Universidad de las Ciencias Informáticas

Facultad 3

Título: "Implementación de un mecanismo de serialización binaria de objetos en C++"

Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas

Autora: Yarimis Palacio López

Tutores: Ing. Yadian Guillermo Pérez Betancourt

Ing. Liset González Polanco

La Habana, Cuba

Junio, 2012

DECLARACIÓN DE AUTORÍA

Declaro ser la única autora de la presente tesis y autorizo a la Universidad de las Ciencias Informáticas a hacer uso de la misma en su beneficio.

Para que así conste firmo la presente a los ___ días del mes de _____ del año _____.

Autora:
Yarimis Palacio López

_____

Tutores:
Ing. Yadian Guillermo Pérez Betancourt

_____

Ing. Liset González Polanco

_____

Si buscas resultados distintos, no hagas siempre lo mismo…

Albert Einstein

AGRADECIMIENTOS

Primeramente doy gracias a dios por la salud que le ha brindado a mi familia, a mí y a todos ustedes, le doy gracias por permitirme estar presentando hoy lo que cambiará mi vida para siempre. Le agradezco a las únicas dos mujeres que he amado en mi vida, mi madre Onelia y mi hermana Tania, porque todo lo que he hecho ha sido por ellas y para ellas, a mi papá por estar a mi lado desde que nací hasta estos momentos, por nunca abandonar a mi madre porque lo más difícil que puede tener una hija es crecer alejada de sus padres. A Yadián que ha sido mi tutor, mi entrenador y sobre todo mi amigo, a su novia y también mi tutora Liset, por apoyarme en todo. A Yoidel por su ayuda incondicional en las pruebas, a Arian, a todas mis amistades que siempre me han apoyando en las pistas, en las clases, en los apartamentos. A mis hermanos Yankiel y Hanny por estar a mi lado en los buenos y malos momentos en la universidad, a Armando por apoyarme siempre, a mi hermano El Negro y mis sobrinos Andy, Yohandy y Yesly, a Yorlandis que su mayor sueño era verme graduada, a su abuela Ramona que la quiero como si fuera la mía. A mi difunta abuela Nené porque aunque está en el cielo se que está orgullosa de mi. A mis amigas y amigos de Venezuela Mileidis, Catherine, Osvaldo, Marvin y en especial a Henry. A mis compañeras de cuarto Dainelis, Diamelis, Mily e Ivian. A Abraham por apoyarme en estos últimos momentos. A mi tribunal por comprenderme y mi oponente Ana Marys por estar siempre presente, a mis entrenadores de atletismo Prisco y Lucia, a todas las personas que me ayudaron cuando comencé la universidad en especial a Tito, de verdad, a todos muchas gracias.

Yarimis

DEDICATORIA

Dedicado este trabajo a mi madre y mi hermana. Las quiero con todas las fuerzas de mi corazón y deseo que dios les de mucha vida para seguir amándolas siempre...

De su única negrita...

Yari

RESUMEN

La serialización de objetos es un mecanismo muy utilizado para la persistencia de datos en el desarrollo de software, se encarga de transformar el estado de los objetos a un formato que puede ser almacenado en forma de archivo, como un búfer de memoria, en bases de datos o enviados a una aplicación remota. La deserialización de objetos es el proceso inverso a la serialización, encargado de devolver los objetos serializados a su estado original, creando una copia exacta del objeto serializado. Los sistemas de almacenamiento de datos necesitan estos mecanismos para hacer persistente la información. Actualmente se desarrolla un sistema de almacenamiento de datos cuyo modelo lógico sean grafos e hipergrafos que puedan ser utilizados para el almacenamiento de información geográfica y relaciones entre usuarios. El sistema es desarrollado en C++ y este lenguaje no posee un mecanismo integrado de serialización que pueda ser utilizado, por lo que se necesita desarrollar un mecanismo que permita realizar estas funciones. La investigación se realiza con la perspectiva de profundizar en el estudio de la serialización y proporcionar un mecanismo de serialización de objetos en C++. Para el desarrollo del mecanismo se diseñan los diagramas correspondientes para efectuar la implementación validando el diseño mediante métricas de calidad. El mecanismo de serialización propuesto para el lenguaje C++ se valida mediante un caso de estudio.

PALABRAS CLAVES: C++, deserialización, mecanismo, serialización de objetos, sistema de almacenamiento de datos.

TABLA DE CONTENIDO

TABLA DE CONTENIDO

INTRODUCCIÓN

En los últimos años en el mundo se han desarrollado diversos sistemas que utilizan técnicas para aumentar la productividad y el control de procesos para la elaboración de software. En la fabricación de software, al igual que en la industria de desarrollo de software, se busca facilitar el trabajo mediante la construcción de componentes básicos que unidos dan como resultado el producto final.[1] Las máquinas se han sofisticado para transformar las actividades de desarrollo de software en un proceso automatizado.[2]

El envío de objetos y el acceso a datos es una de las técnicas más utilizadas en el mundo del software. Existen numerosas alternativas para la selección de un mecanismo de acceso a datos y su posterior manipulación. Su elección depende del contexto en el cuál serán aplicados, impactando de diferentes formas en la arquitectura del sistema. A fin de elegir el mecanismo apropiado es necesario contar con una visión general, objetiva y bien estructurada de las alternativas disponibles, una forma de lograrlo es disponer de una clasificación de los mecanismos de persistencia existentes. Se definen estos mecanismos como serialización de objetos, acceso directo a bases de datos, mapeadores objeto-relacional y generadores de código.

Muchas empresas en la actualidad utilizan el envío o transportación de sus datos, documentos u objetos por la red, para distribuirlos a varias aplicaciones o localizaciones. La serialización se encarga de transformar estos objetos para ser transferidos por un canal a otro sistema, ya sea internet, archivo plano, memoria o base de datos. La serialización es el proceso de convertir el estado de un objeto a un formato que se pueda almacenar o transportar.[3, 4] El modo de realizar el proceso viene dado por el protocolo de comunicación utilizado.[1] Este es un mecanismo ampliamente utilizado en la programación orientada a objetos.

Los patrones de desarrollo de sistemas de estos tiempos han impulsado la aparición de nuevas técnicas para mejorar el funcionamiento de los servicios de comunicación, de manera que se puedan transportar, guardar o enviar datos de una aplicación a otra de forma más rápida y segura.[5] La comunicación entre los componentes distribuidos por la red se

fundamenta mayormente en el envío de objetos y mensajes, lo que obliga a un proceso de serialización de su información.[6]

Cuando se tienen que almacenar objetos para luego utilizarlos en diferentes ambientes, la serialización es una actividad muy importante[7] que facilita el aplanamiento de los objetos a una secuencia de bytes.

Los sistemas de almacenamiento de datos necesitan mecanismos de serialización que faciliten el tratamiento de la información[8]. Estos sistemas se han adaptado a las exigencias del contexto actual, grandes volúmenes de datos son gestionados desde bases de datos con distintos modelos lógicos,[9] muestra de ellos son las innumerables investigaciones para mejorar su rendimiento y escalabilidad.[10,13]

La Universidad de las Ciencias Informáticas (UCI) está llamada a llevar la delantera en la informatización del país. Como parte del desarrollo de software en la universidad, y abogando por la soberanía tecnológica se pretende implementar un sistema de almacenamiento de datos, cuyo modelo lógico sea estructuras complejas como grafos e hipergrafos que puedan ser utilizados para el almacenamiento de información geográfica y relaciones entre usuarios. En dicho sistema, estas estructuras se almacenarán en forma de objetos, por lo que se hace necesario un mecanismo capaz de su serialización y deserialización.

Para el desarrollo de la aplicación se utiliza el lenguaje C++ por su potencia y eficiencia.[14] C++ sin embargo no provee soporte nativo para la serialización,[15] como consecuencia, no se cuenta con un mecanismo de serialización para este sistema, que permita que los objetos puedan ser convertidos a un flujo de bytes y luego reconstruidos. Esto trae consigo que los desarrolladores manejen los objetos a la hora de la serialización como estiman, perdiendo el control y exponiendo al software final a fallas sutiles. Dentro de las fallas pueden encontrase que no se interprete bien la secuencia de byte a la hora de la deserialización, creando en el mejor de los casos una copia del objeto que no se corresponde con el serializado.

Como resultado del análisis de la situación anterior se plantea como **problema a resolver:** ¿Cómo garantizar la serialización de objetos en un sistema de almacenamiento de datos

cuyos modelos lógicos sean estructuras complejas, de modo que los objetos puedan ser convertidos a un flujo de bytes y luego reconstruidos?

Se define como **objeto de estudio** el proceso de serialización de objetos y se enmarca el **campo de acción** en los mecanismos de serialización de objetos para sistemas de almacenamiento de datos.

Para darle solución al problema planteado anteriormente se define como **objetivo general:** Implementar un mecanismo de serialización de objetos en C++ para un sistema de almacenamiento de datos.

Se plantea como **idea a defender**: La implementación de un mecanismo que permita realizar la serialización de objetos en C++, admitirá que el sistema de almacenamiento de datos pueda convertir objetos a un flujo de bytes y luego reconstruirlos.

Para cumplir con el objetivo de la investigación y lograr una solución apropiada a la problemática existente, se plantean las siguientes **tareas de la investigación:**

1. Definición de conceptos asociados a la serialización.
2. Descripción de problemas tipos que existen asociados a la serialización de objetos.
3. Caracterización de algoritmos existentes para la serialización binaria.
4. Implementación de un mecanismo para la serialización y deserialización binaria de objetos en C++.
5. Validación de la solución propuesta.

Métodos científicos de la investigación:

Método Dialéctico: Posibilitó la búsqueda de argumentos existentes dentro de la serialización de objetos y sus relaciones externas, los cambios que se producen en el tema analizado y el paso hacia un nuevo objeto de estudio.

Métodos teóricos de la investigación:

Analítico-Sintético: Permitió analizar las relaciones y componentes de la investigación para lograr descubrir mediante el análisis, relaciones y características generales entre los elementos.

Modelación*:* Se utilizó para el modelado y construcción del mecanismo desarrollado.

Posibles resultados: Obtener la implementación de un mecanismo para realizar la serialización binaria de objetos en C++.

El presente trabajo de diploma está desglosado en los siguientes capítulos:

Capítulo 1: Fundamentación Teórica:

En el capítulo uno es realizado un análisis significativo sobre la serialización de objetos. Se investigan los principales conceptos relacionados con la serialización y deserialización de objetos, además de la situación existente con la serialización en el lenguaje de programación C++. Se realiza un estudio de algoritmos de serialización de objetos en otros lenguajes de programación que son utilizados actualmente. Por último se hace referencia a la metodología, lenguaje, herramienta de desarrollo y herramienta de modelado seleccionada a utilizar en la presente investigación.

Capítulo 2: Mecanismo de serialización:

En el capítulo dos se plantean los detalles relacionados con el diseño y la implementación del mecanismo que se propone, presentando para el diseño los diagramas de clases del diseño, diagrama de casos de uso del sistema, diagramas de secuencia del diseño, así como diagrama de componentes y diagrama de paquetes del sistema.

Capítulo 3: Validación de la solución propuesta:

El capítulo tres se enfoca en validar la solución de la investigación, desarrollando un caso de estudio para verificar la funcionalidad del mecanismo. Para validar el diseño se realiza un análisis de las métricas de calidad que proponen Lorenz y Kidd, además de la familia de métricas propuestas por Chidamber y Kemerer para encontrar las utilizadas en la investigación.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1 Introducción:

En el presente capítulo se abordan los principales conceptos y definiciones relacionados con la serialización y deserialización de objetos. Se presentan los tipos más importantes de serialización, las ventajas de realizar serialización y deserialización, las bibliotecas que realizan serialización de objetos para apoyar el funcionamiento del lenguaje C++, los lenguajes que soportan la serialización de forma directa. Además se define la metodología de desarrollo de software a utilizar, la herramienta y el lenguaje de modelado, el lenguaje de programación y el Entorno de Desarrollo Integrado utilizado para realizar la investigación.

1.2 Marco teórico:

En las ciencias de la computación la serialización es un proceso de codificación de objetos en un medio de almacenamiento, como puede ser un archivo o un búfer de memoria, con el fin de transmitirlo a través de una conexión en red como una serie de bytes[4, 16]. Muchos lenguajes de programación utilizan la serialización como una forma estándar para almacenar objetos en archivos. Este mecanismo permite almacenar en un archivo uno o varios objetos con todas sus referencias.[17] La serialización es empleada fundamentalmente como un mecanismo para el almacenamiento de los datos en la programación orientada a objetos y en la comunicación entre componentes.[18, 19]

El paradigma de la programación orientada a objetos (POO) está fundamentado en la utilización de objetos y sus interacciones para diseñar aplicaciones y programas informáticos. Utiliza la herencia, abstracción, polimorfismo y encapsulamiento.[20]

1.2.1 Concepto de serialización y deserialización:

La serialización consiste en codificar la información o el estado de un objeto o una colección de objetos a una serie de bytes, que puede ser almacenada en memoria, en disco o enviada a una aplicación remota. Para la recuperación de los bytes, el estado o información del objeto se utiliza la deserialización, que es el proceso inverso a la serialización.[21]

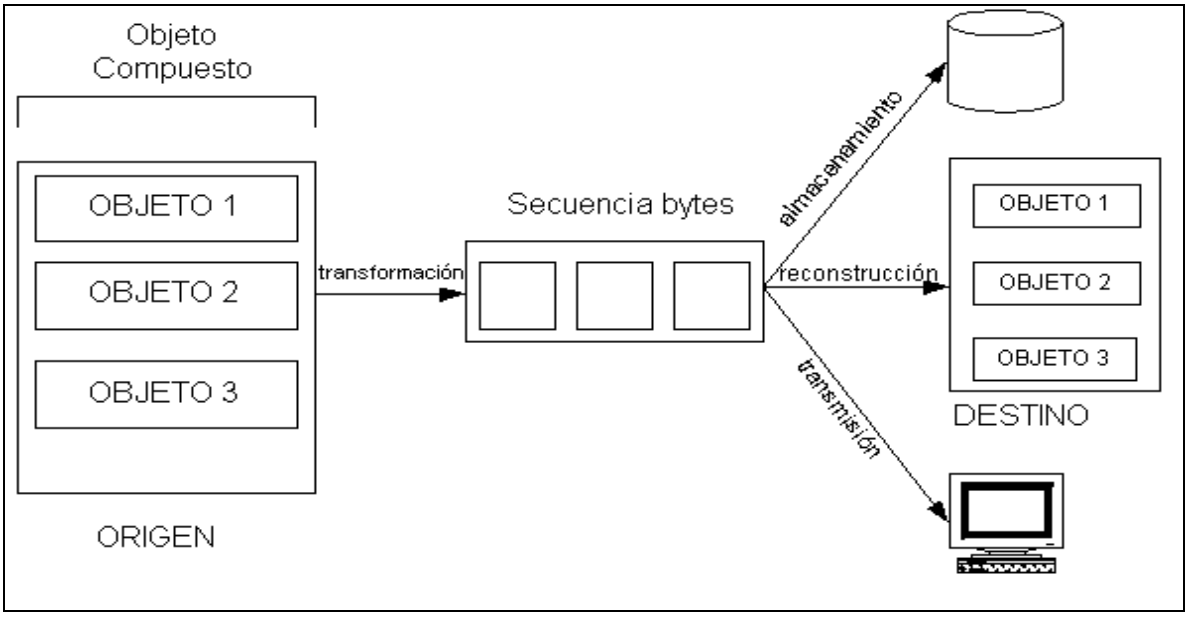

Figura 1: Proceso de serialización y deserialización de objetos. [21]

Para realizar la serialización de objetos se realiza un proceso de observación y etiquetado de atributos, de la misma manera para la deserialización se necesita efectuar un pre-procesado del mensaje, para así garantizar su estructura y la posterior reconstrucción del objeto.[6] Uno de los problemas existentes con la deserialización es que no siempre se puede efectuar, porque no todas las instancias de objetos son serializables, como por ejemplo el caso de las conexiones abiertas a bases de datos.[6]

1.2.2 Definición de serialización:

La serialización se puede definir como el proceso de convertir el estado de un objeto a un medio que se pueda almacenar o transportar por la red. Durante este proceso, los campos públicos y privados del objeto y el nombre de la clase, incluido el ensamblado que contiene la clase, se convierten a una secuencia de bytes que se escriben en una fuente de datos. Las dos razones más importantes para utilizar la serialización, es conservar el estado de un objeto a los medios de almacenamiento y enviar el valor del objeto de un dominio de aplicación a otro.[7]

El estado de un objeto está dado básicamente por el estado de sus campos y serializar un objeto radica en almacenar el estado de sus campos o convertirlos a una secuencia de

bytes. Si un objeto determinado a serializar tiene campos que además son objetos, se procede a serializar primero esos objetos. Este es un proceso recursivo que realiza la serialización de todo un árbol o grafo de objetos, almacenando la información referente a dicho árbol.[7]

1.2.3 Definición de deserialización:

El complemento de la serialización es la deserialización y es que al convertir un objeto a una secuencia de bytes, el proceso de deserialización permite la conversión del mismo a su estado original.[19] Este proceso crea una copia exacta del objeto serializado.[21]

1.2.4 Metodologías de serialización:

Para implementar la serialización de objetos se utilizan dos metodologías, la intrusiva y la no intrusiva.

Intrusiva se encarga de la modificación de la clase para añadir un método de plantilla de nombre fijo *serialize* que acepta un objeto así como el número de versión.

No intrusiva que se encarga de declarar una función de plantilla global que acepta un número, una instancia de la clase y el número de versión.[22]

En la presente investigación para implementar el mecanismo de serialización se utiliza la metodología intrusiva.

1.2.5 Tipos de serialización:

Existen principalmente dos tipos de serialización de objetos, la serialización binaria y la serialización XML (Extensible Markup Language). Para desarrollar un producto se escoge la más recomendada según la necesidad del sistema que se requiere.[23, 24]

1.2.5.1 Serialización binaria:

La serialización binaria toma el tipo de datos y lo convierte a una secuencia binaria, en este proceso los tipos de datos complejos (u objeto) son transformados a una secuencia binaria, que es un estado resistente y facilita su transportación. La serialización binaria conserva la información del tipo de objeto en el flujo de datos generados, esto significa que cuando el objeto se deserializa, el objeto creado es una copia exacta del original. La serialización binaria conserva la fidelidad de tipos, lo que resulta útil para conservar el estado de un

objeto entre distintas llamadas a una aplicación. La serialización binaria es muy recomendada para la comunicación entre componentes.[25]

1.2.5.2 Serialización XML:

XML: es el lenguaje de marcado de documentos basado en texto, que contiene los datos estructurados y extensibles. Este lenguaje puede leerse como un texto normal y debido a que es extensible se puede escribir cualquier tipo de información. Los documentos XML pueden contener texto, anotaciones, etc.[26, 27]

La serialización XML convierte el tipo de datos a secuencia XML que después se puede convertir a un documento XML para transmitirlo por la red, en este proceso se toman los tipos de datos complejos (u objetos) y se transforman a una secuencia XML. Este método solo serializa propiedades públicas y campos de datos, el sector privado no se guardará, por lo que la serialización XML no proporciona en muchos casos plena fidelidad con respecto al objeto original. Esto resulta útil cuando se desea proporcionar o consumir datos sin restringir la aplicación que utiliza los datos. Como XML es un estándar abierto, es una opción atractiva para compartir datos a través de la Web. Este tipo de serialización es ideal para cartografía rápida.[25]

1.2.6 Serialización binaria y serialización XML:

La serialización binaria permite la deserialización del objeto creando una copia exacta del mismo. Este método es el más rápido de la serialización ya que no contiene la sobrecarga de generar un documento XML durante el proceso de serialización. Generalmente la serialización binaria es más compacta que la XML, por lo que consume menos espacio de almacenamiento y puede transmitirse con rapidez. Permite serializar y restaurar todos los campos de un objeto ya sean públicos o no.[25]

1.2.7 Principales ventajas que aportan la serialización y la deserialización:

- Escalabilidad en la aplicación sin importar cuál sea la arquitectura física o lógica.
- Almacenar o enviar el estado de los objetos a cualquier plataforma o lenguaje.
- Utilización de protocolos simples para su envío o almacenamiento.
- Interoperabilidad entre aplicaciones capas o servidores.[4]

1.2.8 Bibliotecas que realizan serialización de objetos:

1.2.8.1 Serialización con Boost:

Boost es un conjunto de bibliotecas de software libre desarrolladas para incrementar las funcionalidades del lenguaje de programación C++. Estas bibliotecas son útiles en muchas aplicaciones. Su licencia permite utilizarse en cualquier tipo de proyecto, ya sean comerciales o no comerciales. Brinda serialización automática de estructuras de la Librería Estándar de Plantillas (STL). Ofrece versionado, o sea, cuando se realiza el proceso de serialización y se crean archivos sobre la base de este, se debe estar seguro que esos archivos sean compatibles con otras aplicaciones.
STL: es un conjunto determinado de estructuras de datos y algoritmos que integran la librería estándar C++, es decir conjunto de recursos que ya están implementados en este lenguaje y su elevado nivel de abstracción[22].

1.2.8.2 Serialización con Protocol Buffers:

Protocol Buffers es un lenguaje de plataforma objetiva, mecanismo eficiente y automatizado utilizado para la serialización de datos estructurados, se utiliza en protocolos de comunicación y almacenamiento de datos. La serialización de datos estructurados mediante Protocol Buffers es muy similar a la serialización con XML pero con características más avanzadas como su simplicidad de uso, su pequeño tamaño, su agilidad y la programación de sus clases de forma más sencilla. Este tiene una forma de codificar los datos estructurados en un formato eficaz pero extensivo. Google utiliza en la mayoría de sus protocolos internos y formatos de archivos los búferes de protocolos.[28]

1.2.9 Lenguajes de programación que soportan serialización:

Varios lenguajes de programación orientados a objetos soportan la serialización de forma directa, como es el caso de Python, Perl y Java.[29]

El lenguaje Python provee la serialización mediante módulos que facilitan esta tarea, como marshal, pickle, cPickle y shelve[30].

El lenguaje Perl permite serialización de código incluido dentro de un objeto, el código de las funciones de evaluación se almacena dentro de los objetos como un puntero a código y para serializarlo solo se llama a una función del módulo *B::Deparse* para obtener las fuentes.[31]

En Java se realiza la serialización mediante la implementación de la interfaz java.io.serializable, en caso contrario no se podrá hacer la serialización del objeto. La interfaz Serializable no tiene métodos, solo es usada para informar a la JVM (Java Virtual Machine o Máquina Virtual de Java) que un objeto va a ser serializado.[5]

1.2.10 Metodologías de Desarrollo de Software:

Las metodologías de desarrollo de software surgieron con el objetivo de dar solución a los problemas existentes en la producción de software. Estas engloban procedimientos, técnicas, documentación y herramientas que se utilizan en la creación de un producto de software.

Se pueden citar entre las metodologías más utilizadas Rational Unified Process (RUP)[32], Extreme Programing (XP)[33] y Microsoft Solution Framework (MSF)[34]:

RUP se adapta mejor a los proyectos de largo plazo, dividiendo el desarrollo de software en cuatro fases desarrolladas iterativamente y abarcando seis disciplinas ingenieriles y tres de apoyo[32].

XP por su parte se recomienda mayormente para el desarrollo de proyectos cortos y tiene como particularidad contar con el usuario final como parte del equipo.[33]

MSF se adapta por su parte a cualquier tipo de proyecto y se centra en los modelos de procesos y de equipo.[34]

Para el desarrollo del mecanismo se ha seleccionado la metodología RUP, debido a su aplicación con resultados positivos en proyectos de gran envergadura, es la utilizada para el desarrollo del sistema de almacenamiento de datos, aunque para este trabajo no se abarca toda la metodología porque el mecanismo propuesto es pequeño.

1.2.10.1 RUP

RUP constituye una de las metodologías más utilizadas en el desarrollo de aplicaciones orientadas a objetos. Es un proceso que se repite a lo largo de una serie de ciclos que constituyen la vida de un sistema. Cada ciclo concluye con una versión del producto para los

clientes y constan de cuatro fases. Cada fase termina con un hito. Como RUP es un proceso, en su modelación define como sus principales elementos:

Trabajadores (quién): Define el comportamiento y responsabilidades (rol) de un individuo, grupo de individuos, sistema automatizado o máquina, que trabajan en conjunto como un equipo. Ellos realizan las actividades y son propietarios de elementos.

Actividades (cómo): Es una tarea que tiene un propósito claro, es realizada por un trabajador y manipula elementos.

Artefactos (qué): Productos tangibles del proyecto que son producidos, modificados y usados por las actividades. Pueden ser modelos, elementos dentro del modelo, código fuente y ejecutables.

Flujo de actividades (cuándo): Secuencia de actividades realizadas por trabajadores y que produce un resultado de valor observable.

Su ciclo de vida se caracteriza por ser:

Dirigido por casos de uso: posibilitando que se refleje lo que los usuarios futuros necesitan y desean, captándose cuando se modela el negocio y se representa a través de los requisitos.

Centrado en la arquitectura: mostrando una visión común del sistema completo, en la que el equipo de proyecto y los usuarios deben estar de acuerdo, describiendo así los elementos del modelo que son más importantes para su construcción.

Iterativo e incremental: permitiendo detectar tempranamente desajustes e inconsistencias entre los requisitos, el diseño e implementación del sistema, de manera que el equipo de desarrollo se mantiene enfocado en producir resultados que satisfagan las necesidades del usuario.[32]

Por las razones antes mencionadas se decide utilizar esta metodología para el desarrollo del sistema.

1.2.11 Herramientas y lenguajes:

1.2.11.1 Notación Unified Modeling Language 2.0 (UML):

UML es un lenguaje, que permite modelar, analizar y diseñar sistemas orientados a objetos. Es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad.[20] Ofrece un estándar para describir un panorama del sistema modelo, incluyendo aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes de software reutilizables y aspectos conceptuales como los procesos de negocios y funciones del sistema. *"UML es un lenguaje para visualizar, especificar, construir y documentar los artefactos de un sistema que involucra una gran cantidad de software"*.[35]

UML está compuesto por diversos elementos gráficos que se combinan para conformar diagramas. La finalidad de los diagramas es presentar diversas perspectivas de un sistema, a las cuales se les conoce como modelo. Debido a que UML es un lenguaje, cuenta con reglas para combinar tales elementos y permite la modelación de sistemas con tecnología orientada a objetos.[35, 36]

1.2.11.2 Visual Paradigm for UML 8.0 Enterprise Edition:

Visual Paradigm es una herramienta que ofrece un entorno de creación de diagramas usando las notaciones UML 2.0, con un diseño centrado en casos de uso y enfocado además al negocio, lo cual genera un software de alta calidad. Esta herramienta fue creada para el ciclo completo de desarrollo del software que lo automatiza y acelera, permitiendo la captura de requisitos, análisis, diseño e implementación. También proporciona características tales como generación del código, ingeniería inversa y generación de informes. Permite escribir toda la especificación de un caso de uso sin necesidad de utilizar una herramienta externa como editor de texto, utilizando plantillas que se encuentran definidas, o que pueden ser creadas por los usuarios. Está diseñado para usuarios interesados en sistemas de software de gran escala, apoya los estándares más recientes de la notación UML. A pesar de todo es importante destacar que la licencia de Visual Paradigm es muy restringida.

Debido a las funcionalidades que brinda Visual Paradigm, al permitir crear diagramas usando las notaciones UML y ser una herramienta multiplataforma que se integra fácilmente con varios Entorno de Desarrollo Integrado (IDE), se decidió usar como herramienta durante el proceso de modelado.[37]

1.2.11.3 Lenguaje de programación C++:

El lenguaje de programación C++ es muy utilizado para el desarrollo de sistemas, permite trabajar tanto a alto como a bajo nivel. Añade utilidades específicas para la programación orientada a objetos y por su potencia y su eficiencia se ha clasificado como el lenguaje orientado a objetos más empelado en las aplicaciones profesionales. Por ser multiparadigma soporta la programación orientada a objetos, programación genérica y programación estructurada. Una particularidad de C++ es la posibilidad de redefinir los operadores o sea la sobrecarga de operadores, y poder crear nuevos tipos que se comporten como tipos fundamentales.[14, 15, 38, 39]

1.2.11.4 Entorno de Desarrollo Integrado NetBeans 7.1:

Es una plataforma para el desarrollo de aplicaciones de escritorio. Permite que las aplicaciones sean desarrolladas a partir de un conjunto de componentes de software llamados módulos. Un módulo es un archivo Java que contiene clases de Java escritas para interactuar con las APIs de NetBeans y un archivo especial (manifest file) que lo identifica como módulo. Las aplicaciones construidas a partir de módulos pueden ser extendidas agregándole nuevos módulos. Debido a que los módulos pueden ser desarrollados independientemente, las aplicaciones basadas en la plataforma NetBeans pueden ser extendidas fácilmente por otros desarrolladores de software. Es un proyecto de código abierto de gran éxito con gran base de usuarios y una comunidad en constante crecimiento. [40, 41]

Sun MicroSystems fundó el proyecto de código abierto NetBeans en junio del año 2000 y continúa siendo el patrocinador principal de los proyectos. El *IDE NetBeans* es una herramienta para programadores pensada para escribir, compilar, depurar y ejecutar programas. Está escrito en Java, pero se puede utilizar para cualquier otro lenguaje de programación. Hoy en día hay disponibles dos productos: el Entorno de Desarrollo Integrado

NetBeans y la Plataforma NetBeans. Es un producto libre y gratuito sin restricciones de uso.[40, 41]

1.3 Conclusiones del capítulo:

La serialización se fundamenta en el envío de objetos por la red, en su almacenamiento en bases de datos, en búfer de memoria o fichero. Existen dos tipos fundamentales de serialización, binaria y XML. La deserialización es el complemento de la serialización que se encarga de devolver el objeto serializado a su estado original. En la investigación se utiliza la metodología RUP, la herramienta para el modelado Visual Paradigm 8.0, UML 2.0 como lenguaje de modelado y el IDE de desarrollo el NetBeans 7.1 para la implementación del mecanismo.

Capítulo 2: MECANISMO DE SERIALIZACIÓN

2.1 Introducción:

En el presente capítulo se desarrolla el diseño y la implementación del mecanismo que se propone. Para lograr la solución de la investigación se diseñó el diagrama de casos de uso del sistema, los diagramas de clases del diseño, los diagramas de secuencia del diseño y el diagrama de componentes y diagrama de paquetes del sistema.

2.2 Diagrama de casos de uso:

En el siguiente diagrama se explica el conjunto de casos de uso que contiene el sistema, el actor, y su relación con los casos de uso.

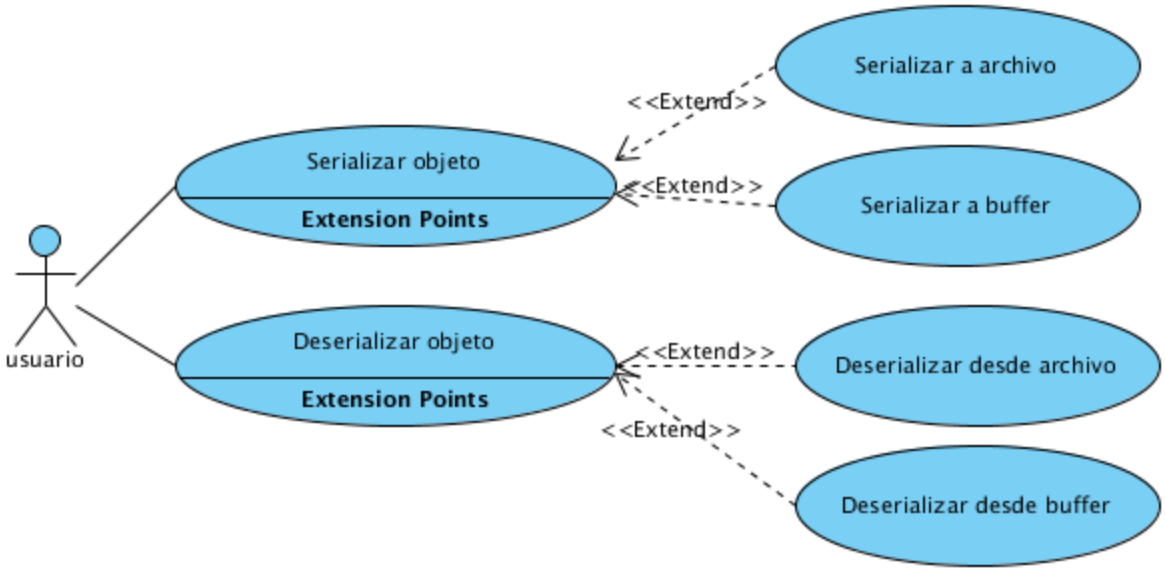

Figura 2: Diagrama de casos de uso del sistema.

2.3 Mecanismo de serialización:

A continuación se describe la propuesta del mecanismo de serialización y sus elementos. La solución facilita la implementación y aumenta el control de la serialización para disminuir las posibilidades de fallas sutiles en los sistemas que necesitan serializar objetos. Este mecanismo para serialización de objetos consiste en tomar todos los campos que son de tipos primitivos y transfórmalos a una secuencia de byte. Si un objeto determinado a

serializar tiene campos que además son objetos, se procede a serializar primero esos objetos. Este es un proceso recursivo que realiza la serialización de todo un árbol o grafo de objetos y como resultado se obtienen los bytes referente a dicho árbol.

2.3.1 Serialización:

Para llevar a cabo el proceso de serialización se definieron seis clases, la clase *DataOutput* que contiene todas las funcionalidades necesarias para serializar los tipos de datos simples y el *writeObject ()* para serializar los tipos de datos complejos, todos estos métodos son virtuales puros por lo que la clase es abstracta.

La clase *Serializable* tiene los métodos virtuales puros *writeObject ()* y *readObject ()* y de ellas deben heredar todas las clases para que puedan ser serializables. Además deben implementar estas dos funcionalidades.

La clase *DataOutputStream* hereda de la clase *DataOutput* e implementa todos los métodos para serializar utilizando el *writeBytes ()* que se define en la clase *DataOutput* y que no se implementa en esta clase.

Se cuenta con las clases *BufferOutputStream* y *FileOutputStream* que están por debajo de *DataOutputStream* en la jerarquía de la herencia. Estas clases son las encargadas de implementar el *writeBytes ()* para almacenar los bytes del objeto serializado a un búfer de memoria o a un fichero.

2.3.1.1 Diagrama de clases del diseño:

Los diagramas de clases muestran las definiciones de las clases que se implementarán, las relaciones entre ellas y los métodos que cada una debe definir. El siguiente diagrama de clases representa el proceso de serialización de objetos.

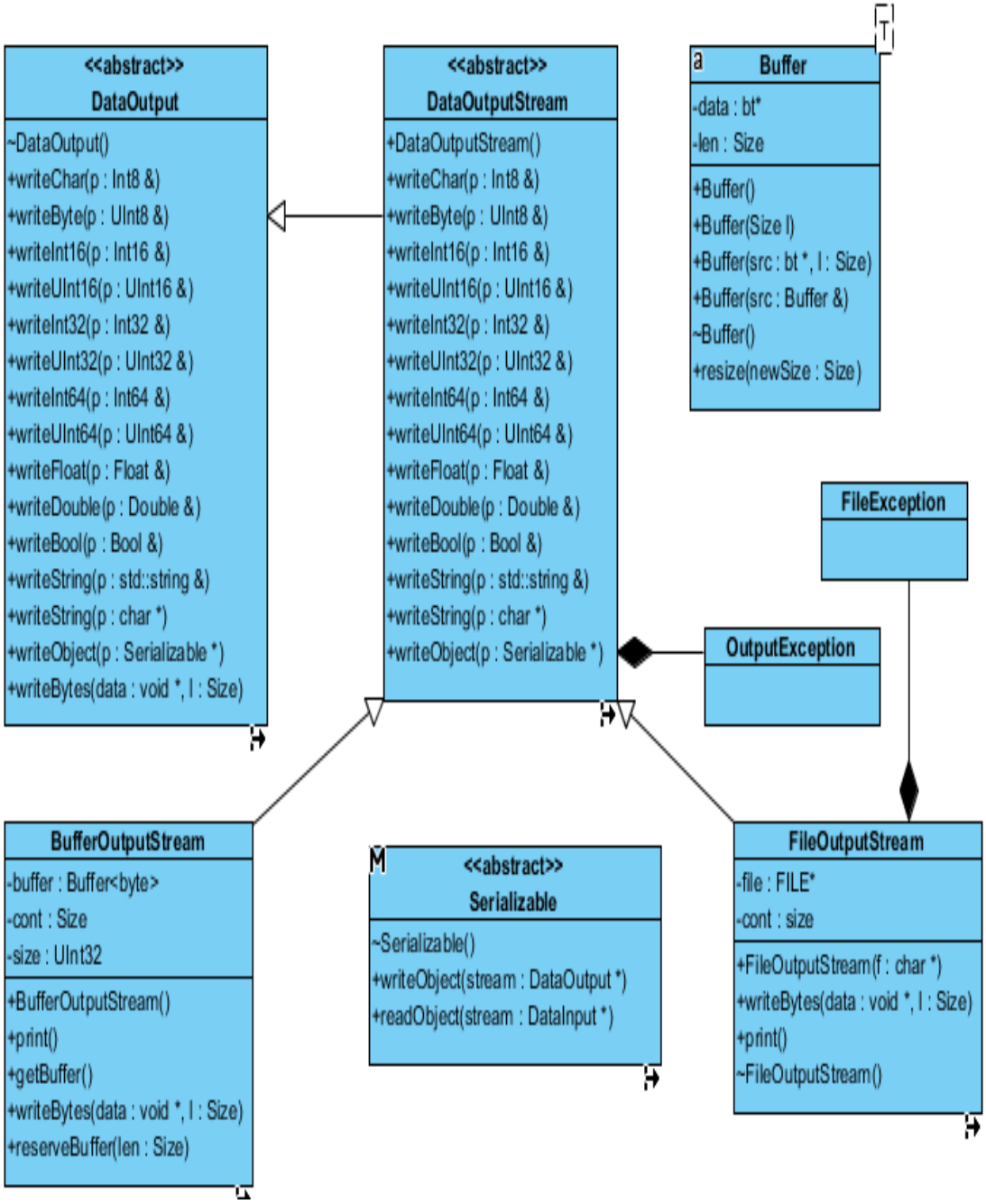

Figura 3: Diagrama de clases del diseño para el proceso de serialización de objetos.

En la siguiente tabla se muestra la descripción del diagrama de clases del proceso de serialización.

1.Nombre de la clase	Descripción de la clase
DataOutput	Esta clase contiene todos los métodos para serializar los tipos de datos simples y el método *writeObject ()* para los objetos complejos, todos sus métodos son virtuales puros, lo que la hace una clase abstracta. Define el método *writeBytes ()* que será implementado en las clases *FileOutputStream* y *BufferOutputStream*.
Nombre del método	**Descripción del método**
writeChar()	Se encarga de escribir un carácter.
writeByte()	Se encarga de escribir un byte.
writeInt16()	Se encarga de escribir un entero de 16 bits.
writeUInt16()	Se encarga de escribir un entero de 16 bits pero sin signo.
writeInt32()	Se encarga de escribir un entero de 32 bits.
writeUInt32()	Se encarga de escribir un entero de 32 bits pero sin signo.
writeInt64()	Se encarga de escribir un entero de 64 bits.
writeUInt64()	Se encarga de escribir un entero de 64 bits pero sin signo.
writeFloat()	Se encarga de escribir un número con comas.
writeDouble()	Se encarga de escribir un número con comas pero de 64 bits.
writeBool()	Se encarga de escribir un booleano.
writeString()	Se encarga de escribir un caracter o una cadena de caracteres.

writeObject()	Se encarga de escribir un objeto.
writeBytes()	Se encarga de serializar objetos compuestos. Se define en esta clase pero se implementa en las clases *FileOutputStream* y *BufferOutputStream*.
2.Nombre de la clase	**Descripción de la clase**
DataOutputStream	Esta clase hereda de la clase *DataOutput* e implementa todos los métodos para serializar, utilizando el método *writeBytes ()* que se define en la clase padre.
3.Nombre de la clase	**Descripción de la clase**
BufferOutputStream	Esta clase hereda de la clase *DataOutputStream* y se encarga de implementar el *writeBytes ()* para almacenar los bytes del objeto en un búfer de memoria.
Nombre del atributo	**Descripción del atributo**
buffer	Se utiliza para almacenar los bytes.
cont	Es el último byte guardado en búfer.
size	Es el tamaño del búfer.
Nombre del método	**Descripción del método**
print ()	Se encarga de imprimir los bytes almacenados en el búfer.
writeBytes()	Se encarga de escribir los bytes a un búfer de memoria.
reserveBuffer ()	Se encarga de reservar más memoria en el búfer.
4.Nombre de la clase	**Descripción de la clase**

FileOutputStream	Esta clase hereda de la clase *DataOutputStream* y se encarga de implementar el *writeBytes ()* para almacenar los bytes del objeto serializado en un fichero.
Nombre del atributo	**Descripción del atributo**
file	Es el fichero donde se va a serializar.
cont	Es la cantidad de bytes que se han guardado en el archivo.
Nombre del método	**Descripción del método**
writeBytes ()	Se encarga de escribir los bytes a un fichero.
print ()	Se encarga de imprimir los bytes almacenados en el fichero.
5.Nombre de la clase	**Descripción de la clase**
Serializable	Esta clase contiene los métodos virtuales puros *writeObject ()* y *readObject (),* por lo que es una clase abstracta. Todas las clases deben heredar de ella para que puedan ser serializables.
Nombre del método	**Descripción del método**
writeObject ()	Se encarga de realizar la serialización o escritura de objeto.
readObject ()	Se encarga de realizar la deserialización o lectura de objeto.
6.Nombre de la clase	**Descripción de la clase**
Buffer	Esta clase se utiliza para almacenar los búferes.
Nombre del atributo	**Descripción del atributo**
data	Es el puntero a los datos que se almacenan.

len	Es el tamaño de los datos que está apuntando a data.
Nombre del método	**Descripción del método**
size ()	Devuelve el tamaño del bloque de memoria.
resize ()	Se encarga de darle más tamaño al búfer.
7.Nombre de la clase	**Descripción de la clase**
FileException	Es utilizada para manejar los errores cuando se realiza la serialización a un fichero.
8.Nombre de la clase	**Descripción de la clase**
OutputException	Se encarga del manejo de los errores con la salida a serializar

Tabla 1: Descripción del diagrama de clases del diseño para el proceso de serialización.

2.3.1.2 Diagrama de secuencia del diseño para el proceso de serialización:

Un diagrama de secuencia es un diagrama de interacción que destaca la ordenación temporal de los mensajes (mensaje simple, síncrono, asíncrono, y/o de retorno). Este tipo de diagrama se destaca por tener la línea de vida y el foco de control que representa el período de tiempo durante el cual un objeto ejecuta una acción. La siguiente figura representa el diagrama de secuencia del proceso de serialización de objetos.

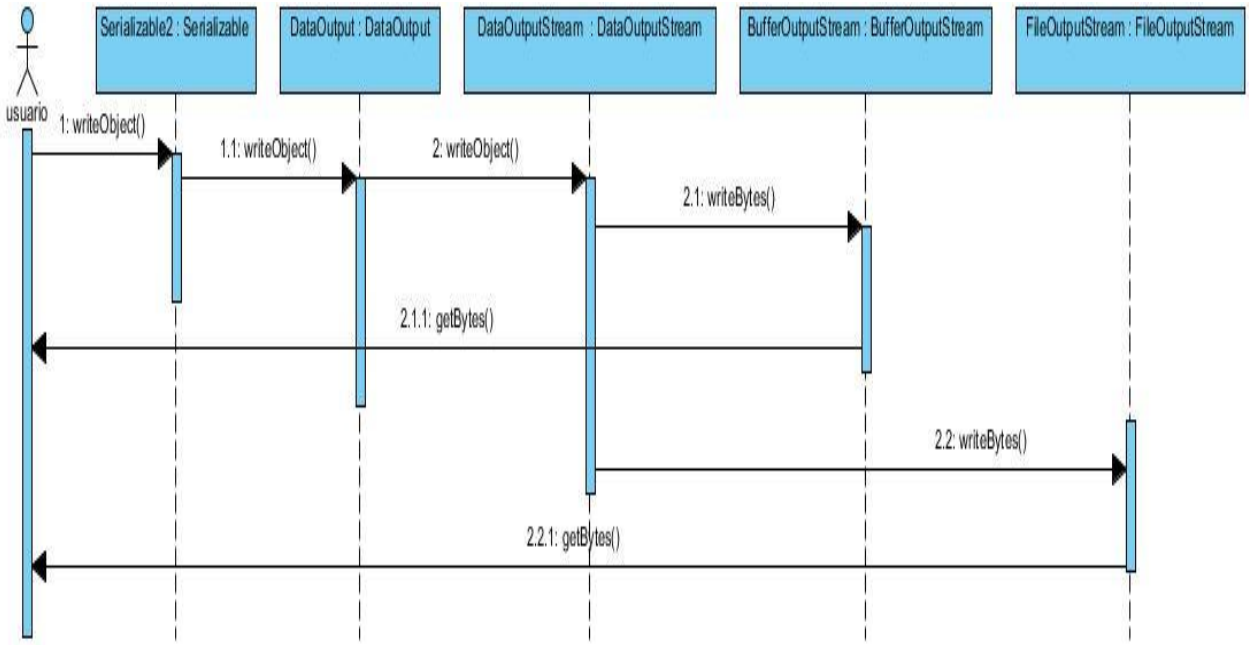

Figura 4: Diagrama de secuencia del diseño para el proceso de serialización de objetos.

2.3.2 Deserialización:

En el proceso de deserialización de objetos se utiliza la clase *DataInput* que define las funcionalidades necesarias para deserializar todos los tipos de datos simples y el método *readObject ()* para deserializar los tipos de datos complejos, estas funcionalidades son virtuales puras y la clase es abstracta.

La clase *DataInputStream* hereda de de la clase *DataInput* e implementa los métodos para leer los bytes del objeto utilizando el método *readBytes ()* que se implementa en sus clases hijas *FileInputStream* y *BufferInputStream*. La clase *FileInputStream* permite leer los bytes desde un fichero y *BufferInputStream* desde un búfer de memoria.

2.3.2.1. Diagrama de clases del diseño:

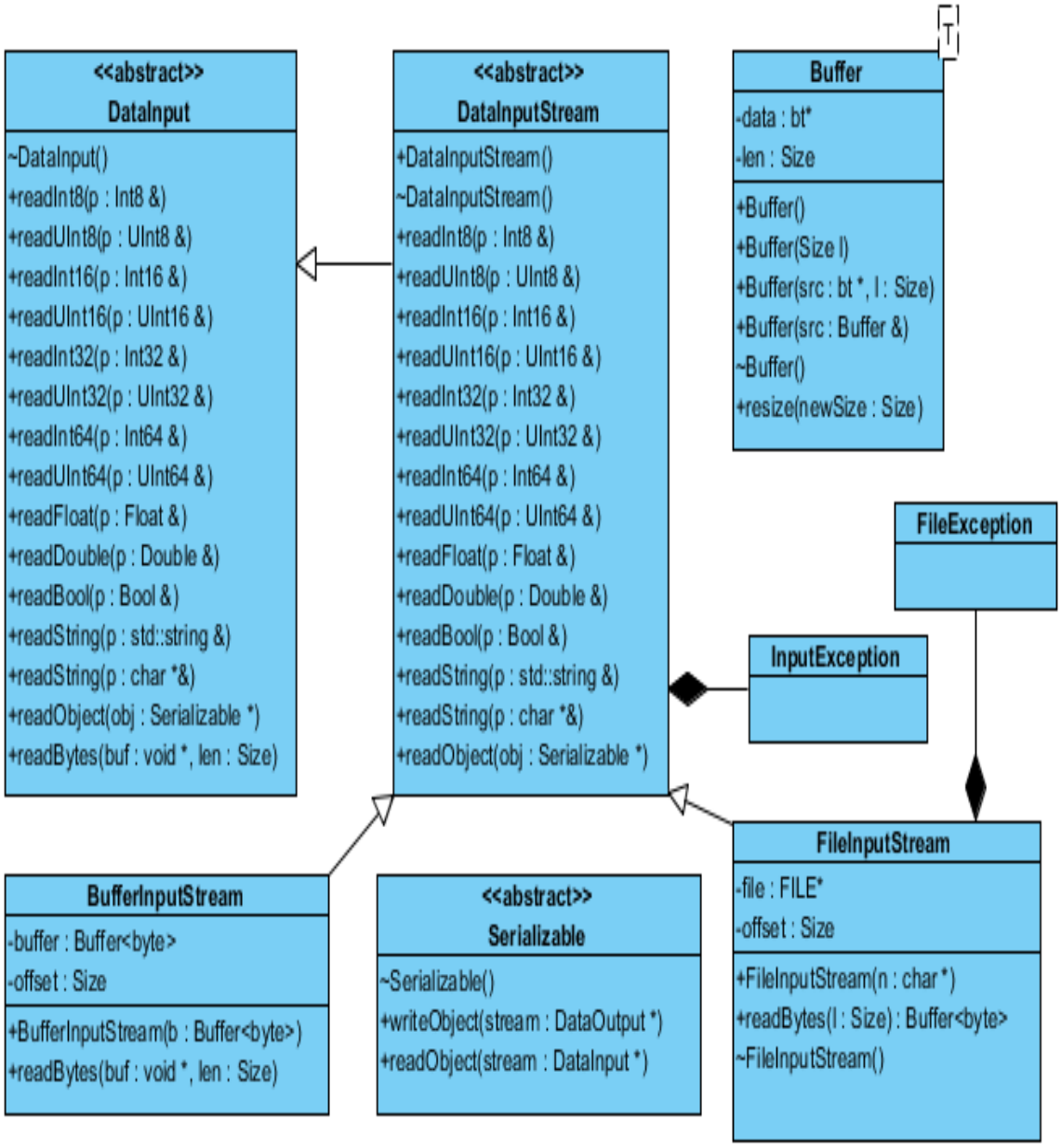

Figura 5: Diagrama de clases del diseño para el proceso de deserialización.

En la siguiente tabla se muestra la descripción del diagrama de clases del proceso de deserialización de objetos.

1.Nombre de la clase	Descripción de la clase
DataInput	Esta clase contiene todos los métodos para deserializar los tipos de datos simples y el método *readObject ()* para deserializar los tipos de datos complejos. Además define el método *readBytes ().* Todos sus métodos son virtuales puros lo que la hace una clase abstracta.
Nombre del método	**Descripción del método**
readInt8()	Se encarga de leer un entero de 8 bits.
readUInt8()	Se encarga de leer un entero de 8 bits pero sin signo.
readInt16()	Se encarga de leer un entero de 16 bits.
readUInt16()	Se encarga de leer un entero de 16 bits pero sin signo.
readInt32()	Se encarga de leer un entero de 32 bits.
readUInt32()	Se encarga de leer un entero de 32 bits pero sin signo.
readInt64()	Se encarga de leer un entero de 64 bits.
readUInt64()	Se encarga de leer un entero de 64 bits pero sin signo.
readFloat()	Se encarga de leer un número con comas.
readDouble()	Se encarga de leer un número con comas pero de 64 bits.
readBool()	Se encarga de leer un booleano.
readString()	Se encarga de leer un caracter o una cadena de caracteres.
readObject()	Es el método del proceso de deserialización. Se encarga de leer el objeto a deserializar.

readBytes()	Se encarga de leer los bytes. Se define en esta clase pero se implementa en las clases *FileInputStream* y *BufferInputStream*.
2.Nombre de la clase	**Descripción de la clase**
DataInputStream	Hereda de la clase *DataInput* e implementa todos los métodos para leer los bytes, utilizando el método *writeBytes ()* que se define en la clase padre y se implementa en las clases *FileInputStream* y *BufferInputStream*.
3.Nombre de la clase	**Descripción de la clase**
FileInputStream	Permite leer los bytes desde un fichero.
Nombre del atributo	**Descripción del atributo**
file	Es el fichero que se va a deserializar.
offset	Es el último bytes leído.
Nombre del método	**Descripción del método**
readBytes()	Se encarga de leer los bytes desde un fichero.
4.Nombre de la clase	**Descripción de la clase**
BufferInputStream	Esta clase se encarga de leer los bytes desde un búfer de memoria.
Nombre del atributo	**Descripción del atributo**
buffer	Es donde están almacenados los bytes.
offset	Es el último byte leído.

Nombre del método	Descripción del método
readBytes()	Se encarga de leer los bytes desde un búfer de memoria.
5.Nombre de la clase	**Descripción de la clase**
InputException	Es utilizada para el manejo de los errores con la entrada a deserializar.

Tabla 2: Descripción del diagrama de clases del diseño para el proceso de deserialización.

2.3.2.2. Diagrama de secuencia del diseño para el proceso de deserialización:

El siguiente diagrama de secuencia muestra el proceso de deserialización objeto.

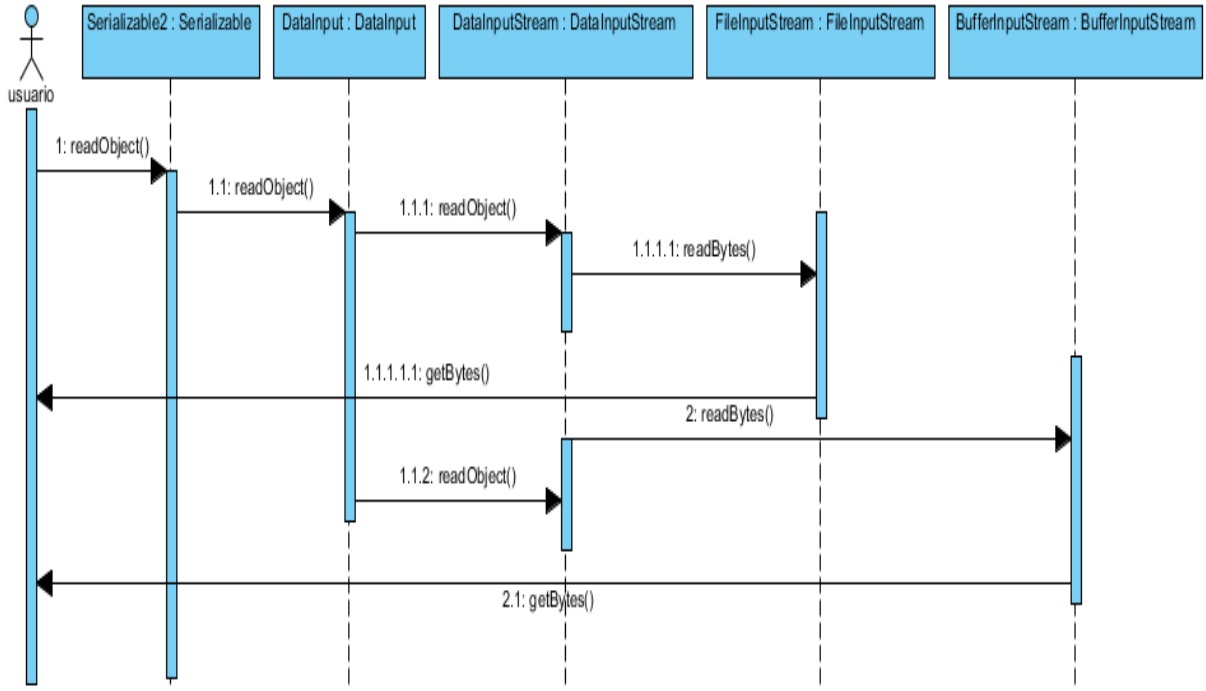

Figura 6: Diagrama de secuencia del diseño para el proceso de deserialización de objetos.

2.3.3 Diagrama de componentes del sistema:

Un diagrama de componentes representa cómo un sistema de software es dividido en componentes y las dependencias existentes entre ellos. Los componentes físicos incluyen archivos, cabeceras, bibliotecas compartidas, módulos, ejecutables o paquetes.[32]

En el siguiente diagrama se muestra la vista interna de los componentes y las clases que son utilizadas para la serialización y la deserialización.

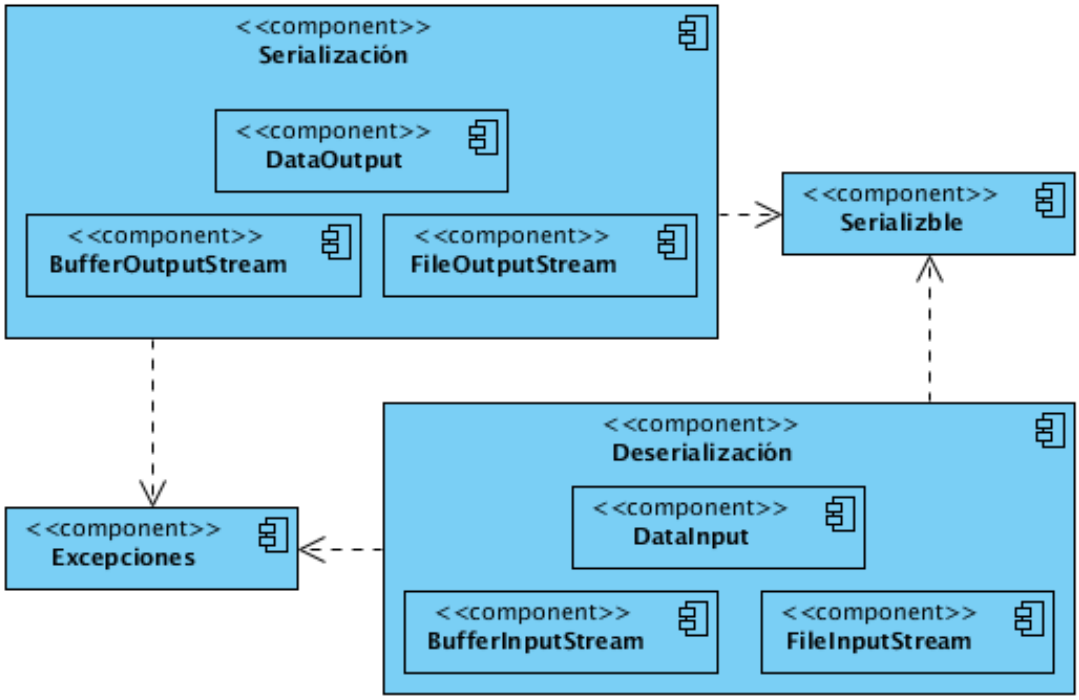

Figura 7: Diagrama de componentes del sistema.

2.3.4 Diagrama de paquetes del sistema:

La implementación del mecanismo propuesto está basada en tres paquetes, que representan agrupamientos lógicos de los elementos. La siguiente figura representa el diagrama de paquetes para la implementación.

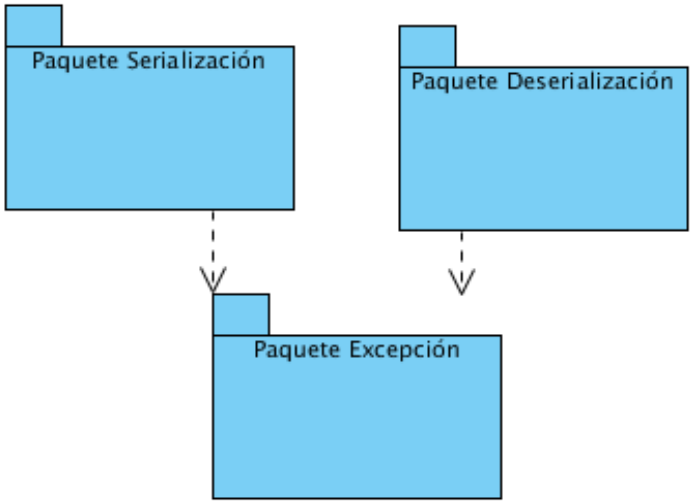

Figura 8: Diagrama de paquetes de la implementación.

El paquete **Excepción** almacena las clases que manejan las excepciones, en él se encuentra la clase *FileException* para manejar los errores cuando se serializa a un fichero o para deserializar desde él. *InputException* para el manejo de los errores con la entrada a deserializar, *OutputException* para el manejo de los errores con la salida a serializar, *SerializableException* para las excepciones relacionadas con objetos no serializables.

El paquete **Serialización** contiene todas las clases para efectuar el proceso de serialización.
El paquete **Deserialización** contiene todas las clases para efectuar la deserialización.

2.4 Patrones de diseño:

Cada patrón describe un problema que ocurre una y otra vez en nuestro ambiente, y luego describe el núcleo de la solución a ese problema, de tal manera que se puede usar esa solución un millón de veces más, sin hacer jamás la misma acción dos veces. En el diseño se hace uso de los patrones relacionados a continuación.[42]

2.4.1 Patrones GRASP:

Los patrones GRASP describen cuatro principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones.[42]

Experto: La aplicación de este patrón permite a cada clase desarrollar las tareas que pueden realizar según la información que poseen, se evidencia en las clases *BufferOutputStream*, *FileOutputStream* que implementan el método *writeBytes ()* y las clases *BufferInputStream*, *FileInputStream* que implementan el método *readBytes ()*.

Bajo acoplamiento: Este patrón soluciona el inconveniente de dar soporte a una dependencia escasa y a un aumento de la reutilización.

Alta cohesión: Este patrón es utilizado para mantener la complejidad dentro de los límites manejables. El diseño obtenido cumple con los patrones de Bajo acoplamiento y Alta cohesión permitiendo la colaboración entre los elementos del diseño, sin verse afectada la reutilización de los mismos y el entendimiento de estos cuando se encuentran aislados.

Indirección: Se utiliza para mejorar el bajo acoplamiento entre dos clases asignando la responsabilidad de la mediación entre ellas a una tercera, se evidencia en *FileOutputStream*, permitiendo la relación con la fuente donde se va a serializar, en este caso a un archivo. Se evita que el *DataOutputStream* tenga dependencia con las fuentes.

2.5 Conclusiones del capítulo:

En el presente capítulo se mostraron los artefactos generados durante el diseño. Se incluye una breve descripción de los mismos, con el objetivo de alcanzar perfectamente los requisitos del software; posibilitando la correcta transformación de los mismos a un diseño que indique cómo debe ser implementado el software. También se describieron los patrones utilizados para el diseño propuesto y se obtuvo la implementación del mecanismo de serialización y deserialización de objetos.

Capítulo 3: VALIDACIÓN DE LA SOLUCIÓN PROPUESTA

3.1 Introducción:

La medición es fundamental para cualquier disciplina de ingeniería, nos permite tener una visión más profunda, proporcionando un mecanismo para la evaluación objetiva.[32, 43] En el presente capítulo se aplica un conjunto de métricas de diseño orientado a objetos y se realiza un análisis de los resultados obtenidos para determinar la calidad del diseño y la implementación.

3.2 Métricas para el Modelo de Diseño:

Las métricas para el diseño proporcionan una mejor visión interna ayudando al diseño a evolucionar a un nivel superior de calidad. Para la aplicación de estas métricas se definen cinco características fundamentales.[43]

1. Localización.
2. Encapsulamiento.
3. Ocultamiento de la información.
4. Herencia.
5. Técnicas de abstracción de objetos.

3.2.1 Métricas orientadas a clases:

Una clase es la unidad primordial de todo sistema orientado a objeto. Por lo que las medidas y métricas para una clase individual, la jerarquía de clases y las colaboraciones de clases resultan sumamente valiosas para un ingeniero de software que necesite estimar la calidad del diseño [43]

3.2.1.1 Métricas propuestas por Lorenz y Kidd:

Las métricas de clases propuestas por Lorenz y Kidd[9] se dividen en cuatro categorías:

- **Tamaño**: Las métricas orientadas a tamaños se centran en el cálculo de atributos y operaciones para una clase individual. Además promedian los valores para el sistema en su totalidad.
- **Herencia**: Las métricas basadas en herencia se centran en la forma de reutilizar las operaciones a lo largo y ancho de la jerarquía de clases.

- **Valores internos:** Las métricas para valores internos de clases examinan la cohesión y todo lo relacionado con el código.
- **Valores externos:** Las métricas orientadas a valores externos examinan el acoplamiento y la reutilización.

Del análisis realizado a las métricas propuestas por Lorenz y Kidd se determinó la aplicación de dos al diseño.

Tamaño operacional de clases (TOC):

Para medir el tamaño operacional de las clases se deben tener en cuenta los siguientes aspectos:

- Total de operaciones, ya sean de la clase o las heredadas de las clases padres o interfaces que implementen.
- Cantidad de atributos, al igual que el anterior, tanto los de ella, como los de los padres.
- Promedio de operaciones y atributos para el sistema completo.[9]

Para evaluar las métricas son necesarios los valores de los umbrales. Las medidas o umbrales para los parámetros de calidad han sido una polémica a nivel mundial en el diseño de sistemas. En la tabla 3 se muestran algunos criterios de medida para estas métricas.[9, 43]

Número de operaciones y/o atributos	
TC	**Umbral**
Pequeño	**<=20**
Medio	**>20 y <=30**
Grande	**>30**

Tabla 3: Umbrales para TOC.

De la métrica TOC aplicada al diseño se obtuvo como resultado que para un total de 12 clases existentes en el diseño, se alcanzó un promedio de casi 1 atributo y 6 operaciones por clase como se muestra en la tabla 4.

Total de clases	Atributos	Operaciones
12	0.9%	6.1%

Tabla 4: Cantidad de clases del diseño, operaciones y atributos promediados.

En correspondencia con los umbrales propuestos en la tabla 3 se obtuvo que para un total de 12 clases que tiene el diseño, todas son de tamaño pequeño, como se muestra en la tabla 5, lo que representa el 100 % de clases pequeñas. Estos valores demuestran que los indicadores de calidad reutilización, implementación y responsabilidad no se ven afectados.

Cantidad de Clases	Umbral	Tamaño
12	<=20	Pequeño

Tabla 5: Cantidad de clases por tamaño.

Número de operaciones redefinidas por una subclase (NOR):

Para aplicar esta métrica es necesario conocer el número de clases que redefinen operaciones heredadas de otras clases. En la tabla 6 se muestran los valores obtenidos para el diseño que se propone.

Total de clases del diseño	Total de clases que redefinen operaciones
12	6

Tabla 6: Total de clases que redefinen operaciones.

Si se calcula el porciento que representa el total de clases que redefinen funciones, se obtiene como resultado que el 50 % de las clases existentes en el diseño redefinen operaciones.

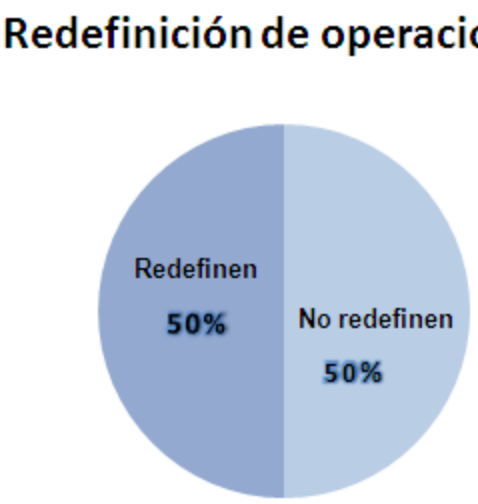

Figura 9. Total de clases que redefinen operaciones.

Se observa que los valores obtenidos para la aplicación de la métrica NOR al diseño propuesto es aceptable, esto trae consigo que no se afecte la calidad del diseño y pueda ser llevado a pruebas o modificado sin dificultad.

3.2.1.2 Familia de métricas propuestas por Chidamber & Kemerer:

El conjunto de métricas propuestas por Chidamber y Kemerer[44] han sido de las más referenciadas, son conocidas como la serie de métricas CK. Chidamber y Kemerer, ellos proponen seis métricas basadas en clases para sistemas orientados a objetos:

- Métodos ponderados por clase.
- Profundidad del árbol de herencia.
- Número de descendientes.
- Acoplamiento entre clase.
- Respuesta para una clase.
- Carencia de cohesión en los métodos.

Estas métricas permiten conocer hasta qué punto están bien definidas las clases y el sistema, lo cual tiene un impacto directo en la mantenibilidad del mismo, tanto por la comprensión de lo desarrollado, como por la dificultad de modificarlo con éxito. De estas seis se aplicaron dos al diseño propuesto.

Árbol de profundidad de herencia (APH):

Esta métrica está definida como la longitud máxima desde el nodo hasta la raíz del árbol. A medida que crece el APH, es más probable que las clases de niveles inferiores hereden muchos métodos. Esto da lugar a posibles dificultades cuando se intenta predecir el comportamiento de una clase. Una jerarquía de clases profunda (con un valor grande de APH) lleva también a una mayor complejidad de diseño. Por el lado positivo, los valores grandes del APH implican que se pueden reutilizar muchos métodos, lo que se debe considerar como un elemento a favor de la mantenibilidad[43]. En la tabla 7 se muestran los umbrales o medidas recomendados[43, 45], estos fueron los aplicados en el diseño de este sistema.

Nivel de jerarquía de clases	
APH	**Umbral**
Sencilla	**<=5**
Compleja	**>5**

Tabla 7. Umbral para la métrica APH.

Al aplicar esta métrica al diseño propuesto, según los umbrales definidos en la tabla 7, se obtiene como resultado que el APH presenta valor 2 como se muestra en la tabla 8, pues sólo hay presencia de herencia entre las clases y las clases abstractas de las cuales heredan sus funciones y las redefinen, lo cual está dentro del umbral definido para determinar que el diseño es sencillo, por lo que evita que exista algún problema si se desea predecir el comportamiento de una clase y no es difícil de dar mantenimiento.

Total de clases	APH
12	**2**

Tabla 8. Resultados aplicando la métrica APH.

Número de descendiente (NDD):

Las subclases que son inmediatamente subordinadas a una clase son denominadas descendientes. A medida que crece el número de descendientes, se incrementa la reutilización, pero también a medida que crece el NDD, la abstracción representada por la clase predecesora puede verse diluida y existe la posibilidad que algunos de los descendientes no sean realmente miembros propios de la clase predecesora.[43]

Número de clases	Número de descendientes
2	1
2	2

Tabla 9: Cantidad de clases por cantidad de descendientes.

Como se puede observar en la tabla 9, existen 2 clases con NDD igual a 1 y 2 con 2 descendientes de un total de 12 clases. Los valores de los NDD alcanzados para el diseño propuesto son pequeños lo que permite que cada descendiente sea realmente miembro de la clase predecesora y también se reduce la cantidad de pruebas necesarias para ejercitar cada uno de los miembros de la jerarquía.

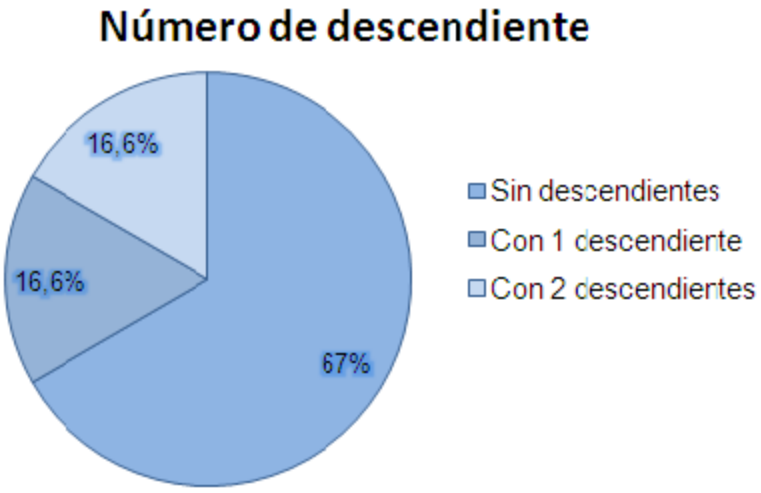

Figura 10. Número de descendientes.

Los resultados obtenidos a partir de la aplicación de las métricas de validación del diseño son aceptables.

3.3 Caso de estudio:

Se desarrolla un caso de estudio para verificar el funcionamiento del mecanismo propuesto. En el ejemplo que se muestra a continuación, se implementaron dos clases para mostrar el proceso de serialización y deserialización de objetos. La clase *Atleta* que hereda de la clase *Serializable* y la clase *Enfrentamiento* que también hereda de *Serializable*. La clase *Atleta* contiene 2 atributos, nombre y edad del atleta, e implementa los métodos *writeObject ()* y *readObject ()*, y la clase *Enfrentamiento* cuenta con 2 atributos de tipo *Atleta* y los métodos *writeObject ()* y *readObject ()* que deben implementar cada clase porque son definidos en la clase *Serializable*.

A continuación se presenta el diagrama de clases correspondiente al caso de estudio y la implementación de los métodos *writeObject ()* y *readObject ()* para cada clase.

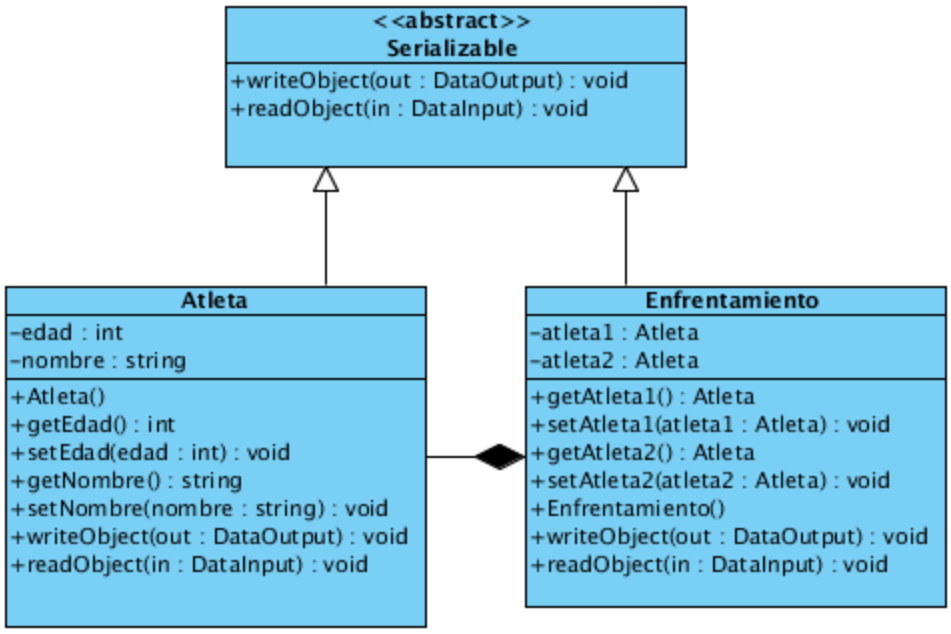

Figura 11: Diagrama de clases del caso de estudio.

Implementación de los métodos *writeObject ()* y *readObject ()* en la clase *Atleta*:

Serialización:

```
void writeObject (serialB::DataOutput * s) {
    s->writeInt32 (edad);
    s->writeString (nombre);
}
void readObject (serialB::DataInput* r) {
    r->readInt32 (edad);
    r->readString (nombre);
}
```

Implementación de los métodos *writeObject ()* y *readObject ()* en la clase *Enfrentamiento*:

```
void readObject (serialB::DataInput* r) {
    r->readObject (atleta1);
    r->readObject (atleta2);
}
void writeObject (serialB::DataOutput * s) {
    s->writeObject (atleta1);
    s->writeObject (atleta2);
}
```

Serialización a un flujo de byte:

Para serializar objetos a un flujo de bytes, se crea una instancia de la clase a serializar.

```
Enfrentamiento* pp = new Enfrentamiento ();
```

Crear la instancia a la fuente en la que se va a serializar.

Para el ejemplo está disponible la serialización a BufferOutputStream y FileOutputStream.

Instancia de BufferOutputStream para serializar a un buffer.

```
serialB::BufferOutputStream* s = new serialB::BufferOutputStream ();
```

Serializar pp en s

pp->writeObject(s);

Deserialización:

Almacenar en aux los bytes del objeto serializado.

aux = s->getBuffer ();

Instancia de BufferInputStream para deserializar, donde aux es el conjunto de bytes para deserializar.

*serialB::BufferInputStream * r = new serialB::BufferInputStream (aux);*

Deserializar d, desde r.

d->readObject (r);

Una vez corrido el ejemplo del proceso de serialización y deserialización de objetos se logra comprobar que la información referente a ellos es correcta, como se muestra en la siguiente figura.

```
<< Datos del objeto a serializar >>
 Nombre 1:  Pedro
 Edad 1:       28
 Nombre 2:  Yarimis
 Edad  2:       23

<< Bytes del objeto serializado >>
28 0 0 0 5 0 0 0 52 48 73 9 25 23 0 0 0 7 0 0 0 92 48 73 9 25 0 0

<< Datos del objeto deserializado >>
 Nombre 1:  Pedro
 Edad 1:       28
 Nombre 2:  Yarimis
 Edad  2:       23
```

Figura 12: Ejemplo del objeto serializado y deserializado.

3.4 Conclusiones parciales:

Se realizó la implementación del mecanismo de serialización de objetos y se obtuvo mediante la validación del caso de estudio que el mecanismo funciona correctamente. Como resultado de la evaluación del diseño propuesto utilizando métricas de validación de software se determina que la solución no tiene complejidad estructural alta y la calidad es aceptable, permitiendo que las pruebas no sean complejas, además presenta un bajo acoplamiento pues la profundidad de los niveles de herencia está acorde con los umbrales. Se puede concluir que el diseño y la implementación poseen una calidad aceptable.

CONCLUSIONES

Una vez concluido el estudio del proceso de serialización de objetos, se propuso un mecanismo de serialización.

Para la realización del diseño se tuvo en cuenta algunos patrones para modelar el sistema, lo que permitió generar artefactos empleando buenas prácticas necesarias para desarrollar un software con mayor robustez. Entre los artefactos generados en el diseño se pueden mencionar, los diagramas de secuencia, diagramas de clases, el diagrama de componentes y de paquetes, imprescindibles para el buen desarrollo de una solución informática.

De forma general los resultados obtenidos a partir del diseño del mecanismo son positivos, tomando como referencia la aplicación de métodos y métricas para validar la solución. Se obtuvo la implementación de un mecanismo de serialización genérico para su utilización en un sistema de almacenamiento de datos y se probó mediante un caso de estudio. Este mecanismo facilita el trabajo a los desarrolladores, aunque no está totalmente automatizado por la carencia de la reflexión en el lenguaje C++.

RECOMENDACIONES

Se recomienda la implementación de un mecanismo de reflexión que permita a la solución propuesta la automatización completa del proceso de serialización y deserialización de objetos.

Adicionar la implementación de otras fuentes de datos para la serialización de objetos en ellas.

REFERENCIAS BIBLIOGRÁFICAS:

1. LEÓN NAJERA, F. *Persistencia de objetos en bases de datos relacionales.* Bogotá 2009,

2. PEREZ-SCHOFIELD, J.B.G. Zero: una máquina virtual persistente, orientada a objetos pura y basada en prototipos. En *Vigo.*

3. SHINTANI, M.;LEE, S.*, et al.* A serialization algorithm for mobile robots using mobile agents with distributed ant colony clustering. En *Proceedings of the 15th international conference on Knowledge-based and intelligent information and engineering systems - Volume Part I. Berlin, Heidelberg. 2011*. p. 260-270.

4. MARZOUK, S.;JEMAA, M.B.*, et al.* A serialization based approach for strong mobility of shared object. En *Proceedings of the 5th international symposium on Principles and practice of programming in Java. New York, NY, USA. 2007*. p. 237-242.

5. BREG, F. y POLYCHRONOPOULOS, C.D. Java virtual machine support for object serialization. En *Proceedings of the 2001 joint ACM-ISCOPE conference on Java Grande. New York, NY, USA. 2001*. p. 173-180.

6. FDEZ, J.M.S.;DEZA, M.C.*, et al.* APROXIMACIÓN AL DISEÑO DE APLICACIONES DE
GESTIÓN BASADA EN SERVICIOS WEB. En *IADIS. VIGO, ESPAÑA 2005*.

7. DIAS, M.;MARTINEZ PECK, M.*, et al.* Fuel: A Fast General Purpose Object Graph Serializer. *Software: Practice and Experience*, June 2012, nº Disponible en: http://hal.inria.fr/hal-00703574.

8. LAMB, C.;LANDIS, G.*, et al.* The ObjectStore database system. *Commun. ACM*, 1991, vol. 34, nº 10, p. 50-63. ISSN 0001-0782.

9. LORENZ, M. y KIDD, J. *Object-Oriented Software Metrics.* Englewood, NJ: Prentice-Hall, 1994,

10. KONSTANTINOU, I.;ANGELOU, E.*, et al.* TIRAMOLA: elastic nosql provisioning through a cloud management platform. En *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data. Scottsdale, Arizona, USA. 2012*. p. 725--728.

11. PLUGGE, E.;HAWKINS, T.*, et al. The Definitive Guide to MongoDB: The NoSQL Database for Cloud and Desktop Computing.* 1st ed. Berkely, CA, USA: Apress, 2010, ISBN 1430230517, 9781430230519.

12. POKORNY, J. NoSQL databases: a step to database scalability in web environment. En *Proceedings of the 13th International Conference on Information Integration and Web-based Applications and Services. Ho Chi Minh City, Vietnam. 2011*. p. 278--283.

13. STEIERT, H.-P. Data Managment for Engineering Applications. *Computer Science,*, 2005, vol. 3551/2005, nº 139, [Consultado el: 25/02/2012]. Disponible en: http://www.springerlink.com/content/tedpyr3l0xh68vy6/.

14. DEWHURST, S.C. y STARK, K.T. *Programming in C++.* Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1989, ISBN 0-13-723156-3.

15. MANSFIELD, J., KENNETH C. y ANTONAKOS, J.L. *An introduction to programming using C++.* Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1997, ISBN 0-13-254921-2.

16. ALVORNOZ, J. [Consultado el: 14 de Noviembre de 2011 Disponible en: http://msdn.microsoft.com/es-es/library/2f7k4746.aspx.

17. BRACY, A. y ROTH, A. Serialization-Aware Mini-Graphs: Performance with Fewer Resources. En *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture. Washington, DC, USA. 2006*. p. 171-184.

18. ALLEN, M.D.;SRIDHARAN, S.*, et al.* Serialization sets: a dynamic dependence-based parallel execution model. *SIGPLAN Not.*, February 2009, vol. 44, nº 4, p. 85-96. Disponible en: http://doi.acm.org/10.1145/1594835.1504190. ISSN 0362-1340.

19. KILLIJIAN, M.-O.;RUIZ, J.-C.*, et al.* Portable serialization of CORBA objects: a reflective approach. *SIGPLAN Not.*, November 2002, vol. 37, nº 11, p. 68-82. Disponible en: http://doi.acm.org/10.1145/583854.582428. ISSN 0362-1340.

20. LARMAN, C. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development (3rd Edition).* Upper Saddle River, NJ, USA: Prentice Hall PTR, 2004, ISBN 0131489062.

21. VÁZQUEZ, C.C. *Desarrollo de un mecanismo de serialización en J2ME.* Madrid, España

22. GROUP, K. *Serialización con Boost* [Consultado el: 25/10/2011 Disponible en: htpp://www.boost.org/doc/libs/serialization/doc/implementation.html.

23. HERICKO, M.;JURIC, M.B.*, et al.* Object serialization analysis and comparison in Java and .NET. *SIGPLAN Not.*, August 2003, vol. 38, nº 8, p. 44-54. Disponible en: http://doi.acm.org/10.1145/944579.944589. ISSN 0362-1340.

24. IMRE, G.;KASZÓ, M.*, et al.* A Novel Cost Model of XML Serialization. *Electron. Notes Theor. Comput. Sci.*, February 2010, vol. 261, nº p. 147-162. [Consultado el:

12/03/2012]. Disponible en: http://dx.doi.org/10.1016/j.entcs.2010.01.010. ISSN 1571-0661.

25. SUMARAY, A. y MAKKI, S.K. A comparison of data serialization formats for optimal efficiency on a mobile platform. En *Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication. New York, NY, USA. 2012*. p. 48:1-48:6.

26. HAROLD, E.R. y MEANS, W.S. *XML in a Nutshell.* Third ed. O'Reilly Media, Inc., 2004, ISBN 0596007647.

27. TATARINOV, I.;VIGLAS, S.D.*, et al.* Storing and querying ordered XML using a relational database system. En *Proceedings of the 2002 ACM SIGMOD international conference on Management of data. Madison, Wisconsin.* p. 204--215.

28. DEAN, J. y GHEMAWAT, S. MapReduce: a flexible data processing tool. *Commun. ACM*, 2010, vol. 53, nº January 2010, p. 72--77. [Consultado el: 25/02/2012]. Disponible en: http://doi.acm.org/10.1145/1629175.1629198. ISSN 0001-0782.

29. DENARO, G. y MARIANI, L. Towards Testing and Analysis of Systems that Use Serialization. *Electron. Notes Theor. Comput. Sci.*, January 2005, vol. 116, nº p. 171-184. Disponible en: http://dx.doi.org/10.1016/j.entcs.2004.02.075. ISSN 1571-0661.

30. DUQUE, R.G. Python para todos. nº

31. GUERVÓS, J.J.M. *Algoritmos evolutivos en Perl* 2002,

32. JACOBSON, I.;BOOCH, G.*, et al. El Proceso Unificado de Desarrollo de Software.* Madrid: Pearson Educación. S.A., 2000, ISBN 84-7829-036-2.

33. MARTIN, R.C. *Agile Software Development: Principles, Patterns, and Practices.* Upper Saddle River, NJ, USA: Prentice Hall PTR, 2003, ISBN 0135974445.

34. TURNER, M.S.V. *Microsoft Solution Framework Essentials: Building Successful Technology Solutions.* O'Reilly, 2009,

35. JACOBSON, I.;BOOCH, G.*, et al. El lenguaje unificado de modelado. Manual de referencia.* Addison Wesley, 1999,

36. LOVELLE, S.P.;VALDÉS, F.O.*, et al.* AUTOMATIZACIÓN DE LA ARQUITECTURA DE COMPONENTES GENÉRICOS USANDO UML. *Ingeniería Industrial*, 2006, vol. 27, nº 1, Disponible en: http://www.doaj.org/doaj?func=fulltext&passMe=http://rii.cujae.edu.cu/index.php/revistaind/article/view/96. ISSN 02585960.

37. *Boost Productivity with Innovative and Intuitive Technologies* [Consultado el: 25/09/2011 Disponible en: http://www.visual-paradigm.com/.

38. DEWHURST, S.C. *C++ Common Knowledge: Essential Intermediate Programming.* Addison-Wesley Professional, 2005, ISBN 0321321928.

39. GOLDSTEIN, T.C. y SLOANE, A.D. The object binary interface: C++ objects for evolvable shared class libraries. En *Proceedings of the 6th conference on USENIX Sixth C++ Technical Conference - Volume 6. Berkeley, CA, USA. 1994*. p. 1-1.

40. KEEGAN, P.;CHAMPENOIS, L.*, et al. Netbeans™ ide field guide: developing desktop, web, enterprise, and mobile applications.* First ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2005, ISBN 0131876201.

41. BENSON, C.;MULLER-PROVE, M.*, et al.* Professional usability in open source projects: GNOME, OpenOffice.org, NetBeans. En *CHI '04 extended abstracts on Human factors in computing systems. Vienna, Austria. 2004*. p. 1083--1084.

42. LARMAN, C. *UML y Patrones. Introducción al análisis y diseño orientado a objetos.* México: Prentice - Hall, 2003, ISBN 970-17-0261-1.

43. PRESSMAN, R.S. *Ingenieria del Software un enfoque practico.* Sexta ed. Mc Graw Hill, 2005,

44. CHIDAMBER, S.R. y KEMERER, C.F. A Metrics Suite for Object -Oriented Design. *IEEE trans. Software Engineering*, 1994, vol. SE-20, nº 6, p. 476-493.

45. CHIDAMBER, S.R. y KEMERER, C.F. Management Use of Metrics for Object-Oriented Software. *IEEE trans. Software Engineering*, 1998, vol. SE-24, nº 8, p. 629-639.