de ser versátiles y genéricos, para que se acoplen a todas las aplicaciones, además hay todavía un gran camino por recorrer, para que sea adoptado por diferentes compañías y a la vez sea aceptado en el mercado. Actualmente POSC EPICENTRE está siguiendo este método de integración de datos.

### 2.2.1.4 Loose.

Para la integración de los datos propone un modelo estándar simple de dato con una codificación de datos específica , además para el intercambio datos propone unas ligas especiales llamadas Half-link, también se tiene que determinar un medio de transporte común, en el cual todas las aplicaciones que deseen trabajar, bajo este sistema deben de estar de acuerdo con los puntos antes mencionados.

Este método tiene una desventaja y es que necesita realizar varias copias de los datos que son solicitados por las diferentes aplicaciones. Esta técnica es adoptada por la herramienta Geoshare.

## 2.2.2 MÉTODOS PARA EL INTERCAMBIO DE DATOS ENTRE APLICACIONES.

Estos métodos se aprecian en la Figura 2.2.

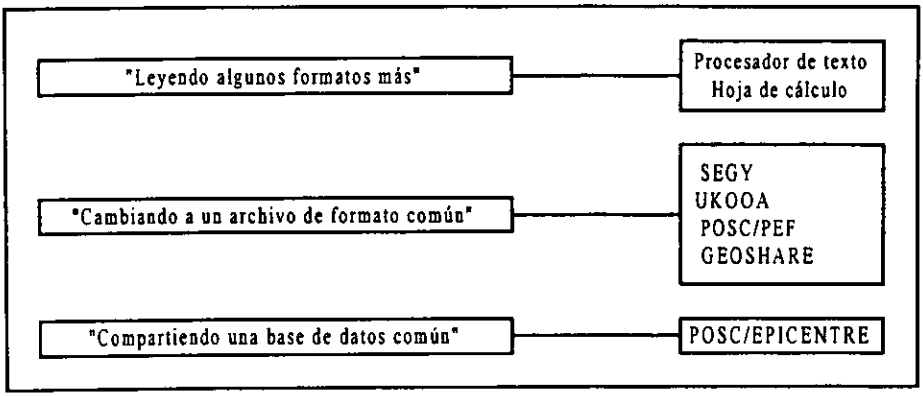



**Figura 2.2.- Métodos para intercambio de datos entre aplicaciones.**

### 2.2.2.1 "Leyendo Algunos Formatos Más".

Este método permite que los datos que se encuentren bajo cierto formato en una aplicación 'X' puedan ser leídas en otras aplicación 'Y'. Esto se da entre aplicaciones que pertenecen a la misma compañía y por ende la forma de almacenamiento de una aplicación a otra varia poco, estos casos son muy extraños que se den en la industria del petróleo y gas.


**Figura 2.3.- "Leyendo algunos formatos más".**

### 2.2.2.2 "Cambiando a un Archivo de Formato Común".

Este método es el más común que se ha venido utilizando dentro de la industria del petróleo y gas, y consiste en establecer un conjunto de reglas que bajo éstas, se logre tener un archivo con un formato común, para poder utilizar la información que contengan los archivos entre aplicaciones que se encuentren bajo esos estándares. Sin tomar en cuenta, la manera en que los datos se encuentran o son almacenados en diferentes aplicaciones, ya que sabemos que ellos varían de aplicación en aplicación.

Esta metodología es seguida por herramientas como SEGY, UKOOA, POSC/PEF, GEOSHARE. En el caso de Geoshare, ésta se encuentra apoyada por diferentes compañías que ya la utilizan.

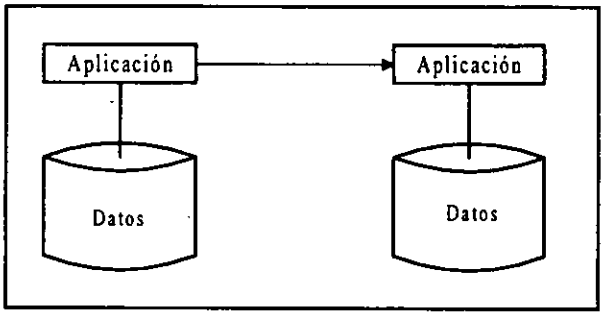



**Figura 2.4.- "Cambiando a un archivo de formato común".**

### 2.2.2.3 "Compartiendo una Base de Datos Común".

Esta metodología está surgiendo y se espera que próximamente sea utilizada por diferentes desarrolladores que se dedican al intercambio de datos entre aplicaciones. Consiste en tener un almacenamiento estándar, donde cualquier aplicación pueda accesar cualquier tipo de dato que se encuentre almacenado en este formato estándar. Actualmente sólo POSC/EPICENTRE, se encuentra desarrollando esta metodología, en realidad esta técnica ha sido utilizada desde años atrás, en el área de negocios, dando resultados satisfactorios, como es el caso de las bases de datos relacional.

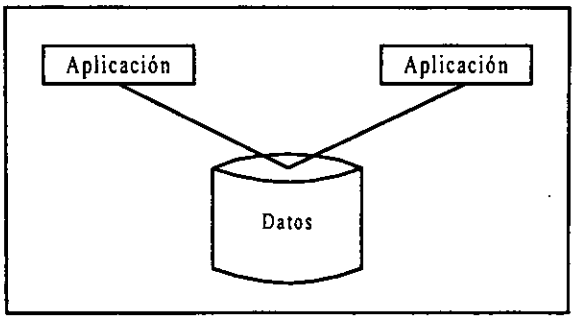



**Figura 2.5.- "Compartiendo una base de datos común".**

## 2.2.3 ¿QUÉ ES GEOSHARE?

### 2.2.3.1 Definición.

Es una herramienta estándar que sirve para transferir datos entre distintas aplicaciones de la industria del petróleo y gas, manteniéndolos en forma integrada.

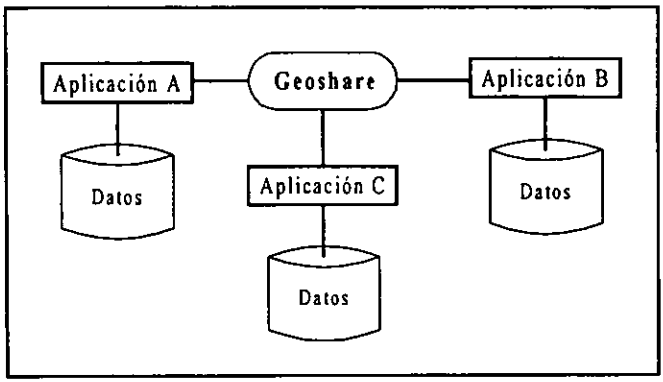



**Figura 2.6.- Herramienta Geoshare.**

### 2.2.3.2 Objetivo.

Tiene como objetivo mapear archivos de datos que se encuentran almacenados bajo la estructura y formato de cierta aplicación, a datos que sean modelados y formateados bajo los estándares de Geoshare conservándolos de manera integra, para poder moverlos de forma sencilla entre aplicaciones, desde el sitio exacto donde se encuentra el dato

seleccionado hacia el sitio correcto donde se hizo la petición de traslado, dando como resultado datos consistentes de cada transferencia realizada.

### 2.2.3.3 Geoshare y las partes que lo integran.

Un estándar es un conjunto de reglas o especificaciones por el cual el software debe ser construido para un propósito particular. En el caso de Geoshare, es un estándar para la integración. Existen dos partes del estándar Geoshare:

1. Datos : Escribe el formato, estructuras y relaciones de los datos.
2. Procesos: Define como los datos son transferidos desde una aplicación a otra y proporciona las especificaciones externas del programa de computación llamada half-links.

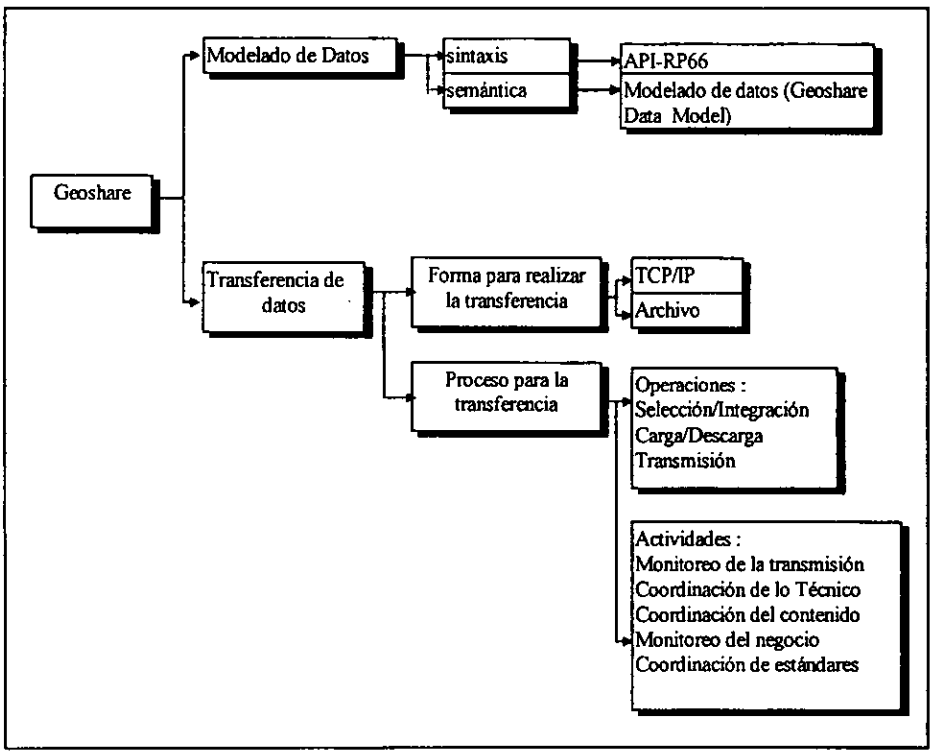



**Figura 2.7.- Geoshare y las partes que lo integran.**

## 2.2.4 MODELADO DE DATOS.

Esta herramienta, permite estructurar a los datos de tal forma que se haga fácil la identificación del tipo de dato. Esta integrado por dos partes :

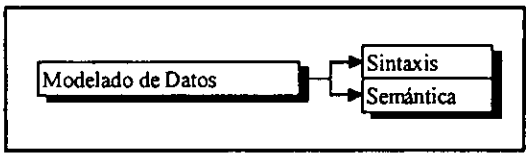



**Figura 2.8.- Modelado de datos.**

### 2.2.4.1 Sintaxis.

En esta sección se trata al dato desde el punto de vista de su almacenamiento, de como se va acomodar el dato en el archivo, teniendo en cuenta que la transferencia se realiza en bytes, así mismo la información descriptiva que debe de acompañarla al dato, y la manera que será guardado en el archivo.

Geoshare toma su sintaxis desde API (American Petroleum Institute), específicamente del RP66. API ha publicado un estándar para cambiar el registro de datos, cual contiene una sintaxis modificada para propósito de Geoshare. Este tiene algunos atributos deseables para esta aplicación, primero su misma descripción que acompaña a cada valor incluido en un archivo RP66 ya sea explícitamente o implícitamente está el nombre, tipo, formato y unidades. En esta parte se hace una descripción lógica del dato y se escribe físicamente a un archivo. La sintaxis es para objetos orientados permitido para grupos lógicos y jerarquía de datos, también es una arquitectura neutral en que algunos formatos de número son permitidos. Finalmente la sintaxis proporciona grandes volúmenes de transferencia de datos.

Algunas características del método de integración PAIR-WISE son ahora características de las bases del estándar Geoshare. Entre ellas esta el concepto de modelo de datos común, para la transferencia de datos.

### 2.2.4.2 Semántica.

Se refiere a como se va a modelar el dato, para ello Geoshare cuenta con una herramienta Modelo de Datos Geoshare (GDM "Geoshare Data Model") , el cual nos proporciona una forma simple para referenciar a diferentes tipos de objetos en un archivo . Este GDM proporciona dos objetos, uno es el EXECUTIVE que va ser la parte principal o raíz y a partir de aquí se puede referenciar a otros objetos, como un árbol, además este objeto contiene los diferentes tipos de objetos que van hacer intercambiados. El segundo objeto va proporcionar toda la información referente al proyecto (nombre, extensión).

GDM puede hacer referencia a los siguientes objetos de datos :

- Mediciones sísmicas.
- Campos de pozos.
- Grupos de superficie.
- Mapa de datos.
- Ondas.
- Perforaciones.
- Red de tierras.
- Lista de código litostratigráfica.
- Velocidad.
- Lista de zona geológica.

Además GDM cuenta con cerca de 138 tipos de grupos y puede transmitir 1500 datos desde una aplicación de petróleo o gas a otra.

## 2.2.5 TRANSFERENCIA DE DATOS.

Esta es una de las partes más importantes de esta herramienta. Básicamente este proceso estándar se encuentra constituida por dos partes, las cuales son :

- Mecanismos para la transferencia de datos
- Reglas para los programas Half -Link

### 2.2.5.1 Mecanismos para la transferencia de datos.

Es el proceso a través del cual, el dato estándar debe de pasar durante cada transferencia realizada, este proceso esta formado por cinco actividades y tres operaciones asociadas a una transferencia.

Operaciones :

1. Transmisión de archivos
2. Carga/Descarga
3. Selección/Integración

Actividades :

1. Monitoreo en la transmisión
2. Coordinación de la técnica
3. Coordinación de contenido
4. Monitoreo en el negocio
5. Coordinación de estándares

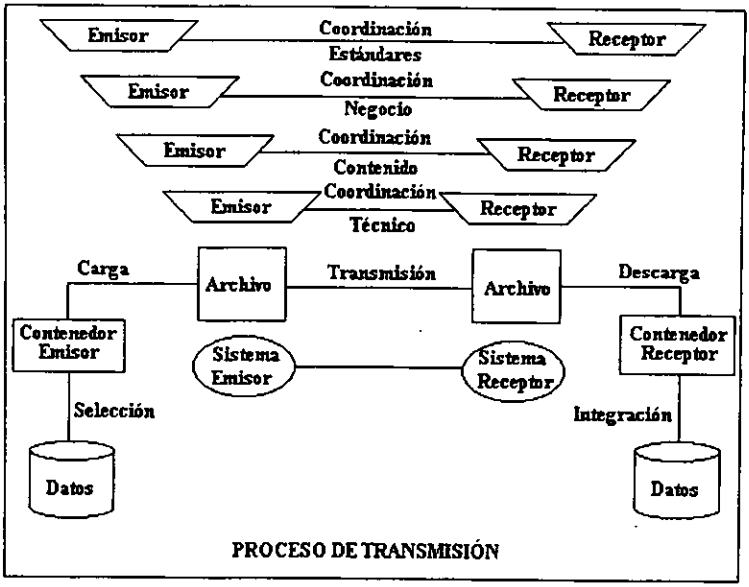



**Figura 2.9.- Fases de la transmisión.**

Los pasos que se deben de seguirse durante la transmisión de algún dato son los siguientes :

1. Selección:

   - Hay que seleccionar el contenido desde donde se encuentra almacenado el dato, que se va enviar (es más o menos fácil de realizarse).

2. Carga:

   - Se tiene que crear el archivo Geoshare.
   - Descargar el archivo, dentro del contenedor del que envía.

3. Transmisión:

   - Transmitir el archivo, consiste en pasar un archivo de un sitio llamado sender (el que envía) a otro llamado receiver (el que recibe). Por motivos de protección e integración de los datos, el archivo que esta siendo trasladado , no debe de ser accesado por nadie más.

4. Descarga:

- El archivo es recibido por el receptor.
- El dato del que envío es leído desde el archivo.
- Descargar el archivo dentro de un repositorio temporal.

5. Integración:

- Después de que el archivo es descargado desde el archivo transferido, debe de ser este trasladado e integrado dentro del almacenamiento del receptor (proceso difícil).

Nota : El software que ejecuta la carga y descarga es llamada Half-link.

### 2.2.5.2 Monitoreo de la transmisión

El archivo que está siendo transmitido, es monitoreado desde el inicio de la transferencia hasta que concluya ésta, se encarga de que el dato llegue bien y si hay algún error corregirlo o bien volverlo a enviar.

Las reglas para el monitores son definidos por Geoshare Ancillary Standars Comitte (ANSC) el cual es el foro para los usuarios Geoshare, para discutir todo lo relacionado al protocolo para realizar el monitoreo.

Coordinación de lo técnico

En esta fase se trata la manera de como se realiza la comunicación entre el que envía y el receptor en base a las opciones y parámetros de las especificaciones del estándar API-RP66.

Coordinación del contenido

En esta etapa de la comunicación entre el que envía y el que recibe, se determina cual contenido debe de ser transferido, sus características, así como sus anomalías y redundancias.

Monitoreo del negocio

Es la comunicación entre el que envía y el que recibe, se refiere a cualquier problema no técnico como puede ser la seguridad y derechos del propietario.

Coordinación de estándares

Se encarga de dirigir y coordinar a los estándares que soportan y son base de la transferencia (POSC, Geoshare, API-RP66).

### 2.2.5.3 Half-Link

Es una herramienta que realiza el cambio de formato que tienen los datos de una aplicación 'X' al formato de Geoshare, permitiendo el movimiento de los datos entre ellos. Este programa realiza la carga y la descarga, del dato transmitido, y se encuentra organizado en tres partes :

- Almacenamiento durante el trabajo, bajo una base de datos orientada a objetos.
- Librerías de Geoshare.
- Aplicación.

Donde las librerías de Geoshare se encarga de decodificar/codificar los requerimientos, bajo el estándar RP66 , proporcionando un área de trabajo en memoria RAM y una interfase hacia la aplicación donde se encuentra el dato almacenado bajo el formato de dicha aplicación.

### 2.2.5.4 Filtro Geoshare

Es usado como auxiliar de una Half-link, ya que su propósito es realizar una relación de cierto tipos de datos de información entre un sitio que envía información hacia otro que la recibe, que en base a ciertos requisitos o criterios se pueda seleccionar cierta información para transferir sólo la determinada información entre el que envía y el que recibe.

## 2.2.6 Tipos de datos que contiene un archivo Geoshare

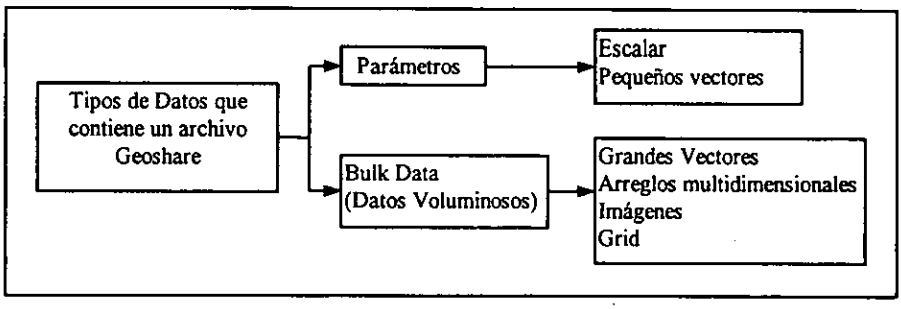



**Figura 2.10.- Tipo de Datos Geoshare.**

Hemos analizado hasta aquí como trabaja Geoshare en una manera general, en base a ello se puede decir que esta herramienta es útil en los casos cuando se tiene que mover datos entre aplicaciones, que no se encuentran almacenados bajo el mismo formato y no tienen una misma estructura, y lo principal es que estos datos contienen información de la industria del petróleo y gas (geo-referencias, datos técnicos o unidades de medición), que no se puede manejar como datos que provienen de empresas que se dedican a otro tipo de

negocios en donde éstos por lo regular no son muy voluminosos. Hay que destacar que las aplicaciones van a seguir guardando sus datos bajo sus estructuras y formatos, solo se hará el cambio a datos Geoshare cuando se pasan a otras aplicaciones.

Geoshare es una herramienta adecuada para alcanzar la integración, es estable y tiene un proceso de especificación fácil de entender, es soportada por diferentes compañías, actualmente cerca de 50 paquetes de software en la industria se comunican usando Geoshare estándar.

## *2.3 DLIS (Digital Log Interchange Standard).*

### 2.3.1 INTRODUCCIÓN.

Un tipo de formato es simplemente un grupo de reglas cual define como el dato esta organizado en una cinta o disco. DLIS (Digital Log Interchange Standard) es un formato estándar que fue elaborado por el API (American Petroleum Institute)[3], la cual se encarga de regular normas referentes al petróleo. DLIS es un formato para datos digitales de pozos.

Requerimiento para un formato:

1. Ser leído por alguna computadora sin importar su tamaño, poder o arquitectura interna.
2. Habilidad para registrar en la cinta información cual normalmente aparece en un registro de encabezado. Detalles acerca del pozo, nombre del cliente, parámetro, interpretación de parámetros .
3. flexibilidad.

### 2.3.2 CARACTERÍSTICAS GENERALES.

DLIS tiene las siguientes características:

- Los datos son representados acorde a su requerimiento dinámico. Tanto el tiempo y profundidad pueden ser intermezclado.
- La representación de datos es generalizada. Los arreglos multidimensionales, las cadenas de texto y nuevos tipos de datos pueden ser representados. Los datos pueden ser públicos o privados (definiciones administradas por el API o cualquiera).
- La información de origen pueden ser incrustada proporcionando mejor identificación de datos y hace fácil la identificación de tipos de datos duplicados.
- DLIS es superior y reemplazará a Log Information Standard (LIS). Proporciona mejor calidad e identificación de datos, un método general para representar forma compleja de datos y una técnica para la representación de canales de datos (Registrando curvas), acorde a sus requerimiento dinámicos.

DLIS es capaz de manejar una variedad de registro de datos de pozos desde herramientas comunes y herramientas de desarrollo, algunas exceden la capacidad de LIS. También

muchos datos de campo de petróleo, además de registros de datos de pozo (ejemplo: sísmicos, explotación, etc.), cumpliendo con el formato DLIS.

## 2.3.3 CONCEPTOS Y TERMINOLOGÍA.

DLIS define la estructura lógica de la información a ser registrada. El medio físico en el cual es registrada es de menor importancia. Aunque la estructura lógica debe adaptarse a los requerimientos y limitaciones de los dispositivos físicos. DLIS especifica algunos detalles de una unidad de cinta.

### 2.3.3.1 Dispositivos físicos y medios.

2.3.3.1.1 Unidad de almacenamiento.

Se refiere al usuario manejando entidades físicas que contiene el dato registrado. Aplicado a cintas magnéticas; unidades de almacenamiento refiriéndose a un simple cartucho o carrete de cinta. Aplicado a disco, refiriéndose a un simple archivo. En productos de cinta en MAXIS, la unidad de almacenamiento es referida por el nombre del dispositivo de la cinta (ejemplo: "MUAO").

2.3.3.1.2 Grupo de almacenamiento.

Se refiere a la colección de uno o más unidades de almacenamiento que contiene una estructura lógica entera (ejemplo: colección de uno o más carretes de cinta). El nombre del grupo de almacenamiento debe ser el mismo en todas las unidades de almacenamiento del grupo de almacenamiento.

### 2.3.3.2 Elementos de la estructura lógica DLIS.

2.3.3.2.1 Byte.

DLIS esta construido enteramente de 8-bit bytes, cual tiene significados y el lector sabe el código usado.

2.3.3.2.2 Representación de códigos.

DLIS define una serie de representación de códigos para identificar como codificar los bits encontrados en lo bytes o grupos de bytes usados para registrar información.

2.3.3.2.3 Registro lógico.

DLIS no supera el tamaño permitido de un registro. DLIS puede empacar múltiples registros lógicos pequeños dentro de un registro físico y puede romper la longitud del registro lógico dentro de segmentos que son puestos sobre múltiples registros físicos.

Hay dos clases de registros lógicos:

- Registros Lógicos Formateados Explícitamente (EFLR "Explicitly Formatted Logical Recorded").
- Registros Lógicos Formateados Indirectamente (IFLR "Inderectly Formatted Logical Recorded").

EFLR algunas veces es referido a un registro estático de datos, mientras que IFLR son algunas veces referidos como registros de datos dinámicos. Los registros lógicos tienen tipos de registros lógicos, los cuales son numerados y son referenciados por mnemónicos definidos por el estándar.

La Figura 2.11 muestra la estructura de un EFLR.

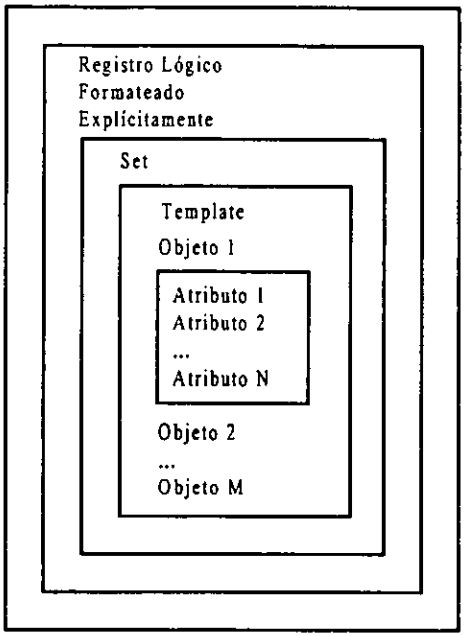



**Figura 2.11.- Estructura de un EFLR.**

2.3.3.2.4 Archivo lógico.

Un Archivo lógico consiste de una secuencia de registros lógicos que empiezan con un encabezado de archivo de un registro lógico (FHLR "File Header Logical Record" ) y finaliza cuando no hay datos que agregar o cuando un nuevo FHLR es encontrado. Un archivo lógico "Es una útil colección de datos relacionados". Un simple archivo lógico puede ser contenido enteramente en una unidad de almacenamiento o puede empezar en uno

o terminar en otro. Múltiples unidades de almacenamiento constituyen un SET de almacenamiento. En la cinta un archivo físico debe contener todo aparte de un simple archivo lógico (datos contenidos entre marcas), sin embargo un archivo físico no contiene marcas de archivo y debe contener uno o más archivos lógicos.

EFLR: Es lógicamente el mismo almacenamiento, contiene toda la información necesaria acerca del dato, tal como su nombre, unidades, código de representación, etc.

2.3.3.2.5 Objeto.

Un objeto DLIS es un dato que tiene un nombre y algunos atributos y pertenece a un particular tipo. Un objeto es un renglón en una tabla con su nombre en la primera columna. Ejemplo el objeto Channel (Canal), tiene los siguientes atributos:

- Long - Name (Longitud del nombre).
- Properties (Propiedades).
- Representation code (Código de representación).
- Units (Unidades).
- etc.

El nombre de un objeto es realmente una triple consistencia de un origen, un número de copias y un identificador (por ejemplo: RMF). Algunos identificadores son diccionarios controlados esto no es hecho en el transcurso pero debe ser registrado en una Base de Datos central que es mantenida por el API o por el productor (ejemplo: Schlumberger). Otros identificadores son creados en el transcurso (ejemplo identificadores del objeto Frame). El número de copia es proporcionado como un significado para hacer nombres de objetos únicos con un archivo lógico. Ejemplo: Si dos GR canales son producidos durante una pasada de una herramienta simple de cadena, ahí tendría dos objetos con el mismo origen y mismo identificados (GR), pero ello deben de obtener diferente número de copia. El completo nombre de esos objetos debe ser [22, 0, GR] y [22, 1, GR] sólo la tripleta debe ser única. El número de copias e identificadores pueden ser replicadas (excepto el origen).

2.3.3.2.6 Set.

Es un dato que tiene un tipo y opcionalmente un nombre y contiene una colección de un mismo tipo. Si un objeto representa un renglón en una tabla entonces Sets representa la tabla entera. Cada EFLR contiene solamente un Set. Hay una perdida de conexión entre tipos de Set y tipos de registros lógicos. Ciertos tipos de Set deben de aparecer solamente en designados tipos de registros lógicos. Algunos tipos de registros lógicos deben de contener uno de muchos de tipos de Set.

Algunos tipos Set son:

- File Header: Es el primer Set en un archivo lógico y marca el inicio y fin de un archivo lógico. Es contenido en un encabezado de archivo de registro lógico (mnemónico FHLR) y contiene un objeto que tiene dos atributos, que se refieren al número de secuencia y al identificador único (id).

- ORIGIN: Los Sets son contenidos en un registro lógico de origen (mnemónico OLR) y contiene uno o más objetos ORIGIN. Cada objeto ORIGIN contiene atributos que describen las condiciones debajo del cual el dato en el archivo lógico fue adquirido o creado, incluyendo tales casos como compañía, pozo, campo, nombre del productor, etc.
- Objetos Parameter: Contiene descripción de parámetros. Ellos contiene tal información como longitud de nombre, zonas, dimensión, eje, valores, unidad, etc.
- CHANNEL: Esta contenido en un registro lógico de canal (mnemónico CHANNEL). El objeto Channel contiene información definiendo y caracterizando canales de datos de registros de pozo. Incluyen longitud de nombres, unidades, código de representación, dimensión, eje.
- Frame: Están contenidos en registros lógicos de Frame (mnemónico FRAME). El objeto Frame identifica y describe un tipo de Frame, cual es una secuencia de IFLR. El objeto Frame describe el tipo de índice de Frame en cuanto a dirección, espaciamiento, etc. El objeto Frame y los objetos Channel contienen toda la información necesaria para describir como el dato de las muestran son escritos en otros registros.

### 2.3.3.3 Tratamiento de datos voluminosos.

Es implementado vía RP66 con objetos Channel. La mayoría de las estructuras Geoshare son grabados como objetos RP66 en EFLRs. Para cada campo en la estructura hay correspondiente atributo en el objeto. Cierto campos contiene arreglos que deben ser de suficiente tamaño al ser considerado Datos voluminosos. Estos campos, antes que sean registrados explícitamente como valores de atributos, en su lugar son registrados como canales.

Múltiples Correlaciones de Objetos Channel son referenciados desde un común objeto Frame. Cuando hay múltiples relaciones uno a uno entre múltiples campos voluminosos de una estructura los correspondientes canales son ajustados dentro de un simple tipo de Frame.

En términos de Base de Datos relacional, un objeto Frame es como la descripción de una tabla, una colección de renglones y columnas de datos. Un objeto Channel es como la descripción de una de las columnas de datos en la tabla. Un Frame es como un renglón en la tabla y Channel es como una columna en la tabla.

Algunos canales son opcionales. Cuando un objeto tiene múltiples campos voluminosos, solamente aquellos campos que están actualmente disponible serán registrados.
Grandes y pequeños Frames. Los canales que contiene datos voluminosos de Geoshare son representados de dos maneras, dependiendo del tamaño supuesto del dato.

1. Cuando el tamaño del campo de un dato voluminosos es menor a algunos límites predefinidos, por lo tanto el campo estará representado por un canal teniendo una no escalar muestra. Todo el dato del campo es registrado en un sólo Frame.
2. Un campo que tiene un tamaño que excede los límites es representado por un canal teniendo muchas muestras donde cada una es un simple valor escalar. En este caso el dato está registrado en varios Frames, cada Frame contiene sólo un valor de dato.

## 2.3.4 ALMACENAMIENTO EN DISCO O CINTA MAGNÉTICA.

### 2.3.4.1 Formato físico.

Archivo físico va estar formado por los siguientes elementos:

- BOT: Marca de inicio.
- ETW: Marca de fin.
- TM: Marca indeleble.
- SUL: Etiqueta de la unidad de almacenamiento.
- LF: Archivo Lógico.

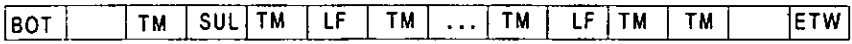



**Figura 2.12.- Archivo Físico.**

### 2.3.4.2 Formato lógico.

Archivo Lógico (LF "Logical File"), consiste de una secuencia de una o más registros lógicos empezando con un encabezado (FHLR "File Header Logical Record"). Un archivo lógico es terminado cuando otro FHLR es encontrado o cuando no más registros lógicos están disponibles para el archivo lógico.

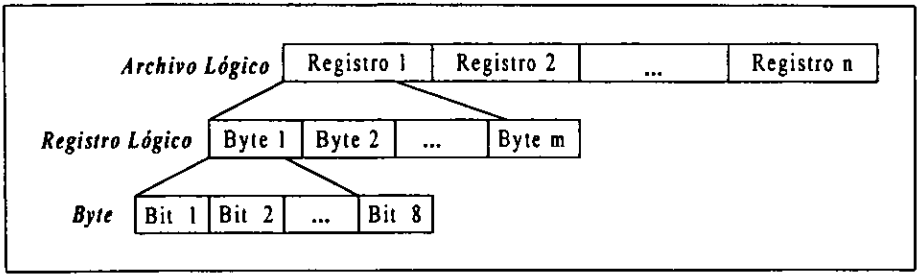



**Figura 2.13.- Formato Lógico.**

Archivo Lógico esta integrado por uno o más registros lógicos.

LRi: Registro Lógico.

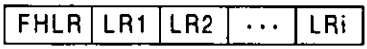



**Figura 2.14.- Archivo Lógico (LF).**

Un Registro Lógico esta formada por:

- VRL: Longitud del Registro visible.
- FV: Versión del Formato.
- LRSi : Segmento de Registro Lógico.

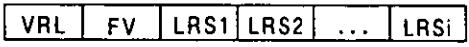



**Figura 2.15.- Registro Lógico (LR).**

Un Segmento de Registro Lógico va estar formado por los siguientes elementos:

- LRSH: Encabezado de Segmento del Registro Lógico.
- LRSB: Cuerpo de Segmento del Registro Lógico.
- LRST: Rastrero del Registro Lógico.


**Figura 2.16.- Segmento de Registro Lógico (LRS).**

### 2.3.4.3 Organización de datos en un segmento de registro lógico.

Un segmento de registro lógico está compuesto por tres partes. Estas se describen a continuación.

2.3.4.3.1 LRSH (Encabezado de Segmento del Registro Lógico).

Es el encabezado de segmento del registro lógico, cuenta con tres atributos:

1. Longitud del segmento del registro lógico.
2. Atributos del segmento del registro lógico.
3. Tipo de registro lógico.

La longitud del segmento del registro lógico, especifica la longitud en bytes del segmento de registro lógico, es almacenada en dos bytes no signados. Un LRS debe contener al menos 16 bytes para facilitar el mapeo de un formato lógico a un formato físico.

Atributos del segmento del registro lógico, consiste de un byte bit string que especifica los atributos del segmento de registro lógico. Su estructura es definida en la Figura 2.17.

Tipo de registro lógico, es un byte, entero no signado (Código de representación USHORT) que especifica el tipo de registro lógico. Su valor indica la semántica general contenida del registro lógico. El mismo valor debe ser usado en todos los segmentos de un Registro lógico.

2.3.4.3.2 LRSB (Cuerpo de Segmento del Registro Lógico).

Es un grupo ordenado de bytes (byte = 8 bits), es la parte medular de un registro lógico, dado que la información que contiene es la esencial de un determinado objeto.

2.3.4.3.3 . LRST (Rastrero del Registro Lógico).

Este registro sigue al LRSB y consiste de una combinación de los atributos padding, checksum, y/o trailing length.

En la Figura 2.18 se aprecia la estructura de un conjunto de segmentos de registro lógico.

| Bit | Descripción |
|---|---|
| 1 | Estructura de un registro lógico<br>0 = Registro Lógico Formateado Indirectamente.<br>1 = Registro Lógico Formateado Explícitamente. |
| 2 | Predecesor<br>0 = Este es el primer Segmento del Registro Lógico.<br>1 = Este no es el primer Segmento del Registro Lógico. |
| 3 | Sucesor<br>0 = Este es el último Segmento del Registro Lógico.<br>1 = Este no es el último Segmento del Registro Lógico. |
| 4 | Encriptación<br>0 = No encriptación.<br>1 = Registro Lógico está encriptado. |
| 5 | Paquete de encriptación<br>0 = No hay un paquete de encriptación para el Segmento de Registro Lógico.<br>1 = Hay un paquete de encriptación para el Segmento de Registro Lógico. |
| 6 | Checksum<br>0 = No checksum.<br>1 = Un checksum está presente en el LRST. |
| 7 | Longitud del Trailing<br>0 = No hay longitud del Trailing.<br>1 = Una copia de la longitud LRS está presente en el LRST. |
| 8 | Padding<br>0 = No esta registrado el padding.<br>1 = Los bytes del Pad están presentes en LRST. |

**Figura 2.17.- Atributos del Segmento de Registro Lógico.**

| | | | | | |
|---|---|---|---|---|---|
| LRSL | 10100110 | body | Checksum | Trailing | LRSL |
| LRSL | 11100110 | body | Checksum | Trailing | LRSL |
| LRSL | 11000111 | body Padding | Checksum | Trailing | LRSL |

**Figura 2.18.- Tres Segmentos de Registro Lógico.**

# 3. CARACTERÍSTICAS DE LA PERSISTENCIA DE OBJETOS.

## *3.1 TECNOLOGÍA ORIENTADA A OBJETOS (TOO).*

### 3.1.1 INTRODUCCIÓN.

Esta tecnología tiene 30 años de existir. Simula fue el primer lenguaje orientado a objetos, y fue éste el que introdujo los conceptos del enfoque orientado a objetos[4]. Actualmente la TOO está logrando llamar la atención de varios sectores de la industria, la razón es que está alcanzando un alto potencial, al proporcionar grandes beneficios a aplicaciones desarrolladas bajo sus normas.

Los sistemas creados bajo las técnicas tradicionales se han encontrado con problemas, en el momento de integrar grandes volúmenes de información. Estos problemas van desde un crecimiento considerable en el código, lo cual a su vez crea problemas de entendimiento, así como los tiempos de programación y terminación de los sistemas van en aumento a raíz de lo mismo. En cambio si aplicamos TOO la complejidad y el crecimiento en el código es combatido gracias al concepto de reuso[5].

### 3.1.2 CARACTERÍSTICAS PRINCIPALES.

La TOO ofrece un desarrollo incremental, altos niveles de reuso y fácil mantenimiento, que hacen que luzca atractiva, para la próxima generación de aplicaciones de sistemas de información. Sin embargo, esto implica un concepto nuevo, en donde las dificultades de aprender TOO ha sido un significativo obstáculo para su adopción, como una corriente de tecnología de desarrollo. Se cree que los objetos son complejos, pero en realidad son una similitud del mundo real y esto no implica que el desarrollador deba conocer la estructura interna de los objetos, en ocasiones el desarrollador sólo tiene que limitarse a conocer su comportamiento y la manera de usarlo.

La tecnología orientada a objetos cuenta con las siguientes características:

- Abstracción de datos
- Encapsulación
- Herencia
- Reutilización del código.
- Semántica entendible.
- Fácil de modificar.

### 3.1.3 METODOLOGÍA TRADICIONAL VERSUS. METODOLOGÍA CON ORIENTACIÓN A OBJETOS.

| METODOLOGÍA POR PROCEDIMIENTOS | METODOLOGÍA CON ORIENTACIÓN A OBJETOS |
|---|---|
| Formado por procedimientos y datos. | Formado de objetos que tienen atributos y métodos. |
| Por medio de rutinas aisladas se logran efectuar tareas especificas. | A través de una clase, es posible ofrecer una serie de servicios a objetos de un tipo particular. |
| Descomposición funcional basada en subprocesos de niveles superiores. | Composición de clases basada en abstracción de datos. |
| Se tienen módulos secuenciales interdependientes que comparten datos. | Por medio de los métodos de los objetos, se define un conjunto de módulos de comunicación independientes, pero con visibilidad limitada entre ellos. |

**Figura 3.1.- Metodologías.**

### 3.1.4 DEFINICIÓN DE TÉRMINOS EN TOO.

*Objeto*: Es una entidad que va tener características y conducta.

*Clases:* Son tipos de datos definidos por el usuario que definen un tipo de objeto. En ella se van a definir los métodos y variables de un especifico tipo de objeto. Un objeto es un modelo o instancia de ese tipo o clase.

*Instancia*: Este término es utilizado para definir que un objeto pertenece a una determinada clase.

*Método*: Es un conjunto de operaciones que residen en el interior de un objeto que van a definir la conducta del objeto.

*Mensaje*: Es la solicitud que hace un objeto a otro objeto, en el cual se le indica que actúe de cierta forma.

Para cada objeto se tiene que describir su conducta con métodos y sus características como variables. Los objetos hacen excelente software modulado por que ellos pueden ser definidos y mantenidos independientemente de uno y otro. Todas las cosas que se "saben" de un objeto son expresados en su variable. Todas las cosas que pueden hacerse son expresadas en sus métodos.

En términos de programación, se puede decir que una clase , es como un tipo de dato, un objeto es como una variable[6]. Ver Figura 3.2.

| PROGRAMACIÓN ORIENTADA A OBJETOS (POO) | PROGRAMACIÓN ESTRUCTURADA |
|---|---|
| Clase | tipo de dato |
| Objeto | variable |

**Figura 3.2.- Tipos de Programación.**

El proceso de programación en un lenguaje orientado a objetos se caracteriza por:

- Crear clases que definen la representación y el comportamiento de los objetos.
- Crear objetos a partir de la definición de la clase.
- Realizar la comunicación entre objetos a través del envío de mensajes (invocación a las funciones miembros).

### 3.1.5 PROPIEDADES FUNDAMENTALES DE POO (PROGRAMACIÓN ORIENTADA A OBJETOS).

1. Abstracción de datos.
2. Encapsulación y ocultación de datos.
3. Herencia.
4. Polimorfismo.
5. Reutilización.

*Abstracción de datos:* Es la capacidad para examinar algo sin preocuparse de sus detalles internos.

*Encapsulamiento y ocultación de la información*: Previene a otros objetos ver su interior, donde el comportamiento de la abstracción se ha realizado. En la práctica se oculta la representación de un objeto, así como la realización de sus métodos.

*Herencia:* En POO las clases se pueden subdividir en subclases, se pueden deducir o derivar de otra clase. La clase original se denomina clase base y tiene la propiedad de heredar a una clase derivada o descendiente puede heredar las estructuras de los datos y los métodos de su clase base.

Dos tipos de herencia:

- Simple: Una clase sólo puede tener un ascendente.
- Múltiple: Un objeto puede tener más de un descendente.

*Polimorfismo:* Es la propiedad que permite a una operación tener diferentes comportamientos en objetos diferentes, es decir objetos diferentes reaccionan de diferente manera al mismo mensaje. El polimorfismo requiere herencia. Nos permite tratar objetos de un tipo diferente de modo similar.

El polimorfismo se aplica únicamente a los métodos heredados en una clase base. Cuando los métodos polimórficos se utilizan, el compilador no puede determinar que función llamar. La función específica llamada, depende de la clase del elemento a la cual se envía el mensaje; en consecuencia, la función a llamar se determina en tiempo de ejecución. Esta operación se conoce como ligadura tardía o postergada ya que ocurre cuando el programa se está ejecutando. La ligadura temprana, anticipada o previa ocurre para métodos no polimórficos.

*Reutilización:* Una vez que una clase se ha escrito, creado, depurado y comprobado, se puede distribuir a otros programas para utilizarla en sus propios programas.
El concepto de herencia en POO proporciona una extensión importante a la idea de reutilización.

## *3.2 PERSISTENCIA.*

### 3.2.1 CLASE DE DATOS.

Al realizar programas se cuenta con dos partes, por un lado se tiene las instrucciones y por el otro los datos, además de apoyarse de los lenguajes de programación. Los datos son los que juegan el papel más importante, ya que después de ser procesados arrojan información, la cual es muy valiosa para tomar decisiones.

Actualmente los lenguajes de programación, manejan los datos de tal manera que estos solo están presentes durante la existencia del programa, es decir, al momento de dejar de existir el programa, se pierden los datos, a no ser que se guarden en un archivo de entrada/salida, logrando que existan más allá de la vida de un programa.

Clase de datos en un programa de aplicación:

a) Dato Volátil: Tiene las siguientes características:

- Es creado por un lenguaje de programación.
- Es manipulado por un lenguaje de programación.
- Reside en memoria RAM.
- Deja de existir cuando el programa termina.

a) Dato Persistente: Tiene la característica :

- Reside en almacenamiento permanente.
- Es trasladado de almacenamiento permanente a memoria RAM para poder ser accesada y manipulado. Lo cual requiere una considerable fracción de código.
- Realiza el almacenamiento y restauración del dato de manera automática.

Con lo anterior queda clara la diferencia entre un dato volátil y un dato persistente, llevándonos al siguiente plano que son los beneficios que conlleva utilizar un sistema que maneje datos persistentes. Desde el punto de vista de un usuario la ventaja principal y más clara es que al terminar una sesión con la computadora, el usuario no debe de ocuparse de guardar los datos, y al momento de reanudar una sesión tampoco debe de ocuparse en restaurar los datos, dando la impresión de que los datos no se han movido.

### 3.2.2 ¿QUÉ ES PERSISTENCIA?

Persistencia es la habilidad de los objetos a vivir más halla de la vida de los programas que los crearon, significa la posibilidad de almacenar objetos en almacenamiento no volátil, permitiendo al programador leerlo más tarde en memoria principal, para un rápido procesamiento aún después de que el programa que creó ese objeto ya no exista. Dotar a los objetos de persistencia significa dotarlos de la capacidad para que, de alguna forma, puedan escribir y leer por ellos mismos.

El objeto persistente debe tener capacidad para extraer de si mismo su contenido, a partir de un flujo de entrada generalizada y también de insertar su propio contenido en un flujo de datos de salida generalizada.

### 3.2.3 ¿PARA QUÉ SE USA?

Para aplicaciones que necesitan guardar información sobre su configuración y estado interno al final de cada sesión, con objeto de que en la próxima sesión se pueda continuar a partir del mismo punto en donde se detuvo el trabajo.

Para aplicaciones que deben interactuar con objetos que varían y se extienden a través del tiempo.

Una motivación para los lenguajes de programación persistente es para manejar los programas de aplicaciones de las bases de datos, del considerable esfuerzo normalmente requerido para transferir datos entre el sistema de almacenamiento de la base de datos y su programa. En la programación de una aplicación de una base de datos con un convencional lenguaje de programación, una gran cantidad de código es dedicada para la ejecución de traslación entre la forma de un dato en el programa (arreglos, registros, etc.), a la forma requerida para un almacenamiento externo (operando sistemas de archivos o encajando por interfase a un sistema de base de datos.

Para aplicaciones que manejan grandes volúmenes de información es adecuada para mantener la información disponible en cualquier momento.

- Para datos complejos.
- Productos con aplicaciones multimedia.

- Modelos económicos.
- Sistema de manejo de documentos.
- Sistemas Cliente/Servidor.
- CAD.
- Ingeniería.
- Sistemas de manufacturación.

Por ejemplo: Para el pronóstico de la economía, el modelo debe requerir objetos que reflejen instantáneamente la condición de la bolsa de valores. El estado del objeto bolsa de valores es descrito por sus atributos, debe persistir por unos pocos segundos debe antes ser reemplazado.

### 3.2.4 CATEGORIAS DE PERSISTENCIA.

Se tiene 3 categorías de persistencia[7]:

- Sesión persistente: La memoria asociada con una sesión es guardada.
- Programa persistente: La memoria asociada con la ejecución de un programa es guardada.
- Objetos persistentes: Objetos que son persistentes.

## *3.3 PRINCIPIOS DE PERSISTENCIA EN POO.*

Cuando se incorpora persistencia en un lenguaje de Programación Orientado a Objetos Los siguientes principios deben de ser considerados[8].

- La persistencia es una propiedad de instancias de objetos y no de tipos de objetos. Esto es, un tipo de objeto dado (o clase), debe de tener ambas instancias persistentes y volátiles.

- Los objetos de algún tipo, incluyendo arbitrariamente tipos complejos definidos por usuarios, deben de ser distribuidos en cada persistencia o volátil almacenamiento.

- Los objetos persistentes y volátiles deben de ser accesados y manipulados exactamente de la misma manera. Esto es el código del programa es expresado independientemente de la persistencia de los objetos que se accesa y manipula. Es decir Una función A con objetos de tipo 0 como parámetro debe de ser posible llamar a A con ambas instancias persistentes y volátiles con 0 como actual parámetro.

  - Debe de ser un mecanismo simple para el movimiento de objetos desde almacenamiento persistente a almacenamiento volátil y viceversa.

## *3.4 IDENTIDAD DE UN OBJETO PERSISTENTE.*

La identidad del objeto, es una importante propiedad que es fundamental al realizar un modelo persistente.

### 3.4.1 IMPLEMENTACIÓN DE LA IDENTIDAD DE UN OBJETO.

La implementación de la identidad de un objeto en una Base de Datos Orientado a Objetos envuelve la porción de 3 clases de independencia.

- Independencia de Localización: Implica que la identidad de un objeto es preservada a despecho de que cambie en su localización física si es almacenamiento volátil o persistente.
- Independencia de Valor: Envuelve la preservación de identidad de objetos a través de cambios en su valor.
- Independencia de Estructura: Envuelve la preservación de la identidad de objetos a través de cambios en su estructura.

### 3.4.2 IDENTIDAD A TRAVÉS DE DIRECCIONES.

Identidad a través de dirección física (real o virtual), proporcionado por los lenguajes de programación convencional, no permiten que un objeto sea movido para mantener su identidad. Por lo tanto no permite tener una localización independiente.

Ejemplo: En Pascal el identificador de un objeto debe ser un apuntador al objeto cual es implementado como una pila virtual de direcciones. El objeto no debe ser movido fuera sin actualizar el apuntador. No obstante la dirección física real y la dirección virtual proporcionan valores y estructuras independientes.

En Smalltalk 80 el identificador de un objeto es implementado en principio como un apuntador, el cual no es una dirección virtual del objeto, pero es una entrada a una tabla objeto, equivale a una dirección indirecta y tiene la ventaja de permitir a los objetos individuales, moverse con un solo espacio de direcciones, sin la necesidad de cambiar su identificador. También cae en valores y estructuras independientes, el esquema no nos permite el compartimiento de objetos.

### 3.4.3 IDENTIDAD A TRAVÉS DE LLAVES.

Debe generarlo el sistema.

Las Relaciones son almacenados en archivos físicos y en estructuras auxiliares tal como un índice Arbol-B, permite accesar rápido a objetos en base al valor de su llave tal implementación permite localizaciones independientes.

Si una tupla cambia de posición en el archivo sólo el índice es actualizado y no el identificador de la tupla (valor de la llave). Si cambia su valor su identificador debe ser cambiada.

No soporta estructuras independientes, desde que valores de las llaves son únicos para tener una relación.

Si el usuario cambia el valor de una llave el debe asegurarse que la integridad de la referencia esté asegurada, es decir que todas las instancias de aquella llave en otra relación son cambiadas.

### 3.4.4 IDENTIDAD A TRAVÉS DE SURROGATES (SUSTITUTOS).

Son sistemas generados globalmente como identificadores únicos, independientes de la localización física. Por lo tanto ofrecen caer en localizaciones independientes. Además, si un surrogate es asociado con cada objeto a pesar de su complejidad, proporciona valores y estructuras independientes. Así cada objeto de algún tipo es asignado a un globalmente único surrogate y su surrogate representa la identidad del objeto a lo largo de su vida. El generado valor surrogate es único con respecto a todos sus surrogates, que existen o han existido en una particular base de datos.

Esto está garantizando a no cambiar aun si el objeto es removido de la base de datos. De esta manera es posible reinstalar un objeto desde el archivo de almacenamiento sin comprometer la integridad de la base de datos. Los usuarios no tienen control sobre los surrogates. Ellos no deben accesar sus valores y de hecho no debe ser enterado de su existencia. Esto previene a los usuarios de introducción de referencia de objetos estén fuera del control del sistema y por lo tanto propenso a errores ambigüedades. En OODB todos los identificadores de objetos y las referencias a otros objetos son vía surrogates.

## *3.5 PERSISTENCIA EN UN LENGUAJE DE PROGRAMACIÓN DE BASE DE DATOS.*

Los lenguajes de programación de base datos han adoptado dos contrastantes enfoques:

- Primer enfoque: (Pionero Lenguaje Pascal/R [Schmidtt, 1977]) es proporcionar una estrecha relación del modelo base de datos relacional con un lenguaje de programación convencional. Las variables de tipo relacional deben de persistir y agregar una estructura de control, basado en el álgebra relacional, que debe de agregarse para

accesar y manipular en forma relacional. De estos lenguajes: Astral, Rigel, Theseus, Plain, Modula/R, RAPP.

- Un segundo grupo de lenguajes adopta un elaborado modelo de persistencia, en el cual los valores deben de persistir, independientemente de su tipo y los objetos persistentes son sujetos a los mismos rigurosos tipos de chequeo como variables de programa. Ejemplo de estos lenguajes son: PS-algol, Amber, Poly y Galileo.

### 3.5.1 PERSISTENCIA EN PASCAL/R.

Sólo las variables de tipo base de datos pueden persistir
La definición de un tipo relacional requiere dos parámetros:

- Un tipo de registro el cual define la estructura de la tupla de la relación.
- El subgrupo de campos del registro el cual contiene la llave primaria.

### 3.5.2 PERSISTENCIA EN PS-ALGOL.

La persistencia es una propiedad ortogonal de dato[9]. Esto es que algunos objetos de datos sin hacer caso del tipo, deben de existir más allá de la vida del programa el cual los creó. PS-Algol no envuelve una nueva estructura pero mejor que eso proporciona persistencia como una propiedad aplicable a todas las estructuras de datos existentes. PS/Algol proporciona procedimientos para manipular tablas incluyendo procedimientos para insertar una pareja en una tabla.

## *3.6 PERSISTENCIA EN UN SISTEMA ORIENTADO A OBJETOS.*

En un ambiente Orientado a Objetos se debe visualizar a la memoria como dos distintas partes: volátil y almacenamiento persistente.

Un OODB (Base de Datos Orientado a Objetos) es una colección de objetos persistentes, los cuales se identifican por un único identificador el cual debe ser visualizado como un identificador a un objeto persistente.

El almacenamiento de un objeto persistente depende de la implementación y la interacción con esos objetos. Son manipulados por un manejador de almacenamiento de objetos.

El acceso a objetos persistentes debe de ser tan rápido como el acceso a objetos volátiles, pero en la práctica ellos son más lentos.

La implementación debe de ser rápida para accesar a objetos persistentes y debe de ser transparente al programador.

## *3.7 PERSISTENCIA A TRAVÉS DE ARCHIVOS.*

El enfoque tradicional de persistencia en programas de aplicación es el uso de sistemas de archivos[10]. Esta técnica a sido usada por décadas. Las aplicaciones usan simples comandos de lectura y escritura para recuperar o colocar registros desde/hacia estructuras de archivos. Esta tecnología permite registrar datos en un medio persistente.

Existen tres fundamentales clases de métodos de acceso para leer registros desde un almacenamiento persistente:

- Secuencial.
- Directa.
- Indexada.

Es responsabilidad del programador del objeto transformar la representación de los objetos en memoria a estructuras de datos que puedan ser almacenados, usando técnicas de organización de registros y métodos de accesos en sistemas de archivos. El código para escribir un objeto a un archivo incluye transformaciones. Algunas transformaciones son triviales, cada dato miembro del objeto es meramente escrito como un campo de un registro. Pero son más comunes las transformaciones que son complejas.

Si un objeto en C++ va hacer salvado y éste apunta a otro objeto, la referencia debe ser salvada. Si la referencia es transitiva no debe de ser guardada en forma persistente, debe de ser transformada a NIL. Si una referencia a un objeto es almacenada persistentemente, entonces aquel apuntador debe de ser transformado a algún valor que esté disponible, para que la aplicación pueda leer y reconstruir la referencia. Si la transformación para escribir el objeto va a cambiar, entonces la transformación para leer el objeto debe cambiar.

En el enfoque de persistencia a través de archivos, los programadores deben escribir y mantener el código que transforme un objeto a /desde registros. El diseño de persistencia a través de un archivo así como la implementación es una tarea no trivial.

## *3.8 OTRAS FUENTES DE PERSISTENCIA.*

### 3.8.1 ODBMSs.

ODBMSs es algunas veces llamado Lenguaje de Programación de Base de Datos. Este sistema se extiende a un lenguaje de programación de objetos para proporcionar persistencia, control de concurrencia, consultas y otras habilidades.

El ODBMS hace que la referencia entre objetos tenga una localización independiente porque la localización de un objeto puede cambiar. El ODBMS proporciona un grado de aislamiento para el almacenamiento físico, el objeto no debe de saber con que método se le

va a accesar, que clase de máquina ni donde. El almacenamiento es separado a través de múltiples estructuras físicas.

Las ventajas son compartición y protección de objetos.

### 3.8.2 RDBMSs.

El programa objeto usa registros como si fueran objetos. Cuando un RDBMS es usado para persistencia, el programador de objeto debe de utilizar un lenguaje (SQL) para definir, accesar y manipular datos persistentes y otros.
Algunos vendedores de RDBMS están evolucionando sus productos para mejorar la incorporación de la tecnología de objetos. Estos sistemas son llamados Modelo Relacional Extendido ellos realizan las RDBMS para incorporar extensos tipos de datos, procedimientos, herencias y otras habilidades. Estas extensiones no han sido formalmente estandarizadas.

Un RDBMS Extendido puede ser un enfoque de elección para objetos persistentes en un ambiente mixto de Base de Datos Relacional y de Objetos.

Fuentes alternativas de objetos persistentes, se ven en la Figura 3.3.

| | Sistema de Archivos | Smalltalk Imagen | ODBMS | RDBMS | RDBMS Extendido |
|---|---|---|---|---|---|
| No tiene dinero para DBMS | Si | Si | No | No | No |
| Usando C++ | Si | No | Si | Si | Si |
| Usando Smalltalk | Si | Si | Si | Si | Si |
| Usando COBOL | Si | No | No | Si | Si |
| Usando C | Si | No | Si | Si | Si |
| Objetos que deben de ser accesados concurrentemente | No | No | Si | Si | Si |
| Objetos que son muy voluminosos para memoria principal | No | No | Si | Si | Si |
| Quieren integración sin costuras | No | No | Si | No | No |
| Necesitan Base de Datos distribuidas | No | No | Si | No | Futuro |
| Necesitan compatibilidad con SQL | No | No | Futuro | Si | Si |
| Necesitan funcionamiento con objetos altamente interrelacionados. | No | No | Si | No | Futuro |

**Figura 3.3.- Tabla comparativa de fuentes alternativas de objetos persistentes.**

## *3.9 NIVELES DE PERSISTENCIA.*

Un objeto tiene uno de cuatro niveles de persistencia[11]. Estos niveles son:

1. No persistencia: No hay mecanismos para almacenar y recuperar objetos.
2. Persistencia Simple: Un nivel de persistencia que proporciona almacenamiento y recuperación de objetos individuales a y desde un archivo. La persistencia simple no conserva apuntadores relacionados entre los objetos persistentes.
3. Persistencia Isomórfica: Un nivel de persistencia que conserva apuntadores relacionados entre los objetos persistentes.
4. Persistencia Polimórfica: El nivel alto de persistencia. persistencia polimórfica conserva apuntadores relacionados entre los objetos persistentes y permite a los procesos restaurados restaurar un objeto sin conocimiento de aquel tipo de objeto.

### 3.9.1 PERSISTENCIA SIMPLE.

Es el almacenamiento y recuperación de un objeto a y desde un archivo. La simple persistencia es un rápido y fácil camino para guardar y restaurar objetos que no tienen apuntadores a otros objetos ni funciones de miembros virtuales. Sin embargo, cuando los objetos que se referencian unos a otros son guardados y entonces restaurados con simple persistencia, la relación de apuntadores, morfología, entre los objetos puede cambiar. Esto es porque la persistencia simple asume que cada apuntador a un objeto en memoria referencia a un único objeto. Así, cuando un objeto es guardado con simple persistencia, dos referencias a la misma localidad en memoria causará dos copias del contenidos de aquella localidad de memoria a ser guardada y más tarde restaurada. No solamente esto hace que se use un espacio extra en el archivo, también causa que el objeto restaurado apunte a dos distintas copias del objeto referenciado.

El problema de este nivel de persistencia consiste en que no mantiene los apuntadores relacionados entre objetos.

Un claro ejemplo de persistencia simple y apuntadores se ve en la Figura 3.4, en ésta se puede apreciar que los objetos que se referencian unos a otros y son guardados con persistencia simple, al momento de ser restaurados con simple persistencia, la morfología entre objetos puede cambiar. Esto es porque la persistencia simple asume que cada apuntador a un objeto en memoria referencia a un único objeto. En este caso ocasionó que se duplicaran los objetos teniendo como consecuencia que se ocupe más espacio en memoria, así como se ve modificada su estructura inicial.

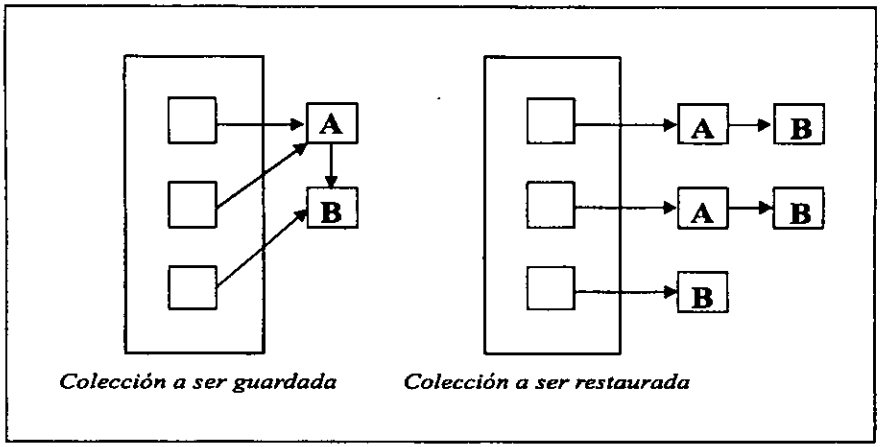



**Figura 3.4.- Persistencia Simple.**

### 3.9.2 PERSISTENCIA ISOMÓRFICA.

Persistencia isomórfica es el almacenamiento y recuperación de objetos a y desde un archivo tal que los apuntadores relacionados entre los objetos son preservados. Si no hay apuntadores relacionados, la persistencia isomórfica efectivamente guarda y restaura objetos de la misma manera que una simple persistencia. Como se observa en la Figura 3.5.

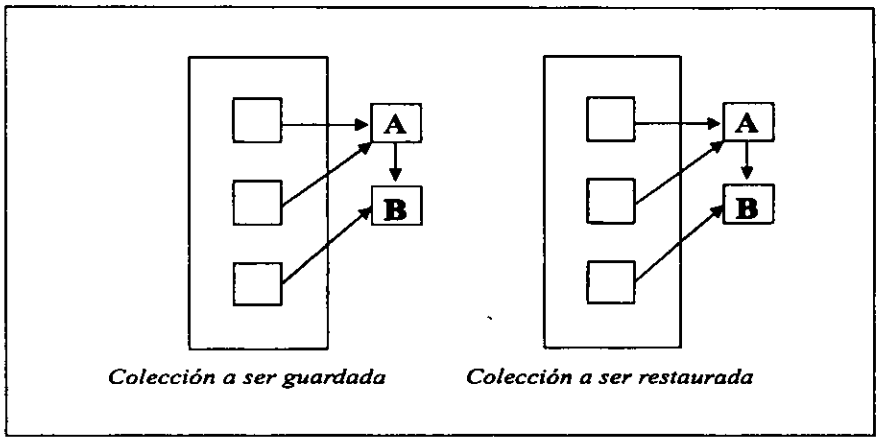



**Figura 3.5.- Persistencia Isomórfica.**

### 3.9.3 PERSISTENCIA ISOMÓRFICA VERSUS PERSISTENCIA SIMPLE.

En la Figura 3.6, una colección de múltiples apuntadores a el mismo objeto es guardado y restaurado desde un archivo, usando persistencia simple. Note que cuando la colección es guardada y recuperada, cada apuntador apunta a distintos objetos, que contienen la misma información y esto se debe a las causas antes mencionadas. En contraste, la persistencia isomórfica de la misma colección, que se puede apreciar en la Figura 3.7, todos los apuntadores son restaurados al mismo objeto, como antes de ser guardados.

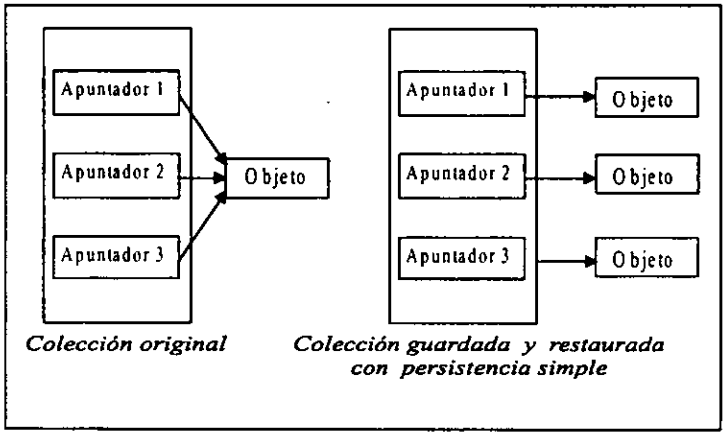



**Figura 3.6.- Guardando y restaurando con Persistencia Simple.**

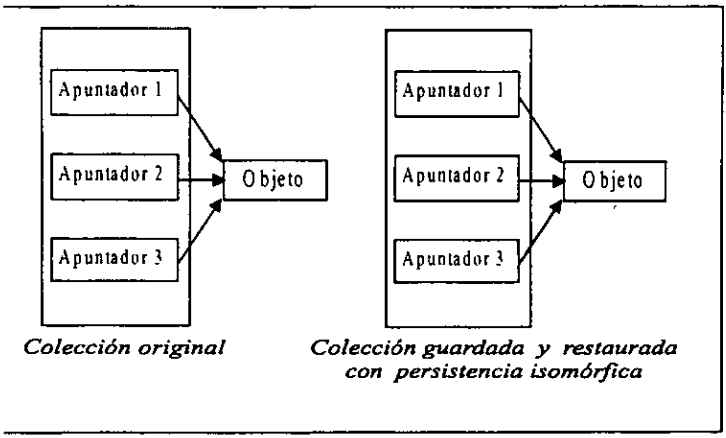



**Figura 3.7.- Guardando y restaurando con Persistencia Isomórfica.**

En la Figura 3.8, se intenta salvar y restaurar una lista ligada circularmente, usando persistencia simple. Como se muestra en la figura, el intento de usar persistencia simple para guardar una lista ligada circularmente cae en un ciclo infinito.

El mecanismo de persistencia simple crea una copia de cada objeto que está apuntando y guarda aquel objeto en un archivo. Pero el mecanismo de persistencia simple no recuerda cuales objetos ya han sido guardados. Cuando el mecanismo de persistencia simple encuentra un apuntador, no tiene manera de conocer si ya ha sido salvado aquel apuntador del objeto. Así que en una lista ligada circularmente, el mecanismo de simple persistencia salva los mismos objetos una y otra vez.

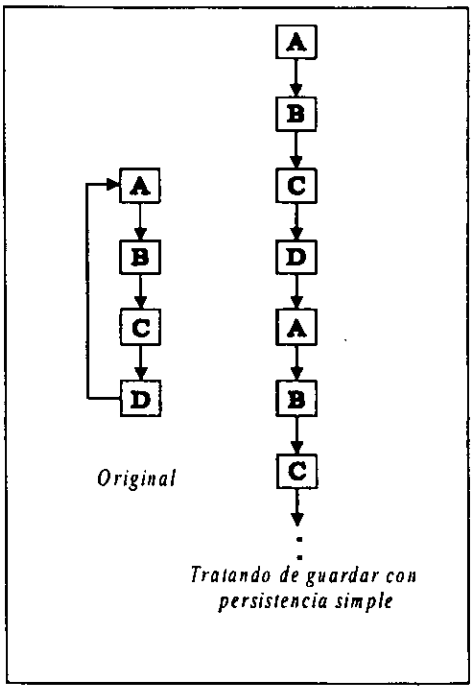



**Figura 3.8.- Guardando y restaurando una lista ligada circularmente con Persistencia Isomórfica.**

En otra forma, como se muestra en la Figura 3.9, la persistencia isomórfica permite guardar listas ligadas circularmente. El mecanismo de persistencia isomórfica usa una tabla para mantener la trayectoria de los apuntadores que han sido salvados. Cuando el mecanismo de persistencia isomórfica encuentra un apuntador a un objeto que todavía no ha sido guardado, copia el dato del objeto, guarda aquel apuntador, entonces mantiene el rastro del apuntador en la tabla guardada. Si el mecanismo de persistencia isomórfica más tarde encuentra un apuntador a el mismo objeto, en lugar de copiar y salvar el dato del objeto, el mecanismo guarda en la tabla la referencia del apuntador.

Cuando el mecanismo de persistencia isomórfica restaura apuntadores a objetos desde archivo, el mecanismo usa una tabla para el proceso de reverso. Cuando este mecanismo encuentra un apuntador a un no restaurado objeto, crea el objeto con los datos desde el archivo, entonces cambia el restaurado apuntador a un punto del recreado objeto. El mecanismo mantiene el rastro del apuntador en la tabla de restauración. Si el mecanismo más tarde encuentra una referencia a un apuntador que ya ha sido restaurado, entonces el mecanismo busca la referencia en la tabla de restauración, y actualiza el apuntador restaurado a apuntar a el objeto referido en la tabla.

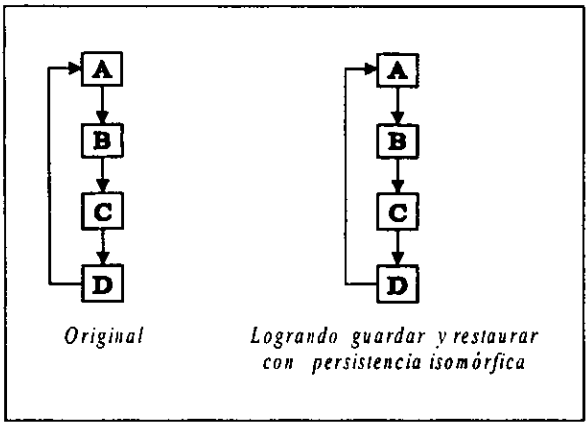



**Figura 3.9.- Guardando y restaurando una lista ligada circularmente con Persistencia Isomórfica.**

### 3.9.4 PERSISTENCIA POLIMÓRFICA.

Preserva apuntadores relacionados entre objetos persistentes, y también permite el proceso restaurar un objeto sin conocer que tipo de objeto es.

### 3.9.5 PRECAUCIONES.

La persistencia es una cualidad útil, pero requiere cuidados en algunas áreas. Aquí están unas pocas cosas que considerar para cuando se use persistencia en objetos:

- Siempre salve un objeto por valor antes de salvar la identidad del objeto por apuntadores. En ambos casos la persistencia isomórfica y la persistencia polimórfica de objetos, nunca se debe poner en un archivo de salida un objeto por apuntador, antes de colocar en el archivo de salida identifique el objeto por su valor. Sin embargo diseñe una clase que contenga un valor y un apuntador a aquel valor. Las clases siempre deben salvar o restaurar el valor , y después el apuntador.
- No salve distintos objetos con la misma dirección.

# 4. MÉTODOS PARA IMPLEMENTAR PERSISTENCIA.

## *4.1 INTRODUCCIÓN.*

Ejecutar un sistema orientado a objetos usualmente nos deja la necesidad de almacenar objetos creados durante la corrida de una sesión. Esto debe ser solucionado utilizando objetos persistentes, es decir, ellos deben de tener la habilidad de existir aún después del tiempo del contexto que los creo.

Almacenamiento de objetos en una forma persistente es la principal tarea de un Sistema Manejador de Base de Datos de Objetos (ODBMS)[12]. Sin embargo, si una base de datos no esta disponible o es muy enorme para usarse (o una aplicación no necesita de todas las características de una base de datos), la persistencia puede ser agregada a un sistema a través de diferentes técnicas, dependiendo de los requerimientos del sistema.

En éste capítulo se analizarán diferentes técnicas para implementar objetos persistentes sin utilizar una Base de Datos.

## *4.2 FACTORES POR LO CUAL EXISTEN DIFERENTES ENFOQUES DE PERSISTENCIA.*

Las diferentes técnicas dependen de los factores como:

- El conocimiento de la prioridad de los tipos de objetos.
- La necesidad de tratar con apuntadores y referencias.
- La necesidad de tener un sistema del almacenamiento completamente separado de la aplicación.
- El grado de generalidad de un sistema de almacenamiento.
- Incrementabilidad.
- Esfuerzo requerido de desarrollo.
- En tiempo de corrida ocurra una sobrecarga.
- Requerimientos de preprocesador y oportuna automatización.
- Portabilidad.

Al momento de desarrollar una técnica para implementar persistencia se deben de tomar en cuenta a un sistema de almacenamiento sofisticado, el cual debe estar completamente separado de la aplicación y además debe de envolver mínimos cambios para el código ya escrito.

Incrementabilidad de objetos persistentes consiste en cargar objetos cuando los va necesitando el programa en ejecución.

## *4.3 DIFICULTADES AL IMPLEMENTAR PERSISTENCIA EN OODB (BASE DE DATOS ORIENTADA A OBJETOS).*

Los siguientes puntos son considerados desventajas al tratar de usar persistencia en una ambiente OODB:

- Espacio extra (overhead): Si los objetos son pequeños el espacio extra impuesto por una Base de Datos Orientada a Objetos para cada objeto puede ser muy grande en comparación con el tamaño del objeto.

- Espacio extra en tiempo de corrida: El tiempo necesitado para accesar objetos en la Base de Datos debe ser muy largo. Esto causa una lenta respuesta, el usuario debe de encontrar como no aceptable esperar dos o tres segundos para una lista de peticiones.

- Programación complicada: Una Base de Datos de objetos puede ser complicado usar, a partir de aplicaciones que deben estar ligadas a una librería de base de datos y un esquema debe ser creado para la aplicación.

- Mantenimiento complicado: Cuando usan una Base de Datos de objetos, hay un requerimiento para el administrador de la Base de Datos (DBA). Esto es bueno para un sitio en evolución, pero para simples casos de usos debe de ser muy complicado.

El problema básico de aquí es que las aplicaciones tiene que pagar por características que no necesitan, por ejemplo aplicaciones que no necesitan control de concurrencia o soportar la arquitectura heterogéneas, incurriendo en tiempo adicional, espacio extra.

## *4.4 CONSIDERACIONES AL HACER UN OBJETO PERSISTENTE.*

Para almacenar y restaurar objetos individuales se deben considerar:

- Las entradas para direccionar el almacenamiento de objetos son:

    1. Identificador de la clase.
    2. Codificación de los tipo de datos.
    3. Codificación de datos complejos tal como contenido y apuntadores del objeto.

- Llave de salida en la restauración de un objeto:

    1. Creación de una instancia actual del objeto usando un simbólico nombre de la clase.
    2. Recuperación de datos desde un flujo y restaurar el valor de los datos.

## *4.5 PROBLEMAS EN EL MANEJO DE APUNTADORES.*

Almacenar/cargar punteros directamente envuelve dos principales problemas:

1. Reconocerlos en objetos declarados por el usuario. A partir de que ellos deben ser tratados diferentemente desde otros datos miembros.

2. Representación de sus referencias en una manera de direcciones independientes cual es válido cada vez que los objetos son cargados.

El problema 1 puede ser solucionado usando técnicas de preprocesamiento. Un precompilador parsea todas las clases y reorganiza todas las palabras claves que indican cuales clases deben ser persistentes y entonces las registra en una clase diccionario. Además, puede reorganizar apuntadores dentro de la declaración de clases y así guardarlos. Esta es una elección de algunas ODBMSs, pero es muy enorme y no puede ser un costo efectivo para los desarrolladores de una simple aplicación o para pequeños número de aplicaciones, que no necesitan una completa ODBMS.

El problema 2 : Pueden ser solucionado a través de un mecanismo de ordenamiento de punteros lógicos que permite la identificación de objetos independientemente de su actual dirección física. Un buen camino es el uso del offset de objetos en el arreglo porque no cambian después de recargarlos y pueden ser manejados fácilmente. En C los apuntadores tienen una conducta que no debe ser modificada, debemos usar una clase que se comporte como un apuntador pero también tengan la habilidad de referenciar a la actual dirección del objeto usando su offset. No necesitamos usar apuntadores de C y el sistema de persistencia no necesita reconocerlos en la declaración de clases.

## *4.6 MÉTODOS.*

### 4.6.1 ENFOQUE USANDO UN FLUJO (STREAM).

Este es un simple métodos usados para implementar objetos persistentes, consiste en que un objeto se almacena el mismo escribiendo cada uno de sus atributos a un archivo de datos secuencial[13]. Si un atributo de un objeto es el mismo, un objeto es simplemente encajado en su flujo de salida de origen. El resultado es un flujo secuencial, en el cual los datos de objetos contenidos son guardados en los datos del objeto origen. Recuperación de un objeto es más o menos fácil, cada uno de sus atributos es leído en forma secuencial desde el archivo de datos.

Una ventaja importante es que los objetos se guardan ellos mismos en su estado actual y cuando se reanuda la sesión los objetos persistentes "recuerdan" todo lo necesario para levantar la sesión a partir de donde se desactivo.

Este mecanismo no es adecuado para objetos que son compartidos. Cuando los objetos son compartidos, colocarlos en el flujo de origen trae problemas de duplicación cada vez que es referenciado. Una solución para este problema sería tener el dueño del objeto guardado. Para este trabajo debe ser claro saber quien es el dueño. Generalmente el propietario de un objeto es la entidad quien originalmente creo el objeto.

Los apuntadores a otros objetos son sin sentido una vez que los objetos han sido colocados en el flujo de salida fuera tal mecanismo debe de ser imposible para un objeto que está siendo restaurado localizar sus objetos compartidos.

Un mejor enfoque para salvar objetos compartidos es hacer que el objeto este enterado que es compartido.

Para programas pequeños los flujos son suficientes, para grandes programas se necesita un flujo para cada objeto que tenga un único identificador de objeto asignado donde el objeto es la raíz del flujo.

El flujo de entrada/salida proporcionado en C++ es el proceso de convertir tipo de objetos dentro de secuencia de caracteres y viceversa. El problema es que el flujo no copia idénticamente construcciones orientada a objetos tal como son las estructuras de datos polimórficas.

### 4.6.1.1 Persistencia para un objeto no Polimórfico en C++.

La única forma es usando operadores << que al aplicarlo a un objeto se insertan los datos miembros del objeto dentro del flujo. La opuesta operación es el operador de extracción >> que debe de ser sobrecargada para asegurar la transferencia del flujo hacia los correspondientes datos miembros del objeto.

#### 4.6.1.1.1 Limitaciones.

Ningún mecanismo es proporcionado para escribir y leer objetos de entidades globales. Al hacer un objeto, el usuario debe de descomponer el objeto en sus constitutivos datos miembros.

Está técnica trabaja solamente si al sobrecargar los operadores << y >>, escriben y leen datos miembros los cuales son conocidos en tiempo de compilación, por lo tanto los

operadores de inserción y extracción deben de haber sido definidos. El iostream.h define estos dos operadores para integrar tipos como char, char*, short, int, long, float, double, etc.[14]

Los siguientes aspectos deben de ser considerados por el programador:

- El programador tiene la responsabilidad de extraer los datos miembros de un objeto en el mismo orden en el cual ellos han sido insertados dentro del flujo.
- El programador tiene que recordar el orden en el cual los objetos pertenecientes a diferentes tipos fueron insertados dentro del flujo.
- Si los objetos contienen datos miembros que hacen referencia a otros objetos. Las referencias no necesitan llegar a ser persistentes solamente las relaciones entre los objetos. Ejemplo En un Arbol B hay que guardar los datos y la estructura del dato no los apuntadores.

Por lo tanto C++ requiere un gran esfuerzo de los programadores.

#### 4.6.1.2 Persistencia para un objeto Polimórfico en C++.

No es un camino totalmente orientado a objetos. El concepto de flujo en C++ no permite salvar objetos como objetos reales, solo salva los datos miembros de esos objetos[15].

El problema con una estructura de datos polimórficos es que contiene elementos de diferentes tipos. Cuando leemos, tenemos que saber de que tipo de elementos será leído. El camino es:

- Insertar primero el nombre de la clase dentro del flujo.
- Seguido por el dato miembro del objeto.

Cuando se extraen los objetos desde el flujo se tiene que:

- El nombre de la clase debe de ser leído primero.
- Conocido el nombre de la clase tenemos que crear un objeto de tal clase.
- El programador debe declarar variables especiales en el programa para crear objetos dinámico.
- El punto malo de este enfoque es que el programador tiene que escribir y adoptar su rutina para crear objetos de tipo de la clase, trabajando totalmente en un camino totalmente no orientado a objetos.

#### 4.6.1.3 Enfoque de objetos compuestos de un sólo tipo de dato.

La solución es usar un único identificador de clase y el orden en el cual los campos son colocados en el flujo.

Las características más importantes en el almacenamiento y restauración de objetos son:

- El identificador de clase para un objeto debe ser colocado dentro del flujo antes que sus campos así que una instancia de un objeto puede ser creado antes que los campos sean restaurados.
- Y los campos deben ser restaurados en el mismo orden en el que ellos fueron guardados, las clases deben ser responsables de las instancias variables definidas.

#### 4.6.1.4 Salvando objetos con apuntadores.

Apuntadores a simples datos apilados: Una razón para utilizar pilas es que la cantidad de espacio necesario para un dato no se sabe de entrada. En el momento que el objeto es salvado, la estructura de la pila ya tiene un tamaño exacto, así que los datos de la pila pueden ser guardados en un lugar en el flujo del objeto.

Cuando un dato de la pila es restaurada es importante saber cuanto espacio de pila asignar antes que sea accesado desde el flujo. Una forma es colocar la longitud dentro del flujo antes del dato actual.

#### 4.6.1.5 Salvando apuntadores a otros objetos.

Un apuntador a un objeto no compartido puede ser tratado como un contenedor de objeto. El objeto es insertado en el flujo en la misma manera como un contenedor de objetos, pero existe una pequeña diferencia cuando el objeto es restaurado desde el flujo. Se tiene que implementar operadores extras para poder recuperar los objetos[16].

En la Figura 4.1 se muestra la estructura que siguen los objetos cuando apuntan a objetos no compartidos y en la
Figura 4.2 se aprecia la forma en que se guardan éstos en un flujo de salida.

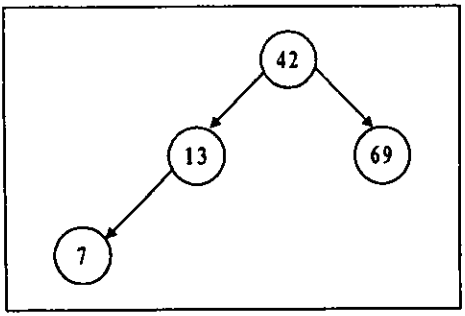



**Figura 4.1.- Objetos no compartidos.**

| | | | | |
|---|---|---|---|---|
| Clase_ArbolBinario | 42 | Clase_ArbolBinario | 13 | |
| Clase_ArbolBinario | 7 | Clase_Vacía | Clase_Vacía | Clase_Vacía |
| Clase_ArbolBinario | 69 | Clase_Vacía | Clase_Vacía | |

**Figura 4.2.- Flujo del objeto para una instancia de la clase ArbolBinario.**

Cuando un objeto es compartido por otros objetos, todos los apuntadores son restaurados con el mismo objeto. Hacer esto requiere mas compatibilidad en la parte del flujo , porque debe recordar la dirección de los objetos que han sido insertado dentro del flujo y la compensación al cual ellos ocurren. Ver Figura 4.3.

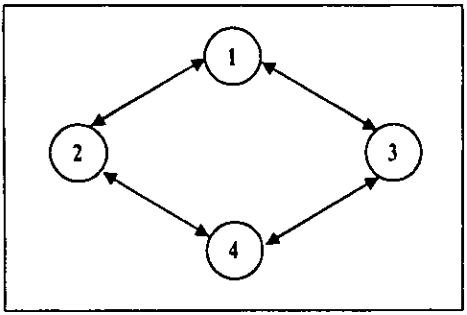



**Figura 4.3.- Objetos compartidos.**

#### 4.6.1.6 Restaurando objetos.

Restaurar un objeto que tiene sólo un tipo de dato base es crear un objeto del tipo correcto. Para objetos complejos se requiere objetos que deben ser creados.

Tipos de Datos Base: Son restaurados fácilmente desde el objeto flujo porque los valores son simplemente copiados desde el flujo a la etiqueta de direcciones.

Apuntadores a objetos: El apuntador de instancia variable puede ser restaurado como algún otro campo.

### 4.6.2 CREANDO OBJETOS BASADOS EN UN IDENTIFICADOR DE CLASE.

El enfoque básico para restaurar objetos es crear una instancia del objeto y entonces restaurar la instancia variable del objeto desde los valores en el flujo. La importancia de este enfoque es la habilidad para crear una instancia de un objeto desde un simbólico nombre de clase. En el lenguaje C++ no hay nada que interprete una constante o cadena (string), como nombre de una clase en tiempo de corrida.

Para implementar esta técnica se puede desarrollar un método dentro de la clase para que regrese un apuntador a una instancia de la clase. Esto no es un problema porque cada clase conoce su propia identidad.

Ejemplo:

```
Class Cubo: public Objeto{
public:
...
static Objecto* creaInstancia(void);
...
};

static Objecto*
Cubo::creaInstancia(void)
{
        return (new Cubo);
}
```

Donde la clase Objeto trabaja como variable que apunta o hace referencia a objetos creados por alguna clase derivada, en este ejemplo hace referencia a la clase derivada Cubo, teniendo el control del mismo.

### 4.6.3 INCREMENTABILIDAD.

Para aplicaciones que no necesitan salvar grandes cantidades de información debe ser suficiente salvar sus datos como un closter de objetos. Para aplicaciones que usan estructuras grandes resulta no práctico guardar toda la estructura en memoria en tiempo de corrida. Tales aplicaciones necesitarán salvar muchos objetos o closters de objetos y restaurar sólo un subgrupo de ellos a través de algún programa en ejecución. El método de Identificador Único de Objetos (OIDs "Unique object identifiers") es necesario para salvar los objetos[17].

Por ejemplo en la Figura 4.4 , se tiene que cada grupo de objetos que tienen asociado un OID y existen un grupo especial que hace referencia a esos grupos. Los objetos que tienen como etiqueta el 2 y 3 no necesitan usar OIDs para hacer referencia al objeto que tiene como etiqueta el 1, porque se encuentran en el mismo closter.

Así de esta forma sólo un subgrupo de objetos almacenados puede ser restaurado para la ejecución de un programa.

En seguida se enuncian esquemas para implementar la incrementablidad de objetos persistentes.

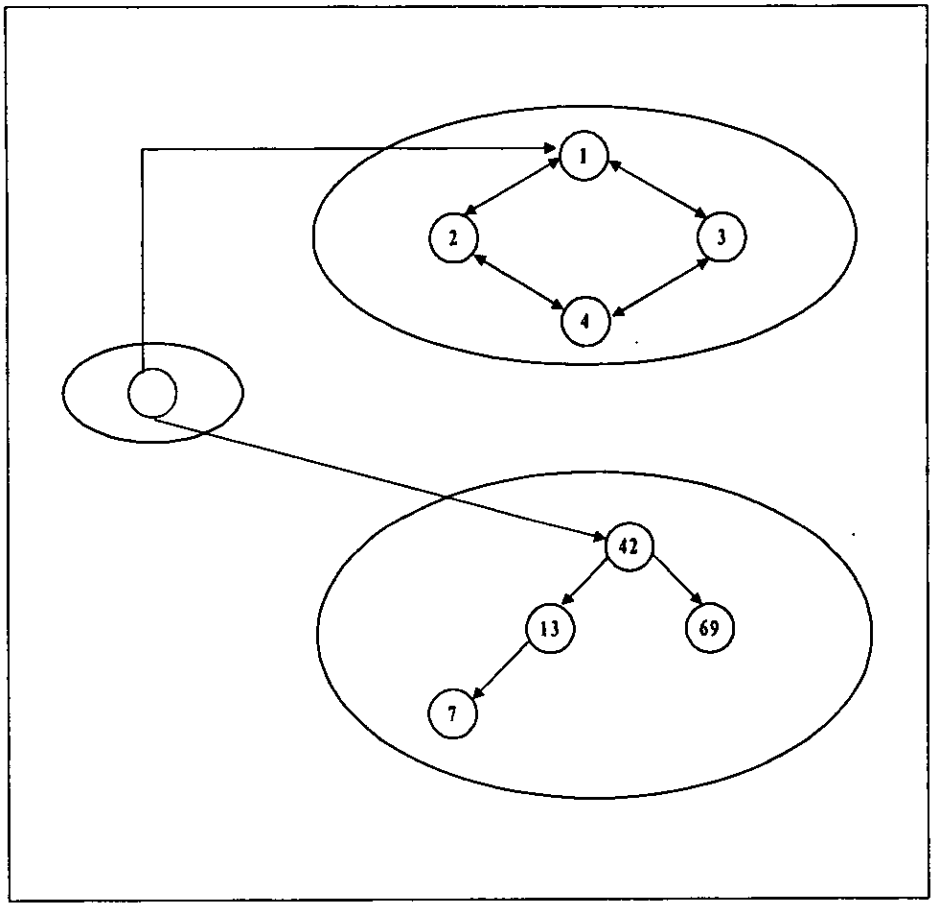



**Figura 4.4.- Grupo de Objetos.**

#### 4.6.3.1 Sin tipo OIDS.

Un esquema de direcciones comúnmente usan un lenguaje interprete sin tipos estáticos es para hacer que la referencia de los datos indirectos sea a través de una dirección de una tabla de traslación. Una referencia a un dato es un offset dentro de la tabla de traslación y la dirección actual es obtenida por extracción de la dirección en el correcto offset. Cada punto de referencia cuesta una tabla extra de búsqueda.

La ventaja del enfoque es que hace fácil cambiar la dirección actual de un objeto sin cambiar su simbólica dirección.

La desventaja de este enfoque es que se introduce una sobrecarga de traslación de direcciones y subvertidos tipos de información para objetos persistentes. El programador está obligado a forzar el tipo del apuntador que es retornado por el método.

#### 4.6.3.2 Surrogates (Sustitutos).

La idea de un surrogate es que permanezca en el lugar del objeto actual. El surrogate permanece en el lugar de un objeto persistente que no ha sido cargado en memoria. En este enfoque no se necesita cálculos de direcciones, porque el surrogate es un objeto actual con una dirección valida. El enfoque toma sobre el primer mensaje enviado el objeto surrogate, reemplaza el surrogate objeto por el actual objeto en la misma dirección y reexpide el mensaje.

Dos temas son importantes para objetos surrogates:

1. El objeto surrogate debe responder al mismo mensaje como el objeto actual, pero debe predominar sobre cada uno de los mensajes así que atrapa y carga el objeto.
2. Porque el objeto actual remplazará el surrogate en la misma localización en memoria. El objeto surrogate debe tomar la misma cantidad de almacenamiento actual.

El uso de surrogates tiene la ventaja de no necesitar producir, sobrecarga para calcular la dirección actual de un objeto. Después de que el primer mensaje al surrogate ha sido recibido, la sobrecarga producida es por C++.

### 4.6.4 USANDO HERENCIA.

Para esta técnica primero se necesita conocer el tamaño del objeto, el cual debe ser derivado desde una clase base por lo tanto debe de definirse las funciones : tamaño() y clase_nombre(). La función tamaño() retorna el tamaño del objeto. La librería puede usar esta función para salvar o recuperar el objeto[18].

Para cada objeto, su class_name y tamaño es salvado, y entonces la binaria representación del objeto es salvada.

El problema aquí es que C++ implementa funciones virtuales para usar apuntadores a una tabla de funciones (conocido como una tabla virtual o "vtbl"). Un apuntador conocido como "vtpr" a esta tabla virtual es puesto en cada objeto que tiene una función virtual o es derivada desde un objeto que tiene funciones virtuales. El valor de este apuntador puede cambiar cada vez que el programa es ejecutado, por lo tanto nosotros no podemos simplemente almacenar, el "vtpr" así nada más en el archivo y usarlo directamente cuando es recuperado.

Un camino para direccionar este problema es cambiar las librerías así que el programador diseñe clases persistentes que deben leer o escribir los atributos de los objetos desde el archivo.

La clase base define una interfase y un protocolo que es usado por la librería para leer o escribir objetos desde archivos. Una variación de este enfoque es usando las librerías de MFC ("Microsoft Foundation Class")[19]. El IOLIB (librería de entrada/salida) tiene dos colecciones de objetos:

- Prototipo: Cual contiene una muestra de objetos de cada clase de objeto.
- Objetos: Las instancias que son leídas o escritas a archivos.

Cuando la librería lee un archivo, encuentra un objeto con el mismo nombre de la clase en la colección de prototipos. Usa el método crear() de los prototipos, para crear una nueva instancia. El método leer() de la nueva instancia es entonces llamado para leer el dato desde el archivo por lo tanto el usuario de la librería tiene que registrar los prototipos con la librería antes de llamar leer().

#### 4.6.4.1 Desventajas.

1. Manual intervención: El programador debe escribir cuatro funciones virtuales para la librería. El crear() y obten_nombre_clase() son más o menos triviales. Sin embargo las funciones leer() y escribir() son a menudo complejas. En particular para objetos grandes con muchos atributos estas funciones pueden ser enteramente largas.

2. Sobrecarga en la Ejecución: Leer los atributos uno a uno puede resultar lento.

3. Falta de generalidad: Este enfoque no trabaja para "concretos" tipos (es decir tipos que no usan herencia y vinculaciones dinámicas).

### 4.6.5 ANALIZADOR DE ARCHIVOS DE CABEZERA (PARSING HEADER FILES).

Otro camino para implementar persistencia en C++ es parse header files de objetos usado en la librería para identificar los atributos de cada objeto. Algunos OODB vendedores (como ObjectStore) usan este enfoque. Donde los atributos y el offset de cada clase de objeto es guardada en una meta clase que está asociada con la librería. Usando una librería meta clase permite a los programadores leer o escribir los objetos fácilmente. La desventaja con analizador de encabezados es que la librería diseñada debe saber donde el compilador coloca el puntero en la tabla de funciones virtuales y el offset de cada atributo. Porque esta solución depende del compilador es inherentemente no portable.

### 4.6.6 USANDO CONSTRUCTORES.

El constructor de una clase es responsable de transformar un flujo binario de entrada en un objeto de la clase. Si invocamos al constructor de una clase dada sobre un pedazo de memoria, éste debe llegar a ser un objeto de esta clase. En particular la tabla virtual debe de ser inicializada por el compilador, el problema con esta solución es que no se puede almacenar la dirección del constructor como una función regular ni puede ser portable la llamada. No hay constructores virtuales en C++ por lo tanto para llamar al constructor la librería debe de saber el tipo de objeto y así debe ser recompilado con cada nueva clase.

Las librerías de entrada y salida tienen información de cada objeto persistente. Por cada objeto la librería mantiene el nombre del objeto, una función que fija C++ apuntadores en objetos para comunes aplicaciones direccionadas y el tamaño del objeto.

La librería Entrada/Salida mantiene las siguiente información de cada objeto:

- El nombre del objeto.
- Su tamaño
- Un apuntador a una función que fija el vtbl apuntador.

La ventaja de esta solución son:

- Es portable y trabaja con cualquier compilador.
- Trabajo no virtual, es decir, no se tiene que conocer donde el compilador coloca los apuntadores.
- Derivación: Los objetos persistentes no tienen que ser derivados de una clase base.

### 4.6.7 USANDO UNA CLASE BASE VIRTUAL.

En este enfoque hay una clase universal que tiene funciones virtuales. Cuando un programa necesita una clase persistente, heredan la clase desde una super clase, usando la herencia virtual que es un mecanismo de C++[20].

Se tendría que implementar una clase almacenable, que tenga funciones miembros de lectura/escritura que son heredados por todas las clases que quieran hacer persistencia:

```
class Almacenable {
        virtual write() const=0;
        static read();
        ..............
};
```

Tales clases simulan una metaclase. Cada clase derivada desde el Almacenamiento debe redifinir la lectura/escritura de funciones. Si una clase derivada es enorme, necesitará cierta cantidad de trabajo para cambiarlo. De hecho si se tiene cierto número de clases organizadas en una jerarquía y se les quiere hacer persistentes, se debe cambiar su código, de forma monótona insertando las apropiadas funciones de lectura y escritura. Esto conceptualmente no resulta complejo, pero es realmente tedioso y consume tiempo.

#### 4.6.7.1 Ventajas.

- El programador obtiene en paquete los objetos en alguna forma que se quiera ejemplo: Si uno quiere declarar el objeto árbol como persistente uno puede empacar el descendiente de un nodo en la localización que más convenga para su rápida recuperación. Así haciendo "closters" es más o menos directo.

- No asume ningún específico soporte desde el sistema operativo subyacente más halla de la habilidad de emitir llamadas para empaquetar, desempaquetar y otras operaciones de disco entrada/salida.

#### 4.6.7.2 Dificultades.

- Si uno necesita un tipo entero que sea persistente, no se puede hacer porque éste no es una clase. Un camino para combatir este problema es definiendo una "Clase-Entera" y hacer una herencia persistente desde la clase del Objeto. Problema a la hora de incrementar el entero, uno necesita accesar el dato miembro del objeto y entonces incrementarlo.

- Existen librerías que no pueden ser utilizadas, si queremos que la clase string sea persistente, hay que definir una nueva clase que sea derivada desde la clase base virtual, por lo tanto las librerías que vienen con el string no pueden ser usadas directamente.

- Dificultad para soportar funciones virtuales.

- Retraso debido al chequeo permanente.

### 4.6.8 USANDO SMART POINTERS (SP).

Un Smart Pointer (SP) son pequeños apuntadores que contienen la dirección de los objetos, un conjunto de SPs forman un arreglo que debe de estar en memoria. La mejor característica de la técnica SP es un buen principio para la obtención del control completo de algunos objetos creados dinámicamente[21].

Los SPs son clases de template que se comportan como apuntadores a otras clases.

Ellos pueden ser definidos como sigue:

```
template<class X>
class Pointer {
        Pointer() : stor(0) {}
        Pointer(X*p) : stor (p) {}
        X* operator->()
                {assert(stor); return stor;}
        .....
        private:
                X* stor;
        .....
        };
```

Aquí unos ejemplos de declaraciones de SP:

```
Pointer<Rectangle>pr=new Rectangle(...);
Pointer<Object>po=new Object(...);
```

La más interesante característica de Sp es, su capacidad de obtener, mantener y aplicar un completo control del objeto apuntado. Este control es obtenido primero con el constructor Pointer<X>::Pointer(X*), cual recibe como argumento la dirección del objeto apuntado. Entonces la dirección es guardada en el dato miembro stor. Finalmente, el control sobre el objeto es aplicado a través de la sobrecarga de las funciones miembros Pointer<X>::operator*() y Pointer<X>::operator->(). De hecho, cada operación llamada en un objeto dinámico debe de pasar a través de estos operadores, asi la operación puede ser fácilmente redirigida al objeto apuntado por el dato miembro stor, y al mismo tiempo, algunas otras operaciones ventajosas para persistencia pueden ser realizadas.

#### 4.6.8.1 Ventajas.

Primero el mecanismo de persistencia es separado de la aplicación puede ser asociado a un gran número de aplicaciones sin modificar el diseño.

Segundo el mecanismo descrito no requiere casi de esfuerzo en la parte de programación. Los SPs pueden ser usados como apuntadores normales a partir de la declaración. Si el usuario necesita un objeto a ser persistente, debe declarar un SP (como un dato miembro) y puede entonces ejecutar todas las operaciones normalmente hechas con apuntadores de C (Puede usar operaciones como assign, duplicate, delete, etc.). El objeto apuntado por el SP puede residir en una predefinida estructura y el usuario puede almacenar y cargar estos objetos sin preocuparse de su contenido.

Una vez que el SPs ha sido declarado el mecanismo de almacenar/cargar llega a ser transparente al usuario. El usuario no necesita escribir palabras claves tal como *persistent* en la declaración de clases o derivar clases de aplicaciones desde una clase almacenable,

para indicar que tales clases deben ser consideradas persistentes, y sus instancias pueden ser almacenadas o recargadas. Evitando estas desventajas involucra evitar cierta cantidad de trabajo cuando se empieza a utilizar un sistema de objetos persistentes, después de que muchas clases han sido implementadas (es decir, no es necesario agregar funciones de lectura/escritura de cada clase a ser almacenada) y cuando se desarrolla una aplicación (es decir, agregar o borrar datos miembros nuevos no involucra la actualización de todas las funciones de lectura/escritura ya implementadas).

Tercero los objetos son almacenados con el mismo formato que ellos tienen en memoria principal; traslaciones desde un formato a otro, no son necesarias.

Es también importante hacer hincapié que el mecanismo difiere de una copia de la memoria ya que el número de objetos de una aplicación no depende de la memoria disponible. En la aplicación los SPs deben residir en memoria principal, mientras el arreglo de objetos debe de residir en disco, permitiendo el manejo de unos miles de objetos. Por lo tanto los objetos en el disco deben ser compartidos entre programas o cargados en otra clase de computadora.

Agregar nuevos tipos de objetos sólo involucra agregar nuevos arreglos de SPs

Finalmente, la separación entre el SP y el espacio de memoria del objeto hacen muy fácil implementar funciones del manejo de la memoria, tal como recursos de protección/desprotección estrategias del manejo de memoria, destrucción automática, colección de basura o recuperación inteligente de error. Para algunos esto sería suficiente para operar sólo en SP. Por ejemplo, la protección/desprotección de objetos debe ser simplemente implementado, agregando un dato miembro que se comporte como una bandera indicando si el objeto esta protegido o no.

Para otras funcionalidades, tal como colección de basura, sería posible pasar a través de SPs y marcar el objeto a apuntar. Cuando todos los objetos han sido visitados, todos los objetos contenidos en memoria no apuntados por un SP serán recargados. Por último, note que cambiar las direcciones del objeto no cambia involucrará solo la actualización de las direcciones de los objetos en los SPs relacionados.

### 4.6.9 ENFOQUE DEL MANEJADOR DE MEMORIA.

Este enfoque es usado por Smalltalk donde la imagen de la memoria entera del programa es descargada a disco y restaurada cuando el programa está corriendo. Salvando y cargando la memoria sin preocuparse del contenido.

Este enfoque no es más que un almacenamiento en un vector de todas las clases de objetos que deben ser persistentes con ello se puede guardar/cargar el entero arreglo en vez de un objeto a la vez.

Para realizar esto es necesario evitar el usual mecanismo de asignación de objetos en C++, para obtener el control de un objeto asignado. Esto es posible a través sobrecargando la función *new*, como objetos asignados a un arreglo predeterminado, en lugar de dispersarlos

en una pila. Es útil pasar al operador new el nombre del arreglo en el cual se quiere asignar un objeto.

```
void* operator new(size_t size, char* arrayname){
        void* ptr;
        ptr=allocate(size, arrayname);
        return ptr;
}
```

Por medio de la apropiada función *allocate*(..), el objeto nuevo es almacenado en el arreglo *arrayname*. Aquí un ejemplo de asignación en diferentes arreglos:

```
char db1[100];
char db2[100];
char db3[100];
class Circle {....};
main() {
        Circle* pc1=new(db1) Circle(....);
        Circle* pc2=new(db2) Circle(....);
        Circle* pc3=new(db3) Circle(....);
}
```

#### 4.6.9.1 Ventajas.

- La principal ventaja de este esquema es que almacenar y cargar son simples operaciones, porque sólo consiste en leer y escribir sobre un arreglo. El usuario solo tiene que definir algunos mecanismos para identificar los objetos una vez que el arreglo ha sido recargado.

#### 4.6.9.2 Desventajas.

- Una desventaja es que el arreglo entero y todos los objetos deben residir en memoria principal. Esto implica que el número de objetos que pueden ser almacenados depende de la cantidad de memoria disponible.

- Pero la super falta de esta técnica es que no permite el almacenamiento de datos miembros que estén apuntando a objetos localizados en otras áreas del arreglo. A decir verdad se pueden escribir en el arreglo pero su contenido (la dirección física del arreglo), no es muy válido cuando el arreglo empieza a cambiar de posición (ej.: el arreglo es recargado o movido), por lo tanto el puntero del dato miembro no puede ser modificado (en el caso de que los objetos apuntados son cambiados).

- El arreglo es para salvar/cargar funciones, sólo como una pieza de memoria, estas funciones no pueden leer objetos o apuntadores, solamente una no definida secuencia de bytes.

Un posible uso de esta técnica es el almacenamiento de simples clases, aquellas clases que incluyan solamente miembros datos de tipo base y no apuntadores. Por ejemplo un arreglo puede ser utilizado para almacenar clases tales como ésta:

```
class Coord{
        char* Name;
        int X,Y,Z;
}
```

Sin embargo, para una aplicación real, es necesario modificar este mecanismo para hacerlo más poderoso.

# 5. MODELO DEL OBJETO PERSISTENTE.

## *5.1 ANÁLISIS.*

La finalidad de esta fase consiste en reconocer y describir de manera clara y consistente los elementos de un problema, para establecer la solución más adecuada.

En el análisis TOO (Tecnología Orientada a Objetos), tiene como premisa modelar un sistema a partir del modelo del mundo real, en caso de que el modelo sea abstracto, el analista debe proponer un modelo en base al conjunto de requerimientos establecidos dentro del dominio para iniciar la fase de modelado del sistema.

### 5.1.1 DOMINIO DEL PROBLEMA.

Determina el medio ambiente en el que se reproducirá "el nuevo sistema", por lo cual define el conjunto de restricciones y requerimientos al que será sometido.

#### 5.1.1.1 Definición del problema.

Los registros geofísicos de un pozo son mediciones de propiedades físicas (resistividad, radiactividad, porosidad, velocidad del sonido, etc.), realizadas en las formaciones de un pozo petrolero por medio de una sonda (equipo de medición de fondo), las cuales son utilizadas en diferentes operaciones de la industria petrolera.

Un registro presenta los siguientes elementos de información, de acuerdo al estándar API del formato de encabezados de registros:

1. Nombre del pozo y compañía.
2. Fecha de operación.
3. Datos de profundidad.
4. Datos del lodo.
5. Temperaturas de agujero.
6. Número de camión y localización.
7. Tiempo.
8. Calibración.
9. Número de corrida.
10. Columna de observaciones.
11. Tipo y número de serie de equipo.
12. Escalas y cambios de escala.
13. Otros servicios.
14. Calibración de datos.
15. Presentación de curvas.

16.Calibración de superficie.
17.Calibración anterior a la operación.
18.Calibración posterior a la operación.

Toda la información es almacenada en cintas como se muestra en la Figura 5.1 teniendo como problemas principales:

- Acceso secuencial: Para poder accesar un registro debe de conocerse su localización.
- Archivos planos: La información es almacenada en forma binaria.
- Información voluminosa.
- Información que no puede ser accesada por diferentes aplicaciones.
- Los cambios hechos a la información por alguna aplicación, dejan de existir cuando el programa termina, al menos que la aplicación o el usuario la almacenen.

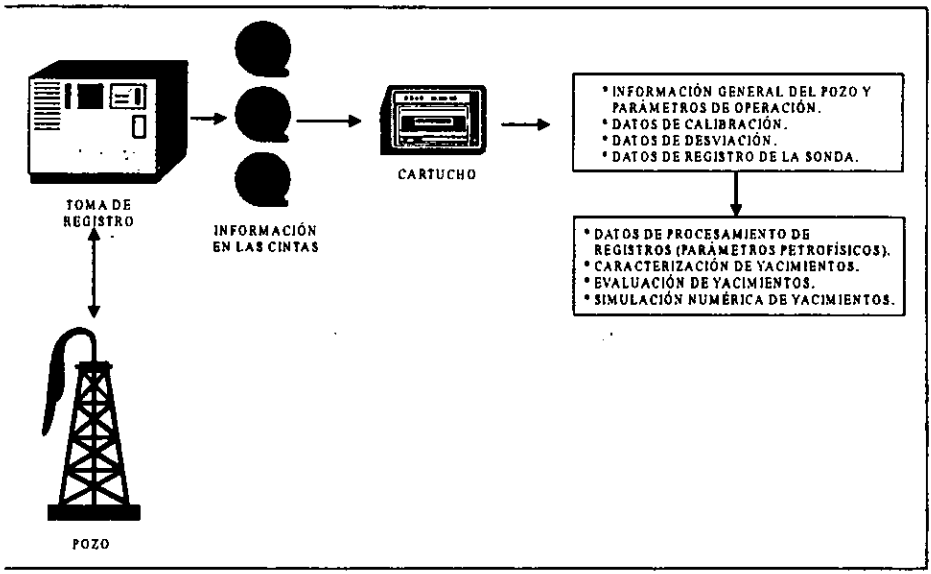



**Figura 5.1.- Información petrolera.**

### 5.1.1.2 Objetivo.

Desarrollar un sistema que permita almacenar información petrolera en una unidad con características de persistencia, para la utilización en diferentes procesos, siguiendo la metodología orientada a objetos en particular la de persistencia.

Para la integración de la información petrolera se va a utilizar la TOO, conformándose a través de entidades que van a tener características y conducta, proporcionándole al usuario, un manejo más claro y exacta de la información, dado que se verá a la información como objetos y no como un conjunto de bytes pertenecientes a un archivo.

Se dotará de persistencia a los objetos. Esto es, que los objetos tendrán la capacidad para que de alguna forma, puedan escribir y leer por ellos mismos, es decir, llevar a cabo el almacenamiento y restauración de datos de manera automática, de forma transparente al usuario, no teniendo que ocuparse de saber si el objeto está en almacenamiento permanente o transitorio o bien si el objeto está apuntado a otros objetos.

Además se utilizará las normas establecidas en los estándares Geoshare y RP66 (DLIS), para que la información pueda ser accesada a través de distintas aplicaciones.

La Figura 5.2 muestra en forma general el papel que juega la implementación de objetos persistentes con información petrolera.

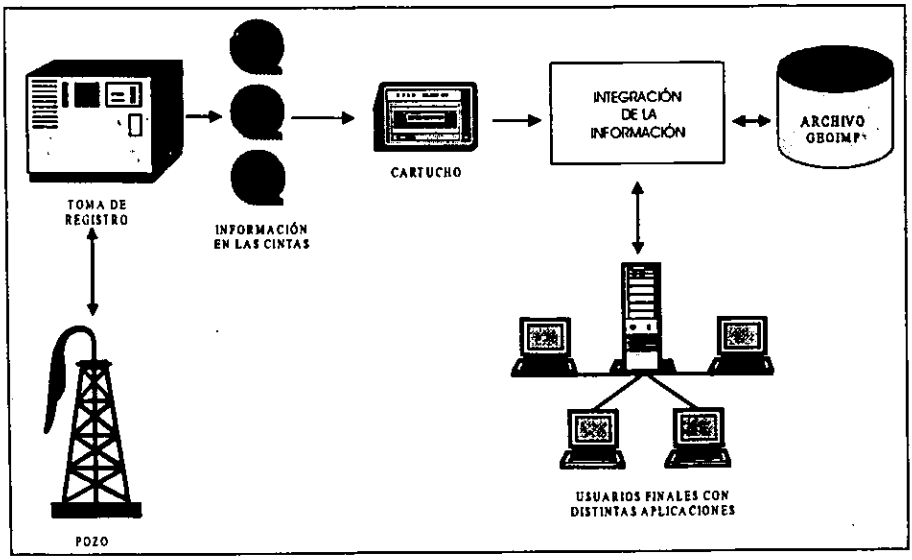



**Figura 5.2.- Implementación de objetos persistentes con información petrolera.**

### 5.1.1.3 Metodología empleada.

Actualmente los sistemas de cómputo creados bajo las técnicas tradicionales se han encontrado con problemas, por eso se está tendiendo a emplear una metodología de análisis y de diseño orientado a objetos, por las ventajas que se ofrecen al desarrollar un sistema bajo esta metodología, tales como: abstracción de datos, encapsulación, reutilización del código, semántica entendible, facilidad para modificar, etc.

La metodología que se presenta en el presente trabajo se le conoce con el nombre de "Desarrollo de Software con Objetos Orientados a Calidad -DSOOC-[22], los motivos por lo que se eligió este método son:

- Es un estándar que se utiliza en la elaboración de sistemas de cómputo en la Línea de Negocios, Adquisición y Procesamiento de Información de Pozos del Instituto Mexicano del Petróleo.
- Está derivada de los trabajos de Grady Booch, Ivan Jacobson y las técnicas de evaluabilidad de software y los requerimientos impuestos en la norma 2167 del Departamento de los Estados Unidos de América, ver Figura 5.3.
- Contempla para cada fase del desarrollo del sistema, la elaboración de un documento, para integrar el manual del sistema.

El esquema general de la metodología DSOOC, se puede apreciar en la Figura 5.4 y la descripción de cada una de las fases de la misma, se presenta a lo largo de este capítulo.

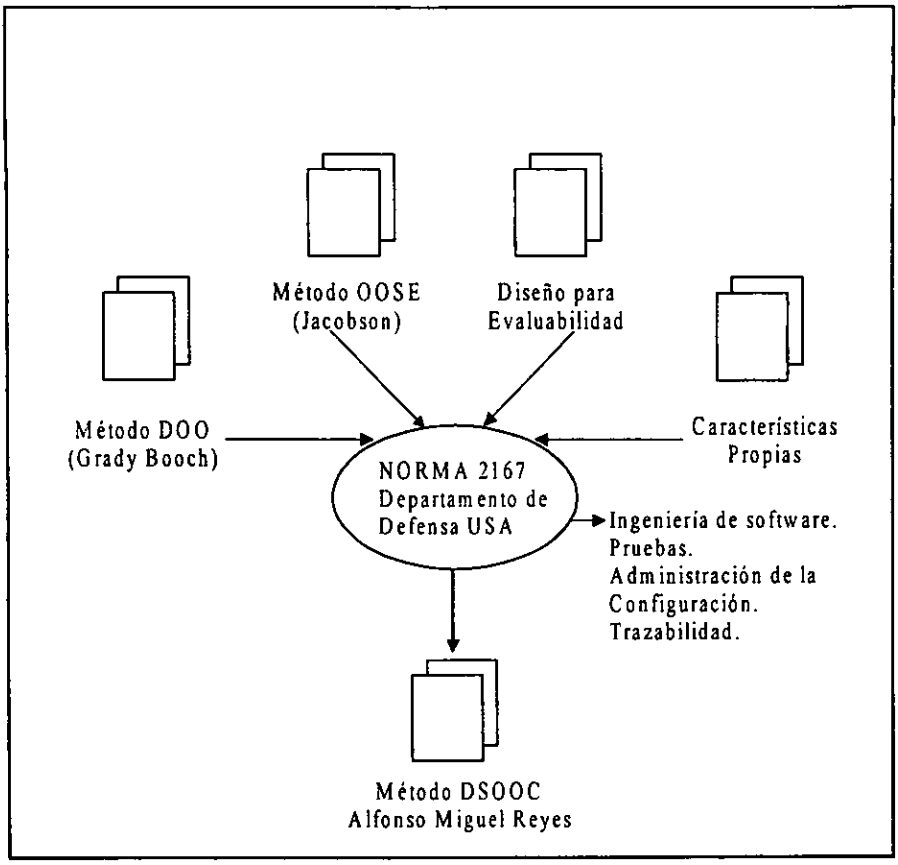



**Figura 5.3.- Método DSOOC.**

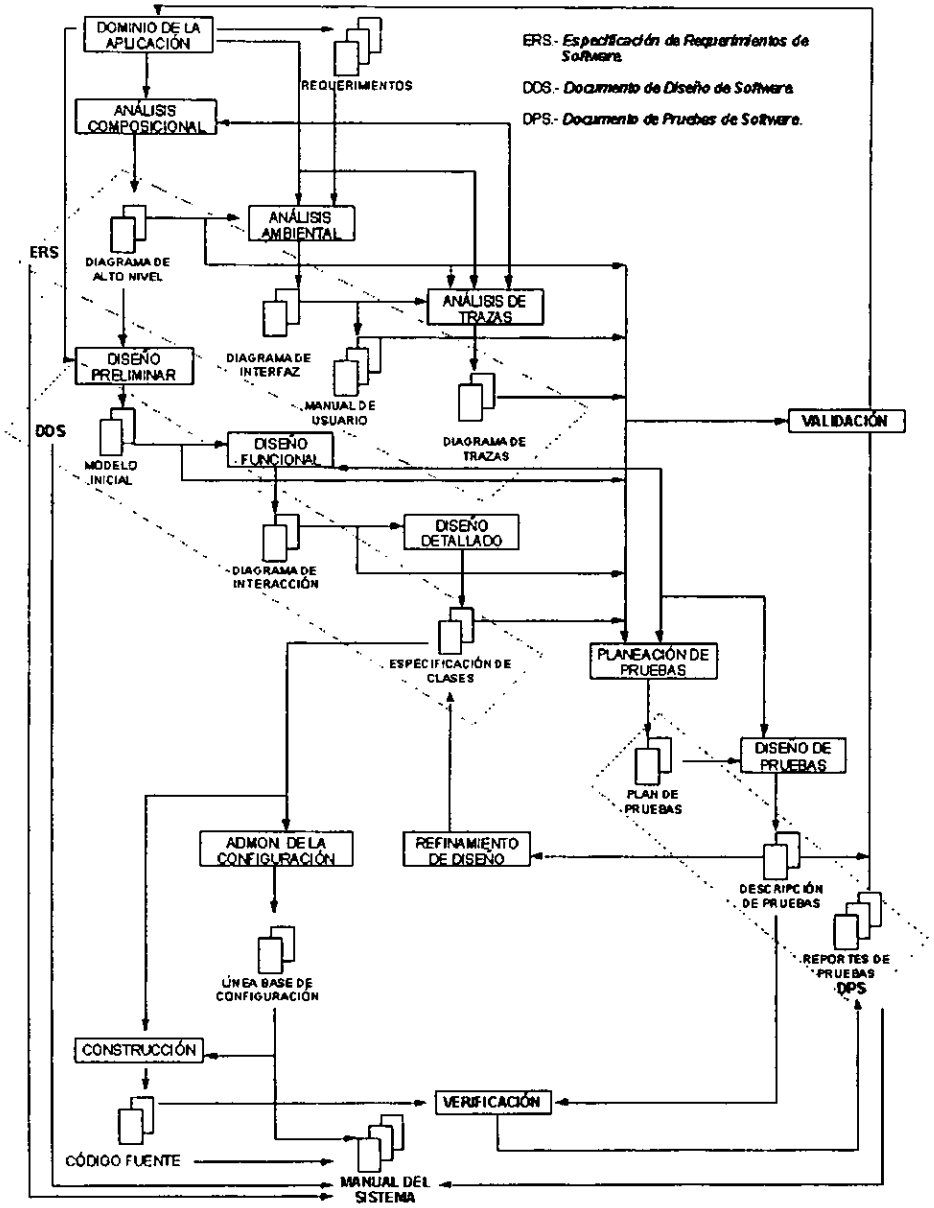



**Figura 5.4.- Metodología DSOOC.**

### 5.1.1.4 Consideraciones.

La integración de la información tiene que guiarse por las normas establecidas en la industria petrolera (capítulo 2):

- Geoshare.
- RP66 (DLIS).

Para el caso del formato Geoshare, se va trabajar con el modelo de datos que se muestra en la Figura 5.5. Éste sólo incluye tres objetos 217-FIELD, 217-WELL-HEADER Y 217-LOG-RUN [23], suficiente para demostrar como se trabaja con el estándar Geoshare.

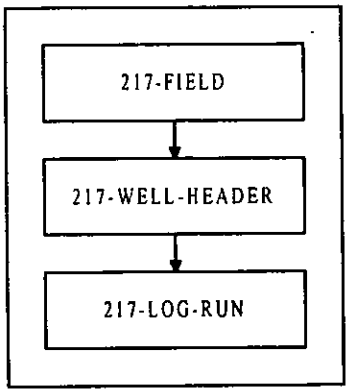



**Figura 5.5.- Modelo de datos Geoshare.**

Los objetos 217-FIELD y 217-WELL-HEADER, son indispensables en cada archivo lógico, independientemente de la información de los pozos que se tengan registrados. Además proporciona la descripción de un pozo, así como la del campo, de una manera general, lo cual da una visión general de la información que se tiene en un archivo lógico.

### 5.1.1.5 Características del Manejo de Información Petrolera en Formato Geoshare.

Un campo tiene desde 1 hasta *n* pozos y un pozo puede tener uno o más registros.

Para la información de cada campo se genera un objeto Geoshare de tipo 217-FIELD.

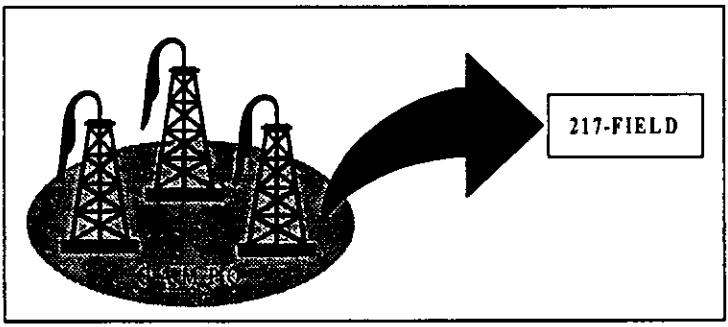



**Figura 5.6.- Objeto 217-FIELD generado a partir de la información del campo.**

Para la información de cada pozo se genera un objeto Geoshare de tipo 217-WELL-HEADER.

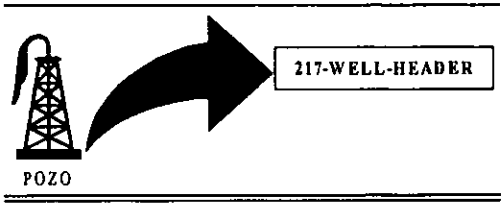



**Figura 5.7.- Objeto 217-WELL-HEADER generado a partir de la información del pozo.**

Para la información de cada toma de registro en un pozo se genera un objeto Geoshare de tipo 217-LOG-RUN.

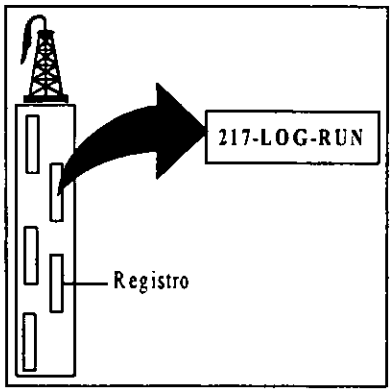



**Figura 5.8.- Objeto 217-Log-Run generado a partir de la información de un registro.**

La información que se encuentra como un objeto Geoshare, es almacenada en disco en forma binaria, siguiendo el formato RP66 (DLIS), sin perder las características de persistencia ni la estructura Geoshare, por lo tanto se obtendrá un archivo híbrido llamado GEOIMP (*.IMP). Como se aprecia en la Figura 5.9.

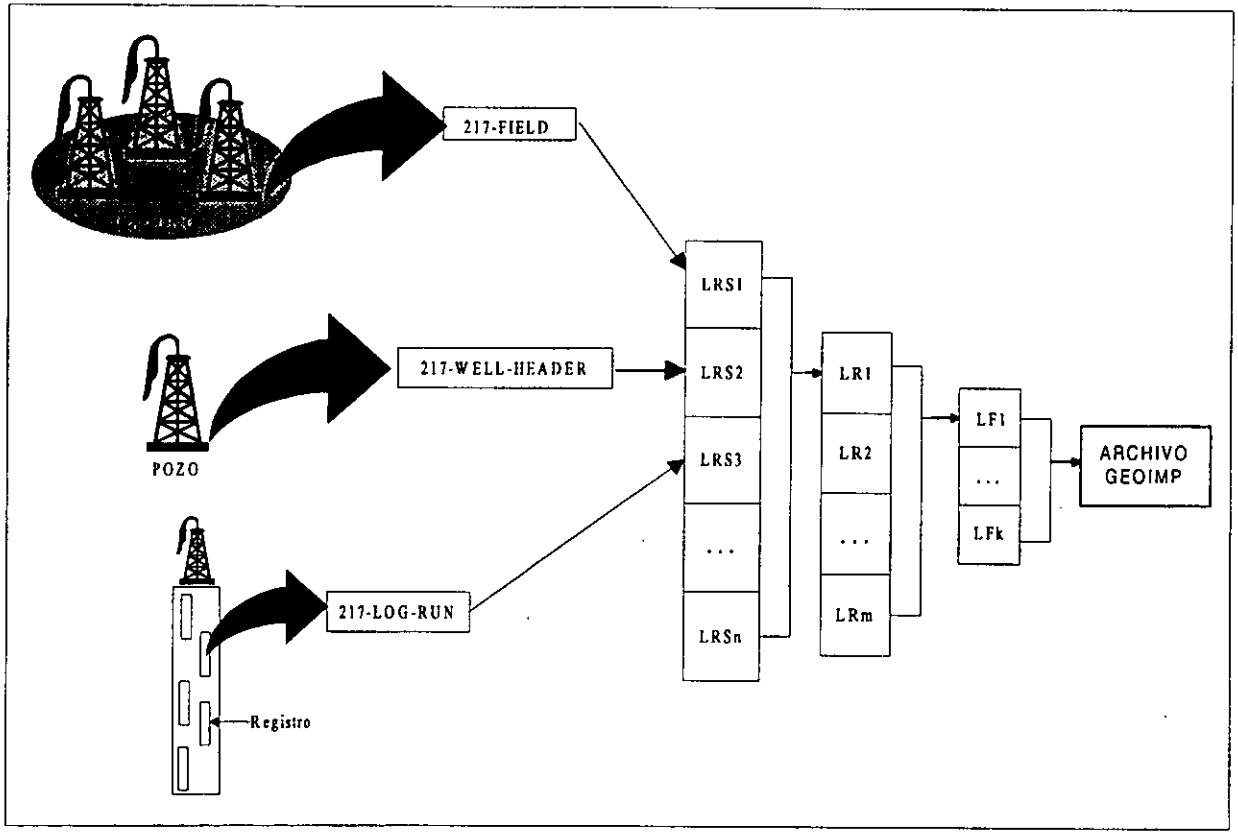



**Figura 5.9.- Estructura de un archivo GEOIMP.**

Donde:

LRS = Segmento de Registro Lógico.
LR = Registro Lógico.
FL = Archivo Lógico.

Estas particiones así como la forma de almacenamiento de la información en forma binaria se establecen en el formato RP66 (DLIS) el cual se explicó en el capítulo 2.

El archivo GEOIMP tendrá la extensión *.IMP, el cual debe cumplir con las siguientes características:

1. Compatible al formato Geoshare.
2. Basado en RP66 (DLIS).
3. Persistente.

#### 5.1.1.6 Restricciones de hardware y software.

Se utilizará una computadora personal con sistema operativo Windows 95, memoria RAM de 16 MB,100 MHz, espacio en disco duro de 2.0 GB, el lenguaje de programación a utilizar es Builder C++.

### 5.1.2 ANÁLISIS COMPOSICIONAL.

El objetivo del análisis composicional es obtener el modelo inicial del sistema, que sirva como base del diseño final en función de los requerimientos solicitados, el cual normalmente es extraído del dominio de la aplicación. En este caso el modelo real no existe por lo cual debe ser propuesto en función de los requerimientos y la naturaleza del problema a resolver.

#### 5.1.2.1 Modelo inicial del sistema.

El modelo inicial propuesto como base al análisis, es mostrado en la Figura 5.10.

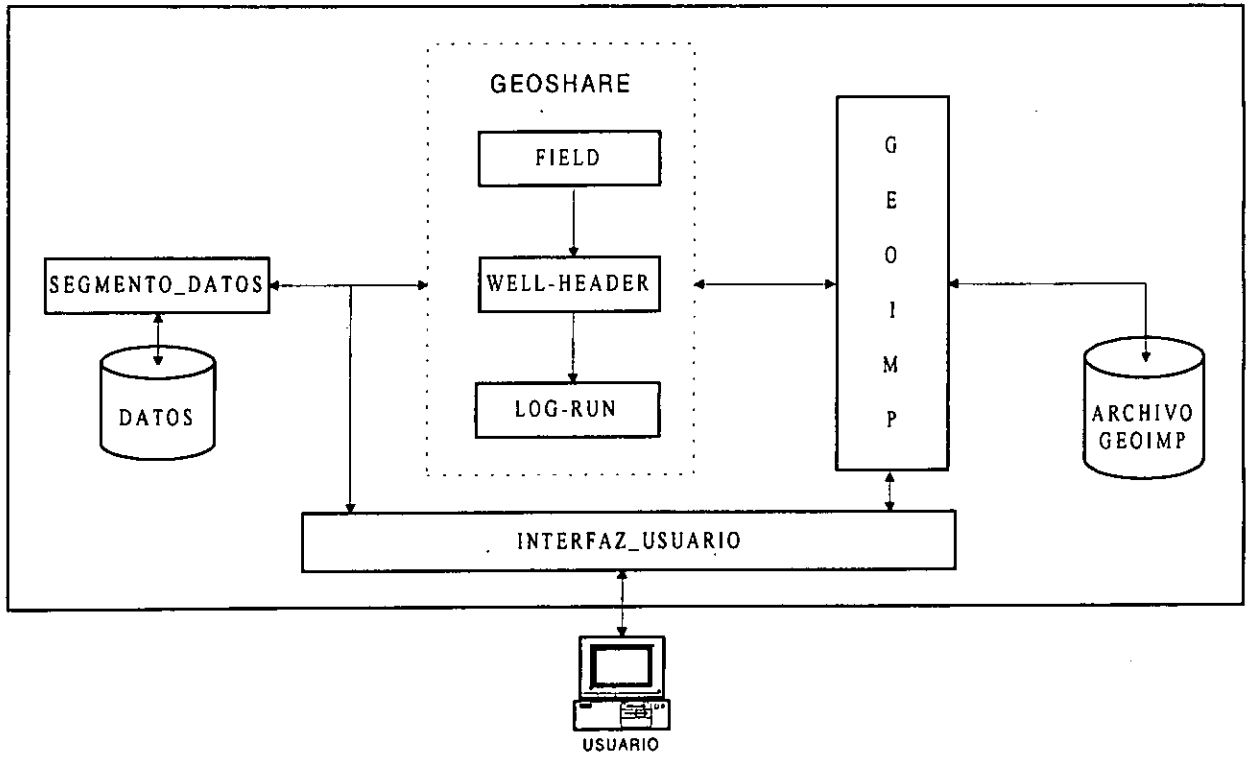



**Figura 5.10.- Diagrama de alto nivel.**

### 5.1.2.2 Especificación preliminar de componentes.

5.1.2.2.1 GEOIMP.

Este elemento realiza los siguientes procesos:

a) Recibe datos que se encuentran encapsulados en formato GEOSHARE.
b) Aplica persistencia a los objetos, así como cuida la integridad de los mismos.
c) Convierte la información de formato RP66 (DLIS) a formato GEOSHARE, y viceversa, conservando el concepto de persistencia.
d) Recupera la información del archivo GEOIMP.
e) Envía la información del archivo GEOIMP a un elemento de tipo GEOSHARE.
f) Verifica y valida el archivo GEOIMP.

5.1.2.2.2 FIELD elemento de Geoshare.

En el momento que recibe información, la encapsula y convierte en un objeto GEOSHARE de tipo 217-FIELD, para ser enviada al elemento GEOIMP, así como también recibe datos del mismo.

*5.1.2.2.2.1 Alcance del objeto 217-FIELD.*

Proporciona información para la transferencia de datos, relacionados a una región geográfica rica en depósito de petróleo, si está actualmente en exploración activa, produciendo petróleo o gas, o abandonada. Además, por medio de subestructuras proporciona información, acerca de secciones transversales mostrando su geología, respecto a las zonas de subsuperficie y una lista de pozos.

*5.1.2.2.2.2 Atributos del objeto 217-FIELD.*

Un objeto 217- FIELD cuenta con los siguientes atributos:

1. FIELD-NAME : El nombre del campo debe poder distinguirse entre varios campos, que han sido descritos en un conjunto de datos Geoshare. Este atributo no puede ser nulo (NN).
2. FIELD-TYPE : Tipo de campo. Se sugiere que sea una combinación de recursos de interés y de todo los estados del campo.
3. DISCOVERY-WELL : Es el identificador de pozo único para cada pozo descubierto en el campo.
4. ZONE-VALUES : Es una lista de que apunta al objeto 217-PARAMETER, que proporcionan las propiedades dependientes de la profundidad como densidad, salinidad o porosidad.
5. AREAL-EXTENT : Es una lista de referencia(s) a un objeto(s) 217-MAP-POLYLINE, el cual define el área de extensión del campo.

6. CARTO-PROJECTION : Es una referencia a un objeto 217-CARTOGRAPHIC-PROJECTION, el cual define la proyección usada. Este atributo no puede ser nulo (NN).
7. WELLS-IN-FIELD : Es una lista de Referencia al objeto 217-WELL-HEADER, el cual describe los pozos en el campo.
8. GEOL-CROSS-SECTION : Es una lista de objetos a 217-GEOLOGICAL-CROSS-SECTION, el cual describe secciones transversales contenidas en el campo.
9. TARGETS : Es una referencia a una lista de objetos 217-WELL-TARGET definido por el campo.

En la Figura 5.11 se puede observar la estructura del objeto 217-FIELD y sus atributos, así como los objetos que son referenciados desde estos atributos.

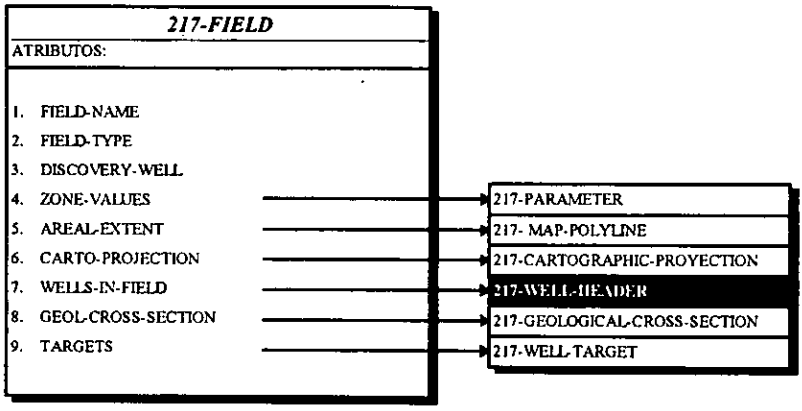



**Figura 5.11.- Atributos del objeto 217-FIELD.**

Es importante señalar que sólo los atributos 1,2 y 7 de este objeto, se tomarán en cuenta, para el desarrollo del sistema, dado que son suficientes para el propósito del mismo.

5.1.2.2.3 WELL-HEADER elemento de Geoshare.

Se encarga de recibir del segmento de datos o desde la interfaz de usuario los datos del pozo, para encapsularlos en un objeto GEOSHARE de tipo 217-WELL-HEADER, y enviar éste al elemento GEOIMP, así como también recibe datos de GEOIMP.

*5.1.2.2.3.1 Alcance del objeto 217- WELL-HEADER..*

Los pozos son agujeros perforados en la tierra, para lograr accesar formaciones subterráneas, junto con su asociados pertrechos: cubierta, cemento, árboles, etc. El uso de un pozo debe cambiar durante su vida y debe incluir alguno o todos las siguientes fases:

- Produce petróleo y/o gas.
- Inyecta agua u otro fluido para estimular la producción para cercano pozo.
- Obtiene información acerca de formaciones para predecir o maximizar la cantera de producción.
- Dispone de agua salada, fluidos de perforación, etc.

Este objeto contiene la descripción de un pozo de una manera general, haciendo referencias a otros objetos.

Topología Pozo.

Es usual que durante la perforación de un pozo se tenga más de una desviación sobre un mismo agujero debido a diferentes factores. Ello ha provocado un desacuerdo en la industria a cerca de si "Pozo" significa un sólo camino desde abajo hacia arriba, o la colección de todos los caminos. Geoshare soporta ambos puntos de vista de un pozo.

- 217-WELLPATH-SEGMENT o "como medida".
- 217-WELLPATH-SECTION-TIE Múltiples desviaciones.

*5.1.2.2.3.2 Atributos del objeto 217-WELL-HEADER..*

Un objeto 217-WELL-HEADER cuenta con los siguientes atributos:

1. UNIQUE-WELL-ID : Identificador único del pozo (UWI) en la emisión. Este atributo no puede ser nulo (NN).
2. UNIQUE-WELL-ID-NAMING-SYSTEM : Identificador para el sistema que envía. Este atributo no puede ser nulo (NN).
3. PLOT SYMBOL : Nombre del símbolo inicial a usar por el pozo. Una lista candidata puede ser encontrada en la reseña de 217-MAP-SYMBOL.
4. DEVIATION SURVEY : Referencia a uno o más objetos de tipo 217-WELL-DEVIATION SURVEY.
5. WELL-NAME : El nombre completo legal del pozo, debe contener al menos un nombre y se le puede concatenar un número.
6. WELL-NAME-NAMING-SYSTEM : El nombre completo legal del sistema, este debe ser al menos un nombre.
7. WELL-NUMBER : Número de pozos.
8. WELL-NUMBER-NAMING-SYSTEM : Número de sistema.
9. PLOT-NAME : Nombre acortado para representación de la venta del pozo.
10. SHORT-NAME : Acortado o nombre informal del pozo.
11. SHORT-NAME-NAMING-SYSTEM : Acortado o nombre informal del sistema.
12. OPERATOR : Nombre de la compañía que actualmente opera el pozo.

13. LICENSEE : Licencia del contrato del pozo.
14. AGENT : Nombre del agente.
15. WELL-CLASS : Clase del pozo.
16. CURRENT-STATUS : Indica el estado del pozo, es decir, la disposición de un pozo en un tiempo dado, desde el tiempo de su concepción hasta su abandono.
17. ORIGINAL-STATUS : Estado original del pozo.
18. PREVIOUS-STATUS : Estado previo del pozo.
19. TOP-LOCATION : Referencia a un objeto 217-MAP-LOCATION dando la localización del agujero en la parte de arriba.
20. TOP-HOLE-LEGALS : Referencia a un objeto de tipo 217-LEGAL-LOCATION describiendo la localización legal del tope del pozo.
21. BOTTOM-LOCATION : Referencia a un objeto 217-MAP-LOCATION dando la localización del agujero en la parte baja.
22. BOTTOM-HOLE-LEGALS : Referencia a un objeto 217-LEGAL-LOCATION describiendo la localización legal del fondo del pozo.
23. OFFSHORE : Referencia a un objeto de tipo 217-WELL-PLATFORM describiendo una plataforma de perforación marítima si el pozo es marítimo.
24. TOTAL-DEPTH : Referencia a un objeto de tipo 217-WELL-TOTAL-DEPTH describiendo la localización del pozo en total profundidad.
25. DATUM-ELEVATION : Elevación del ELEV-PHYSICAL -REF (dato en operación) con respecto a el ELEV - HISTORICAL - REF (dato de referencia) en el control de 217-CARTOGRAPHIC-PROJECTION.
26. ELEV-PHYSICAL-REF : Punto de referencia física (dato en operación) por elevaciones.
27. DEPTH-UNIT : Unidad para profundidades.
28. TIME-UNIT : Unidad de tiempos.
29. ORIGINAL-UNITS-SYSTEM : El sistema de unidades en el cual el dato original fue registrado.
30. GROUND-ELEVATION : Elevación de pozo desde el dato referenciado (no el que esta operando).
31. KELLY-BUSHING - ELEVATION : A la elevación KELLY-BUSHING de la medida del pozo, desde el dato referenciado.
32. CASING-FLANGE-ELEVATION : Elevación de la cubierta base del pozo, medida desde el dato de referencia.
33. CONFIDENTIALITY : Referencia a un objeto de tipo 217-WELL-CONFIDENTIALITY describiendo los puntos claves del pozo.
34. CHECK- SHOT-SURVEY : Referencia a uno o más objetos de tipo 217-WELL-CHECK-SHOT-SURVEY, describiendo algún chequeo de tiro de medición. Si alguno es hecho en el pozo.
35. DRILLING-INFORMATION : Referencia a un objeto de tipo 217-DRILLING-INFORMATION, describe la perforación del pozo.
36. PRIMARY-SOURCE : El nombre de la fuente primaria de información acerca del pozo.
37. PROPRIETARY : Si actualmente este pozo contiene información registrada.
38. DISCOVERY : Si actualmente, este es un pozo descubierto.
39. FAULT : Si actualmente este pozo tiene una sección de falla.
40. LAST-UPDATE : La fecha de la última actualización de la información del pozo.
41. SPUD-UPDATE : Fecha del SPUD del pozo.
42. COMPLETION-DATE : Fecha de finalización del pozo.

43. RIG-RELEASE-DATE : La fecha de estreno de la torre de perforación del pozo.
44. ON-PRODUCTION-DATE : La fecha de inicio de la producción del pozo.
45. CALCULATED-ON-PRODUCTION : La fecha calculada de inicio de la producción del pozo.
46. ON-INJECTION-DATE : La fecha de inicio de inyección del pozo.
47. STATUS-DATE : La fecha del último estado registrado del pozo.
48. FINAL-DRILLING-DATE : Fecha de la perforación final.
49. WELL-PURPOSE : Indica porque el pozo fue perforado o porque será perforado.
50. WELL-FLOWING-MODE : Indica la corriente del flujo del pozo.
51. WELL-FLOWING-FLUID : Indica el flujo del fluido. Un valor de "Múltiple" u "Otro" debe ser explicada usando atributo WELL-REMARKS.
52. SHOW-TYPE : Indica uno o más demostraciones asociadas a un pozo.
53. LOG-RUNS : Una lista a objetos de tipo 217-LOG-RUN, el cual describe la corrida de registros en el pozo.
54. SEISMIC-WELL-TIES : Una lista de referencia a objetos de tipo 217-SEISMIC-WELL-TIE, el cual describe los vínculos del pozo para el pozo.
55. STRAT-COLUMNS : Una referencia a un objeto de tipo 217-STRATIGRAPHIC-COLUMN, cual da el nombre, mención y la lista ordenada de las capas estratigráficas, para este pozo.
56. CORES : Una lista de referencia a objetos de tipo 217-WELL-CORE, el cual describe los núcleos del pozo tomada en este pozo.
57. MARKERS : Una lista de referencia a objetos de tipo 217-WELL-MARKER, el cual describe los pozos marcados asociados con este pozo.
58. PRODUCTION : Una lista de referencia a objetos de tipo 217-WELL-PRODUCTION, el cual describe la producción histórica de este pozo.
59. WELL-REMARKS : Una lista de referencias a objetos 217-WELL-REMARKS, para llevar un conjunto de señalamientos de un índice por profundidad.
60. WELL-TUBULARS : Una lista de referencias a objetos de tipo 217-WELL-TUBULAR, el cual describe los tubulares en el pozo.
61. ZONE-VALUES : Una lista de referencias o objetos de tipo PARAMETER, cada una describe una variable de la zona y sus valores en el pozo.
62. SYNTHETIC-TRACES : Una lista de referencia a 217-SYNTHETIC-TRACE, objetos para datos de rastros sísmicos sintético.
63. TEST : Una lista de referencia a objetos de tipo 217-WELL-TEST, el cual describe datos de prueba del pozo.
64. TARGETS : Nombres relacionados con el atributo NAME de los objetos 217-WELL-TARGET a ser perforados en el futuro o ya perforados en este pozo.
65. PATH-OF-WELL : Referencia a uno o más objetos 217-WELLPATH, demostrando la composición de los caminos del pozo en términos de un número de secciones examinadas.
66. WELLPATH-SEGMENT : Referencia a uno o más objeto de tipo 217-WELLPATH-SEGMENT . Este demuestra la composición de los caminos del pozo en términos de los individuales segmentos perforados.
67. DRILL-RUNS : Referencia a una o más objetos 217-DRILL-RUN, para información acerca de perforación en ejecución. Los registros obtenidos durante la perforación y el equipo usado para los agujeros realizados.
68. COMPANY : El nombre del dueño de la organización o que tenga derechos para operar el pozo, o bien que tenga campos o contrato de arrendamiento. Este es típicamente una compañía nacional o privada.

69. WELLBORE-SHARE : La forma del asociado pozo-perforado, cuando hay solo uno, o cuando un pozo-agujeros, es predominantemente asociado con el pozo. Este sería el mismo pozo-taladrado, asociado con algunas marcas, registros, historia de producción, desviación y pruebas de chequeo.

En la Figura 5.12 y en la Figura 5.13, se puede observar la estructura del objeto 217-WELL-HEADER y sus atributos, así como los objetos que son referenciados desde estos atributos.

Es fundamental indicar que únicamente los atributos 1, 5, 7, 10, 16, 17, 18, 24, 25, 26, 27, 28, 29, 30, 31, 32, 40, 53 y 68 de este objeto se tomarán en cuenta, para el desarrollo del sistema, debido a que son suficientes para el propósito del mismo.

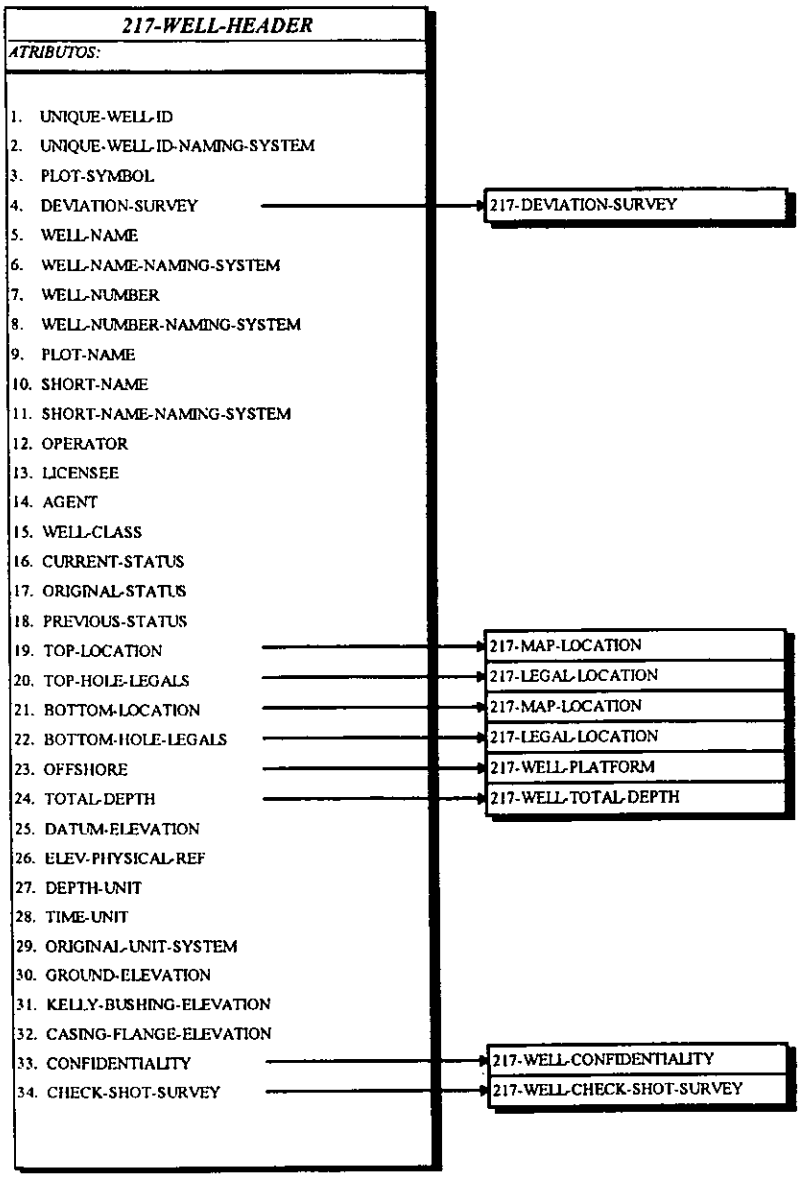



**Figura 5.12.- Atributos del objeto 217-WELL-HEADER.**

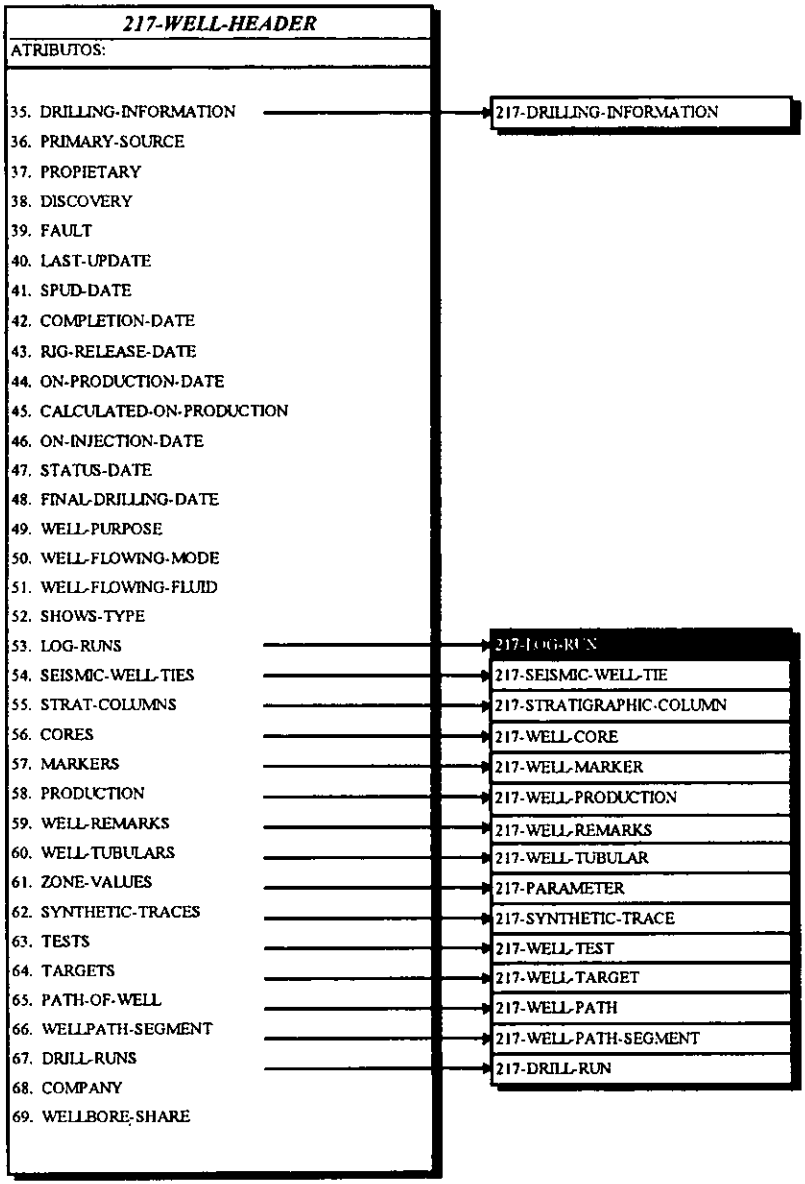



**Figura 5.13.- Atributos del objeto 217-WELL-HEADER.**

5.1.2.2.4 LOG_RUN elemento de Geoshare.

Se encarga de recibir del segmento de datos o desde la interfaz de usuario los datos de la corrida de un registro geofísico. Una vez obtenida la información, la encapsula en un objeto GEOSHARE de tipo 217-LOG-RUN, y la envía al elemento GEOIMP, así como también recibe datos del mismo.

*5.1.2.2.4.1 Alcance del objeto 217- LOG-RUN.*

Los objetos en este conjunto describen una visita de una compañía de explotación a un lugar pozo, usualmente asociado con una simple orden de servicio.

*5.1.2.2.4.2 Atributos del objeto 217- LOG-RUN.*

Un objeto 217-LOG-RUN puede contar con los siguientes atributos:

1. RUN-NUMBER : Número de corrida de registro. Este debe ser único a lo largo de todos las corridas de registros asociados con el pozo. Este atributo no puede ser nulo (NN).
2. SERVICE-COMPANY : Nombre de la compañía de servicio que proporciona el servicio de explotación
3. RUN-DATE : Día en el cual la corrida del registro fue iniciada.
4. LOGGING-UNIT-NUMBER : Identificación del camión o unidad que realiza la explotación.
5. LOGGING-UNIT-LOCATION : Casa base de la unidad o camión.
6. RECORDER : Nombre de la persona que hace las grabaciones de los registros.
7. WITNESS : Nombre de la persona testigo de la actividad de la explotación.
8. ELEV-REF-HEIGHT : Elevación de la ELEV-PHYSICAL-REF con respecto a la ELEV-HISTORYCAL-REF en el controlador 217-CARTOGRAPHIC-PROJECTION.
9. ELEV-PHYSICAL-REF : Nombre del dato en operación.
10. MUD-TYPE : Tipo de lodo de perforación en uso en el tiempo de la corrida de la explotación.
11. MUD-SALINITY : Salinidad del lodo en o durante la corrida de la explotación.
12. MUD-DENSITY : Densidad del lodo en el tiempo de la corrida de la explotación.
13. MUD-VISCOSITY : Viscosidad del lodo en el tiempo de la corrida de la explotación.
14. MUD-FLUID-LOSS : Medida de la pérdida de fluidos hasta la formación.
15. MUD-ACITY : Acidez del lodo durante la corrida de la explotación. La unidad es asumida a ser PH. La unidad no es requerida desde que RP66 no soporta "PH".
16. MUD-RESIST : Resistividad del lodo básico.
17. MUD-FILTRATE-RESIST : Resistividad del lodo filtrado.
18. MUD-CAKE-RESIST : Resistividad del bloque del lodo.
19. MUD-RESIST-SOURCE : Descripción de la fuente de la resistividad del lodo.
20. MUD-BHT-RESIST : Básica resistividad del lodo a la temperatura del fondo del hoyo.
21. MUD-BHT-FILTRATE-RESIST : Resistividad del lodo filtrado a la temperatura del fondo del hoyo.

22. MUD-BHT-CAKE-RESIST : Resistividad del bloque del lodo a la temperatura del fondo del hoyo.
23. MUD-BHT-RESIST-SOURCE : Fuente de la resistividad del lodo a la temperatura del fondo del hoyo. Este es usualmente el nombre de la herramienta, desde el cual la resistividad del lodo es derivada.
24. MUD-SOURCE : Origen del lodo usado para la medición de atributos del lodo, tal como MUD-TYPE, MUD-SALINITY, etc., típicos valores son "MUD-PIT", "MUD-PUMP", "MUD-LOGGER", "CUSTOMER", etc.
25. MUD-TEMPERATURE : Temperatura de la muestra del lodo en el tiempo, el valor para el atributo MUD-RESIST está medido. La resistividad del lodo cambia en una manera previsible con la temperatura, estos valores deben ser medidos al mismo tiempo.
26. MUD-FILTRATE-SOURCE : Fuente del lodo desde el cual el filtro es tomado para la medición de MUD-FILTRATE-RESIST. Si hay una pérdida, los lectores deben de asumir el que mismo valor de MUD-SOURCE.
27. MUD-FILTRATE-TEMPERATURE : Temperatura de la muestra de lodo filtrado en el tiempo, el valor para el MUD-FILTRATE-RESIST, atributo es medido. La resistividad del lodo filtrado cambia de una manera previsible con la temperatura filtrada, estos valores deben ser medidos en el mismo tiempo. Si hay una pérdida, los lectores deben asumir que es el mismo valor de MUD-TEMPERATURE.
28. MUD-CAKE-SOURCE : Fuente del lodo desde el cual el bloque de lodo es llevado para la medición de MUD-CAKE-RESIST. Si hay una pérdida, los lectores deben asumir que es el mismo valor de MUD-SOURCE.
29. MUD-CAKE-TEMPERATURE : Temperatura de la muestra del bloque de lodo en el tiempo, el valor para el atributo MUD-CAKE-RESIST es medido. La resistividad en el bloque del lodo cambia de una manera previsible, con la temperatura del bloque, así que estos valores deben ser medidos en el mismo tiempo. Si hay una pérdida, el lector debe asumir que es el mismo valor de MUD-TEMPERATURE.
30. TIME-CIRCULATION-STOP : Tiempo en el cual la circulación se detiene.
31. BOTTOM-TEMP-MAX : La temperatura máxima registrada desde el fondo del hoyo.
32. LOG-PASSES : Una lista de referencias a objetos de tipo 217-LOG-PASS
33. DRILL-MEASURED-DEPTH : Profundidad total de la perforación.
34. LOG-MEASURED-DEPTH : Profundidad máxima registrada.
35. BOTTOM-PRESSURE-MAX : La presión más alta del fondo del pozo registrada sobre todas las pasadas en la corrida de un registro.
36. SURFACE-HOLE-TEMPERATURE : Temperatura del hoyo al nivel de la tierra.
37. WELLPATH-SEGMENT : Nombre de un segmento de la trayectoria de un pozo, para el cual esta corrida de registro da los datos que son adquiridos o esperados.
38. PLANNING-DATA : Verdadero si el dato esta planeado, esperado o modelado. Si el atributo tiene valor de falso o está perdido, el dato es asumido para ser realmente un dato registrado, o el resultado de procesar en un dato realmente registrado.
39. SERVICE-ORDER : Número de orden de servicio del registrador para esta corrida de registro.
40. PERMANENT-DATUM-ELEVATION : Elevación del dato permanente con respecto a la ELEV-HISTORICAL-REF (dato de referencia), en el controlador 217-CARTOGRAPHIC-PROJECTION. Si el dato permanente está a 15 metros arriba de la elevación de referencia, entonces este atributo tendría el valor de 15.0(m).
41. PERMANENT-DATUM : Identidad del dato permanente. Este es proporcionado solamente para que la información del dato permanente, pueda ser referida con el dato.

Geoshare no hace uso del dato de información permanente para resolución de elevaciones.

En la Figura 5.14 y en Figura 5.15 se puede observar la estructura del objeto 217-LOG-RUN y sus atributos, así como los objetos que son referenciados desde estos atributos.

Es importante mencionar que para este objeto se tomarán en cuenta todos sus atributos, para el desarrollo del sistema.

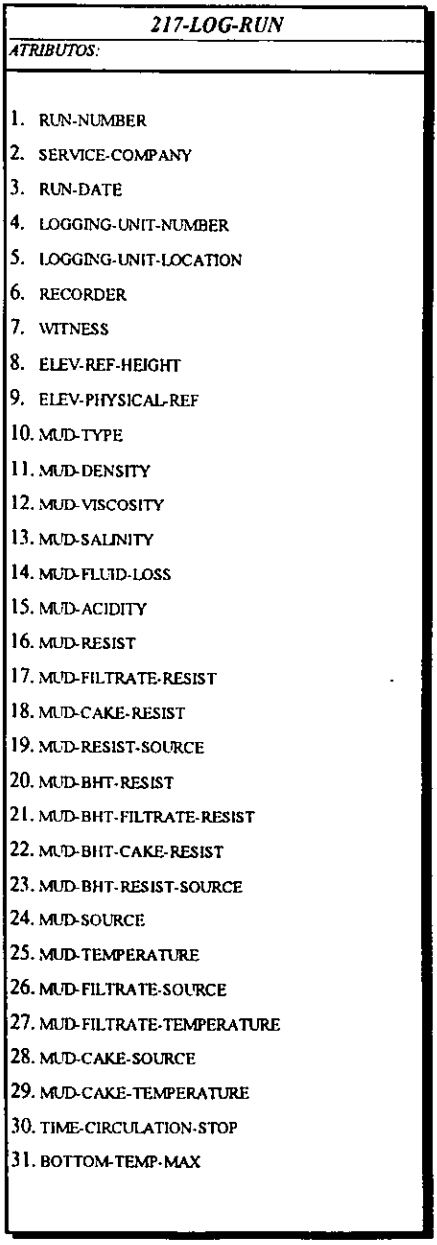



**Figura 5.14.- Atributos del objeto 217-LOG-RUN.**

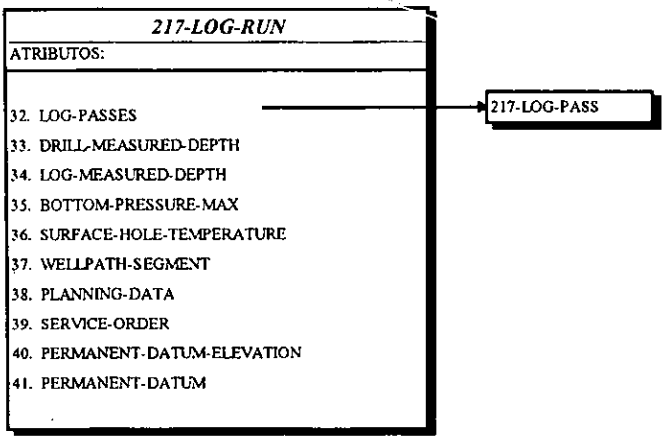



**Figura 5.15.- Atributos del objeto 217-LOG-RUN.**

5.1.2.2.5 SEGMENTO_DATOS.

Extrae de los archivos fuentes (LAS, LIS, TXT), los datos de interés para el archivo GEOIMP, para el sistema sólo se usará los archivos tipo TXT.

5.1.2.2.6 INTERFAZ_USUARIO.

Es el componente encargado de administrar y controlar todos los eventos realizados por el usuario, así como el de interactuar con los componentes del sistema.

### 5.1.3 ANÁLISIS AMBIENTAL.

El análisis ambiental tiene como objetivo obtener del dominio de la aplicación, el conjunto de eventos o acciones que permitan identificar los escenarios que deben ser reproducidos por el sistema y que además conforman la interacción de los usuarios. Este permite elaborar un prototipo o lista de todos los posibles eventos originados por el usuario.

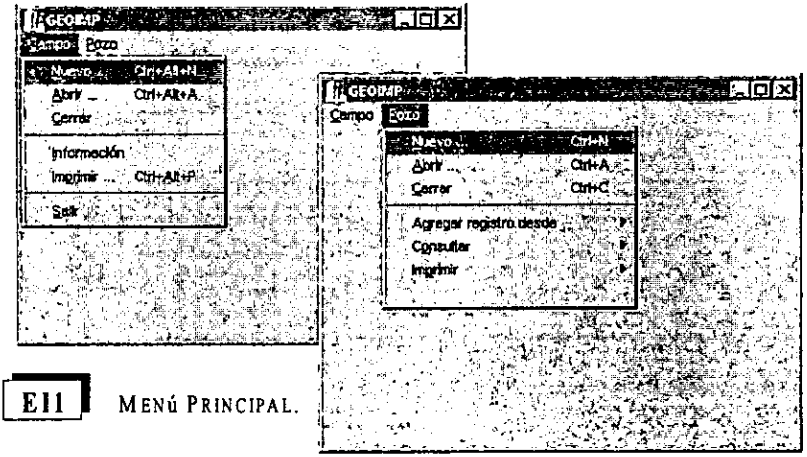
En esta fase se conjuntan todas las pantallas de interacción del usuario y del sistema, en un catálogo que establece las bases para definir las características de operación y a la vez comenzar el manual respectivo.

#### 5.1.3.1 Elementos de Interfaz.

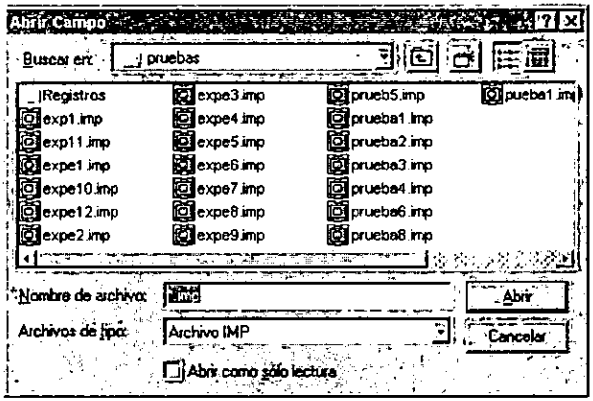
En ella se definen los escenarios como unidades únicas, las cuales resultan ser fáciles de manejar, controlar y modificar durante la fase de diseño, que además resultan ser la base para el desarrollo de un prototipo del sistema. La Figura 5.16 proporciona una lista de los elementos básicos de la interfaz.

| ID | NOMBRE | NEMÓNICO |
|---|---|---|
| EI1 | Menú principal | EI1 |
| EI2 | Selección de un archivo *.imp | EI2 |
| EI3 | Selección de un archivo *.txt | EI3 |
| EI4 | Crear un archivo *.imp | EI4 |
| EI5 | Petición de sobreescritura | EI5 |
| EI6 | Información: no puede abrir el archivo | EI6 |
| EI7 | Información: no existe el pozo | EI7 |
| EI8 | Información: no existen registros | EI8 |
| EI9 | Información: número de registros anexados | EI9 |
| EI10 | Confirmación de salida del sistema | EI10 |
| EI11 | Confirmación de cerrar el pozo | EI11 |
| EI12 | Solicitud del ID del pozo | EI12 |
| EI13 | Solicitud de datos de un campo | EI13 |
| EI14 | Solicitud de datos de un pozo | EI14 |
| EI15 | Solicitud de datos de un registro | EI15 |
| EI16 | Muestra datos de un campo | EI16 |
| EI17 | Muestra datos de un pozo | EI17 |
| EI18 | Muestra datos de un registro | EI18 |
| EI19 | Muestra datos de un conjunto de registros | EI19 |
| EI20 | Diálogo de impresión | EI20 |

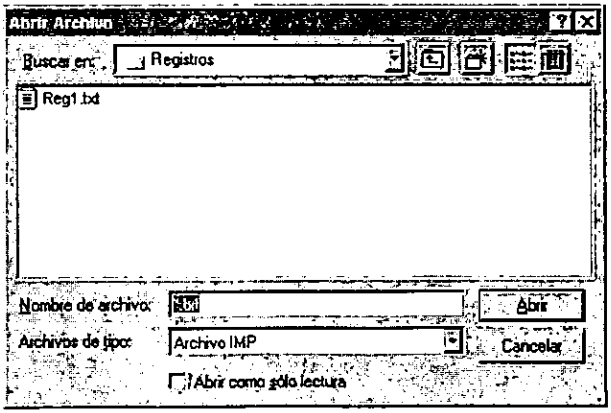
**Figura 5.16.- Tabla de elementos de interfaz.**



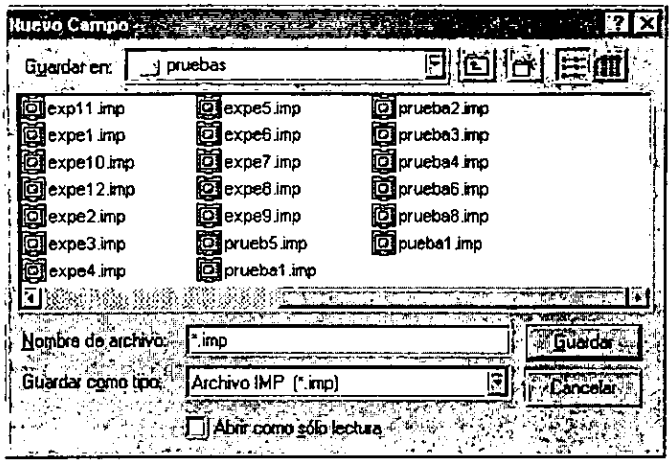

**E11** MENÚ PRINCIPAL.



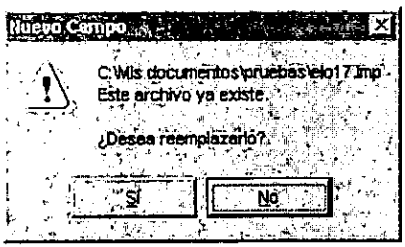

**E12** SELECCIÓN DE UN ARCHIVO *.IMP.



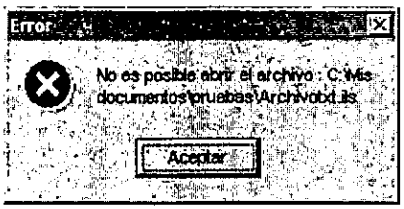

**E13** SELECCIÓN DE UN ARCHIVO *.TXT.



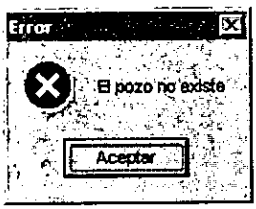

**EI4** CREAR UN ARCHIVO *.IMP.



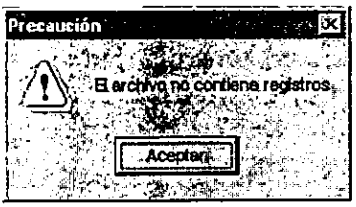

**EI5** PETICIÓN DE SOBREESCRITURA.



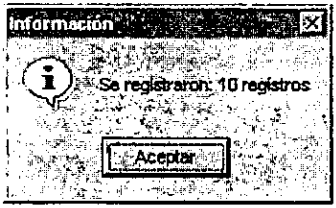

**EI6** INFORMACIÓN: NO PUEDE ABRIR EL ARCHIVO.



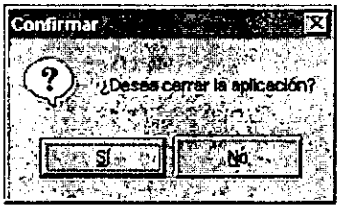

**EI7** INFORMACIÓN: NO EXISTE EL POZO.



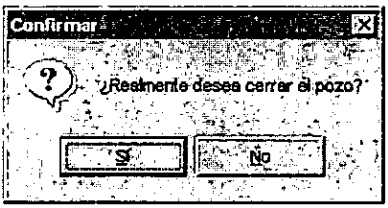

**EI8** INFORMACIÓN: NO EXISTEN REGISTROS.



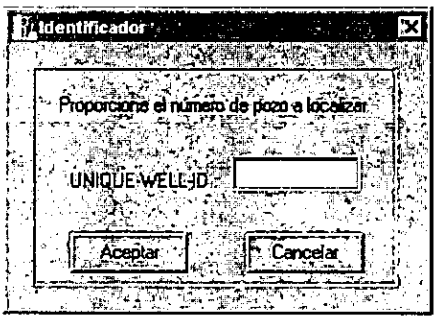

EI9 INFORMACIÓN NÚMERO DE REGISTROS ANEXADOS.



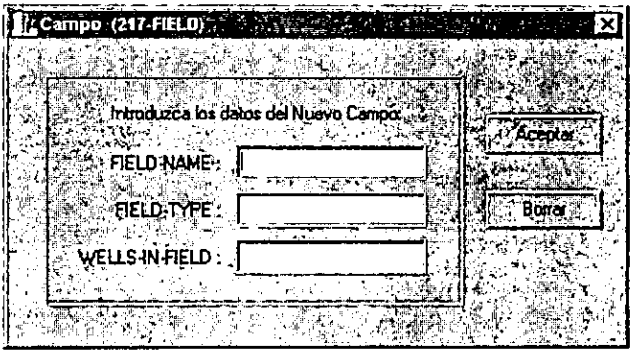

EI10 CONFIRMACIÓN DE SALIDA DEL SISTEMA.



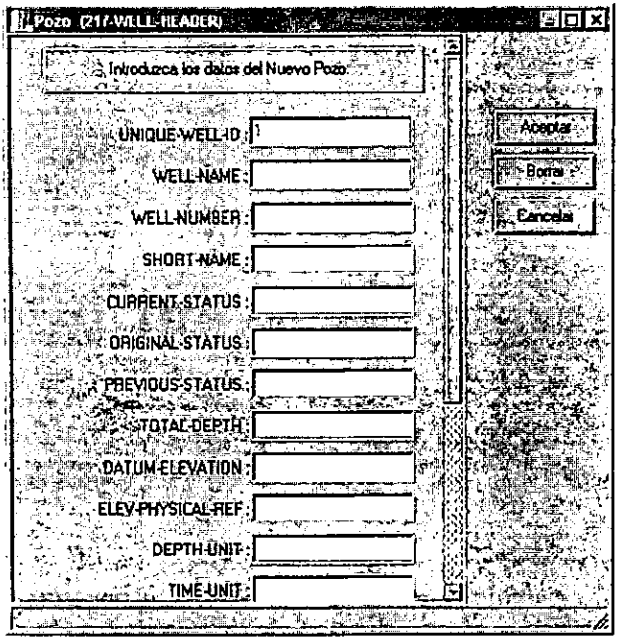

EI11 CONFIRMACIÓN DE CERRAR EL POZO.



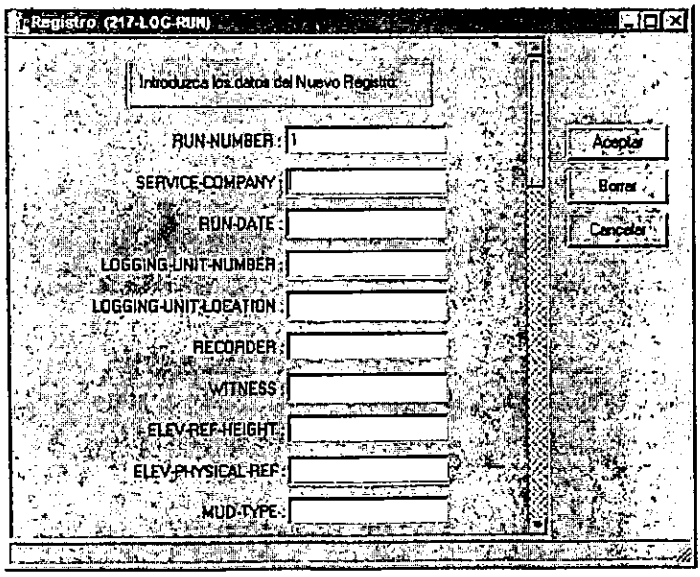

EI12 SOLICITUD DEL ID DEL POZO.



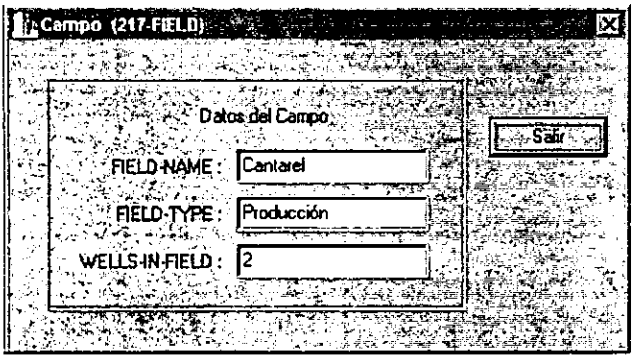

EI13 SOLICITUD DE DATOS DE UN CAMPO.



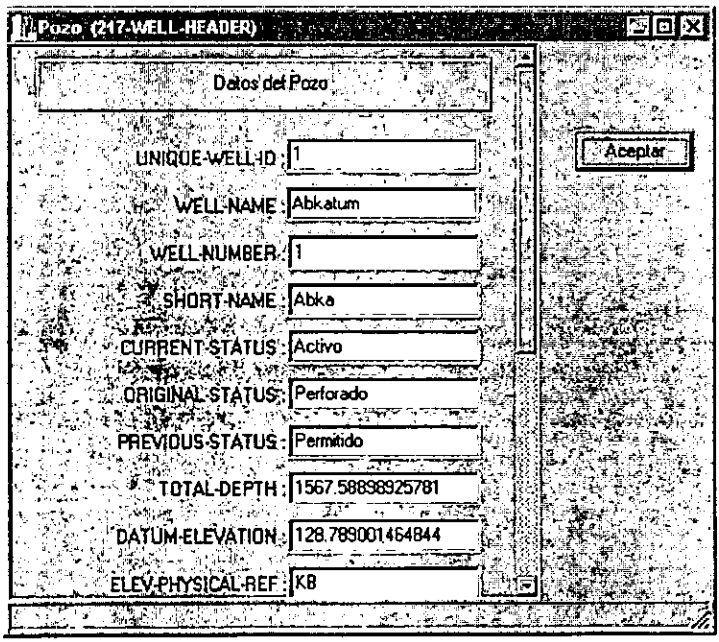

**E114** SOLICITUD DE DATOS DE UN POZO.



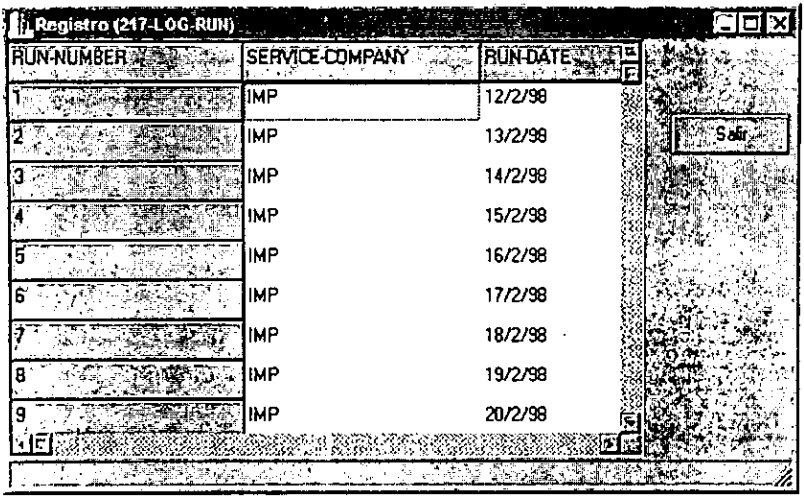

**E115** SOLICITUD DE DATOS DE UN REGISTRO.



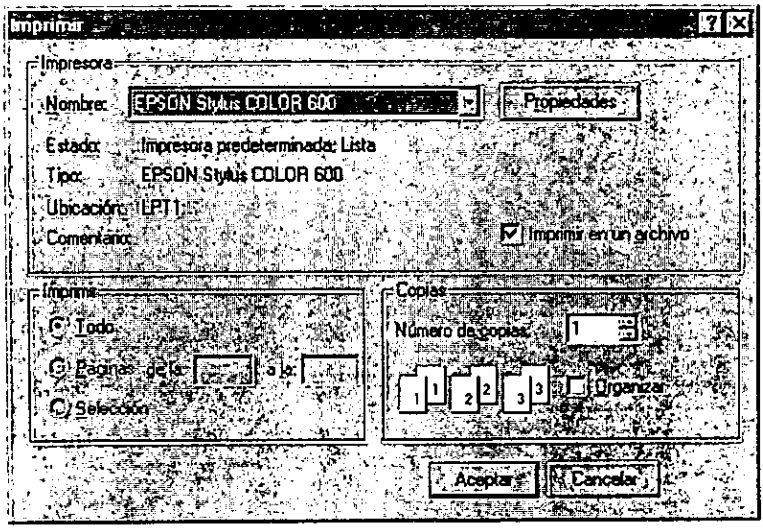

**EI16** MUESTRA DATOS DE UN CAMPO.




**EI17** MUESTRA DATOS DE UN POZO.




**E118** MUESTRA DATOS DE UN CONJUNTO DE REGISTROS.




**E119** DIÁLOGO DE IMPRESIÓN.

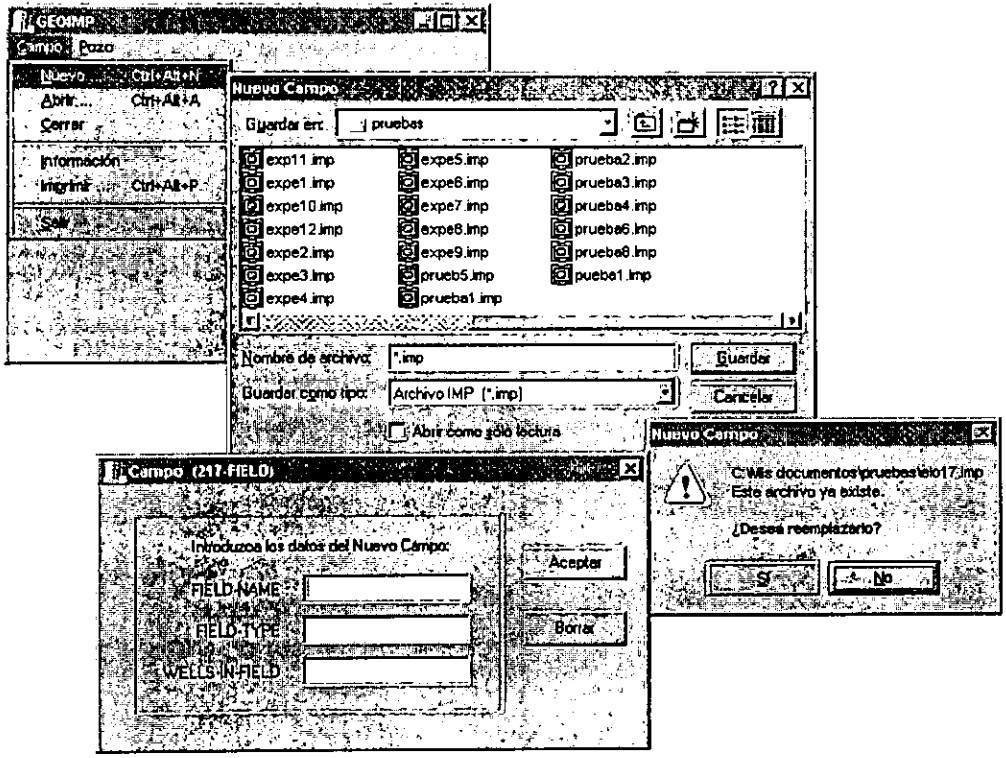
### 5.1.3.2 Diagramas de Interfaz

Es una lista de las principales interfaces del sistema con los usuarios. Los diagramas muestran el conjunto de escenarios (elementos de interfaz) que se activan al ocurrir un evento determinado. La Figura 5.17 proporciona una lista de las interfaces fundamentales del sistema con el usuario.

| ID | NOMBRE | NEMÓNICO |
|---|---|---|
| DI1 | Nuevo Campo | DI1 |
| DI2 | Abrir Campo | DI2 |
| DI3 | Cerrar Campo | DI3 |
| DI4 | Información Campo | DI4 |
| DI5 | Imprimir Campo | DI5 |
| DI6 | Salir Campo | DI6 |
| DI7 | Nuevo Pozo | DI7 |
| DI8 | Abrir Pozo | DI8 |
| DI9 | Cerrar Pozo | DI9 |
| DI10 | Agrega Registro desde Teclado | DI10 |
| DI11 | Agrega Registro desde Archivo | DI11 |
| DI12 | Consulta Registro | DI12 |
| DI13 | Consulta Pozo | DI13 |
| DI14 | Imprimir Registro | DI14 |
| DI15 | Imprimir Pozo | DI15 |

**Figura 5.17.- Tabla de diagramas de interfaz**

DIAGRAMA DE INTERFAZ

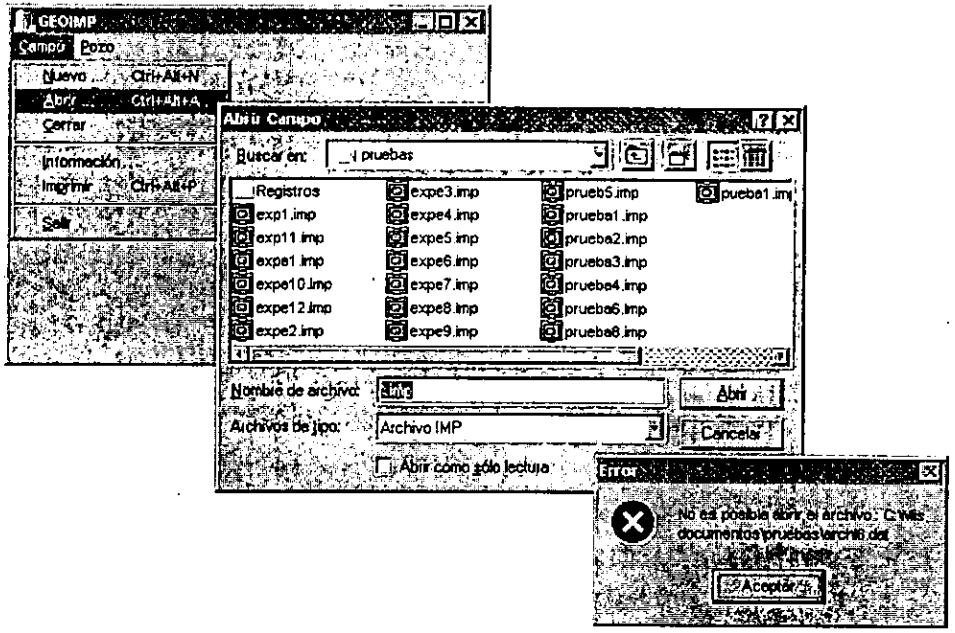



**DI1:**

**NUEVO CAMPO**

Al seleccionar la opción Nuevo, se abre una ventana que pide el nombre y localización del nuevo archivo, en caso de que ya exista se pregunta si se quiere reemplazarlo. En seguida el usuario debe introducir los datos del campo.

DIAGRAMA DE INTERFAZ

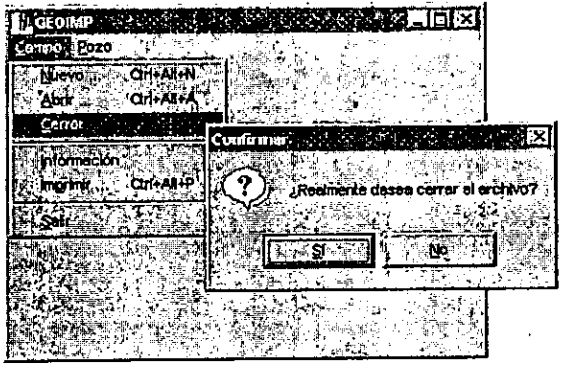



**D12:**

**ABRIR CAMPO**

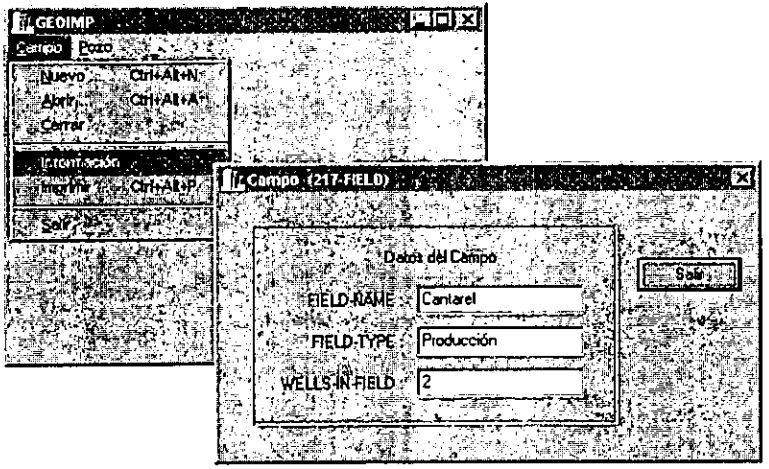
Al elegir la opción Abrir, se muestra una ventana que solicita al usuario la unidad, directorio y el nombre del archivo (*.imp).

DIAGRAMA DE INTERFAZ




**DI3:**
**CERRAR CAMPO**
Al seleccionar la opción Cerrar, se presenta una ventana que pregunta al usuario si realmente desea abandonar el archivo.

DIAGRAMA DE INTERFAZ

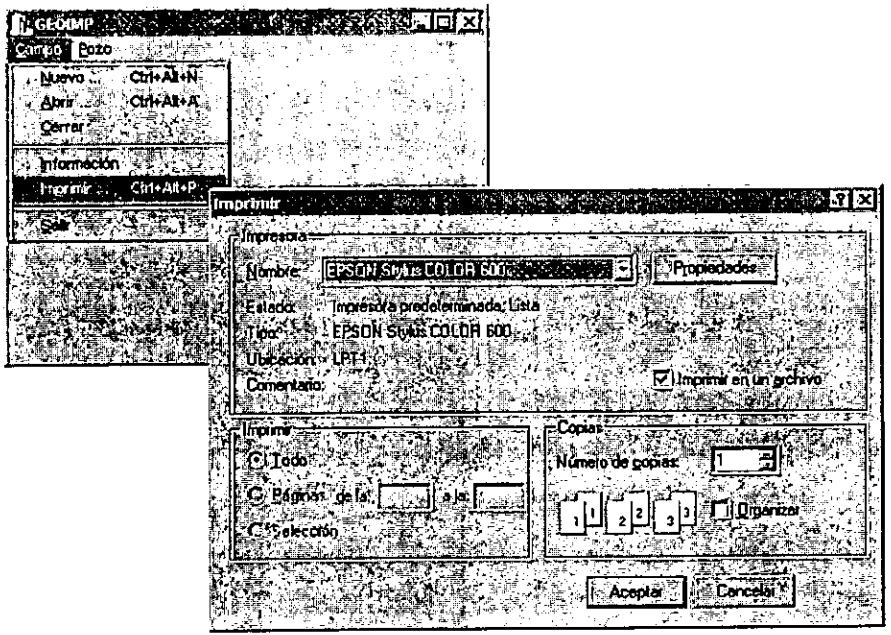



**D14:**

**INFORMACIÓN CAMPO**

Al escoger la opción Información, se presenta una ventana que muestra todos los datos referente al campo actual.

DIAGRAMA DE INTERFAZ

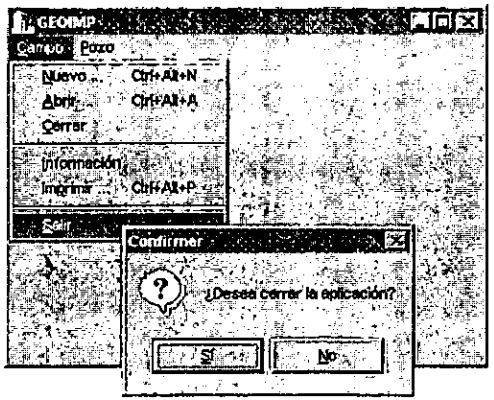



**DI5:**

**IMPRIMIR CAMPO**

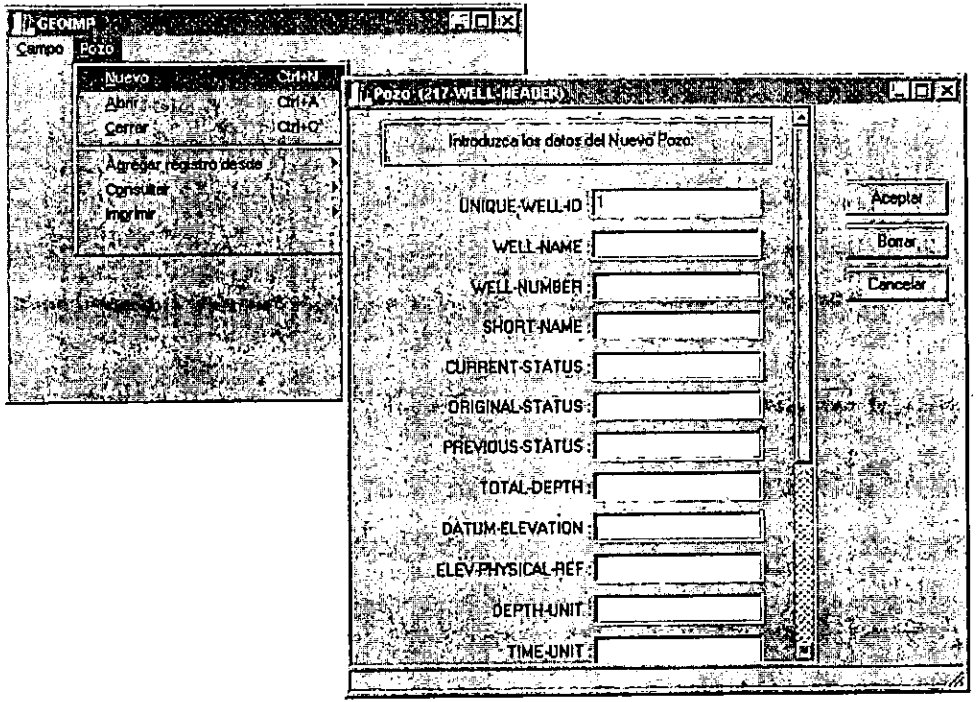
Al elegir la opción Imprimir se presenta una ventana en la que se tiene la alternativa de imprimir a un archivo o a la impresora. Si se imprime a un archivo se pide el nombre de éste.

DIAGRAMA DE INTERFAZ




**DI6:**
**SALIR CAMPO**
Al seleccionar la opción Salir, se presenta una ventana que pregunta al usuario si realmente desea salir del sistema.

DIAGRAMA DE INTERFAZ

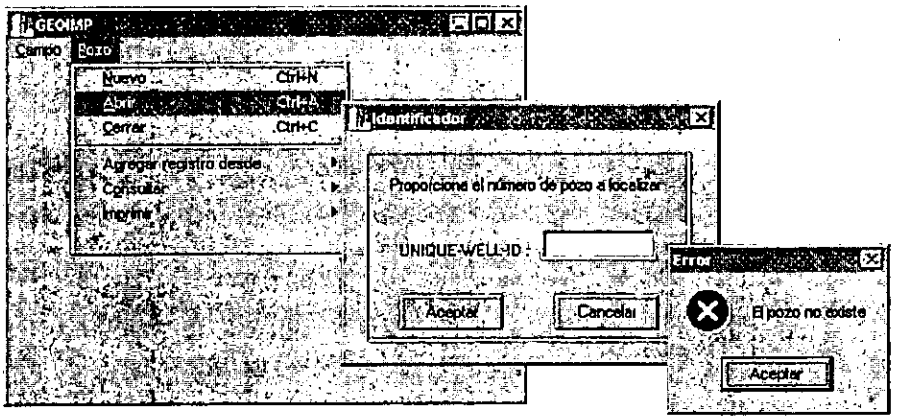



**DI7:**

**NUEVO POZO**

Al seleccionar la opción Nuevo, se presenta una ventana que solicita al usuario que introduzca la información del pozo. Nota: La información que se introduzca se guardará dentro del campo actual.

DIAGRAMA DE INTERFAZ

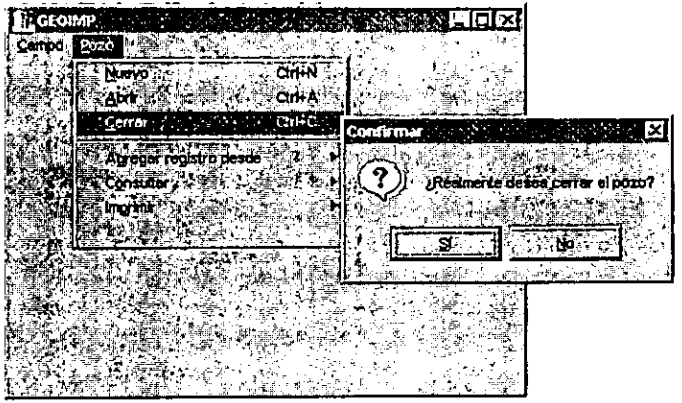



**DI8:**

**ABRIR POZO**

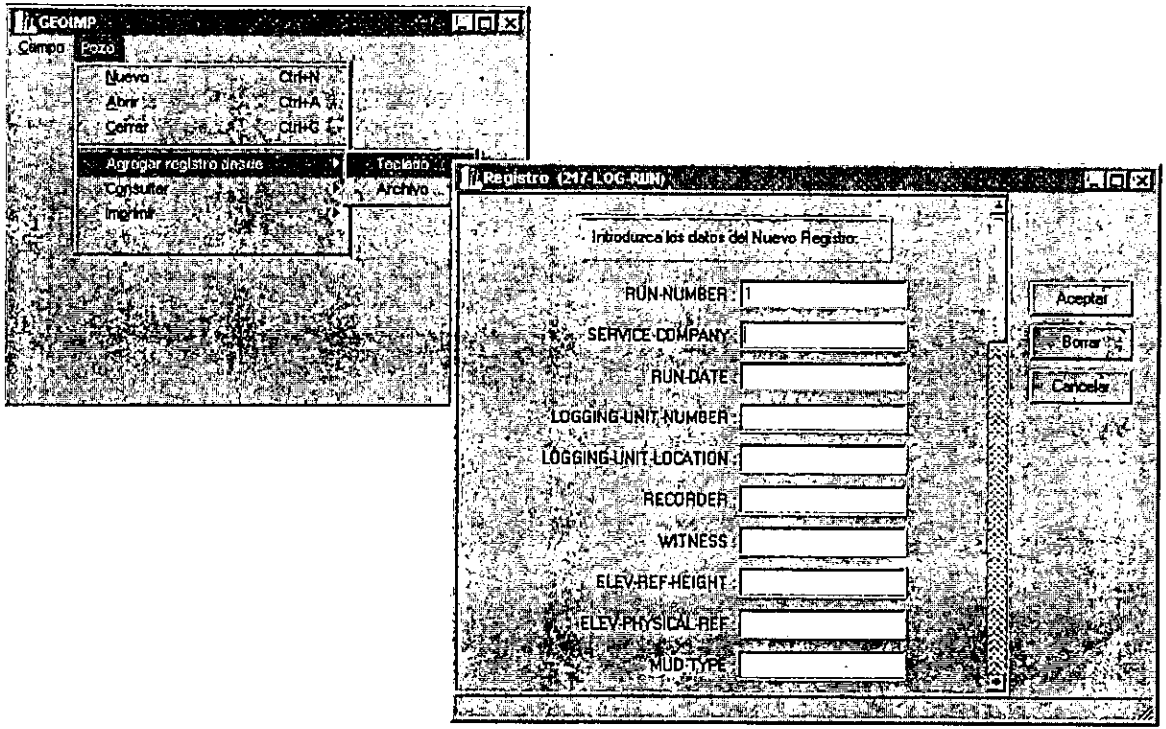
Al escoger la opción Abrir, se muestra una ventana que pide el ID (identificador), del Pozo.
En caso de que éste no exista en el campo actual se indica.

DIAGRAMA DE INTERFAZ




**D19:**
**CERRAR POZO**
Al seleccionar la opción Cerrar, se presenta una ventana que pregunta al usuario si realmente desea abandonar el pozo.

DIAGRAMA DE INTERFAZ

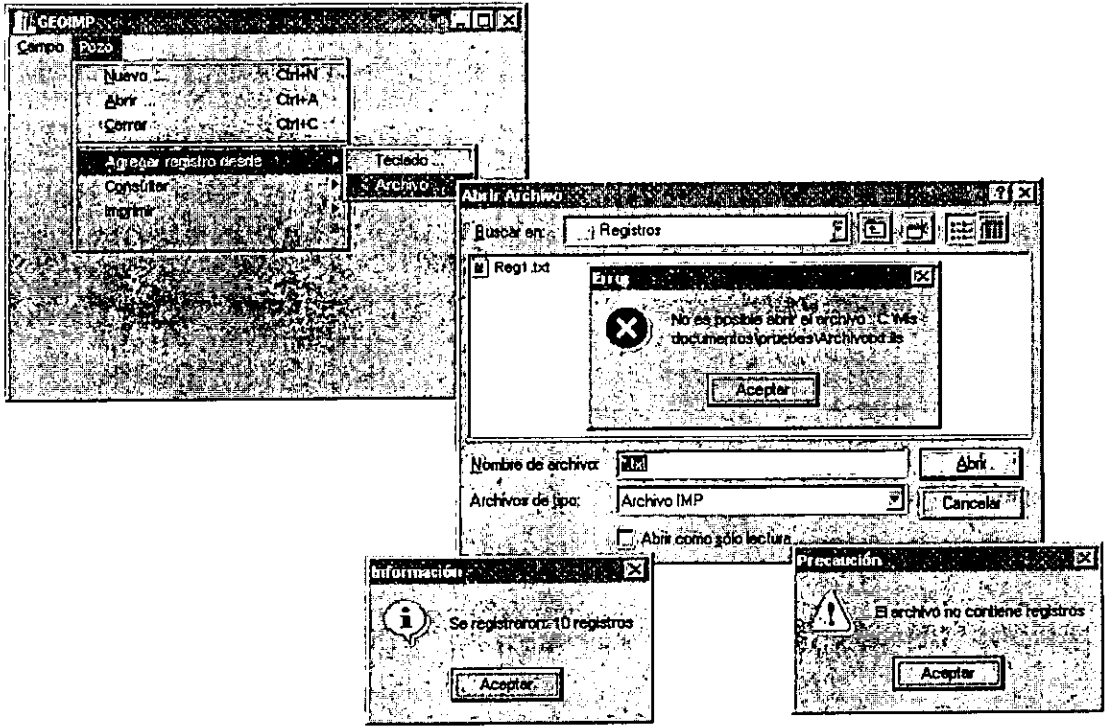



**DI10:**

**AGREGA REGISTRO (TECLADO)**

Muestra una ventana que solicita información del registro perteneciente al pozo actual.

DIAGRAMA DE INTERFAZ

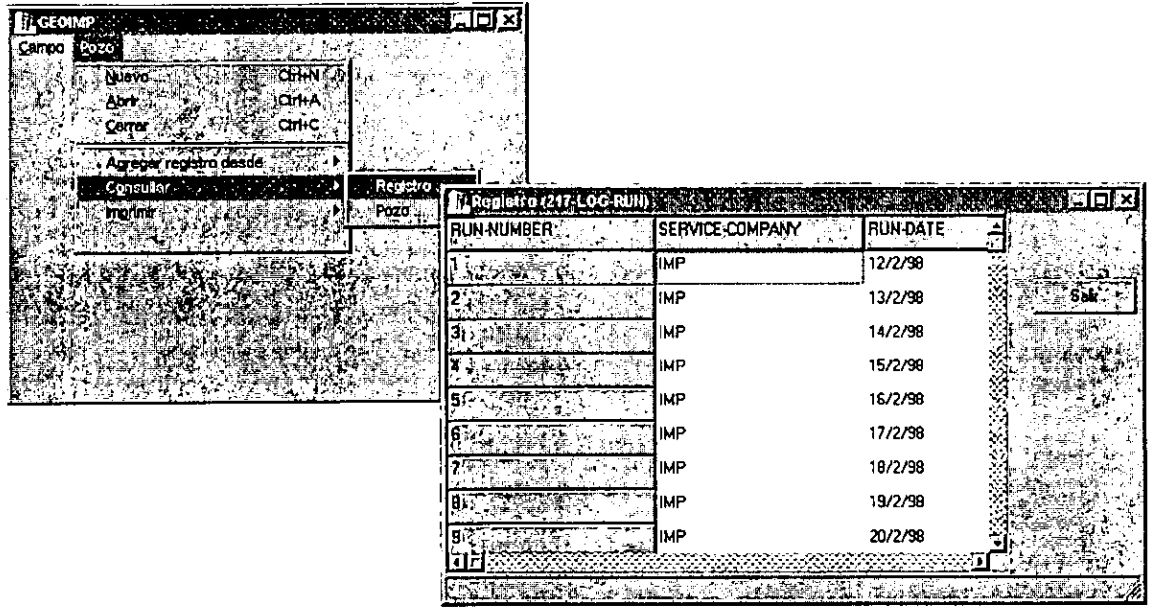



**DI11:**

**AGREGA REGISTRO (ARCHIVO)**

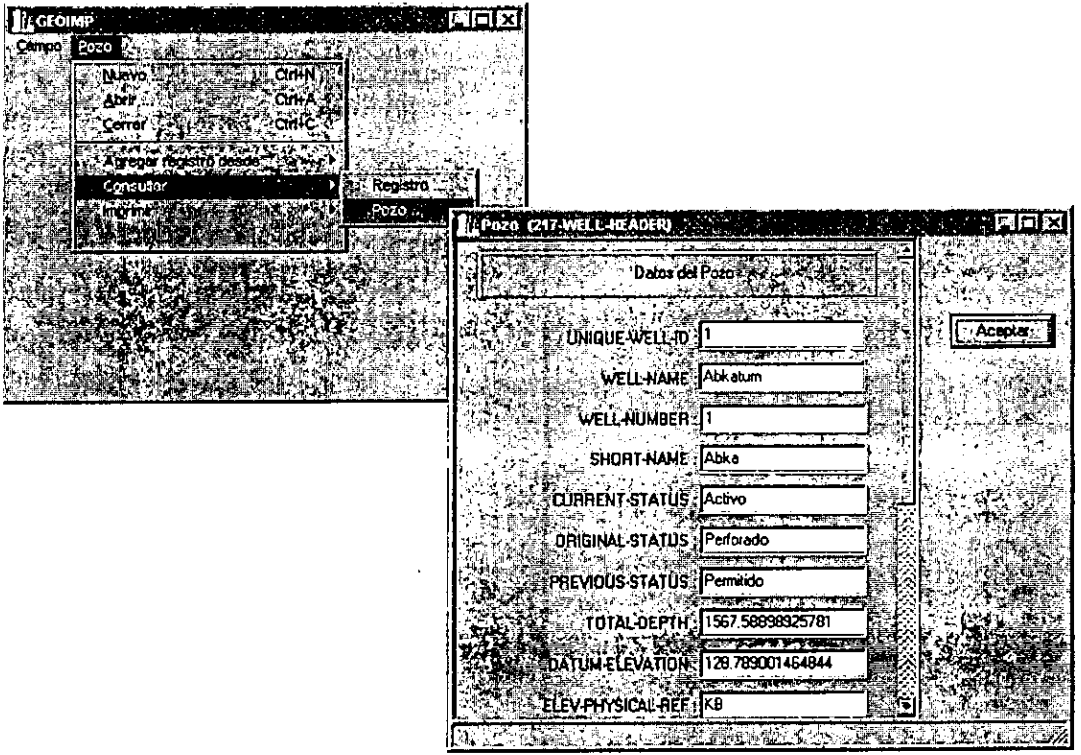
Presenta una ventana de diálogo que solicita el nombre y localización del archivo, al terminar de leerlo se muestra información de cuantos registros se agregaron al pozo actual.

DIAGRAMA DE INTERFAZ



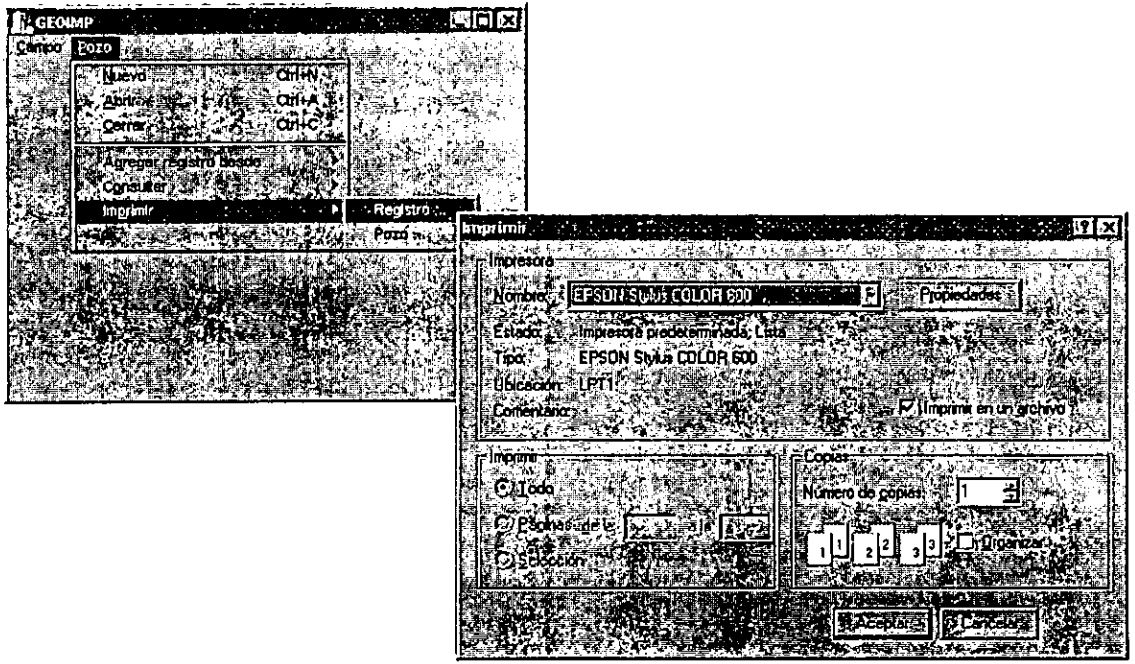

**D112:**
**CONSULTAR REGISTRO**
Muestra información de todos los registros que pertenecen al pozo actual.

DIAGRAMA DE INTERFAZ




**DI13:**
**CONSULTAR POZO**
Muestra información del pozo actual.

DIAGRAMA DE INTERFAZ

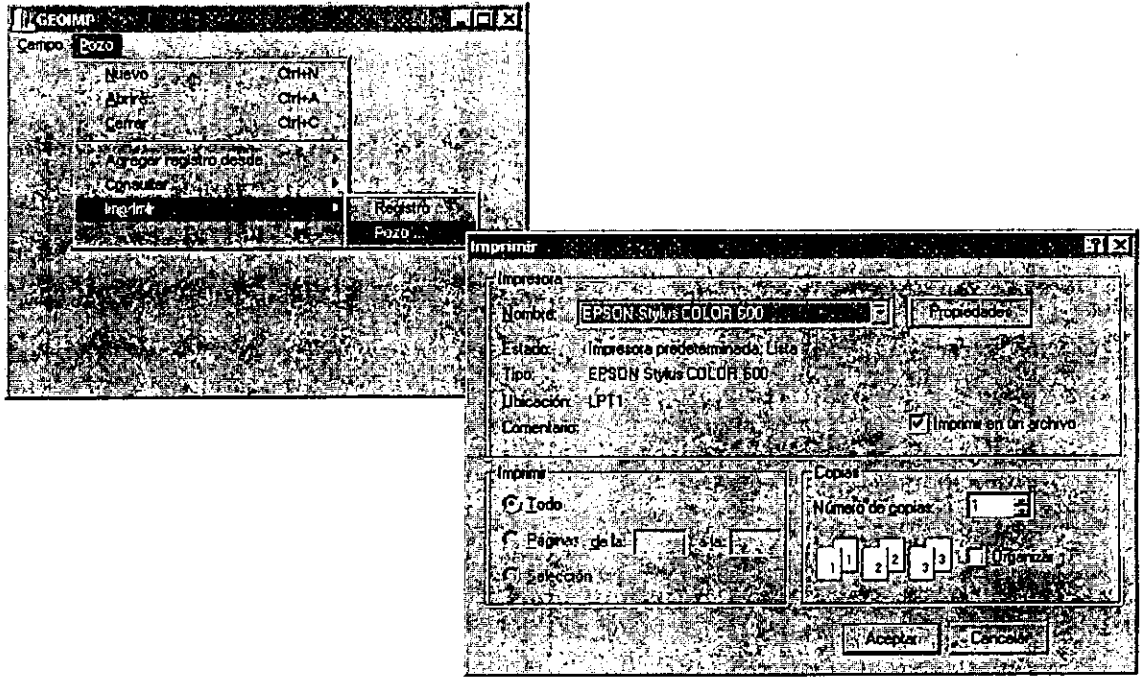



**DI14:**

**IMPRIMIR REGISTRO**

Imprime todos los registros que pertenecen al pozo actual, se puede imprimir a un archivo o a la impresora. Si se imprime a un archivo se pide el nombre de éste.

DIAGRAMA DE INTERFAZ




**D115:**

**IMPRIMIR POZO**

Imprime la información del Pozo actual, teniendo la opción de imprimir a un archivo o a la impresora. Si se imprime a un archivo se pide el nombre de éste

### 5.1.4 ANÁLISIS DE TRAZAS.

El análisis realizado permite comprender la funcionalidad externa esperada (análisis ambiental) y la estructura preliminar (análisis composicional), faltando la funcionalidad interna del sistema, la cual es definida por la relación entre los requerimientos externos (funcionalidad externa) y los componentes del sistema.

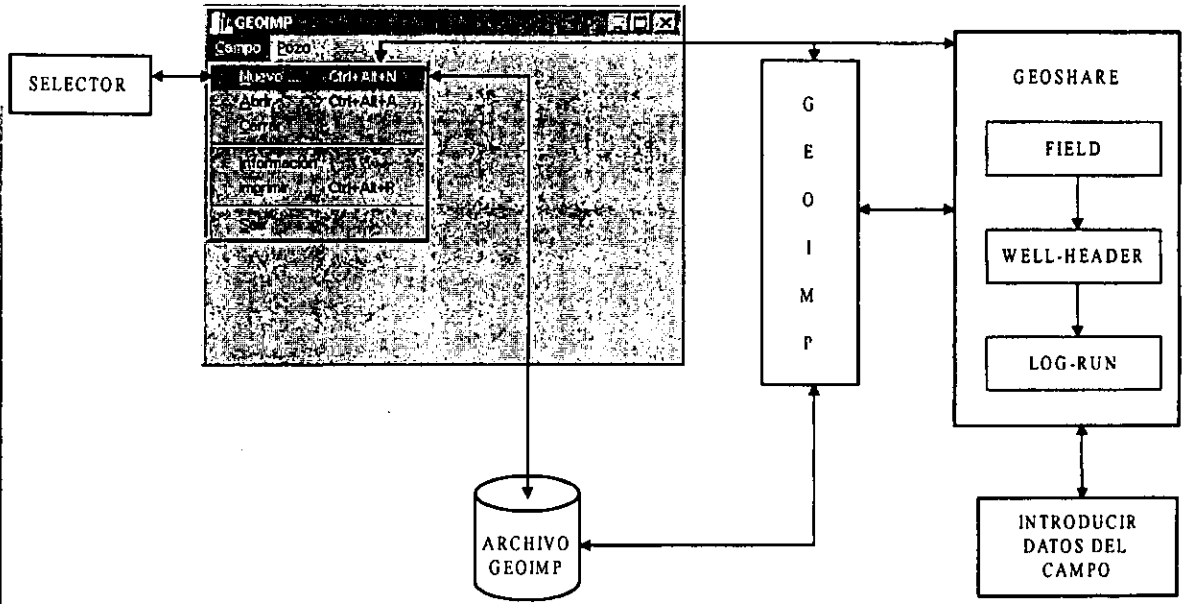
La funcionalidad interna es definida por trazas, las cuales son la liga o relación entre la estructura y los requerimientos del sistema. Por lo tanto una traza es " Un camino específico de uso del sistema para realizar un segmento de funcionalidad, el cual identifica la interacción entre varios elementos para realizar una función específica".

El diagrama EI1 (Menú principal) del análisis ambiental, es la base de identificación de trazas del sistema, los cuales se mencionan en la Figura 5.18.

| ID | NOMBRE | NEMÓNICO |
|---|---|---|
| DT1 | Nuevo Campo | TNC |
| DT2 | Abrir Campo | TAC |
| DT3 | Cerrar Campo | TCC |
| DT4 | Información Campo | TIC |
| DT5 | Imprimir Campo | TPC |
| DT6 | Salir Campo | TSC |
| DT7 | Nuevo Pozo | TNP |
| DT8 | Abrir Pozo | TAP |
| DT9 | Cerrar Pozo | TCP |
| DT10 | Agrega Registro (Teclado) | TART |
| DT11 | Agrega Registro (Archivo) | TARA |
| DT12 | Consultar Registro | TCR |
| DT13 | Consultar Pozo | TCP |
| DT14 | Imprimir Registro | TIR |
| DT15 | Imprimir Pozo | TIP |

**Figura 5.18.- Tabla de diagramas de trazas.**

## DIAGRAMA DE TRAZA

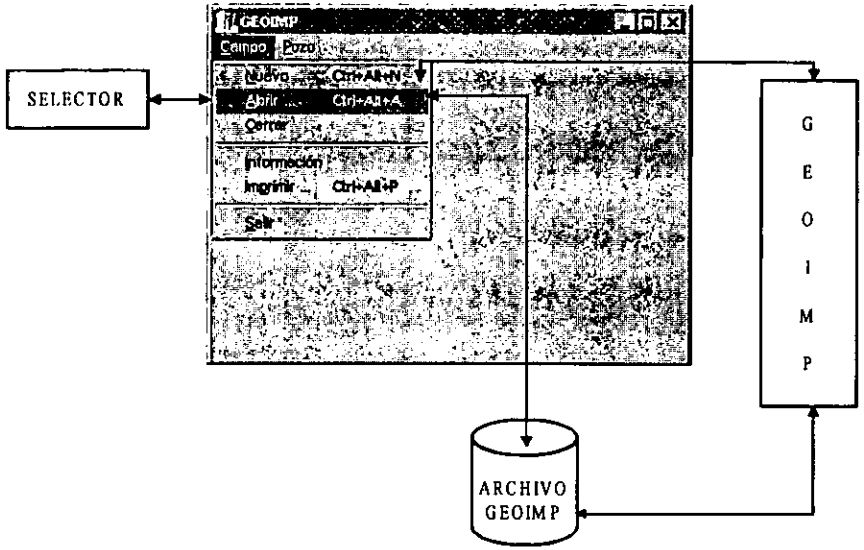



**DT1:**

**NUEVO CAMPO**

1. El usuario selecciona la opción Nuevo del menú Campo.
2. Aparece una ventana de diálogo (Selector), pidiendo el nombre del nuevo archivo (*.imp).
3. Se verifica si no hay otro archivo con el mismo nombre en la unidad de almacenamiento.
4. GEOSHARE solicita al usuario la información del nuevo campo y genera el objeto campo (217-FIELD).
5. GEOIMP recibe el objeto campo; lo transforma a formato RP66 (DLIS) y aplica persistencia.
6. GEOIMP almacena la información en el archivo (*.imp).

DIAGRAMA DE TRAZA

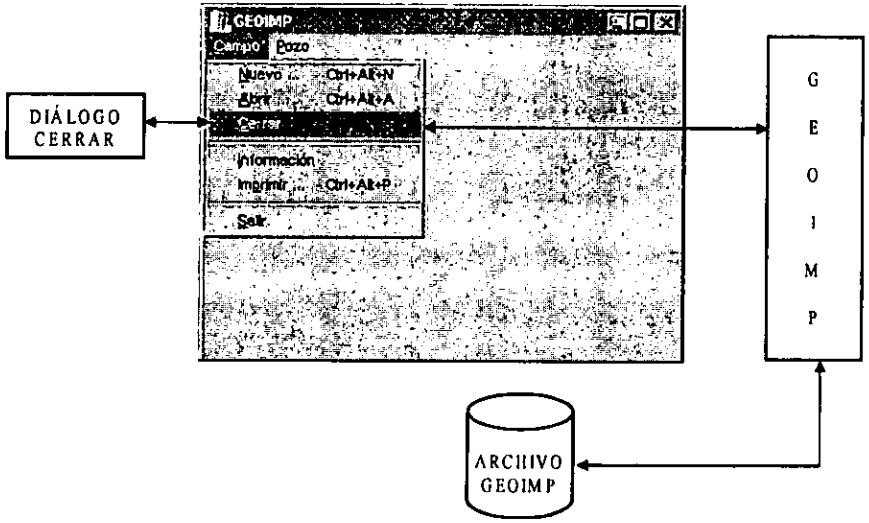



**DT2:**

**ABRIR CAMPO**

1. El usuario selecciona la opción Abrir del menú Campo.
2. Se presenta una ventana de diálogo (Selector), solicitando el nombre del archivo (*.imp).
3. Se verifica si existe el archivo en la unida de almacenamiento.
4. GEOIMP extrae la información principal del archivo.
5. Si existe algún problema durante el proceso se le indica al usuario.

DIAGRAMA DE TRAZA

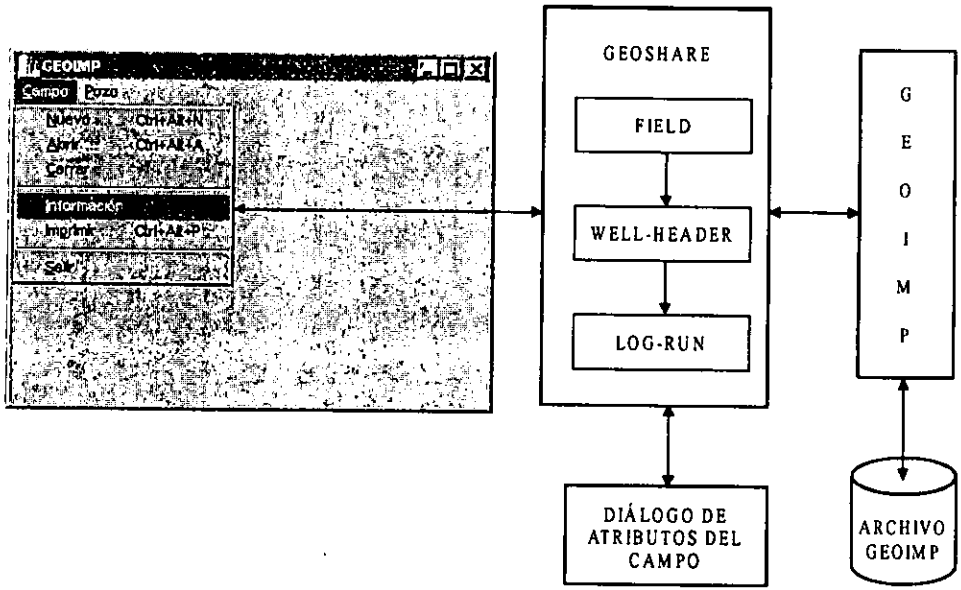



**DT3:**

**CERRAR CAMPO**

1. El usuario selecciona la opción Cerrar del menú Campo.
2. Se muestra una ventana para confirmar.
3. GEOIMP guarda información que se encuentre en memoria, y cierra el archivo (*.imp).
4. Si existe algún problema durante el proceso se le indica al usuario.

DIAGRAMA DE TRAZA

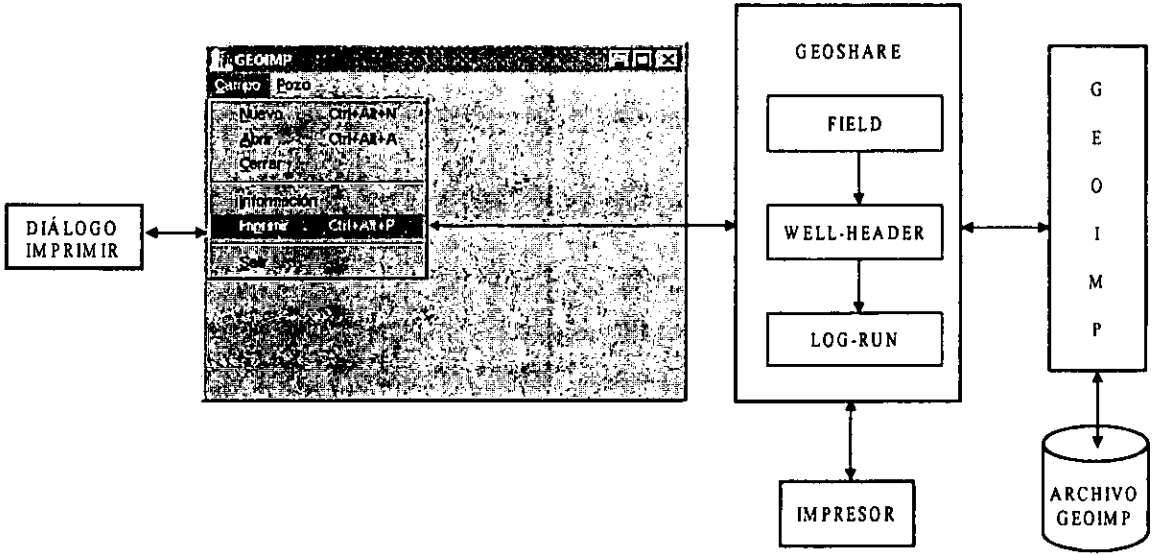



**DT4:**

**INFORMACIÓN CAMPO**

1. El usuario selecciona la opción Información del menú Campo.
2. GEOSHARE llama a GEOIMP.
3. GEOIMP por medio de la persistencia busca el objeto campo en el archivo abierto (*.imp) y lo extrae.
4. GEOSHARE recibe un objeto de tipo Geoshare.
5. La información del campo actual, es presentada al usuario.
6. Si existe algún problema durante el proceso se le indica al usuario.

DIAGRAMA DE TRAZA

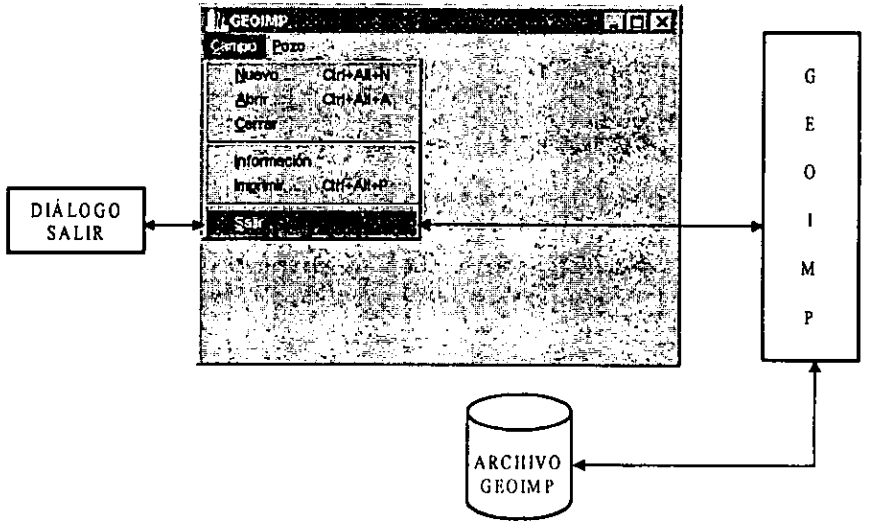



**DT5:**

**IMPRIMIR CAMPO**

1. El usuario elige la opción Imprimir del menú Campo.
2. Se presenta un diálogo de impresión con opciones de imprimir a un archivo o a una impresora.
3. GEOSHARE llama a GEOIMP.
4. GEOIMP por medio de la persistencia busca el objeto campo en el archivo abierto (*.imp) y lo extrae.
5. GEOSHARE recibe un objeto de tipo Geoshare.
6. Se imprime.

DIAGRAMA DE TRAZA

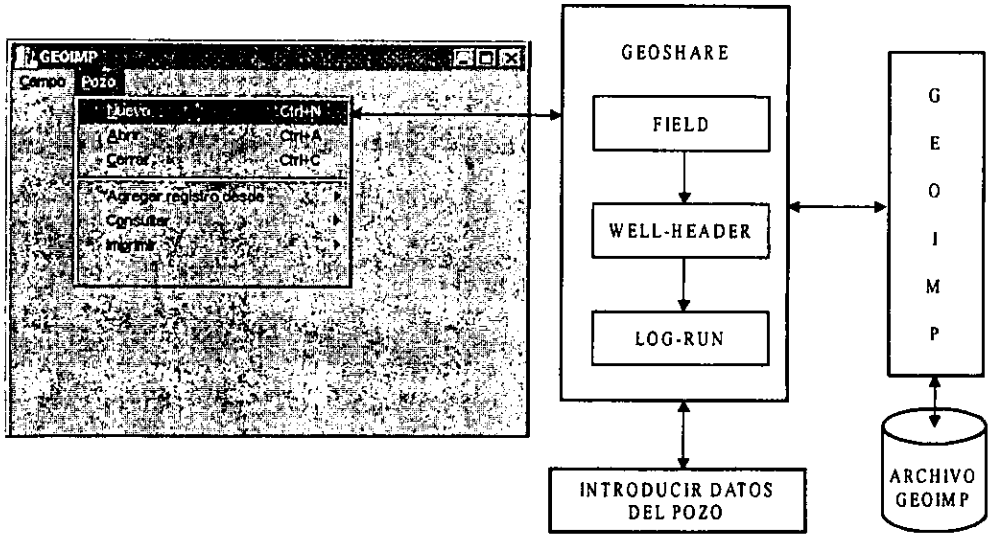



**DT6:**

**SALIR CAMPO**

1. El usuario selecciona la opción Salir del menú Campo.
2. Se presenta una ventana para confirmar.
3. GEOIMP verifica si el archivo ya fue cerrado, en caso de que no, guarda la información que se encuentra en RAM y cierra el archivo.
4. Se cierra el sistema.
5. Si existe algún problema durante el proceso se le indica al usuario.

DIAGRAMA DE TRAZA

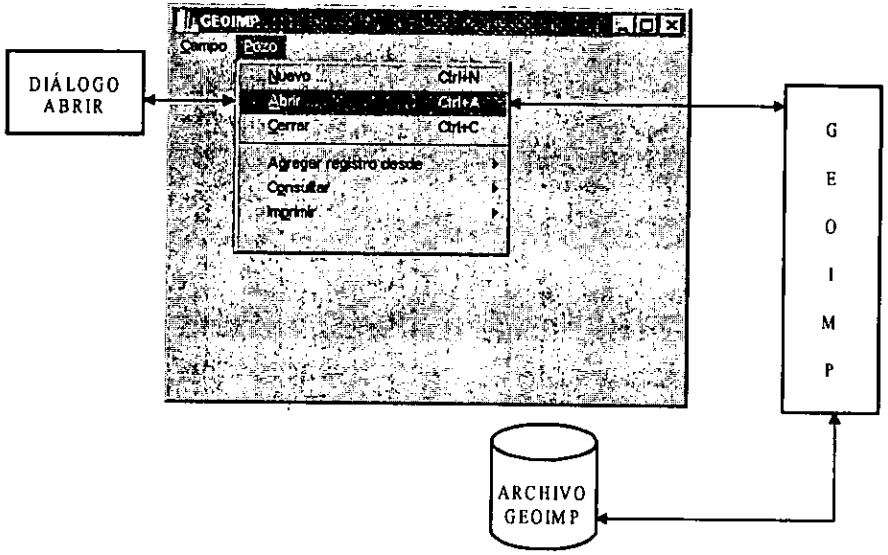



**DT7:**

**NUEVO POZO**

1. El usuario elige la opción Nuevo del menú Pozo.
2. Recibe la petición GEOSHARE, muestra una ventana solicitando al usuario que introduzca los datos del pozo y genera el objeto pozo (217-WELL-HEADER).
3. GEOIMP recibe el objeto pozo lo transforma a formato RP66 (DLIS) y le aplica persistencia.
4. Finalmente GEOIMP almacena la información del pozo en el archivo (*.imp).
5. Si existe algún problema durante el proceso se le indica al usuario.

DIAGRAMA DE TRAZA

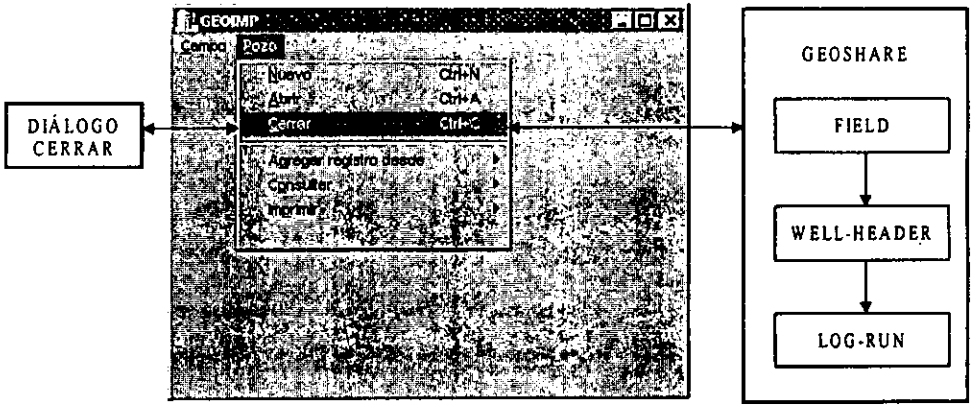



**DT8:**

**ABRIR POZO**

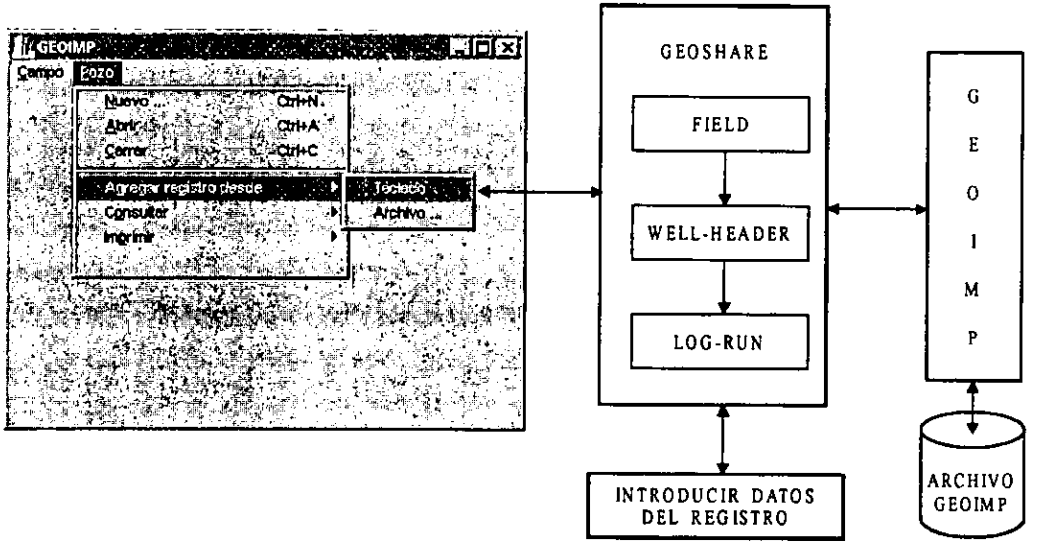
1. El usuario selecciona la opción Abrir del menú Pozo.
2. Se pide al usuario que introduzca del ID(Identificador único) del pozo .
3. GEOIMP recibe el ID y busca si existe el pozo en el Campo actual.
4. Si existe algún problema durante el proceso se le indica al usuario.

DIAGRAMA DE TRAZA




**DT9:**
**CERRAR POZO**
1. El usuario selecciona la opción Cerrar del menú Pozo.
2. Se muestra una ventana para confirmar.
3. GEOSHARE cierra el pozo actual.
4. Si existe algún problema durante el proceso se le indica al usuario.

DIAGRAMA DE TRAZA

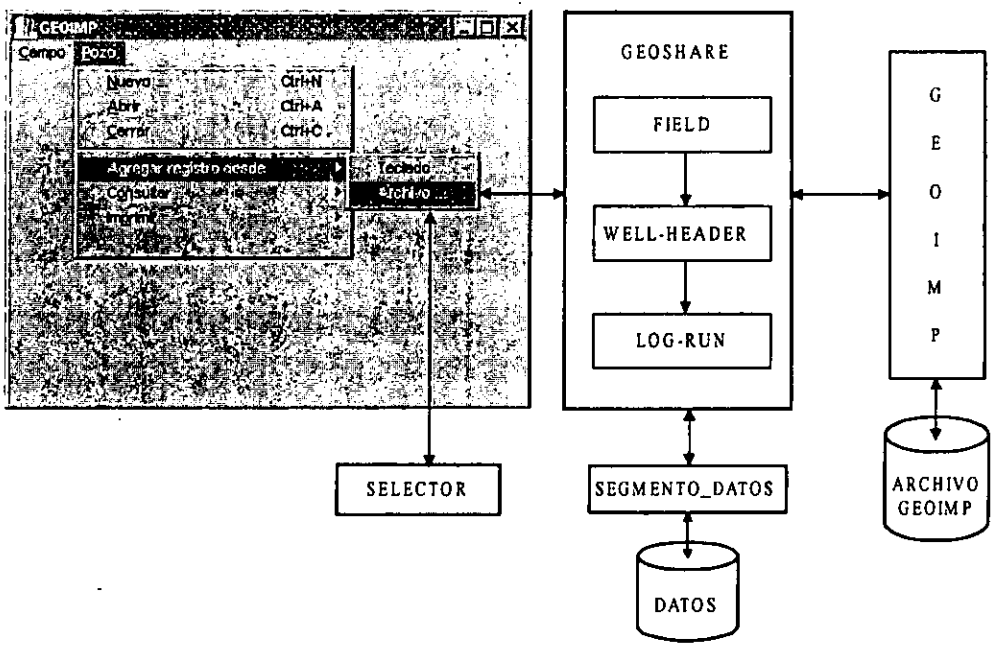



**DT10:**

**AGREGA REGISTRO (TECLADO)**

1. El usuario elige la opción Agregar registro desde... Teclado de menú Pozo.
2. GEOSHARE solicita al usuario que introduzca los datos del registro.
3. GEOSHARE genera el objeto registro (217-LOG-RUN).
4. GEOIMP recibe el objeto registro y lo transforma a formato RP66 (DLIS) y aplica persistencia.
5. Finalmente GEOIMP almacena la información del pozo en el archivo (*.imp).
6. Si existe algún problema durante el proceso se le indica al usuario.

## DIAGRAMA DE TRAZA

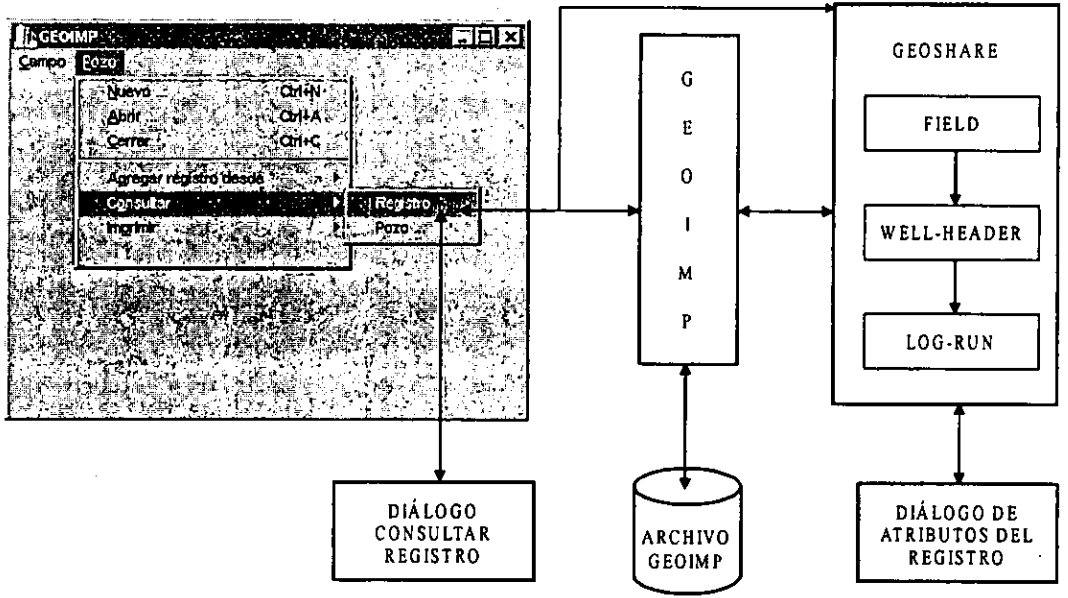



**DT11:**

**AGREGA REGISTRO (ARCHIVO)**

1. El usuario elige la opción Agregar registro desde... Archivo de menú Pozo.
2. Se presenta una ventana de diálogo que solicita al usuario el nombre del archivo.
3. GEOSHARE solicita al SEGMENTO_DATOS que lea la información del archivo y por cada registro genera un objeto registro (217-LOG-RUN).
4. GEOIMP recibe cada objeto registro y lo transforma a formato RP66 (DLIS) y aplica persistencia.
5. Finalmente GEOIMP almacena la información del pozo en el archivo (*.imp).

DIAGRAMA DE TRAZA

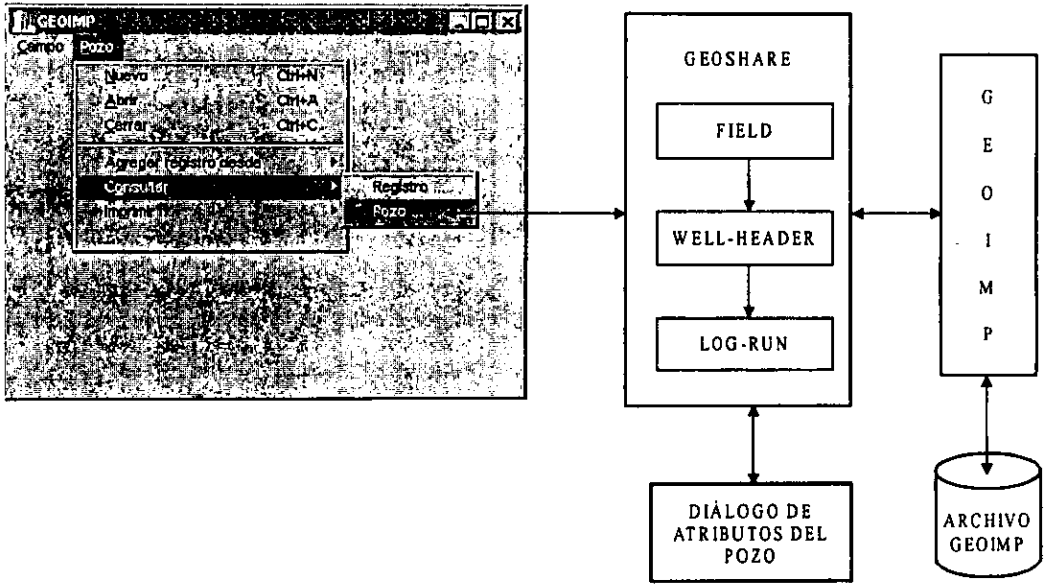



**DT12:**

**CONSULTAR REGISTRO**

1. El usuario elige la opción Consultar ... Registro del menú Pozo.
2. GEOSHARE llama a GEOIMP.
3. GEOIMP por medio de la persistencia busca todos los objetos registros del pozo actual en el archivo abierto (*.imp) y los extrae.
4. GEOSHARE recibe los objetos tipo Geoshare.
5. La información de los registros se muestra al usuario.

DIAGRAMA DE TRAZA

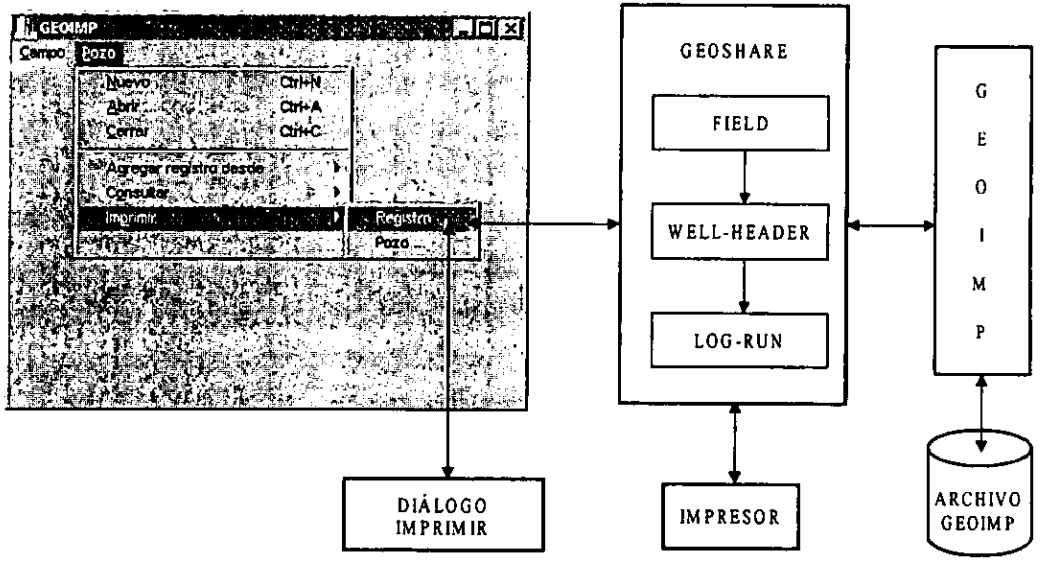



**DT13:**

**CONSULTAR POZO**

1. El usuario elige la opción Consultar ... Pozo del menú Pozo.
2. GEOSHARE llama a GEOIMP.
3. GEOIMP por medio de la persistencia busca el objeto pozo actual en el archivo abierto (*.imp) y lo extrae.
4. GEOSHARE recibe un objeto de tipo Geoshare.
5. La información del pozo actual se muestra al usuario.
6. Si existe algún problema durante el proceso se le indica al usuario.

DIAGRAMA DE TRAZA

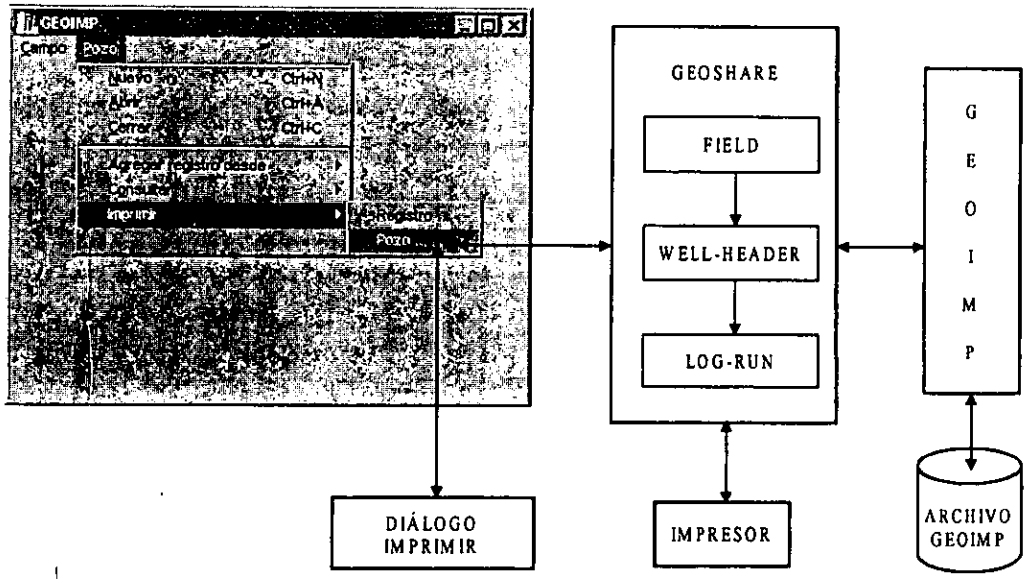



**DT14:**

**IMPRIMIR REGISTRO**

1. El usuario elige la opción Imprimir... Registro del menú Pozo.
2. Se muestra una caja de diálogo de impresión con opciones de imprimir a un archivo o a una impresora.
3. GEOSHARE llama a GEOIMP.
4. GEOIMP por medio de la persistencia busca todos los objetos registros del pozo actual en el archivo abierto (*.imp) y los extrae.
5. GEOSHARE recibe los objetos tipo Geoshare.
6. Se imprimen.

DIAGRAMA DE TRAZA




**DT15:**

**IMPRIMIR POZO**

1. El usuario elige la opción Imprimir... Pozo del menú Pozo.
2. Se muestra una caja de diálogo de impresión con opciones de imprimir a un archivo o a una impresora.
3. GEOSHARE llama a GEOIMP.
4. GEOIMP por medio de la persistencia busca el objeto pozo actual en el archivo abierto (*.imp) y lo extrae.
5. GEOSHARE recibe un objeto tipo Geoshare.
6. Se imprime.

## 5.2 DISEÑO DEL SISTEMA.

El diseño tiene como objetivo proporcionar la arquitectura de una estructura que soporte los requerimientos del análisis y su funcionalidad.

El modelo de diseño del sistema es comúnmente representado por diagramas de clase y objetos, arquitectura y los mecanismos necesarios para manejo de errores, administración de memoria, almacenamiento de datos, etc.

Finalmente el análisis y el diseño proveen un modelo o prototipo básico de desarrollo del sistema, el cual debe ser validado en el dominio de la aplicación.

Esta fase es muy parecida a la obtención del plano del diseño (diagrama) de un componente electrónico, donde cada línea es definida y se conocen los tiempos de conmutación de reloj, antes de ensamblar los elementos en la tarjeta de circuito impreso que corresponden a la fase de implementación.

El modelo de diseño requiere:

- Obtención del modelo inicial del sistema.
- Identificar el ambiente de implantación.
- Describir la interacción entre los componentes del modelo.
- Realizar el diseño detallado de cada uno de los componentes.

### 5.2.1 MODELO INICIAL DEL SISTEMA.

En el modelo inicial del sistema se establecen las condiciones y características generales de diseño del sistema, cuales son el lineamientos para el diseño del sistema, además se describe un modelo básico con las relaciones que se perciben de los componentes, el cual será perfeccionado a lo largo de esta fase.

La Figura 5.19 muestra el modelo inicial del sistema para implementar persistencia, en base a los resultados obtenidos en la fase de análisis.

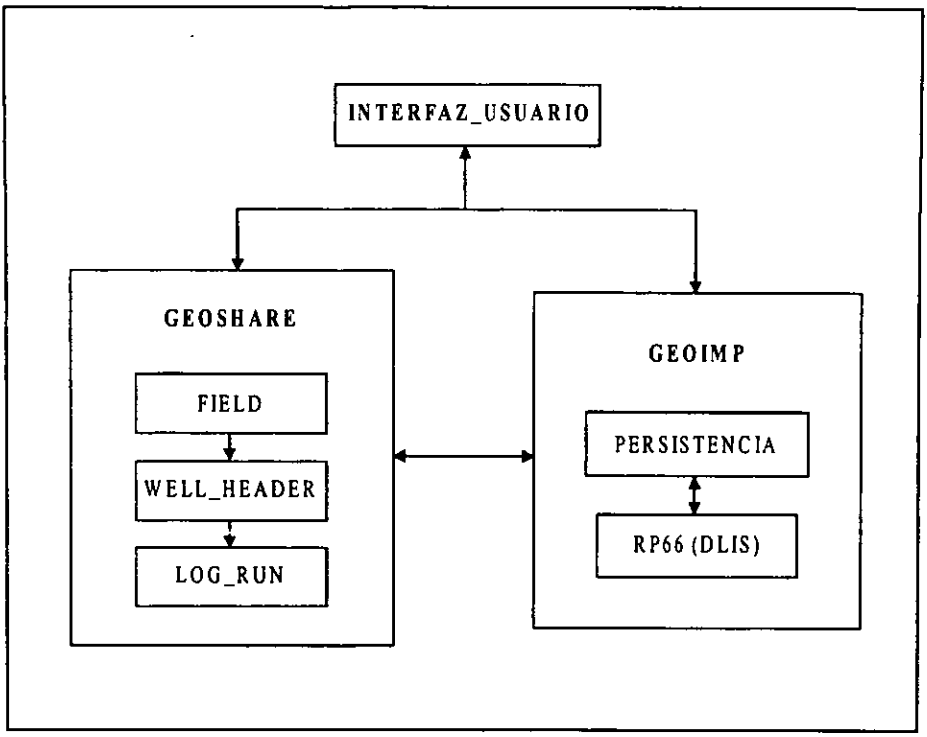



**Figura 5.19.- Modelo inicial de diseño.**

### 5.2.1.1 INTERFAZ_USUARIO.

Este componente se encarga de administrar y controlar todos los eventos realizados por el usuario, ya que permite realizar el conjunto de operaciones definidas por el sistema, así como interactúa con los componentes del sistema. Se encuentra representado por el elemento de interfaz EI1.

### 5.2.1.2 GEOSHARE.

Tiene como tarea principal encapsular los datos de entrada en objetos de tipo Geoshare, así como puede crear, mostrar e imprimir los valores de un objeto tipo Geoshare. Los datos de entrada pueden ser dados desde teclado o un archivo *.txt. En la parte de impresión permite imprimir hacia un archivo o bien hacia la impresora. Para el sistema sólo se consideran tres tipos de objetos del formato Geoshare (217-Field , 217-Well-Header y 217-Log-Run).

GEOIMP.

El propósito de éste es dotar de persistencia a los objetos tipo Geoshare, así como el guardar y recuperar objetos siguiendo el formato RP66 (DLIS), este elemento trabaja principalmente con el elemento GEOSHARE, de donde recibe la petición de almacenar o restaurar un objeto tipo Geoshare hacia o desde un archivo híbrido denominado ARCHIVO_GEOIMP (*.imp), además hay una relación con la interfaz de usuario, para validar el archivo tipo *.imp.

## 5.2.2 DISEÑO FUNCIONAL.

Tiene como objetivo fundamental especificar la interacción entre los elementos o componentes del sistema, la cual es realizada a través de diagramas de interacción.

### 5.2.2.1 Diagramas de Interacción.

El diagrama de interacción describe cómo se comunicas cada uno de los elementos de la traza definida durante el análisis. Esto es similar al diseño de un componente electrónico, cuando se definen cada una de las señales entre diferentes elementos que lo constituyen y la lógica de conmutación de éstas.

El diagrama muestra como los objetos de una traza interactúan debido al envío de estímulos de uno a otro, además tienen como objetivo principal definir el protocolo de comunicación entre los objetos, la notación empleada por éstos se muestra en la Figura 5.20.

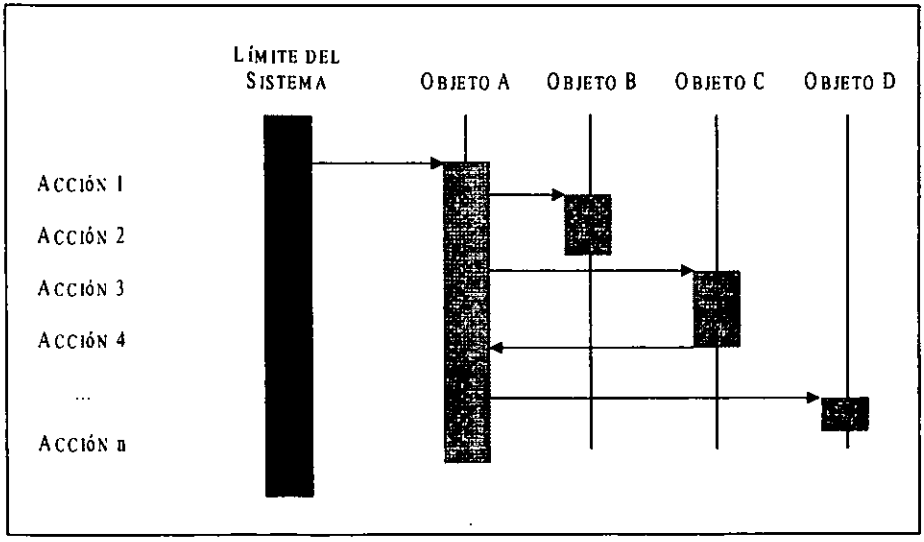



**Figura 5.20.- Diagrama de interacción.**

Cada objeto participante se representa con una barra, éstas son dibujadas como líneas verticales en el diagrama, el límite del sistema es representado por otra barra más gruesa, e identifica la interfase del sistema. En el lado izquierdo del límite del sistema se describe la

secuencia de eventos de la traza que se está diseñando, la cual puede utilizar seudocódigo o texto estructurado, éste describe que está pasando en esta parte de la traza con el objeto que está usando y se denomina operación. Las operaciones establecen servicios o funciones que los objetos realizan para la satisfacción del evento al cual pertenece la traza. La sincronización de operaciones se realiza a través del envío de estímulos o la petición de operación de un objeto a otro.

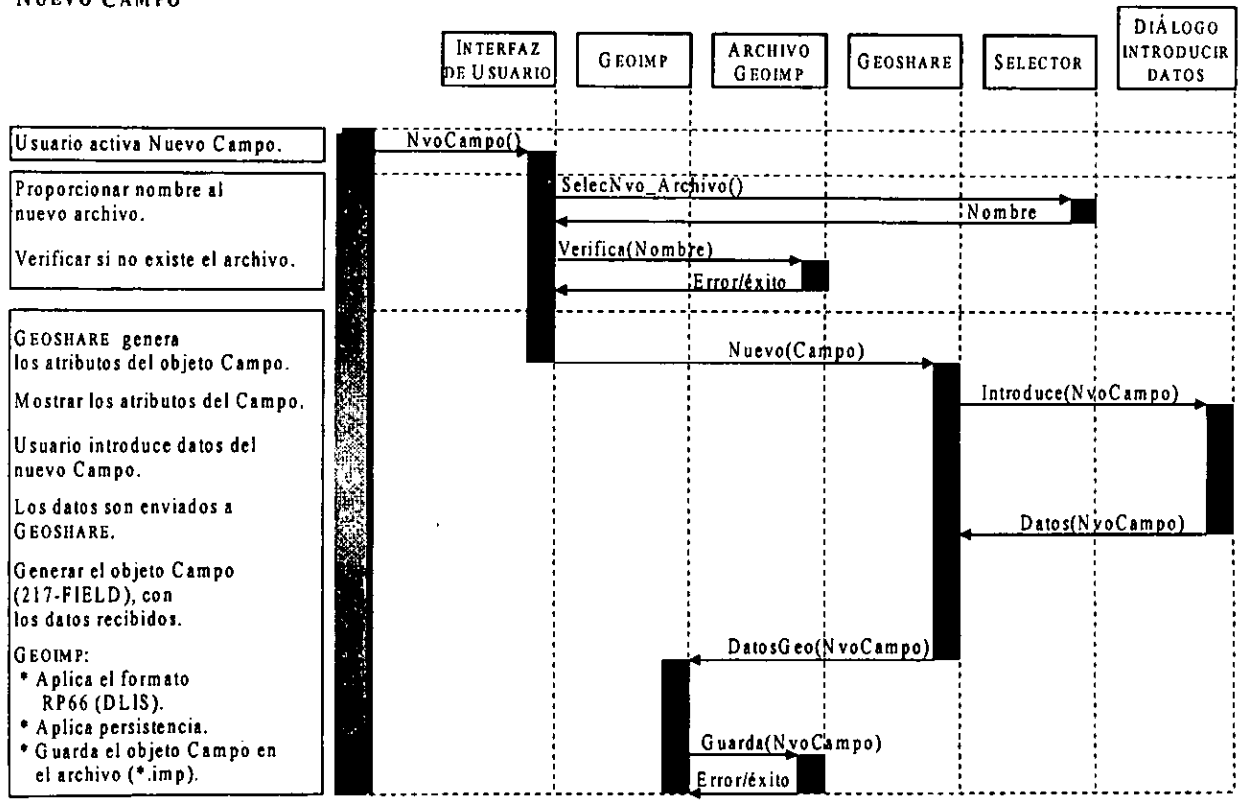
La Figura 5.21 muestra la relación de diagramas de interacción desarrollados para el sistema del objeto persistente, donde la numeración muestra la relación de éstos con los diagramas de traza.

| ID | NOMBRE |
|---|---|
| DINT1 | Nuevo Campo |
| DINT2 | Abrir Campo |
| DINT3 | Cerrar Campo |
| DINT4 | Información Campo |
| DINT5 | Imprimir Campo |
| DINT6 | Salir Campo |
| DINT7 | Nuevo Pozo |
| DINT8 | Abrir Pozo |
| DINT9 | Cerrar Pozo |
| DINT10 | Agrega Registro desde Teclado |
| DINT11 | Agrega Registro desde Archivo |
| DINT12 | Consultar Registro |
| DINT13 | Consultar Pozo |
| DINT14 | Imprimir Registro |
| DINT15 | Imprimir Pozo |

**Figura 5.21.- Tabla de diagramas de interacción.**

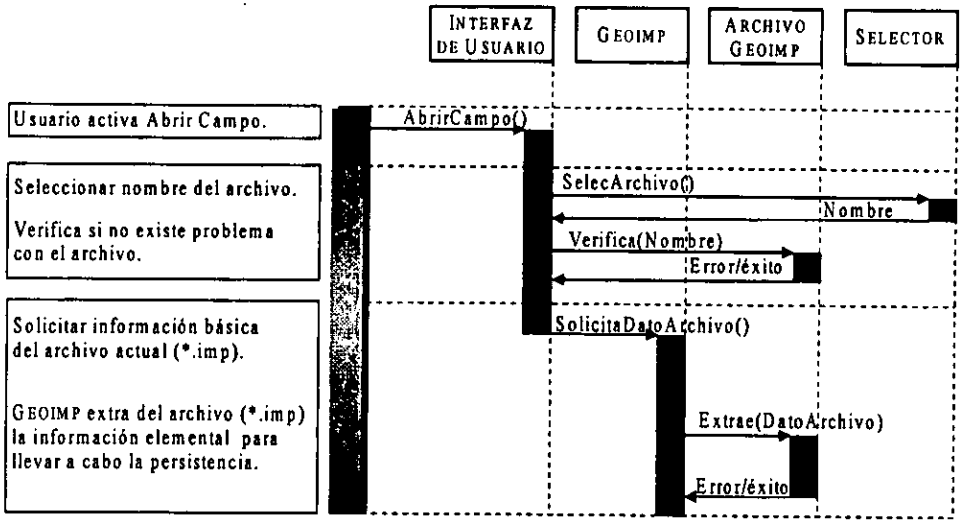
**DINT1:**
**NUEVO CAMPO**

**DIAGRAMA DE INTERACCIÓN**



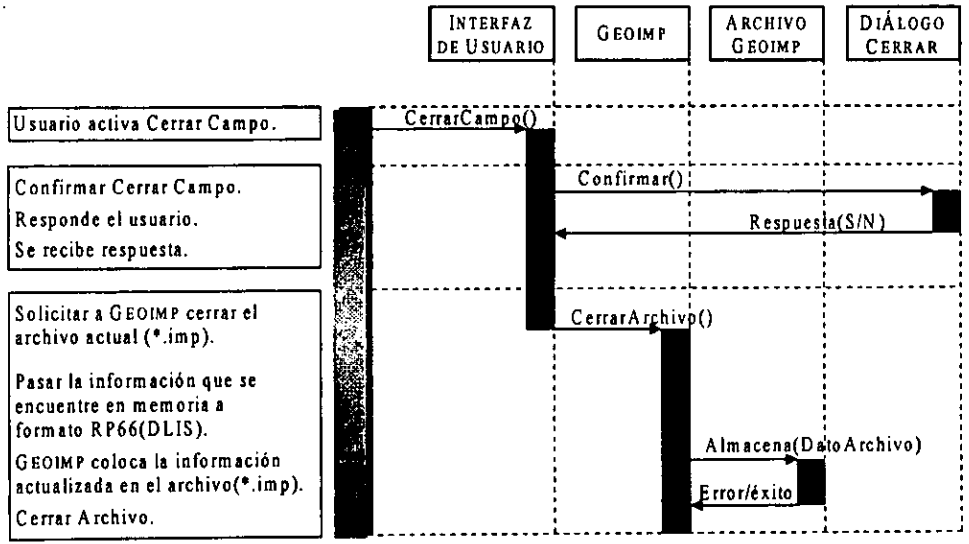
**DINT2:**
**ABRIR CAMPO**

DIAGRAMA DE INTERACCIÓN



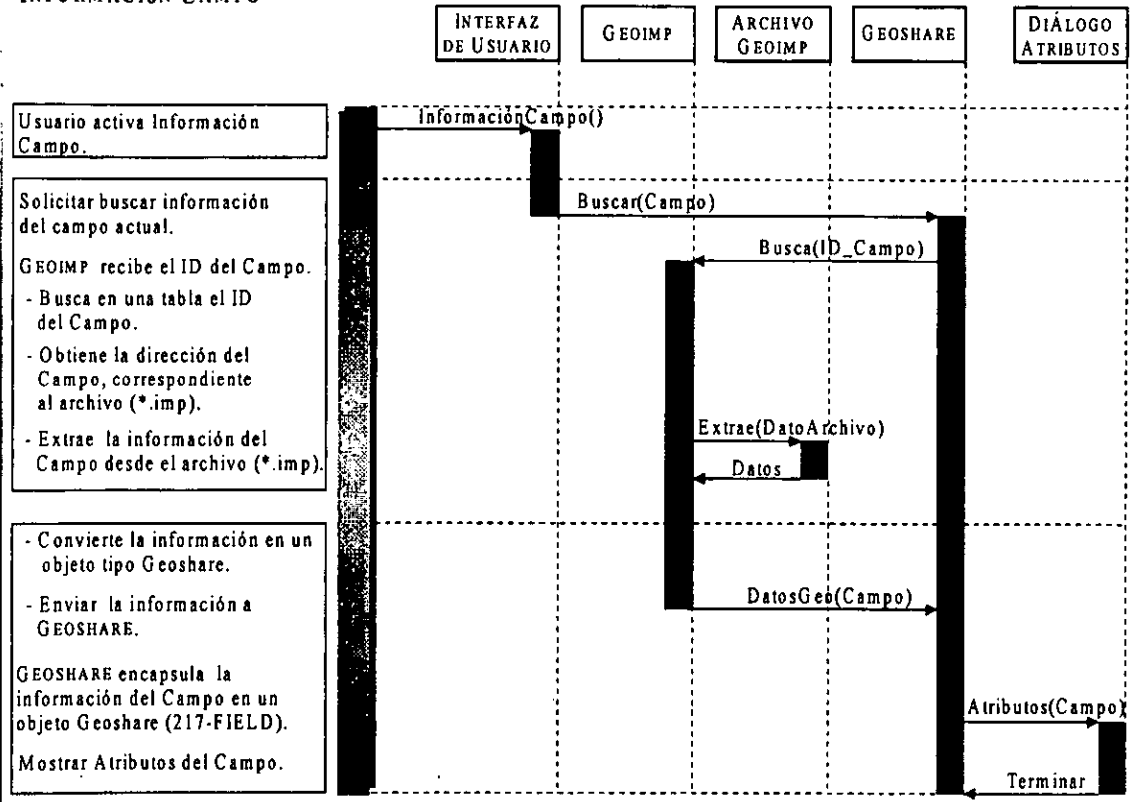
**DINT3:**
**CERRAR CAMPO**





**DINT4:**
**INFORMACIÓN CAMPO**

**DIAGRAMA DE INTERACCIÓN**



**DINT5:**
**IMPRIMIR CAMPO**

DIAGRAMA DE INTERACCIÓN

Usuario activa Imprimir Campo.
Mostrar opciones de impresión.

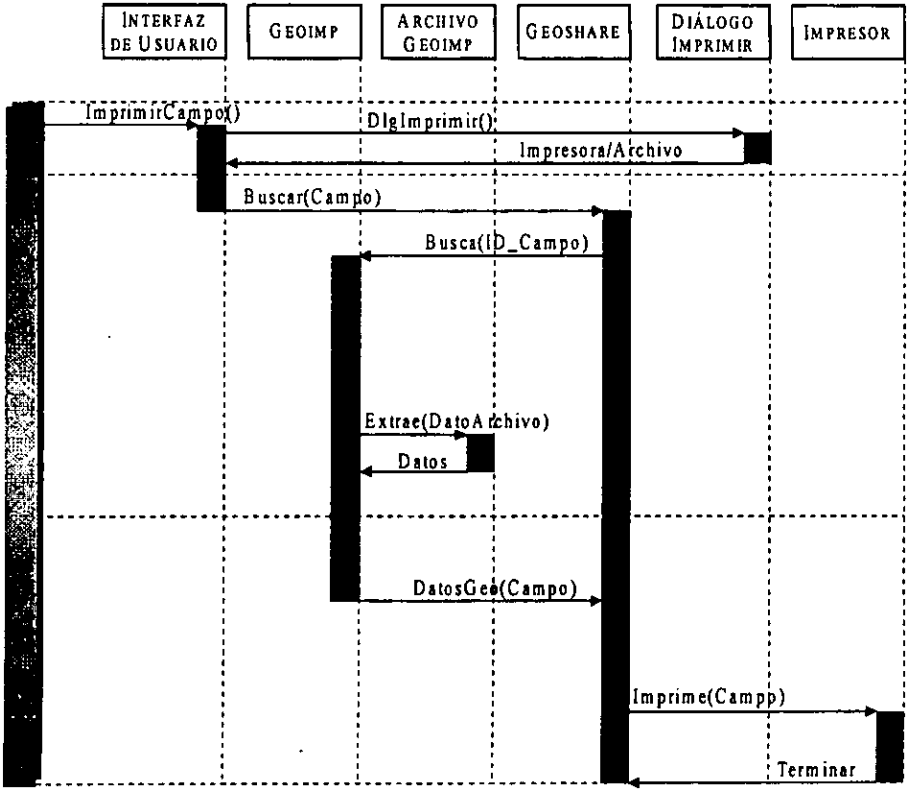
Solicitar buscar información del campo actual.

GEOIMP recibe el ID del Campo.
- Busca en una tabla el ID del Campo.
- Obtiene la dirección del Campo, correspondiente al archivo (*.imp).
- Extrae la información del Campo desde el archivo (*.imp).

- Convierte la información en un objeto tipo Geoshare.
- Enviar la información a GEOSHARE.

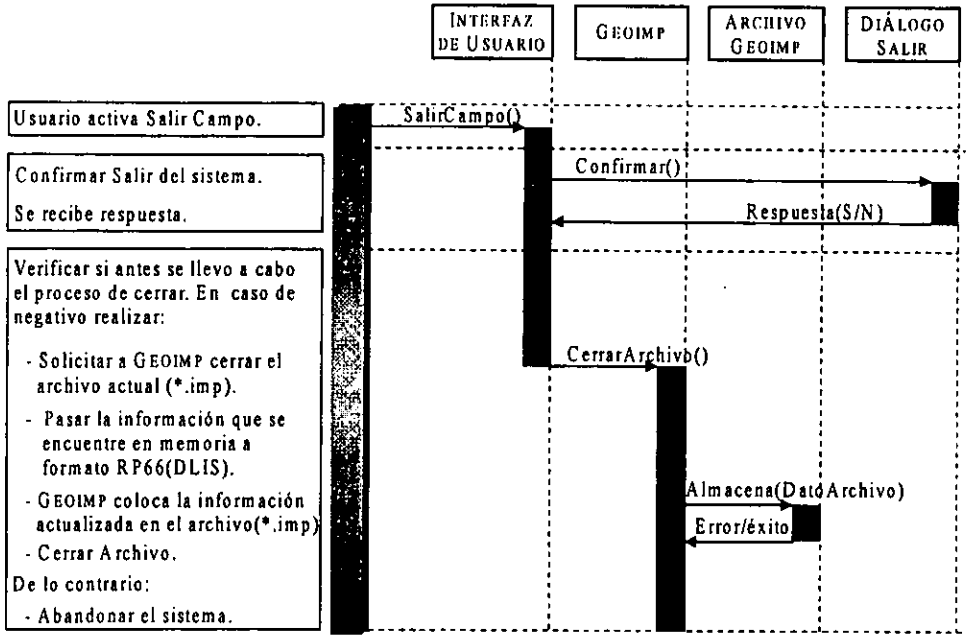
GEOSHARE encapsula la información del Campo en un objeto Geoshare (217-FIELD).

Enviar a imprimir(Campo).



**DINT6:**
**SALIR CAMPO**

DIAGRAMA DE INTERACCIÓN



**DINT7:**
**NUEVO POZO**

DIAGRAMA DE INTERACCIÓN

Usuario activa Nuevo Pozo.

GEOSHARE genera los atributos del objeto Pozo.

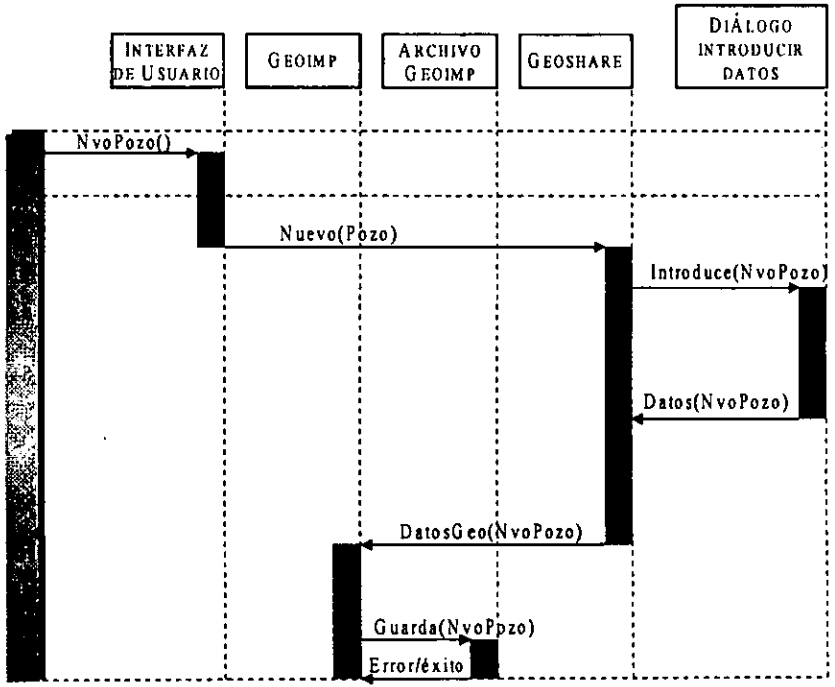
Mostrar los atributos del Pozo.

Usuario introduce datos del nuevo Pozo.

Los datos son enviados a GEOSHARE.

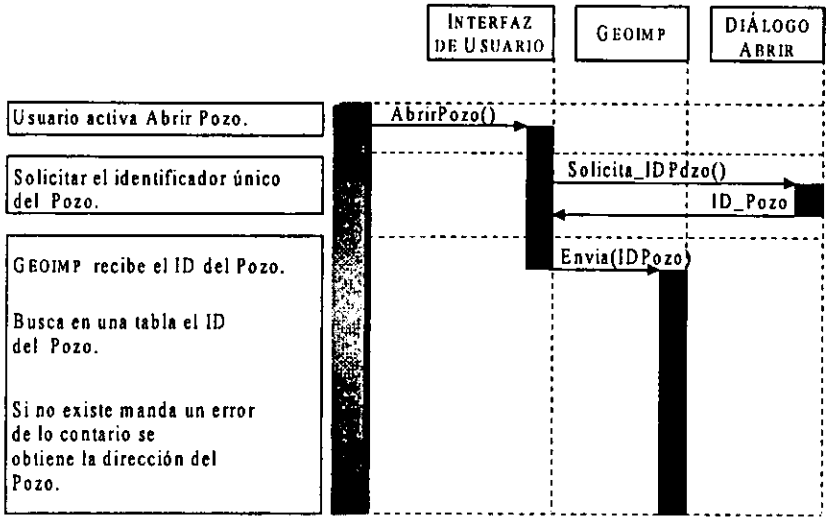
Generar el objeto Pozo (217-WELL-HEADER), con los datos recibidos.

GEOIMP:
* Aplica el formato RP66 (DLIS).
* Aplica persistencia.
* Guarda el objeto Pozo en el archivo (*.imp).



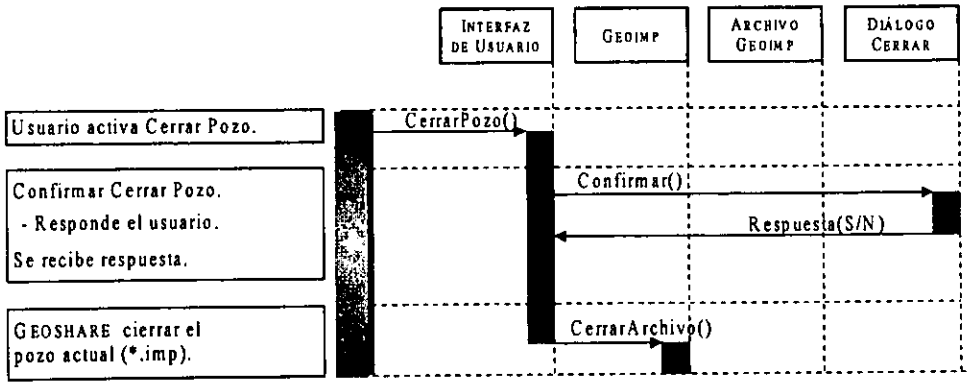
**DINT8:**
**ABRIR POZO**

DIAGRAMA DE INTERACCIÓN

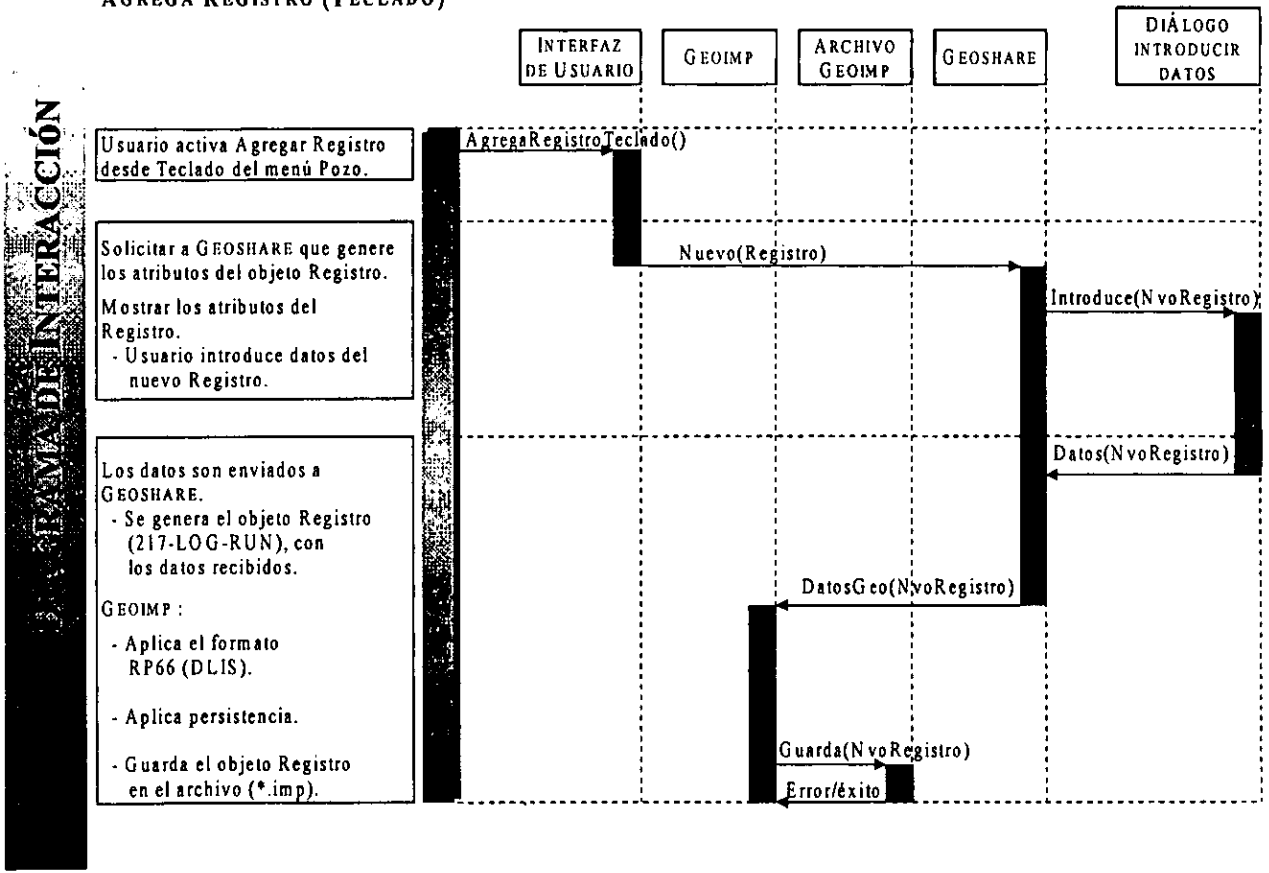


**DINT9:**
**CERRAR POZO**

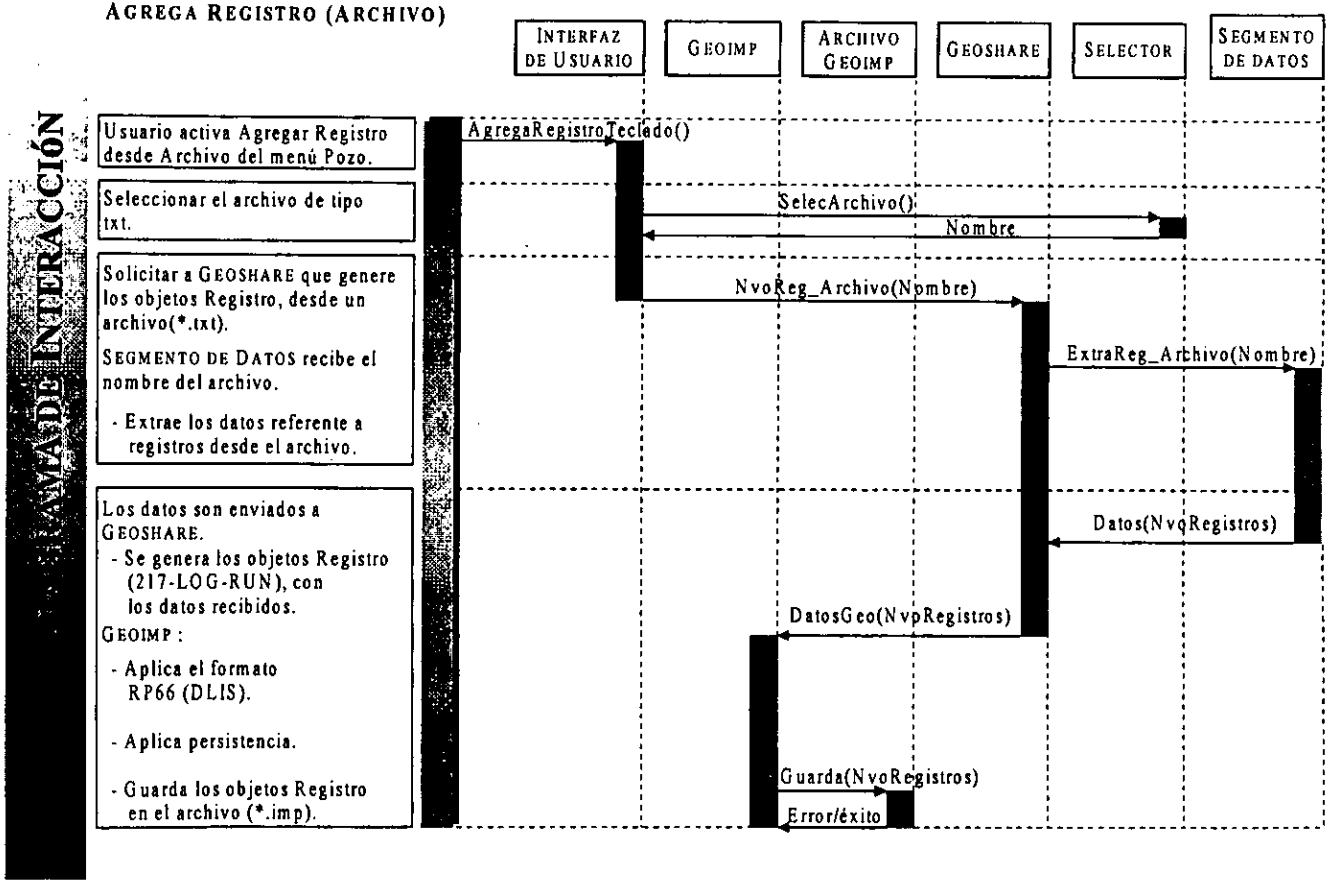
DIAGRAMA DE INTERACCIÓN



**DINT10:**
**AGREGA REGISTRO (TECLADO)**



**DINT11:**
**AGREGA REGISTRO (ARCHIVO)**



**DINT12:**
**CONSULTAR REGISTRO**

DIAGRAMA DE INTERACCIÓN

Usuario activa Consultar Registro.

Solicitar buscar información de todos los registros del pozo actual.

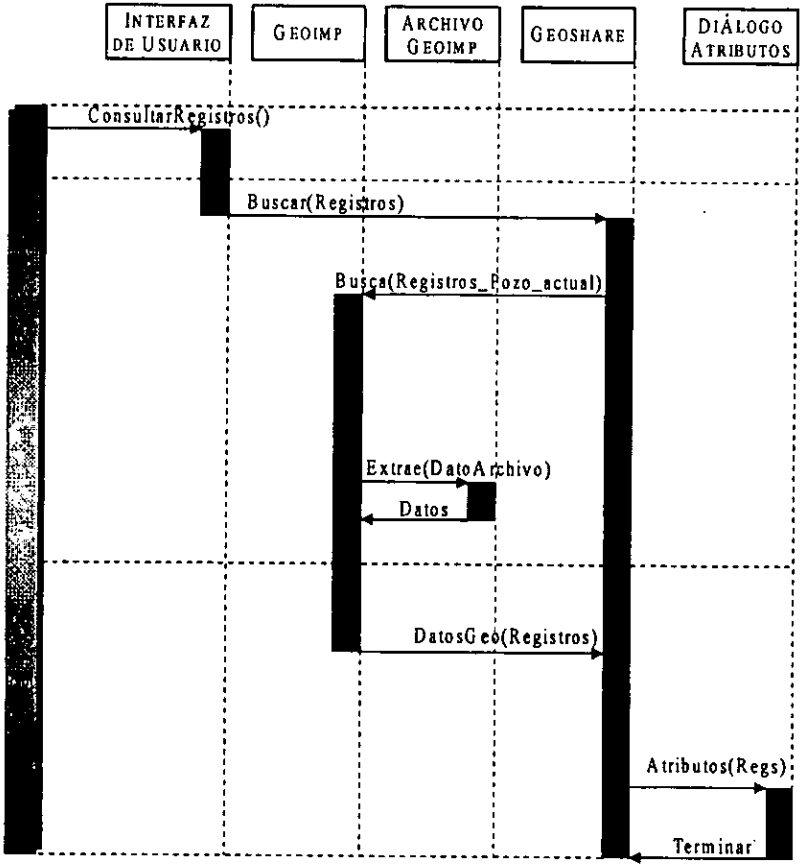
GEOIMP:

- Busca en una tabla los ID de los registros.
- Obtiene la dirección de cada uno de los registros correspondientes al pozo actual.
- Extrae la información de los registros desde el archivo (*.imp).

- Convierte la información en objetos tipo Geoshare.
- Envía la información a GEOSHARE.

GEOSHARE encapsula la información de los registros en objetos Geoshare (217-LOG-RUN).

Mostrar Registros.



**DINT13:**
**CONSULTAR POZO**

**DIAGRAMA DE INTERACCIÓN**

Usuario activa Consultar Pozo.

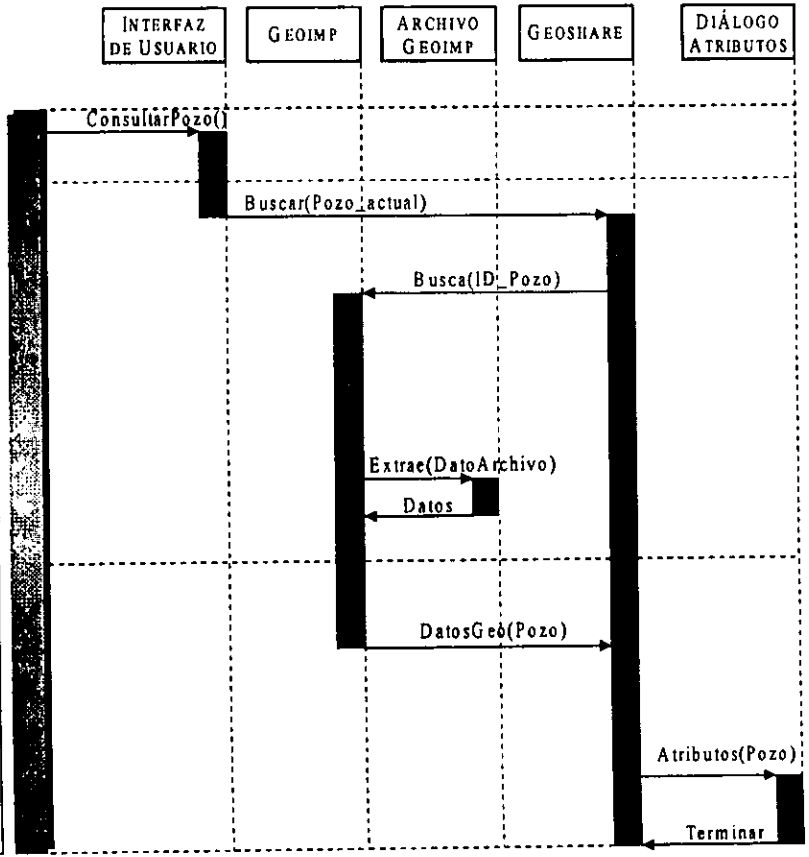
Solicitar buscar información del pozo actual.

GEOIMP:
- Busca en una tabla el ID del pozo.
- Obtiene la dirección del pozo actual correspondiente al archivo abierto (*.imp).
- Extrae la información del pozo desde el archivo (*.imp).

- Convierte la información en un objeto tipo Geoshare.
- Envía la información a GEOSHARE.

GEOSHARE encapsula la información del pozo en un objeto Geoshare (217-WELL-HEADER).

Mostrar atributos del Pozo.



**DINT14:**
**IMPRIMIR REGISTROS**

DIAGRAMA DE INTERACCIÓN

Usuario activa Imprimir Campo.
Mostrar opciones de impresión.

Solicitar buscar información de todos los registros del pozo actual.

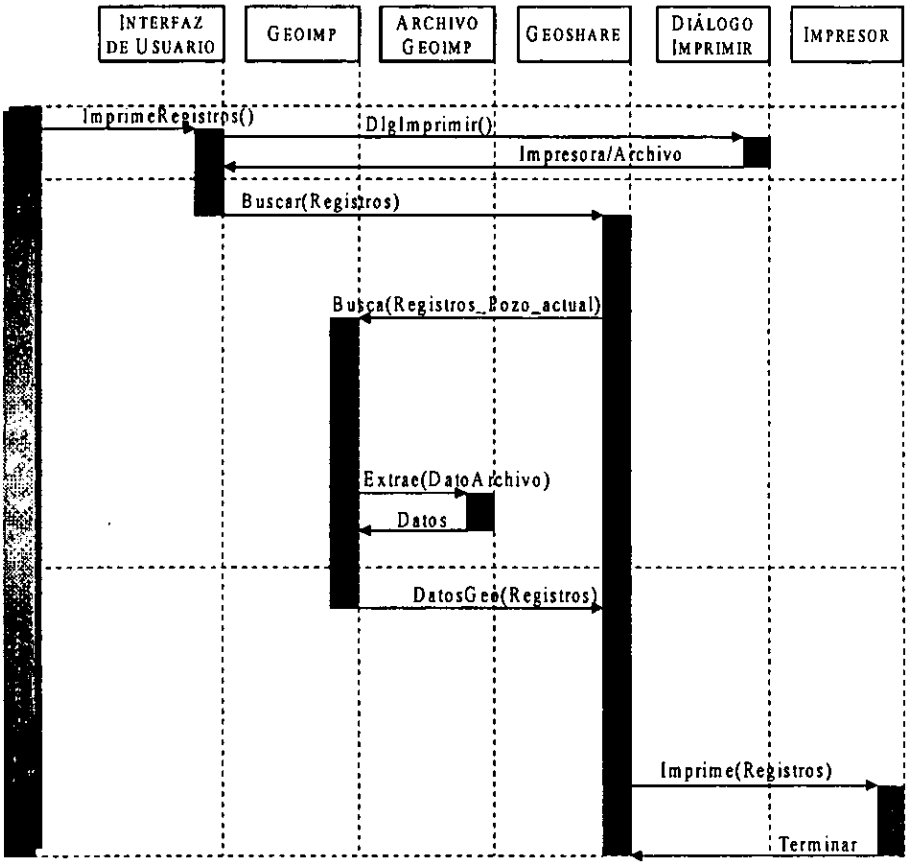
GEOIMP:

- Busca en una tabla los ID de los registros.
- Obtiene la dirección de cada uno de los registros correspondientes al pozo actual.
- Extrae la información de los registros desde el archivo (*.imp).

- Convierte la información en objetos tipo Geoshare.
- Envía la información a GEOSHARE.

GEOSHARE encapsula la información de los registros en objetos Geoshare (217-LOG-RUN).

Enviar a imprimir(Registros).



**DINT15:**
**IMPRIMIR POZO**

DIAGRAMA DE INTERACCIÓN

Usuario activa Imprimir Pozo.
Mostrar opciones de impresión.

Solicitar buscar información del pozo actual.

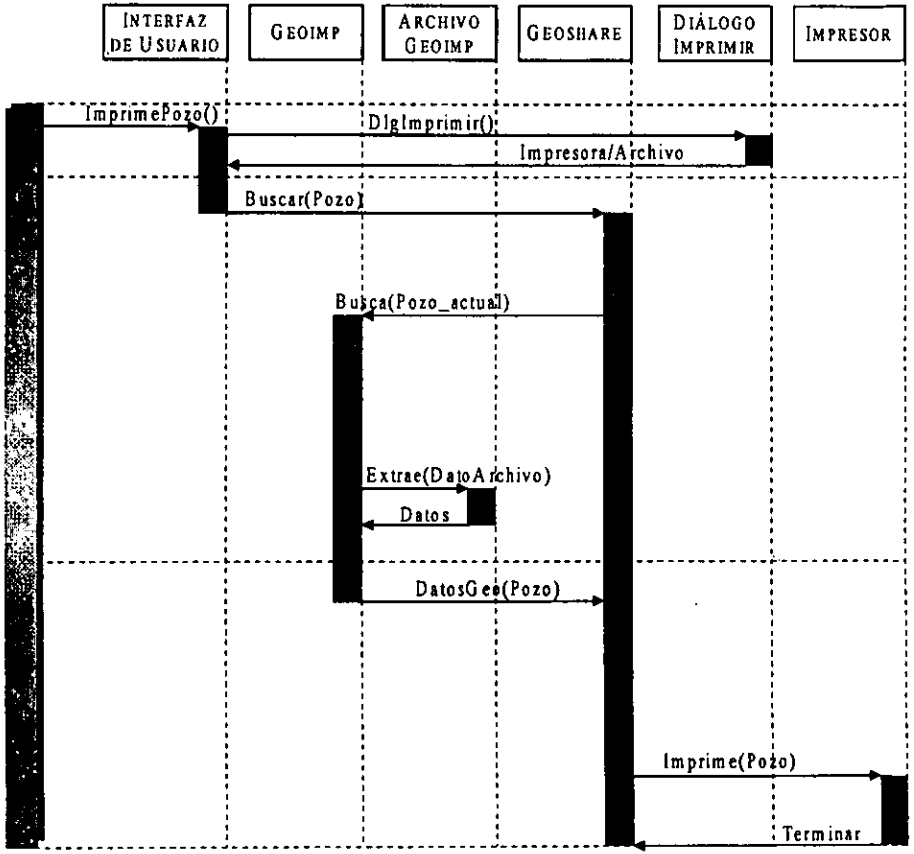
GEOIMP:

- Busca en una tabla el ID del pozo.
- Obtiene la dirección del pozo actual correspondiente al archivo abierto(*.imp).
- Extrae la información del pozo desde el archivo (*.imp).

- Convierte la información en un objeto tipo Geoshare.
- Envía la información a GEOSHARE.

GEOSHARE encapsula la información del pozo en un objeto Geoshare (217-WELL-HEADER).

Enviar a imprimir(Pozo).



# 6. IMPLEMENTACIÓN DEL MODELO.

## *6.1 DISEÑO DETALLADO.*

El diseño detallado tiene como objetivo identificar los atributos de las clases y los mecanismos necesarios para satisfacer los requerimientos solicitados, de cada uno de los componentes que conforman el sistema, muchos de los cuales ya han sido definidos a través del proceso interactivo e incrementativo entre el modelo de análisis, el de diseño y finalmente el de implementación.

El diseño detallado tiene como actividades principales:

1. Especificar el conjunto de requerimientos externos del componente o clase.
2. Si es un componente, se descomponen en clases o más componentes de acuerdo a sus requerimientos.
3. Si es una clase se deben especificar:

    - Atributos.
    - Estructuras.
    - Restricciones.
    - Mecanismos de memoria, almacenamiento de datos.

### 6.1.1 INTERFAZ DE USUARIO.

Es el componente encargado de administrar y controlar todos los eventos realizados por el usuario, así como el de interactuar con los componentes del sistema. Su representación es hecha a través del menú principal proporcionando todas las opciones que pueden ser usadas en el sistema.

Dentro del menú principal existen comandos por combinación de teclas (Alt + "N"), que permiten realizar acciones de una forma más rápida.

El menú principal es la ventana principal de la aplicación, se conforma por dos menús Campo y Pozo, en la Figura 6.1, se observa la estructura del menú principal. Es importante indicar que se hará uso de los recursos propios del compilador, para la realización de los elementos como: menús, aceleradores, diálogos implementados por el compilador y por el propio sistema operativo, etc.

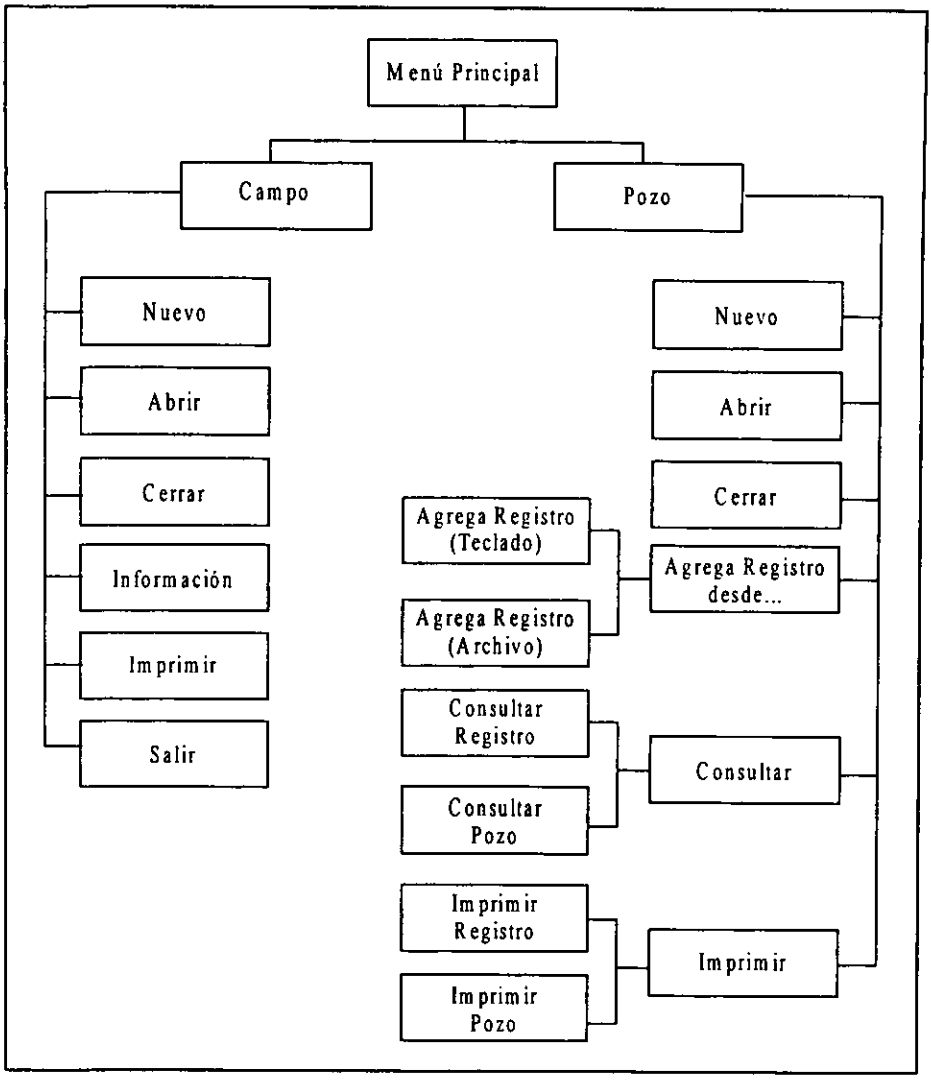



**Figura 6.1.- Modelo del Componente Interfaz de Usuario (Menú Principal).**

A continuación se definen las funciones necesarias para que la interfaz de usuario cumpla con todos sus requerimientos.

## 6.1.2 DEFINICIÓN DE LAS CLASES Y MÉTODOS DEL COMPONENTE INTERFAZ DE USUARIO.

| MENÚ: CAMPO | TIPO | ACTIVIDAD |
|---|---|---|
| MnuCampo_Nuevo ( ) | TMenuItem | Crea un nuevo archivo para el nuevo campo. |
| MnuCampo_Abrir ( ) | TMenuItem | Abre un archivo *.imp. |
| MnuCampo_Cerrar ( ) | TMenuItem | Cierra el archivo actual. |
| MnuCampo_Informar ( ) | TMenuItem | Muestra los datos del campo actual. |
| MnuCampo_Imprimir ( ) | TMenuItem | Imprime la información del campo actual. |
| MnuCampo_Salir ( ) | TMenuItem | Sale de la aplicación. |

| MENÚ: POZO | TIPO | ACTIVIDAD |
|---|---|---|
| MnuPozo_Nuevo ( ) | TMenuItem | Crea un nuevo pozo. |
| MnuPozo_Abrir ( ) | TMenuItem | Abre un pozo a partir del identificador único (ID). |
| MnuPozo_Cerrar ( ) | TMenuItem | Cierra el pozo actual. |
| SubMnuAgre_Teclado ( ) | TMenuItem | Crea un nuevo registro desde teclado. |
| SubMnuAgre_Archivo ( ) | TMenuItem | Crea uno o más registros desde un archivo (*.txt). |
| SubMnuCon_Reg ( ) | TMenuItem | Realiza una consulta de todos los registros que hay en el pozo actual. |
| SubMnuCon_Pozo ( ) | TMenuItem | Muestra la información del pozo actual. |
| SubMnuImp_Reg ( ) | TMenuItem | Imprime todos los registros que hay en el pozo actual. |
| SubMnuImp_Pozo ( ) | TMenuItem | Imprime la información del pozo actual. |

| MENÚ: POZO | CLASE | ACTIVIDAD |
|---|---|---|
| InicializaMnuCampo_1 ( ) | TFrmPrincipal | Inicializa Menú principal. |
| InicializaMnuCampo_2 ( ) | TFrmPrincipal | Inicializa Menú principal. |
| InicializaMnuPozo_1 ( ) | TFrmPrincipal | Inicializa Menú principal. |
| InicializaMnuPozo_2 ( ) | TFrmPrincipal | Inicializa Menú principal. |
| InicializaMnuPozo_3 ( ) | TFrmPrincipal | Inicializa Menú principal. |

CLASES PERTENECIENTES AL LENGUAJE DE LAS CUALES SE AUXILIA LA INTERFAZ DE USUARIO.

| | TIPO | ACTIVIDAD |
|---|---|---|
| DlgCampo_Nuevo | TSaveDialog | Guarda un archivo *.imp |
| DlgCampo_Abrir | TOpenDialog | Abre un archivo *.imp, *.txt. |
| DlgImprimir | TPrintDialog | Envía los trabajos a imprimir. |

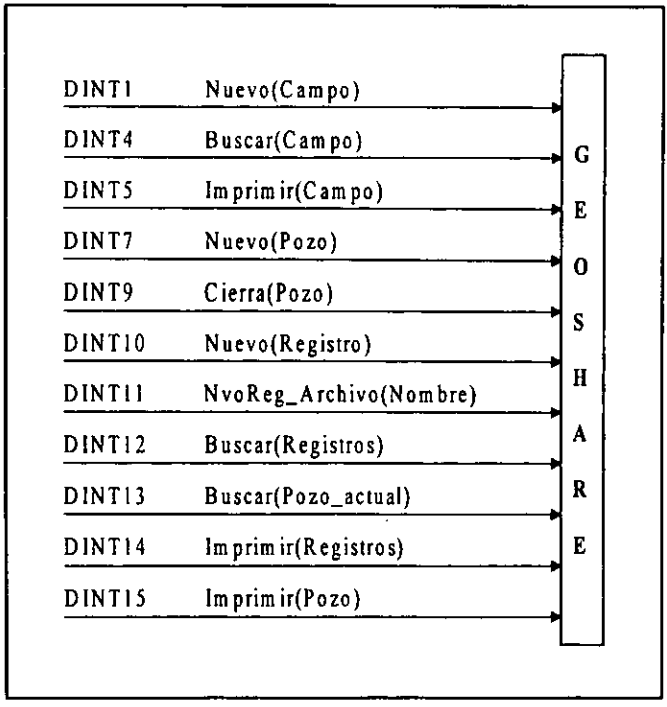
### 6.1.3 GEOSHARE.

Este componente tiene como tarea principal encapsular los datos de entrada en objetos de tipo Geoshare, así como puede crear, mostrar e imprimir los valores de un objeto tipo Geoshare. Los datos de entrada pueden ser dados desde teclado o un archivo *.txt. En la parte de impresión permite imprimir hacia un archivo o bien hacia la impresora. Para mostrar los valores de un determinado objeto pide el identificador de éste y una vez que es recuperado desde archivo muestra en un ventana de diálogo los datos del objeto.

Para el sistema sólo se van a implementar tres tipos de elementos :

1) 217-FIELD para un campo.
2) 217-WELL-HEADER para un pozo.
3) 217-LOG-RUN para un registro.

#### 6.1.3.1 Requerimientos de GEOSHARE.



### 6.1.3.2 Definición de las clases y métodos del componente GEOSHARE.

En base al análisis y diseño funcional, este componente va tener una clase abstracta llamada Geoshare, la cual tiene como propósito capturar la *comunalidad* de varias clases derivadas que sí podrán crear objetos, estas clases son FIELD, WELL_HEADER y LOG_RUN, a la vez la clase base va derivar de la clase TObject propia del compilador.

TObject es el último antecesor de todos los componentes y objetos en Borland C++Builder[24], cuenta con métodos que proporcionan:

- La habilidad para crear, mantener y destruir una instancia de un objeto por asignación, inicialización y liberación de la memoria necesaria para aquel objeto.
- Tipo de clase e información de la instancia sobre un objeto e información que se da en tiempo de corrida (RTTI "runtime type information").
- Soporte para el manejo de mensajes.

Las clases Field, Well_Header y Log_Run encapsulan los atributos de las estructuras 217-Field, 217-Log-Run y 217-Well-Header respectivamente del formato Geoshare, así como tienen métodos para poder crear un objeto, mostrar e imprimir los valores de sus datos miembros. La relación de estas clases será controlado internamente por el sistema, donde se va conservar la relación de la estructura del formato Geoshare.

La clase abstracta (Geoshare), sirve de patrón o modelo para la creación de nuevas clases, agrupa las características más generales, para que se deriven clases más específicas, con ello se establece una jerarquía de clasificación[25]. Es importante señalar, que de una clase abstracta no se pueden crear instancias (objetos) y esta clase al menos debe de contener una función virtual pura, es decir, una función de la cual sólo se especifica su encabezamiento pero no se define su cuerpo[26].

La Figura 6.2 muestra el modelo del componente GEOSHARE.

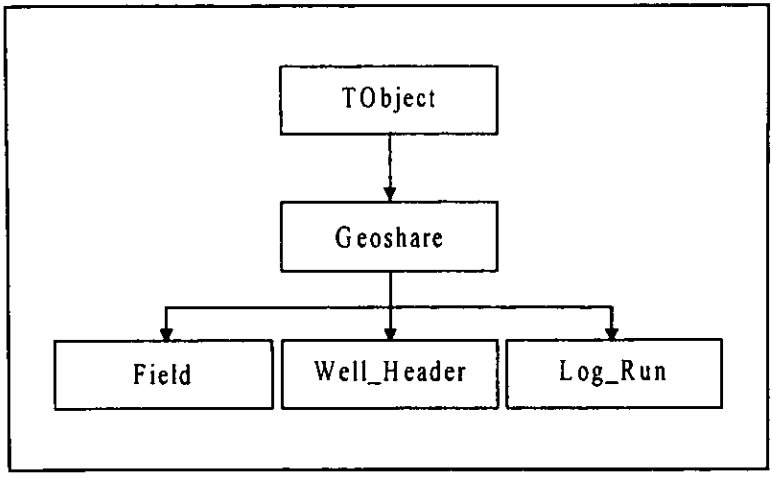



**Figura 6.2.- Modelo del componente GEOSHARE.**

CLASS GEOSHARE : PUBLIC TOBJECT (CLASE BASE ABSTRACTA)

| MÉTODOS PÚBLICOS | ACTIVIDAD |
|---|---|
| Geoshare ( ) | Constructor. |
| virtual bool Nuevo(void) = 0 | Función virtual pura, crea un nuevo objeto tipo Geoshare, para las clases derivadas. |
| virtual void MuestraValores(void)= 0 | Función virtual pura, muestra los valores de los datos miembros de un objeto tipo Geoshare, para las clases derivadas. |
| virtual void Imprime(void) = 0 | Función virtual pura, imprime los datos miembros de un objeto tipo Geoshare, para las clases derivadas. |
| ~Geoshare ( ) | Destructor. |

CLASS FIELD : PUBLIC GEOSHARE (HERENCIA SIMPLE)

| DATOS MIEMBROS PRIVADOS | TIPO |
|---|---|
| Field_Name[30] | char |
| Field_Type[20] | char |
| Wells_In_Field | int |

| MÉTODOS PÚBLICOS | ACTIVIDAD |
|---|---|
| Field ( ) | Constructor. |
| virtual bool Nuevo(void) | Función virtual pura, crea un nuevo objeto tipo 217-Field. |
| virtual void MuestraValores(void) | Función virtual pura, muestra los valores de los datos miembros de un objeto tipo 217-Field. |
| virtual void Imprime(void) | Función virtual pura, imprime los datos miembros de un objeto tipo 217-Field. |
| ~Field ( ) | Destructor. |

CLASS WELL_HEADER : PUBLIC GEOSHARE (HERENCIA SIMPLE)

| DATOS MIEMBROS PRIVADOS | TIPO |
|---|---|
| Unique_Well_Id | int |
| Well_Name[21] | char |
| Well_Number | int |
| Short_Name[11] | char |
| Current_Status[23] | char |
| Original_Status[23] | char |
| Previous_Status[23] | char |
| Well_Total_Depth | float |
| Datum_Elevation | float |
| Elev_Physical_Ref[3] | char |
| Depth_Unit[15] | char |
| Time_Unit[11] | char |
| Original_Unit_System | int |
| Ground_Elevation | float |
| Kelly_Bushing_Elevation | float |
| Casing_flange_Elevation | float |
| Last_Update[11] | char |
| Log_Runs | int |
| Company[21] | char |

| MÉTODOS PÚBLICOS | ACTIVIDAD |
|---|---|
| Well_Header( ) | Constructor. |
| virtual bool Nuevo(void) | Función virtual pura, crea un nuevo objeto tipo 217-Well-Header. |
| virtual void MuestraValores(void) | Función virtual pura, muestra los valores de los datos miembros de un objeto tipo 217-Well-Header. |
| virtual void Imprime(void) | Función virtual pura, imprime los datos miembros de un objeto tipo 217-Well-Header. |
| ~Well_Header( ) | Destructor. |

CLASS LOG_RUN : PUBLIC GEOSHARE

| DATOS MIEMBROS PRIVADOS | TIPO |
|---|---|
| Run_Number | int |
| Service_Company[20] | char |
| Run_Data[11] | char |
| Logging_Unit_Number[10] | char |
| Logging_Unit_Location[10] | char |
| Recorder[30] | char |
| Witness[30] | char |
| Elev_Ref_Height | float |
| Elev_Physical_Ref[3] | char |
| Mud_Type[10] | char |
| Mud_Salinity | float |
| Mud_Density | float |
| Mud_Viscosity | float |
| Mud_Fluid_loss | float |
| Mud_Acidity | float |
| Mud_Resist | float |
| Mud_Filtrate_Resist | float |
| Mud_Cake_Resist | float |
| Mud_Resist_Source[10] | char |
| Mud_Bht_Resist | float |
| Mud_Bht_Filtrate_Resist | float |
| Mud_Bht_Cake_Resist | float |
| Mud_Bht_Resist_Source[10] | char |
| Mud_Source[10] | char |
| Mud_Temperature | float |
| Mud_Filtrate_Source[10] | char |
| Mud_Filtrate_Temperature | float |
| Mud_Cake_Source[10] | char |
| Mud_Cake_Temperature | float |
| Time_Circulation_Stop[11] | char |
| Bottom_Temp_Max | float |
| Log_Passes[2] | char |
| Drill_Measured_Depth | float |
| Log_Measured_Depth | float |
| Bottom_Pressure_Max | float |
| Surface_Hole_Temperature | float |
| Wellpath_Segment[15] | char |
| Planning_Data[1] | char |
| Service_Order[5] | char |
| Permanent_Datum_Elevation | float |
| Permanent_Datum[15] | char |

| MÉTODOS PRIVADOS | ACTIVIDAD |
|---|---|
| bool AlmacenaNvoArch(String *contlog) | Genera un objeto para cada registro que se encuentre en un archivo *.txt. |

| MÉTODOS PÚBLICOS | ACTIVIDAD |
|---|---|
| Log_Run( ) | Constructor. |
| virtual bool Nuevo(void) | Función virtual pura, crea un nuevo objeto tipo 217- Log-Run. |
| virtual void MuestraValores(void) | Función virtual pura, muestra los valores de los datos miembros de un objeto tipo 217- Log-Run. |
| virtual void Imprime(void) | Función virtual pura, imprime los datos miembros de un objeto tipo 217- Log-Run. |
| bool NvoArch(const char *nom_archivo) | Abre un archivo *.txt y extrae los nuevos registros para el pozo actual. |
| ~Well_Header( ) | Destructor. |

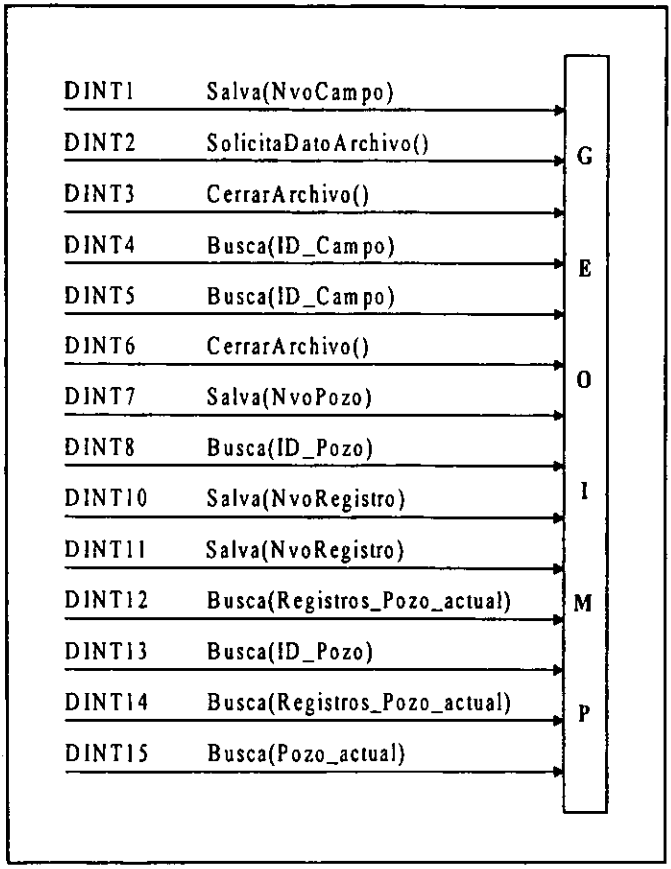
FUNCIONES GLOBALES

| Funciones | ACTIVIDAD |
|---|---|
| void PrintLinea(TPrinter *Impresor, int LineaPag, String &pStr) | Imprime una línea. |
| void ConFigImp(String *Linea, int TotLinea) | Configura la impresora. |

### 6.1.4 GEOIMP.

Tiene como objetivos dotar de persistencia a los objetos tipo Geoshare, así como aplicar el formato RP66 (DLIS), este elemento trabaja principalmente con el elemento GEOSHARE, de donde recibe la petición de guardar o recuperar un objeto tipo Geoshare hacia/desde un archivo híbrido denominado ARCHIVO_GEOIMP, también existe una relación con la interfaz de usuario, para validar el archivo *.imp.

#### 6.1.4.1 Requerimientos de GEOIMP.



### 6.1.4.2 Definición de las clases y métodos del componente GEOIMP.

De acuerdo al análisis y diseño funcional, este componente va estar formado de dos partes básicamente una dedicada a implementar la persistencia y otra destinada a aplicar el formato RP66 (DLIS).

Implementación del formato RP66 (DLIS).

Para esta parte el formato se lleva a cabo a nivel estructuras, es decir, no se usa la conversión a nivel bytes como lo indica el mismo, pero eso no es motivo para no cumplir con las reglas de almacenamiento.

Se tiene una clase base abstracta denominada FL que en si se encarga de englobar las características comunes que tiene cada bloque que forman parte de un archivo lógico, para que se deriven las clases LR y LRS.

Es necesario recordar que cada archivo lógico va estar compuesto de varios registros lógicos (LR) y cada LR va estar formado por la longitud del registro visible (VRL), versión del formato (FV) y segmentos de registro lógico (LRS) y a la vez este último va estar integrado por un encabezado de segmento del registro lógico (LRSH), cuerpo de segmento del registro lógico (LRSB) y por rastrero del registro lógico (LRST). Ver Figura 6.3.

En el cuerpo de cada segmento de registro lógico se va almacenar el nombre de la clase del objeto, el objeto y el offset del mismo. El offset va indicar el objeto del cual proviene de acuerdo al formato Geoshare, por ejemplo un objeto tipo Log_Run, tiene como antecesor un objeto tipo Well_Header, con ello se garantiza que la relación entre ellos se mantenga. Ver Figura 6.4.

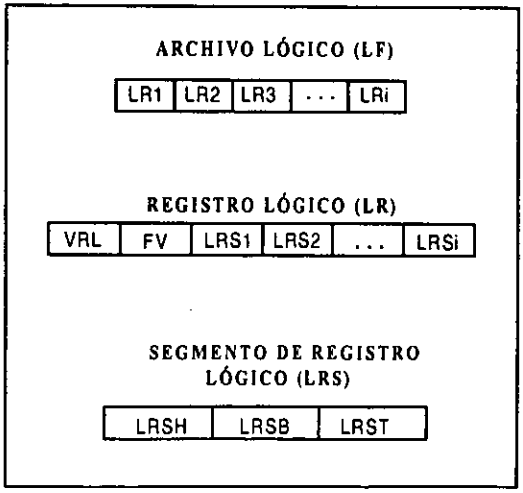



**Figura 6.3.- Estructura de un archivo en formato RP66 (DLIS).**

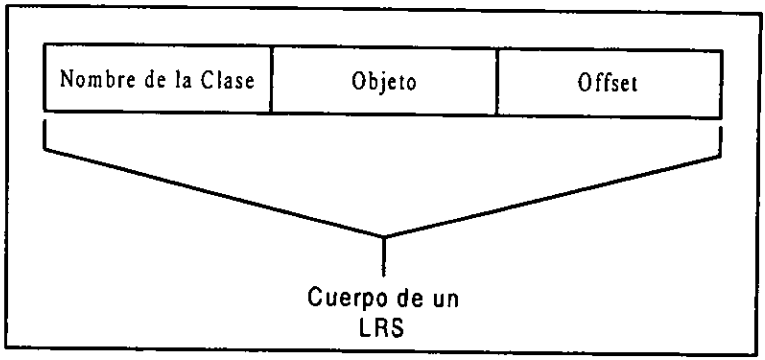



**Figura 6.4.- Almacenamiento de un objeto.**

Implementación de la persistencia.

Se implementará la persistencia isomórfica, es decir, se conservará los apuntadores relacionados entre los objetos persistentes. Para implementar ésta se desarrolla una técnica basada en el manejo de memoria y apuntadores inteligentes (SPs "Smart Pointers"), pero la técnica basada solamente en la clásica definición de SP no es suficiente, por ello se realizarán cambios esenciales para poder cubrir con todos los requerimientos antes establecidos.

Un SP es un apuntador inteligente hacia un objeto, suministra una extra redirección, cual, sin embargo, permite oportunas operaciones a ser agregadas a aquellas que normalmente son ejecutadas por la construcción de un apuntador, proporcionando un control completo de los objetos creados dinámicamente.

Para el sistema un SP es una estructura en donde se guarda los cuerpos de los apuntadores inteligentes (SP "smart pointer"), se conforma de dos partes una es el número de registro lógico donde se encuentra el objeto, con respecto al archivo actual y la otra parte es la dirección relativa del segmento de registro lógico con respecto al registro lógico.

En la Figura 6.5, se aprecia el funcionamiento de un SP, en este caso el objeto se encuentra en el registro lógico 2 con un desplazamiento de 40 bytes.

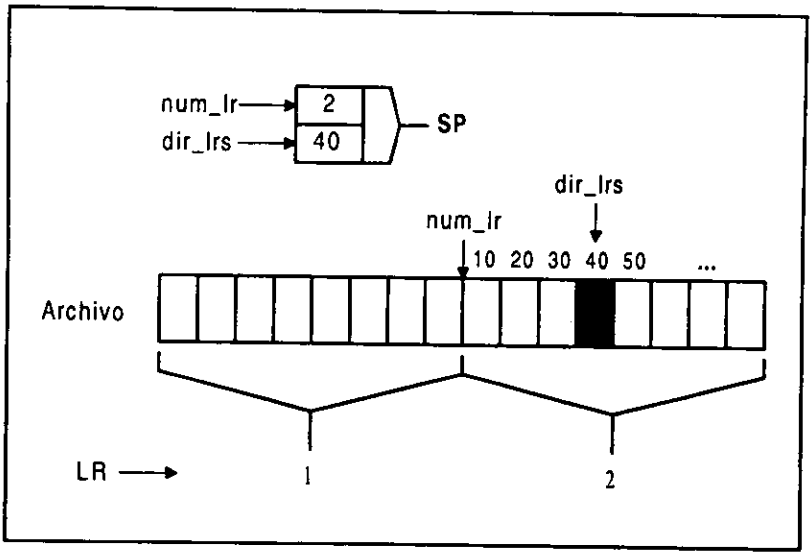



**Figura 6.5.- Estructura de un SP.**

Se tienen arreglos de SPs para cada clase de tipo Geoshare, por consiguiente para las clases Field, Well_Header y Log_Run, se va tener los arreglos SPCampo, SPPozo y SPReg respectivamente, como se muestra en la Figura 6.6.

Los arreglos de SPs pueden contener unos miles de SPs, se van a localizar en memoria RAM, y van a ir almacenando SP por cada objeto que se cree, al final de una sesión estos arreglos deben de ser almacenado en memoria secundaria y se van a colocar en el primer LR de cada archivo, en donde se encuentran los objetos. Ver Figura 6.7.

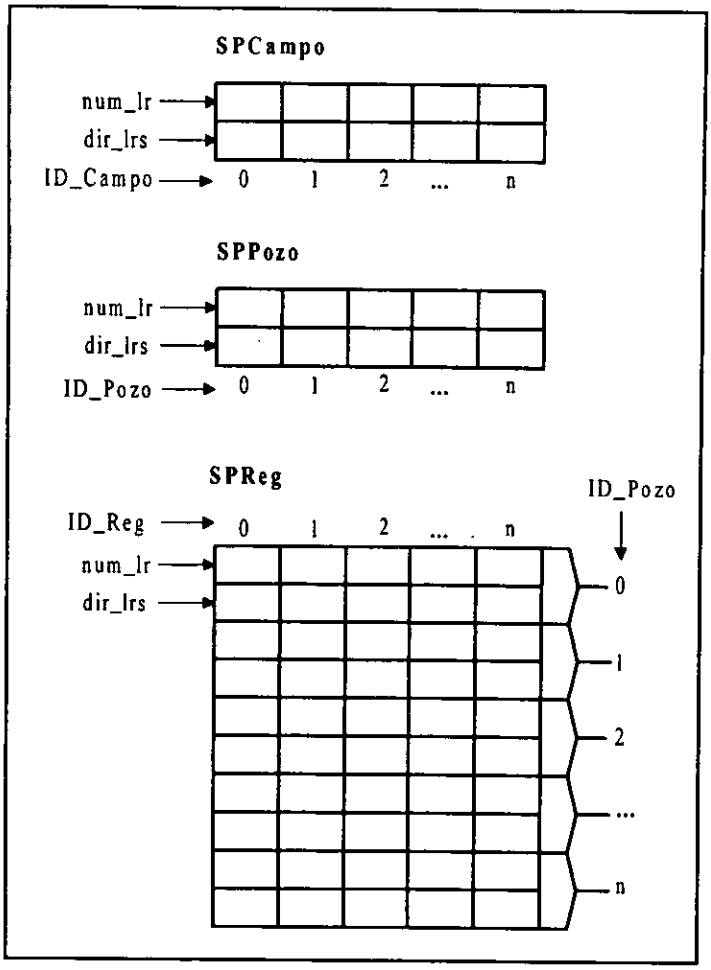



**Figura 6.6.- Tipo de arreglos de SPs que existen en el sistema.**

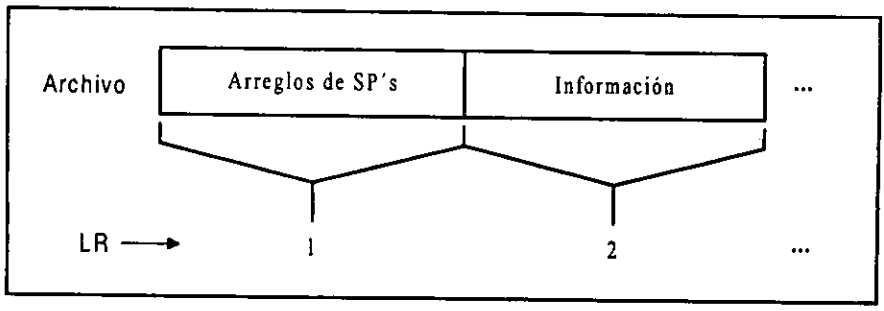



**Figura 6.7.- Almacenamiento de los arreglos de SPs en un archivo.**

Un punto importante es el identificador de cada objeto, estos son generados por la clase Genera_ID, en si representan el índice de cada arreglo de SPs. Para poder accesar a un objeto en particular el usuario usará el ID del objeto y a partir de ahí puede accesar a los objetos que éste se encuentre apuntado, por ejemplo si se quiere accesar al objeto pozo con ID igual a 4, a partir de esa información se puede accesar a todos los objetos registros que pertenezcan al objeto pozo, dado que el SPReg mantiene como parte de su índice el identificador del pozo (ID_Pozo).

En la Figura 6.8 muestra el modelo del componente GEOIMP.

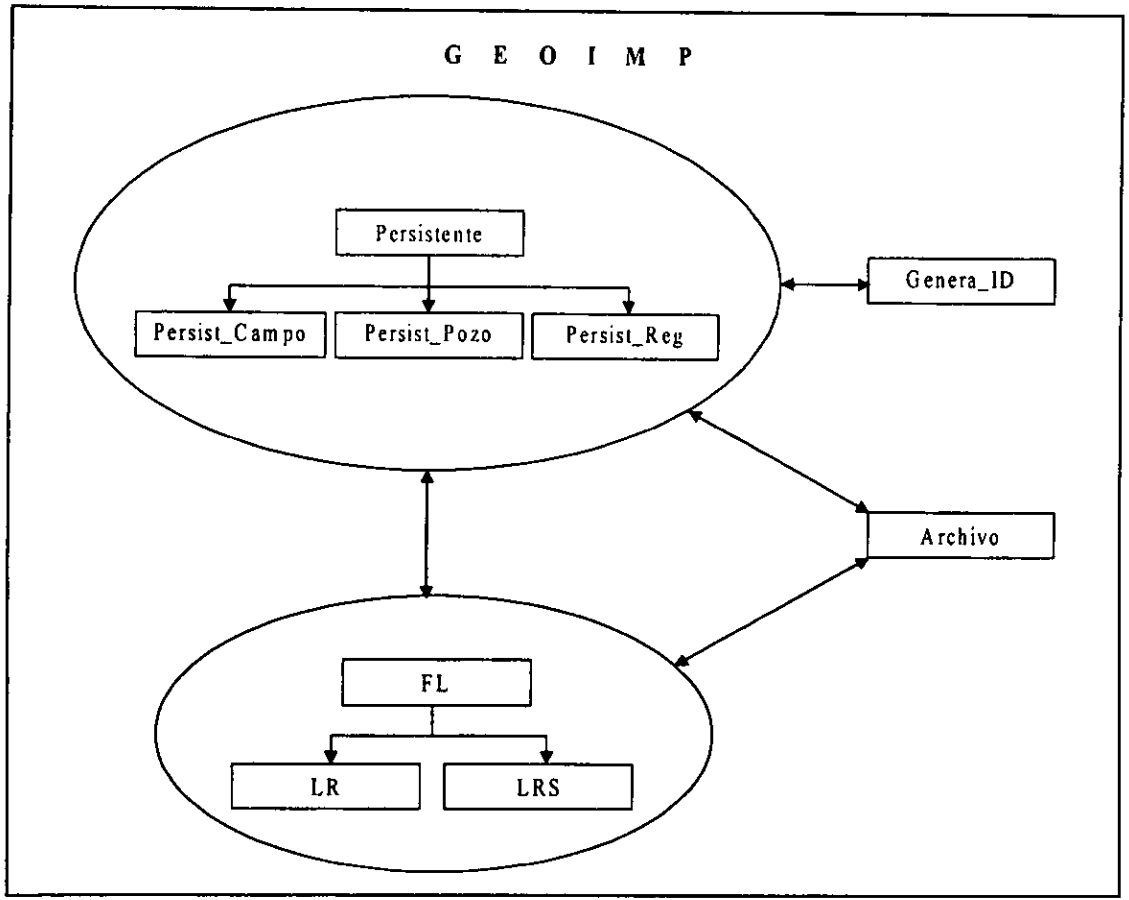



**Figura 6.8.- Modelo del componente GEOIMP.**

CLASS PERSISTENTE

| DATOS MIEMBROS PROTEGIDOS | TIPO |
|---|---|
| maneja_Campo | int |
| maneja_Pozo | int |
| maneja_Reg | int |
| nombre_Clase[30] | char |

| MÉTODOS PROTEGIDOS | ACTIVIDAD |
|---|---|
| int AsignaSP(String &nomb_clase) | A partir del tipo de objeto se da valor al SP del nuevo objeto, el SP sólo contiene la dirección relativa del objeto en el archivo actual. Finalmente regresa un indicador del objeto del cual se deriva. |
| struct SP Localiza(String &nomb_clase) | Encuentra el número de LR y el desplazamiento del LRS que contiene al objeto, regresa la información a través de una estructura tipo SP. |
| void Busca(struct SP *localiza, int &tamlrsh) | Busca en el arreglo de SPs la localidad del objeto en el archivo actual, y coloca el apuntador del archivo donde se encuentra el objeto, la búsqueda es en forma binaria. |

| MÉTODOS PÚBLICOS | ACTIVIDAD |
|---|---|
| Persistente(int idcampo, int idpozo, int idreg) | Constructor recibe los ID's del Campo, Pozo u/o Registro actual y lo asigna a sus datos miembros. |
| ~Persistente ( ) | Destructor. |

PERSIST_CAMPO: PUBLIC PERSISTENTE (HERENCIA SIMPLE)

| MÉTODOS PRIVADOS | ACTIVIDAD |
|---|---|
| int Almacena(class Field* objeto, int offset) | Recibe un apuntador al objeto Field y su offset, guarda en disco el nombre de la clase, el objeto y offset del mismo, y regresa el tamaño en bytes del objeto tipo Field. El offset indica de que objeto proviene. |
| Field* Recupera( ) | Recupera un objeto persistente (de tipo Field) desde el archivo actual, a partir de su dirección en disco y regresa un apuntador al tipo de objeto. |

| MÉTODOS PÚBLICOS | ACTIVIDAD |
|---|---|
| Persist_Campo(int idcampo, int idpozo, int idreg) : Persistente(idcampo, idpozo, idreg) | Constructor recibe los ID's del Campo, Pozo u/o Registro actual y lo asigna a sus datos miembros. |
| void Salva(Field* objeto) | Guarda en el archivo actual un objeto de tipo Field bajo el formato RP66 y aplica persistencia, tiene como entrada un apuntador al objeto a guardar. |
| Field* Reconstruye(String nomb_clase) | Recupera desde el archivo actual un objeto persistente de tipo Field bajo el formato RP66 y regresa un objeto de tipo Geoshare. |
| ~Persist_Campo ( ) | Destructor. |

PERSIST_POZO : PUBLIC PERSISTENTE (HERENCIA SIMPLE)

| MÉTODOS PRIVADOS | ACTIVIDAD |
|---|---|
| int Almacena(class Well_Header* objeto, int offset) | Recibe un apuntador al objeto Well_Header y su offset, guarda en disco el nombre de la clase, el objeto y offset del mismo, y regresa el tamaño en bytes del objeto tipo Well_Header. |
| Well_Header * Recupera( ) | Recupera un objeto persistente (de tipo Well_Header) desde el archivo actual, a partir de su dirección en disco y regresa un apuntador al tipo de objeto. |

| MÉTODOS PÚBLICOS | ACTIVIDAD |
|---|---|
| Persist_Pozo(int idcampo, int idpozo, int idreg) : Persistente(idcampo, idpozo, idreg) | Constructor recibe los ID's del Campo, Pozo u/o Registro actual y lo asigna a sus datos miembros. |
| void Salva(Well_Header* objeto) | Guarda en el archivo actual un objeto de tipo Well_Header bajo el formato RP66 y aplica persistencia, tiene como entrada un apuntador al objeto a guardar. |
| Well_Header* Reconstruye(String nomb_clase) | Recupera desde el archivo actual un objeto persistente de tipo Well_Header bajo el formato RP66 y regresa un objeto de tipo Geoshare. |
| ~Persist_Pozo ( ) | Destructor. |

PERSIST_REG: PUBLIC PERSISTENTE (HERENCIA SIMPLE)

| MÉTODOS PRIVADOS | ACTIVIDAD |
|---|---|
| int Almacena(class Log_Run* objeto, int offset) | Recibe un apuntador al objeto Log_Run y su offset, guarda en disco el nombre de la clase, el objeto y offset del mismo, y regresa el tamaño en bytes del objeto tipo Log_Run. |
| Log_Run* Recupera( ) | Recupera un objeto persistente (de tipo Log_Run) desde el archivo actual, a partir de su dirección en disco y regresa un apuntador al tipo de objeto. |

| MÉTODOS PÚBLICOS | ACTIVIDAD |
|---|---|
| Persist_Reg(int idcampo, int idpozo, int idreg) : Persistente(idcampo, idpozo, idreg) | Constructor recibe los ID's del Campo, Pozo u/o Registro actual y lo asigna a sus datos miembros. |
| void Salva(Log_Run* objeto) | Guarda en el archivo actual un objeto de tipo Log_Run bajo el formato RP66 y aplica persistencia, tiene como entrada un apuntador al objeto a guardar. |
| Log_Run* Reconstruye(String nomb_clase) | Recupera desde el archivo actual un objeto persistente de tipo Log_Run bajo el formato RP66 y regresa un objeto de tipo Geoshare. |
| ~Persist_Reg ( ) | Destructor. |

CLASS FL (CLASE BASE ABSTRACTA)

| MÉTODOS PÚBLICOS | ACTIVIDAD |
|---|---|
| FL ( ) | Constructor. |
| virtual void Encabezado(void) = 0 | Función virtual pura, crea un nuevo encabezado tipo FL, para las clases derivadas. |
| virtual int AsignaValEnc(int &tam_bloque) = 0 | Función virtual pura, asigna un nuevo valor al encabezado tipo FL, para las clases derivadas. |
| ~FL ( ) | Destructor. |

CLASS LR : PUBLIC FL (HERENCIA SIMPLE)

| MÉTODOS PRIVADOS | ACTIVIDAD |
|---|---|
| bool NvoIr(void) | Verifica si el nuevo objeto será guardado en el LR actual o en uno nuevo. |

DATO MIEMBRO PROTEGIDO

STRUCT FHLR : CONTIENE ENCABEZADO DEL LR

| VARIABLES | ACTIVIDAD |
|---|---|
| int vrl | Longitud del registro visible. |
| char fv[3] | Versión del formato. |
| int num_lrs | Número de LRS registrados |

| MÉTODOS PÚBLICOS | ACTIVIDAD |
|---|---|
| LR ( ) | Constructor. |
| virtual void Encabezado(void) | Función virtual pura, crea un nuevo encabezado LR en el archivo actual. |
| virtual int AsignaValEnc(int &tam_bloque) | Función virtual pura, actualiza el encabezado de un LR, tiene como entrada el tamaño actual del nuevo LRS. |
| ~LR ( ) | Destructor. |

CLASS LRS : PUBLIC FL

DATOS MIEMBROS PROTEGIDOS

STRUCT LRSH: CONTIENE ENCABEZADO DEL LRS

| VARIABLES | ACTIVIDAD |
|---|---|
| int lrsl | Longitud del registro lógico. |
| char atributo | Atributos del LRS. |
| char lrt | Tipo de registro lógico. |

STRUCT LRSFIN: CONTIENE FIN DEL LRS

| VARIABLE | ACTIVIDAD |
|---|---|
| char lrst[10] | Trailing del LRS |

| MÉTODOS PRIVADOS | ACTIVIDAD |
|---|---|
| void AsignaValFin(void) | Da valores al encabezado de un LRS. |

| MÉTODOS PÚBLICOS | ACTIVIDAD |
|---|---|
| LRS ( ) | Constructor. |
| virtual void Encabezado(void) | Función virtual pura, crea un nuevo encabezado LR en el archivo actual. |
| virtual int AsignaValEnc(int &tam_bloque) | Función virtual pura, actualiza el encabezado de un LR, tiene como entrada el tamaño actual del nuevo LRS. |
| int Fin(void) | Coloca en el archivo actual el lrst de un LRS y regresa el tamaño en bytes de la estructura lrsfin. |
| int Tam_lrsh() | Da el tamaño en bytes de la estructura lrsh y regresa el tamaño en bytes de la estructura lrsh. |
| ~LRS ( ) | Destructor. |

CLASS ARCHIVO

| MÉTODOS PÚBLICOS | ACTIVIDAD |
|---|---|
| Archivo ( ) | Constructor no abre ningún archivo y no inicializa, se utiliza al cerrar el archivo. (Sobrecarga del constructor). |
| Archivo(const char *nom_archivo) | Constructor abre un archivo(*.imp) e inicializa los arreglos de SPs, se utiliza al abrir o al crear un archivo. (Sobrecarga del constructor). |
| bool Carga_SP() | Toma desde el archivo actual los arreglos de SPs y los coloca en memoria RAM. Regresa un valor de verdadero si no hubo problemas. |
| bool Descarga_SP() | Coloca en memoria secundaria los arreglos de SPs que están en RAM. Regresa un valor de verdadero si no hubo problemas. |
| ~Archivo ( ) | Destructor. |

CLASS GENERA_ID

| MÉTODOS PÚBLICOS | ACTIVIDAD |
|---|---|
| Genera_ID ( ) | Constructor. |
| int ID() | Genera un ID para un nuevo Campo. En este sistema siempre será uno, dado que solamente se puede tener un campo por archivo. |
| int ID(int idcampo) | Genera un ID para un nuevo Pozo, tiene como entrada el ID del campo al que pertenece. |
| Int ID(int idcampo, int idpozo) | Genera un ID para un nuevo Registro, tiene como entrada el ID del pozo al que pertenece. |
| ~ Genera_ID( ) | Destructor. |

CONSTANTES GLOBALES

| CONSTANTES | ACTIVIDAD |
|---|---|
| const unsigned short int NUMLR | Número máximo de LR que puede haber en un archivo. |
| const unsigned short int NUMLRS | Número máximo de LRS que puede haber en un LR. |
| const unsigned short int NUMCAMPOS | Número máximo de Campos que puede haber en un archivo. |
| const unsigned short int NUMPOZOS | Número máximo de Pozos que puede haber en un archivo. |
| const unsigned short int NUMREG | Número máximo de Registros que puede haber en un archivo. |

VARIABLES GLOBALES

STRUCT SP: CONTIENE LA ESTRUCTURA DE UN SMART POINTER.

| VARIABLES | ACTIVIDAD |
|---|---|
| int num_lr | Número de LR en el archivo actual. |
| int dir_lrs | Dirección relativa del LRS con respecto a LR |

| Variables | ACTIVIDAD |
|---|---|
| fstream f | Apuntador a un archivo. |
| SP SPCampo[NUMCAMPOS] | Arreglo de SPs para el Campo. |
| SP SPPozo[NUMPOZOS] | Arreglo de SPs para los Pozos. |
| SP SPReg[NUMPOZOS][NUMREG] | Arreglo de SPs para los Registros. |
| int tam_lr[NUMLR] | Arreglo que contiene el tamaño en bytes de cada bloque de LR. |
| int loca_Campo | Número de objetos de tipo 217-FIELD. |
| int loca_Pozo | Número de objetos de tipo 217-WELL_HEADER |
| int loca_lr | Número de LR registrados |
| int loca_lrs | Número de LRS registrados en el LR actual |

| FUNCIONES GLOBALES | ACTIVIDAD |
|---|---|
| bool Verifica_IDCampo(int id) | Verifica si existe el ID_Campo en el archivo actual. |
| bool Verifica_IDPozo(int id) | Verifica si existe el ID_Pozo en el archivo actual. |
| bool Verifica_IDReg(int idpozo) | Verifica si existe el ID_Reg en el archivo actual. |

### 6.1.5 PRUEBAS.

Las pruebas realizadas para verificar y validar el funcionamiento del sistema fueron divididas en tres niveles, éstas son:

- Pruebas de unidad.
- Pruebas de integración.
- Pruebas de sistema.

El equipo donde se realizaron las pruebas fue en una computadora personal con procesador pentium a 100 MHz, 16 MB en memoria RAM, disco duro de 2.0 GB (espacio libre 500 MB), con sistema operativo Windows 95.

#### 6.1.5.1 Pruebas de unidad.

Consiste en probar el funcionamiento de una clase por sí misma. Para el sistema se probaron las clases:

A) Elemento GEOSHARE:

- Field
- Well_Header
- Log_Run

B) Elemento GEOIMP:

- Persist_Campo
- Persist_Pozo
- Persist_Reg
- Genera_ID
- FL
- LR
- LRS
- Archivo

Las pruebas realizadas verificaron que cada unidad tomada por sí misma funciona correctamente. Para las clases abstractas Geoshare y Persistente no fue posible realizar las pruebas dado que éstos sólo contienen funciones virtuales puras, las cuales sólo especifican su encabezamiento pero no definen su cuerpo, por lo cual no es posible crear instancias de éstas, pero se validaron en forma indirecta , ya que las clases que derivaron de ellas funcionaron de manera satisfactoria.

### 6.1.5.2 Pruebas de integración.

Las pruebas de integración deben probar la integración de cada clase con las demás. Para este nivel se probó que cada elemento cumpliera con los requerimientos que se establecieron en la fase de diseño así como se verificó la forma en que interactúan con los demás componentes, en base a ello se obtuvo que todos los elementos trabajan de manera correcta.

Para el elemento GEOSHARE las pruebas de integración se resumen en la Figura 6.9, donde se tiene como entradas los requerimientos de los demás elementos, así como, se tiene como salida peticiones a otros elementos para poder llevar a cabo las tareas asignadas, con ello se establece la integración de éste con los demás. Se reporta que se cumplió con todos los requerimientos.

Para el elemento GEOIMP las pruebas de integración se resumen en la Figura 6.10, donde se tiene como entradas los requerimientos de los demás elementos, además, se tiene como salida peticiones a otros elementos para poder llevar a cabo las tareas asignadas, con esto se establece la integración de éste con los demás. Se reporta que se cumplió con todos los requerimientos.

Básicamente los elementos GEOSHARE y GEOIMP cubren todos los requerimientos del sistema, sin dejar a fuera a la interfaz de usuario que juega un papel no menos importante.

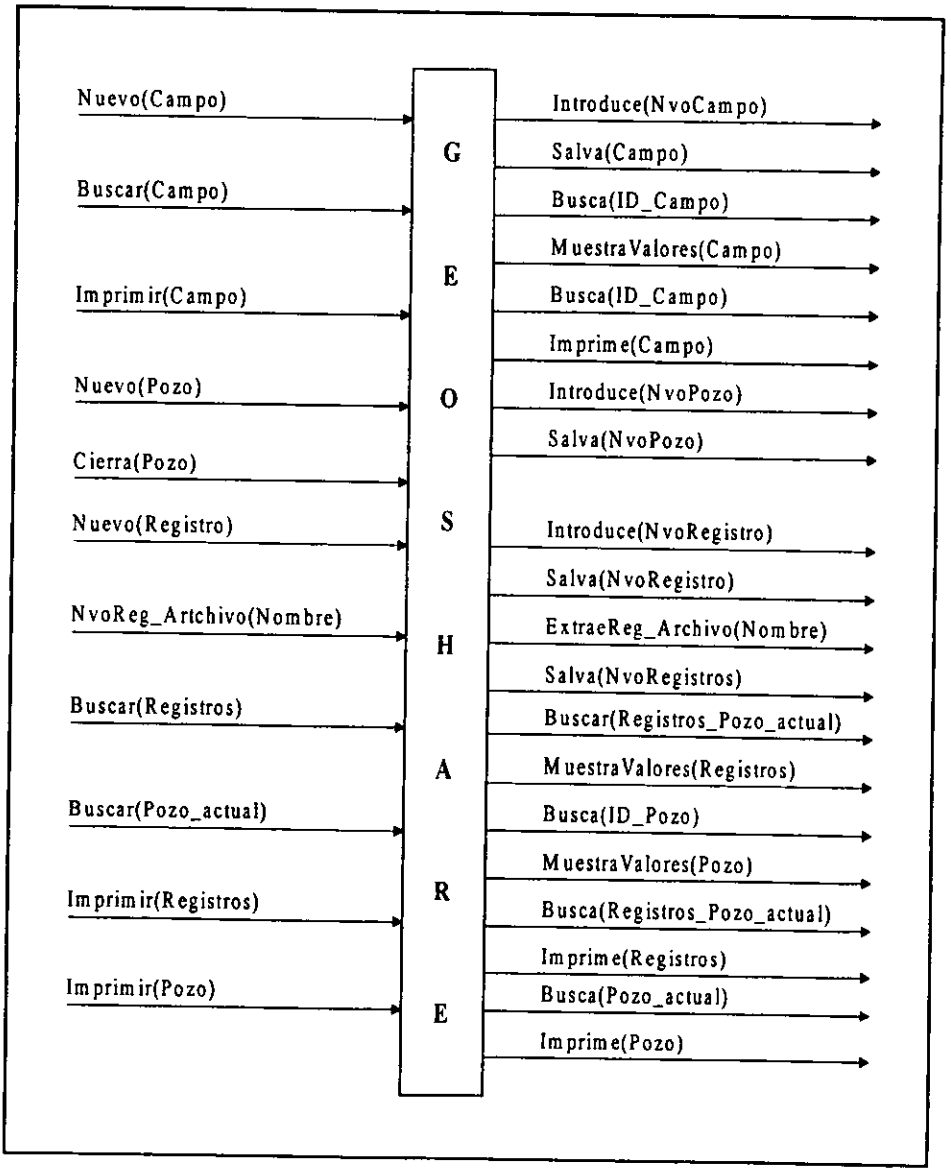



**Figura 6.9.- Pruebas de Integración del elemento GEOSHARE.**

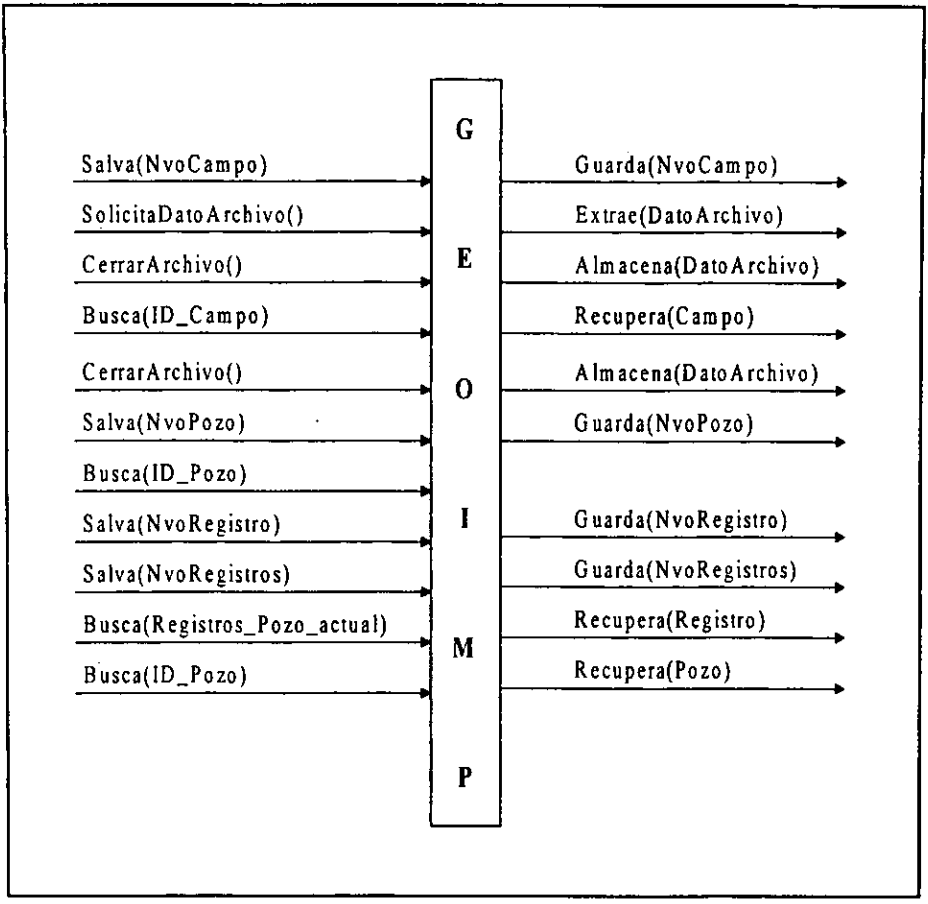



**Figura 6.10.- Pruebas de integración del elemento GEOIMP.**

### 6.1.5.3 Pruebas de sistema.

Las pruebas de sistema, se fundamenta básicamente en hilos de prueba y puede ser definido como:

- Una secuencia de instalación de máquina.
- Una secuencia de instalación de fuente.
- Un escenario de uso normal.
- Un caso de prueba de nivel de sistema.
- El ambiente que resulta desde una secuencia de entradas a nivel de sistema.

Después de instalar el sistema en el equipo antes especificado se hicieron pruebas de éste, que consistió en comparar los resultados arrojados con los esperados, para ello se hicieron pruebas desde el menú principal activando los diferentes eventos de éste, con lo que se constató que el sistema funciona adecuadamente. En seguida se enuncian algunas de estas pruebas.

Se intentó abrir un archivo *.txt que no contenía registros y el sistema respondió debidamente no trató de insertar la información para no sufrir alguna alteración o mal funcionamiento, así como envió un mensaje de error, por lo tanto la validación del formato del archivo *.txt se lleva a cabo de manera correcta garantizando la integridad de la información. Ver Figura 6.11.

De la misma forma al intentar abrir un archivo *.imp se verifica que éste sea realmente de éste tipo, o al querer crear uno nuevo con un nombre que ya existe se le manda al usuario un mensaje indicándole el error.

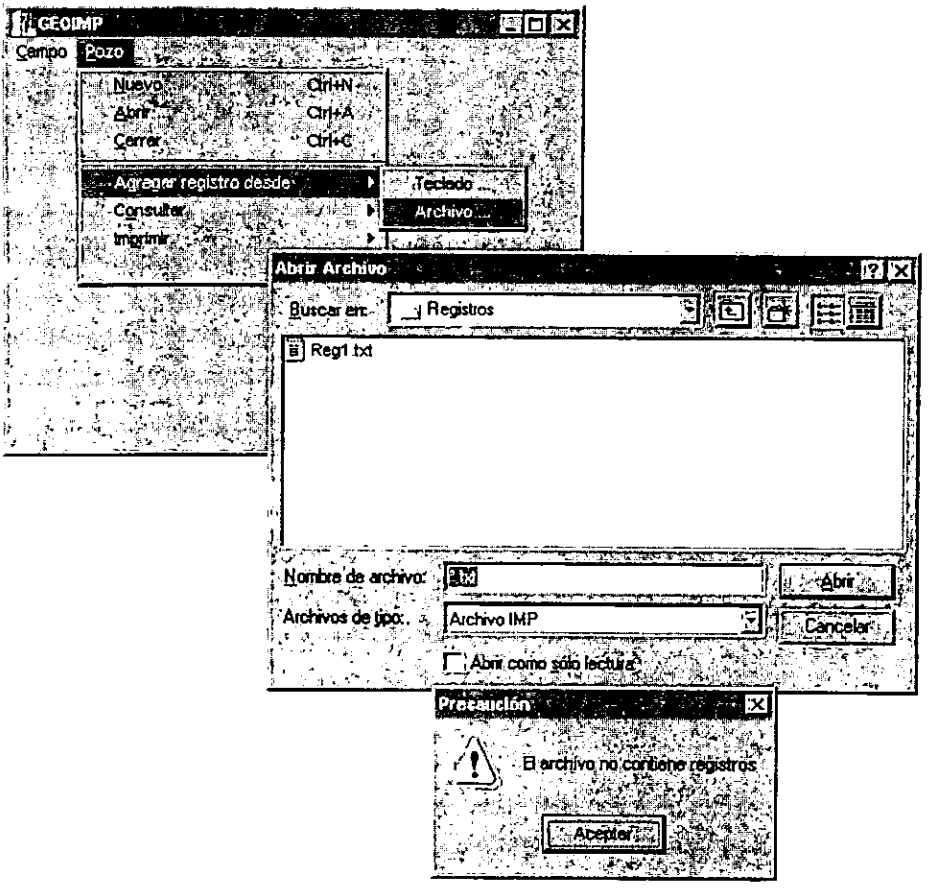



**Figura 6.11.- Manejo de error en un archivo *.txt.**

Se intentó recuperar un objeto tipo Well_Header que no existía en el archivo y el sistema respondió adecuadamente no presentó ninguna alteración o mal funcionamiento, además envió un mensaje de error, por lo tanto la validación de la persistencia de objetos en la fases de recuperación es de forma adecuada.

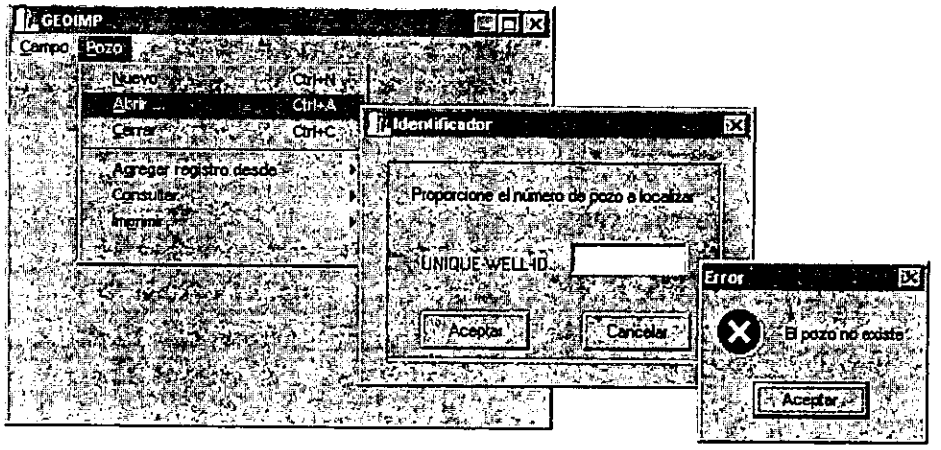



**Figura 6.12.- Manejo de error al abrir un pozo.**

Al agregar nuevos registros a un determinado pozo que anteriormente tenía registros, se logró comprobar que verdaderamente se conservan los anteriores registros, y además no se pierde la relación que hay entre ellos, es decir, que cada pozo conserva los apuntadores a sus registros, con esto se comprueba una vez más la persistencia de los objetos.

Se puede pensar que en el archivo sólo se guarda la dirección en memoria RAM del objeto, y así se le puede recuperar sin perder la integridad del mismo, esto no es verdad y se comprobó al crear un archivo e introducir nuevos objetos , se apagó el equipo y posteriormente se inicializó el sistema se verificó que los objetos se conservaban.

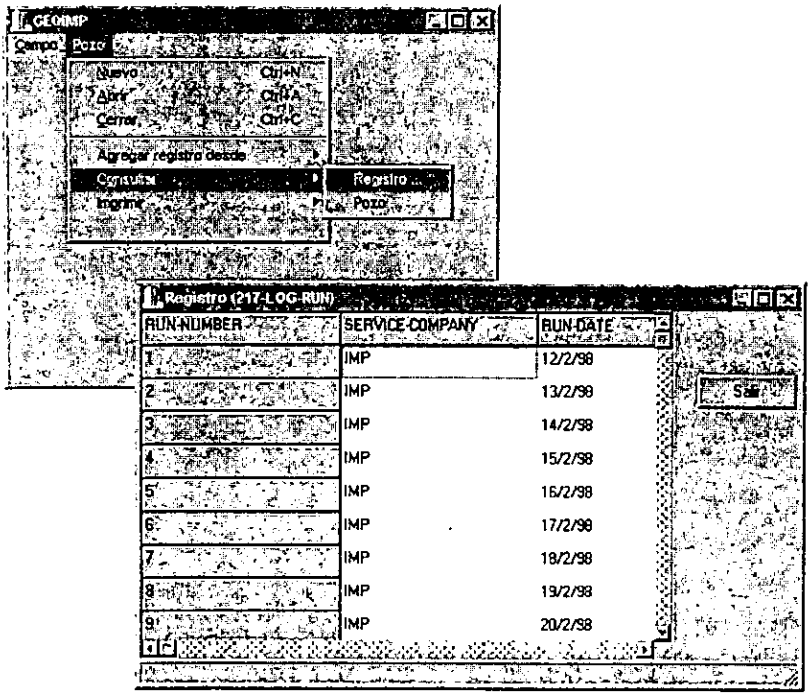



**Figura 6.13.- Consulta de los registros de un pozo.**

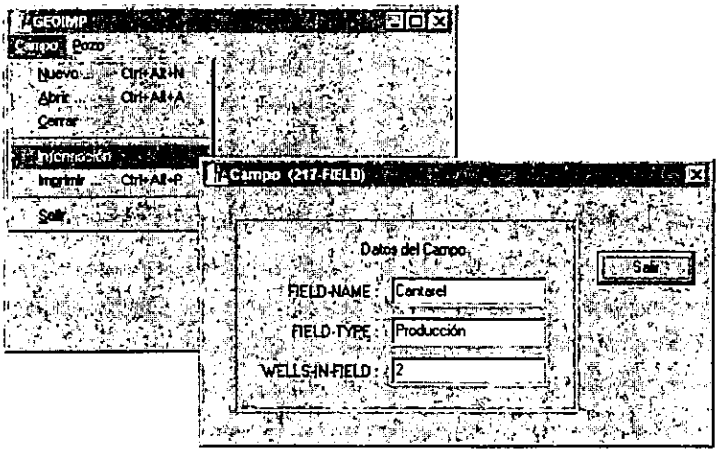



**Figura 6.14.- Consulta de un campo.**

Al crear un nuevo objeto se genera un identificador único para éste, y se conserva la secuencia aún después de que se cierre y se vuelva abrir el archivo, además el sistema no permite que el usuario pueda alterar los identificadores, con esto se logra tener un control total sobre todos los objetos.

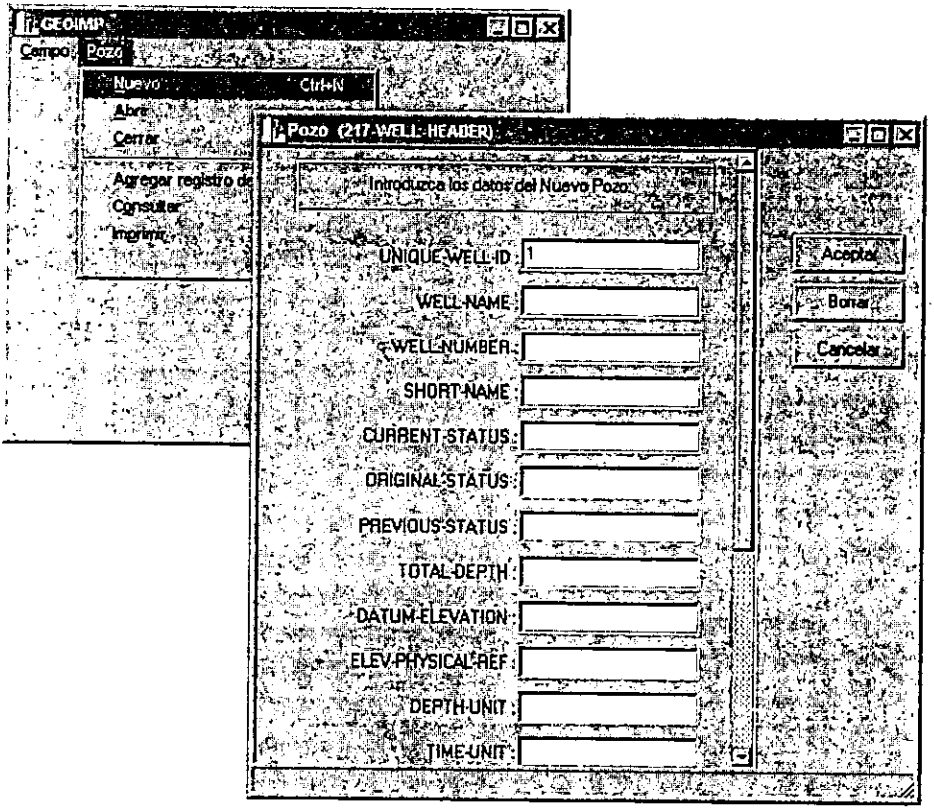



**Figura 6.15.- Genera un ID para cada objeto.**

# 7. CONCLUSIONES.

Al finalizar el presente trabajo se obtuvo un sistema prototipo que aplica persistencia a los objetos que integran la información petrolera. Es relevante enfatizar que a lo largo del trabajo se manejó ampliamente el concepto de persistencia el cual es una característica de la TOO (Tecnología Orientada a Objetos), así como se cumplió con los objetivos estipulados al inicio del mismo.

Actualmente existe una tendencia hacia la TOO por distintas características que ofrece como son abstracción de datos, encapsulación, herencia, reutilización del código, semántica entendible, etc. que hacen más fácil el entendimiento, manejo y mantenimiento de la información a través de objetos; resultando ideal para el manejo de grandes volúmenes de información.

Los datos constituyen la parte más valiosa de una empresa, porque una vez procesados, proporcionan información que es fundamental para la toma de decisiones. Por lo tanto la mayoría de los sistemas de información almacenan sus datos de manera persistente, para que estos no se pierdan, teniendo la capacidad de escribirlos y leerlos.

Existen dos grandes arquitecturas que permiten hacer persistentes a los objetos:

- Bases de Datos.
- Archivos.

La elección depende de los requerimientos solicitados; por ejemplo, si se requiere de aplicaciones que necesitan control de concurrencia o soporte de arquitecturas heterogéneas, etc., se puede optar por una OODB (Base de Datos Orientada a Objetos); pero si sólo se requieren algunos recursos, la mejor opción es el manejo de archivos, para no incurrir en tiempo adicional y espacio extra; aunque el diseño de persistencia a través de un archivo así como la implementación no es una tarea trivial.

Otro punto importante es el nivel de persistencia de objetos requerida (simple, isomórfica o polimórfica), debido a que ésta determina la complejidad de la forma de almacenamiento/recuperación.

Se debe tener presente que el manejo de datos u objetos en un lenguaje de programación no persistente, es distinta a la de un archivo, por lo cual se debe escribir código para transferir desde memoria en formato de un programa variable a un formato de un archivo y viceversa, provocando un gran esfuerzo por parte del programador.

A diferencia de otros modelos de programación, en el modelo de objetos, los datos representan información que pueden ser estructuras complejas con una conducta asociada.

## BIBLIOGRAFÍA.


1 *Principios/Aplicaciones de la Interpretación de Registros Físicos.*
Shlumberger, 1991.

2 *Geoshare Data Model 8.0 Encoding Manual.*
Houston, Texas USA, Schlumberger, 1997.

3 *RP66: Digital Log Interchange Standard (DLIS), Versión 1.0.*
USA, API (American Petroleum Institute), 1991, 82 pp.

4 KHOSHAFIAN, Setrag y ABNOUS, Razmik.
*Object Orientation: Concepts, Language, Databases.*
USA, WILEY, 1990, 433 pp.

5 HENRY, Sallie y HUMPHREY, Matthew.
*"Object-Oriented vs. Procedural Programming Languages".*
Journal of Object-Oriented Programming, Vol. 6, No. 3 (Junio 1993) , p. 41-49.

6 KHAN, Emdad.
*"Object Oriented Programming for Structured Procedural Programmers".*
IEEE Computer, Vol. 28, No.10 (Octubre 1995), p. 48-57.

7 ATKINSON y BUNEMAN.
*"Types and Persistence in Database Programming Languages".*
Computing Survey, Vol. 19, No. 2 (Junio 87), p. 105-190.

8 HUGHES, John.
*Object-Oriented Databases.*
Inglaterra, Prentice-Hall, 1991, 280 pp.

9 ATKINSON, Malcom.
*"Types and Persistence in Database Programming Languages".*
ACM Computing Survey, Vol. 19, No. 2 (Junio 1987), p. 105-189.

10 LOOMIS, M.E.S.
*"Making Objects Persistent".*
Journal of Object-Oriented Programming, Vol. 6, No. 6 (Octubre 1993) , p. 25-28.

11 Http://www.cartelon.ca/CCS/Software.

12 SHILLING, John.
*"How to Roll Your Own Persistent Objects in C++".*
Journal of Object-Oriented Programming, Vol. 7, No. 4 (Julio 1994), p. 25-32.

26 CRAIG, Arnush.
*Aprendiendo Borland C++ 5.*
México, Pretice-Hall, 1997, 834 pp.

# GLOSARIO.

**Field (217-FIELD)** : Es un elemento de Geoshare, proporciona información relacionada a la región geográfica de un depósito de petróleo.

**Geoimp** : Es un archivo híbrido que resulta de la combinación al aplicar el formato Geoshare, formato RP66(DLIS) y persistencia. El cual tiene la extensión "IMP".

**Geoshare**: Es una formato estándar que sirve para transferir datos entre aplicaciones de la industria del petróleo y gas.

**Log-Run (217-LOG-RUN)** : En un elemento de Geoshare, describe la visita de una compañía de explotación realizada a un pozo.

**Persistencia** : Es la habilidad de los objetos a vivir más halla de la vida de los programas que los crearon.

**RP66 DLIS (Digital Log Interchange Standard)** : Es un formato estándar, que establece la manera en que los datos petroleros deben de ser almacenados en archivo.

**Well-Header (217-WELL-HEADER)** : Es un elemento de Geoshare, contiene la descripción de un pozo petrolero de una manera general.