HABILITACION PROFESIONAL

Curso 4K4

Workflow Diseño

Establecimiento Metalúrgico David E. Bognanno

Docentes de Cátedra:
Zohil, Julio Cesar Nelson

Aquino, Francisco Alejandro

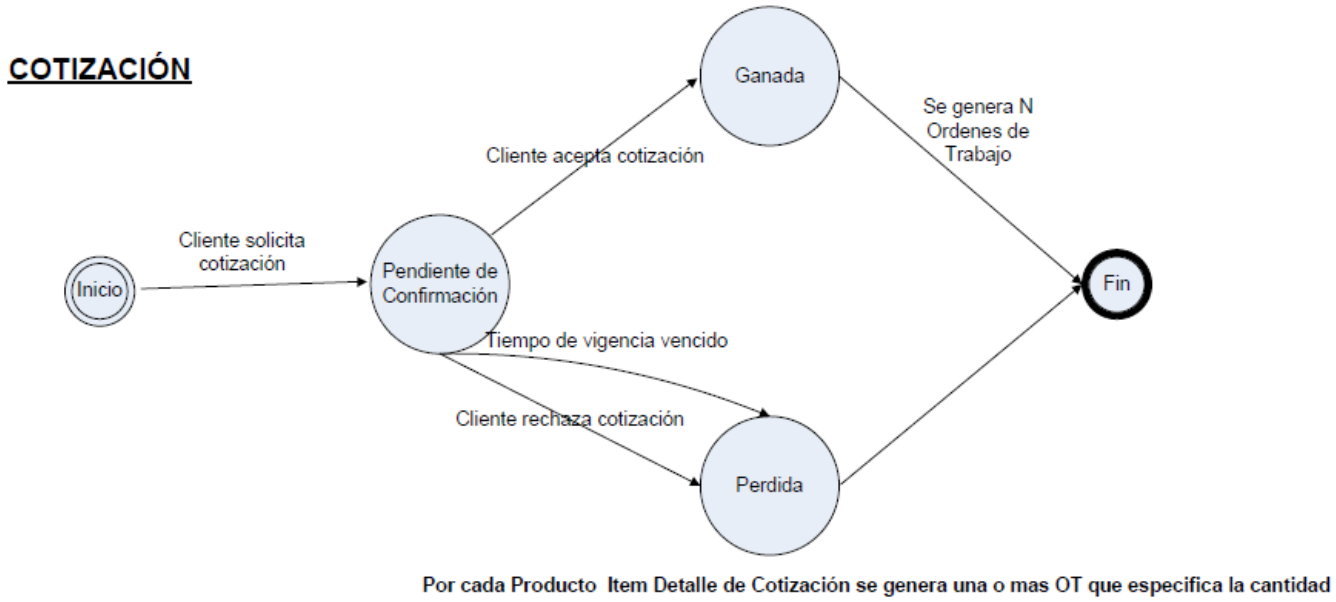
Jaime, María Natalia

Grupo N° 4

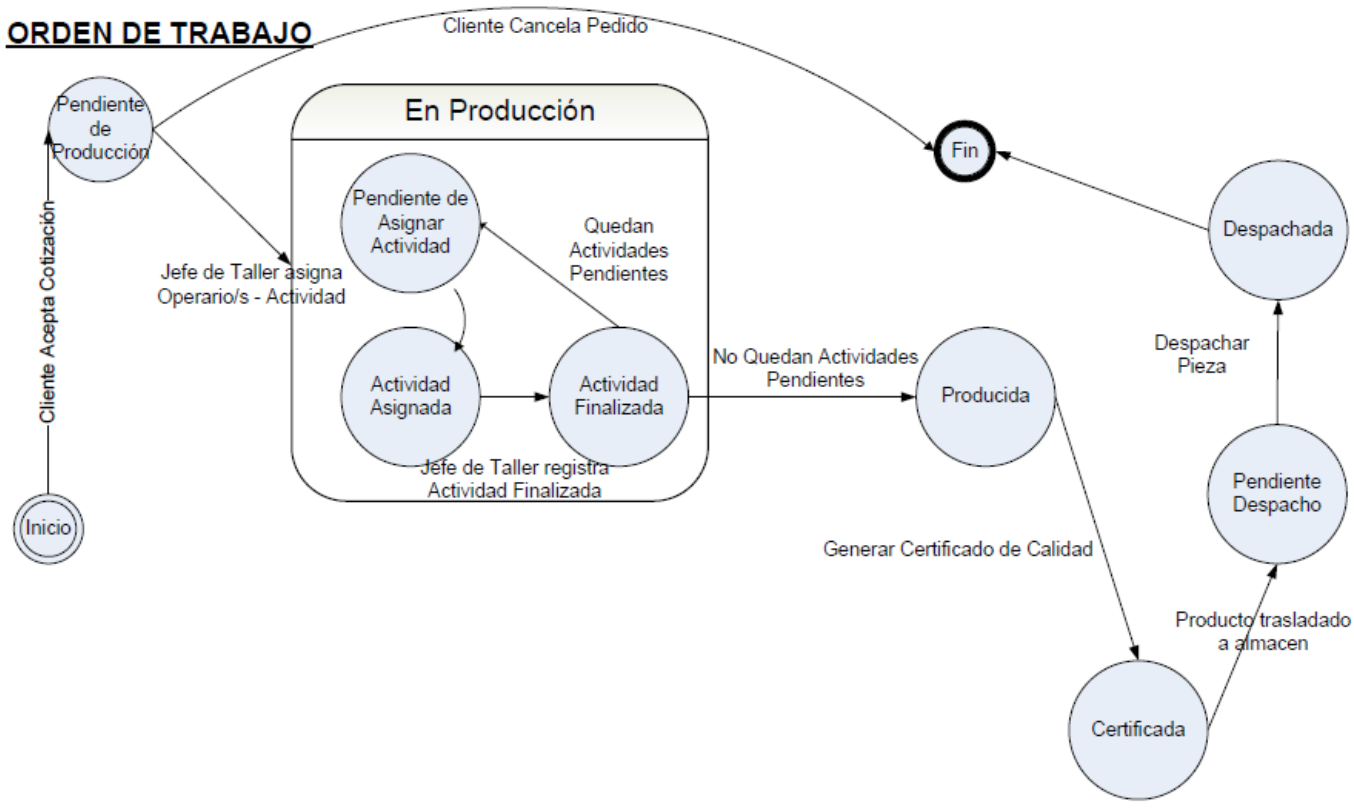
Juárez, Silvina	legajo 35284	silvina_v_juarez@hotmail.com
Martín, Pablo Andrés	legajo 47345	pablomartincmf@hotmail.com
Mengual, Rogelio Nicolás	legajo 53449	nicomengual@gmail.com
Molina, Mariano	legajo 28566	mmariano16@gmail.com

ÍNDICE

Diagramas de Transición de Estado



Por cada Producto Item Detalle de Cotización se genera una o mas OT que especifica la cantidad



Estados de clases

- ✓ **Cliente:** Activo - De Baja
- ✓ **Empleado:** Activo - De Baja
- ✓ **Producto:** Habilitado - Deshabilitado
- ✓ **Inspección:** Aprobada - No Aprobada
- ✓ **Pre-Diseño**: Cotizado - Sin Cotizar
- ✓ **Encuesta de Satisfacción:** Pendiente – Completa
- ✓ **Preguntas de encuesta:** Habilitada - Deshabilitada

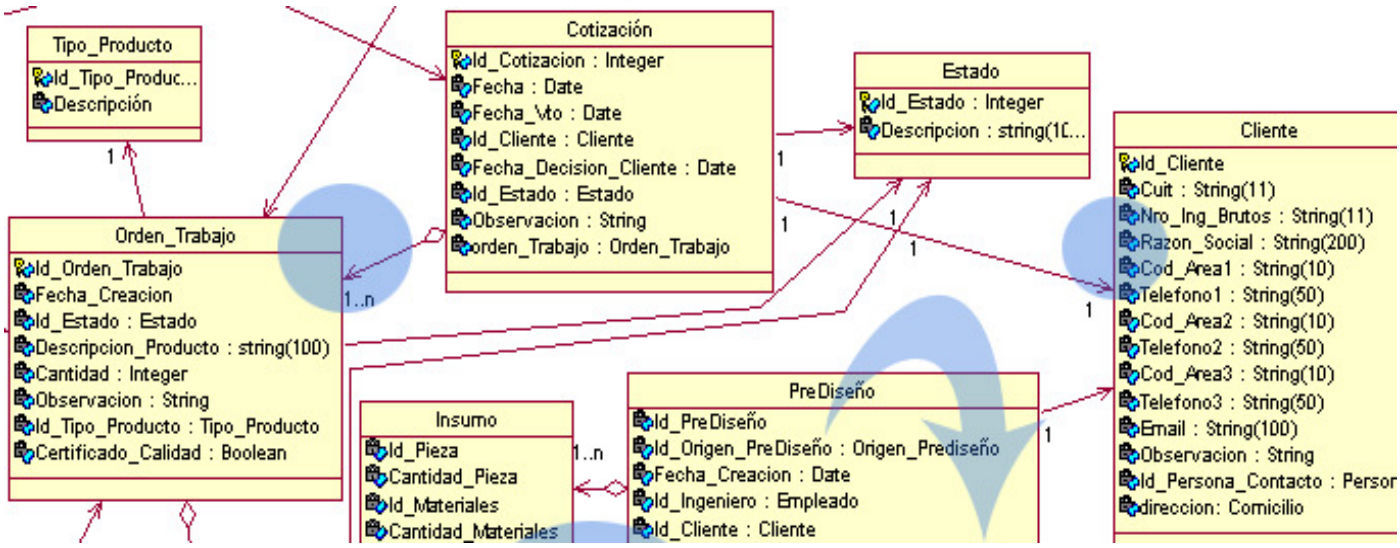
Diagrama de Clases de Diseño

Explicación de Patrones de Diseño utilizados

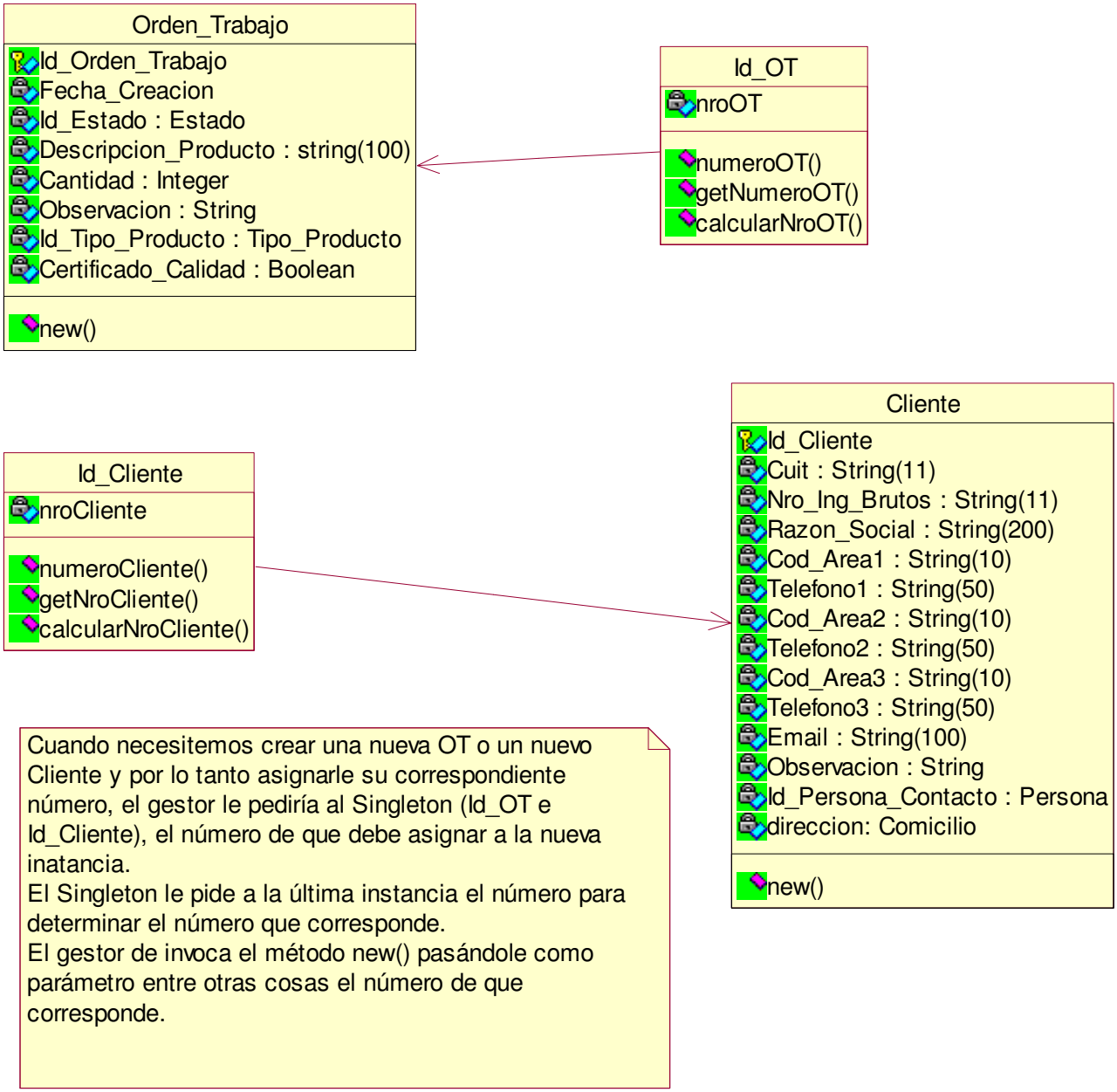
Patrones de Comportamiento

- **Singleton:** Muy útil cuando debe existir una única instancia de una clase, accesible globalmente.

Usaremos el patrón Singleton para la clase Orden de Trabajo, y para la clase Cliente:

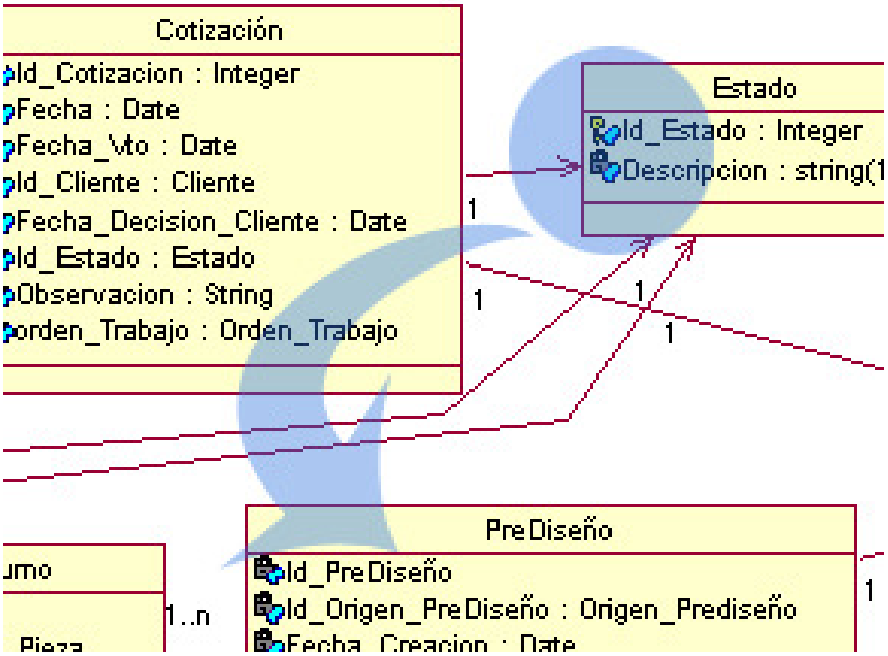


Aplicación

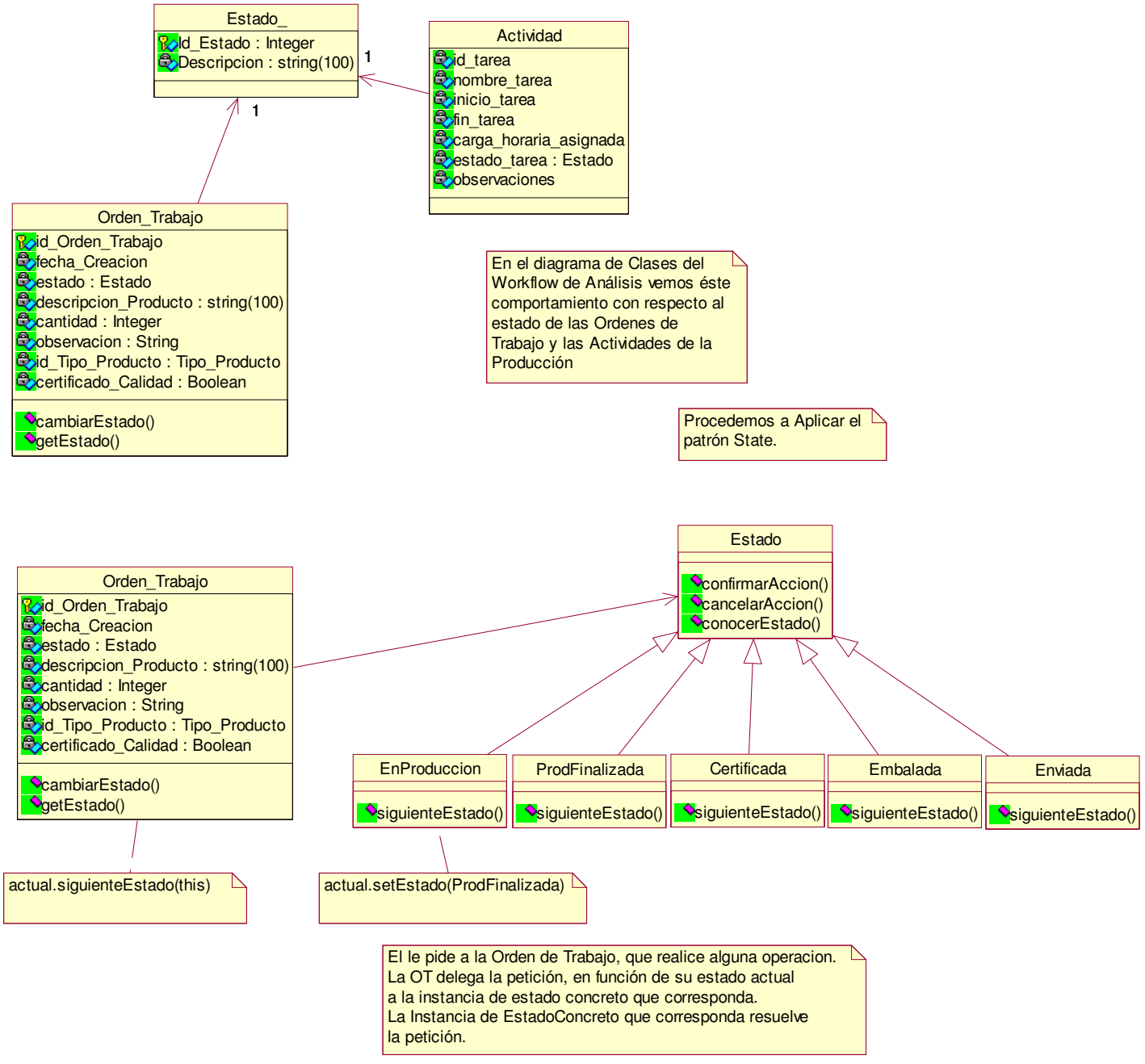

Orden_Trabajo
Id_Orden_Trabajo
Fecha_Creacion
Id_Estado : Estado
Descripcion_Producto : string(100)
Cantidad : Integer
Observacion : String
Id_Tipo_Producto : Tipo_Producto
Certificado_Calidad : Boolean
new()
Id_OT
nroOT
numeroOT()
getNumeroOT()
calcularNroOT()
Id_Cliente
nroCliente
numeroCliente()
getNroCliente()
calcularNroCliente()
Cliente
Id_Cliente
Cuit : String(11)
Nro_Ing_Brutos : String(11)
Razon_Social : String(200)
Cod_Area1 : String(10)
Telefono1 : String(50)
Cod_Area2 : String(10)
Telefono2 : String(50)
Cod_Area3 : String(10)
Telefono3 : String(50)
Email : String(100)
Observacion : String
Id_Persona_Contacto : Persona
direccion: Comicilio
new()
Cuando necesitemos crear una nueva OT o un nuevo Cliente y por lo tanto asignarle su correspondiente número, el gestor le pediría al Singleton (Id_OT e Id_Cliente), el número de que debe asignar a la nueva inatancia.
El Singleton le pide a la última instancia el número para determinar el número que corresponde.
El gestor de invoca el método new() pasándole como parámetro entre otras cosas el número de que corresponde.

- **State:** El patrón State permite que un objeto modifique su comportamiento cada vez que cambia su estado interno.

Esto nos será de utilidad para representar los estados de las Órdenes de Trabajo, y las Actividades de la producción:

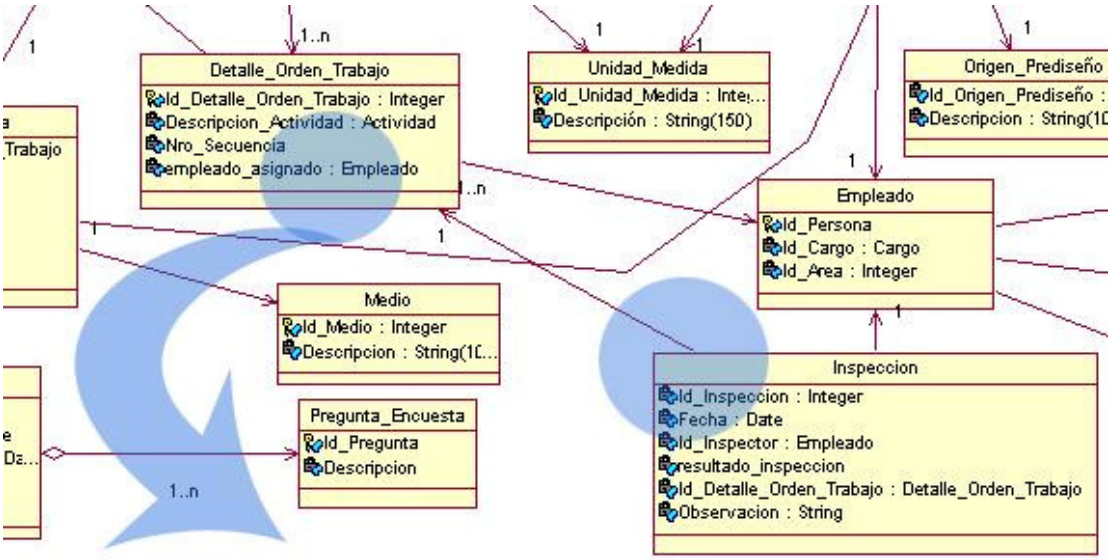


Aplicación


Estado_
Id_Estado : Integer
Descripcion : string(100)
1
1
Actividad
id_tarea
nombre_tarea
inicio_tarea
fin_tarea
carga_horaria_asignada
estado_tarea : Estado
observaciones
Orden_Trabajo
id_Orden_Trabajo
fecha_Creacion
estado : Estado
descripcion_Producto : string(100)
cantidad : Integer
observacion : String
id_Tipo_Producto : Tipo_Producto
certificado_Calidad : Boolean
cambiarEstado()
getEstado()
En el diagrama de Clases del Workflow de Análisis vemos éste comportamiento con respecto al estado de las Ordenes de Trabajo y las Actividades de la Producción
Procedemos a Aplicar el patrón State.
Estado
confirmarAccion()
cancelarAccion()
conocerEstado()
Orden_Trabajo
id_Orden_Trabajo
fecha_Creacion
estado : Estado
descripcion_Producto : string(100)
cantidad : Integer
observacion : String
id_Tipo_Producto : Tipo_Producto
certificado_Calidad : Boolean
cambiarEstado()
getEstado()
EnProduccion
siguienteEstado()
ProdFinalizada
siguienteEstado()
Certificada
siguienteEstado()
Embalada
siguienteEstado()
Enviada
siguienteEstado()
actual.siguienteEstado(this)
actual.setEstado(ProdFinalizada)
El le pide a la Orden de Trabajo, que realice alguna operacion.
La OT delega la petición, en función de su estado actual
a la instancia de estado concreto que corresponda.
La Instancia de EstadoConcreto que corresponda resuelve
la petición.

- **Observer:** Define una dependencia del tipo uno-a-muchos entre objetos, de manera que cuando uno de los objetos cambia su estado, el observador se encarga de notificar este cambio a todos los otros objetos dependientes, quienes actúan en concordancia.

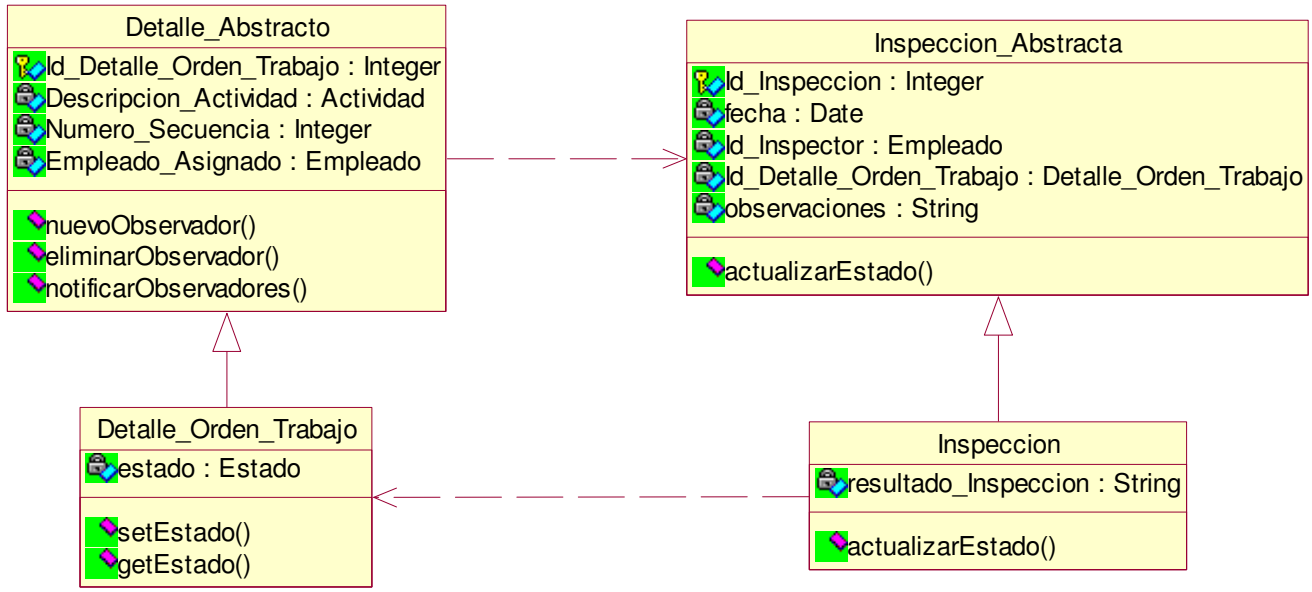
Usaremos el patrón Observer para la clase Detalle_Orden_Trabajo (Clase observada), y para la clase Inspección (Clase Observadora):



Patro Observer Implementado sobre clases Detalle_Orden_Trabajo e Inspeccion

Aplicación

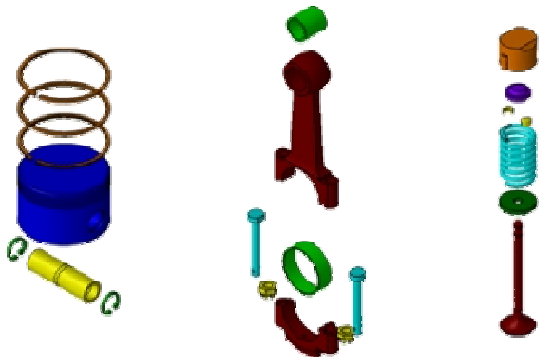
La Clase Inspeccion es el Observador que se interesa en saber cuando el estado de un Detalle_Orden_Trabajo para a estado "Finalizado", para cambiar su propio estado a "Inspeccion Pendiente" y asi habilitar al Inspector para realizar la inspeccion.

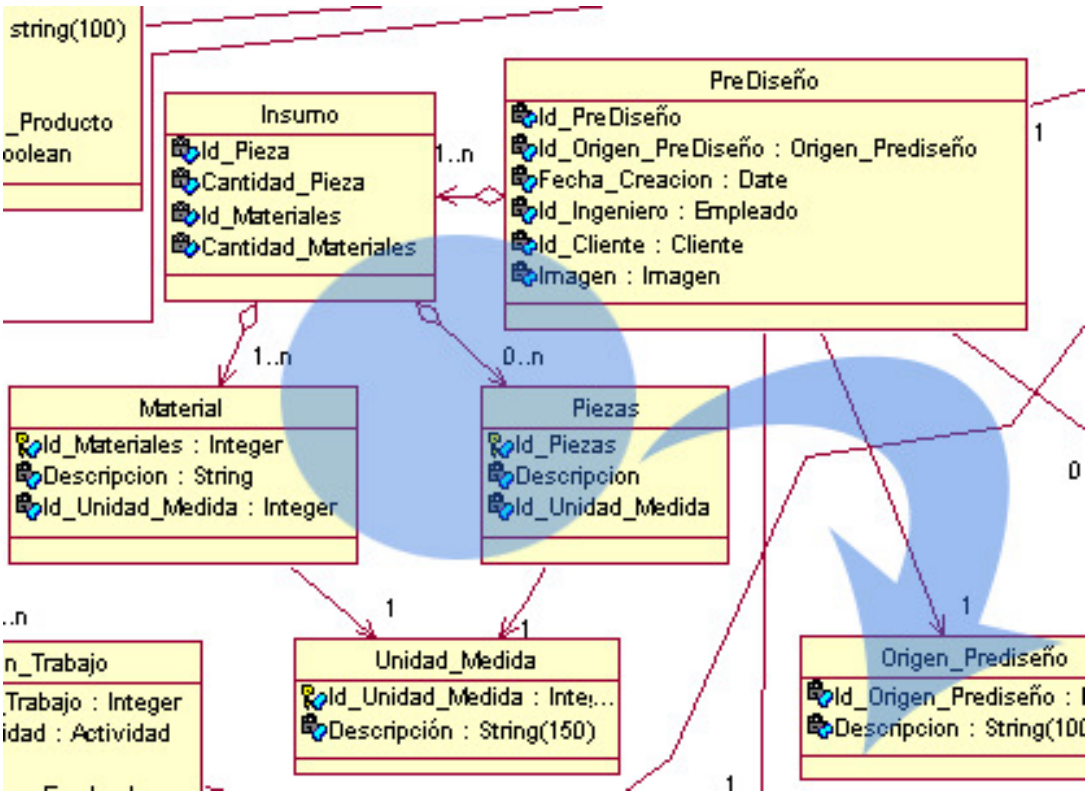


Patrones de Estructura

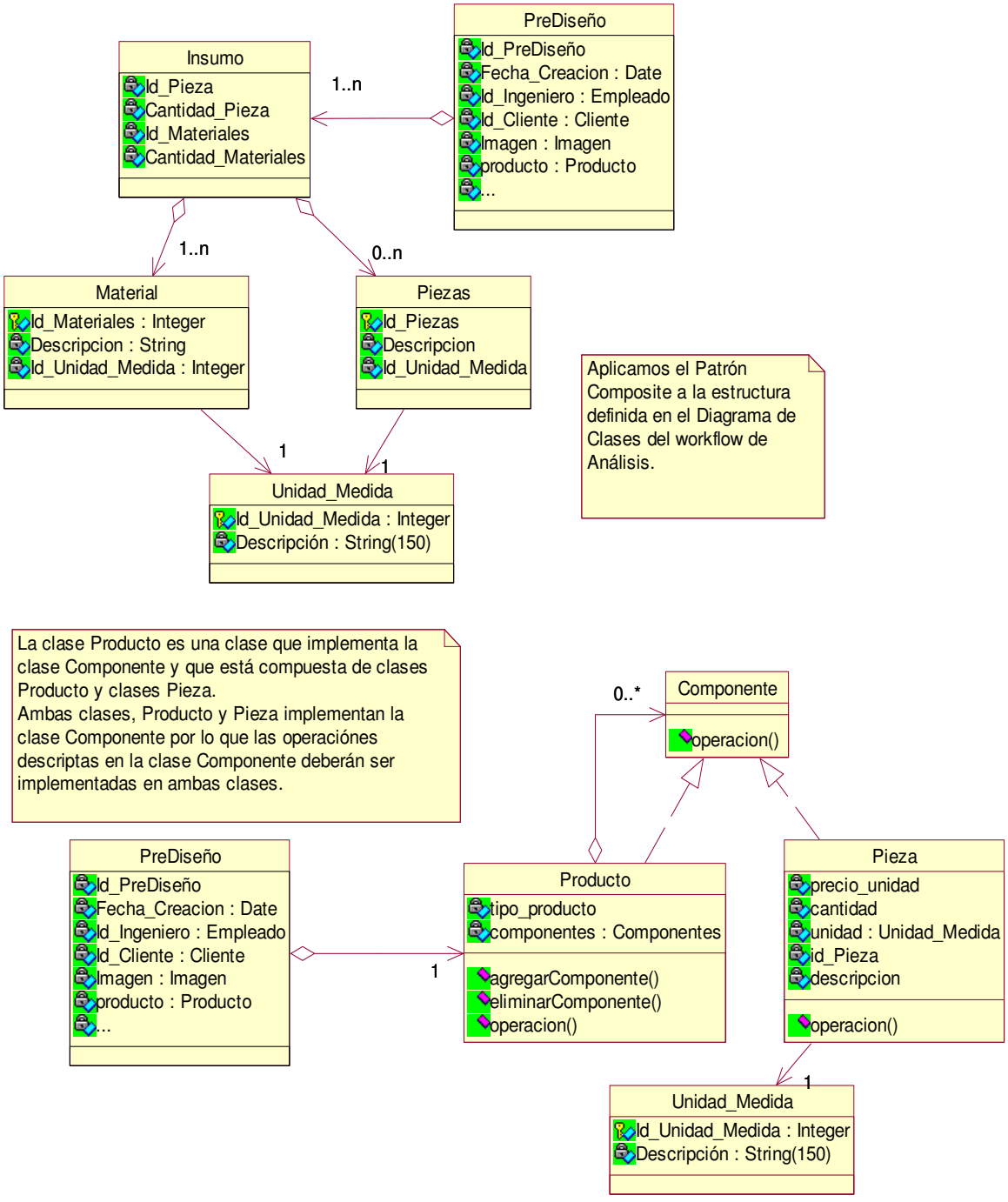
- **Composite:** Resulta útil para definir la estructura de productos y partes de productos que lo conforman.

Modelaremos la estructura que usa la empresa para la producción de partes únicas (Producto), formada de partes con las que contamos como materia prima (Pieza):

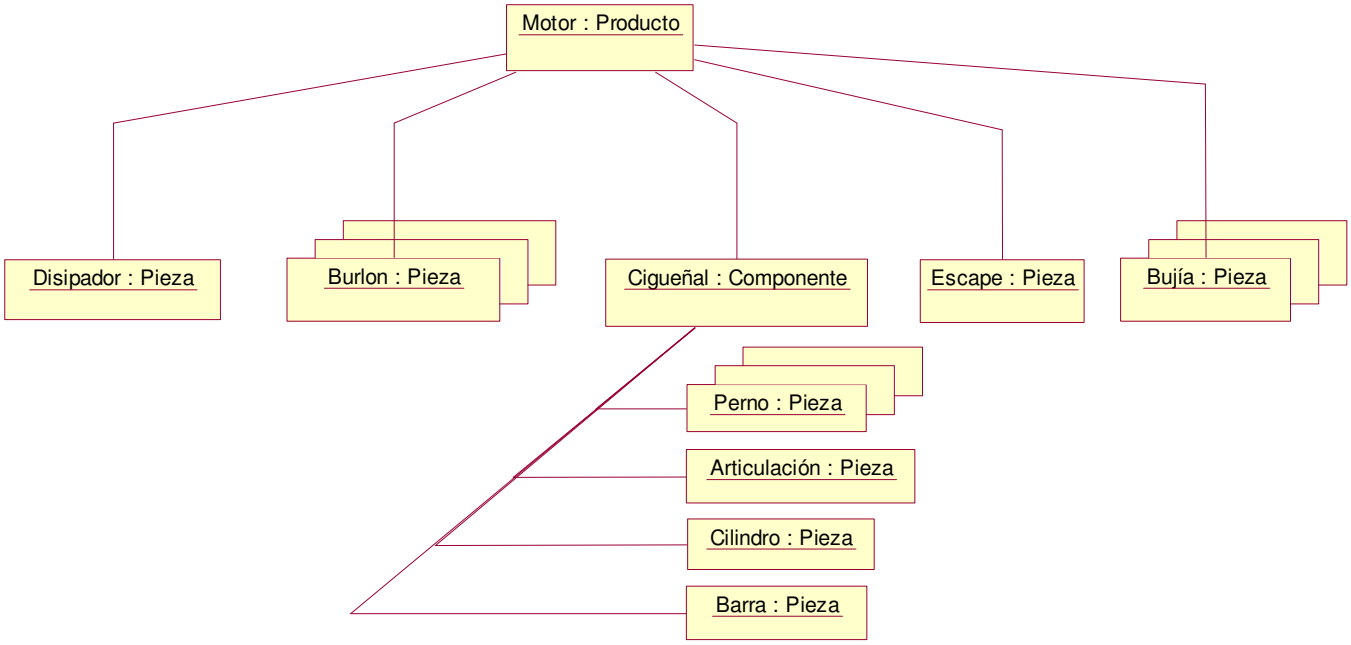




Aplicación


Insumo
Id_Pieza
Cantidad_Pieza
Id_Materiales
Cantidad_Materiales
1..n
PreDiseño
Id_PreDiseño
Fecha_Creacion : Date
Id_Ingeniero : Empleado
Id_Cliente : Cliente
Imagen : Imagen
producto : Producto
...
1..n
0..n
Material
Id_Materiales : Integer
Descripcion : String
Id_Unidad_Medida : Integer
Piezas
Id_Piezas
Descripcion
Id_Unidad_Medida
1
1
Unidad_Medida
Id_Unidad_Medida : Integer
Descripción : String(150)
Aplicamos el Patrón Composite a la estructura definida en el Diagrama de Clases del workflow de Análisis.
La clase Producto es una clase que implementa la clase Componente y que está compuesta de clases Producto y clases Pieza.
Ambas clases, Producto y Pieza implementan la clase Componente por lo que las operaciónes descriptas en la clase Componente deberán ser implementadas en ambas clases.
0..*
Componente
operacion()
PreDiseño
Id_PreDiseño
Fecha_Creacion : Date
Id_Ingeniero : Empleado
Id_Cliente : Cliente
Imagen : Imagen
producto : Producto
...
1
Producto
tipo_producto
componentes : Componentes
agregarComponente()
eliminarComponente()
operacion()
Pieza
precio_unidad
cantidad
unidad : Unidad_Medida
id_Pieza
descripcion
operacion()
1
Unidad_Medida
Id_Unidad_Medida : Integer
Descripción : String(150)

Ejemplo del patrón instanciado

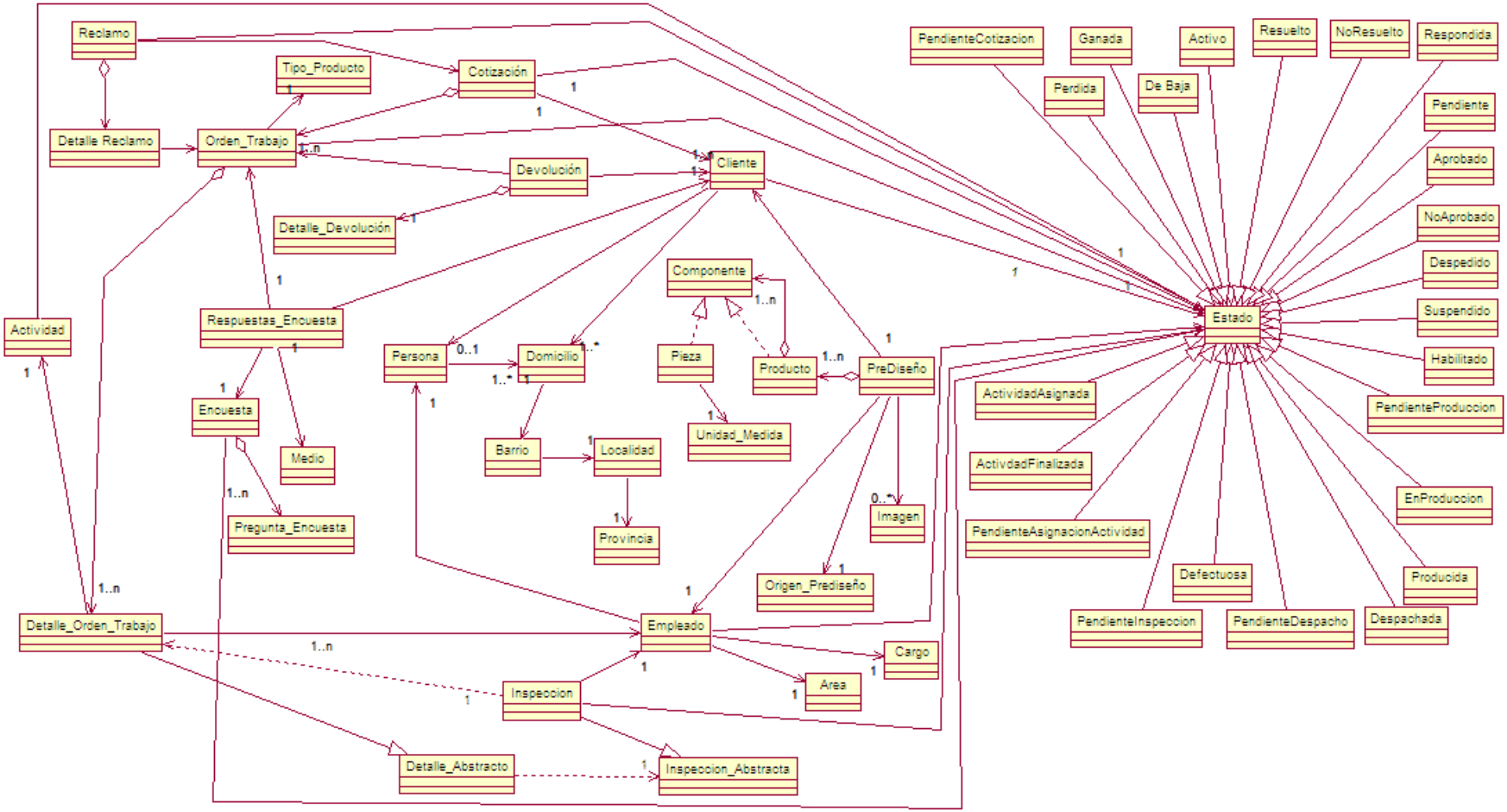


Aquí podemos reconocer:

2 clases Producto (que es el compuesto del patrón composite)

8 clases Pieza que son las clases que componen el Producto

Diagrama de Clases con aplicación de Patrones



Detalle de Clases

CLASES DE DISEÑO