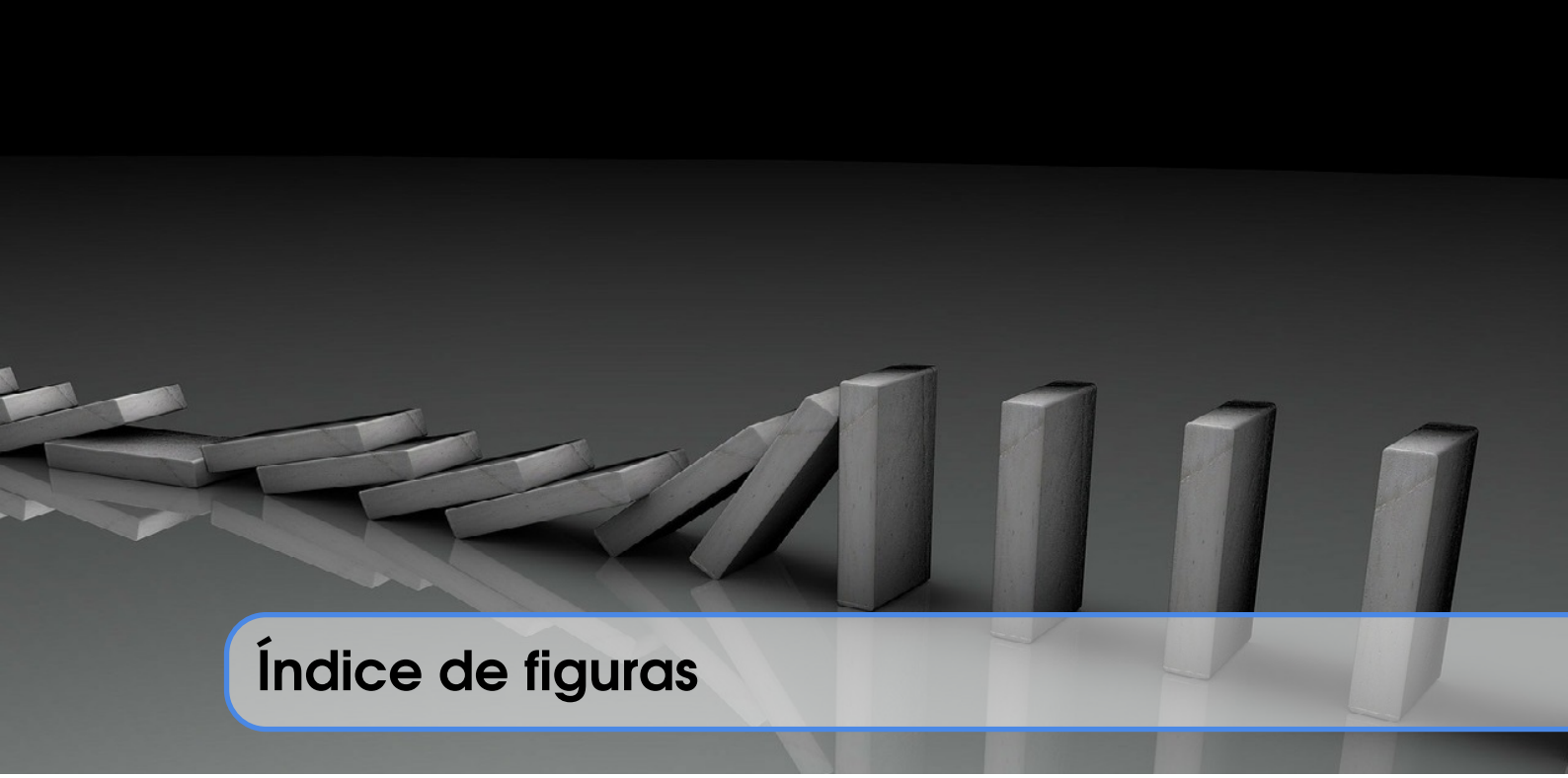
# GESTIÓN Y MANEJO DEL SISTEMA DE APOYO ALIMENTARIO DE LA UNIVERSIDAD DISTRITAL.

(nombre de la app aun no decidido)

Daniel Garcia Perea 20141020212
Daniel Antonio Moreno Ramirez 20132020620
Edwar Diaz Ruiz 20141020004
Universidad Distrital Francisco José de Caldas

# Índice general



# Índice de figuras

# I Contextualización

# 1. Problema

## 1.1 Problema

Dificultad que tienen los estudiantes para aprovechar el apoyo alimentario que ofrece la Universidad Distrital Francisco José de Caldas.

## 1.2 Objetivos

Crear una aplicación que permita agilizar el proceso de entrega de almuerzo Mejorar el servicio de apoyo alimentario Automatizar el proceso registro de asistencia de usuarios

**Pregunta: ¿Cómo agilizar el proceso de entrega de almuerzos en la Universidad ?**

## 1.3 Hipotesis:

Una estrategia para agilizar el proceso de entrega de almuerzos en la universidad es la construcción de un aplicativo web que permita la entrega de papeles, organización de horarios y la verificación de asistencia al apoyo alimentario.
Es necesario que a través del aplicativo se pueda organizar horarios ya que si solo se gestiona la asistencia y entrega de papeles no habra una solucion al problema a largo plazo pero si gestionan los horarios estudiantiles se pueden crear estrategias para evitar el aglutinamiento de personas a la hora de recibir el almuerzo.

## 1.4 Justificación:

En la Universidad Distrital Francisco José de Caldas existe un serio problema de gestión en la distribución de almuerzo a los estudiante: la lentitud en la entrega ha ocasionado que los estudiantes no puedan aprovechar el servicio sin correr el riesgo de comprometer la llegada a sus clases. Muchos estudiantes necesitan el apoyo alimentario pero el llegar tarde a clase da lugar incidentes entre profesores y alumnos lo que puede afectar, a su vez, el desempeño académico

# 2. Metodología

## 2.1 Manifiesto Agil

Antes de decir que es Scrum, vale la pena saber que es el manifiesto agil y de que se compone.

El manifiesto agil se compone por 5 valores y 12 Principios:

### 2.1.1 Valores

#### Valorar a las personas y las interacciones entre ellas por sobre los procesos y las herramientas

Las personas son el principal factor de éxito de un proyecto de software. Es más importante construir un buen equipo que construir el contexto. Muchas veces se comete el error de construir primero el entorno de trabajo y esperar que el equipo se adapte automáticamente. Por el contrario, la agilidad propone crear el equipo y que éste construya su propio entorno y procesos en base a sus necesidades.

#### Valorar el software funcionando por sobre la documentación detallada

La regla a seguir es "no producir documentos a menos que sean necesarios de forma inmediata para tomar un decisión importante". Estos documentos deben ser cortos y centrarse en lo esencial. La documentación (diseño, especificación técnica de un sistema) no es más que un resultado intermedio y su finalidad no es dar valor en forma directa al usuario o cliente del proyecto. Medir avance en función de resultados intermedios se convierte en una simple ïlusión de progreso".

**Valorar la colaboración con el cliente por sobre la negociación de contratos**

Se propone que exista una interacción constante entre el cliente y el equipo de desarrollo. Esta mutua colaboración será la que dicte la marcha del proyecto y asegure su éxito.

**Valorar la respuesta a los cambios por sobre el seguimiento estricto de los planes**

La habilidad de responder a los cambios que puedan surgir a lo largo del proyecto (cambios en los requisitos, en la tecnología, en el equipo, etc.) determina también su éxito o fracaso. Por lo tanto, la planificación no debe ser estricta sino flexible y abierta.

### 2.1.2 Principios

Los valores anteriores son los pilares sobre los cuales se construyen los doce principios del Manifiesto Ágil. De estos doce principios, los dos primeros son generales y resumen gran parte del espíritu ágil del desarrollo de software, mientras que los siguientes son más específicos y orientados al proceso o al equipo de desarrollo:

1. Nuestra mayor prioridad es satisfacer al cliente a través de entregas tempranas y frecuentes de software con valor.
2. Aceptar el cambio incluso en etapas tardías del desarrollo. Los procesos ágiles aprovechan los cambios para darle al cliente ventajas competitivas.
3. Entregar software funcionando en formafrecuente , desde un par de semanas a un par de meses, prefiriendo el periodo de tiempo más corto.
4. Expertos del negocio y desarrolladores deben trabajar juntos diariamente durante la ejecución del proyecto.
5. Construir proyectos en torno a personas motivadas, generándoles el ambiente necesario, atendiendo sus necesidades y confiando en que ellos van a poder hacer el trabajo.
6. La manera más eficiente y efectiva de compartir la información dentro de un equipo de desarrollo es la conversación cara a cara.
7. El software funcionando es la principal métrica de progreso.
8. Los procesos ágiles promueven el desarrollo sostenible. Los sponsors, desarrolladores y usuarios deben poder mantener un ritmo constante indefinidamente.
9. La atención continua a la excelencia técnica y buenos diseños incrementan la agilidad.
10. La simplicidad –el arte de maximizar la cantidad de trabajo no hecho- es esencial.
11. Las mejores arquitecturas, requerimientos y diseños emergen de equipos auto-organizados.
12. A intervalos regulares, el equipo reflexiona acerca de cómo convertirse en más efectivos, luego mejora y ajusta su comportamiento adecuadamente.

## 2.2 Scrum

Scrum es una de las metodologia agiles mas conocidas al igual que XP, es una herramienta en la cual se tienen varios grupos de trabajo enfocados en diversas tareas con un unico objetivo, en el cual se espera generar resultados y/o entregas al cliente en cortos periodos de tiempo.

El equipo de trabajo en esta metodologia esta apoyado por 2 roles: El ScrumMaster y el Product Owner. el ScrumMaster o tambien consederado Coach es aquel que vela por que se utilice Scrum , por remover las impedimentos y da asistencia al equipo para que logre el mayor nivel de

rendimiento posible. El Product Owner es quien representa al negocio, stakeholders ( trabajadores, organizaciones sociales, accionistas y proveedores, entre muchos otros actores clave ), cliente y usuarios finales.

### 2.2.1 En que se basa

Esta basada en tres pilares:

Transparencia: Todos los implicados tienen conocimiento de qué ocurre y en el proyecto y cómo ocurre. Esto hace que haya un entendimiento “común” del proyecto, una visión global.

Inspección: Los miembros del equipo Scrum frecuentemente inspeccionan el progreso para detectar posibles problemas. La inspección no es un examen diario, sino una forma de saber que el trabajo fluye y que el equipo funciona de manera auto-organizada.

Adaptación: Cuando hay algo que cambiar, el equipo se ajusta para conseguir el objetivo del sprint. Esta es la clave para conseguir éxito en proyectos complejos, donde los requisitos son cambiantes o poco definidos y en donde la adaptación, la innovación, la complejidad y flexibilidad son fundamentales.

# II Diseño

# 3. Requerimientos

## 3.1 Casos de uso

El caso de uso es un marco de trabajo que busca hacer representar las acciones que un usuario dado de la aplicación puede hacer en ella, pero no en términos computacionales, en donde el desarrollador crea las funciones de la capa lógica y de persistencia, sino en un sentido mas natural para el, siendo esto los subproblemas derivados del problema principal.

Los casos de uso se crean para refinar un conjunto de requisitos de acuerdo con una función o tarea. En lugar de la tradicional lista de requisitos que quizá no trate de forma directa el uso de la solución, los casos de uso reúnen requisitos comunes basados en el tipo de función u objetivo. Los casos de uso definen qué harán los usuarios o funciones en la solución y un proceso empresarial define cómo realizarán esas funciones [2].

Para el diseño de casos de uso se debe seguir la siguiente nomenclatura:

- El usuario que va a usar la aplicación es una figura humanioide y debajo de este esta descrito el rol, que es su papel en el programa
- Cada acción del usuario se representa con un circulo y dentro se agrega una breve descripción de la funcionalidad.
- Para relacionar a un usuario con una acción en particular se usa una linea sin direcciones si la interacción es bidireccional, o con dirección de acuerdo a orientación de la operación
- Para relacionar dos casos de uso, es mediante una flecha segmentada, en donde hay dos tipos de relaciones principales:
  - **include:** Representa una relación de dependencia, en donde el caso de uso origen necesita del caso de uso de llegada.
  - **extend:** Describe una relación de necesidad opcional, en donde el caso de uso del que parte puede es opcional del caso de uso al que llega.

A en la figura 3.1 se puede visualizar el caso de uso destinado a este software, allí, se describe lo qué debe hacer el programa.

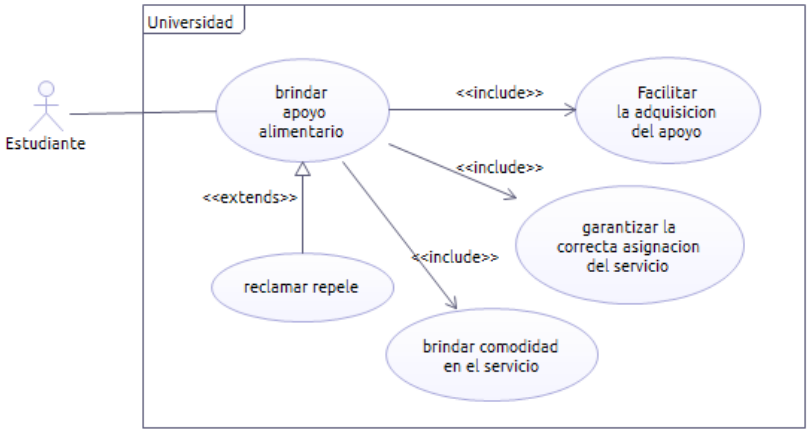



Figura 3.1: Diagrama de caso de uso

## 3.2 Requerimientos de Casos de uso

Los requerimientos de caso de uso entregan el plan de interacción entre el usuario y cada caso de uso: la descripción del caso de uso, los pasos para realizarse, las posibles fallas a las que puede incurrir, la prioridad y urgencia y comentarios acerca del mismo, todo para delimitar correctamente la relación usuario-aplicación

Cuadro 3.1: Caso de Uso CU-1

| | |
|---|---|
| **Nombre** | Brindar Apoyo Alimentario |
| **Actores** | Estudiantes |
| **Escenario** | |
| **Primario** | Brindar Apoyo Alimentario |
| **Secundario** | No coinciden Horarios |
| **Excepciones** | Solicitud de Inscripción no aceptada, Estudiante no Registrado, Se acabo el alimento, Ser retirado del beneficio por alguna falla |

Cuadro 3.2: Caso de Uso CU-2

| | |
|---|---|
| **Nombre** | Reclamar Repele |
| **Actores** | Estudiantes |
| **Escenario** | |
| **Primario** | Reclamar Repele |
| **Secundario** | |
| **Excepciones** | No sobro alimento, Estudiante no Registrado, Ser retirado del beneficio, La hora del reclamo no es la adecuada |

Cuadro 3.3: Caso de Uso CU-3

| | |
|---|---|
| **Nombre** | Brindar Comodidad en el Servicio |
| **Actores** | Administrativos, Estudiantes |
| **Escenario** | |
| **Primario** | Brindar Comodidad en el Servicio |
| **Secundario** | Presentar Inscripción en la fechas no estipuladas |
| **Excepciones** | Se presenta algún error en la Infreestructura, No se abríran convocatorias No se cuentan con los espacios o horarios disponibles. |

Cuadro 3.4: Casso de Uso CU-4

| | |
|---|---|
| **Nombre** | Garantizar Correcta Asignación del Servicio |
| **Actores** | Empleados y Administrativos encargados del apoyo |
| **Escenario** | |
| **Primario** | Garantizar Correcta Asignación del Servicio |
| **Secundario** | Presentarse en las horas no adecuadas, Presentar alguna inconveniente en un requerimiento del Apoyo |
| **Excepciones** | Fallos en la Infreestructura, Falla del Personal encargado de brindar el servicio, Inconvenientes presentados tanto en espacios como horarios habilitados para la entrega del servicio |

Cuadro 3.5: Caso de Uso CU-5

| | |
|---|---|
| **Nombre** | Facilitar Adquisición del Apoyo |
| **Actores** | Administrativos, Personal encargado de brindar el servicio |
| **Escenario** | |
| **Primario** | Facilitar Adquisión del Apoyo |
| **Secundario** | Presentar alguna inconsistencia en la inscripción o documentos que soportan la misma, |
| **Excepciones** | Fallos en la Infreestructura, No poder optar por el beneficio por falta de cupos, No tener el puntaje necesarios para recibir el beneficio |

# 4. Análisis del sistema e interacción

La interacción del sistema se puede visualizar mediante diagramas de secuencia, con los cuales se podrá hacer un análisis más clarificado sobre cómo es el ciclo de vida del programa y de qué forma son las relaciones internas de éste.

Para detallar un poco más sobre la comunicación de las partes del sistema, es indispensable un diagrama de comunicación, en donde se podrá detallar los que se transmite entre los objetos, es similar al diagrama de secuencia pero no describe un orden de transmisión de mensajes sino que se centra en los mismos mensajes dados.

Por ultimo, para aclarar cómo se compone el sistema, se explicarán los diagramas de clases, éstos describen cómo están conformadas las calases del sistema, cómo son sus relaciones y sobretodo, cuales son los patrones de diseño que se implementan en la aplicación y cómo optimizan el funcionamiento de la misma.

## 4.1 Diagramas de secuencia

Un diagrama de secuencia muestra una interacción, que representa la secuencia de mensajes entre instancias de clases, componentes, subsistemas o actores. El tiempo fluye por el diagrama y muestra el flujo de control de un participante a otro. Utilice diagramas de secuencia para visualizar instancias y eventos, en lugar de clases y métodos. En el diagrama, puede aparecer más de una instancia del mismo tipo. También puede haber más de una ocurrencia del mismo mensaje [1].

Un diagrama de secuencia es aquel en el que se muestra la interacción entre actores, objetos, interfaces, controles, bases de datos y otras entidades a través del tiempo según el enfoque, el cual puede ser:

- **Enfoque de objetos**: donde son participes las relaciones que existen entre éstos y con actores para realizar una tarea determinada.
- **Enfoque de modelado a tres capas**: donde el actor se comunica con una entidad interfaz, y ésta a su vez con una entidad control para llegar a una persistencia.

Es importante resaltar que cada diagrama de secuencia se realiza en torno a la descripción de un caso de uso dado, ya que éste define el orden en que se deban distribuir las tareas para cada una de las entidades que son participes a la hora de representar cada caso.

Para este proyecto se realizaron 5 casos de uso de los cuales se va a crear un diagrama de secuencia por cada caso y uno para cada enfoque. Inicialmente el caso de uso de .

A continuación se describirá cada diagrama de secuencia:

### Diagrama del control de asistencia

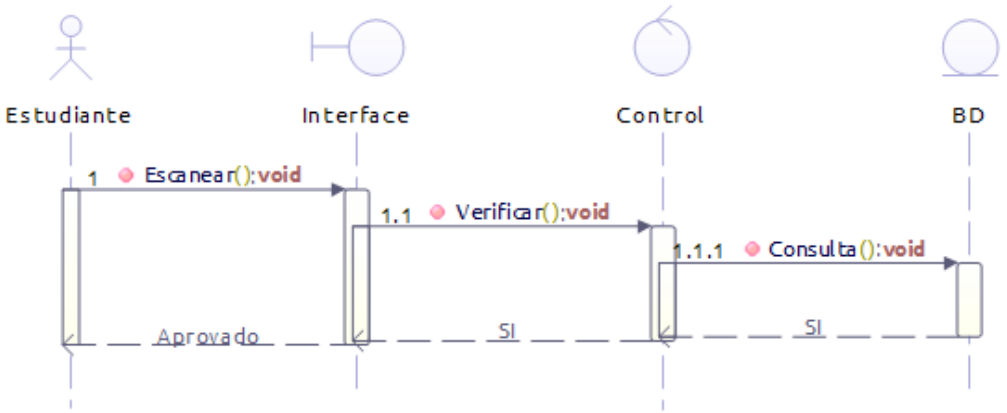



Figura 4.1: Diagrama de secuencia para el registro del control de asistencia

Este diagrama de secuencia explica el funcionamiento del programa frente al control de asistencia de los estudiantes al apoyo alimentario.
Para este caso el estudiante representa el usuario que ingresa al programa y que mediante el escaneo del codigo del carnet realiza la actualizacion de asistencia en la base de datos.

### Diagrama del Registro de usuarios

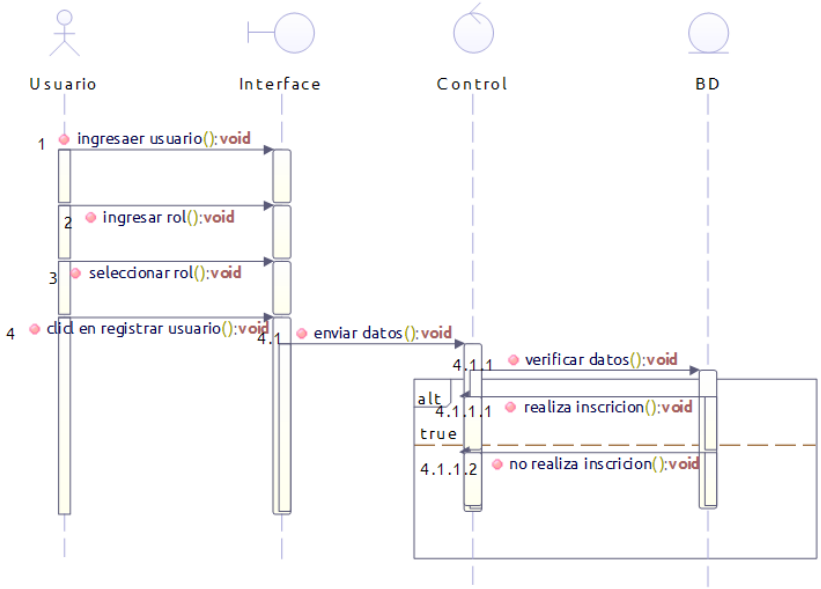



Figura 4.2: Diagrama de secuencia para el registro de usuarios

Este diagrama de secuencia representa el registro de un usuario. Para lo cual el usuario ingresa sus datos selecciona un rol y realiza el click para registrar los datos y consulta dentro de la base de datos la existencia del usuario y si existe se envia un mensaje que alerta sobre la existencia del usuario.

### Diagrama para crear nuevas convocatorias

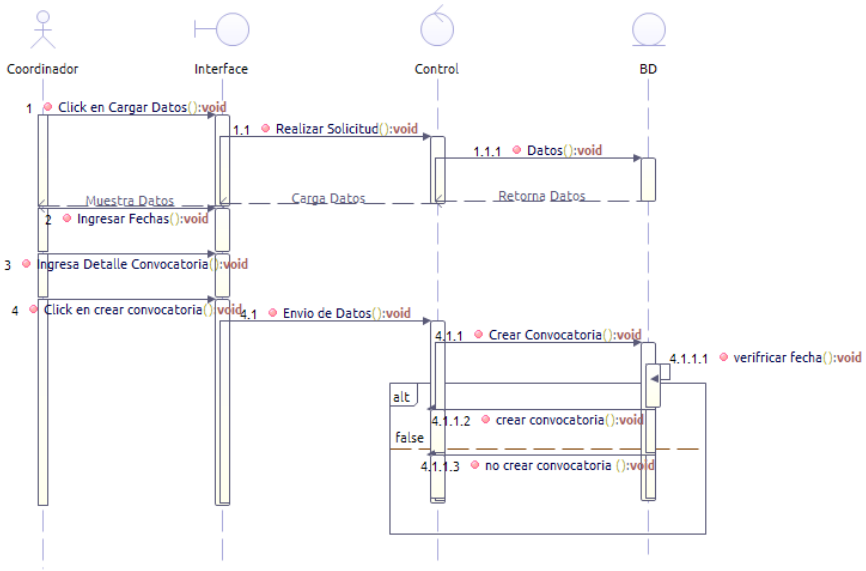



Figura 4.3: Diagrama de secuencia para crear una nueva convocatoria

Este diagrama de secuencia representa el registro de convocatorias de apoyo alimentario dentro del aplicativo.El programa registra los datos para los cuales nescesita realizar la carga datos, fechas y detalles especificos de cada convocatoria que al final mediante una consulta a la base de datos define si se puede crear o no la convocatoria.

### Diagrama del registro a convocatoria existentes

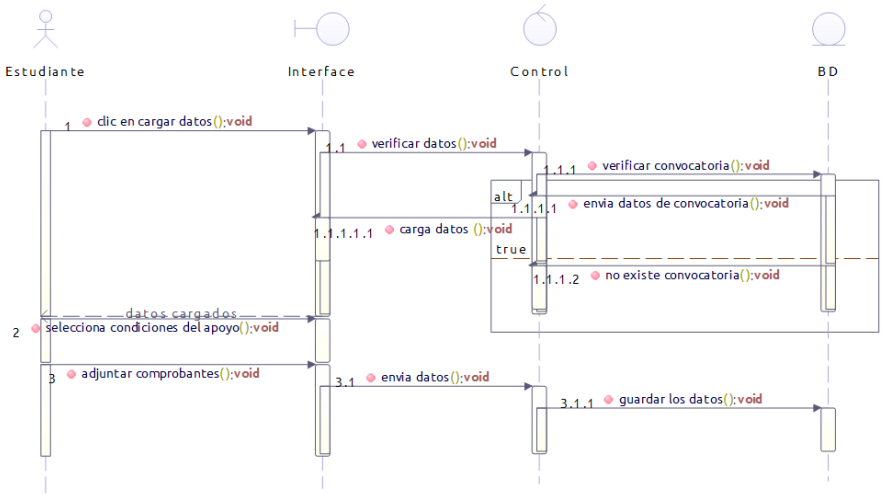



Figura 4.4: Diagrama de secuencia para el registro a una convocatoria

En este diagrama se realiza la solicitud de los usuarios a una nueva convocatoria. Para este proceso es nescesario que los usuarios carguen toda la informacion que luego se registra en la base de datos.

### Diagrama de la verificacion de una solicitud a una convocatoria

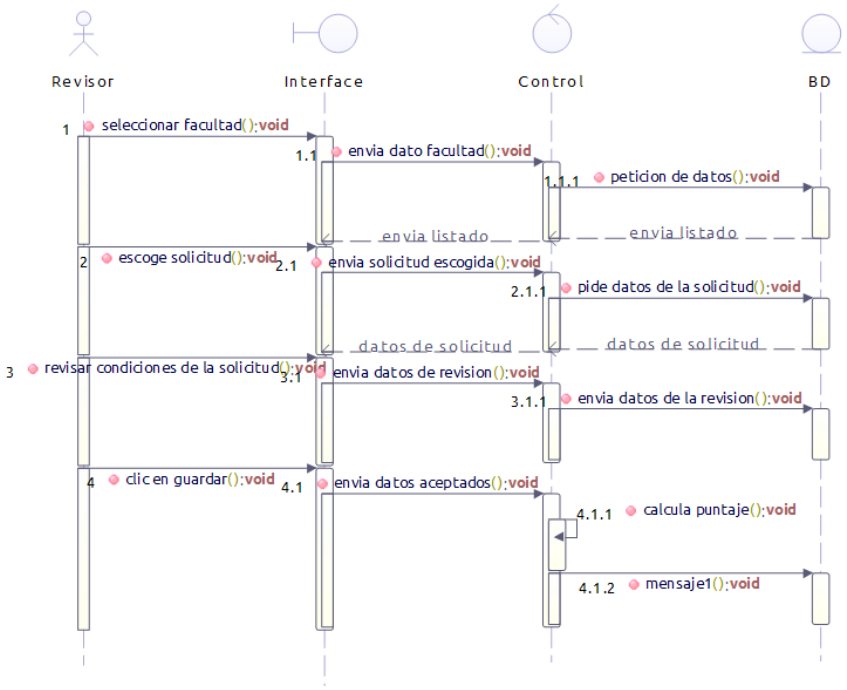



Figura 4.5: Diagrama de secuencia de verificacion de una solicitud a una convocatoria

Para este diagrama representamos la intereaccion que tiene un usuario revisor con las solicitudes presentes en la base de datos. En este caso el revisor realiza la seleccion de la facultad y se muestra un listado de las solicitudes de dicha facultad y se escoge la solicitud y consulta en la base de datos la solicitud escogida. Envia los datos de la revision y los datos aceptados a los cuales le calcula el puntaje y envia los datos a la base de datos

## 4.2 Diagramas de comunicación

También llamados Diagramas de Colaboración. Nos sirven para enfatizar los vínculos de datos entre los participantes de una interacción. Un diagrama de Comunicación modela las interacciones entre objetos o partes en términos de mensajes en secuencia. Los diagramas de Comunicación representan una combinación de información tomada desde el diagrama de Clases, Secuencia, y Diagrama de casos de uso describiendo tanto la estructura estática como el comportamiento dinámico de un sistema [3].

Los diagramas de comunicación tienen la función de simplificar la visualización de los modelos, dado que se enfocan exclusivamente en los objetos y su interacción, la cual se hace por medio de mensajes. Para seguir la lectura de un diagrama de comunicación se procede a ubicar el mensaje con la enumeración de primer orden, y se sigue la dirección indicada por la flecha adjunta, y así sucesivamente hasta llegar al último mensaje. Este tipo de diagrama se encuentra muy relacionado con el diagrama de secuencia, dado un isomorfismo entre éstos; Son equivalentes, pero tienen dos puntos de vista.

### Diagrama del control de asistencia

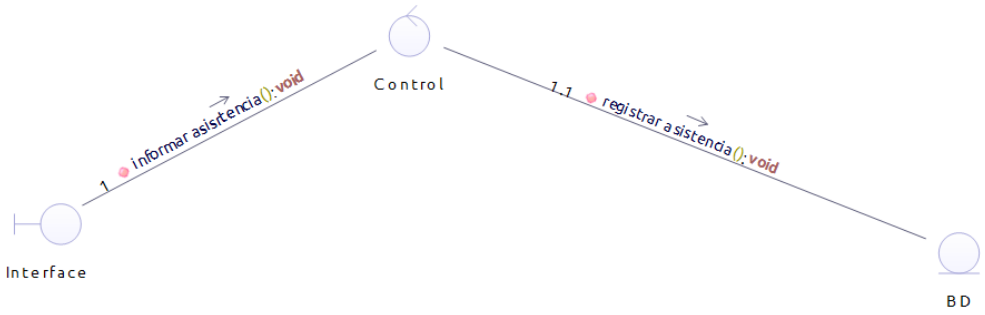



Figura 4.6: Diagrama de Comunicación del control de asistencia

En la figura 4.6 la interfaz envia el mensaje ïnformar asistencia().ªl objeto control que a su vez envia un mensaje regisrarasistencia().ª la base de datos.

### Diagrama del registro de usuarios

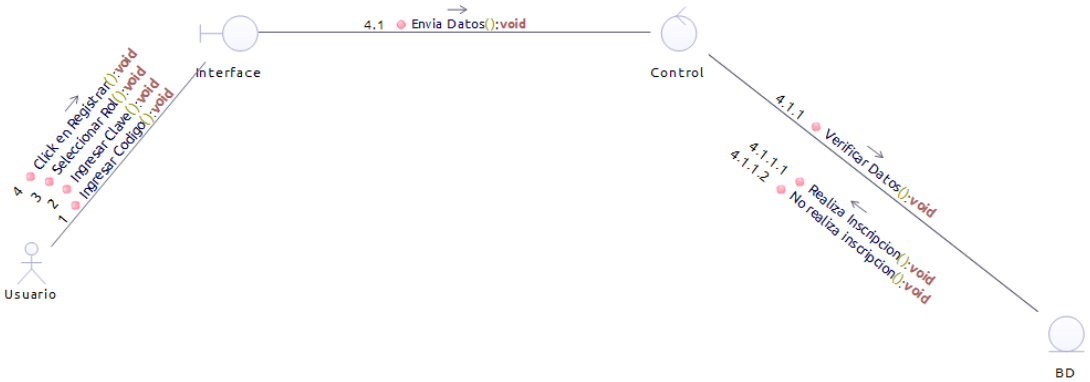



Figura 4.7: Diagrama de Comunicación del registro de usuarios

En la figura 4.7 el usuario envia los mensajes ÏngresarClave()", ÏngrearCodigo().ª la interface,la interface envia el siguiente mensaje .Enviar Datos().ªl control

### Diagrama de la creacion de una convocatoria

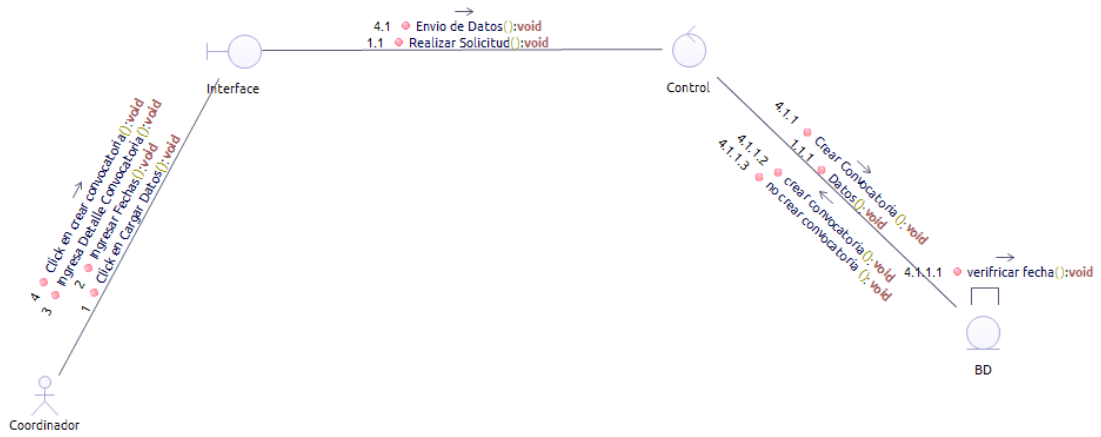



Figura 4.8: Diagrama de Comunicación de la creacion de una convocatoria

En la figura 4.8 el coordinador envia los datos a la interfaz mediante el mensaje çargarDatos()z la interface mediante .Envio Datos().envia los datos a la base de datos y mediante RealizarSolicitud().envia datos de la solicitud a la base de datos, la base de datos recive "datos()z se envia un mensaje de verificacion de datos "verificarfecha()".

### Diagrama para la solicitud a una convocatoria

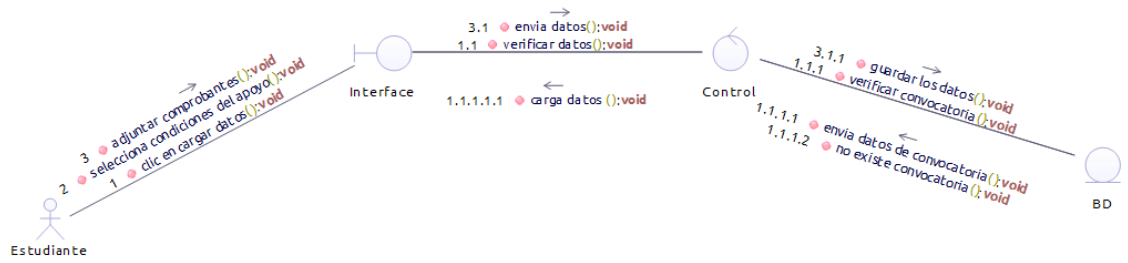



Figura 4.9: Diagrama de Comunicación para la solicitud a una convocatoria

Para la figura 4.9 el estudiante envia los datos para la solicitud de la convocatoria a la interfaz y la interfaz envia y recive mensaje a la base de datos

### Diagrama de la verificacion de una solicitud

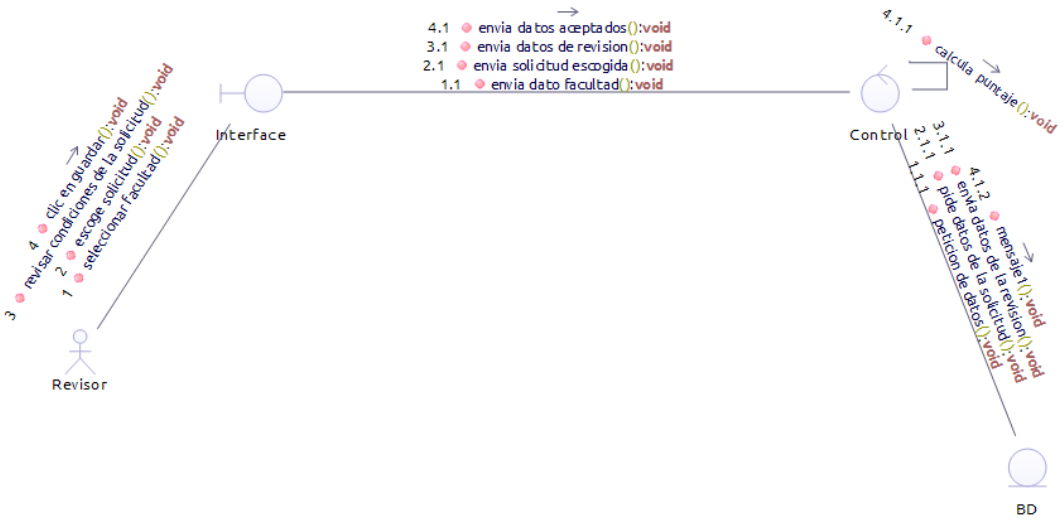



Figura 4.10: Diagrama de Comunicación de la verificacion de una solicitud

Para figura 4.10 el revisor realiza las peticiones de las solicitudes mediante la interfaz a la base de datos y revisa los datos de los solicitantes para que despues se calcule el puntaje

## 4.3 Diagramas de clases

Para una análisis orientado al desarrollo del aplicativo se plantean los diagramas de clases, que básicamente representa la interacción entre distintas entidades que en conjunto logran cumplir con la funcionalidad del programa. Los diagramas de clases se componen principalmente de:

- **Clase**: Esta es la entidad que representara un objeto que interviene en la solución del problema. Su ilustración es un rectángulo dividido en 3 niveles: En el primero esta el nombre, el segundo es para los atributos y el ultimo es para las operaciones de dicho objeto
- **Relación**: Las relaciones se encargan de mostrar como es la interacción entre las clases de un diagrama, permite la comunicación entre las mismas y se dividen en 4 fundamentalmente
  - ***Herencia***: Se encuentra en las clases cuando una clase .$^{es}$ unaçlase aparte ya definida, en donde la herencia permite realizar una especialización, o en sentido inverso, hacer una generalización de unas características o comportamientos.
  - ***Dependencia***: Se da cuando una clase depende de otra para su existencia o uso, pero la existencia de la primera no interfiere en la existencia de la segunda.
  - ***Asociación***: Ocurre cuando una clase puede usar otra sin que sea obligatorio inicializarla al iniciar la clase principal
  - ***Composición***: Es parecida a la asociación, sin embargo, esta relación hace obligatoria la inicialización de la clase agregada al momento de usar la clase principal.

Haciendo uso de las clases y relaciones en un diagrama se puede llegar a una solución para el problema en un momento especifico en el tiempo, pero hay que tener en cuenta que el software debe poder perdurar en el tiempo de forma limpia, sin necesidad de hacer cambios como tal en el código existente sino solo permitir la adición de componentes o clases, para ello se usan los patrones de diseño que no son mas sino estructuras de clases y relaciones enfocadas solucionar los problemas del mantenimiento del software, mejorando así el ciclo de vida de la aplicación.

En los siguientes diagramas se mostraran algunos patrones de diseño en nuestra aplicación enfocados a distintos problemas al momento de plasmar la aplicación en un diagrama de clases.

### 4.3.1 Creacionales

#### Builder

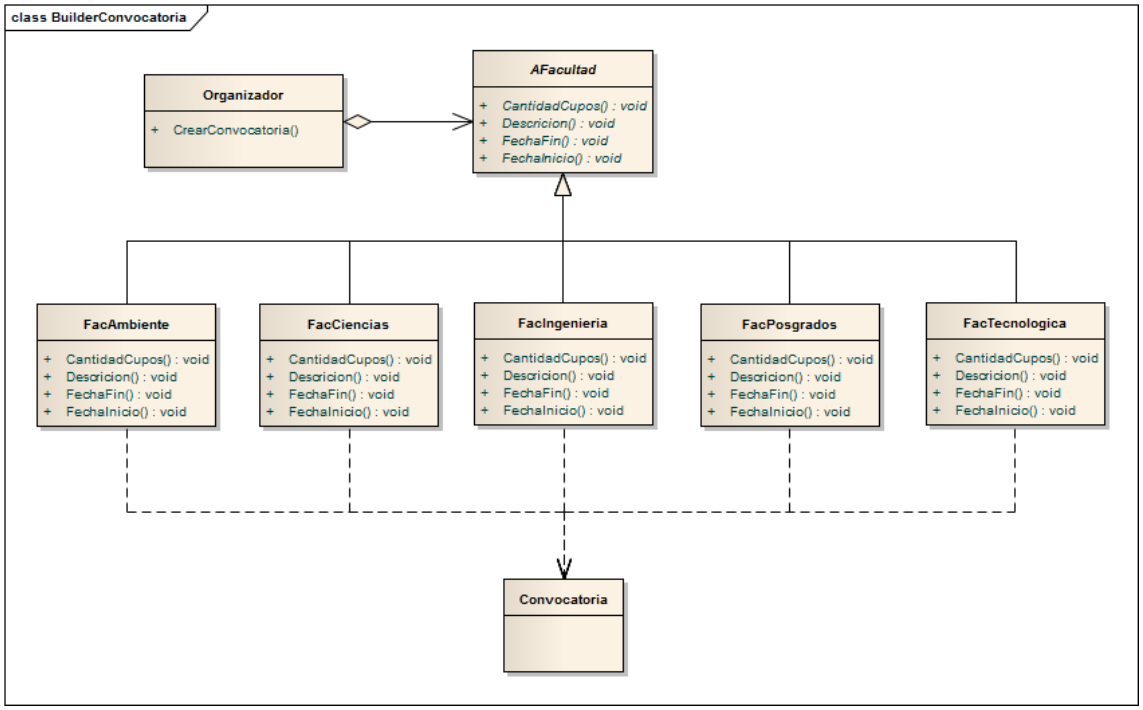



Figura 4.11: Diagrama de Clase para creación de de convocatorias por facultad

Como Patrón de diseño, el patrón builder (Constructor) es usado para permitir la creación de una variedad de objetos complejos desde un objeto fuente (Producto), el objeto fuente se compone de una variedad de partes que contribuyen individualmente a la creación de cada objeto complejo a través de un conjunto de llamadas a interfaces comunes de la clase Abstract Builder. En este caso la construccion se hace a travez de la creacion de convocatorias de cada facultad.

### Factory

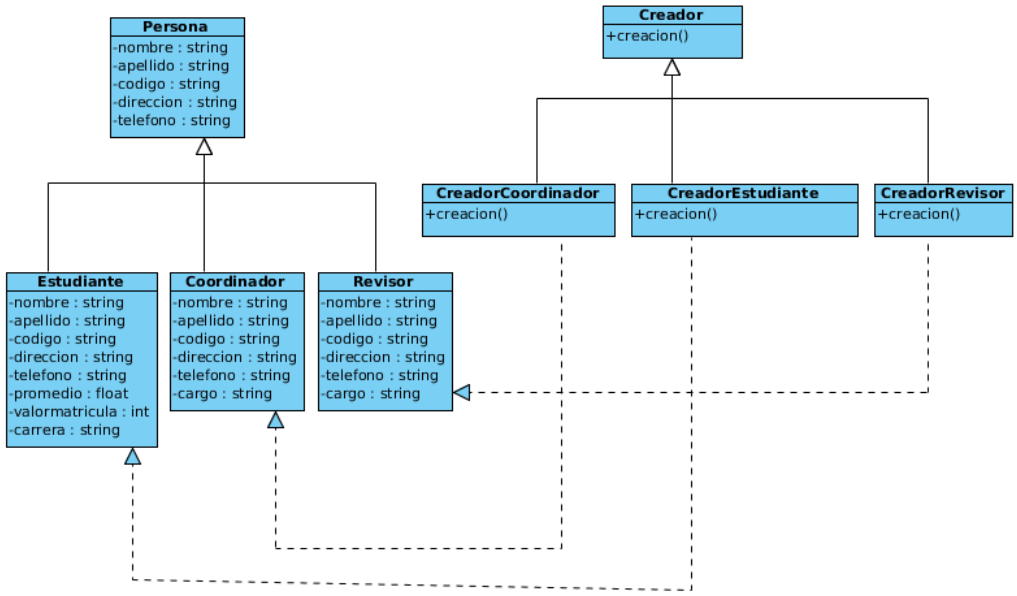



Figura 4.12: Diagrama de Clase para el manejo de la conexión a distintas bases de datos a través del patrón de diseño Facade

En diseño de software, el patrón de diseño Factory Method consiste en utilizar una clase constructora (al estilo del Abstract Factory) abstracta con unos cuantos métodos definidos y otro(s) abstracto(s): el dedicado a la construcción de objetos de un subtipo de un tipo determinado. Es una simplificación del Abstract Factory, en la que la clase abstracta tiene métodos concretos que usan algunos de los abstractos; según usemos una u otra hija de esta clase abstracta, tendremos uno u otro comportamiento. En esta caso implementamos el patron Factory Method para la creacion de diferentes personas que pueden ser estudiantes,coordinador y revisor ya que a pesar de ser hijos del mismo tipo las clases realizan distintas funciones en el software.

**Singleton**

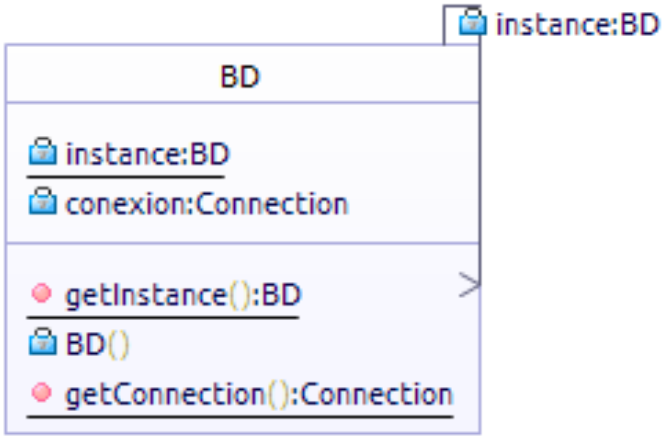



Figura 4.13: Diagrama de Clase para el manejo de las operaciones de escritura/lectura sobre base de datos a través del patrón de diseño Bridge

En ingeniería de software, singleton o instancia única es un patrón de diseño que permite restringir la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto. Su intención consiste en garantizar que una clase sólo tenga una instancia y proporcionar un punto de acceso global a ella.El patrón singleton se implementa creando en nuestra clase un método que crea una instancia del objeto sólo si todavía no existe alguna. Para asegurar que la clase no puede ser instanciada nuevamente se regula el alcance del constructor (con modificadores de acceso como protegido o privado). En este caso aplicamos este patron para controlar el acceso a la base de datos.

### 4.3.2 Comportamiento

#### Iterador

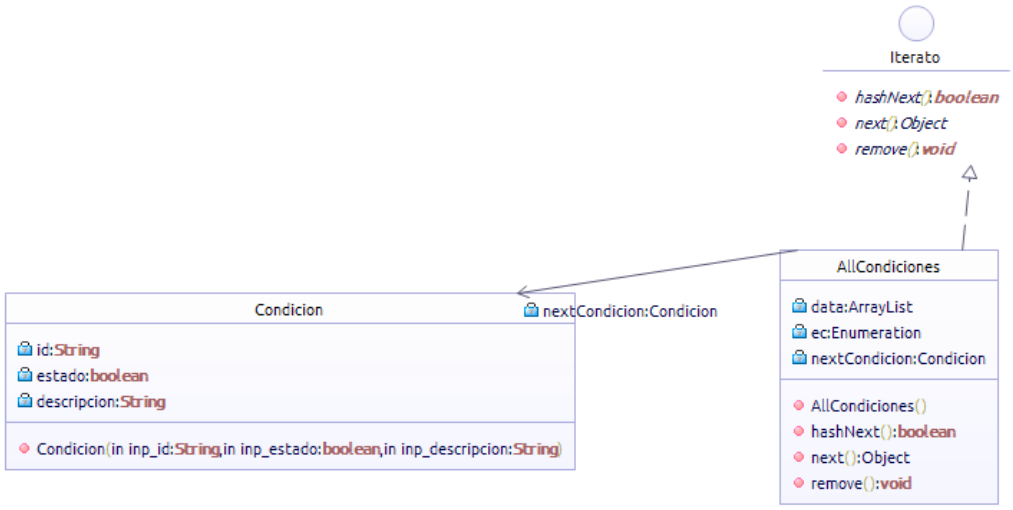



Figura 4.14: Diagrama de Clase para la ejecucion de acciones de CRUD mediante el patron de diseño Comando

Proporcionar una forma de acceder a los elementos de un objeto agregado de forma secuencial sin exponer sus detalles. En este caso me permite analizar todas las condiciones que hay para evaluar el puntaje de la solicitud al apoyo alimentario.

#### State

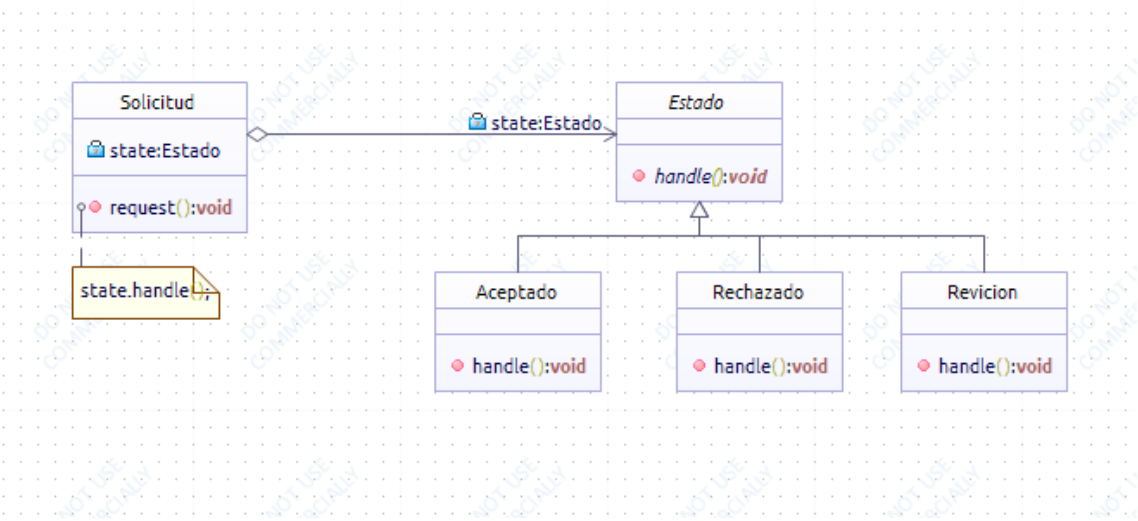



Figura 4.15: Diagrama de Clase para la relacion entre flujos financieros y sugerencias mediante el patron de diseño Mediator

El patrón de diseño State se utiliza cuando el comportamiento de un objeto cambia dependiendo del estado del mismo. Por ejemplo: una alarma puede tener diferentes estados, como desactivada, activada, en configuración. En este caso permite manejar el estado el cambio de una solicitud de aceptado, rechazado y revicion.

# 5. Estados

Un estado es una situación durante la vida de un objeto, de forma que cuando dicha situación se satisface se lleva a cabo alguna acción o se espera por un evento. El estado de un objeto se puede caracterizar por el valor de una o varias de las características de su clase, además, el estado de un objeto también se puede caracterizar por la existencia de un enlace con otro objeto. El diagrama de estados y transiciones incluye todos los mensajes que un objeto puede enviar o recibir. En un diagrama de estados, un escenario simboliza un camino dentro del diagrama. Dado que generalmente el espacio entre dos envíos de mensajes representa un estado, se pueden utilizar los diagramas de secuencia para buscar los diferentes estados de un objeto [**Pw5DE**].

Nuestro software posee elementos que tiene diferentes estados, para poder observar como estos estados cambian, y cuales son las condiciones de transición entre estos, de mostrará a continuación un diagrama de estados donde se resume todo esto.

## 5.1 Diagrama de Estados

Los diagramas de estado son un método conocido para explicar el comportamiento de un sistema. Que explican todos los estados posibles en los que puede ingresar un objeto particular y la manera en que modifica el estado del objeto, como resultado de los eventos que llegan a el. Permite identificar bajo qué pruebas se ejecuta cada uno de los procesos y en qué momento podrían tener una variación. El diagrama de estados permite visualizar de una forma ordenada la ejecución de cada uno de los procesos [**Pw5DE**].

La figura mostrada a continuación tiene el diagrama de estados del ciclo de vida que sufre la clase FlujoEconomico la cual puede encontrarse en tres diferentes estados, ACTIVO, EN ESPERA y FINALIZADO. El disparo a nivel general es la operación verificarTiempo la cual recibe como parámetro un tiempoActual, y es una referencia para poder realizar comparaciones y establecer así el efecto que se lleva acabo de forma determinada. Ahora bien, existen tres transiciones entre cada fase:

Desde el estado ACTIVO hacia el estado EN ESPERA la condición es que el tiempoActual sea diferente el tiempoDeEjecucion y que sea de naturaleza cíclica para poder pasar al efecto de suspender.

Desde el estado EN ESPERA hacia el estado ACTIVO la condición es que el tiempoActual sea igual al tiempoDeEjecucion y cuyo efecto es el de notificar.

Desde el estado ACTIVO a FINALIZADO la condición es que es que el tiempoActual sea mayor al tiempoDeEjecución y no sea ciclico para desembocar en el efecto de finalizar.

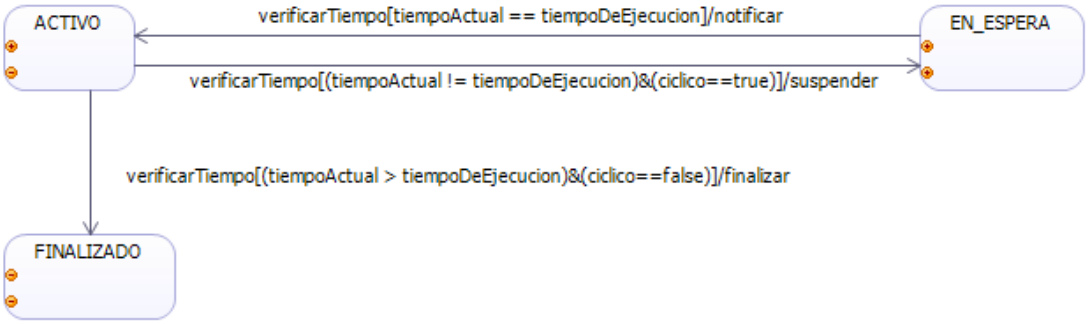



Figura 5.1: Diagrama de Estado

### Diagrama de clases

En el siguiente esquema se muestra la clase foco que es FlujoEconomico, ésta es la que cambia de estados según las condiciones dadas, para ello debe tener como atributos un ESTADO, que es el que va a variar, y los métodos necesarios para actuar a la hora de cambiar de estado.

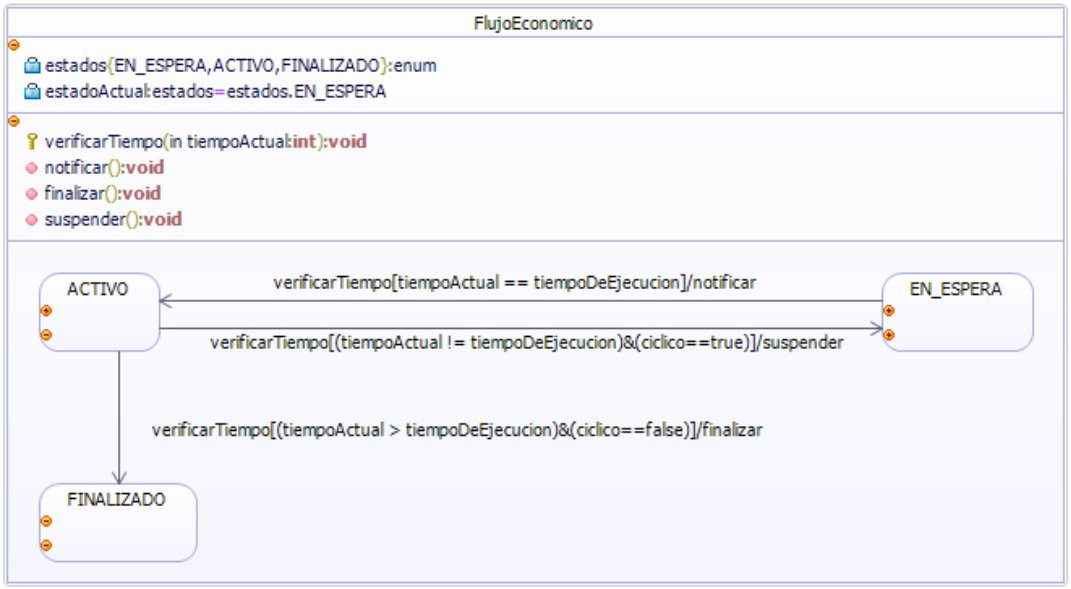



Figura 5.2: Diagrama de Clase FlujoEconomico

### Código

El siguiente es código de nuestra clase, éste es generado por el programa de software Coloso, para la generación de éste se involucra el panorama de Java y sirve como plantilla para generar la clase correspondiente.

```
1  import static com.componentes.diseno.Imc.marcosDeReferencia.computacion.Computacion.*;
2  import static com.componentes.diseno.Imc.marcosDeReferencia.conjuntos.Conjuntos.*;
3  import static com.componentes.diseno.Imc.marcosDeReferencia.constantes.Constantes.*;
4  import static com.componentes.diseno.Imc.marcosDeReferencia.estadistica.Estadistica.*;
5  import static com.componentes.diseno.Imc.marcosDeReferencia.matematica.Matematica.*;
6  import static com.componentes.diseno.Imc.marcosDeReferencia.probabilidad.Probabilidad.*;
7  public class FlujoEconomico{
8  private enum estados{EN_ESPERA,ACTIVO,FINALIZADO};
9  private estados estadoActual=estados.EN_ESPERA;
10 protected void verificarTiempo(int tiempoActual){}
11 public void notificar(){}
12 public void finalizar(){}
13 public void suspender(){}
14 }
```

Figura 5.3: código de Clase FlujoEconomico

## 5.2 Diagrama de flujo de trabajo

Los diagramas de flujo de trabajo describen el comportamiento del sistema cuando tiene una actividad lo suficientemente grande para requerir un detallado de ésta, de las partes y de cómo interactuan entre sí.

Para desarrollarlo se implementan diferentes elementos como:

- **Particiones**: Es una abstracción de las partes del sistema que interactua entre sí.
- **Actividades**: Son las tareas que le corresponde realizar a cada partición.
- **Transiciones**: Dan el orden en que las particiones deben desarrollar sus tareas siendo cada una dependiente de la anterior.
- **Estado final**: Es el estado al que debe llegar el flujo cuando se culmine el proceso y no hayan más transiciones.

Para el software de EconomApp podemos encontrar, entre sus diferentes actividades, dos actividades relevantes: Inserción de un flujo a la base de datos y Selección de sugerencias, estas tienen un grado de complejidad mayor al resto de actividades y por ello a continuación se describirán los flujos de trabajo para cada uno.

### Inserción de un flujo a la base de datos

La inserción de un nuevo flujo económico exige la abstracción de tres particiones, el usuario, la plataforma y la persistencia, los cuales se relacionan de forma que , cuando el usuario quiera añadir nuevos flujos económicos tenga la capacidad de visualizar aquellos que ya existen, y llegado el caso que encuentre uno similar, solicite a la plataforma, ya sea diferenciarlo del que se encontró debido a que es estrictamente necesario añadirlo o simplemente darse cuenta que no necesita añadir uno nuevo y cancelar la operación.

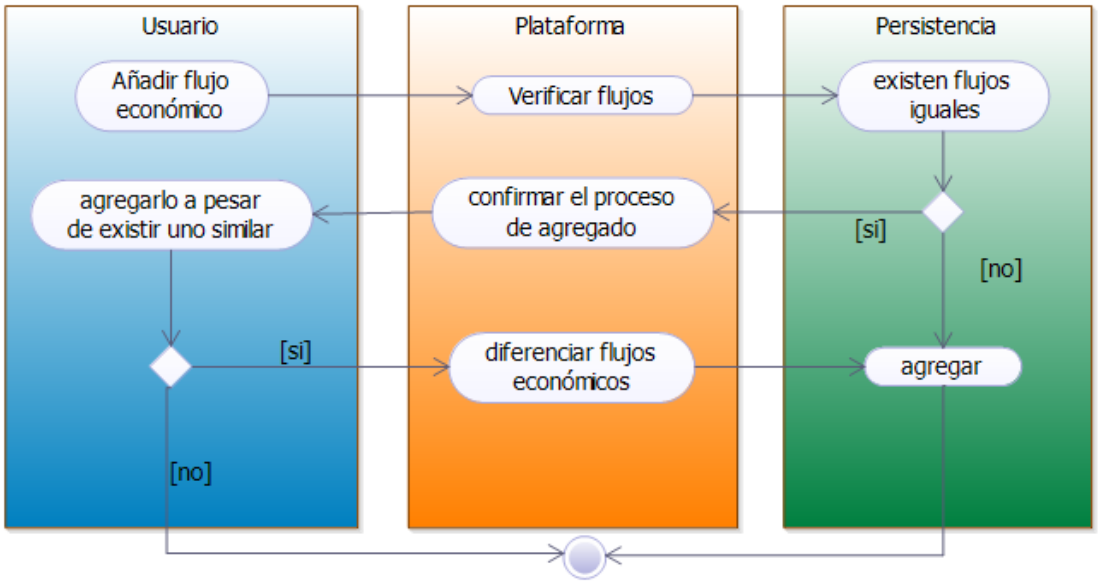



Figura 5.4: Diagrama de flujo de trabajo para agregar un flujo nuevo

### Selección de sugerencias

Las sugerencias se realizan automáticamente teniendo en cuenta un flujo económico particular, por lo que la plataforma debe solicitar a la persistencia los flujos existentes para seleccionar uno, evaluar sus características y poder hacer la creación. Ahora bien, estas sugerencias son mostradas al usuario para que éste tome la decisión de guardarla o simplemente ignorarla. Terminado el

proceso de selección, la plataforma, consciente de las sugerencias seleccionadas interactua con la persistencia para guardarlas y ella misma notifica al usuario finalmente.

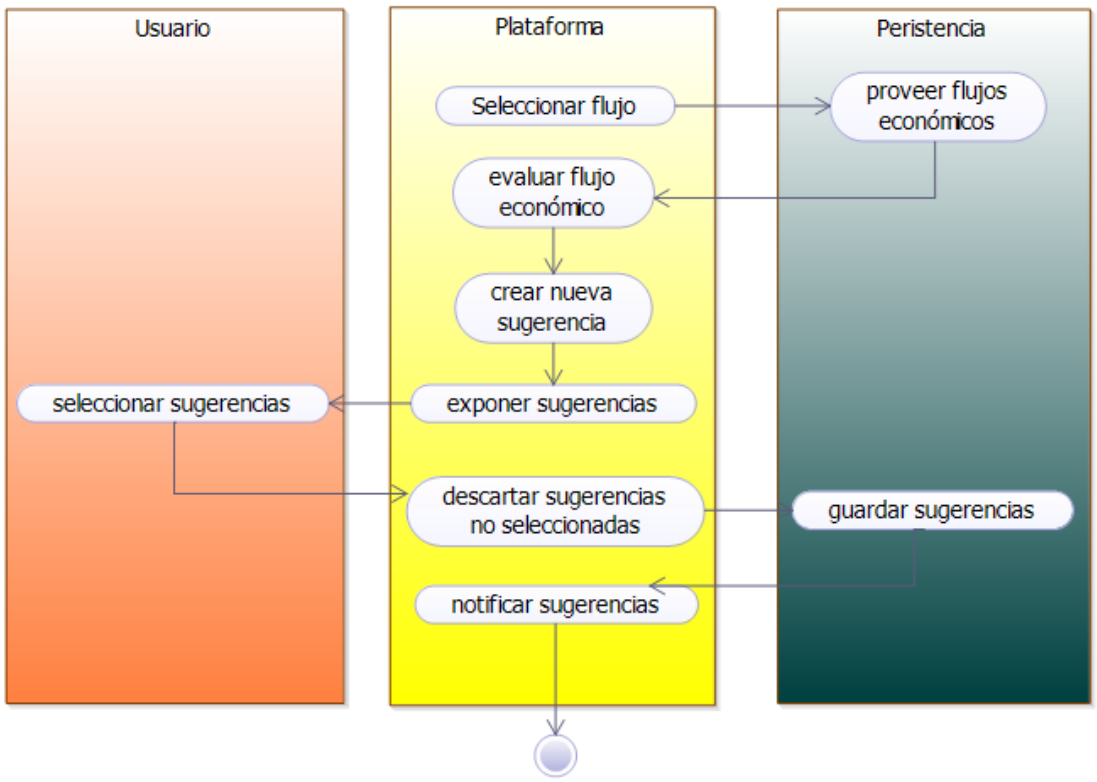



Figura 5.5: Diagrama de Flujo de trabajo para la selección de sugerencias

# III Reflexiones

# 6. Conclusiones

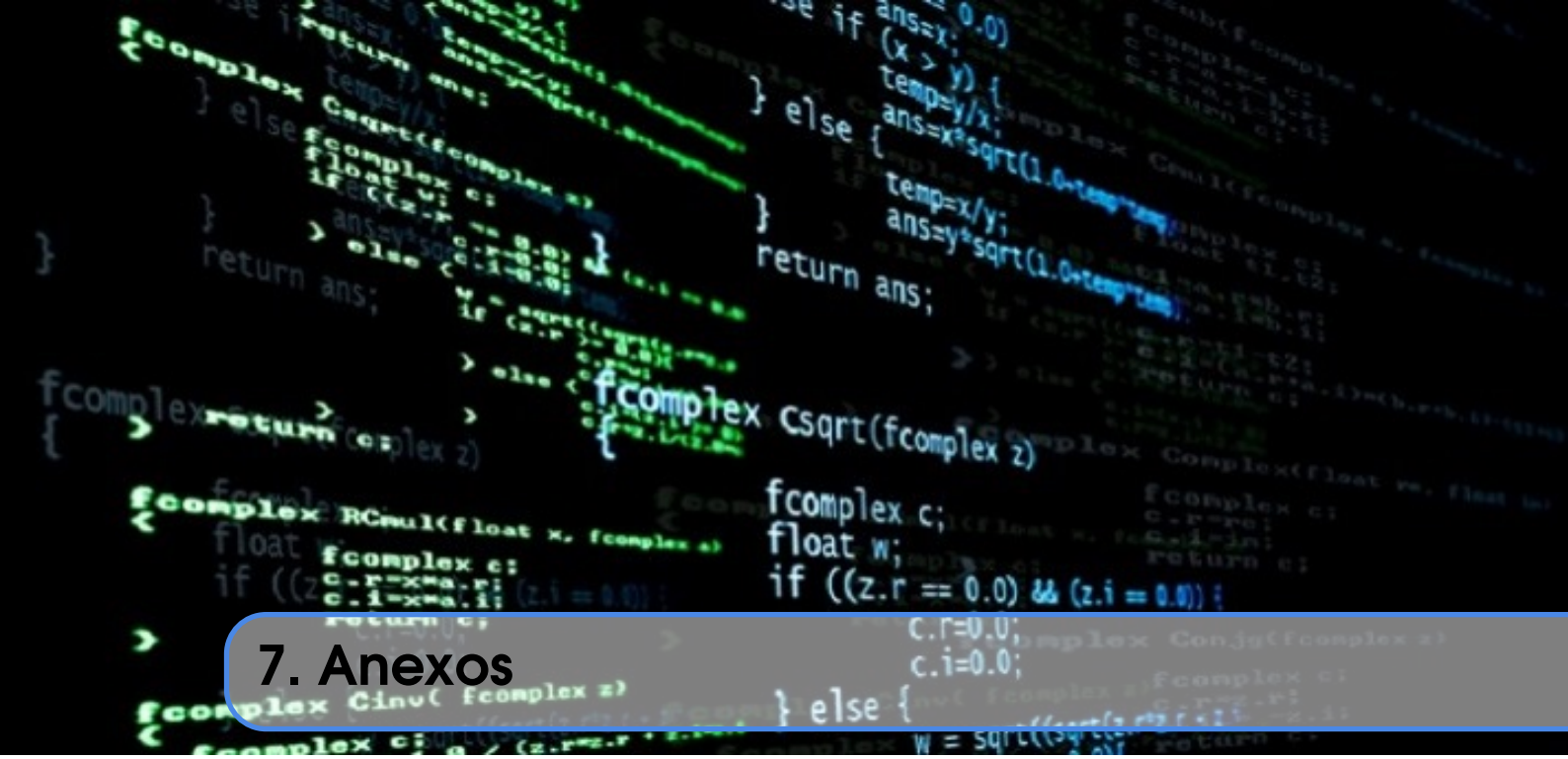
## 6.1 Conclusiones

Después de realizar el recorrido a lo largo del estudio de los procesos y metodologías de software, y además, mirar las estructuras que definían el diseño de software para luego implementarlo en el proyecto del curso, fue posible realizar las siguientes conclusiones:

- El desarrollo de software se basa esencialmente en la definición y posterior ejecución de tres grandes aspectos: la especificación, el desarrollo y el mantenimiento. Estos grandes pilares representan la base a la hora de emprender cualquier proyecto de software y se encuentran en cualquiera de los posibles caminos que especifique una metodología, ya sea de forma explícita o implícitamente.
- A lo largo de los años se han venido construyendo propuestas sobre las cuales trabajar para poder fijar el rumbo de un proyecto, todas siempre sustentadas en la experiencia y que tienen en cuenta situaciones específicas. Para el proyecto particular de EcnonomApp, el hecho de haber escogido la metodología RUP fue precisamente debido a que sus protocolos se ajustaban a las necesidades de la construcción de la aplicación, es decir, la capacidad de no tener estados rígidos y dependientes sino una serie de procesos adaptables, prioridades balanceadas, adquisición de valor de forma iterativa, entre otras, se ajustaban a las necesidades del equipo.
- Una vez establecida la forma en la que se realizan las cosas llega la hora de trabajarlas. Es allí donde la definición de requerimientos adquiere una importancia natural, porque fija el punto de partida, señala lo que el sistema final debe realizar, establece una característica de rendimiento con la cual evaluar el resultado, y desde la cual establecer las correcciones necesarias llegado el caso que no se logre lo esperado.
- El lenguaje UML proporciona una serie de herramientas estandarizadas que se ofrecen con el fin de realizar una explicación asertiva del sistema al cual hace referencia, por lo que brinda la capacidad de que cualquier persona, incluso que no se halla encontrado en la construcción de un proyecto, lo entienda e interprete.
- El objetivo del lenguaje UML es la respuesta al ¿cómo? del sistema, una vez expuesto el ¿qué? en el establecimiento de los requerimientos y los casos de uso. Para empezar a describir

el ¿cómo? se empieza desde la parte de la identificación de objetos y su comunicación, esto no es de extrañar dado que UMl nace en el marco del paradigma de la programación orientada a objetos.

- Después de las posterior generalización de los objetos en clases, surgen conceptos. Dentro de la especificación de los pilares de la orientación a objetos se habla de que los programas deben ser abiertos para la extensión y cerrados para la modificación, es allí donde nacen dos conceptos completamente nuevos, y son las cajas blancas y las cajas negras, donde las primeras son abstracciones encaminadas a un acoplamiento máximo y una seguridad mínima, y las segundas ofrecen seguridad máxima y acoplamiento mínimo.
- En el diseño de los programas se mide el nivel de confianza de una estructura pensada en función del número de interfaces y clases abstractas que contenga, por lo que si se visualizan exceso de clases concretas, es necesario preocuparse por la forma como se modeló el problema.
- Muchas veces un elemento en el contexto del diseño de sotware sufre cambios en su estado, los cuales afectan de forma directa el comportamiento de éste y su uso en determinadas situaciones. Es por ello que se utiliza el diagrama de máquina de estados, para visualizar la interacción entre diferentes estados a partir de disparos y transiciones.
- En un sistema es importante poder visualizar el flujo de algún elemento y la señalización del paso de una actividad a otra, es decir los procesos en alto nivel, es por ello que se utiliza el diagrama de actividades, y es allí donde recae su importancia.
- Llegando al final del diseño uno de los temas que va aumentando su nivel de complejidad es el de la agrupación y la organización de modulos, empezando por el concepto de sistemas dónde se analiza una característica principal y un grado alto de afinidad modular para segmentar y clasificar las clases y guardarlas en paquetes. Después, pasando por el concepto de componentes en los cuales se busca cual es el trabajo de determinados sistemas y se procede a segmentarlos en unidades aún mas diferenciables, y que además tengan capacidades de ser coherentes en sí mismos, es decir, no depender de ninguna otra instancia para poder saber cual es su función. Finalmente, se llega al concepto de nodo, en el cual, desde un punto de vista más alejado del software y mas general, se corresponde a poner determinados sistemas en deterinados nodos dependiendo de cual es el recurso que éstos necesitan.
- Todos estos aspectos se utilizan para poder hacer frente de una forma especializada a la dificil tarea de la construcción de soluciones, pero gracias a las buenas prácticas de diseño, se simplifica de forma que sea un aspecto manejable y posible, aterrizando todos los experimentos reales en una cuestión práctica, concreta y tangible.



# 7. Anexos

## 7.1 Código de los patrones

### 7.1.1 Patron Estado

```
1  public class Solicitud{
2    private Estado state;
3
4
5    public void request(){
6    state.handle();
7
8  }
9  }
10
11
12
13 public class Aceptado extends Estado{
14 public void handle(){}
15 }
16
17 public class Rechazado extends Estado{
18 public void handle(){}
19 }
20
21 public class Revicion extends Estado{
22 public void handle(){}
23 }
```

codigo/Patrones/Estado.ts

### 7.1.2 Patron Builder

```
1  public abstract class AFacultad {
2
3  public AFacultad m_AFacultad;
4
5  public AFacultad(){
6
7  }
8
9  public void finalize() throws Throwable {
10
11 }
12
13 public abstract void CantidadCupos();
14
15 public abstract void Descripcion();
16
17 public abstract void FechaFin();
18
19 public abstract void FechaInicio();
20
21 }
22
23
24 public class Convocatoria {
25
26 public Convocatoria(){
27
28 }
29
30 public void finalize() throws Throwable {
31
32 }
33
34 }
35
36
37 public class FacAmbiente extends AFacultad {
38
39 public FacAmbiente(){
40
41 }
42
43 public void finalize() throws Throwable {
44 super.finalize();
45 }
46
47 public void CantidadCupos(){
48
49 }
50
51 public void Descripcion(){
52
53 }
54
55 public void FechaFin(){
56
57 }
58
59 public void FechaInicio(){
```

```
60
61 }
62
63 }
64
65
66
67 public class FacCiencias extends AFacultad {
68
69 public FacCiencias(){
70
71 }
72
73 public void finalize() throws Throwable {
74 super.finalize();
75 }
76
77 public void CantidadCupos(){
78
79 }
80
81 public void Descripcion(){
82
83 }
84
85 public void FechaFin(){
86
87 }
88
89 public void FechaInicio(){
90
91 }
92
93 }
94
95
96 public class FacIngenieria extends AFacultad {
97
98 public FacIngenieria(){
99
100 }
101
102 public void finalize() throws Throwable {
103 super.finalize();
104 }
105
106 public void CantidadCupos(){
107
108 }
109
110 public void Descripcion(){
111
112 }
113
114 public void FechaFin(){
115
116 }
117
118 public void FechaInicio(){
119
120 }
```

```
121
122 }
123
124
125 public class FacPosgrados extends AFacultad {
126
127 public FacPosgrados(){
128
129 }
130
131 public void finalize() throws Throwable {
132 super.finalize();
133 }
134
135 public void CantidadCupos(){
136
137 }
138
139 public void Descripcion(){
140
141 }
142
143 public void FechaFin(){
144
145 }
146
147 public void FechaInicio(){
148
149 }
150
151 }
152
153 public class FacTecnologica extends AFacultad {
154
155 public FacTecnologica(){
156
157 }
158
159 public void finalize() throws Throwable {
160 super.finalize();
161 }
162
163 public void CantidadCupos(){
164
165 }
166
167 public void Descripcion(){
168
169 }
170
171 public void FechaFin(){
172
173 }
174
175 public void FechaInicio(){
176
177 }
178
179 }
180
181
```

```
182 public class Organizador {
183
184 public AFacultad m_AFacultad;
185
186 public Organizador(){
187
188 }
189
190 public void finalize() throws Throwable {
191
192 }
193
194 }
```

codigo/Patrones/Builder.ts

### 7.1.3 Patron Singleton

```
1 package ;
2 public class BD{
3 private static BD instance;
4 private Connection conexion;
5 public static BD getInstance(){}
6 private  BD(){}
7 public static Connection getConnection(){}
8 }
```

codigo/Patrones/Singleton.ts

### 7.1.4 Patron Iterador

```
1 package ;
2 public class AllCondiciones implements Iterato{
3 private ArrayList data;
4 private Enumeration ec;
5 private Condicion nextCondicion;
6 public  AllCondiciones(){}
7 public boolean hashNext(){}
8 public Object next(){}
9 public void remove(){}
10 }
11
12
13 package ;
14 public class Condicion{
15 private String id;
16 private boolean estado;
17 private String descripcion;
18 public  Condicion(String inp_id,boolean inp_estado,String inp_descripcion)
      {}
19 }
20
21
22 package ;
23 public interface Iterato{
24 /**
25
26 */
27 boolean hashNext();
```

```
28 /**
29
30 */
31 Object next();
32 /**
33
34 */
35 void remove();
36 }
```

codigo/Patrones/Iterator.ts

# Bibliografía


[1] *Diagramas de secuencia UML: Referencia*. 2015. URL: https://msdn.microsoft.com/es-co/library/dd409377.aspx (visitado 20-05-2017) (véase página 21).

[2] IBM. *Casos de Uso*. 2013. URL: https://www.ibm.com/support/knowledgecenter/es/SSWSR9{_}11.0.0/com.ibm.pim.dev.doc/pim{_}tsk{_}arc{_}definingusecases.html (visitado 20-05-2017) (véase página 17).

[3] *UML: LOS DIAGRAMAS DE COMUNICACIÓN*. 2009. URL: http://diagramas-comunicacion-uml.blogspot.com.co/2009/02/los-diagramas-de-comunicacion.html (visitado 20-05-2017) (véase página 25).