DISEÑO DE UNA HERRAMIENTA DE MODELADO DE DISEÑO ESTRUCTURADO

JAVIER CARDENAS PEREZ
MARCUS SANCHEZ DIAZ

Monografía para optar a el título de
Ingeniero Sistemas

Director
GIOVANNI VÁSQUEZ MENDOZA
MSc. Ciencias Computacionales

Universidad Tecnológica de Bolívar
Facultad de Ingeniería de Sistemas
Cartagena de Indias
2006

Nota de aceptación:

_____

_____

_____

_____

_____

_____

_____
Presidente del Jurado

_____
Jurado

_____
Jurado

Cartagena, 23 de enero de 2006

1. RESUMEN

Esta monografía trata sobre el diseño de una aplicación de modelamiento estructurado, enfocándose en los diagramas de flujo. La monografía describe las tecnologías, técnicas, consideraciones y pasos a seguir en el diseño de una aplicación de estas características. Describe qué son los DFD, así como su relación con la documentación de diversos tipos de procesos. Se comenta su valor histórico y actual, sus usos más comunes como lo es la documentación de procesos informáticos u organizacionales.

Abarca desde una introducción general a la teoría de diagramas de flujo, pasando por la descripción de la documentación implícita y necesaria de estos, hasta las guías y reglas para la construcción de los mismos

Se centra en las tecnologías específicas necesarias para su eventual ejecución, a las que les siguen ejemplos de implementación. En concreto, profundiza sobre las tecnologías clave de Java empleadas para el diseño, tales como la impresión, serialización y manejo de imágenes mediante la API Java2D.

Las conclusiones de esta pequeña investigación son una visión general sobre el proceso de diseño de una herramienta de modelado estructurado, el cual ha

jugado un papel fundamental en la historia de la ingeniería de software, y las tecnologías necesarias para llevarlo a cabo.

Las conclusiones también son una comprensión de las distintas actividades que integran el desarrollo de sistemas, utilizando UML. Como lo son la obtención de requerimientos y la creación de los diagramas necesarios para la posterior implementación.

CONTENIDO

Página

2. LISTA DE FIGURAS

Pagina

3. LISTA DE CUADROS

Pagina

4. INTRODUCCION

El diseño de una herramienta CASE para la edición de diagramas de flujo y Diccionario de datos, se fundamenta en la importancia para el alumno del entendimiento de las técnicas de análisis estructurado, que se constituye en una de las herramientas básicas para el diseño de sistemas de información y la documentación del funcionamiento de los diferentes componentes que integran estos proyectos.

Este estudio consistirá en desarrollar el diseño de esta herramienta CASE, de acuerdo a las estructuras, procedimientos y actividades propias de la Ingeniería de Software, para que con esto se fundamenten los criterios pertinentes y necesarios al momento de implementar dicho diseño.

La investigación se realizará mediante la observación directa de diversas herramientas similares disponibles. Así mismo, se recurrirá a investigaciones anteriores relacionadas con la construcción de diagramas de flujos realizadas por diferentes autores. Se espera que esta investigación, aporte una herramienta útil a los diseñadores, tanto expertos como principiantes, en sus actividades de análisis y diseño estructurado de sistemas. Además de ser una guía para los programadores en algunas temáticas relacionadas con el lenguaje de programación Java.

5. FUNDAMENTOS TEORICOS

El análisis estructurado siempre ha sido una disciplina de alto contenido grafico, basándose principalmente en diagramas en lugar de texto narrativo. Su herramienta principal es el diagrama de flujo de datos o DFD. El diagrama de flujo de datos es una herramienta de modelado que nos permite representar un sistema como una red de procesos funcionales, conectados unos con otros mediante “tuberías” y”tanques de almacenamiento” de datos, que muestran como fluyen los datos hacia, desde y dentro de un sistema de información y los procesos que los transforman.

5.1. DIAGRAMA DE FLUJO DE DATOS

Introducción.

[1]Los diagramas de flujos de datos son una de las herramientas de modelado de sistemas mas usadas comúnmente, especialmente en sistemas donde las funciones del sistemas son de mucha importancia y mas complejas que los datos que se manipulan. Fueron utilizados por primera vez a principios de la década de los años 70 en el campo de la ingeniería de software como una notación para estudiar problemas de diseño de sistemas, apareciendo en artículos y libros tales como (Stevens, Myers, and Constantine. 1974),

[1] Pressman, R.S. (1993). Pagina 218. Ingeniería del Software, Un enfoque práctico. 3ª edición. McGraw-Hill

(Yourdon and Constantine, 1975), (Myers, 1975), et al. Sin embargo estas notaciones habían sido tomadas de trabajos anteriores relacionados con la teoría de grafos, y continúa siendo utilizada como una notación conveniente para ingenieros de sistemas preocupados con la directa implementación de los modelos de requerimientos de usuario.

No esta de más aclarar que los diagramas de flujo de datos no solo sirven para modelar sistemas de procesamiento de información, sino que también es una forma válida de modelar organizaciones completas, utilizándose como una herramienta de planeación de negocios y planeación estratégica.

Los diagramas de flujo de datos son utilizados para proveer una representaron clara de cualquier función del sistema. Esta técnica empieza con un grafico general del sistema y continua mediante el análisis de cada una de las áreas funcionales de interés. Este análisis puede llevarse a cabo con precisión hasta el nivel de detalle que se requiera, mediante de la explotación de un método llamado expansión de arriba-abajo para conducir el análisis en la dirección que se pretenda.

Notación.

Existen cuatro símbolos principales a la hora de elaborar un Diagrama de flujo de datos. Entidades, procesos, flujo de datos y almacenes de datos. A continuación se explican cada uno de ellos y sus diversas representaciones graficas[2].

5.1.1 Entidades.

Una entidad, también llamados terminales, es la fuente o destino de datos es decir, las entidades proveen datos al sistema o los reciben, Representan entidades externas con las cuales el sistema se comunica. Generalmente una Entidad es una persona o grupo de personas, por ejemplo, una organización externa, un departamento dentro de la misma empresa u organización o posiblemente otro sistema computacional con el que se puede comunicar, todos ellos estando fuera del control del sistema que se esta modelando.

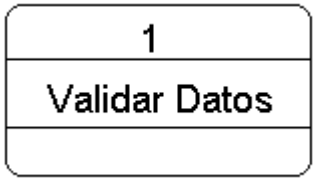
Según la notación, pueden tener varias representaciones graficas:

[2] Yourdon, E. ***Structured Analysis***. http://www.yourdon.com/strucanalysis/chapters/ch9.html

5.1.2 Procesos.

Un proceso, también denominado Actividad o Función, muestra la transformación o manipulación de flujos de datos dentro del sistema, ejecutando computaciones, tomando decisiones, redireccionando flujos de datos basándose en las reglas del negocio. En otras palabras, un proceso recibe una entrada y genera una salida. Los nombres de los procesos (verbos simples, como “Validar datos”, “Encriptar información”) usualmente describen la transformación. Los procesos se pueden dibujar mediante óvalos o por rectángulos segmentados con los bordes redondeados, e incluyen el nombre y numero de identificación del proceso. Por ejemplo:

Notación *Yourdon – Coad*



Notación *Gane – Sarson*

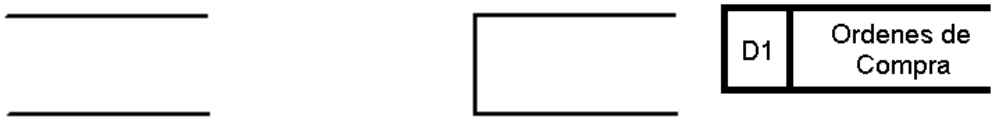
5.1.3. Flujo de Datos.

Un flujo es representado gráficamente por una flecha desde o hacia un proceso. Los flujos son utilizados para describir el movimiento de pedazos o paquetes de información de una parte del sistema a otro, en este caso entre las entidades, procesos y depósitos de datos. Es decir, el flujo representa datos en movimiento, mientras que los depósitos representan datos reposo.

El flujo de datos en un DFD es nombrado para que refleje la naturaleza de los datos usados, los cuales deben ser únicos dentro de un DFD específico.

5.1.4 Deposito de Datos.

Los depósitos se usan para modelar una colección de datos en reposo dentro del sistema. Por lo general, el nombre para identificar un depósito es el plural del nombre de los paquetes que entran y salen en flujos de dicho depósito. Según la notación pueden dibujarse de las siguientes formas:



Notación Gane and Sarson *Notación Yourdon and Coad*

En el según ejemplo, se observa un compartimiento adicional del lado izquierdo. En él se especifica el tipo de depósito de datos mediante un a inicial mayúscula[3]:

D = **D**atos computados (Bases de Datos)

M = **M**anual (Carpeta de Archivos).

T = Datos **T**emporales

T (M) = **T**emporales **M**anual (buzón de correo)

[3] Tomado de http://www.cems.uwe.ac.uk/~tdrewry/dfds.htm

5.1.5. Guías para la creación de DFDs.

Los Diagramas de flujo pueden ser expresados como una serie capas o niveles. Inicialmente, es recomendable hacer una lista de las actividades del sistema para determinar los elementos del DFD (Entidades externas, flujos de datos, procesos y depósito de datos).

A continuación se construye el ***diagrama de contexto,*** el cual muestra un único proceso que representa al sistema completo, y las entidades externas que contribuyen o reciben información del sistema. Este diagrama nos permite visualizar el contexto dentro del cual encaja el proceso del sistema o negocio.

Figura 1. Ejemplo Diagrama de contexto.

Una vez definido el diagrama de contexto, se procede a crea un Diagrama de Nivel 0, en donde se revelan los procesos generales más importantes y los depósitos de datos. A partir de este diagrama se generan los diagramas hijos (Nivel 1 en adelante), los cuales consisten en detallar cada uno de los procesos que hacen parte de procesos superiores, en este caso, los del Diagrama de Nivel 0.

Figura 2. Ejemplo de Un Diagrama de Flujo.

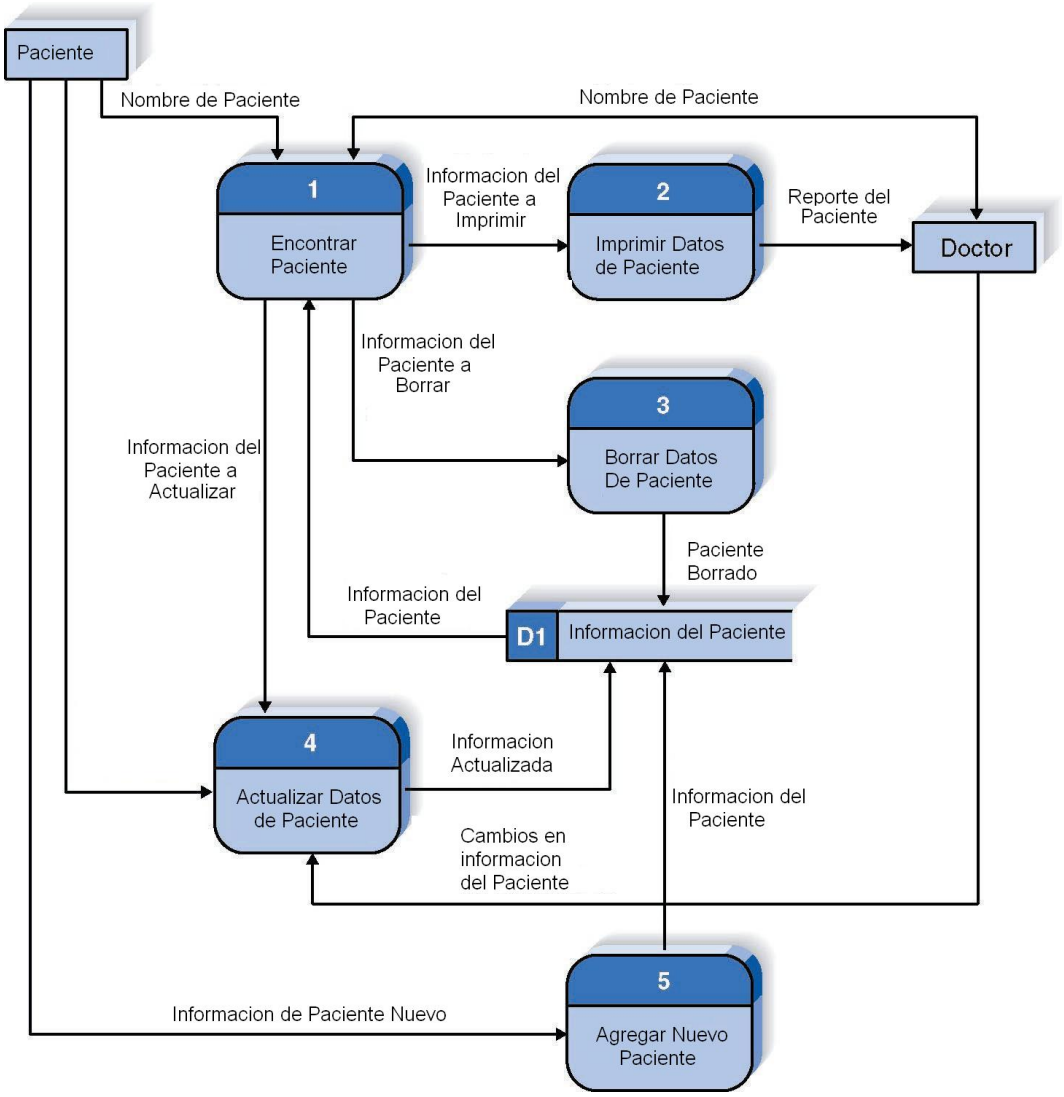
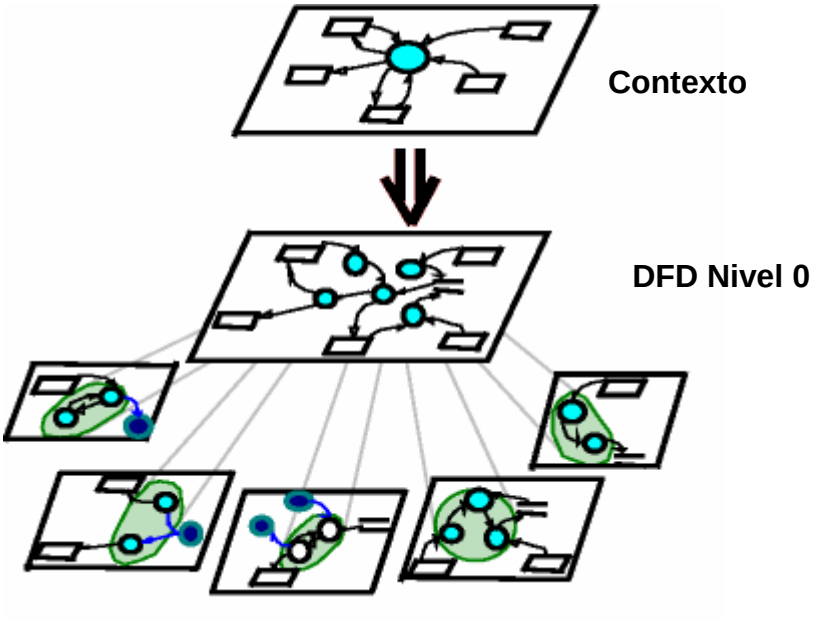


Figura 3. Niveles de un DFD.



5.1.6. Reglas.

Antes de iniciar la creación de Diagramas de flujo de datos, es necesario conocer de antemano algunas reglas que rigen el comportamiento de dichos modelo.

Los depósitos de datos son áreas de almacenamiento estáticas o pasivas; por lo tanto, no es posible tener un flujo de datos entre dos depósitos de datos ya que ninguno tiene la posibilidad de iniciar comunicaciones.

Los depósitos de datos mantienen la información en un formato interno, mientras que las entidades, que representan personas o sistemas externos, tiene información que podría no ser consistentes o sintácticamente correcta, no

es buena idea tener un flujo de datos entre un deposito y una entidad, sin importar la dirección.

Por lo tanto, los flujos de dato solo pueden ocurrir en alguno de los siguientes escenarios:

- Entre un proceso y una entidad (en cualquier dirección).
- Entre un proceso y un depósito de datos (en cualquier dirección).
- Entre dos procesos que solo pueden ejecutarse simultáneamente.

[4]En lo que respecta a los procesos, estos deben tener al menos un flujo de entrada, debido a que si no es así, estarían procesando información a partir de ninguna fuente o evento, a estos proceso mal formados se le denominan ***Hoyo Gris***. Los procesos también deben tener al menos un flujo de salida, de lo contrario se perdería la información procesada, a estos proceso se les denomina ***Hoyos Negros.***

En general cada DFD no debería tener más de seis o siete procesos y no más de seis depósitos de datos. Cada uno de estos procesos debe estar al mismo nivel de detalle conceptual. Si una parte del sistema es muy grande o muy complicada para describirse en un diagrama, es recomendable desglosarlo en dos o más diagramas de menor nivel.

[4] Wixom, D , Wixom, H. (2000) ***System Analysis and Design.*** John Wiley and Sons, Inc

Además de estas reglas, también hay que tener en cuenta algunas recomendaciones[5]:

- Los datos que viajan juntos deberían estar en el mismo flujo.
- Los datos deben ser enviados solo a los procesos que los necesitan.
- Un depósito dentro de un DFD usualmente necesita tener un flujo de entrada.
- No deben existir procesos que tengan solo flujos de salida. A estos procesos erróneos se le denomina ***milagros.***
- No deben haber procesos que tengan flujos de entrada insuficientes para producir una salida.

[5] Le Vie, D.S., Jr. Página 4. Understanding Data Flow Diagrams.

5.2. DICCIONARIO DE DATOS

Durante la etapa de análisis y diseño del sistema, es necesario tener una guía en donde se resuma toda la información que se desarrolló durante la etapa de modelamiento de los DFD.

El diccionario de datos (DD), es una lista ordenada de ítems de datos, en donde cada uno ellos tienen una definición de los componentes que lo conforman que incluyen nombre, descripción, contenido, alias, etc. Sirve para identificar los procesos donde se emplean los datos y la forma como son accesados[6].

Procesos.

Para el caso de los procesos, la información a guardar debe incluir: El nombre, el número de identificación del proceso, los flujos de entrada y salida y un resumen de la lógica del proceso en pseudocódigo. También es valido incluir notas sobre el propósito o la implementación física del proceso, detalles de diseño y cualquier tipo de reseña que sea útil para el equipo de analistas.

Depósitos de Datos.

Al documentar los depósitos de datos, se deben especificar el nombre, una breve descripción que lo distinga de los demás depósitos, los flujos de entrada y salida, y notas. Es de vital importancia describir el tipo, la organización,

[6] Tomado de http://www.monografias.com/trabajos5/inso/inso2.shtml

volumen y el listado de estructuras de datos en caso de ser una tabla de una base de datos.

Flujos de Datos.

Durante la creación de un diccionario de datos, es imprescindible tener la información necesaria acerca de los flujos de datos. En ella se encuentra una descripción de los datos, el origen y destino de los mismos. Además de otras anotaciones de importancia como la de si necesita una conexión a Internet, LAN para realizar la comunicación o la frecuencia de este flujo.

Entidades Externas.

En la información concerniente a las entidades, encontraremos la forma como interactúa con el sistema, que datos recibe o envía y restricciones de las interfaces y protocolos.

6. TECNOLOGIAS JAVA APLICABLES

6.1. SERIALIZACIÓN DE OBJETOS EN JAVA

Debido a la naturaleza de la aplicación a diseñar, es indispensable tener una forma de guardar los datos que hemos creado durante la construcción de un DFD. Por lo tanto, es necesario demostrar como incorporar la persistencia de de objetos en una aplicación Java usando el mecanismo de Serialización proveído por el JSDK a partir de la versión 1.1.

[7]La Serialización implica guardar el estado actual de un objeto en un ***stream,*** y restaurar un objeto equivalente de ese ***stream.*** El ***stream*** funciona como una especie de contenedor para el objeto. Su contenido incluye una representación parcial de la estructura interna del objeto, donde se incluyen variables, nombres y valores. Estos contenedores puede ser de dos tipos: transient (basado en RAM) o persistente (basado en disco), para el caso particular de la aplicación a diseñar, nos centraremos en este ultimo tipo de Serialización, ya que nos permite recuperar los datos almacenados, después de haber finalizado una sesión.

[7] Downs, A. ***Java Serialization***. http://www.mactech.com/articles/mactech/Vol.14/14.04/JavaSerialization

En la etapa de implementación, para que un objeto pueda ser serializado, debe ser una instancia de una clase que implemente la interfaz `Serializable`, la cual permite la persistencia de datos asociados con las variables de un objeto.

La interfaz `Serializable` depende de el mecanismo por defecto de la maquina Java para guardar el estado de un objeto. Escribir un objeto al disco es posible por medio del método `writeObject()` de la clase `ObjectOutputStream` o de la interfaz `ObjectOutput`. Escribir un valor primitivo se puede a través de el método `write<tipo-de-dato>()` apropiado. El proceso de lectura ocurre de una manera análoga, la lectura de un objeto serializado es posible mediante el método `readObject()` de la clase `ObjectInputStream` class, y lo primitivos pueden ser leídos usando los diversos métodos `read<tipo-de-dato>()`.

Cuadro 1. Ejemplo de Serialización y Deserialización en Java.

```
import java.io.Serializable;
import java.io.FileOutputStream;
import java.io.ObjectOutputStream;
import java.io.IOException;
import java.io.FileInputStream;
import java.io.ObjectInputStream;
import java.util.Vector;
import java.awt.*;

public class DFD implements Serializable{

    //...
    Vector<Item> items;

    DFD(){
        //...
        items = new Vector <Item>();
    }

    public void paint(Graphics g){
        //...
        for (Item it : items) {
            it.paint(g);
        }
```

```
    }

    public void guardar(String archivo){
        DFD dfdAGuardar = this;
        FileOutputStream fos = null;
        ObjectOutputStream out = null;

        try {
            fos = new FileOutputStream( archivo );
            out = new ObjectOutputStream( fos );
            out.writeObject( dfdAGuardar );
            out.close();
        }
        catch(IOException ex) {
            ex.printStackTrace();
        }
    }

    public void cargar(String archivo){
        DFD dfdACargar = null;
        FileInputStream fis = null;
        ObjectInputStream in = null;

        try {
            fis = new FileInputStream( archivo );
            in = new ObjectInputStream( fis );
            dfdACargar = (DFD)in.readObject();
            in.close();
        }
        catch(IOException ex){
            ex.printStackTrace();
        }
        catch(ClassNotFoundException ex) {
            ex.printStackTrace();
        }
        this.items = dfdACargar.items;
    }
}

class Item implements Serializable{

    //...
       public void paint(Graphics g){
        //Dibujar el item
       }
}
```

Al momento de serializar no solo guardamos los valores de variables de un objeto, sino además otros objetos que hacen referencia al objeto que estamos

serializando. Por ejemplo, si el objeto a serializar fuera una instancia de `JFrame` y que además contenga un `JPanel`, `JLabels` o `JTextAreas`, estas referencias y sus datos asociados también son convertidos y escritos al stream. Toda esta información almacenada es necesaria para reconstruir el `JFrame` original y cualquier otro objeto que tuviera alguna referencia a este.

Si estos otros objetos o su formatos no fueran almacenados, al momento de reconstruir la ventana, tendría referencias nulas y los componentes no existirían, además cualquier método relacionado con la existencia de ellos lanzarían excepciones.

En el caso de la herramienta que se esta diseñando, esto seria de gran utilidad, puesto que al almacenar un objeto que represente un DFD en su totalidad, al serializarlo, se guardaría toda la información relacionada con los ítems que lo conforman, llámense procesos, depósitos, flujos y entidades, y la información relacionada con ellos como podría ser las coordenadas, colores, etiquetas, descripciones, etc.

Profundizando mas en el tema, encontramos que en ciertas ocasiones se desea prevenir que ciertos campos sean almacenados en el objeto serializado. La interfaz `Serializable` permite a la clase implementadora especificar que algunos de sus atributos no sean guardados o reestablecidos. Esto se realiza mediante el uso de la palabra reservada `transient` antes de la declaración del tipo de dato. Por ejemplo, algunos datos confidenciales pueden ser leídos

desde un archivo y no almacenados en el objeto serializado, o preservar el estado privado de alguna variable, de lo contrario, todos los campos serian escritos sin ningún problema por la clase.

Adicionalmente, aquellos campos declarados como `transient`, campos estáticos no pueden ser serializados, es decir escritos al disco, y por lo tanto no pueden ser deserializados.

La habilidad de guardar y restaurar objetos, puede generar algunos inconvenientes. Supongamos el caso de un objeto que ha estado almacenado por un largo periodo de tiempo, que al ser restaurado se encuentra que su formato ha sido modificado por una nueva versión de su clase.

El flujo que lee la representación serializada del objeto es responsable de detectar cualquier diferencia. La intención es que la nueva clase Java sea capaz de interactuar con representaciones anteriores de la misma clase, siempre y cuando no hubiese ciertos cambios en la estructura de la clase. Aunque esto no suceda de igual forma para una clase mas antigua y una nueva representación de ella.

Para lograr esta meta, necesitamos encontrar una forma de determinar si se tiene la compatibilidad necesaria, durante la etapa de deserialización. Para ello, Java integro a partir de la versión 1.1, la capacidad de especificar los cambios

ejercidos a las clases mediante el uso de un número de versión. Una variable especifica de la clase, `serialVersionUID`, que representa el Identificador Único de Flujo (SUID por sus siglas en ingles), puede ser usado para especificar la versión más antigua de la clase que puede ser deserializada. El SUID se declara de la siguiente manera:

```
static final long serialVersionUID = 2L;
```

Esta declaración en particular y su asignación especifican que la versión numero 2 es la mas antigua hasta donde puede legar la deserialización

No es compatible con un objeto escrito por la versión 1 y no puede escribir un objeto en su versión numero 1. Si se encuentra con un objeto cuya versión sea la uno dentro del flujo, durante la lectura del archivo, se lanzara una excepción del tipo `InvalidClassException.`

El SUID es una medida de compatibilidad, el mismo SUID pude ser utilizado por múltiples representaciones de una clase, siempre y cuando las nuevas versiones puedan leer las anteriores.

Si al momento de codificar la clase no se asigna explícitamente un SUID, un valor por defecto se le será asignado cuando el objeto sea serializado. Esta identificación es un valor numérico único, también conocidos como ***hash***, el cual es computado usando el nombre de la clase, sus interfaces, métodos y atributos mediante el Secure Hash Algorithm (SHA).

Entonces, como se puede determinar que cambios entre las versiones de la clase son aceptables? Para una versión más antigua, que puede contener pocos campos, tratar de leer un objeto serializado de una versión mas reciente puede causar problemas, puesto que existe la tendencia, por así decirlo, de agregar campos a una clase mientras evoluciona, lo que significa que versión anterior no tiene conocimiento es estos nuevos campos. En contraste, una nueva versión de la clase puede buscar campos que no están presentan en la versión anterior, esta le asigna valores por defecto a estos campos.

La documentación de Sun lista varios cambios al formato de la clase que pueden afectar la restauración de un objeto. Algunos de ellos incluyen:

- Borrar un campo, o cambiar su modificador no estático o no transient a uno estático o transient, respectivamente.
- Cambiar la posición de la clase dentro de una jerarquía.
- Cambiar el tipo de dato de un campo primitivo
- Cambiar la interfaz de la clase de `Serializable` a `Externalizable` y viceversa.

Sin embargo, no todos los cambios tendrán un efecto negativo sobre nuestras clases. A continuación se listan algunos cambios que no afectan el comportamiento del objeto:

- Agregar campos, lo cual resultará en la asignación de valores por defecto, basándose en el tipo de dato, a los nuevos campos después de la restauración.
- Agregar nuevas clases, sin embargo, estas tendrán con valores por defecto en sus campos.
- Agregar o quitar los métodos `writeObject()` y `readObject()` pertenecientes a la interfaz Serializable.
- Cambiar el modificador de acceso de un campo, ya que aun es posible asignarle un valor a ese campo.
- Borrar un campo, o cambiar su modificador estático o `transient` a uno no estático o no `transient`, respectivamente.

6.2. JAVA 2D

Para el diseño de la aplicación, la cual contendrá un basto contenido grafico, siendo su principal objetivo permitir la creación de diagramas de flujo de datos de una manera visual e intuitiva, se ha elegido la API Java2D como tecnología base, Java2D es una de los mas grandes herramientas para trabajar con gráficos incluidos en un lenguaje, para explotar toda esta potencialidad es necesario un entendimiento profundo de su filosofía de trabajo y de los servicios que esta presta, a continuación se presentara una inducción al trabajo de gráficos en java a través de esta API.

[8]La API Java2D proporciona una base muy completa y flexible para manipular gráficos de una forma independiente del dispositivo y/o resolución en cualquier programa Java, Java2D extiende las clases `Graphics` y `Imaging` definidas en `java.awt` esta API nos permite incorporar fácilmente gráficos 2D, texto, e imágenes de alta calidad en aplicaciones y applets Java.

Java2D, originalmente desarrollada por Sun Microsystems y Adobe Systems Incorporated, proporciona un modelo bidimensional para líneas, texto e imágenes, en el que se trata uniformemente color, transformaciones espaciales, y composiciones. Con Java2D, se utiliza el mismo modelo de imagen para la pantalla y la impresión, lo que proporciona una forma sencilla de e intuitiva para crear cualquier composición, esto es una impresión en pantalla y una impresión en cualquier otro dispositivo se realizara con los mismos comandos de la clase `Graphics`.

[8] Tomado de java.sun.com\docs\books\tutorial\2d\index.html

Junto con otras API Java, la API Java2D fue creada para facilitar a los programadores el desarrollo de aplicaciones que incorporen interfaces avanzadas de usuario. Los servicios de la API Java2D incluyen:

- Gráficos, textos e imágenes de alta calidad e independientes del dispositivo.
- Solución sencilla y compacta para imágenes y gráficos 2D.
- Complementar otras tecnologías Java, para complementar interfaces y multimedia.

Características

Cuadro 2. Características de Java2D.

Graficas	Suavizado de bordes Curvas Transformaciones Composiciones Atributos de Renderización extendidos Estilo de rellenos arbitrarios Transparencias
Texto	Manejo Avanzado de Imágenes Estilos Avanzados de Texto
Imágenes	Manipulación de imágenes flexibles en memoria Operaciones avanzadas de imagines.(convoluciones, tabla de índices, transformaciones de enfoque y desenfoque)
Dispositivos	Especificaciones claras para soportar dispositivos arbitrarios de gráficos, como impresoras y pantallas de toda clase.

Java2D proporciona muchas ventajas a los desarrolladores que desean incorporar los gráficos, texto, e imágenes en sus aplicaciones y/o applets, es decir la Java2D beneficia virtualmente todos los desarrolladores que programen en Java. Sin duda una buena elección en el desarrollo de una aplicación con tal alto contenido visual como lo es una herramienta CASE.

Gráficos 2D, imágenes y AWT

Java2D se convierte en parte de las bibliotecas de los paquetes base `java.awt` y de `java.awt.image`. Extendiendo las clases existentes, manteniendo así la compatibilidad con los programas existentes y permite que los programas legados se ajusten sin problemas a las características proporcionadas por `java.awt` y la misma API.

Fundamentos de Java 2D.

Java2D maneja formas texto, e imágenes arbitrarias y proporcionan un mecanismo uniforme para realizar transformaciones tales como rotación y escalamiento. La API Java2D también proporciona soporte de fuentes y de colores.

6.2.1 Un Modelo Sencillo de Dibujo

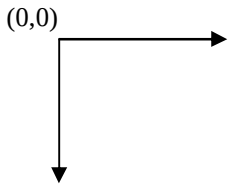
Java2D utiliza el modelo de dibujo definido por el paquete de `java.awt` para dibujar en pantalla, cada objeto implementa un método "`paint`" que es invocado automáticamente cuando se necesita dibujar algo, Cuando el método `paint` es invocado, se le pasa un objeto `Graphics` propio del entorno actual, y que contiene la implementación correcta de todos los métodos de a clase. Java2D nos provee uno de estos objetos subclase de `Graphics`, `Graphics2D`, con características adicionales para especificar estilos de decorado, definiendo formas complejas, y controlando el proceso de renderizado. Java2D trata las líneas, el texto, y las imágenes uniformemente; todos pueden ser rotados, escalados, sesgados, y ser utilizados en una composición con una serie de métodos comunes entre ellos.

El proceso de dibujo básico es igual para cualquier componente:

1. Se especifica los atributos apropiados para los gráficos que se quiere dibujar fijando los atributos de dibujo en el objeto de `Graphics2D`.
2. Se define la forma, el texto, o la imagen que se desea dibujar.
3. Se usa el objeto `Graphics2D` para renderizar la forma, el texto, o la imagen llamando uno de los métodos de renderización de `Graphics2D`.

6.2.1.1. Coordenadas Espaciales

Java2D define dos espacios de coordenadas: ***las del usuario*** y el ***espacio del dispositivo***. El origen del espacio de dispositivo comienza en la esquina izquierda superior los valores de ***X*** aumentan a la derecha y los de ***Y*** aumentan hacia abajo.



Todos los objetos gráficos son descritos de una manera independiente al dispositivo, hasta que se renderizan en un dispositivo tal como una pantalla o impresora. El estado de representación de un objeto de `Graphics2D` asociado al dispositivo incluye un objeto del `Transform` que convierte las coordenadas de usuario del objeto a coordenadas de dispositivo. El defecto transforma resultados en un espacio del coordenada de usuario el resultado por defectos es una coordenada de usuario con la orientación correcta en el dispositivo.

6.2.1.2 Líneas

La API Java2D proporciona una implementación de la interfaz `Path` (Trayectoria) que puede ser usada para definir formas complejas. Esta clase, `BezierPath`, permite describir una forma usando una combinación de líneas y curvas. Usar la clase `BezierPath` para definir la forma de un objeto permite que se controle la localización del objeto (en espacio de coordenadas de

usuario) así como su forma. La forma se puede transformar a una diversa posición, tamaño, y la orientación en el espacio.

```
public void paint(Graphics g) {
  // Definir parámetros del pincel
  g.setColor(AlgunColor);
  // Dibujar una linea
  g.drawLine(...)
}
```

6.2.1.3 Texto

Java2D proporciona avanzados manejos sobre el texto, soportando desde el uso de fuentes simples hasta un uso verdaderamente profesional y de alta calidad. El texto es tratado con las más versátiles técnicas, puede ser dibujado, transformado, utilizado como mascara, y realizar composiciones con ellos como cualquier otro elemento gráfico.

Fuentes

Java2D define una clase `Font` que provee un gran control sobre las fuentes, superando a las operaciones básicas antes ofrecidas por la clase `java.awt.Font`, Esta nueva clase también permite obtener información acerca de una fuente en particular, e incluso líneas y curvas de una letra en especial.

La información que describe una fuente, tal como su nombre, estilo y parámetros es almacenada en `Font` y en `FontFeature`, esta última clase

es utilizada para describir las particularidades de cada tipo de fuente, de manera que cada objeto `Font` tiene un arreglo de objetos `FontFeature` que describe todas las características de la fuente

La clase `Font` también permite el acceso a fontmetrics, cada objeto `Font`, nos permite acceso completo a la información de sus medidas a través de los métodos `getDesingMetrics`, `getGlyphMetrics` y `getGlyphOutline`.

Estilos

Antes de que cualquier porción de texto sea impresa, es necesario determinar exactamente donde se debe colocar cada carácter, cada FontFeature puede alterar la posición de la siguiente letra, como el tamaño el estilo, la dirección, etc. estos estilos configurables son conocidos como ***Fonts Layout***, o estilos de fuente.

Java2D provee facilidades que manejan la mayoría de las combinaciones necesarias para manejar textos, incluyendo Fuentes mezcladas, diversos lenguajes, y texto bidireccional.

Aunque no es nuestro caso, Java2D provee a los clientes avanzados que quieran hacer sus propias implementaciones de estilos ejercer este derecho extendiendo las clases correspondientes y sobrescribiendo comportamientos a la medida.

6.2.2 Imágenes

Java2D AP brinda un gran rango de posibilidades para manejar imágenes provistas primeramente por las clases `java.awt` y `java.awt.image`, a las que se fueron añadidas un gran numero de clases que incluyen `BufferedImage`, `Tile`, `Channel`, `ComponentColorModel` y `ColorSpace`.

Estas clases proveen al programador de avanzadas herramientas, que permiten gran control sobre las imágenes y la forma de crearlas, permitiéndoles crear imágenes por en gammas de colores por fuera de la RGB, permitiendo una creación mas exacta de las mismas, la clase `BufferedImage` de Java2D incluso nos permite especificar como son cargados y descargados los píxeles en la memoria, lo que es prueba irrefutable del inmenso control cedido a los programadores por esta API.

Como todos los objetos gráficos las imágenes son transformadas por la clase `Transform` asociada al objeto Graphics2D cuando son dibujados, lo que significa que las imágenes pueden ser rotadas, escaladas, cortadas, en otras palabras transformadas de la misma manera que si fueran texto o líneas. Sin dejar a un lado que mantienen sus propios atributos como modelo de color, datos de color, tamaño etc. Cabe destacar que una imagen puede ser usada como un dispositivo más y renderizarlas dinámicamente con un objeto Graphics2D.

6.2.3 Figuras (Shape)

Una vez que se maneja las nociones fundamentales de Java2D como las líneas, texto imágenes, podemos comenzar a usar funciones un nivel mas alto, en Java2D existe una clase `Shape`, figura en español de la cual existen muchas extensiones que nos ayudan a crear formas básicas con tan solo usar funciones que proveen mecanismos para esto y con tan solo aportar los atributos básicos de la forma. Existen diversas formas como elipses, cuadrados, círculos, cuadrados redondeados, etc. Todas estas figuras de Java2D están definidas dentro de la clase `Graphics2D`, luego todos los métodos `paint` para renderizar imágenes tienen un objeto Graphics por lo que se debe hacer un casting antes de poder usar las características agregadas con Java2D:

Cuadro 3. Métodos de la clase `Graphics2D.`

```
public void paintComponent(Graphics g) {
  // Casting necesario a Graphics2D
  Graphics2D g2d = (Graphics2D)g;
  // Se utiliza el objeto Graphics2D
  g2d.setPaint(ColorDeLlenadoOPatronDeLlenado);
  g2d.setStroke(GrosorDelLapiz);
  g2d.setFont(CualquierFuente);
  g2d.translate(...);
```

```
g2d.rotate(...);
g2d.scale(...);
g2d.shear(...);
g2d.setTransform(ObjetoTransform);
```

Figuras.

Los argumentos de Graphics2D implementan la interfaz Shape, de manera que siempre se esta dibujando un “Shape”, esto es una forma, entendiendo como forma incluso las líneas sencillas, podemos crear nuestra propia figura que implemente la interfaz Shape, una vez se crea la figura, el proceso solo consiste hacer la composición necesaria con estas, entre las figuras que implementan shape tenemos.

- Arc2D.Double, Arc2D.Float
- Área (esta es una figura que se forma de la sustraccion o suma de varias figures mas)
- CubicCurve2D.Double, CubicCurve2D.Float
- Ellipse2D.Double, Ellipse2D.Float
- GeneralPath (esto es una serie de figuras conectadas)
- Line2D.Double, Line2D.Float
- Polygon
- QuadCurve2D
- Rectangle2D.
- RoundRectangle2D

6.3. IMPRESIÓN EN JAVA.

En este apartado, se explicará la terminología básica usada en la impresión e introducir el modelo de impresión Java y su API.

Unidades de medición.

[9]Cuando se trabajaron la clase `Graphics2D`, es importante entender la diferencia entre ***espacio de maquina*** y ***espacio de usuario***. En el espacio de maquina, se trabaja en píxeles usando la resolución de la maquina. Lo cual quiere decir que un cuadrado de 100 x 100 píxeles impreso en una maquina con mayor cantidad de píxeles por centímetro cuadrado, resultara en un cuadrado más pequeño.

El espacio de usuario, en contraste, nos permite pensar en términos de unidades de medidas, sin importar la resolución de la maquina. Cuando se crea un objeto de la clase `Graphics2D` para una maquina en particular, ya sea una impresora o la pantalla, una transformación por defecto es generada para guardar el espacio de usuario a el espacio de la maquina. En el espacio de usuario el valor por defecto está establecido en 72 coordenadas por pulgada, el equivalente 28.3 coordenadas por centímetro. En lugar de pensar en términos de píxeles, se piensa en términos de

[9] Dubé, J. ***Printing In Java.*** http://www.javaworld.com/javaworld/jw-10-2000/jw-1020-print.html

unidades, sabiendo que un cuadrado de de 1 pulgada cuadrada tiene 72 unidades cuadradas, una pagina tamaño carta cuyas dimensiones son de 8.5 por 11 pulgadas tendrá 612 por 792 puntos. Cuando se use la API de impresión, se debe tener en cuenta que se trabajará con unidades porque todas las clases del paquete `java.awt.print` trabajan en espacio de usuario.

El sistema de impresión de Java ha mejorado considerablemente en sus últimas versiones. A partir de la versión 1.2, el sistema de impresión nos permite usar la API Java 2D, la cual es una de las APIs graficas mas avanzadas construidas como parte de un lenguaje de programación, para renderizar una pagina. Esta API permite que todo lo que sea dibujado en la pantalla sea trasladado al papel.

A pesar de sus avances, la API de impresión solo soporta a la impresora seleccionada como predeterminada por el usuario en cualquier momento. Lo cual quiere decir que Java no soporta la obtención de una lista de impresoras locales o compartidas en red disponibles y sus propiedades en un computador determinado. La única forma para que un usuario seleccione una impresora es mediante la aparición del cuadro de dialogo de impresión, ya que no existe una forma de obtener esta lista mediante la programación.

El modelo de impresión ha cambiado completamente desde Java 1.2. En versiones anteriores, el proceso de renderización no estaba optimizado, por ejemplo, en Java 1.1 la impresión de una sola página requería una gran cantidad de memoria y era un poco lenta. En Java 1.2 se actualizó y optimizo el proceso de renderización. Esta API rediseñada está basada en un modelo de ***callback***, en donde el subsistema de impresión, no nuestro programa, controla cuando se renderiza una pagina, Este proceso presenta una naturaleza mas orientada a objetos que la usada en la versión 1.1, en donde la aplicación estaba encargada de este proceso.

Para simplificar el concepto, nuestros programas tienen un contrato con el subsistema de impresión para proveer cierta página en determinado memento. El subsistema de impresión podría necesitar que la aplicación renderice una pagina más de una vez, o renderizar paginas fuera de secuencia. Este modelo provee varias ventajas. En primer lugar, el hecho de enviar secciones de una página en vez de la página completa a la impresora, permite a la aplicación imprimir documentos complejos que normalmente requerirían más memoria de la disponible. La aplicación no necesita saber como imprimir cada sección, únicamente debe saber como renderizar cierta pagina, dejando el resto a la API. En este caso, el subsistema de impresión puede pedir que una página sea renderizada varias veces dependiendo del número de secciones necesarias para imprimir una página en su totalidad.

6.3.1. Modelos de renderización.

En java existen dos modelos de impresión: trabajos de tipo `Printable` y trabajos de tipo `Pageable`.

Printable.

Los trabajos `Printable` son los más simples de los dos modelos. Ese modelo solo usa un `PagePrinter` para todo el documento. Las paginas son renderizadas en secuencia empezando a partir de la pagina cero, cuando la ultima pagina se ha haya impreso, el `PagePrinter` debe retornar el valor `NO_SUCH_VALUE`. El subsistema siempre espera que la aplicación renderice las paginas en secuencia, como ejemplo, si a una aplicación se le pide que imprima de la pagina cinco hasta la ocho, el subsistema de impresión pedirá todas las páginas, pero solo imprimirá partir de la quinta en adelante, además si la aplicación muestra un diálogo de impresión, el numero total de paginas a imprimir no será desplegado, ya que es imposible saber de antemano el numero d paginas en el documento cuando se utiliza este modelo.

Pageables.

Los trabajos de tipo `Pageable` ofrecen más flexibilidad que los anteriores, puesto que en cada página de un trabajo `Pageable` puede presentar un layout diferente. Este tipo de modelo se utiliza con más frecuencia junto con los `Books`, que no son más que una colección de páginas que pueden tener diferentes formatos.

Un trabajo `Pagable` tiene las siguientes características:

- Cada página puede tener su propio painter. Por ejemplo, se puede implementar un painter para una portada, otro para imágenes y un tercero para el resto de un documento.
- Se puede establecer un formato diferente para cada página en el libro, mezclando entre páginas de tipo retrato o paisaje.
- El subsistema de impresión puede pedir a la aplicación imprimir paginas fuera de secuencia, y algunas paginas pueden ser obviadas si es necesario. No nos debemos preocupar por ello siempre y cuando le proveamos cualquier página en el documento por demanda.
- El trabajo de tipo `Pageable` no necesita saber cuantas páginas hay en un documento.

Books.

Aparecen dentro de la API de impresión de Java a partir de la versión 1.2. Esta clase permite crear documentos con muchas páginas. Cada página tiene su propio formato y painter, dando la oportunidad y flexibilidad de crear documentos sofisticados.

La clase `Book` representa una colección de páginas. Cuando se instancia por primera vez el objeto `Book` está vacío. Para añadir paginas, simplemente se utilizan alguno de los dos métodos `append()`. Los parámetros de estos métodos son un objeto de la clase `PageFormat`, el cual define las características físicas de la página, y un objeto `PagePainter`, el cual debe implementar la interfaz `Printable`. Si no se conoce el numero de paginas a imprimir simplemente se pasa el valor `UNKNOWN_NUMBER_OF_PAGES` al método `append()`. El sistema de impresión automáticamente encuentra el número de páginas llamando a los painter de cada página del libro hasta que reciba el valor `NO_SUCH_PAGE`.

7. DOCUMENTO DE REQUERIMIENTOS

7.1. INTRODUCCIÓN

Este documento describe claramente nuestro proyecto. Que consta de una herramienta CASE (Computer Assisted System Engineering) para el modelado de aplicaciones, centrándonos en las técnicas de diseño estructurado: Diagramas de Flujos de Datos y Diccionario de datos.

Ya que el fin de este es agilizar el proceso de documentación del software, dando la posibilidad de dibujar intuitivamente diagramas de flujos de datos y generar su respectivo diccionario de datos.

Con el siguiente documento pretendemos dar a conocer el propósito, objetivos, requerimientos y casos de uso de nuestra aplicación con los cuales estamos detallando los pasos, precondiciones y garantías de éxito de la misma.

Adicionalmente, el programa tendrá la capacidad de guardar, abrir y editar en archivos los avances en cada diagrama, también contara con la opción de imprimir los diagramas, para incluir estos diagramas en cualquier documento de diseño en el que se este trabajando.

7.1.1 Propósito del documento de Requerimientos.

Brindar al usuario la documentación necesaria donde se declaran las funciones que proveerá el sistema, haciendo énfasis en la obtención de requerimientos.

7.1.2 Alcance del documento de Requerimientos.

El siguiente documento se ha elaborado como requisito fundamental para el desarrollo del contenido de este trabajo investigativo. En éste describiremos los requerimientos necesarios del sistema, aplicando los conceptos de la ingeniería de software.

7.1.3 Descripción del documento de Requerimientos.

Este documento especifica los requerimientos obtenidos, necesarios para el correcto desarrollo de nuestro software.

7.2. DESCRIPCIÓN.

7.2.1. Funciones del producto.

El producto permitirá al usuario simplificar las tareas relacionadas con el desarrollo de aplicaciones basadas en el diseño estructurado, al mismo tiempo que sirva como herramienta de aprendizaje al usuario inexperto.

En la materia Ingeniería de Software I, donde se imparten estos conceptos de diseño estructurado, será de gran ayuda para los estudiantes, el diseño amigable y simplificado de la herramienta fomentara el aprendizaje de estos temas a la vez que servirá de apoyo para el desarrollo de los proyectos que se proponen en esta asignatura.

Adicionalmente, el programa tendrá la capacidad de generar el diccionario de datos, a partir de la información adicional digitada en el momento de edición del DFD.

7.2.2. Características del usuario.

Se considerarán dos tipos de usuarios. El primero, será el alumno el cual tendrá poca experiencia en la creación de estos diagramas y la estructura de estos. El segundo el usuario experimentado que fue instruido con estas técnicas de modelado.

7.2.3. Problema del usuario.

El usuario académico no posee una herramienta de uso libre y dirigido exclusivamente a este tópico, el problema en este sentido se debe a la complejidad de las herramientas actuales que dificultan la comprensión del tema, además al usuario experimentado le es difícil hallar una herramienta novedosa que se centre en el diseño estructurado.

7.2.4. Objetivo del usuario

Solucionar su problema mediante una aplicación que automatice la construcción de diccionario de datos, y el DFD, centrándose en el diseño estructurado, mostrándose como una herramienta simple y de uso libre.

7.3. OBJETIVOS DEL SISTEMA.

Cuadro 4. Objetivos del sistema.

[01]	Graficar Diagramas de Flujo de Datos (DFD)
Actores	Usuario
Descripción	El sistema debe proporcionar las herramientas para facilitar el diagramado del proyecto.
Importancia	Vital.
Urgencia	Inmediata.
Comentarios	Ninguno.

[02]	Generación De Diccionario de Datos
Actores	Usuario.
Descripción	El sistema deberá encargarse de la creación del diccionario de datos correspondiente a un DFD, si el usuario así lo desea.
Importancia	Vital.
Urgencia	Inmediata.
Comentarios	Ninguno.

[03]	Generación de Imágenes
Actores	Usuario.
Descripción	El sistema deberá exportar los diagramas a imágenes, para su manipulación fuera del sistema.
Importancia	Moderada.
Urgencia	Moderada.
Comentarios	Ninguno.

.

7.4. REQUERIMIENTOS DEL SISTEMA.

7.4.1. Requerimientos funcionales

La aplicación debe permitir dibujar fácilmente un DFD y al darle la información necesaria generar el respectivo diccionario de datos.

El sistema deberá poder exportar los diagramas a una imagen.

El sistema debe validar que no existan incoherencias con respecto a las normas de los DFD, y avisar cual ha sido el error y en lo posible corregirlos.

7.4.2. Definición de actores.

Cuadro 5. Usuarios del sistema.

[01]	Usuario
Descripción	Representa el usuario que manipula la herramienta.
Comentario	Será el único.

7.5. CASOS DE USO.

Cuadro 6. Caso De Uso Crear Nuevo Proyecto.

[01]	Crear nuevo Proyecto DFD	
Actores	Usuario	
Descripción	Crear un nuevo proyecto para dibujar un DFD	
Precondición	La aplicación se debe iniciar.	
Secuencia normal	Paso	Acción
	1	Ir al menú archivo – nuevo – Diagrama DFD
	2	El sistema crea una nueva ventana hija.
	3	Muestra una barra de herramientas con todas las opciones disponibles.
Post condición	Se ha creado un nuevo espacio de trabajo	
Excepciones		
Rendimiento	Paso	Cota de tiempo
	2	1 seg.
	3	1 seg.
Frecuencia esperada	Indeterminada.	
Importancia	Vital.	
Urgencia	Inmediata	
Comentarios	Ninguno	

Cuadro 7. Caso De Uso Guardar Proyecto.

[02]	Guardar Proyecto DFD	
Actores	Usuario	
Descripción	Guarda en disco los avances del proyecto	
Precondición	Se debe haber creado un proyecto	
Secuencia normal	Paso	Acción
	1	Ir al menú Archivo – Guardar
	2	Se muestra un dialogo para seleccionar la ubicación y el nombre del proyecto.
	3	El sistema serializa los objetos y son guardados en el disco.
Post condición	Los objetos que componen el proyecto se han guardado en el disco, para su posterior manipulación.	
Excepciones		
Rendimiento	Paso	Cota de tiempo
	2	1 seg.
	3	1 seg.
Frecuencia esperada	Indeterminada.	
Importancia	Vital.	
Urgencia	Inmediata	
Comentarios	Ninguno	

Cuadro 8. Caso De Uso Abrir Proyecto.

[03]	Abrir Proyecto DFD	
Actores	Usuario	
Descripción	Carga en el espacio de trabajo un proyecto guardado previamente.	
Precondición	Se debe haber guardado un proyecto.	
Secuencia normal	Paso	Acción
	1	Ir al menú Archivo – Abrir
	2	Se muestra un dialogo para seleccionar la ubicación y el nombre del proyecto que se va a abrir.
	3	El sistema deserializa los objetos y son cargados al programa.
Post condición	Los objetos que componen el proyecto se han cargado desde el disco, permitiendo seguir la construcción del proyecto.	
Excepciones		
Rendimiento	Paso	Cota de tiempo
	2	1 seg.
	3	1 seg.
Frecuencia esperada	Indeterminada.	
Importancia	Vital.	
Urgencia	Inmediata	
Comentarios	Ninguno	

Cuadro 9. Caso De Uso Crear Nueva Entidad.

[04]	Creación de una Entidad Externa.	
Actores	Usuario.	
Descripción	El usuario dibujara una entidad externa.	
Precondición	Crear un espacio de Trabajo.	
Secuencia normal	Paso	Acción
	1	El usuario selecciona el botón Entidad externa, del menú
	2	El usuario hace clic en el área donde desea colocar la entidad.
	3	El sistema dibuja desde ese punto un cuadro que representa dicha entidad.
	4	Se hace doble clic en el nuevo objeto y se edita su información (Nombre y Descripción).
Post condición	El usuario ha agregado la entidad satisfactoriamente	
Excepciones	2	El usuario hace clic fuera del área de trabajo, y se muestra un mensaje de advertencia
Rendimiento	Paso	Tiempo
	3	1 seg.
	4	1 seg.
Frecuencia esperada	Indeterminada	
Importancia	Moderada	
Urgencia	Se requiere	
Comentarios		

Cuadro 10. Caso De Uso Crear Nuevo Proceso.

[05]	Crear un Proceso	
Actores	Usuario	
Descripción	Se crea un proceso en el DFD	
Precondición	Crear un área de trabajo	
Secuencia normal	Paso	Acción
	1	El usuario selecciona el botón proceso, del menú
	2	El usuario hace clic en el área donde desea colocar el proceso.
	3	El sistema dibuja desde ese punto un círculo que representa el proceso.
	4	Se hace doble clic en el nuevo objeto y se edita su información (Nombre, Descripción y Resumen de la lógica).
Post condición	Se ha agregado un nuevo proceso al área de trabajo	
Excepciones	2	El usuario hace clic fuera del área de trabajo, y se muestra un mensaje de advertencia.
Rendimiento	Paso	Tiempo
	3	1 seg.
	4	1 seg.
Frecuencia esperada	Indeterminada (al menos una vez)	
Importancia	Vital	
Urgencia	Se requiere	
Comentarios		

Cuadro 11. Caso De Uso Crear Nuevo Flujo.

[06]	Crear un flujo de datos	
Actores	Usuario	
Descripción	Se crea un nuevo flujo de datos en el DFD	
Precondición	Se debe crear un área de trabajo, además de al menos dos objetos relacionables.	
Secuencia normal	Paso	Acción
	1	El usuario selecciona el botón flujo de datos, del menú
	2	El usuario hace clic y deja presionado el botón sobre el objeto de origen, y arrastra el cursor hasta el objeto destino.
	3	El sistema dibuja desde ese punto un cuadrado con una muesca que representa el depósito de datos.
	4	Se hace doble clic en el nuevo flujo de datos y se edita su información (Nombre, Descripción, Tipo).
Post condición	El flujo de datos se ha creado satisfactoriamente	
Excepciones	2	El usuario no hizo clic sobre un objeto
	3	El usuario suelta el cursor en un espacio vacío.
	3	El usuario no arrastra el curso hasta otro objeto valido.
Rendimiento	Paso	Tiempo
	3	1 seg.
	5	1seg.
Frecuencia	Mas de una vez	
Importancia	Vital	
Urgencia	Importante	
Comentarios		

Cuadro 12. Caso De Uso Crear Nuevo Deposito.

[07]	Crear un deposito de datos	
Actores	Usuario	
Descripción	Se crea un nuevo deposito de datos en el DFD	
Precondición	Se debe crear un área de trabajo.	
Secuencia normal	Paso	Acción
	1	El usuario selecciona el botón Diccionario de datos, del menú
	2	El usuario hace clic en el área donde desea colocar el depósito de datos
	3	El sistema dibuja desde ese punto un cuadrado con una muesca que representa el depósito de datos.
	4	Se hace doble clic en el nuevo objeto y se edita su información (Nombre, Descripción, Estructura de Datos, volumen y acceso).
Post condición	El deposito de datos se ha creado satisfactoriamente	
Excepciones	2	El usuario no hizo clic sobre un objeto
	3	El usuario suelta el cursor en un espacio vacío.
Rendimiento	Paso	Tiempo
	1	2 seg.
	1	1 seg.
Frecuencia	Indeterminada	
Importancia	Vital	
Urgencia	Moderada	
Comentarios	Ninguno	

Cuadro 13. Caso de Uso Validar DFD.

[08]	Validar el DFD	
Actores	Usuario.	
Descripción	El Usuario puede selecciona esta opción para saber si el proyecto sigue las reglas de la construcción de DFDs	
Precondición	Se debe haber creado un proyecto.	
Secuencia normal	Paso	Acción
	1	Se elige la opción Validar DFD del menú
	2	El sistema hará un recorrido por cado uno de los elementos que conforman el DFD.
	3	El sistema verifica si se cumplen todas las reglas para cada uno de los elementos
	4	El sistema genera un mensaje dependiendo del resultado.
Post condición	El DFD ha sido analizado y se muestra un mensaje con el resultado de este análisis.	
Excepciones	2	No se han agregado ítems.
Rendimiento	Paso	Tiempo
	2	1.5 seg.
	3	2 seg.
	4	1 seg.
Frecuencia esperada	Indeterminada	
Importancia	Vital	
Urgencia	Moderada	
Comentarios	En caso de que el DFD presente una malformación, el sistema indicara que ítems presentan problemas.	

Cuadro 14. Caso De Uso Generar Diccionario de Datos.

[09]	Generar Diccionario de Datos.	
Actores	Usuario.	
Descripción	El Usuario puede generar un diccionario de datos a partir de un Diagrama de flujo de datos	
Precondición	Se debe haber dibujado un DFD sin errores.	
Secuencia normal	Paso	Acción
	1	Se elige la opción generar diccionario de datos del menú
	2	El sistema hace un chequeo para revisar la validez del diagrama (Ver caso de uso #8).
	3	El sistema hace una recorrido por todos los objetos recopilando información
	4	El sistema genera de una manera legible el diccionario de datos
	5	El sistema muestra en pantalla el diccionario generado.
	6	El sistema pregunta al usuario si desea guardar el documento.
	7	El diccionario de datos se guarda en el disco.
Post condición	Se ha generado el diccionario de datos correspondiente al DFD activo.	
Excepciones	2	El diagrama esta mal construido.
Rendimiento	Paso	Tiempo
	2	1.5 seg.
	3	2 seg.
	4	2 seg.
Frecuencia esperada	Indeterminada	
Importancia	Moderada	

Urgencia	Moderada
Comentarios	Ninguno

Cuadro 15. Caso De Uso Imprimir Diccionario de Datos.

[10]	Imprimir Diccionario de Datos.	
Actores	Usuario.	
Descripción	El Usuario puede imprimir el diccionario de datos generado a partir de un Diagrama de flujo de datos.	
Precondición	Se debe haber creado un proyecto y construido un DFD.	
Secuencia normal	Paso	Acción
	1	Se elige la opción imprimir diccionario de datos del menú
	2	El sistema verifica si ya se ha creado un diccionario de datos para el proyecto actual.
	2.1	Si no hay un diccionario de datos asociado, éste se genera (Ver caso de uso #9)
	3	El sistema muestra el cuadro de dialogo de impresión.
	4	El usuario ajusta las propiedades de la impresión.
	5	El sistema imprime el diccionario por medio de la impresora seleccionada.
Post condición	Se ha generado el diccionario de datos correspondiente al DFD activo.	
Excepciones	2.1	El diagrama esta mal construido.
	4	No hay una impresora disponible.
Rendimiento	Paso	Tiempo
	2	1 seg.
	2.1	3 seg.
	4,5	Variable.
Frecuencia esperada	Indeterminada	
Importancia	Moderada	

Urgencia	Moderada
Comentarios	Ninguno

Cuadro 16. Caso de Uso Exportar Imagen.

[11]	Exportar una Imagen.	
Actores	Usuario.	
Descripción	El Usuario puede generar una imagen partir de un Diagrama de flujo de datos	
Precondición	Se debe haber dibujado un DFD.	
Secuencia normal	Paso	Acción
	1	Se elige la opción Exportar imagen, en el menú.
	2	Aparece cuadro de dialogo para guardar el archivo grafico.
	3	El sistema genera una imagen y la guarda en disco.
Post condición	Se ha exportado el DFD a un formato grafico.	
Excepciones		
Rendimiento	Paso	Tiempo
	3	1 seg.
Frecuencia esperada	Indeterminada	
Importancia	Moderada	
Urgencia	Moderada	
Comentarios	Ninguno	

Cuadro 17 Caso de Uso Explotar Proceso.

[11]	Explotar Proceso.	
Actores	Usuario.	
Descripción	El Usuario creara un nuevo DFD a partir de un proceso ya definido.	
Precondición	El Usuario debe haber creado un proceso con anterioridad.	
Secuencia normal	Paso	Acción
	1	Se hace doble clic en el proceso que se desea expandir.
	2	El sistema crea un nuevo nivel en la jerarquía del DFD que es reflejado en el JTree.
	3	Se dibuja en el canvas el nuevo DFD con los flujos heredados de su proceso padre.
Post condición	El proceso ha sido expandido y se crea un nuevo DFD listo para su edicion.	
Excepciones		
Rendimiento	Paso	Tiempo
	3	1 seg.
Frecuencia esperada	Indeterminada	
Importancia	Vital.	
Urgencia	Moderada	
Comentarios	Ninguno	

Cuadro 18 Caso de Eliminar Ítem

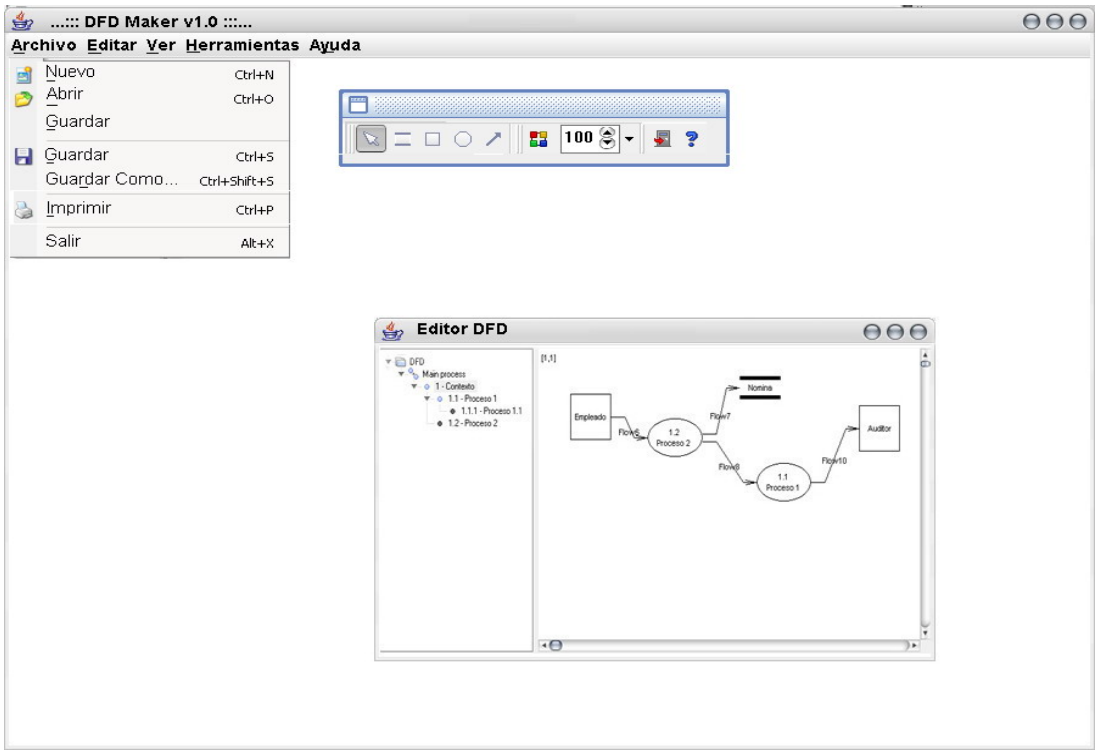
[11]	Eliminar Ítem.	
Actores	Usuario.	
Descripción	El Usuario eliminara un Ítem ya existente del DFD.	
Precondición	El Usuario debe haber creado un ítem con anterioridad.	
Secuencia normal	Paso	Acción
	1	Se hace clic en el ítem.
	2	Se escoge la opción suprimir del menú de edición.
	3	Se elimina el Ítem del vector que lo contiene y los flujos asociados a el, y se refresca el canvas.
	4	Se actualiza el canvas con las modificaciones.
Post condición	El ítem se ha borrado, y no pertenece mas al DFD.	
Excepciones		
Rendimiento	Paso	Tiempo
	3	1 seg.
Frecuencia esperada	Indeterminada	
Importancia	Moderado.	
Urgencia	Moderada	
Comentarios	Ninguno	

8. DISEÑO DE LA APLICACIÓN

La arquitectura propuesta para la aplicación, es la siguiente:

- Debe ser una herramienta ***Stand Alone*** o de escritorio debido a su naturaleza no requiere de capas intermedias o trabajo en equipo.
- Es una herramienta multiplataforma ya que será desarrollada en el lenguaje Java de programación, permitiendo su ejecución en cualquier sistema operativo.
- Será una herramienta WYSWYG, facilitando al máximo el proceso de diseño de los diagramas.
- Será empaquetada en un paquete .JAR para su fácil ejecución desde cualquier sistema operativo, liberando al usuario de la necesidad de un instalador, esta opción llevara el grado de portabilidad al máximo, pudiéndose ejecutar desde cualquier carpeta.

Para la interfaz se ha planteado el siguiente esquema:



En esta figura se aprecia una ventana o JFrame, que será la ventana principal, en ella se encuentra un JInternalFrame con un proyecto DFD abierto. En esta ventana hija, se puede apreciar un objeto JTree y un Canvas. Se pretende que al momento de editar un DFD la jerarquía de los procesos creados se actualice automáticamente en el JTree.

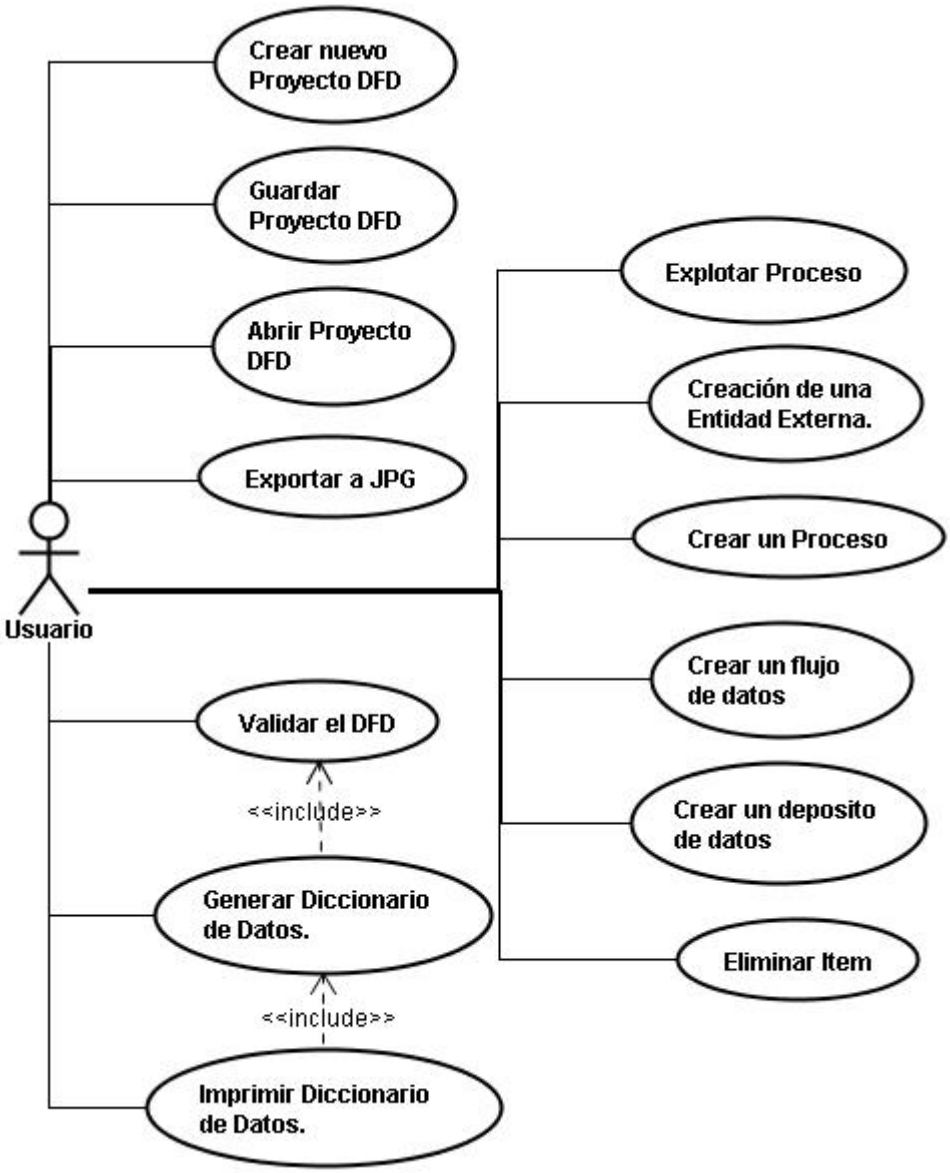
Además. Se encuentra desplegado el menú principal, donde se encuentran las opciones básicas del programa.

Por ultimo, se muestra una pequeña JToolBar, en la cual se encuentran las opciones de diseño, entre ellas: creación de procesos, entidades, depósitos,

flujos, la opción de modificar el nivel de la escala de visualización y un botón de ayuda.

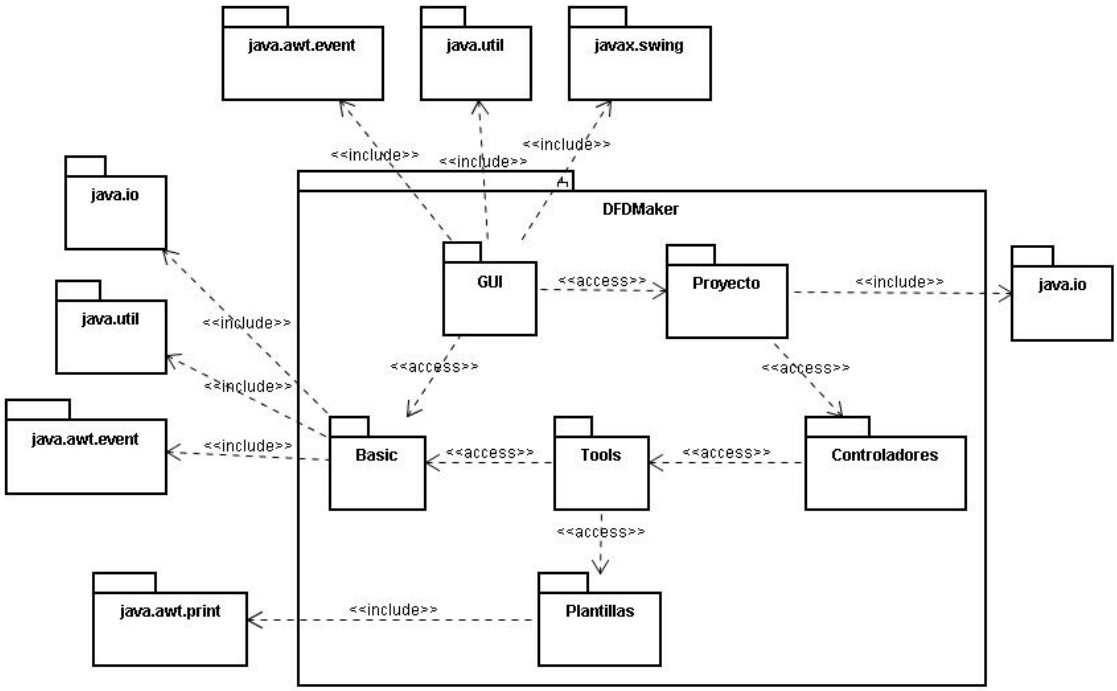
8.1. DIAGRAMA DE CASOS DE USO

Figura 4. Diagrama de Casos de Uso.



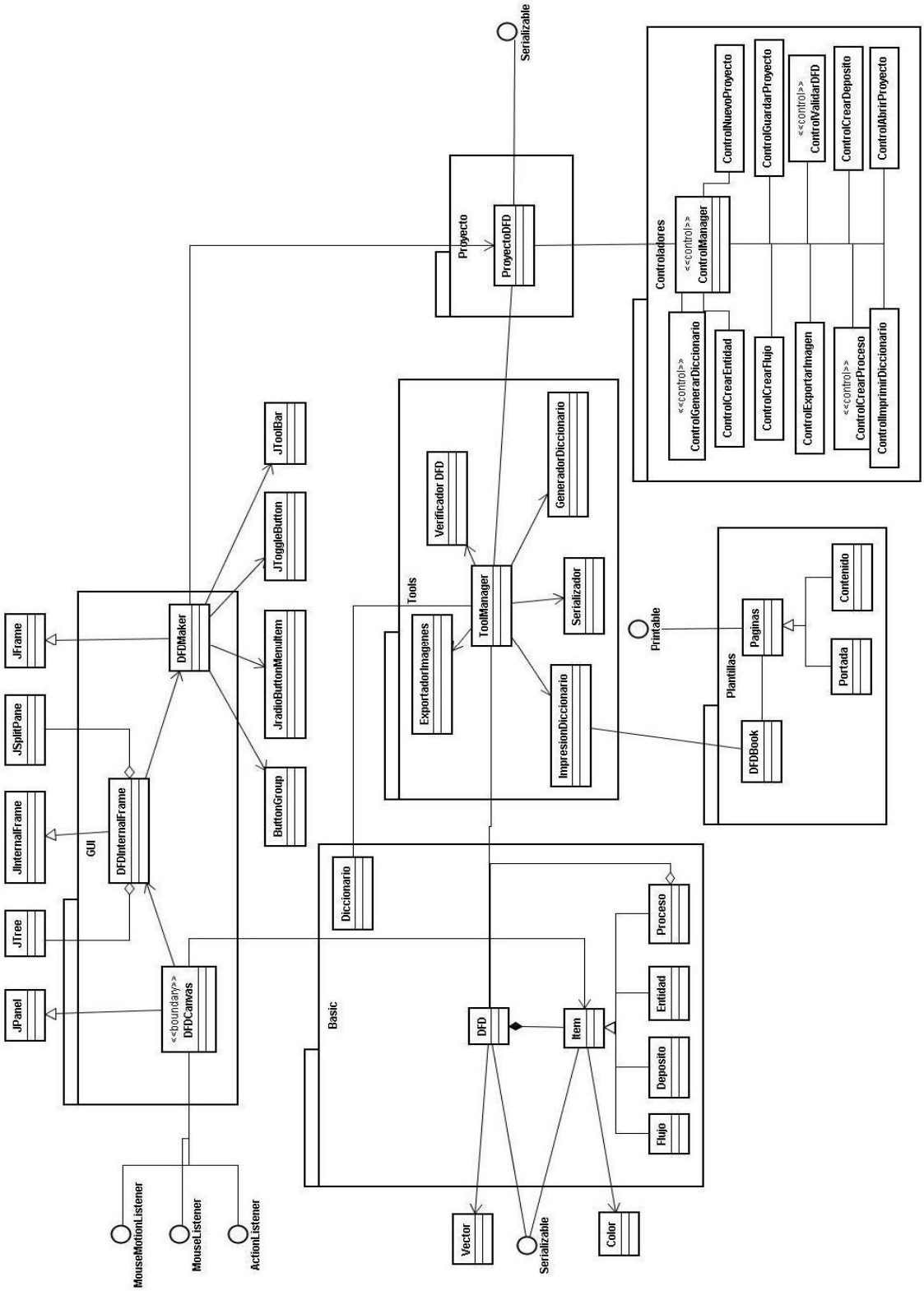
8.2. DIAGRAMA DE PAQUETES

Figura 5. Diagrama de Paquetes.



8.3 DIAGRAMA DE CLASES.

Figura 6. Diagrama de Clases.



8.3.1 Descripción Diagrama de Clases

En la figura anterior planteamos lo que será la base de nuestra herramienta de modelado. En este diagrama se encuentran los paquetes principales junto con sus clases y las relaciones entre ellas, las cuales se describen a continuación:

- **Paquete Basic**: Este paquete encapsula las clases que representan cada una de las clases encargadas de representar los ítems que componen un diagrama de flujo: Procesos, Entidades, Depósitos y Flujos de Datos.

    - **Clase Item:** Esta clase abstracta reúne las características comunes a todos los ítems de un DFD. Entre los atributos se encuentran: Cadenas de texto que describen su nombre, descripción, el color con que se dibujará, su número de identificación, la notación grafica a utilizar, tamaño del ítem y las coordenadas (x, y) que serán de gran utilidad en el proceso de ubicación y dibujo de los distintos ítems.

      En cuanto a los métodos, tiene el método abstracto `public void paint(Graphics g)`, el cual deberá ser sobrescrito por cada clase que herede de ella. Además se encontraran los

respectivos setters y getters de cada uno de los atributos anteriormente mencionados.

Esta clase además, implementa la interfaz ***Serializable***, gracias a esta podremos serializar el objeto DFD a un archivo en disco, permitiéndonos su posterior edición y manejo. Cabe aclarar que debido a la implementación de esta interfaz, ya no seria necesario que sus clases hijas la implementan, esto debido a la herencia.

- **Clase Entidad:** Esta clase es la representación grafica y lógica de una Entidad externa. Esta clase hereda de Item y recibe todos sus atributos. Esta y todas las clases que hereden de Item deberán implementar distintamente el `método public void paint(Graphics g).` En este método se le dará sus formas geométricas, dependiendo de la notación seleccionada, mediante los diferentes métodos que provee la clase `Graphics2D.`

- **Clase Deposito: .**Como se muestra en el modelo de clases, esta clase hereda también de Ítem. Como se explicó anteriormente ella debe sobrescribir el método `paint(Graphics),` donde se le dará su forma geométrica dependiendo de la notación preferida: La Notación Yourdon/Coad o la Gane/Sarson.

Entre los atributos únicos de esta clase se encuentra un campo de texto para señalar que tipo de depósito de datos es: texto, clase de datos o archivo físico; capos para describir la estructura de la tabla en caso de ser una base de datos, e información sobre su ubicación en el disco duro o en alguna red de computadores.

- **Clase Proceso:** Esta clase encapsula los datos relacionados con cada uno de los procesos, Implementa el método `paint(Graphics)` distinto a las otras clases que heredan de Ítem.

  Esta clase es un poco más compleja que las anteriores, en esta se encuentra una instancia de la clase DFD, que representa la anidación de capas presente en cualquier DFD. Puede tener máximo una instancia de este objeto en caso de no ser un proceso de último nivel.

- **Clase Flujo:** Esta clase comprende lo relacionado con los flujos de datos. Esta clase es muy compleja, ya que dependerá de los objetos que ya existen dentro del DFD.

  Posee coordenadas adicionales, en caso de que se quiera dibujar la línea del flujo con varios quiebres.

- **Clase DFD**: Esta clase actúa como un contenedor de ítem y representa al Diagrama de Flujos en su totalidad. En el se encuentra un `Vector` genérico de Ítems en donde se van agregando cada uno de los ítems del DFD.

  Esta clase también posee el método public void `paint(Graphics g)`. En este método será el encargado de llamar a cada uno de los métodos paint de los ítems presentes en el vector, mediante un recorrido por su vector. De esta forma se simplifica el proceso de dibujo de todos los ítems.

  Esta clase posee los métodos necesarios para agregar, eliminar y cambiar de posición a los ítems del vector.

  Además implementa la interfaz `Serializable` por la necesidad de guardar este objeto al disco duro.

- **El paquete GUI:** Este paquete será el encargado de contener las clases que comprenderá todo lo relacionado con la interfaz con el usuario. Consta principalmente de tres clases: DFDMaker, DFDInternalFrame y DFDCanvas.

  - **Clase DFDMaker:** Esta clase será el punto de entrada a la aplicación. Es una clase que hereda de `JFrame` y que consta de

diversos objetos de interfaz visual, tales como: Un `JMenuBar`, `JMenuItems`, `JToolBar`, `JToggleButtons`, entre otros; los cuales permitirán una interacción amigable con el usuario.

Dentro del `JMenuBar` se encontraran los menús habituales de Archivo donde se encuentran las opciones de: Nuevo, Abrir, Guardar, Imprimir, Salir entre otros. Además de los menús de Herramientas, Ayuda y Edición.

El `JToolBar` permitirá al usuario seleccionar el `JToggleButton` asociado con cada uno de los ítems de un DFD, para su posterior inclusión al proyecto.

Otro Componente importante son los `JInternalFrames`, los cuales se almacenan dentro de un vector. Estos `JInternalFrames` se describen a continuación.

- **Clase DFDInternalFrame:** Esta clase hereda de `JInternalFrame`. Mediante este objeto, se le brindará al usuario la posibilidad de editar varios proyectos de construcción de DFDs al mismo tiempo.

Las distintas instancias de esta clase residirán en un vector ubicado en el `JFrame` que los contiene, el cual fue descrito previamente.

Esta ventana hija será el contenedor de tres componentes adicionales. `JSplitPane, JTree` y la clase fundamental DFDCanvas.

El objeto de la clase JSplitPane permitirá dividir el área del JInternalFrame en dos mitades: El lado izquierdo se destinará a ubicar un JTree y el lado derecho el objeto DFDCanvas.

El JTree nos dará la funcionalidad de visualizar los distintos niveles que presente el DFD. En donde cada una de las hojas del árbol representa la capa de un nivel de DFD. Cada vez que se adicione un proceso, automáticamente se agregará una hoja dentro de la jerarquía del árbol. Luego de ser adicionado el proceso, el usuario podrá hacer clic en la hoja correspondiente, para realizar la expansión de ese proceso.

- **Clase DFDCanvas:** Probablemente una de las clases más complejas dentro del Diseño de la aplicación. Esta clase es componte visual el cual heredad de JPanel. Este panel nos

servirá de lienzo en donde podremos dibujar y editar los distintos elementos del DFD,

Esta clase tiene diversas características. En ella se maneja una instancia de un objeto tipo DFD. Esta clase, como alguna de las anteriores, tiene el método `public void paint(Graphics)` en donde se encargará de llamar al método paint del objeto DFD pasándole el objeto `Graphics` a este método. Después de declarado este método, podremos repintar automáticamente nuestra área de edición en cualquier momento mediante el llamado al método `repaint()` del DFDCanvas.

Esta clase deberá implementar las distintas interfaces relacionadas con los eventos del cursor del Mouse. Las cuales son: `MouseMotionListener`, `MouseListener` y `ActionListener`.

Mediante la implementación de los métodos proveídos por estas interfaces podremos crear, editar y manipular los ítems ubicados en el DFDCanvas, gracias a los métodos que manipulan estos eventos, como lo son: `mouseClicked(MouseEvent e)`, `mousePressed(MouseEvent e)`, `mouseDragged( MouseEvent e)`.

- **El Paquete Tools:** Este paquete tendrá las clases necesarias para efectuar las operaciones más especializadas de la aplicación, entre ellas encontramos: La serialización, impresión, manejo de imágenes, entre otras.

    - **Clase ToolManager:** Esta clase será la encargada de administrar cada una de las clases incluidas en este paquete. Este objeto tendrá una referencia al objeto DFD que se encuentre editando, y lo pasará como parámetro a la clase respectiva en el momento en que el usuario invoque alguna de estas operaciones.

      Su función es la de crear el objeto requerido para el uso de alguna herramienta, esto es actúa como un Proxy facilitándole servicios al DFD.

    - **Clase ExportadorDeImágenes:** Esta clase es la encargada de renderizar el DFD en una imagen y exportarlo como tal en un archivo, en su responsabilidad se encuentra el formato calidad, posibles marcos, etc.

    - **Clase Serializador:** Esta clase es la encargada de darle persistencia al proyecto serializándolo a disco, irrecuperándolo del mismo cuando se necesite editar un proyecto existente.

- **Clase VerificadorDFD:** Esta clase es la encargada de verificar la creación de los flujos, como también la verificación general del DFD de manera que cumpla con las mínimas reglas de creación.

- **Clase GenerarDiccionario:** Esta es la clase encargada de la creación del diccionario a partir de las descripciones y atributos de las entidades procesos y flujos del DFD.

- **Clase ImpresionDiccionario:** Esta es la clase encargada de la creación de las páginas y formateado del diccionario de datos para su impresión, su responsabilidad es crear los objetos correspondientes e imprimir el Diccionario de datos.

- **El Paquete Controladores:**
  - **Clase ControlManager:** esta clase es el Proxy de los controladores, su función es crear las distintas clases de controles de la aplicación, manejar los eventos generados por el usuario y delegar la respuesta a estos al controlador indicado y luego adicionar este control creado al DFD.

- **Clase ControlCrearDeposito:** Esta clase es la encargada de la creación de un item Deposito verificando sus atributos sean los indicados.

- **Clase ControlCrearEntidad:** Esta clase se encarga de crear correctamente una entidad.

- **Clase ControlCrearProceso:** Esta clase es la encargada de la creación correcta de un proceso el control debe verificar el nivel del de profundidad del proceso, si este es un proceso final o se trata de un proceso extensible en el diagrama.

  Esta clase también es la encargada durante la explosión de un proceso de crear un nuevo DFD con los flujos y atributos del proceso y asociado a ella, también deberá notificar al correspondiente al ControlManager la actualización de la jerarquía en el JTree, y la visualización del nuevo DFD en el canvas.

- **Clase ControlCrearFlujo:** Esta clase es la encargada de crear correctamente un flujo, siendo este uno de los controladores con más trabajo deberá notificar al ControlManager de la actualización del canvas a medida que el flujo es creado, y verificar su validez una vez terminado, esta comprobación se hace a través del control ValidarDFD.

- **Clase ValidarDFD:** Este control es el encargado de la validación de todo DFD, su función es prestar los servicios de validación del DFD como un todo y por sus componentes de manera que validaciones sobre ítems específicos puedan ser realizadas transparentemente.

  Esta validación tiene lugar cada vez que explícitamente el usuario la requiera, cada vez que se cree un nuevo flujo y cuando se trata de generar el diccionario de datos. El proceso de esta validación incluye el recorrido de todos los ítems en el DFD revisando que todos aquellos ítems del mismo nivel cumplan las normas mínimas de creación. Esto es se recorre el arreglo de ítems, se identifican cuales intervienen en el proceso y se chequea la operación

- **Clase ControlCrearNuevoProyecto:** Esta es la clase encargada de cerciorarse que de la correcta creación de un nuevo proyecto, incluye en su responsabilidad el manejo del serializador para escribir el proyecto a disco.

- **Clase ControlAbrirProyecto:** Esta es la clase encargada de restablecer un proyecto previamente guardado, es el encargado de deserializar el archivo en disco y restablecer el DFD.

- **El paquete Plantillas:**
    - **Clase DFDBook:** Esta clase es la encargada del diseño de impresión general del diccionario de datos.

    - **Clase Pagina:** Esta clase representa de una manera general las páginas incorporables en el DFD.

    - **Clase Portada:** Esta clase que extiende a Página, es la representación de lo que es una página de portada para el diccionario de datos.

    - **Clase Contenido:** Esta clase que extiende a Página, es la representación de las páginas internas del diccionario de datos.

8.4. DIAGRAMAS DE SECUENCIA

Figura 7. Diagrama de Secuencia Crear Nuevo Proyecto

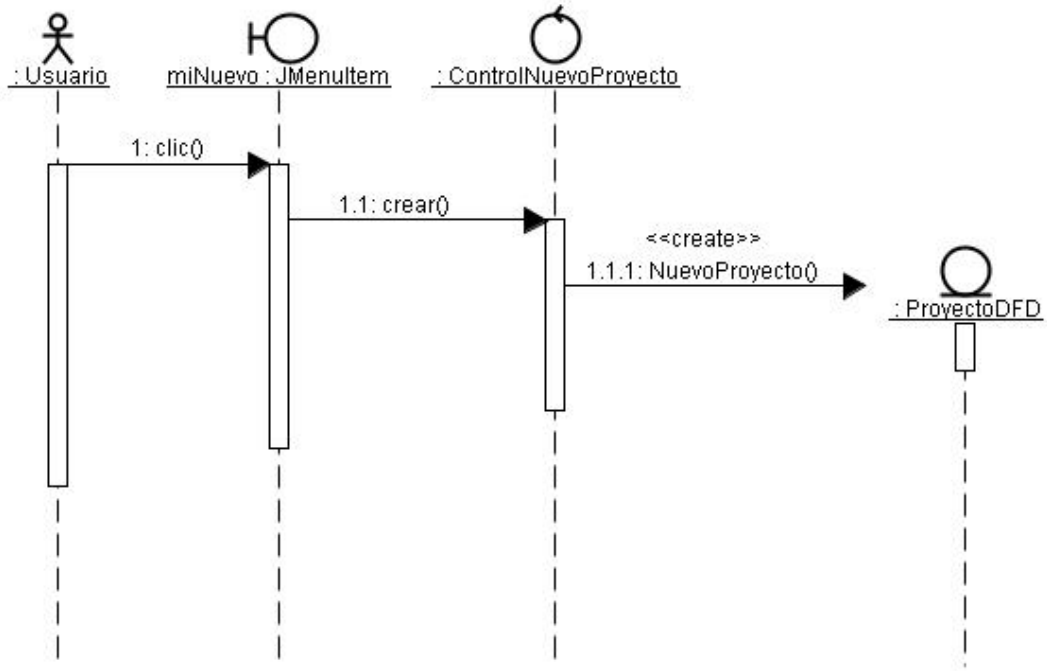


Figura 8. Diagrama de Secuencia Guardar Proyecto.

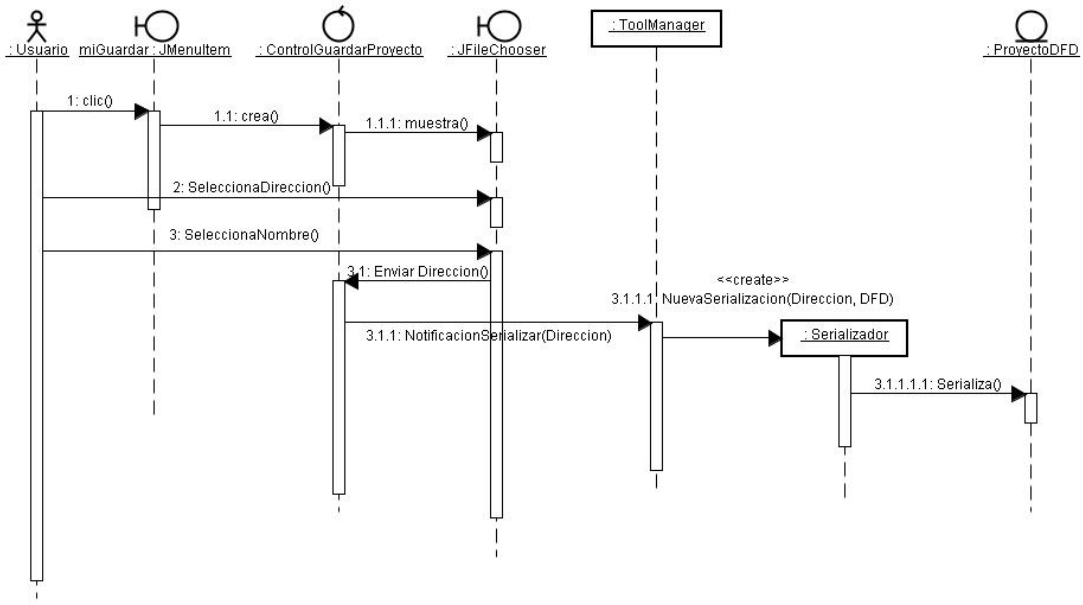


Figura 9. Diagrama de Secuencia Abrir Proyecto.

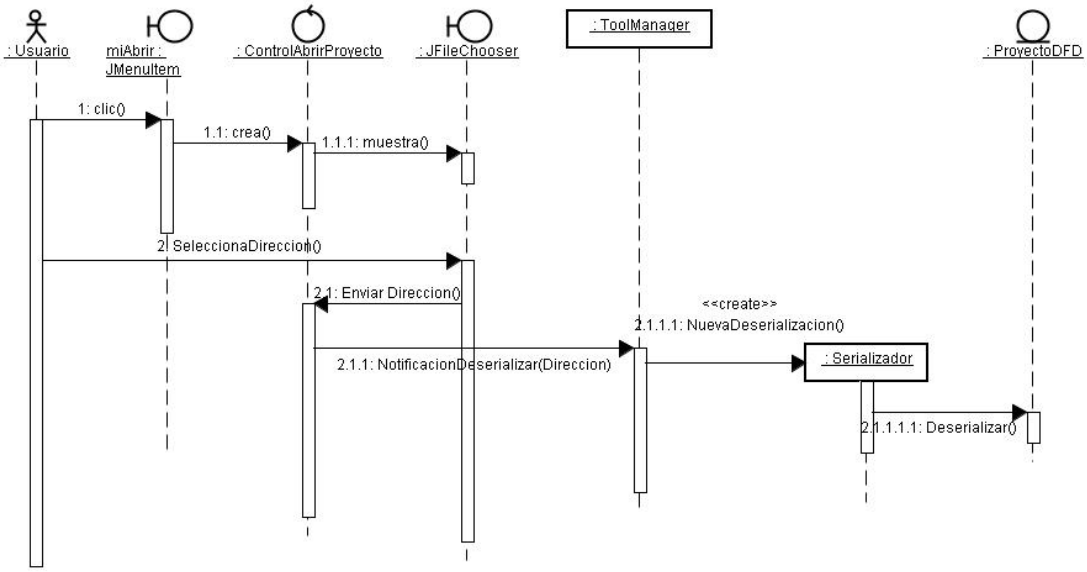


Figura 10. Diagrama de Secuencia Crear Nueva Entidad.

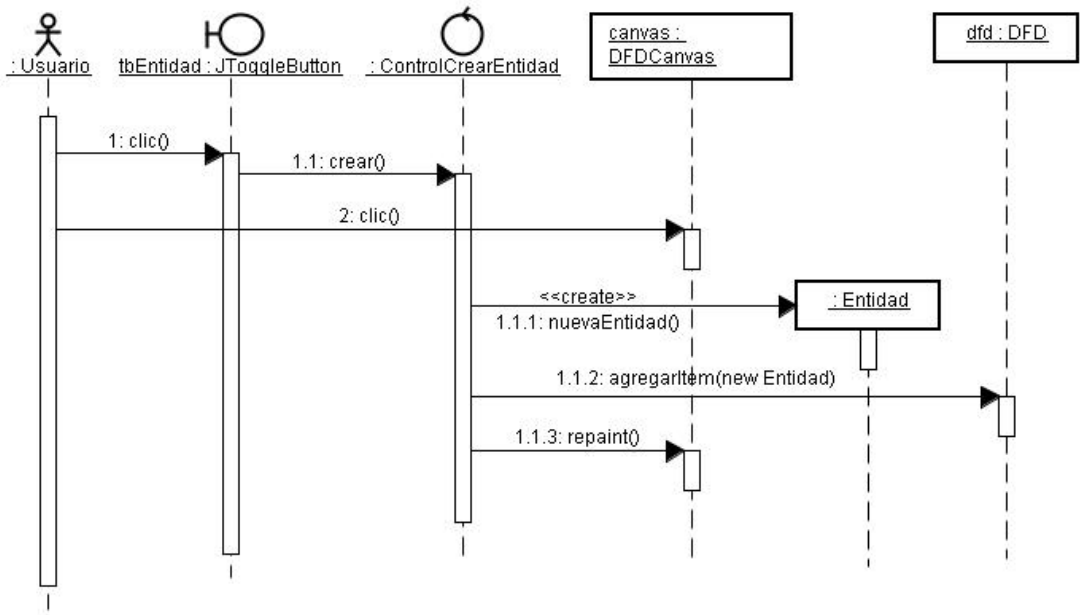


Figura 11. Diagrama de Secuencia Crear Nuevo Proceso.

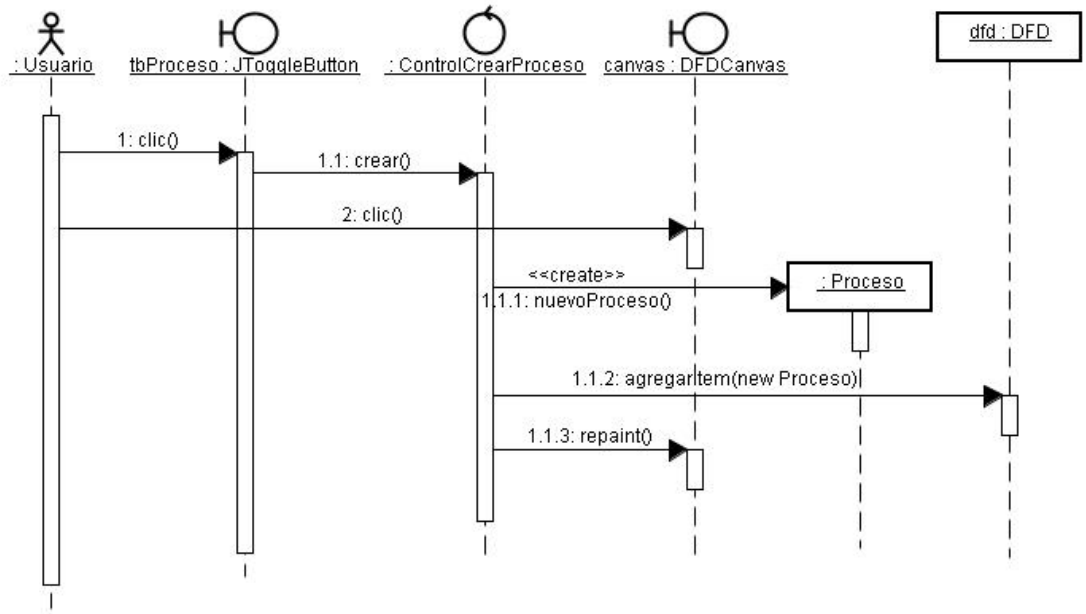


Figura 12. Diagrama de Secuencia Crear Nuevo Flujo

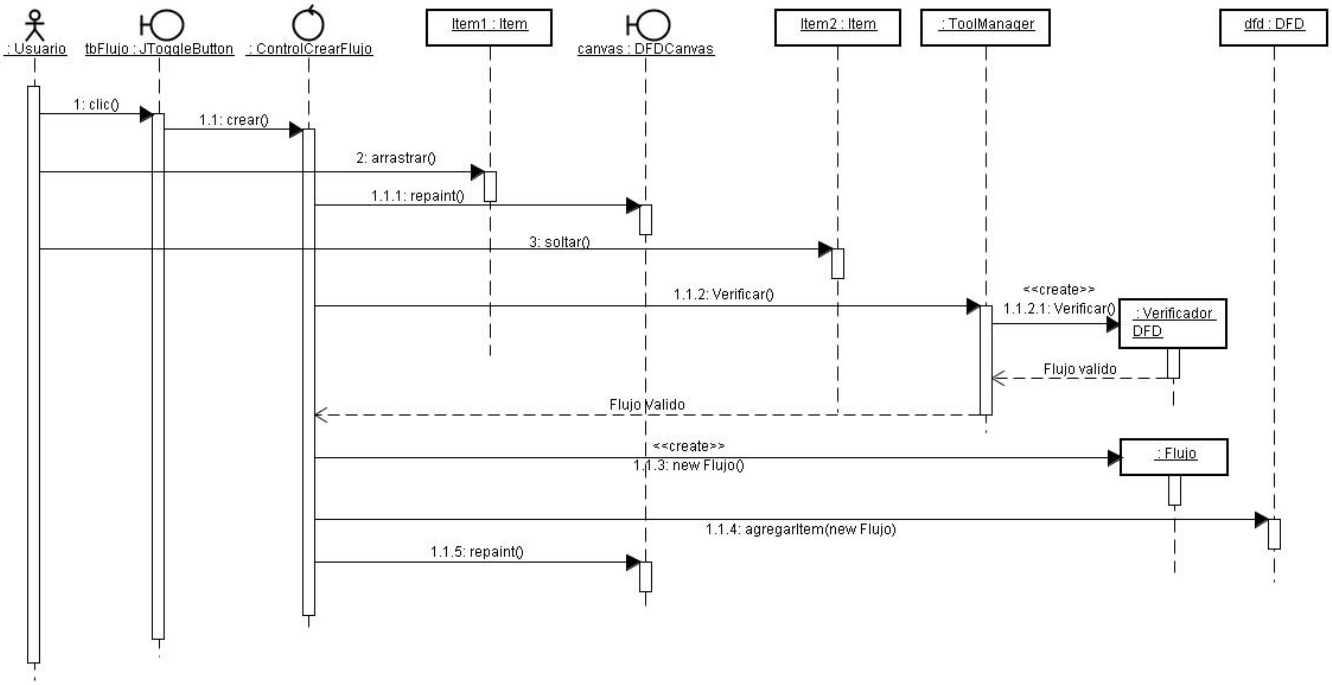


Figura 13. Diagrama de Secuencia Crear Nuevo Deposito.

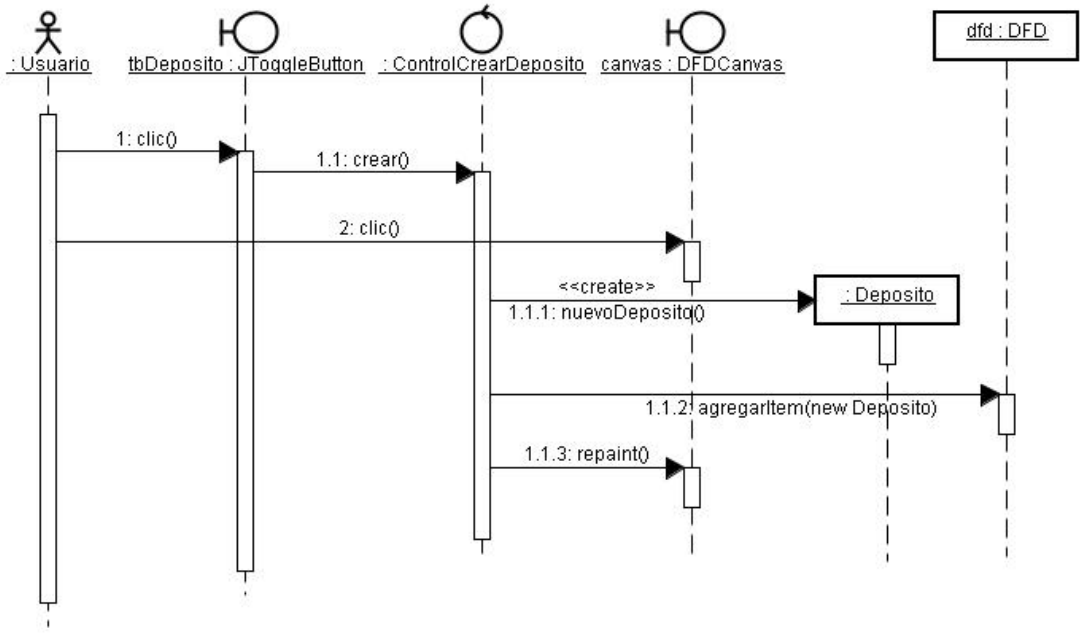


Figura 14. Diagrama de Secuencia Generar Diccionario de Datos.

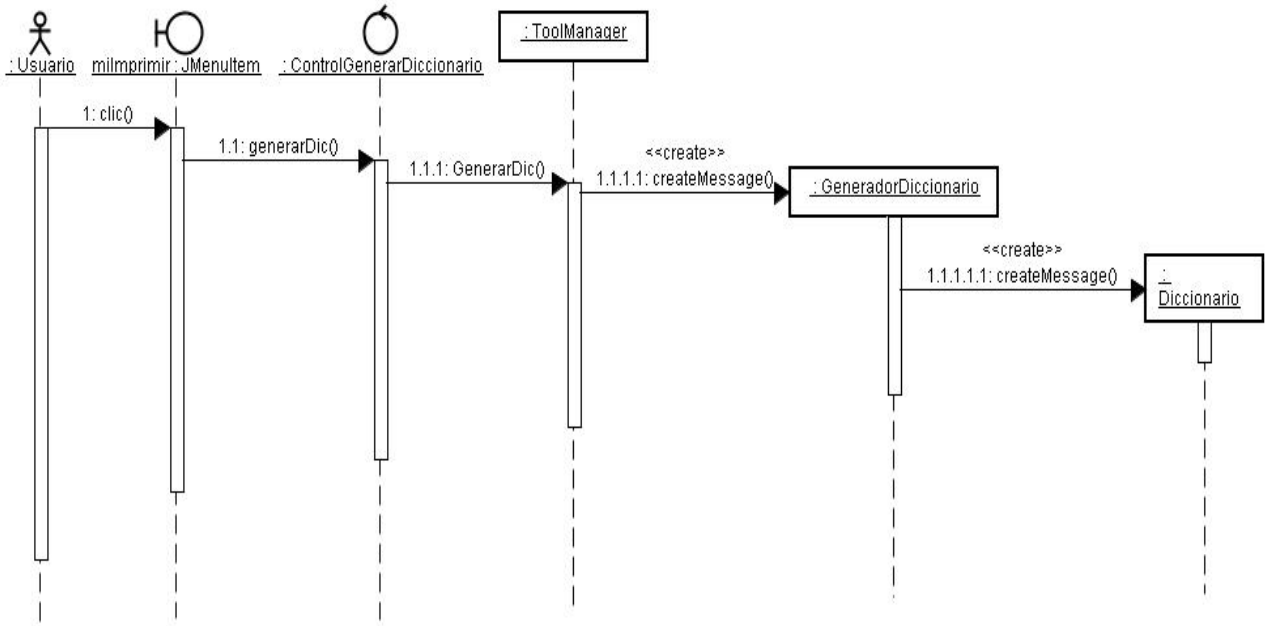


Figura 15. Diagrama de Secuencia Imprimir Diccionario de Datos.

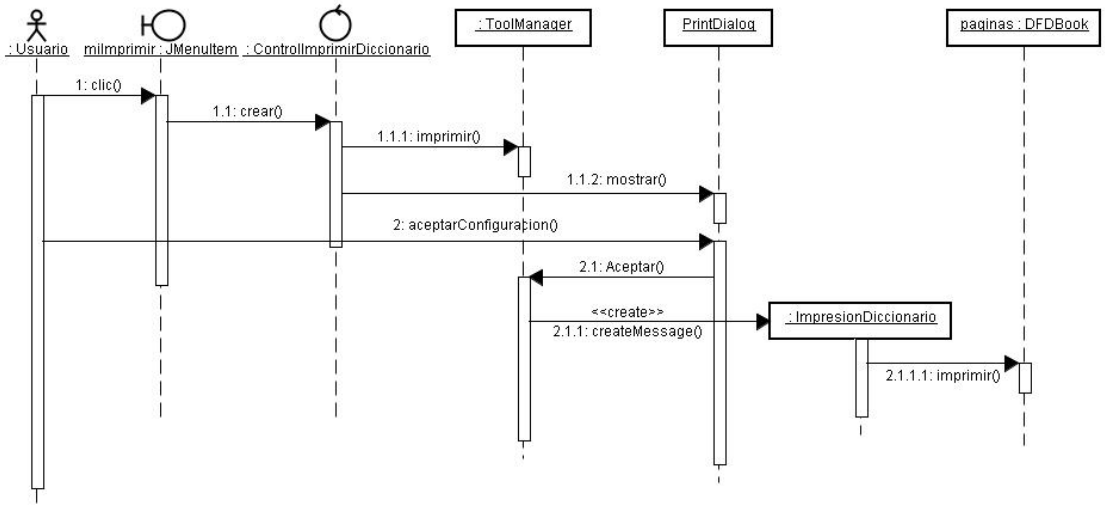


Figura 16. Diagrama de Secuencia Exportar Imagen.

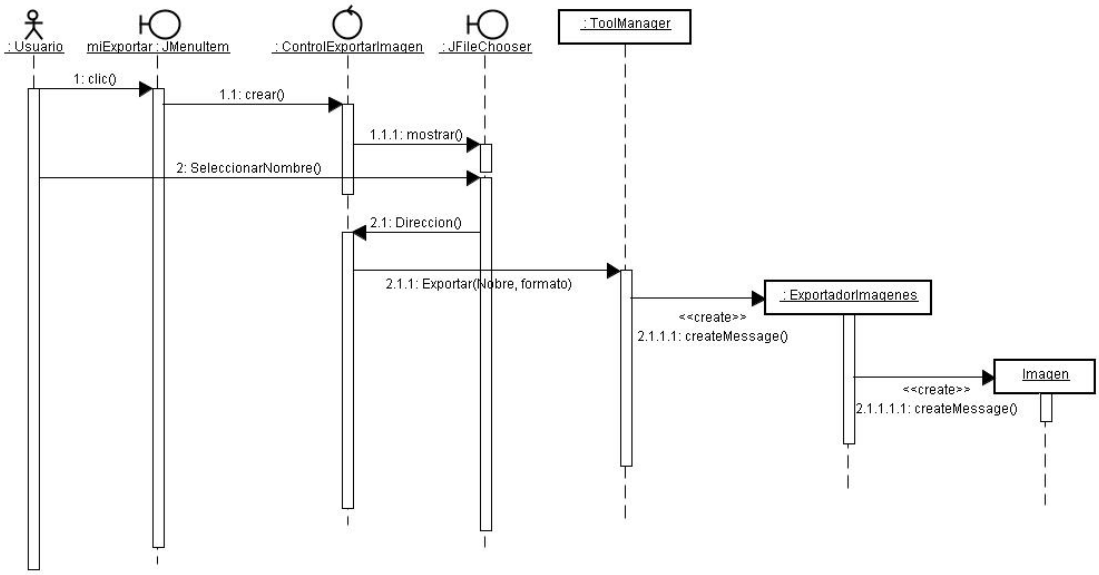


Figura 17. Diagrama de Secuencia Explotar Proceso.

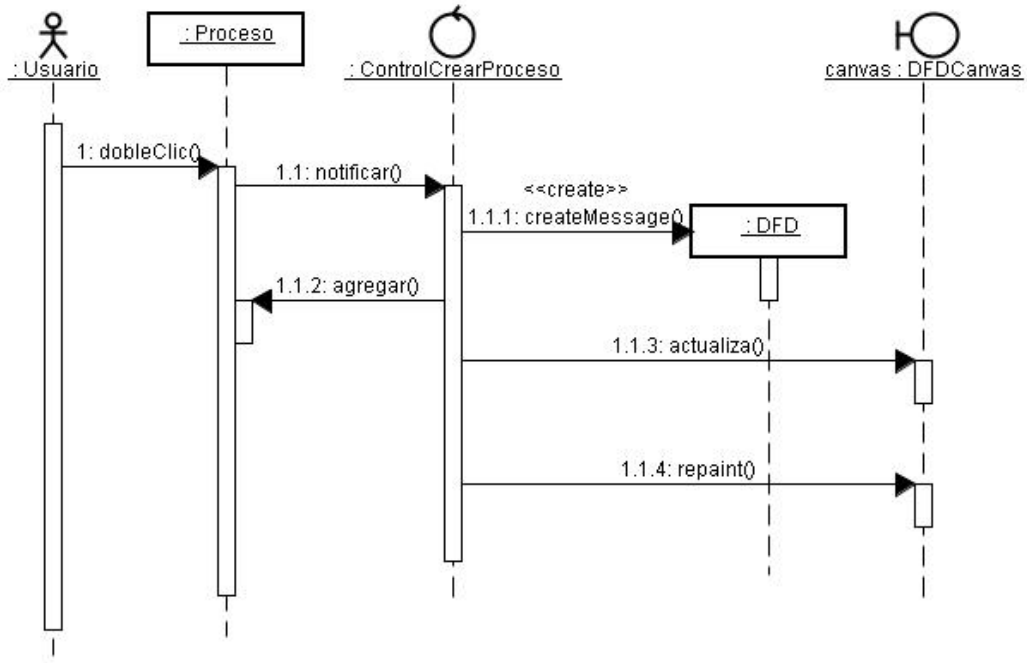
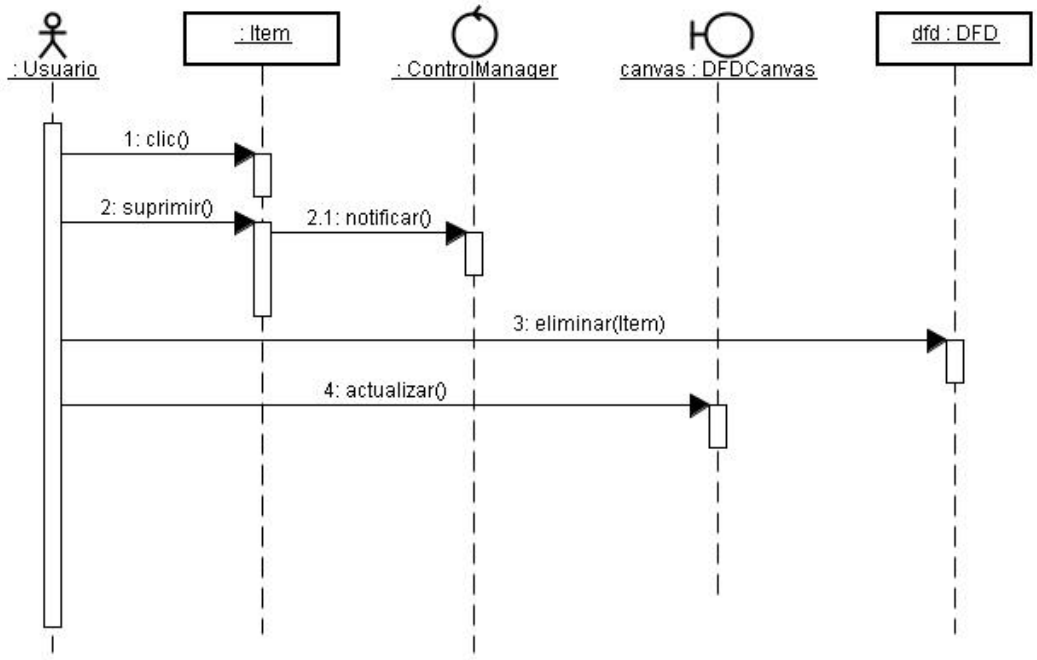


Figura 18. Diagrama de Secuencia Eliminar Ítem.



9. CONCLUSIONES

Gracias a este trabajo investigativo, hemos apreciado la importancia las diferentes técnicas de análisis y diseño de sistemas de información, incluyendo el análisis estructurado, el cual es el objetivo de la herramienta, hasta las nuevas metodologías como UML y el Proceso Ágil Unificado (AUP) y las técnicas de desarrollo actuales como lo la programación extrema (XP), las cuales se han vuelto indispensables en la disciplina de la Ingeniería de Software durante transcurso de los últimos años.

Entre los aportes que brinda esta monografía se encuentra el hecho de haber establecido las bases del diseño de una herramienta CASE para el diseño estructurado, enfocado en la construcción de los DFD y generación del Diccionario de datos, considerados como los pilares fundamentales del análisis estructurado.

Este modelado y diseño contiene no solamente los fundamentos para construir la aplicación, si no que es una fuente de información y consulta para aquellos individuos que se interesen en la aplicación de técnicas de modelado con el enfoque del análisis estructurado, el cual ha sido utilizado desde muchos años atrás obteniendo excelentes resultados.

Este trabajo contiene además sugerencias sobre todos los elementos que permitirán acelerar el proceso de implementación, a cualquier equipo de desarrollo que se embarque en la tarea de llevar a cabo este proyecto de ingeniería de software.

Desde el punto de vista tecnológico, introducimos al lector en las tecnologías Java de gran utilidad para éste, y cualquier otro proyecto relacionado con el diseño y desarrollo de aplicaciones que requieran de programación visual, la cual está muy bien respaldada por la API de Java.

En general este trabajo es la recopilación y puesta en práctica de todos los conceptos y teorías vistas en las asignaturas de Ingeniera de Sistemas y el Minor de Ingeniería de Software; aplicados conjuntamente para concretar el diseño de este proyecto

10. RECOMENDACIONES

Para llevar a cabo este proyecto de ingeniería de software es absolutamente indispensable contar las herramientas CASE de diseño y desarrollo disponibles hasta la fecha.

Debido a la naturaleza de este proyecto, Para alumnos Por los Alumnos, recomendamos el uso de herramientas de carácter libre. Empezando por programas de diseño UML como Fujaba y JUDE, esta última reducirá sustancialmente el salto del diseño a la implementación, mediante la importación y exportación de código Java; junto con la documentación generada por éste.

Adicionalmente sugerimos el uso de Eclipse IDE. El cual es estos momentos uno de los entornos de desarrollo integrado mas completos y versátiles en la actualidad. Esta herramienta proporcionara opciones útiles, como lo son la refactorización, generación de código y el correcto uso del lenguaje de programación Java.

En cuanto a la metodología, sería útil el adopte del Proceso Ágil Unificado, mediante el cual se aplicarían las actividades esenciales como la obtención de requerimientos, programación extrema y la ejecución frecuente de pruebas.

11. BIBLIOGRAFIA

DEITEL, Harvey M. y DEITEL, Paul J., Como programar en Java. Méjico: Pearson Education, 2004. 1325p.

PENDER, Tom. UML Bible. Estados Unidos: Wiley Publishing, Inc., 2003. 940p.

HORSTMANN, Cay, CORNELL, Gary. Core Java2 Fundamentals. Estados Unidos: Sun Microsystems Press, 2005. 762p.

BRUEGGE, Bernd, DUTOIT, Allen H. Ingeniería de software Orientado a Objetos. Estados Unidos: Prentice Hall, 2002. 553p.

SOMMERVILLE, Ian. Ingeniería de software. 6a. ed. Addison-Wesley Iberoamericana.

PRESSMAN, Roger S. Ingeniería del software: Un enfoque practico, 3a ed. McGraw-Hill.

Java World Tutorials – Online. 2000, disponible en Internet:

http://www.javaworld.com/javaworld/jw-10-2000/jw-1020-print-p1.html

Java Serialization – Online, disponible en Internet:

http://www.mactech.com/articles/mactech/Vol.14/14.04/JavaSerialization/

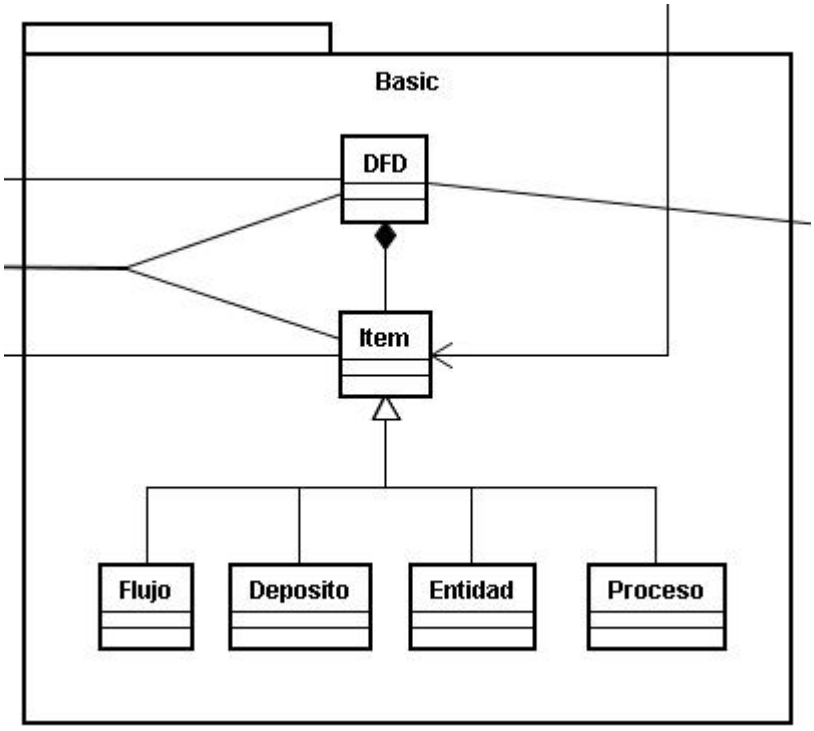
Java2D Tutorial – Online. Disponible en Internet:

http://java.sun.com/docs/books/tutorial/index.html

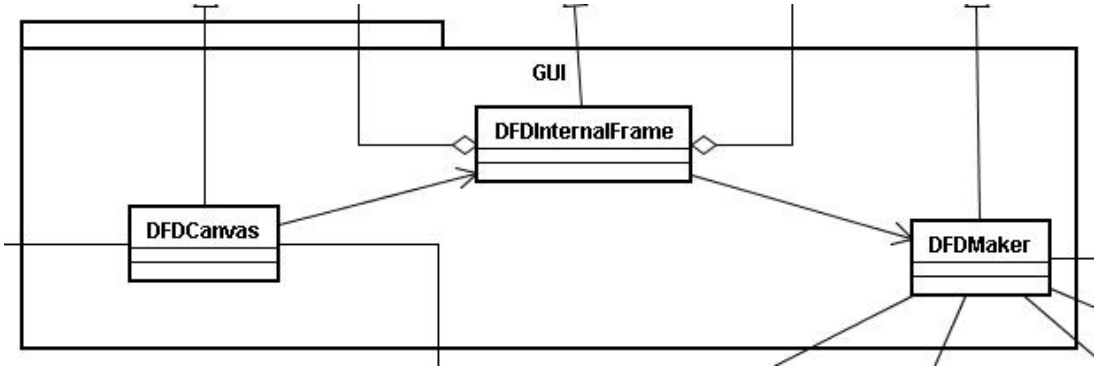
12. APENDICES Y ANEXOS.

Diagrama de Clases por paquetes.

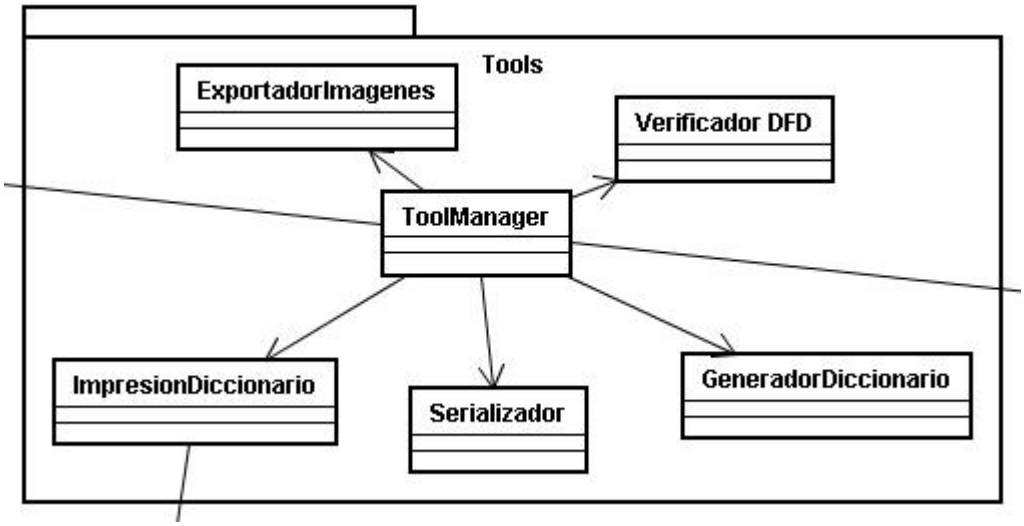
Paquete Basic



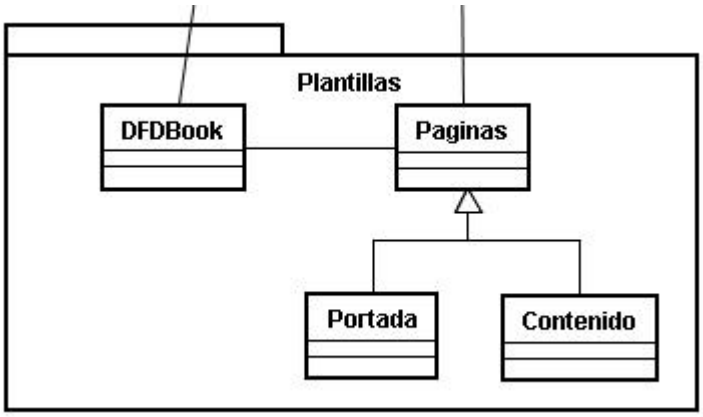
Paquete GUI



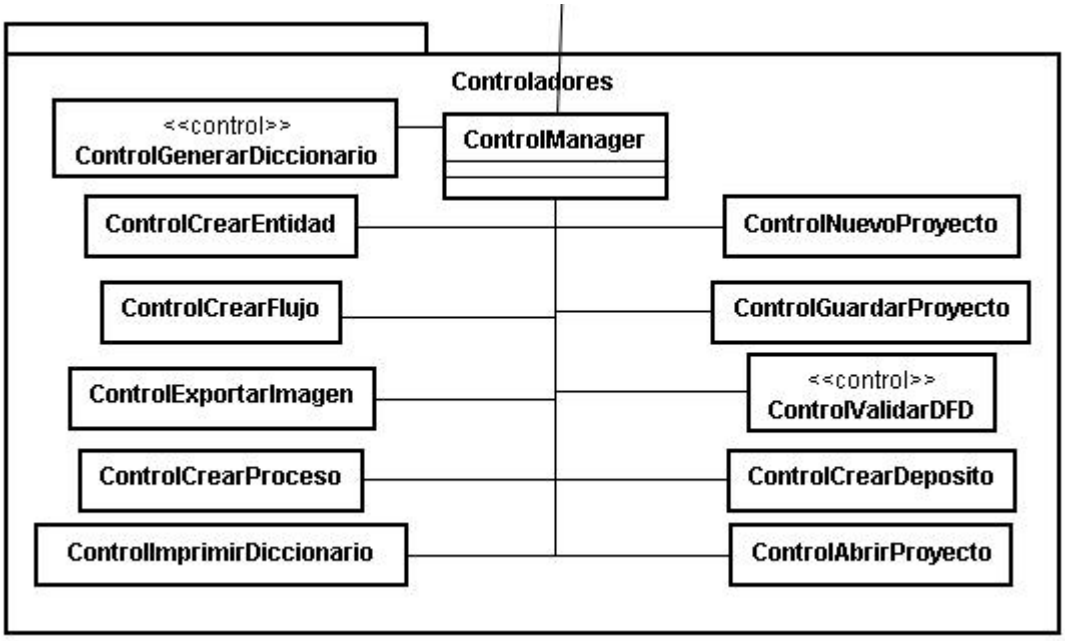
Paquete Tools



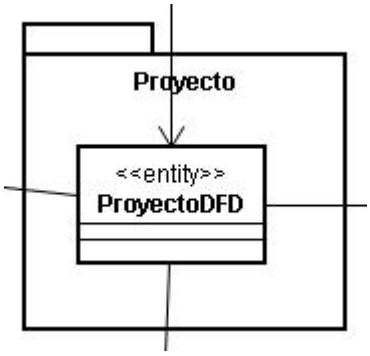
Paquete Plantillas



Paquete Controladores



Paquete Proyecto



GLOSARIO

API: (Application Programming Interface) Interfaz de Programación de Aplicaciones, es un conjunto de especificaciones de comunicación entre componentes software.

Representa un método para conseguir abstracción en la programación, generalmente entre los niveles o capas inferiores y los superiores del software.

Es el conjunto de rutinas del sistema que se pueden usar en un programa para la gestión de entrada/salida, gestión de ficheros etc.

AWT: (Abstract Windowing Toolkit) Paquete de Java que contiene todas las clases necesarias para crear interfaces y el dibujado de graficas e imágenes.

CASE: Computer Assisted Software Engineering. Colección de herramientas y técnicas que brindan aumentos en la productividad del analista y del programador. Las dos tecnologías entregadas prominentes son los generadores de aplicaciones y los sitios de trabajo basados en PCs que proporcionan una automatización gráficamente orientada del proceso del desarrollo.

DFD: Diagrama de Flujo de Datos

Diccionario de Datos: (DD) Una herramienta automatizada para recoger y organizar la información detallada sobre componentes del sistema. Los diccionarios de los datos mantienen instalaciones para documentar elementos de datos, expedientes, programas, sistemas, archivos, usuarios, y otros componentes del sistema. Un diccionario de los datos es un depósito para las descripciones de los tipos de datos o de los tipos del objeto. Los diccionarios

de los datos se utilizan para almacenar definiciones de, por ejemplo, tipos del artículo y sus cualidades.

Graphics2D: Clase de Java que extiende de Graphics. Provee un control mas sofisticado sobre la geometría, coordenadas, transformaciones, manejo del color y ubicación del texto. Es la clase fundamental para renderizar figuras bidimensionales, texto e imágenes sobre la plataforma Java.

Hash: Función o método para generar claves o llaves que representen de manera unívoca a un documento, registro, archivo, etc.

JAVA: Java es una plataforma de software desarrollada por Sun Microsystems, de tal manera que los programas creados en ella puedan ejecutarse sin cambios en diferentes tipos de arquitecturas y dispositivos computacionales. Lenguaje de programación orientado a objeto

Java2D: biblioteca gráfica de bajo nivel para Java, formando parte a la vez de JFC y Java Medios de comunicación. Permite trazar toda clase de figuras geométricas en 2 dimensiones.

JDK: Java Development Kit. Es un paquete de programación de software (SDK) para producir programas en Java. El JDK está desarrollado por la división JavaSoft de Sun Microsystems.

RAM: Random Access Memory: Memoria de acceso aleatorio. Memoria donde la computadora almacena datos que le permiten al procesador acceder rápidamente al sistema operativo, las aplicaciones y los datos en uso.

Serialización: En Java, la serialización es el almacenar el estado actual de un objeto en cualquier medio del almacenamiento permanente para una reutilización más tarde. Se hace esto cuando se utiliza la interfaz serializable o cuando usa las clases de ObjectOutputStream y de ObjectInputStream.

Serializable: El uso de esta interfaz permite que los objetos de esa clase se puedan conservar su estado cuando son enviados a través de un stream.

SerialVersionUID: Identificador universal de la versión para una clase de Serializable. En la deserialización se utiliza este número para asegurarse de que una clase cargada corresponde exactamente a un objeto serializado. Si no se encuentra ningún identificador, entonces se lanza una excepción de tipo InvalidClassException.

Stream: Representación conveniente para la lectura y escritura los datos sin importar si los datos vienen de un archivo, de una conexión de red, de la consola, de otro hilo, o de otra fuente.

Transient: En Java, las variables declaradas de este tipo no se guardan o se restauran por el mecanismo estándar de restauración. Este proceso quedará en manos del programador

UML: es un lenguaje estándar para especificar, visualizar, construir, y documentar los artefactos de los sistemas de software, estandarizados por el OMG (grupo de gerencia de objetos). UML simplifica el proceso complejo del diseño del software usando modelos para la construcción del software. La adopción extensa de UML es una de las fuerzas que contribuyen a la demanda de los desarrolladores por las herramientas que pueden representar una información relacionada con el problema más intencional.