Contenido

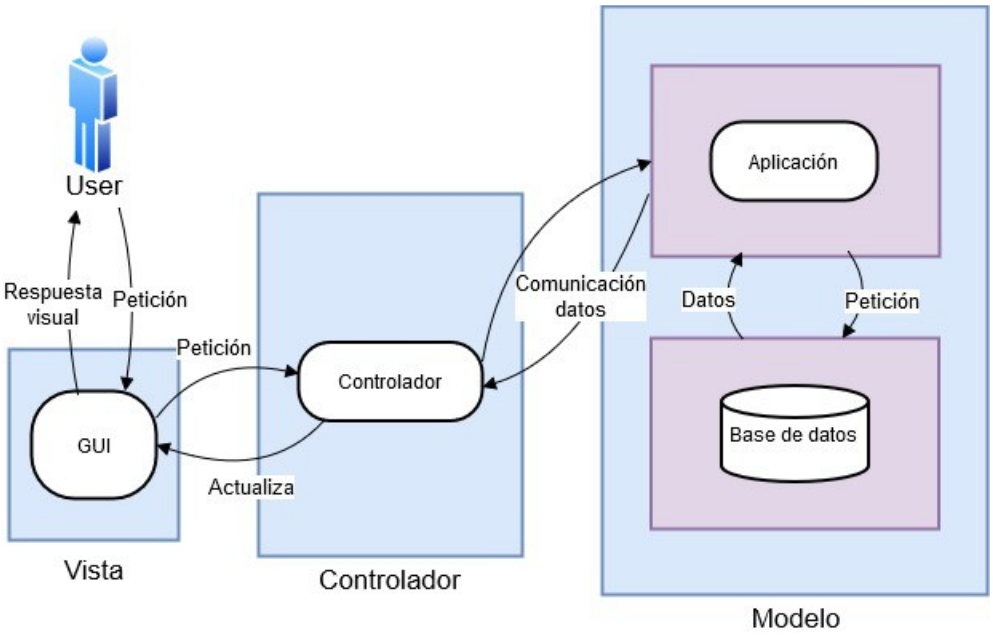
10. Aplicación de patrones

10.1. Patrón de arquitectura Modelo-Vista-Controlador

<Explicación textual de como se ha utilizado el patrón e imagen del diagrama (clases y/o componentes) en el que se vea su aplicación.>

Para conseguir desacoplar los objetos del dominio y las vistas además de gestionar correctamente los eventos generados por los actores, hemos recurrido a una arquitectura MVC. Gracias a esta, conseguimos que el intermediario entre los objetos de dominio y las vistas sea únicamente el controlador. Actúa como nexo de conexión entre ambos elementos, trasladando a la vista los datos que se requieran del modelo, y viceversa. Decidirá qué elementos del modelo intervienen para responder a cada evento del sistema y aportar esos datos. Por ello, para cada posible evento generado por un autor el controlador queda dotado de las funciones necesarias para pedir a las clases de dominio que realicen su parte del trabajo en el momento y orden que este decide.

Arquitectura MVC gráfica:



10.2. Patrones de diseño

<Para cada uno de los patrones de diseño elegidos, al menos tres, explicación textual de como se ha utilizado el patrón e imagen del diagrama (clases y/o colaboración) en el que se vea su aplicación. Se utilizarán patrones vistos en teoría y no se puede utilizar el patón Controlador ya que se usa en el patrón de arquitectura>

10.2.1. Fachada de Base de Datos

La fachada ofrece una interfaz única y común al servidor de base de datos para todas las clases y funciones del sistema.

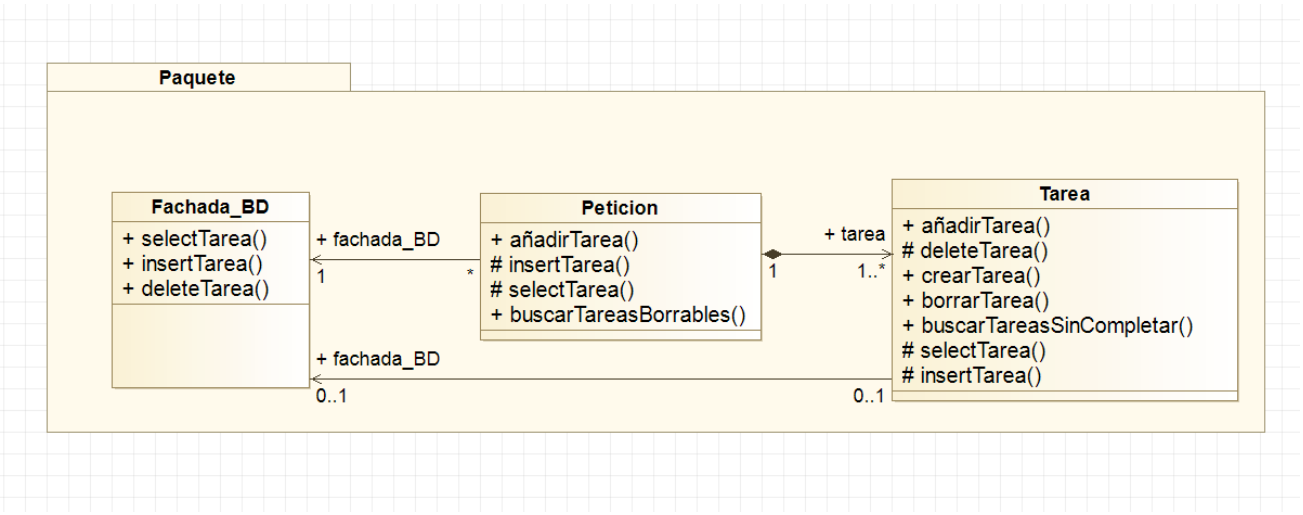
Se ha implementado como una única clase con métodos para seleccionar, insertar y eliminar instancias de cada tabla de la base de datos. Salvo por las clases hijas cada clase está relacionada con una tabla en la base de datos. Las clases con relación de herencia comparten tabla en el modelo relacional.

Como cada método de la fachada está relacionado a una clase, la manera por las que las clases llaman a la fachada es mediante un método con visibilidad protegida en la clase que llama a la fachada. Es decir, las clases tienen métodos protegidos que llaman a los métodos respectivos de la fachada, los cuales son de visibilidad pública.

Los métodos protegidos en las clases se utilizan mediante llamadas por el resto de los métodos de la propia clase, así pueden trabajar con la base de datos sin requerir manejarla directamente.

Por esta solución se aumenta la cohesión relacionando directamente todas las clases con la base de datos. También reduce el acoplamiento ya que existe una clase intermedia entre los métodos llamadores y la propia base de datos. Gracias a este patrón la mantenibilidad se ve beneficiada y debería de significar un aumento en la calidad y rendimiento totales de la aplicación.

Como ejemplo de una aplicación del patrón fachada, este es el diagrama que representa las relaciones con la clase Fachada_BD para gestionar las instancias de clase Tareas. (Por simplificar el diagrama no se han incluido atributos ni parámetros, solo los métodos relacionados y si son públicos, privados o protegidos).



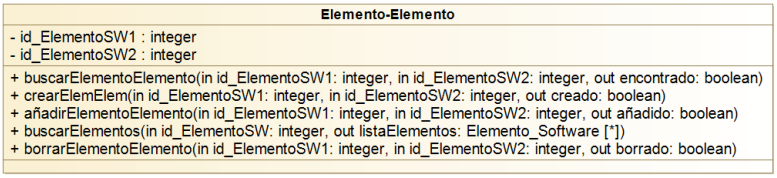
10.2.2. Experto

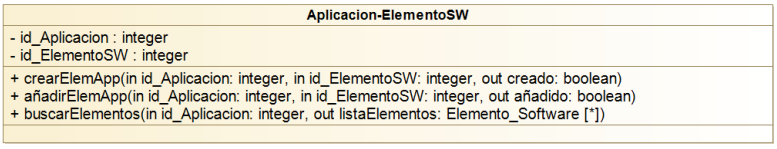
Existen clases intermedias que gestionan las relaciones entre clases, estas son UsuarioCliente-Aplicacion, Aplicacion-ElementoSW, Elemento-Elemento y Peticion-Elemento.

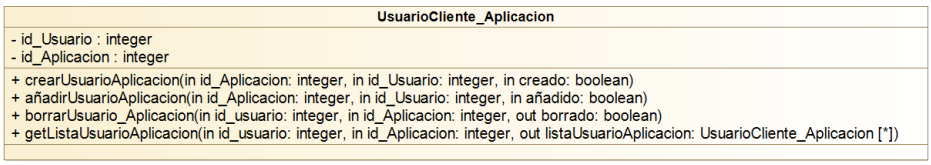
Estas clases asumen la responsabilidad de manejar la información que se transfiere entre las dos clases que se designan en su nombre, separadas por un guion “-“.

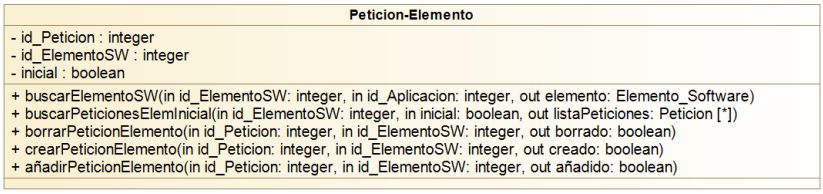
Conviene utilizar un patrón para estas relaciones ya que mediante ellas se asignan usuarios a aplicaciones, elementos software a aplicaciones y relacionarlos entre ellos. Estas asignaciones son información clave y dedicada y se debe tratar independientemente de los objetos que relaciona en sí.

Los diagramas para cada una de las clases mencionadas sería la siguiente, solo se relacionaría a las clases mencionadas en el nombre de cada experto:









Cabe mencionar que el uso de expertos modifica los diagramas de colaboración, estos han sido anexados al final del documento para explicar cómo se aplican los patrones expertos.

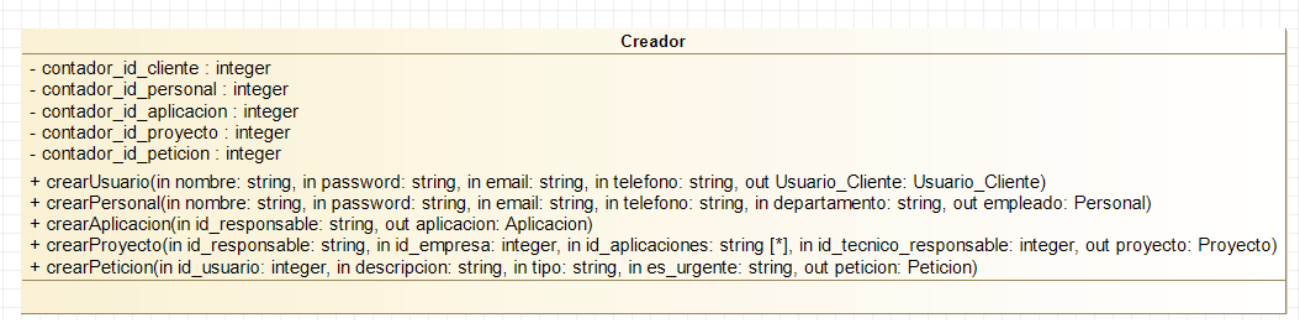
10.2.3. Creador

Existen funciones creadoras de objetos ajenas a las clases y no aparecen en el diagrama de clases. Particularmente para crear usuarios, proyectos, aplicaciones y peticiones.

Estas se utilizan mediante la interfaz de la aplicación web a la hora de inscribirse como usuario, donde genera los objetos relacionados a la cuenta y al cliente o empleado. También se refiere a las clases que utiliza el sistema cuando un usuario crea una aplicación, petición o proyecto, dependiendo de si es un usuario cliente o un empleado de la empresa.

Las tareas y los elementos software son creados por las clases petición y aplicación respectivamente, estas dos clases podrían beneficiarse del uso de un patrón creador.

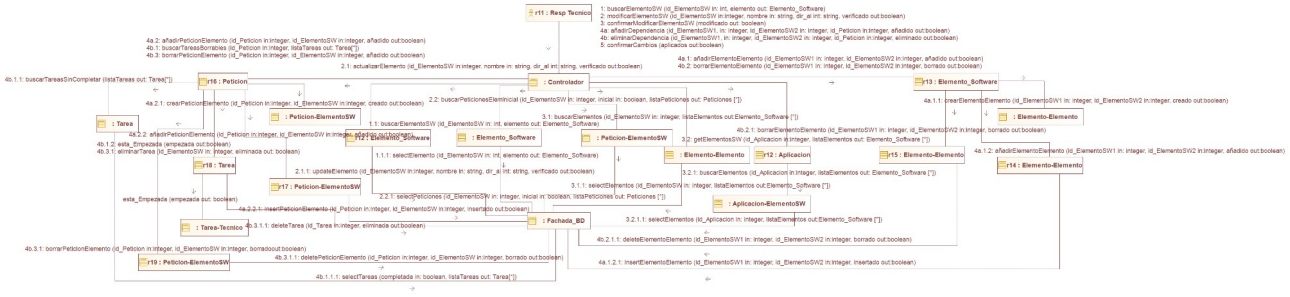
Para su implementación contiene atributos que toman el valor del identificador más reciente de cada clase que crea, de forma que al crear una instancia no repita el identificador y aumente automáticamente. Su diagrama sería el siguiente:



Anexo. Diagramas de colaboracion con clase experto

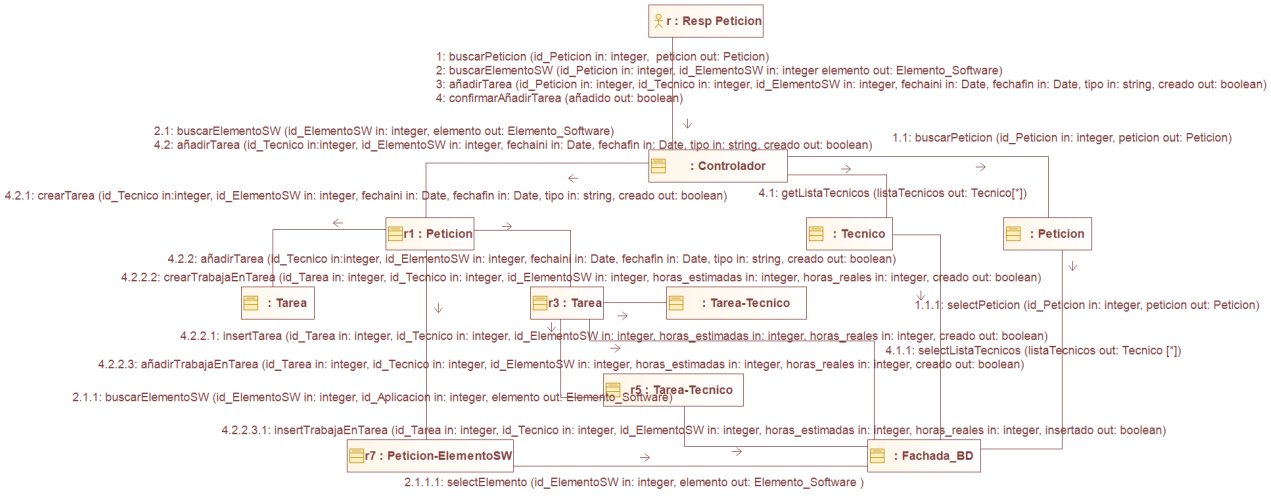
Caso de Uso – Modificar elementos SW

Uso de expertos Peticion-ElementoSW, Aplicación-ElemetoSW y Elemento-Elemento.



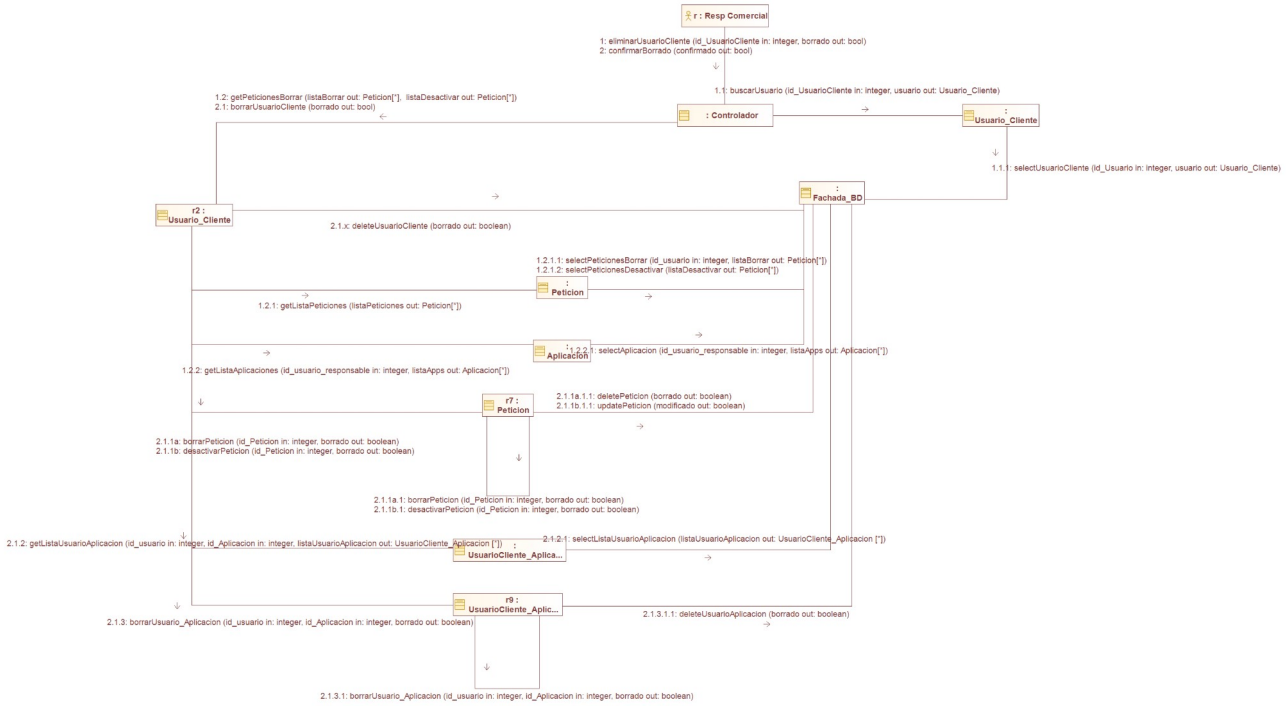
Caso de Uso – Crear tareas asociadas a petición

Uso de expertos Peticion-ElementoSW y Tarea-Tecnico.



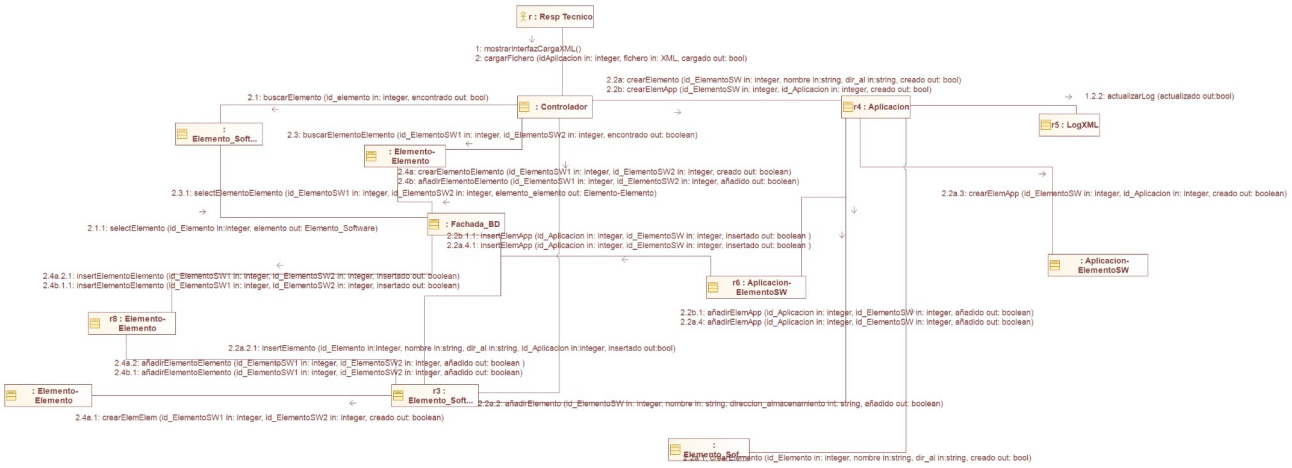
Caso de Uso – Borra usuarios cliente

Uso de expertos Usuario_Cliente-Aplicación.



Caso de Uso – Cargar ficheros XML

Uso de expertos Aplicación-ElemetoSW y Elemento-Elemento


r : Resp Tecnico
1: mostrarInterfazCargaXML()
2: cargarFichero (idAplicacion in: integer, fichero in: XML, cargado out: bool)
2.2a: crearElemento (id_ElementoSW in: integer, nombre in:string, dir_al in:string, creado out: bool)
2.2b: crearElemApp (id_ElementoSW in: integer, id_Aplicacion in: integer, creado out: bool)
1.2.2: actualizarLog (actualizado out:bool)
2.1: buscarElemento (id_elemento in: integer, encontrado out: bool)
: Controlador
r4 : Aplicacion
r5 : LogXML
: Elemento_Soft...
2.3: buscarElementoElemento (id_ElementoSW1 in: integer, id_ElementoSW2 in: integer, encontrado out: boolean)
: Elemento-Elemento
2.4a: crearElementoElemento (id_ElementoSW1 in: integer, id_ElementoSW2 in: integer, creado out: boolean)
2.4b: añadirElementoElemento (id_ElementoSW1 in: integer, id_ElementoSW2 in: integer, añadido out: boolean)
2.3.1: selectElementoElemento (id_ElementoSW1 in: integer, id_ElementoSW2 in: integer, elemento_elemento out: Elemento-Elemento)
2.2a.3: crearElemApp (id_ElementoSW in: integer, id_Aplicacion in: integer, creado out: boolean)
2.1.1: selectElemento (id_Elemento in:integer, elemento out: Elemento_Software)
: Fachada_BD
2.2b.1.1: insertElemApp (id_Aplicacion in: integer, id_ElementoSW in: integer, insertado out: boolean)
2.2a.4.1: insertElemApp (id_Aplicacion in: integer, id_ElementoSW in: integer, insertado out: boolean)
: Aplicacion-ElementoSW
2.4a.2.1: insertElementoElemento (id_ElementoSW1 in: integer, id_ElementoSW2 in: integer, insertado out: boolean)
2.4b.1.1: insertElementoElemento (id_ElementoSW1 in: integer, id_ElementoSW2 in: integer, insertado out: boolean)
r6 : Aplicacion-ElementoSW
r8 : Elemento-Elemento
2.2b.1: añadirElemApp (id_Aplicacion in: integer, id_ElementoSW in: integer, añadido out: boolean)
2.2a.4: añadirElemApp (id_Aplicacion in: integer, id_ElementoSW in: integer, añadido out: boolean)
2.2a.2.1: insertElemento (id_Elemento in:integer, nombre in:string, dir_al in:string, id_Aplicacion in:integer, insertado out:bool)
2.4a.2: añadirElementoElemento (id_ElementoSW1 in: integer, id_ElementoSW2 in: integer, añadido out: boolean)
2.4b.1: añadirElementoElemento (id_ElementoSW1 in: integer, id_ElementoSW2 in: integer, añadido out: boolean)
: Elemento-Elemento
r3 : Elemento_Soft...
2.2a.2: añadirElemento (id_ElementoSW in: integer, nombre in: string, direccion_almacenamiento int: string, añadido out: boolean)
2.4a.1: crearElemElem (id_ElementoSW1 in: integer, id_ElementoSW2 in: integer, creado out: boolean)
: Elemento_Sof...
2.2a.1: crearElemento (id_Elemento in: integer, nombre in:string, dir_al in:string, creado out: bool)