

# Proyecto Final

Estefanía García González, Sebastián Mora Sabogal

28 de mayo de 2019



# Índice general

<b>I</b>	<b>PROYECTO</b>	<b>7</b>
<b>1.</b>	<b>Caso de Estudio</b>	<b>9</b>
1.1.	Introducción . . . . .	9
1.2.	Objetivo General . . . . .	9
1.3.	Objetivos Específicos . . . . .	9
1.4.	Descripción del problema . . . . .	10
1.5.	Alcance . . . . .	10
<b>2.</b>	<b>Metodología</b>	<b>11</b>
2.1.	Introducción . . . . .	11
2.2.	Proceso de Software . . . . .	11
2.2.1.	Metodología de implementación . . . . .	12
2.3.	Open Source . . . . .	13
<b>II</b>	<b>DISEÑO</b>	<b>15</b>
<b>3.</b>	<b>Requerimientos</b>	<b>17</b>
3.1.	Introducción . . . . .	17
3.2.	Requerimientos del Cliente . . . . .	17
3.3.	Casos de uso . . . . .	18
3.4.	Diagramas de secuencia . . . . .	20
3.5.	Diagramas de comunicación . . . . .	23
<b>4.</b>	<b>Interacción</b>	<b>33</b>
4.1.	Introducción . . . . .	33
<b>5.</b>	<b>Clases</b>	<b>35</b>
5.1.	Introducción . . . . .	35
5.2.	Teoría . . . . .	35

<b>6. Patrones</b>	<b>37</b>
6.1. Introducción . . . . .	37
6.2. Patrón Composite . . . . .	38
6.3. Patrón agrupador . . . . .	38
6.4. Patrón Fábrica Abstracta . . . . .	39
6.5. Patrón Estrategia . . . . .	40
<b>7. Estados</b>	<b>41</b>
7.1. Introducción . . . . .	41
<b>8. Componentes</b>	<b>43</b>
8.1. Introducción . . . . .	43
<b>9. Nodos</b>	<b>45</b>
9.1. Introducción . . . . .	45
<b>10. Actividades</b>	<b>47</b>
10.1. Introducción . . . . .	47
<b>III REFLEXIONES</b>	<b>49</b>
<b>11. Conclusiones</b>	<b>51</b>
11.1. Introducción . . . . .	51
<b>A. Apéndice capítulo 6: Patrones</b>	<b>53</b>
A.1. Patrón composite . . . . .	53
A.1.1. Clase Tarea . . . . .	53
A.1.2. Clase Subtarea . . . . .	53
A.1.3. Clase Tarea trabajo . . . . .	54
A.1.4. Clase Tarea consulta . . . . .	54
A.1.5. Clase Tarea lectura . . . . .	54
A.2. Patrón agrupador . . . . .	54
A.2.1. Clase horario . . . . .	54
A.2.2. Clase franja . . . . .	55
A.3. Patrón Estrategia . . . . .	56
A.3.1. Clase EstrategiaMod . . . . .	56
A.3.2. Clase EstrategiaConcretaA . . . . .	56
A.3.3. Clase EstrategiaConcretaB . . . . .	56
A.3.4. Clase EstrategiaConcretaC . . . . .	56
A.3.5. Clase Tarea . . . . .	56

A.3.6. Clase Subtarea . . . . .	56
A.3.7. Clase Tarea trabajo . . . . .	57
A.3.8. Clase Tarea consulta . . . . .	57
A.3.9. Clase Tarea lectura . . . . .	57
A.4. Patrón Fábrica Abstracta . . . . .	57
A.4.1. Clase Tarea . . . . .	57
A.4.2. Clase Subtarea . . . . .	58
A.4.3. Clase Tarea trabajo . . . . .	58
A.4.4. Clase Tarea consulta . . . . .	58
A.4.5. Clase Tarea lectura . . . . .	58
A.4.6. Clase FabricaAbstracta . . . . .	59
A.4.7. Clase FabricaTTrabajo . . . . .	59
A.4.8. Clase FabricaTLectura . . . . .	59
A.4.9. Clase FabricaTConsulta . . . . .	59
A.4.10. Clase Cliente . . . . .	59



# Índice de figuras

2.1. Cronograma. Diagrama de Gantt . . . . .	12
2.2. Scrum . . . . .	13
3.1. Primer diagrama de caso de uso . . . . .	19
3.2. Segundo diagrama de caso de uso . . . . .	19
3.3. Tercero diagrama de caso de uso . . . . .	20
3.4. Cuarto diagrama de caso de uso . . . . .	20
3.5. Diagrama de secuencia caso de uso 01. . . . .	21
3.6. Diagrama de secuencia caso de uso 02. . . . .	21
3.7. Diagrama de secuencia caso de uso 03. . . . .	21
3.8. Diagrama de secuencia caso de uso 05. . . . .	21
3.9. Diagrama de secuencia caso de uso 14. . . . .	22
3.10. Diagrama de secuencia caso de uso 15. . . . .	22
3.11. Diagrama de secuencia caso de uso 16. . . . .	22
3.12. Diagrama de secuencia caso de uso 19. . . . .	22
3.13. Diagrama de comunicación caso de uso 01. . . . .	23
3.14. Diagrama de comunicación caso de uso 02. . . . .	23
3.15. Diagrama de comunicación caso de uso 03. . . . .	23
3.16. Diagrama de comunicación caso de uso 05. . . . .	24
3.17. Diagrama de comunicación caso de uso 14. . . . .	24
3.18. Diagrama de comunicación caso de uso 15. . . . .	24
3.19. Diagrama de comunicación caso de uso 16. . . . .	24
3.20. Diagrama de comunicación caso de uso 19. . . . .	24
5.1. Relaciones UML. Tomada de internet . . . . .	36
6.1. Patrón Componente . . . . .	38
6.2. Patrón Agrupador . . . . .	39
6.3. Patrón Fábrica Abstracta . . . . .	40
6.4. Patrón Estrategia . . . . .	40





Parte I

**PROYECTO**



# Capítulo 1

## Caso de Estudio

### 1.1. Introducción

Desde que el ser humano cuenta con raciocinio , ha buscado organizarse, desarrollar metodologías y nuevas tecnologías que faciliten su diario vivir. Ha habido un recorrido histórico en el cual las necesidades humanas de optimización de tiempo y recursos han ido en aumento, así mismo las soluciones a éstas. En los últimos años se ha podido apreciar una constante migración al uso de tecnologías de la información que permiten realizar a cabo tareas en todos los ámbitos de forma óptima. Uno de los actores que más se han visto inmersos en la revolución digital son los estudiantes, pero en su contexto universitario, hace falta desarrollar estrategias que le permitan mejorar la gestión de tiempo de sus actividades académicas; por lo cual se buscará una solución tecnológica que se adapte a las necesidades de los universitarios.

### 1.2. Objetivo General

Desarrollar un software que gestione actividades y tiempos de las asignaciones académicas a estudiantes universitarios, utilizando los modelos y metodologías de ingeniería de software para mejorar la productividad del universitario.

### 1.3. Objetivos Específicos

1. Analizar el problema teniendo en cuenta la observación de las necesidades del estudiante, para así enfocarse en estos elementos primordiales a la hora de desarrollar el software.

2. Presentar una solución a nivel de software a partir del previo análisis del problema para finalmente implementarlo.

#### 1.4. Descripción del problema

La vida universitaria y académica suele ser difícil de manejar debido a la cantidad de trabajos que se deben entregar diariamente, a la prioridad que cada una es para el usuario y a la gestión de tiempo para poder realizarlos. Tareas, trabajos, talleres y grandes proyectos son algunas de las actividades que un estudiante realiza durante su semestre; además de que cada uno tiene complejidad y tiempo de realización diferentes estimados por el estudiante. Una solución factible es la utilización de un software gestor de tareas orientado a la organización y optimización de actividades académicas.

#### 1.5. Alcance

Este software tendrá la capacidad de gestionar los horarios de los estudiantes, añadir recordatorios de trabajos próximos a presentar y ofrecer el servicio de organizar en horarios la realización de las tareas pendientes. Esto se llevará a cabo de acuerdo a la complejidad de la actividad a realizar, en la cual se tomará en cuenta el nivel de dificultad, si se puede desarrollar en diferentes etapas y la fecha de entrega.

El estudiante estará en la capacidad de añadir actividades, determinar la complejidad de éstas y asignarles un horario de realización que puede ser repartido en varios bloques cuando la tarea requiere de mucho tiempo. Adicionalmente, las actividades podrán personalizarse añadiéndoles objetivos a cumplir o subactividades.

## Capítulo 2

# Metodología

### 2.1. Introducción

La metodología del proceso de software que se debe seguir, es fundamental pues define las acciones generales que se deben llevar, a modo de conseguir un desarrollo del proyecto optimo, pasando por cada una de las fases del proceso elegido.

### 2.2. Proceso de Software

Parte importante de un proyecto de software es definir el, o los ciclos de vida que se manejarán dentro del proyecto, ya que estos determinarán estrategias para planificar, desarrollar y mantener el software. Por esta razón, se definirá el modelo de procesos a utilizar, tomando en cuenta los siguientes criterios:

- Es necesaria una metodología que sea pertinente para un proyecto de software pequeño con pocos desarrolladores.
- Se considera importante la verificación en cada fase del ciclo de vida, ya que permite sentar buenas bases dentro del proyecto y reducir el riesgo.
- Además de la verificación, es necesaria una retroalimentación constante, ya que es posible ver con mayor claridad las falencias y carencias del proyecto.

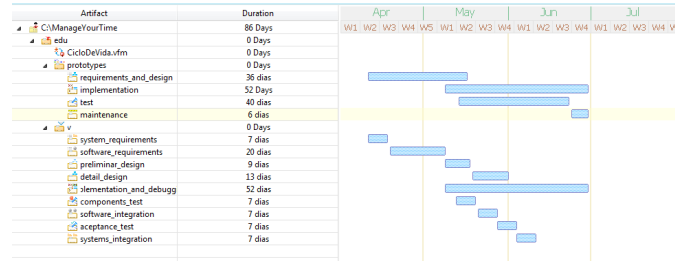


Figura 2.1: Cronograma. Diagrama de Gantt

- Como último criterio fundamental, se contempla la necesidad de desarrollar algunas partes de software de forma rápida, ya que esto facilitaría la retroalimentación del sistema.

Para cumplir con las pautas anteriormente mencionadas, los ciclos de vida que se elegirán son prototipo y V. Cada uno de estos modelos obedece solo a algunas de las especificaciones, pero juntos se complementan de la siguiente manera:

- El modelo V es perfecto para equipos de trabajo pequeños, ya que es sencillo, de fácil aprendizaje, robusto e incluye pruebas en cada fase, lo que facilita el trabajo cuando hay pocas personas.
- Gracias a los dos ciclos de vida, es posible hacer una verificación y retroalimentación de forma efectiva, ya que con el modelo en V se hacen pruebas en cada fase y con el prototipo es posible obtener resultados a corto plazo que se pueden ir revisando y evaluando.
- El modelo de prototipo brinda la posibilidad de construir partes del proyecto de forma prematura, por lo que es posible realizar pruebas y verificar qué cosas es necesario cambiar o añadir.

### 2.2.1. Metodología de implementación

Los criterios que se establecieron al momento de justificar la elección los procesos de software prototipo y V, cuentan con la misma validez para determinar la metodología de implementación, debido a que para esta etapa también es necesario tener un plan de acción que beneficie la gestión de tiempos del proyecto, complemente los procesos de software, cumpliendo un proceder de forma organizada. Por esta razón se utilizará Scrum como metodología para implementar.

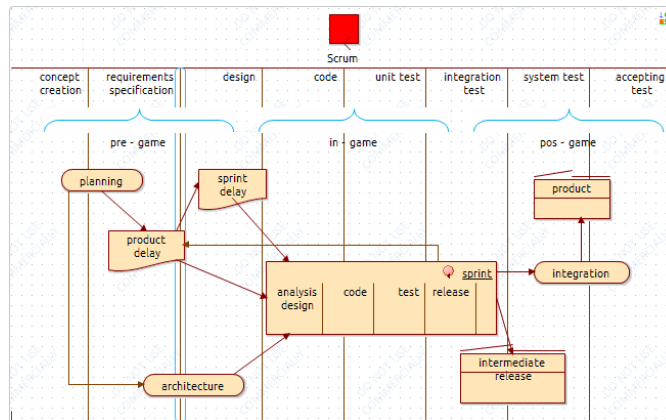


Figura 2.2: Scrum

## 2.3. Open Source

Desde que las personas empezaron a desarrollar software, han empezado a indagar en diferentes formas de realizar las cosas, a fin de obtener la solución computacional que solucione su necesidad. Con el tiempo estos pensamientos han devenido en ideologías que orientan la variedad de metodologías disponibles para desarrollar software.

El pensamiento o filosofía que entra en cuestión, es la del software libre, donde uno de sus principios, consiste en la reutilización del conocimiento, en este caso, el código. Es aquí donde entra el Open Source, que se relaciona con el código abierto, y con su revisión por parte de una comunidad de desarrolladores externos. Siguiendo el principio de filosofía libre, se pretende utilizar el concepto O.S con la intención de obtener una ayuda en momentos donde la implementación se torne complicada, llegandose a extrapolar a diversos casos en los que se necesite la apreciación del problema que se está trabajando por parte de un externo el cual ya lo haya desarrollado.





Parte II

**DISEÑO**



## Capítulo 3

# Requerimientos

### 3.1. Introducción

Para cualquier proyecto de software, es un punto fundamental conocer cuál es la necesidad y el problema que el cliente desea resolver. Para tener una visión holística del problema, se hace necesario definir los requerimientos que satisfagan al cliente y resuelvan el problema.

### 3.2. Requerimientos del Cliente

Se entiende como lo que el cliente espera encontrar cuando interactúe con la aplicación. Bajo la anterior premisa, se definieron los siguientes requerimientos:

1. Añadir una tarea.
2. Añadir subtareas para una tarea.
3. Añadir un horario universitario.
4. Añadir un horario de descanso (dormir).
5. Añadir un horario de transporte.
6. Añadir una tarea a una materia.
7. Mostrar todas las tareas pendientes.
8. Mostrar las tareas pendientes por materia.
9. Mostrar las tareas pendientes por tipo.

10. Mostrar las tareas pendientes para una fecha.
11. Mostrar las tareas pendientes por dificultad.
12. Mostrar el horario general del usuario.
13. Mostrar los horarios asignados para las tareas pendientes.
14. Modificar horario.
15. Modificar tarea.
16. Sugerir horarios para realizar tareas.
17. Sugerir cuánto tiempo podría tomar una tarea.
18. Sugerir tiempos de pausas activas durante la realización de una tarea.
19. Alertar de la próxima entrega de una tarea.
20. Advertir si se debe sacrificar algún espacio de descanso.

### 3.3. Casos de uso

Los casos de uso describen la interacción del usuario con las diversas funcionalidades planteadas, permitiendo obtener una forma de comunicar los requerimientos de tal forma que sea entendida tanto por usuario como por desarrolladores. Como se podrá observar a continuación, serán cuatro diagramas de caso de uso los que se presentan, donde el motivo por el cual los casos de uso comparten diagrama, es porque se considera que existe cierta relación entre ellos.

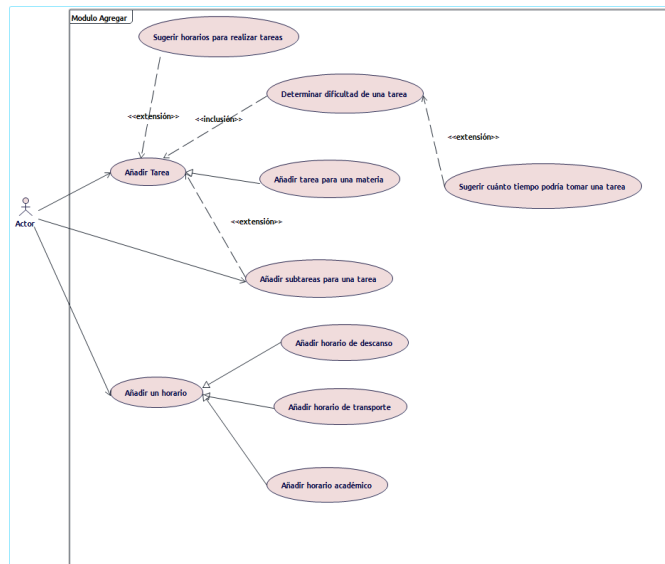


Figura 3.1: Primer diagrama de caso de uso

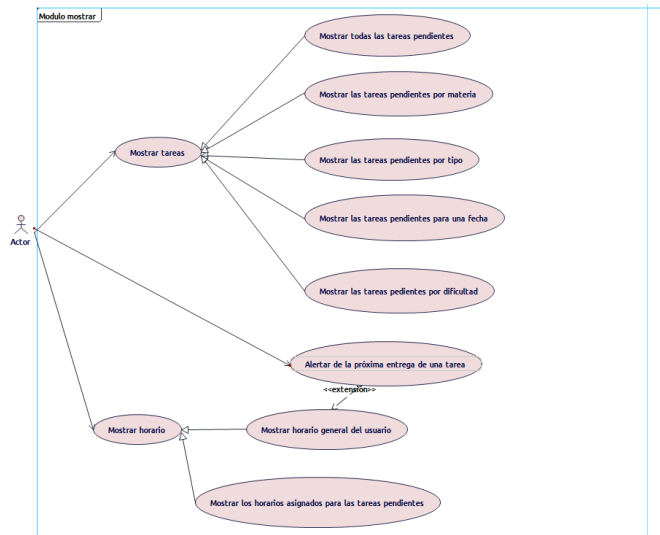


Figura 3.2: Segundo diagrama de caso de uso

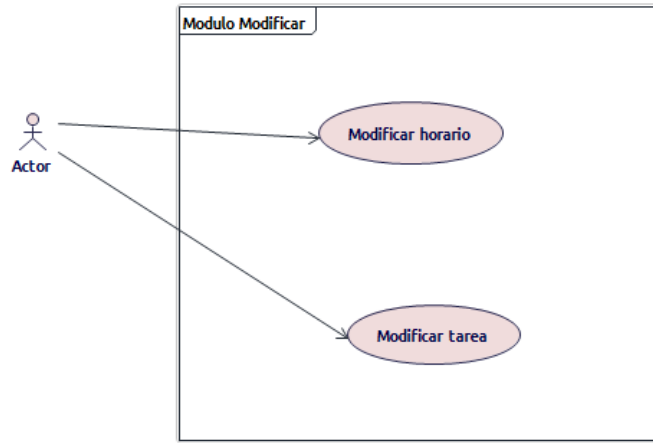


Figura 3.3: Tercero diagrama de caso de uso

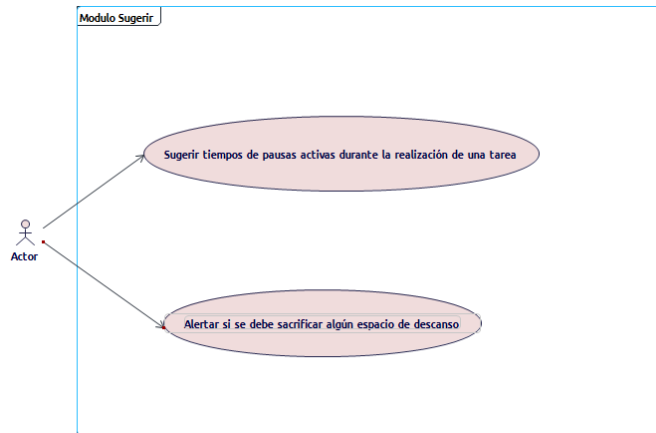


Figura 3.4: Cuarto diagrama de caso de uso

Los diagramas a continuación representan gran importancia complementando la definición de los requerimientos.

### 3.4. Diagramas de secuencia

Los diagramas de secuencia permiten observar la realización del caso de uso, responden el como se va a hacer el requerimiento. Los siguientes son los diagramas de secuencia de 4 casos de uso que se consideran de mayor importancia.

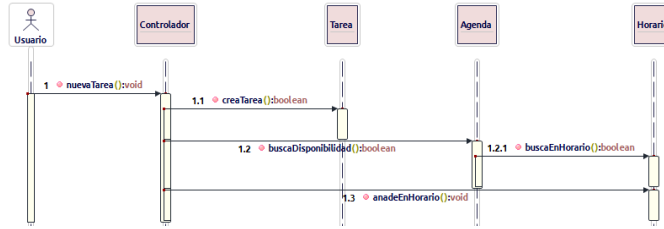


Figura 3.5: Diagrama de secuencia caso de uso 01.

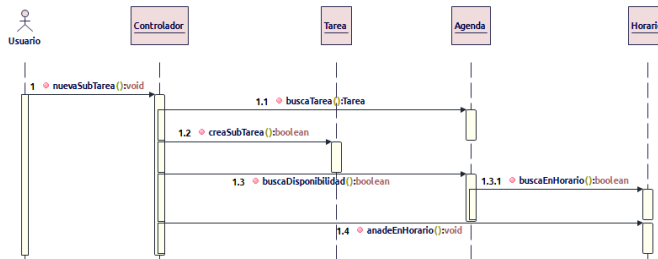


Figura 3.6: Diagrama de secuencia caso de uso 02.

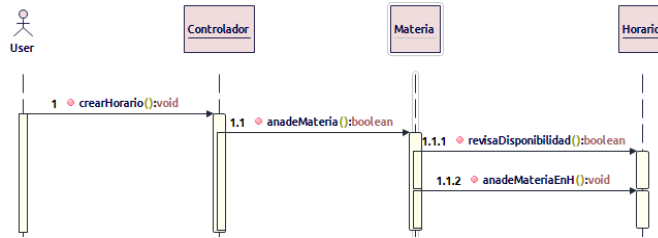


Figura 3.7: Diagrama de secuencia caso de uso 03.

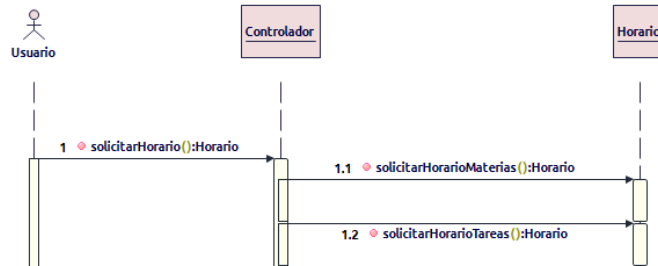


Figura 3.8: Diagrama de secuencia caso de uso 05.

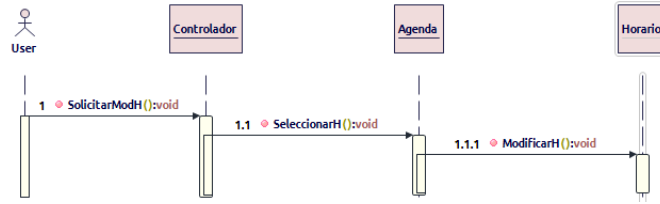


Figura 3.9: Diagrama de secuencia caso de uso 14.

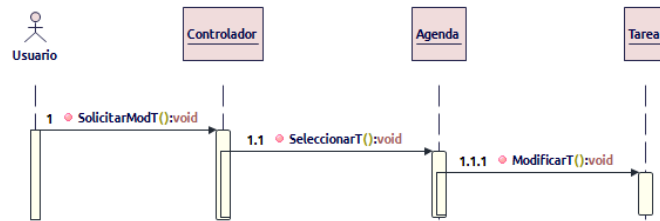


Figura 3.10: Diagrama de secuencia caso de uso 15.

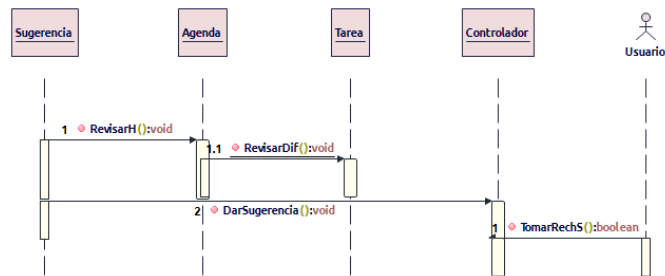


Figura 3.11: Diagrama de secuencia caso de uso 16.

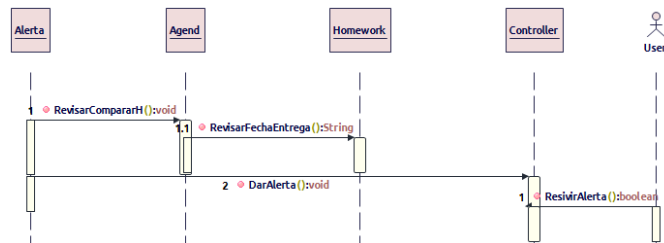


Figura 3.12: Diagrama de secuencia caso de uso 19.



### 3.5. Diagramas de comunicación

Igualmente relacionados con los diagramas anteriores, principalmente con el diagrama de secuencia. Su función como su nombre lo indica, consiste en detallar en como se comunican los objetos que solucionan el requerimiento.

Se presentan los diagramas correspondientes a los ya expuestos diagramas de secuencia:

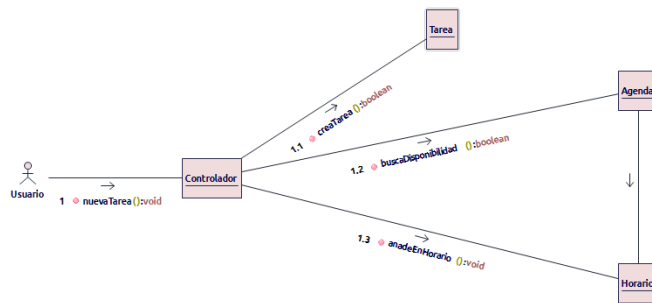


Figura 3.13: Diagrama de comunicación caso de uso 01.

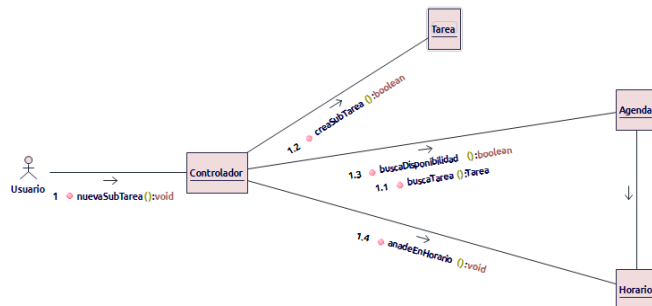


Figura 3.14: Diagrama de comunicación caso de uso 02.

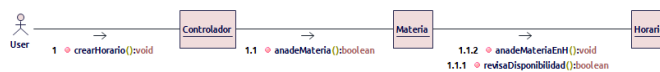


Figura 3.15: Diagrama de comunicación caso de uso 03.

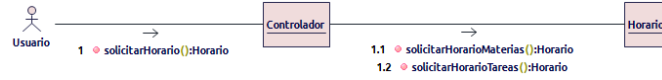


Figura 3.16: Diagrama de comunicación caso de uso 05.

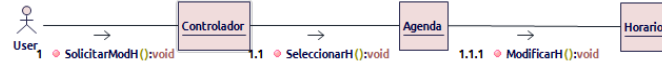


Figura 3.17: Diagrama de comunicación caso de uso 14.



Figura 3.18: Diagrama de comunicación caso de uso 15.

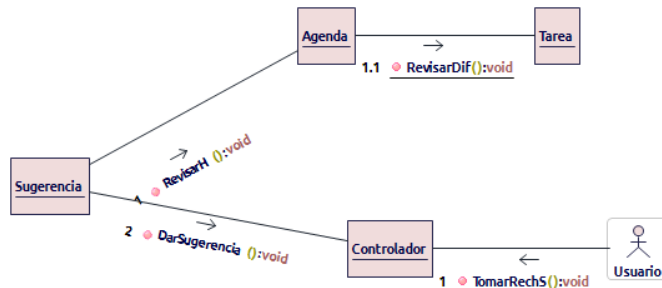


Figura 3.19: Diagrama de comunicación caso de uso 16.

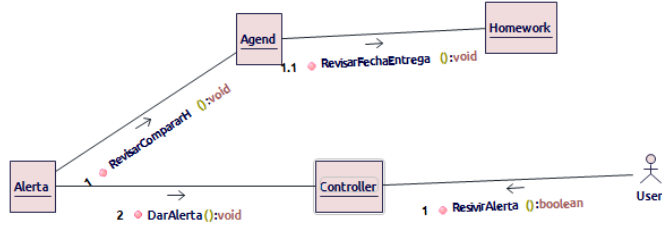


Figura 3.20: Diagrama de comunicación caso de uso 19.

Las siguientes tablas, son la especificación de los requerimientos que se considera que tienen un nivel de importancia alta.

RF-01	Añadir una tarea	
Descripción	El usuario añade una tarea pendiente por desarrollar.	
Precondición	El usuario debe tener un horario	
Secuencia	Paso	Acción
	1	El usuario selecciona la opción de crear tarea.
	2	El usuario proporciona la información requerida (nombre de la tarea, tipo, materia a la que pertenece)
	3	El usuario verifica la información registrada.
	4	El usuario hace selecciona el botón aceptar.
Postcondición	El sistema muestra la tarea recién asignada con sus especificaciones y su recomendación de tiempo de realización y de horario	
Excepciones	Paso	Acción
	4	Se añade una tarea que requiere urgencia (Imprevisto). El usuario elige que horario sacrificará para realizar la tarea.
	4	Se añade una tarea que es imposible de realizar debido al tiempo u horario. Es necesario modificar los tiempos u horarios en los que se realizará la tarea o elegir si sacrificar una frnaja de horario.
Rendimiento	Paso	Cota de tiempo
	1	1 segundo
	2	40 segundos
	3	5 segundos
	4	1 segundo
Importancia	Muy importante	
Urgencia	urgente	

RF-02	Añadir subtareas para una tarea.	
Descripción	Se añade una subtask a una tarea.	
Precondición	Debe existir alguna tarea pendiente.	
Secuencia	Paso	Acción
	1	El usuario selecciona la tarea a la que desea añadirle una subtask.
	2	Seleccionar la opción de añadir subtask.
	3	Se añade la subtask como una tarea (RF-01)
	4	El usuario verifica la información.
	5	El pulsa la opción de aceptar.
Postcondición	El sistema añadirá la subtask a la tarea, mostrará sus especificaciones y recomendación de tiempo de realización y de horario	
Excepciones	Paso	Acción
	1	No existe una tarea para añadirle una subtask.
	3	La subtask es de carácter urgente.
	4	Se añade una subtask y esta hace que la tarea sea imposible de terminar debido al tiempo u horario.
Rendimiento	Paso	Cota de tiempo
	1	5 segundos
	2	1 segundo
	3	40 segundos
	4	5 segundos
	5	1 segundo
Importancia	Importante	
Urgencia	No urgente	
Comentarios	No.	Descripción
	1	Añadir una subtask es lo mismo que añadir una tarea, la deferencia es que está anidada dentro de una tarea general.

RF-03	Añadir un horario universitario	
Descripción	Se crea un horario con materias de la universidad.	
Precondición	Ser un usuario registrado.	
Secuencia	Paso	Acción
	1	Seleccionar la opción de crear horario.
	2	Escribir el nombre de cada materia y su respectiva hora de inicio y fin y los días en que se repite.
	3	El usuario añade la materia y repite el proceso cuantas veces sea necesario.
	4	Pulsar en el botón de aceptar.
Postcondición	El sistema guardará el horario asignado para el usuario.	
Excepciones	Paso	Acción
	3	El horario de la universidad llena totalmente los espacios disponibles.
	3	No hay espacios disponibles para añadir más materias al horario.
Rendimiento	Paso	Cota de tiempo
	1	1 segundo
	2	40 segundos
	3	2 minutos
	4	1 segundo
Importancia	Muy importante	
Urgencia	Urgente	

RF-04	Mostrar todas las tareas pendientes.	
Descripción	Se muestra la lista de tareas pendientes.	
Precondición	Debe existir al menos una tarea pendiente.	
Secuencia	Paso	Acción
	1	Seleccionar la opción de ver las tareas pendientes.
Postcondición	El sistema mostrará todas las tareas pendientes	
Excepciones	Paso	Acción
	1	No hay tareas pendientes para mostrar.
Rendimiento	Paso	Cota de tiempo
	1	1 segundo
Importancia	Vital	
Urgencia	Urgente?	

RF-05	Mostrar el horario general del usuario.	
Descripción	Se muestra el horario completo del estudiante.	
Precondición	El usuario debe haber creado un horario antes.	
Secuencia	Paso	Acción
	1	Seleccionar la opción de mostrar el horario.
Postcondición	El sistema mostrará el horario con las materias, los descansos, los horario de transporte y las tareas pendientes	
Excepciones	Paso	Acción
	1	No hay un horario para presentar.
Rendimiento	Paso	Cota de tiempo
	1	1 segundo
Importancia	Importante	
Urgencia	Puede esperar	

RF-14	Modificar horarios	
Descripción	Se selecciona y modifica una franja del horario.	
Precondición	El horario que lo que se desea modificar debe estar asignado.	
Secuencia	Paso	Acción
	1	Se selecciona la opción de modificar horario.
	2	Se selecciona el horario de la tarea que se desea cambiar.
	3	Se selecciona la nueva franja de horario en la que se acomodara la tarea.
	4	El cambio de horario se ha realizado.
Postcondición	El horario es modificado y el sistema puede ofrecer sugerencia de tiempo de realización, o incluso si se debe sacrificar algún espacio de descanso.	
Excepciones	Paso	Acción
	1	No hay ningún horario para seleccionar, en este caso el caso de uso acaba.
	3	No existe ninguna franja disponible para cambiar, en este caso el caso de uso acaba.
Rendimiento	Paso	Cota de tiempo
	1	1 segundo
	2	1 segundo
	3	10 segundos
	4	1 segundo
Importancia	Importante	
Urgencia	Hay presión	

RF-15	Modificar Tareas.	
Descripción	Se permite modificar los diferentes campos de una tarea.	
Precondición	Debe existir alguna tarea para modificar.	
Secuencia	Paso	Acción
	1	Se solicita modificar una tarea.
	2	El usuario selecciona la tarea que desea modificar.
	3	El usuario selecciona el campo de la tarea que desea modificar.
	4	Se modifica el campo de la tarea.
	5	Se permite elegir realizar otro cambio o terminar.
Postcondición	La tarea tiene un campo modificado, según el tipo podría haber una sugerencia, como en el caso de dificultad, o cambio de horario para realizarse.	
Excepciones	Paso	Acción
	3	Si la tarea solo tiene los campos minimos se puede agregar el cambio, así el caso de uso continua.
Rendimiento	Paso	Cota de tiempo
	1	1 segundo
	2	1 segundo
	3	1 segundo
	4	5 segundos
	5	1 segundo
Importancia	vital	
Urgencia	Hay presión	

RF-16	Sugerir horarios para realizar tareas.	
Descripción	Según los horarios disponibles, al momentos de adicionar una tarea se hace una sugerencia de horario para realizarla.	
Precondición	Se debe haber agregado una tarea, y deben haber horarios disponibles para hacer la recomendación.	
Secuencia	Paso	Acción
	1	Al momento de agregar una tarea, se revisan horarios disponibles.
	2	Se revisan las variables de la tarea (dificultad, fecha de entrega).
	3	Se hace la recomendación.
Postcondición	La recomendación se dará al usuario dandole la posibilidad de tomarla o dejarla.	
Excepciones	Paso	Acción
	1	Puede pasar que no haya horarios disponibles para hacer la recomendación, así el caso de uso termina.
	2	Si no hay variables de tarea definidos, se pasa al siguiente paso.
Rendimiento	Paso	Cota de tiempo
	1	5 segundos
	2	5 segundos
	2	1 segundo
Importancia	Normal	
Urgencia	Puede esperar	
Comentarios	No.	Descripción
	1	El caso de uso esta bastante ligado a otros casos de uso, como puede apreciarse en el diagrama, sin embargo su relevancia no es la misma como la de los casos a los que apoya.



RF-19	Alertar de la próxima entrega de una tarea.	
Descripción	Se pretende avisar con tiempo prudencial que el ciclo de una tarea esta por finalizar, lo cual significa que debe ser terminada para ser entregada.	
Precondición Secuencia	La existencia de la tarea.	
	Paso	Acción
	1	Se revisa el tiempo para que la tarea deba ser terminada comparandolo con el prudencial.
	2	Se realiza el aviso.
Postcondición	El usuario será avisado sobre la proximidad de su tarea.	
Excepciones	Paso	Acción
	1	Si la tarea no cuenta con tiempo de realización se considera indefinida, así el caso de uso termina.
Rendimiento	Paso	Cota de tiempo
	1	1 segundo
	2	1 segundo
Importancia	Importante	
Urgencia	Puede esperar	



## Capítulo 4

# Interacción

### 4.1. Introducción

contenido...



# Capítulo 5

## Clases

### 5.1. Introducción

Los diagramas de clase son parte importante del diseño de un software, ya que estos permiten generar diseños que plasmen la solución a un problema, la cual será entendible para todos aquellos conozan del lenguaje unificado de modelado (UML).

Además es necesario utilizar diagramas de clase para representar gráficamente y de forma estática la estructura general del sistema, mostrando sus clases e interacciones.

### 5.2. Teoría

Los diagramas de clase sirven para visualizar las relaciones entre las clases que involucran el sistema, entre estas se encuentran:

- Dependencia.
- Asociación.
- Agregación.
- Composición.
- Generalización.
- Realización.

Estas relaciones pueden subdividirse en dos grandes grupos: Las relaciones cliente/proveedor, en las cuales entran las dependencias y asociaciones

(asociación, agregación y composición), las cuales generan un alto acoplamiento en el software, pero también lo hacen seguro. Por otro lado están las relaciones de generalización en las cuales están la especialización e implementación. Estas poseen un bajo acoplamiento, pero no poseen la seguridad de las de cliente/proveedor. El ideal es crear un diagrama de clases en el que haya un equilibrio entre el acoplamiento y la seguridad.

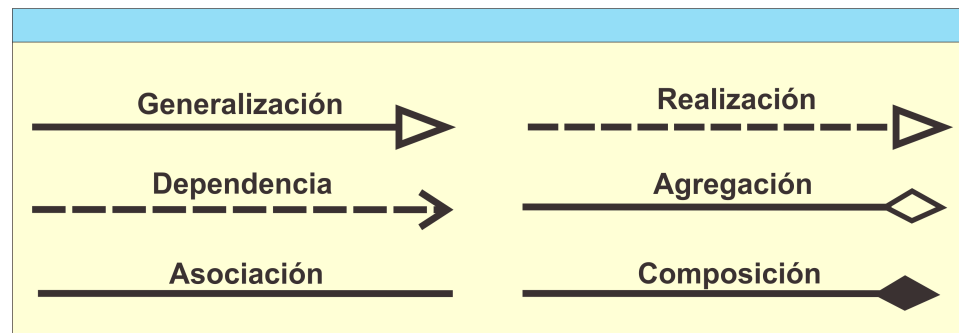


Figura 5.1: Relaciones UML. Tomada de internet

Por otro lado, las clases se representan por medio de un rectángulo que se divide en tres:

- Nombre de la clase: Cómo se denomina la clase. Es un sustantivo.
- Atributos de la clase: Pueden ser de diferentes tipos (booleano, numérico, cadenas y T.D.A), tienen modificadores de visibilidad (public, private, protected, de paquete), nombre y propiedades (final, const) si es necesario.
- Métodos de la clase: Son las operaciones que realiza la clase, deben denominarse con verbos, poseen modificadores de visibilidad y pueden retornar diferentes tipos de datos (void, numérico, cadenas y T.D.A) Adicionalmente, los métodos pueden recibir argumentos, que se representan en las operaciones en forma de parámetros los cuales son variables que poseen nombre y tipo.

## Capítulo 6

# Patrones

### 6.1. Introducción

Aunque históricamente el software pueda considerarse como primitivo si se compara con otras disciplinas que llevan siglos de ventaja en cuanto a su desarrollo, actualmente no es así. La anterior reflexión se deduce debido a que el software ha evolucionado lo suficiente para convertirse en una herramienta que va mas allá de solo programar, también se encarga de diseñar y modelar. En este caso, el diseño de software puede generar soluciones generalizadas a problemas reincidentes. Así es como llegamos a los patrones de diseño, los cuales para cualquier proyecto, independientemente de su complejidad y tamaño, seguramente se verán involucrados.

A partir de lo anterior, es claro que los patrones de diseño aparecerán en la propuesta de estructura del proyecto, más conocida como diagrama de clases, con la intención de obtener un diagrama razonable desde un punto de vista de principios de diseño así como los principios del paradigma orientado a objetos.

Se debe mencionar también que los patrones propuestos en este documento, no necesariamente todos están incluidos en el Gof (the Gang of Four), pues existen otras soluciones que aunque no tan conocidas, pueden tener un impacto positivo debido en el desarrollo del software.

## 6.2. Patrón Composite

El patrón de diseño Composite (Componente), nos permite construir estructuras complejas partiendo de otras estructuras mucho mas simples, lo cual permite crear estructuras compuestas conformadas por otras mas pequeñas. Este patrón resulta útil para la creación de subtareas dentro de una tarea mas general, ya que se genera una estructura en forma de árbol gracias a la recursividad con la que funciona el patrón. Otra ventaja de su utilización, en este desarrollo específicamente, es que se puede representar la jerarquía de tarea-subtarea, además de añadir dinamismo a la tarea, ya que ésta puede tener subtareas de diferentes tipos. Además es posible tratar la subtarea como tarea.

En conclusión, el patrón componente posibilita la solución del problema de las subtareas, ya que permite jerarquizar, añadir dinamismo a la tarea por medio de subtareas y construir la tarea general por medio de subtareas, por esta razón será utilizado dentro de este software.

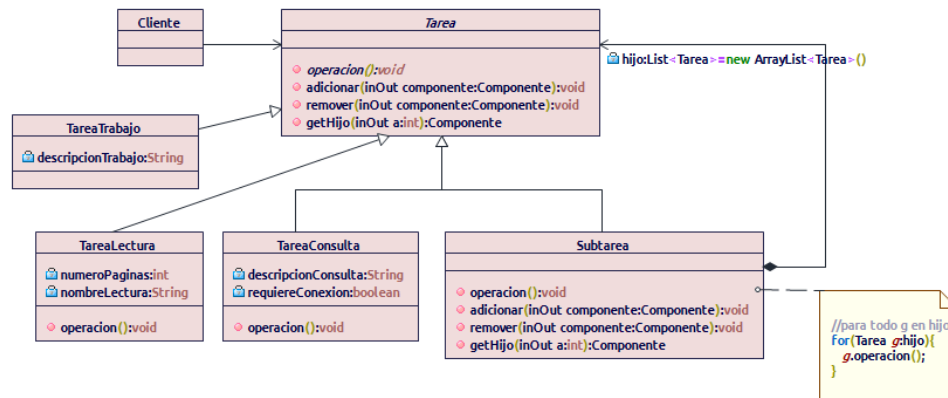


Figura 6.1: Patrón Componente

## 6.3. Patrón agrupador

Un horario puede mostrarse como la constitución de diversas franjas en un orden lógico; estas deben agruparse para que haya orden y se puedan manejar conjuntamente. Por esta razón el patrón agrupación será de utilidad, ya que permite generar una estructura en la cual los módulos, que en este caso son las franjas, puedan ser agrupados para invocarse de forma colectiva como el horario, de esta forma se centraliza el control de la estructura en una sola clase.



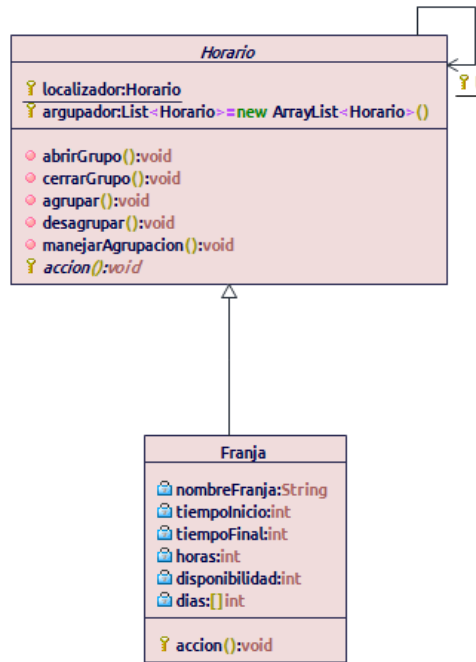


Figura 6.2: Patrón Agrupador

## 6.4. Patrón Fábrica Abstracta

La creación de objetos es una situación que aparece en prácticamente cualquier proyecto, por lo que se debe buscar una estructura que permita llevar esta creación de la mejor manera, incluso en mayor medida cuando el objeto a crear pertenece a una familia de otros objetos que también pueden ser creados, y por supuesto en un caso más general en caso de que hayan varias de estas familias de objetos.

La fábrica abstracta lo que hace es esto, proveer una interfaz para la creación de estas familias de objetos relacionados, sin especificar sus clases concretas.

Para el caso del proyecto, la fábrica abstracta es útil pues existen dos familias de objetos a crear, una de tareas referente a lo académico y otra la de tareas que no tienen que ver con la universidad.

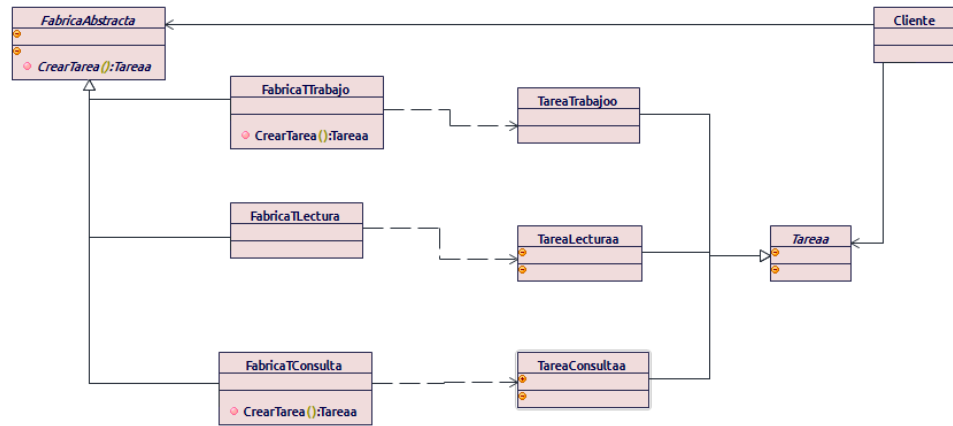


Figura 6.3: Patrón Fábrica Abstracta

## 6.5. Patrón Estrategia

Dada la situación de que un programa pueda ofrecer un servicio, el cual se pueda realizar de varias maneras hace alusión a este patrón. La intención de este es poder seleccionar la alternativa más adecuada para el cliente, durante tiempo de ejecución.

En la mención a la fábrica abstracta se mencionaron las familias de objetos de tarea y categoria, las cuales en algún momento necesitarán de la posibilidad de una modificación de alguno de sus objetos, operación que no solo involucra a este último en cuestión, sino que abarca principalmente el modulo externo de manejo de base de datos. Dependiendo de cada objeto, la misma operación debe hacerse de manera distinta, haciendo así que entre el patrón estrategia.

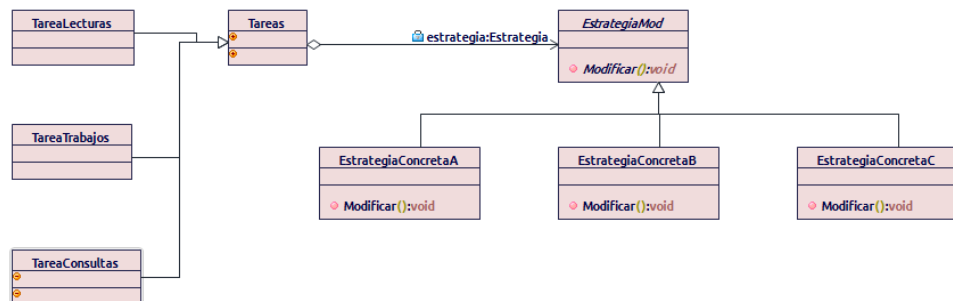


Figura 6.4: Patrón Estrategia

## Capítulo 7

# Estados

### 7.1. Introducción

A medida que se realiza una solución a un problema, los objetos deberán interactuar para llegar a esta solución, y en medio de esas interacciones, habrán acciones que incentiven el cambio de estado de uno o varios de esos objetos

## 7.2. Marco Teórico

## 7.3. Diagramas de estado

Abarcando el problema que se esta resolviendo, se determinó que el objeto al cual se le deben aclarar sus estados, es el de tarea, pues hace parte fundamental en la solución del problema, así que el entendimiento de sus estados a lo largo de la ejecución del software que pretende implementarla será crucial.

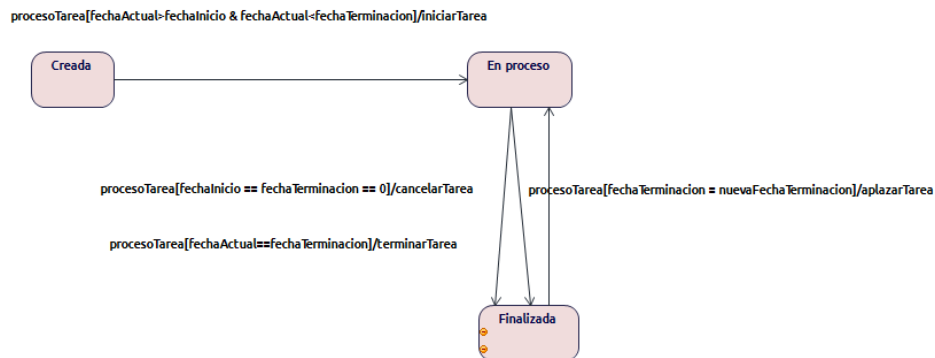


Figura 7.1: Diagrama de estados para el objeto Tarea.

Cabe mencionar, de manera obligatoria, que este diagrama tendrá su correspondiente planeación estructural, que en otras palabras vendría a ser su espacio en el diagrama de clases de toda la solución. Para esto, existe un patrón de comportamiento, que precisamente tiene como objetivo cumplir el principio de diseño abierto/cerrado en el contexto de estados de un objeto.

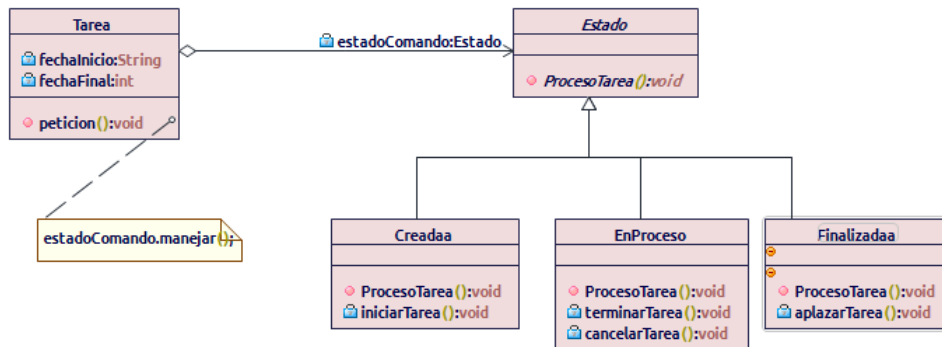


Figura 7.2: Patrón state para solucionar los estados del objeto Tarea.

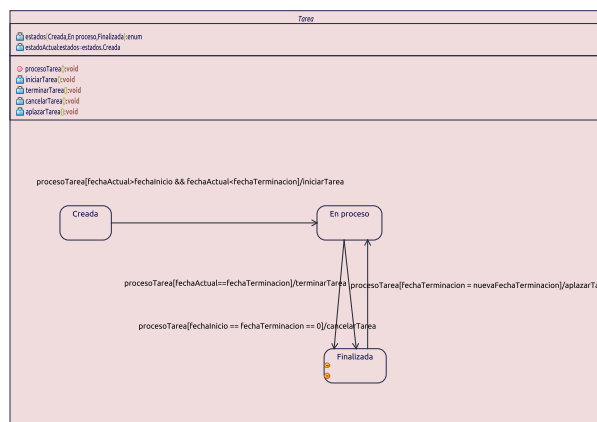


Figura 7.3: Detalle de la clase Tarea bajo el contexto del patrón Estado.



## Capítulo 8

# Componentes

### 8.1. Introducción

contenido...





## Capítulo 9

# Nodos

### 9.1. Introducción

contenido...



## Capítulo 10

# Actividades

### 10.1. Introducción

contenido...



Parte III

**REFLEXIONES**



## Capítulo 11

# Conclusiones

### 11.1. Introducción

contenido...





## Apéndice A

# Apéndice capítulo 6: Patrones

### A.1. Patrón composite

#### A.1.1. Clase Tarea

```
1 public abstract class Tarea{
2     public abstract void operacion();
3     public void adicionar(Componente componente){}
4     public void remover(Componente componente){}
5     public Componente getHijo(int a){
6         return null;
7     }
8 }
```

#### A.1.2. Clase Subtarea

```
9 import java.util.List;
10 import java.util.ArrayList;
11 public class Subtarea extends Tarea{
12     private List<Tarea> hijo=new ArrayList<Tarea>();
13     public void operacion(){
14         //para todo g en hijo
15         for(Tarea g:hijo){
16             g.operacion();
17         }
18     }
19     public void adicionar(Componente componente){
20         hijo.add(componente);
21     }
22 }
```

```
21 }
22 public void remover(Componente componente){
23     hijo.remove(componente);
24 }
25 public Componente getHijo(int a){
26     return hijo.get(a);
27 }
28 }
```

### A.1.3. Clase Tarea trabajo

```
29 public class TareaTrabajo extends Tarea{
30     private String descripcionTrabajo;
31 }
```

### A.1.4. Clase Tarea consulta

```
32 public class TareaConsulta extends Tarea{
33     private String descripcionConsulta;
34     private boolean requiereConexion;
35     public void operacion(){}
36 }
```

### A.1.5. Clase Tarea lectura

```
37 public class TareaLectura extends Tarea{
38     private int numeroPaginas;
39     private String nombreLectura;
40     public void operacion(){null}
41 }
```

## A.2. Patrón agrupador

### A.2.1. Clase horario

```
42 import java.util.List;
43 import java.util.ArrayList;
44 public abstract class Horario{
45     protected static Horario localizador;
46     protected List<Horario> argupador=new ArrayList<Horario>
47         >();
48     public static void main(String[] args){
49         Horario A = new Franja();
50         A.abrirGrupo();
51         A.agrupar();
52         A.manejarAgrupacion();
53     }
```

```

52         Horario B = new AgrupadorB();
53         B.agrupar();
54         B.manejarAgrupacion();
55     }
56     public void abrirGrupo() {
57         if(localizador==null){
58             localizador=this;
59             argupador=new ArrayList<Horario>();
60         }
61     }
62     public void cerrarGrupo() {
63         localizador = null;
64     }
65     public void agrupar() {
66         if(localizador!=null){
67             (argupador = localizador.argupador).add(this);
68         }
69     }
70     public void desagrupar() {
71         if(localizador!=null){
72             localizador.argupador.remove(this);
73         }
74     }
75     public void manejarAgrupacion() {
76         for(Horario a:argupador){
77             a.accion();
78         }
79     }
80     protected abstract void accion();
81 }

```

### A.2.2. Clase franja

```

82 import static com.componentes.disenio.lmc.marcosDeReferencia
    .computacion.Computacion.*;
83 import static com.componentes.disenio.lmc.marcosDeReferencia
    .emoticons.Emoticons.*;
84 public class Franja extends Horario{
85     private String nombreFranja;
86     private int tiempoInicio;
87     private int tiempoFinal;
88     private int horas;
89     private int disponibilidad;
90     private []int dias;
91     protected void accion() {

```

```
92         System.out.println(SERIOUS);
93     }
94 }
```

### A.3. Patrón Estrategia

#### A.3.1. Clase EstrategiaMod

```
95 public abstract class EstrategiaMod{
96     public abstract void Modificar();
97 }
```

#### A.3.2. Clase EstrategiaConcretaA

```
98 public class EstrategiaConcretaA extends EstrategiaMod{
99     public void Modificar(){}
100 }
```

#### A.3.3. Clase EstrategiaConcretaB

```
101 public class EstrategiaConcretaB extends EstrategiaMod{
102     public void Modificar(){}
103 }
```

#### A.3.4. Clase EstrategiaConcretaC

```
104 public class EstrategiaConcretaC extends EstrategiaMod{
105     public void Modificar(){}
106 }
```

#### A.3.5. Clase Tarea

```
107 public abstract class Tarea{
108     public abstract void operacion();
109     public void adicionar(Componente componente){}
110     public void remover(Componente componente){}
111     public Componente getHijo(int a){
112         return null;
113     }
114 }
```

#### A.3.6. Clase Subtarea

```
115 import java.util.List;
116 import java.util.ArrayList;
117 public class Subtarea extends Tarea{
```

```
118 private List<Tarea> hijo=new ArrayList<Tarea>();
119 public void operacion(){
120     //para todo g en hijo
121     for(Tarea g:hijo){
122         g.operacion();
123     }
124 }
125 public void adicionar(Componente componente){
126     hijo.add(componente);
127 }
128 public void remover(Componente componente){
129     hijo.remove(componente);
130 }
131 public Componente getHijo(int a){
132     return hijo.get(a);
133 }
134 }
```

#### A.3.7. Clase Tarea trabajo

```
135 public class TareaTrabajo extends Tarea{
136     private String descripcionTrabajo;
137 }
```

#### A.3.8. Clase Tarea consulta

```
138 public class TareaConsulta extends Tarea{
139     private String descripcionConsulta;
140     private boolean requiereConexion;
141     public void operacion(){}
142 }
```

#### A.3.9. Clase Tarea lectura

```
143 public class TareaLectura extends Tarea{
144     private int numeroPaginas;
145     private String nombreLectura;
146     public void operacion(){null}
147 }
```

### A.4. Patrón Fábrica Abstracta

#### A.4.1. Clase Tarea

```
148 public abstract class Tarea{
149     public abstract void operacion();
```

```
150 public void adicionar(Componente componente){}
151 public void remover(Componente componente){}
152 public Componente getHijo(int a){
153     return null;
154 }
155 }
```

#### A.4.2. Clase Subtarea

```
156 import java.util.List;
157 import java.util.ArrayList;
158 public class Subtarea extends Tarea{
159     private List<Tarea> hijo=new ArrayList<Tarea>();
160     public void operacion(){
161         //para todo g en hijo
162         for(Tarea g:hijo){
163             g.operacion();
164         }
165     }
166     public void adicionar(Componente componente){
167         hijo.add(componente);
168     }
169     public void remover(Componente componente){
170         hijo.remove(componente);
171     }
172     public Componente getHijo(int a){
173         return hijo.get(a);
174     }
175 }
```

#### A.4.3. Clase Tarea trabajo

```
176 public class TareaTrabajo extends Tarea{
177     private String descripcionTrabajo;
178 }
```

#### A.4.4. Clase Tarea consulta

```
179 public class TareaConsulta extends Tarea{
180     private String descripcionConsulta;
181     private boolean requiereConexion;
182     public void operacion(){}
183 }
```

#### A.4.5. Clase Tarea lectura

```
184 public class TareaLectura extends Tarea{
185     private int numeroPaginas;
186     private String nombreLectura;
187     public void operacion(){null}
188 }
```

#### A.4.6. Clase FabricaAbstracta

```
189 public abstract class FabricaAbstracta{
190     public abstract Tareaa CrearTarea();
191 }
```

#### A.4.7. Clase FabricaTTrabajo

```
192 public class FabricaTTrabajo extends FabricaAbstracta{
193     public Tareaa CrearTarea(){}
194 }
```

#### A.4.8. Clase FabricaTLectura

```
195 public class FabricaTLectura extends FabricaAbstracta{
196     public Tareaa CrearTarea(){}
197 }
```

#### A.4.9. Clase FabricaTConsulta

```
198 public class FabricaTConsulta extends FabricaAbstracta{
199     public Tareaa CrearTarea(){}
200 }
```

#### A.4.10. Clase Cliente

```
201 public class Cliente{
202     public static void main(String[] args){
203         FabricaAbstracta fab = new FabricaTTrabajo();
204         Tareaa Tar1 = fab.CrearTarea();
205         fab = new FabricaTLectura();
206         Tareaa Tar2 = fab.CrearTarea();
207     }
208 }
```





# Bibliografía