Modelos de análisis y diseño

GESTIÓN DE PERSONAL UCM

Miguel Pascual Domínguez, Javier Pellejero Ortega, Isabel Pérez Pereda, Iván Prada Cazalla, Jesús Recio Herranz, Álvaro Rodríguez García

Gestor personal UCM

Proyecto Ingeniería del Software

Email portavoz: alvarr11@ucm.es

Contenido

Tabla de modificaciones

Autor	Fecha	Versión	Descripción
Grupo completo	25/3/2016	1.0	Creación del documento
Iván Prada Cazalla	8/4/2016	1.1	División en trozos del apartado de analisis y comienzo de la tarea
Iván Prada Cazalla	15/4/2016	1.2	Completar análisis
Iván Prada Cazalla	22/4/2016	1.3	Inicio del apartado de diseño. Estructuración
Iván Prada Cazalla	29/4/2016	2.0	Añadidos diagramas
Iván Prada Cazalla	13/5/2016	2.1	Añadidos diagramas
Jesús Recio Herranz e Isabel Pérez Pereda	24/5/2016	2.2	Casos de Prueba
Miguel Pacual Dominguez	25/5/2016	2.3	Revisión
Jesús Recio Herranz	25/5/2016	2.4	Revisión
Isabel Pérez Pereda	25/5/2016	2.5	Revisión
Álvaro Rodriguez Garcia	25/5/2016	2.6	Revisión
Iván Prada Cazalla	25/5/2016	3.0	Revisión

Modelo de Análisis

Análisis del sistema de software

En este apartado nos vamos a dedicar a hacer un analisis pormenorizado del software. Se trata de realizar un sistema que se encargará de simular el funcinamiento de una aplicación encargada de llevar el sistema de gestión del personal de la universidad complutense.

Los objetivos, requisitos y restricciones del sistema ya quedáron expuestos en el modelo de dominio y en los documentos de requisitos.

Vamos a hacer una división en una serie de niveles contextuales que nos mostrarán una mejor visión del sistema.

Análisis de la arquitectura (paquetes, clases, requisitos)

En este apartado procederemos a identificar los paquetes, clases evidentes y requisitos especiales comunes que formarán inicialmente parte de nuestro sistema.

Para la identificación de paquetes, agruparemos por funcionalidad común (para los dos grandes subsistemas), y organizaremos los paquetes por capas, para dar una cierta estructuración a nuestra aplicación (ya que tenemos que utilizar un MVC y una arquitectura multicapa).

Tendremos los siguientes paquetes agrupados por funcionalidad común(subsistemas):

- **Empleado**
  El subsistema de empleado será el encargado de realizar todas las operaciones correspondientes a la inserción, busqueda, visualización y edición de los empleados que se demanden.
- **Usuario**
  El subsistema de usuario será el encargado de realizar todas las operaciones relacionadas con la creación y la eliminación de los usuarios que pueden acceder al sistema, así como de gestionar los permisos asociados a estos.

Estos subsistemas a su vez estarán divididos en una serie de paquetes, que agruparán una funcionalidad común, y que estarán en cada uno de los subsistemas, además de las clases principales que tendrian:

- **Vistas y controlador**

Paquetes encargados de gestionar las interfaces gráficas
Clases iniciales para empleado y usuario: vista genérica.

- **Lógica y reglas**
  Encargados de la seguridad y la lógica de nuestro programa
  Clases iniciales para empleado: *Empleados*, *AreaDeTrabajo*, *Contrato*, *EmpleadoPAS*, *EmpleadoPDI*, *TipoBaja*, *TipoDocente*
  Clases iniciales para usuario: *Usuario*, *TipoFacultad*, *TipoPermiso*.

La agrupación por capas sería la siguiente(aunque esto se considera diseño):

- *Capa de integración*
  Contendra a todas las clases y paquetes que realicen una comunicación con la base de datos.
- *Capa de negocio*
  En ella se situarán todas las clases y paquetes encargados de llevar la lógica de negocio.
- *Capa de presentación*
  En esta capa irán todas las clases y paquetes destinados a la visualización de los datos solicitados por el actor y procesados por el sistema.

Descripción del dominio de la información

En este apartado haremos una descripción del dominio de la información, es decir, de los tipos de datos que manejará nuestra aplicación.

Se manejaran los distintos tipos de datos:

Tipo empleado: representa a una persona dada de alta en nuestro sistema, y que vendrá descrita por :

- Identificador: es un numero unico y designado por el sistema, con el que un empleado quedará identificado.
- Nombre: nombre del empleado que se introducirá en el sistema.
- Apellido1: primer apellido del empleado.
- Apellido2: segundo apellido del empleado.

- Path_foto: foto adjunta al empleado que quedará registrada en el sistema.
- Dirección: ubicación en la que vive el empleado.
- Idiomas: conjunto de idiomas que conoce el empleado.
- HistorialPath: historial referente al empleado.
- NominaPath: nomina del empleado.
- Contrato: contrato del empleado.
- CurriculumPath: curriculum del empleado.
- TipoBaja: en caso de haber una baja de un empleado cursada, quedará aquí registrada.
- Facultad: facultad en la que se encuentra el empleado.

Si es PDI:

- Departamento: departamento al que pertenece el empleado.
- Especialidad: materia en la que el empleado está especializado.
- Despacho: despacho destinado al empleado.
- TipoDocente: a seleccionar entre ayudante, asociado, titular, catedratico.

Si es PAS:

- ÁreaTrabajo: asignación de tareas que desempeña el empleado.
- Categoría: categoria a la que pertenece el empleado.

Tipo contrato: información almacenada referente a un contrato. Vendrá descrito por:

- Cuenta bancaria: número de cuenta del banco en la que se hará el ingreso al empleado.
- HorasDeTrabajo: número de horas de trabajo designadas a ese empleado.
- FechaFin: fecha del fin de contrato, en caso de que sea temporal.
- esTemporal: nos indicará si un empleado es temporal.

Tipo Usuario: representará a un usuario del sistema, y vendrá definido por:;

- Nombre: nombre del usuario, con el que accederá a su sesión.
- Contraseña: clave del usuario para acceder al sistema.
- TipoFacultad: facultad a la que pertenece (matematicas, fisica, quimica, biologia, informatica, medicina, ninguna…).

- TipoPermiso: permiso que restringe la consulta de acceso a empleados(superusuario, administrador_rectorado, administrador_facultad, secretario_pas, secretario_pdi).

Modelo de Diseño

En este apartado pasamos a diseñar el sistema a partir de los puntos estudiados en la parte de requisitos y análisis.

En primera instancia haremos un esquema general de los subsistemas ya mencionados en la sección de análisis, y pasaremos a describir las partes que lo componen. Para ello utilizaremos diagramas de clases, de secuencia, y de actividades.

Subsistema Empleado: diagramas de clases

Como hemos dicho en el análisis, este subsistema se encarga de gestionar todo lo relativo a los empleados: añadir, ver, editar y eliminar. Dividiremos el sistema en tres capas, utilizando la arquitectura multicapa. De esta forma, tendremos la capa de presentación (encargada de ls visualización de la aplicación), la capa de negocio (encargada de la gestión interna de la aplicación y de su lógica), y la capa de integración (que gestiona la base de datos).

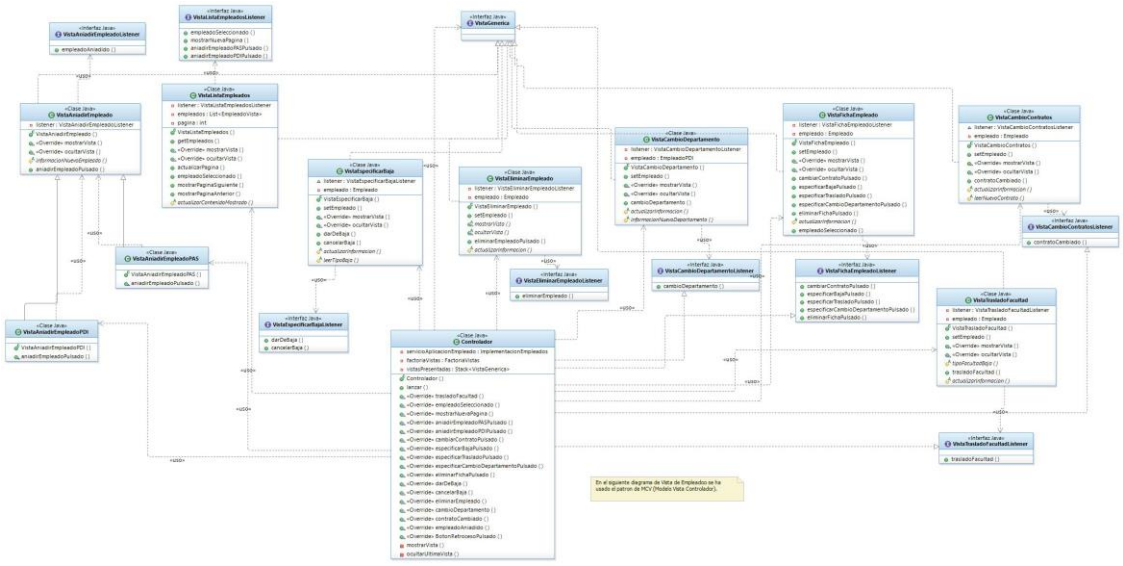
Capa de presentación

Para realizar la visualización de nuestro subsistema, utilizaremos un patrón Modelo-Vista-Controlador (MVC). Para ello creamos una interfaz llamada *VistaGenérica* que implementará cada una de nuestras vistas. Cada vista tiene un objeto *Listener* que cumple un propósito similar al *Patrón Observador* de forma que los cambios provocados por el usuario (como por ejemplo pulsar un boton) sean notificados a la clase que implementa el correspondiente *listener*. En nuestro caso es el controlador tanto el que maneja las vistas, mostrandolas y ocultandolas, como el que implementa todos estos *listener*; de forma que es el controlador el que reacciona a todas las acciones del actor/usuario, creando este nuevas vistas, o interaccionando con el servicio de aplicación si es necesario.

Lo importante del patrón MVC es que permite tener en clases separadas el código que se encarga de mostrar las cosas (*vista*) del modelo (*model*) y de la lógica que se encarga de, por ejemplo, elegir qué vista se muestra en cada momento o solicitar la información de un empleado a la base de datos (*controller*).

Es interesante observar que hemos implementado el *Patrón Factoria* para crear nuestras vistas. En efecto, todas nuestras vistas son abstractas para no limitarnos a una implementación concreta, y estar abiertos a distintas posibles interfaces gráficas. De esta forma, el controlador

solicita la vista apropiada a una factoria abstracta que proporcionara la vistra adecuada según el tipo de interfaz en el que estemos (ventana, consola…).



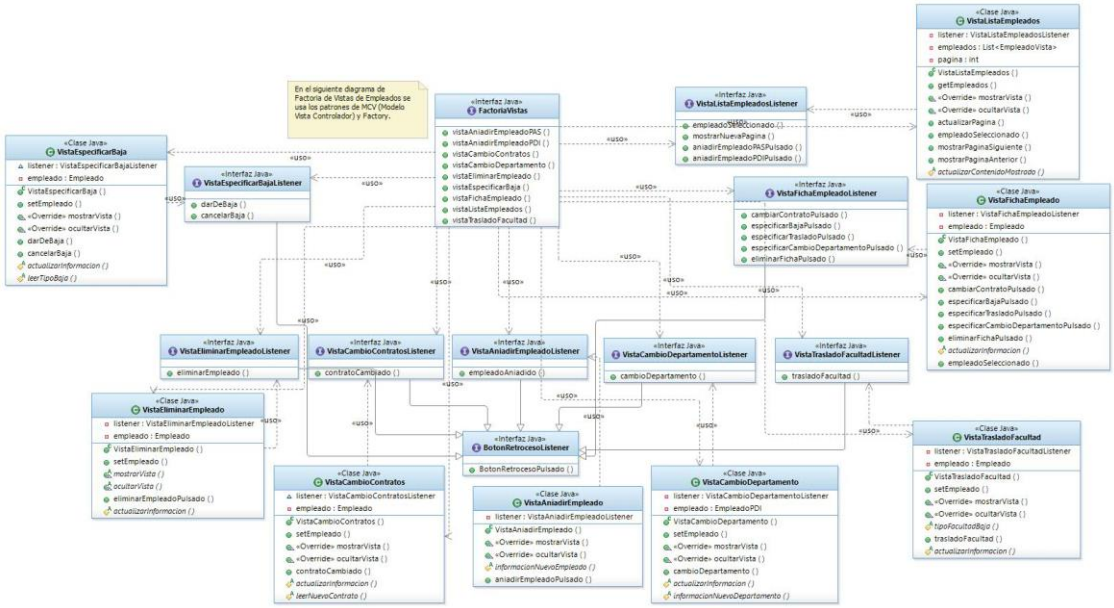
Como cada una de las vistas se refiere a un caso de uso, cuando hablamos de implementar el formulario asociado a una acción, este tiene los campos que ya enumeramos en el caso de uso correspondiente (ver documento *Especificacion_de_requisitos*).

A continuación describiremos la funcionalidad de cada una de las vistas implementadas en el diagrama:

- *VistaAniadirEmpleado*: crea el formulario para añadir un empleado de tipo genérico.
- *VistaAniadirEmpleadoPAS*: es una subclase de la anterior que proporciona los campos concretos de un *EmpleadoPAS*.
- *VistaAniadirEmpleadoPDI*: es una subclase *VistaAniadirEmpleado* de la que proporciona los campos concretos de un *EmpleadoPDI*
- *VistaCambioContratos*: implementa el formulario para cambiar el contrato de un empleado.
- *VistaCambioDepartamento*: permite cambiar el departamento de un empleado.
- *VistaEliminarEmpleado*: muetra la opción al usuario de eliminar un empleado del sistema.

- *VistaEspecificarBaja*: muestra el formulario para dar de baja a un usuario o cancelar una baja ya establecida.
- *VistaFichaEmpleados*: muestra la información completa de un empleado concreto.
- *VistaListaEmpleados*: muestrea una lista de empleados (por nombre y apellidos), por páginas (20 empleados por cada página), realizado así para optimizar la visualización de estos.
- *VistaTransladoFacultad*: muestra un formulario para especificar el translado de facultad.

El *listener* asociado a cada vista se nombra mediante: "<nombre de la vista>*Listener".*



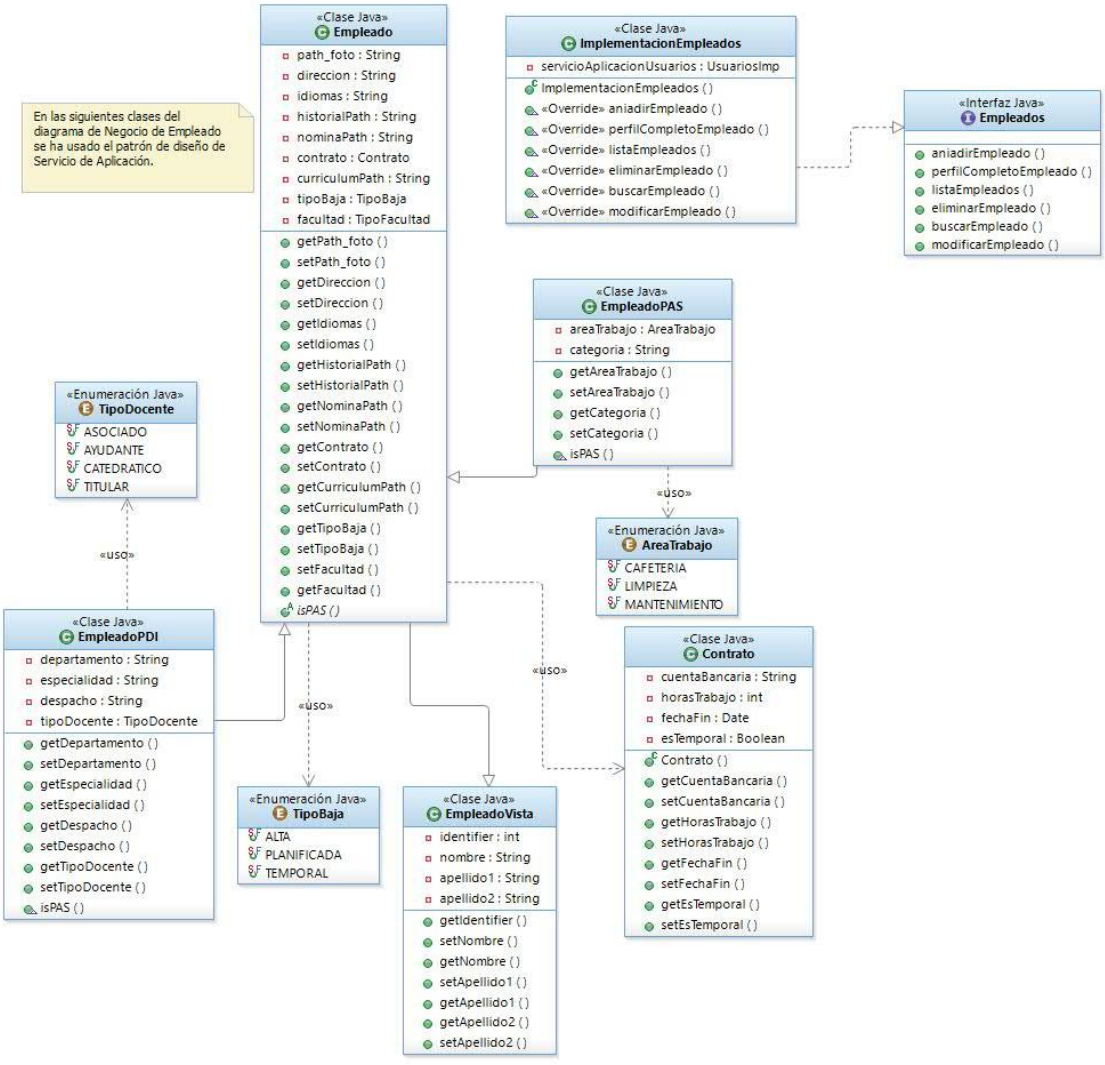
La clase *FactoríaVistas* implementa la factoria ya comentada, y tiene un método correspondiente a cada tipo de vista que se puede instanciar.

La interfaz *BotonRetrocesoListener* es la parte común de todas las interfaces de cada uno de los tipos de vistas. Proporciona funcionalidad para salir, retroceder, cancelar…, en general, volver a la vista anterior.

Capa de negocio

En esta capa se encontrará la lógica de nuestra aplicación, que será la encargada de procesar las peticiones de la vista (que comprobará, y que en caso de ser válidas llevará a cabo) y comunicará estas peticiones a la capa de integración, que desarrollaremos posteriormente.

En esta capa utilizaremos el patrón *Servicio de aplicación* para permitir a la capa de presentación interactuar con la capa de negocio.



Hemos dividido este subsistema en tres grandes grupos de clases. Por un lado, tenemos la *lógica* de nuestra aplicación, que es la parte que implementa el patrón *Servicio de aplicación*; por otro lado, tenemos las *reglas*, que son los modelos de los objetos del mundo real (empleados, facultades, contratos…); y por último tenemos los *objetos de transferencia*, que aunque sean compartidos por todas las capas, tienen sentido en esta capa debido a que es la que más los utiliza.

Lógica

Esta compuesta tan solo por la interfaz *empleados* y una clase que la implementa, *implementaciónEmpleados.* La utilidad de esta interfaz es aislar la implementación de los métodos frente a las demás capas, de forma que puedan utilizar los métodos sin conocer su funcionamiento interno. Estos métodos que implementa nuestra clase se encargan de interaccionar con la capa de integración, y también contienen toda la lógica, que en este caso se limita a comprobar los permisos. Para ello, tiene acceso a la clase análoga de la *capa de negocio* del módulo *Usuarios*, de nuevo a través de una interfaz.

Reglas

Son el modelo de nuestra aplicación, que incluye las clases:

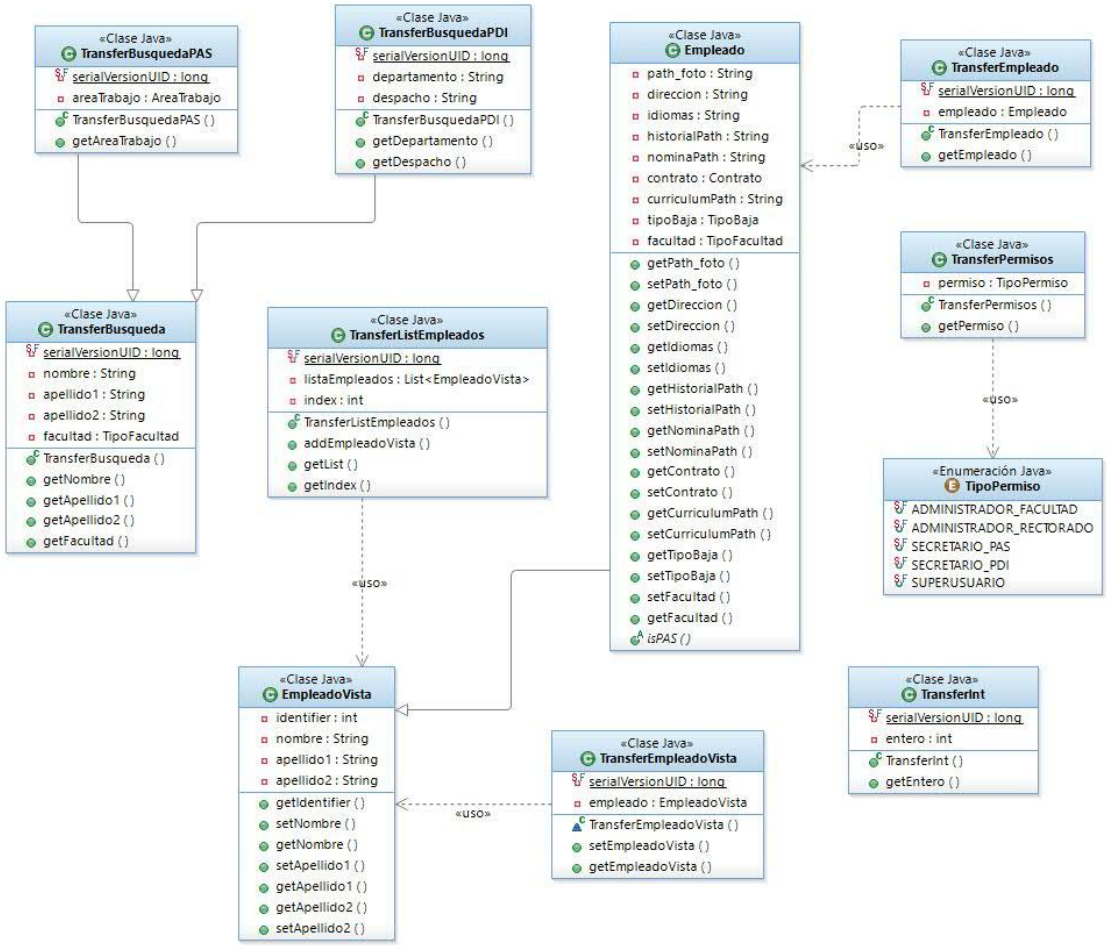
- *ÁreaTrabajo*: designa que tipo de actividad realiza el empleado (cafeteria, limpieza, mantenimiento…).
- *Contrato*: almacenará toda la información relativa a un contrato (horas de trabajo, cuenta bancaria, fecha de fin del contrato…).
- *Empleado*: definirá un empleado.
- *EmpleadoPAS*: definirá un empleado de tipo PAS.
- *EmpleadoPDI*: definirá un empleado de tipo PDI.
- *EmpleadoVista*: almacena los datos referentes a un empleado para mostrarse.
- *TipoBaja*: define si la baja es una alta, temporal, y planificada.
- *TipoDocente*: (asociado, ayudante, catedrático y titular)

Objetos de transferencia

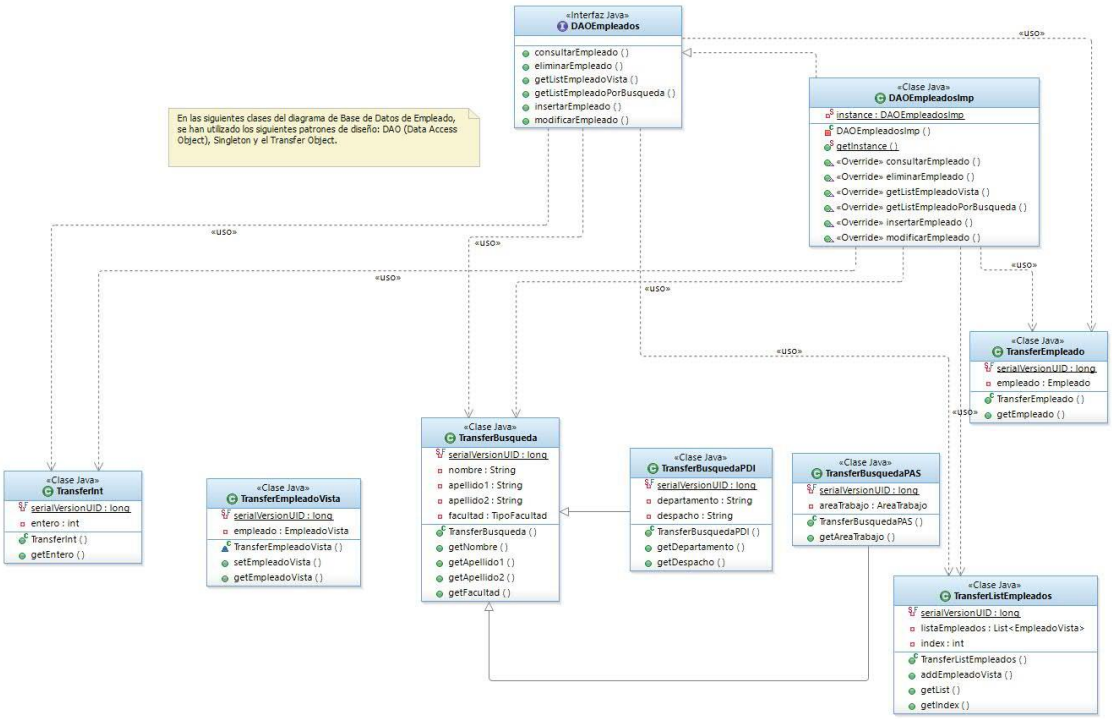
Los objetos de transferencia son usados por todo el módulo para transferir información entre distintas capas. Tenemos los siguientes tipos: *TransferBusqueda*, *TransferBusquedaPAS*, *TransferBusquedaPDI*, *TransferEmpleado*, *TransferEmpleadoVista*, *TransferInt*, *TransferListEmpleados*, *TransferPermisos*, cuyo nombre es descriptivo de lo que contienen. Los

objetos de transferencia no tienen lógica ni código, solo encapsulan la información de otra cosa (un entero, un empleado...).

El diagrama de estos objetos de transferencia se adjunta a continuación:



Capa de integración



En esta capa se realizará el intercambio de datos con la unidad de almacenamiento de información. Dicha unidad puede ser física, digital, etc. En nuestro caso, los requisitos de la aplicación especificaban que nuestra información se guardaba en una base de datos, aunque nuestro código no hace asunciones en ese sentido.

Hemos utilizado el Patrón de diseño *DAO* ("*Data Access Objec*t"), que nos permite transferir datos específicos sin la necesidad de saber los detalles del almacenamiento que haya detrás. Como consecuencia, la utilización de este Patrón aportará seguridad en la capa de negocio ante cambios en la organización o en la estructura de los datos en la base de datos.

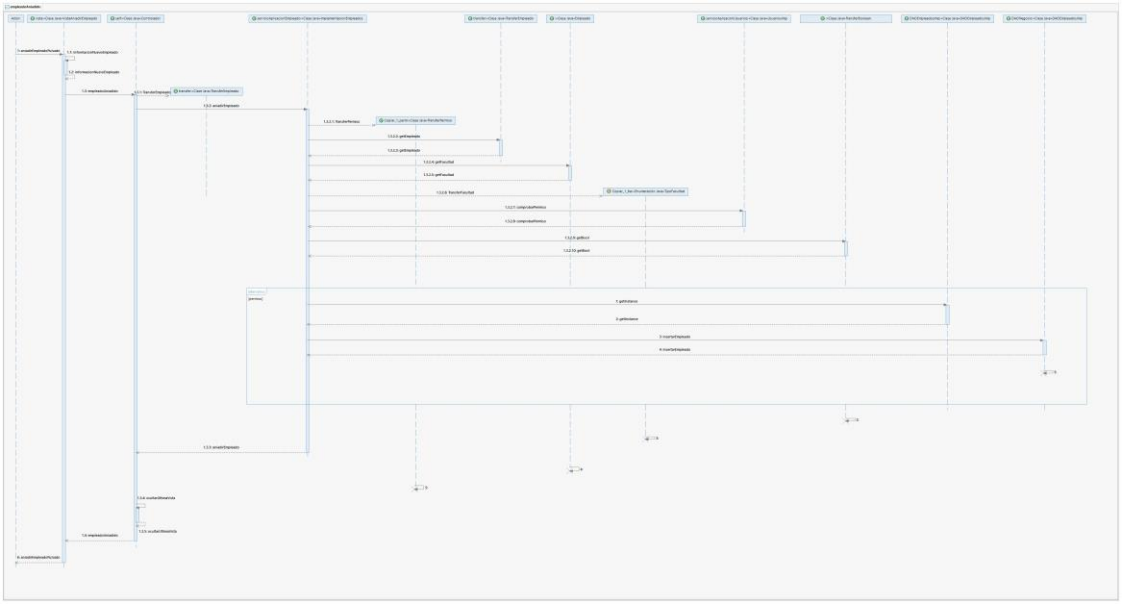
Así, la capa de integración se compone básicamente de una interfaz "*DAOEmpleados*", que proporciona métodos específicos donde devolverá objetos de transferencia. Esta interfaz será implementada por la clase "*DAOEmpleadosImp*". Para evitar que circulen un número grande de instancias de este tipo por nuestra aplicación y evitar conexiones simultáneas de varios objetos

distintos a la base de datos, hemos implementado el Patrón *Singleton*. De esta forma, privatizamos el constructor y sólo se crea una única instancia de *DAOEmpleadosImp*, que es compartida por todos los objetos que la necesitan a través del método *getInstance*.

A continuación pasaremos a hacer un recorrido por los distintos casos de uso que forman parte de este subsistema, describiendolas mediante diagramas de secuencia.

Subsistema empleados: diagramas de secuencia

Añiadir empleado



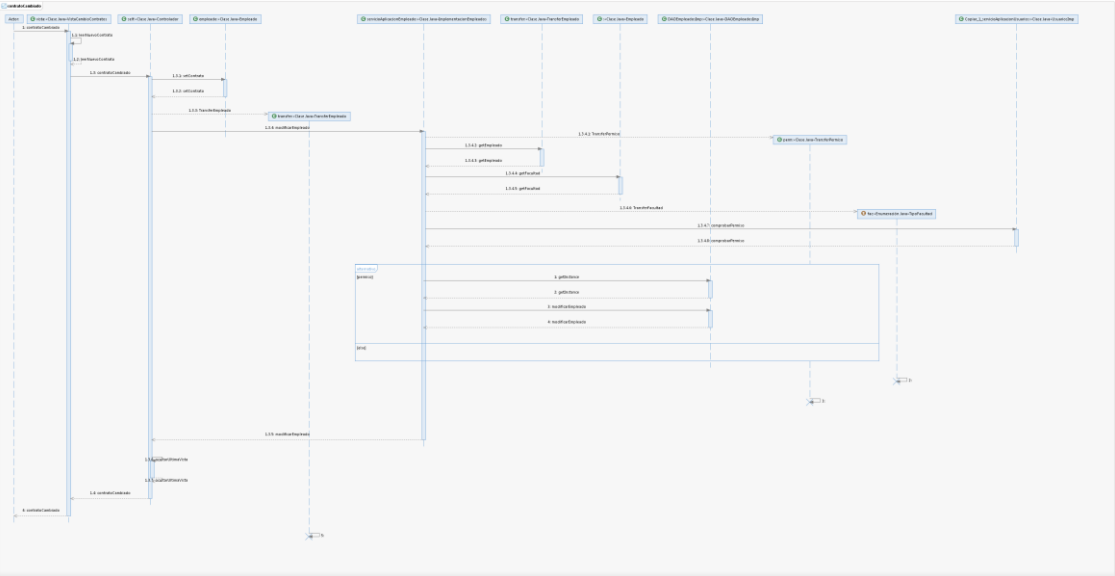
El siguiente diagrama de secuencias describe el caso de uso correspondiente a añadir un nuevo empleado independientemente si es de PDI o PAS a la base de datos de empleado.

La secuencia comienza con la acción del actor o usuario llamando a la acción de crear un nuevo empleado. Se pedirá al usuario la información correspondiente al nuevo usuario, la cual sera encapsulada en un *TransferEmpleado*, que obtendrá el *Controlador*. Este controlador llamará a su método *aniadirEmpleado()*, el cual obtendra la facultad del *TransferEmpleado* mediante el metodo *getFacultad()* el cual es posible gracias a haber obtenido el empleado completo con *getEmpleado()*.

Esta facultad es un dato que el *serviciodeAplicaciónEmpleado* de la capa de negocio necesita para llamar a *comprobacionPermisos*, que recibe dos argumentos de tipo *TransferPermiso* y

TransferFacultad. Este método que determinará si el usuario posee los permisos necesarios para crearlo. En el caso de que el usuario tenga permiso, la capa de negocio, mediante el servicio de aplicación, le pedirá al *DAO* que inserte el nuevo empleado en la base de datos con su método *insertarEmpleado()*, en caso contrario el *Controlador* mostrara un mensaje de error y no se realizará la acción de añadir empleado. Finalmente el *Controlador* ocultará la vista de añadir empleado y se recuperará la vista anterior en la que comenzó.

Cambiar contrato

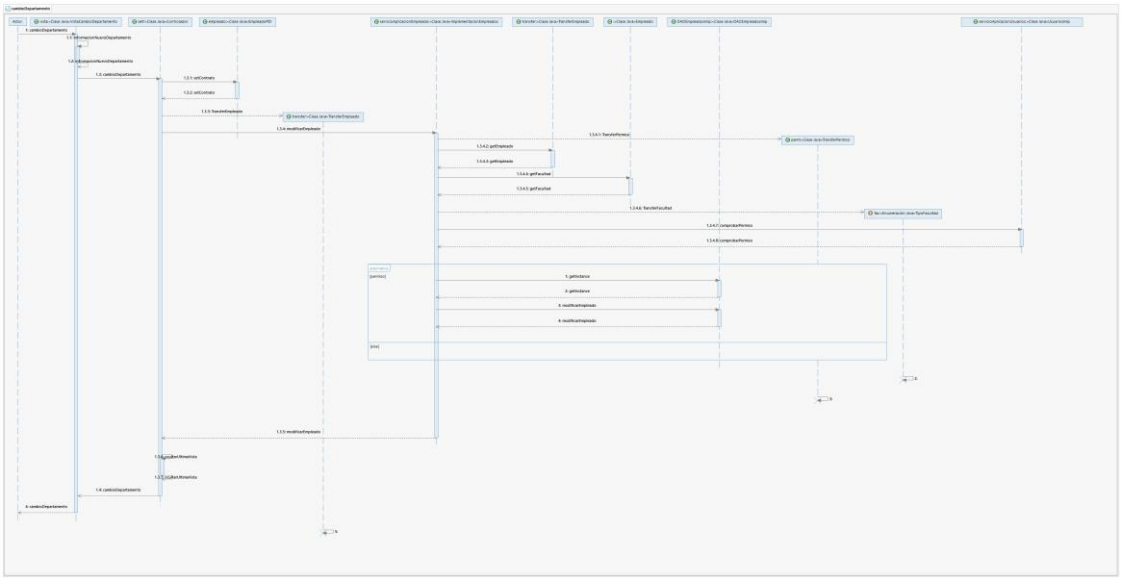


Este diagrama de secuencias describe el caso de uso correspondiente a cambiar el contrato de un empleado de la base de datos de empleado.

La secuencia comienza con un actor o usuario llamando a *contratoCambiado()* que mostrara la *VistaCambioContrato*. Acto seguido la *VistaCambioContrato* pedirá al usuario la nueva información del contrato y llamará al *Controlador*, que modificará el contrato del *Empleado* del que se quiera cambiar el contrato con *setContrato()*. Entonces, creará un nuevo *TransferEmpleado* que utilizará para invocar *modificarEmpleado()* de *ImplementaciónEmpleados*. Este método obtendrá la facultad del empleado primero con *getEmpleado()* y luego con *getFacultad()*, para así poder comprobar los permisos del usuario haciendo uso de la función *comprobarPermisos()* del servicioAplicación de la capa de negocio del subsistema de *Usuario*.

Si el usuario posee los permisos necesarios entonces el servicioAplicación de Empleado llamará al *DAOEmpleados* y modificará el empleado de la base de datos. Por último el *Controlador* ocultará la vista volviendo así al vista de antes de comenzar la secuencia.

Cambiar departamento



El siguiente diagrama de secuencias describe el caso de uso correspondiente a cambiar el departamento de un empleado de PDI de la base de datos de empleado.

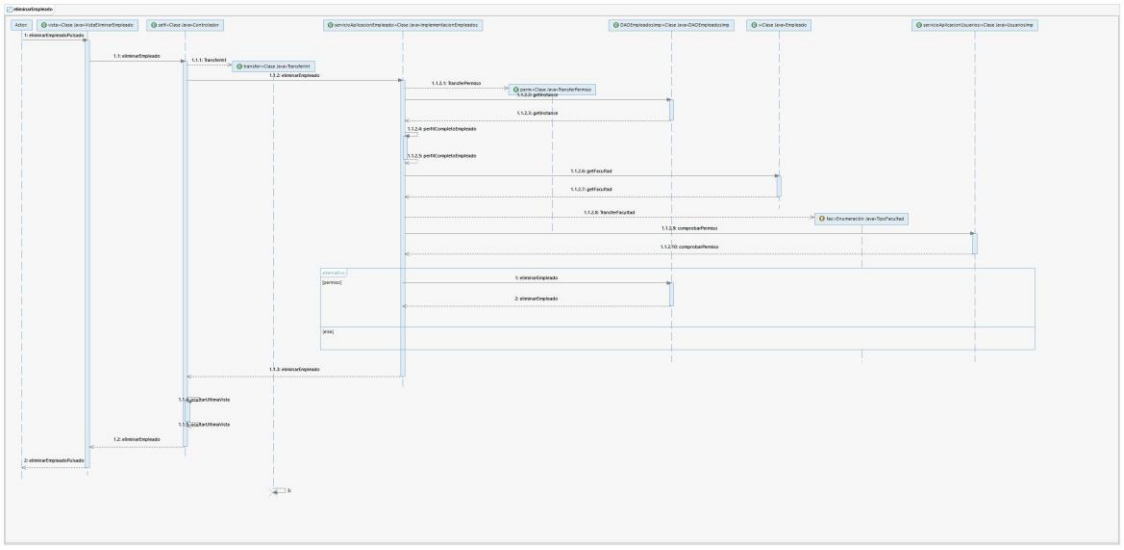
La secuencia comienza con un actor o usuario llamando a *cambioDepartamento()* que mostrara la *VistaCambioDepartamento*. A continuación la *VistaCambioDepartemento* solicitará al usuario el nuevo departamento al cual el empleado va a cambiar y llamará al *Controlador* que modificara el departamento del objeto empleado del que se quiera cambiar con *setDepartamento()*.

Este *Controlador* creará un nuevo *TransferEmpleado* que le pasará a la capa de negocio que mediante su clase *ImplementaciónEmpleados* obtendra la facultad del empleado primero con *getEmpleado()* y luego con *getFacultad()*, para así poder comprobar los permisos del usuario haciendo uso de la función *comprobarPermisos()* del servicioAplicación de la capa de negocio del subsistema de *Usuario*.

Si el usuario posee los permisos necesarios entonces el servicioAplicación de *Empleado* llamará al *DAOEmpleados* y modificará el empleado de la base de datos cambiando así el departamento del empleado.

Por último el *Controlador* ocultará la vista volviendo así al vista de antes de comenzar la secuencia.

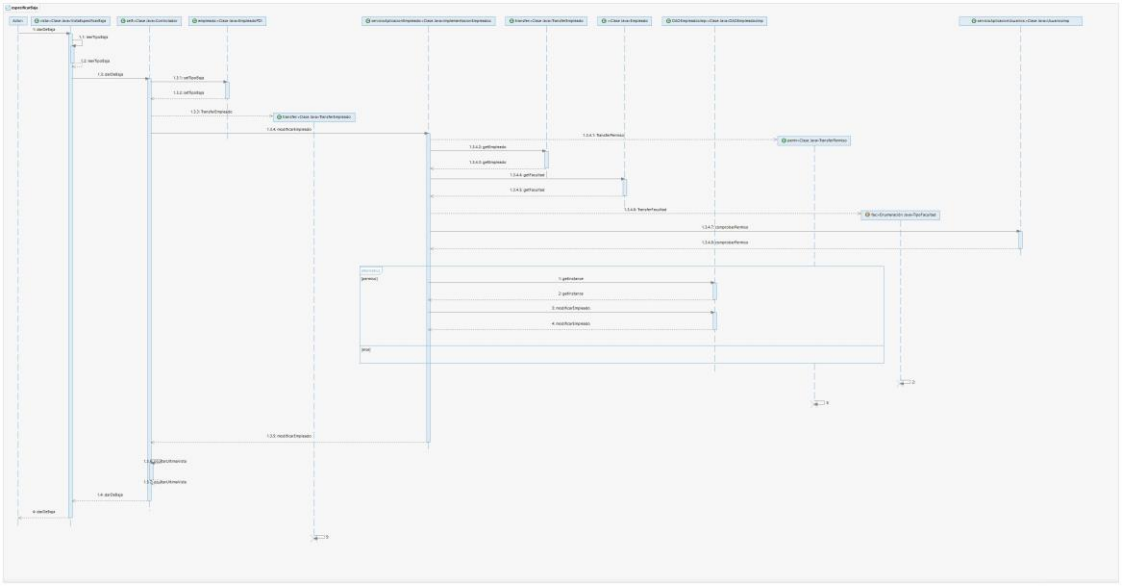
Eliminar empleado



El siguiente diagrama de secuencia describe el caso de uso correspondiente a *Eliminar ficha* que es un subcaso de *Ver ficha empleado*.

La secuencia comienza cuando el actor llama a la acción eliminar empleado y se accede a la vista correspondiente (*VistaEliminarEmpleado*), en la que el actor proporciona la información necesaria para eliminar el empleado que desea. La vista llama entonces al *Controlador* que gestiona sus listeners. Se envía por tanto un entero que identifica al empleado que se desea eliminar al *Controlador* llamando a su método *eliminarEmpleado()*. Este último método crea un *TransferInt* para poder enviarselo al servicio de aplicación *ImplementacionEmpleados* a través del método *eliminarEmpleado()*. En este método comprobamos ciertas cosas como que el perfil del empleado a elimiar esté completo y que el usuario que vaya a eliminarlo tenga permisos para tal acción. Una vez que se comprueba que todo es correcto, llamamos al *DAOEmpleadosImp* con *getInstance()* para enviarle el *TransferInt* que hemos creado anteriormente y que este se lo pase a la base de datos con el fin de eliminar el empleado que corresponda a ese Id.

Especificar baja



El siguiente diagrama de secuencias describe el caso de uso correspondiente a *Especificar una baja* de un empleado de la base de datos de empleado.

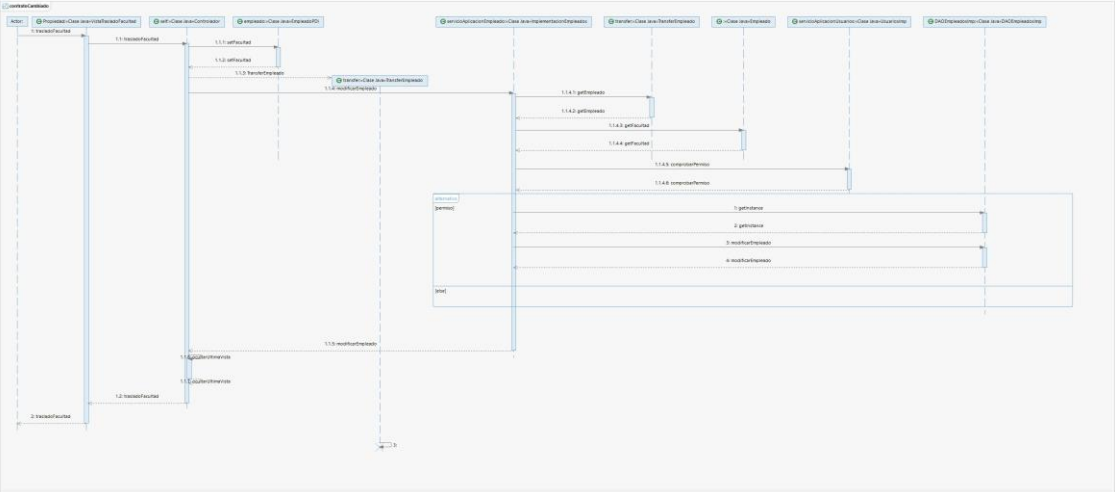
La secuencia comienza con un actor o usuario llamando a *darDeBaja()* que mostrara la *VistaEspecificarBaja*. A continuación la *VistaEspecificarBaja* solicitará al usuario el tipo de baja la cual el empleado haya solicitado y llamará al *Controlador* que modificara el valor del *EnumBaja* del objeto empleado del que se quiera cambiar con *setTipoBaja()*.

Este *Controlador* creará un nuevo *TransferEmpleado* que le pasará a la capa de negocio, la cual, mediante su clase *ImplementaciónEmpleados*, obtendra la facultad del empleado primero con *getEmpleado()* y luego con *getFacultad()*, para así poder comprobar los permisos del usuario haciendo uso de la función *comprobarPermisos* del servicioAplicación de la capa de negocio del subsistema de *Usuario*.

Si el usuario posee los permisos necesarios entonces el servicioAplicación de *Empleado* llamará al *DAOEmpleados* y modificará el empleado de la base de datos cambiando así el tipo de baja del empleado.

Por último el *Controlador* ocultará la vista volviendo así al vista de antes de comenzar la secuencia.

Especificar traslado de facultad



El siguiente diagrama de secuencias describe el caso de uso correspondiente a Especificar un traslado de facultad de un empleado de la base de datos de empleado.

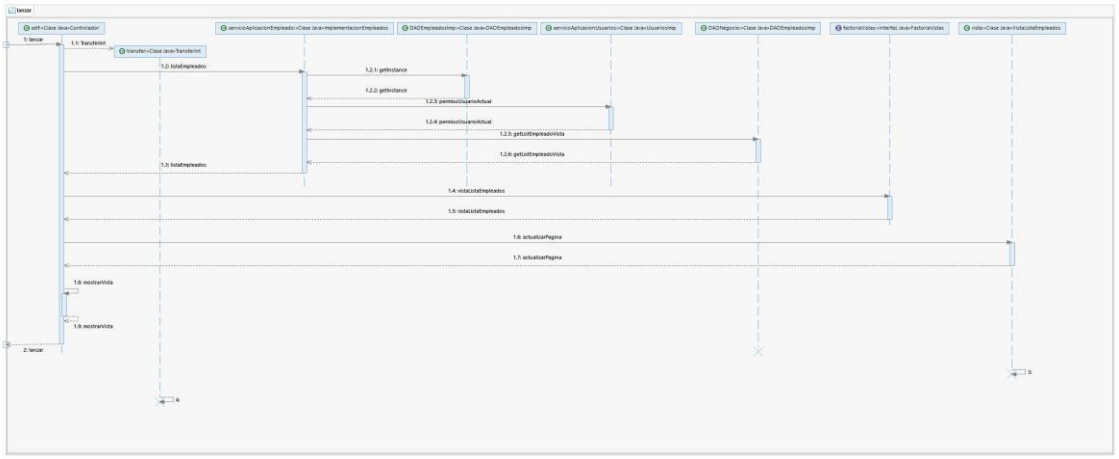
La secuencia comienza con un actor o usuario llamando a *trasladoFacultad()* que mostrara la *VistaTrasladoFacultad*. A continuación la *VistaTrasladoFacultad* solicitará al usuario el cambio de facultad que el empleado haya solicitado o se le haya asignado y llamará al *Controlador* que modificara la facultad del objeto empleado del que se quiera cambiar con *setFacultad()*.

Este Controlador creará un nuevo *TransferEmpleado* que le pasará a la capa de negocio que,mediante su clase *ImplementaciónEmpleados*, obtendra la facultad del empleado primero con *getEmpleado()* y luego con getFacultad, para así poder comprobar los permisos del usuario haciendo uso de la función *comprobarPermisos()* del servicioAplicación de la capa de negocio del subsistema de *Usuario*.

Si el usuario posee los permisos necesarios entonces el servicioAplicación de *Empleado* llamará al *DAOEmpleados* y modificará el empleado de la base de datos cambiando así el tipo de baja del empleado.

Por último el *Controlador* ocultará la vista volviendo así a la vista de antes de comenzar la secuencia.

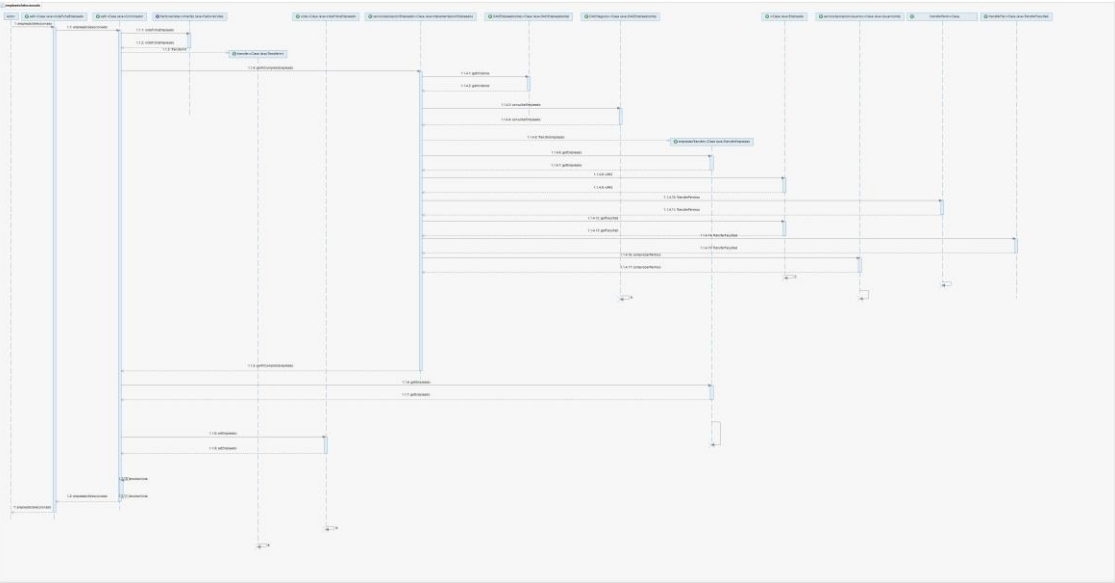
Mostrar lista empleados



El siguiente diagrama de secuencia describe el caso de uso correspondiente a *Ver base de datos*.

La secuencia comienza cuando se llama al método *lanzar()* del *Controlador* que crea un *TransferInt* que hace referencia a la página x en la que se encuentra el usuario que está visualizando la base de datos de los empleados. El *Controlador* llama a *listaEmpleados()*, método del servicio de aplicación *ImplementacionEmpleados*. En este método, primero se obtiene el tipo de permiso que tiene el usuario que quiere ver la base de datos de empleados a través del método *permisoUsuarioActual()*. Luego, llama al *DAOEmpleadosImp* a través de *getInstance()* y le pasa un *TransferInt*, que representa la pégina, y un *TransferPermisos*, que representa el permiso del usuario que desea ver la base de datos, al *DAOEmpleadosImp* para que este busque en la base de datos de empleados y devuelva la lista a mostrar. Finalmente, *lanzar()* crea la vista *VistaListaEmpleados* y muestra la lista creada en la vista.

Ver ficha empleado



El siguiente diagrama de secuencias describe el caso de uso correspondiente *a Ver ficha de empleado*.

La secuencia comienza con un actor o usuario llamando a empleadoSeleccionado que mostrara la VistaFichaEmpleado. A continuación la VistaFichaEmpleado obtendrá del empleado seleccionado (del que solo se tiene nombre, apellidos e id) su id, con el cual creará un *TransferId* para poder buscarlo en la base de datos.

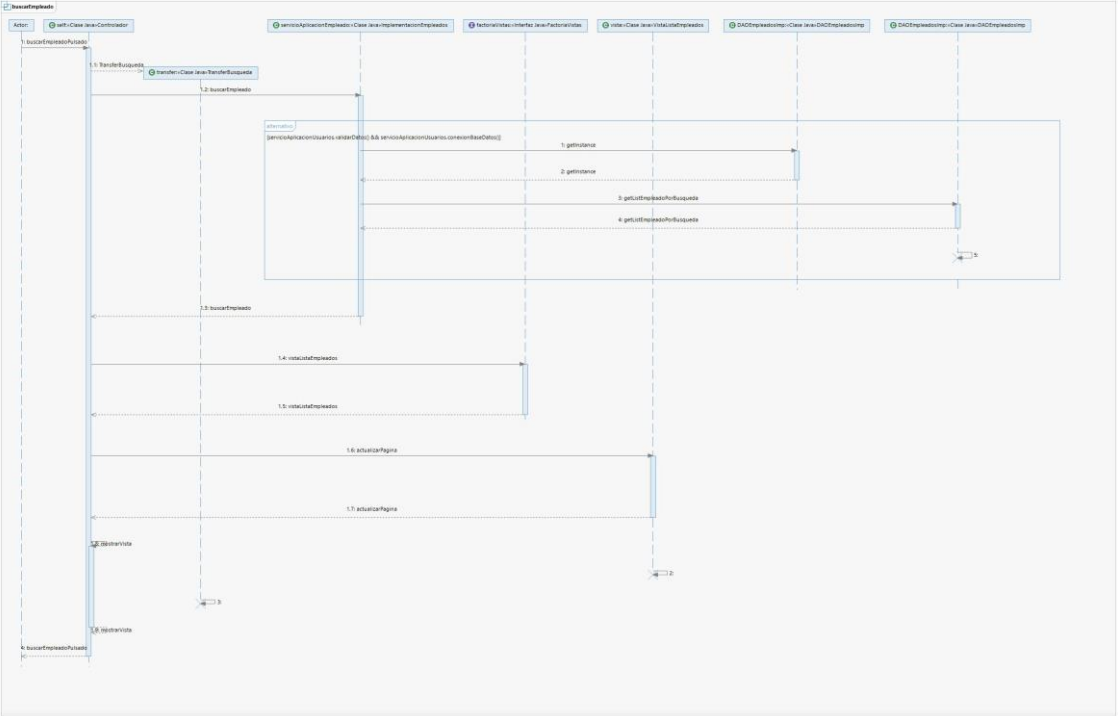
Después el servicio de aplicación de empleados usará este *TransferId* para obtener un *TransferEmpleado* a través de *consultarEmpleado* del *DAOEmpleado*, y de este empleado obtendrá la facultad con *getEmpleado* primero y a posteriori *getFacultad*. Así podrá comprobar los permisos del usuario haciendo uso de la función *comprobarPermisos* del servicioAplicación de la capa de negocio del subsistema de Usuario.

Si el usuario posee los permisos necesarios entonces el *DAOEmpleado* devolverá el valor correspondiente del empleado y sino el *DAOEmpleado* devolverá un valor vacío, por lo que si los posee el controlador lo obtendrá con getEmpleado y los mostrara en la VistaFichaEmpleado y de no ser así al no tener valor no mostrará nada excepto un mensaje de error.

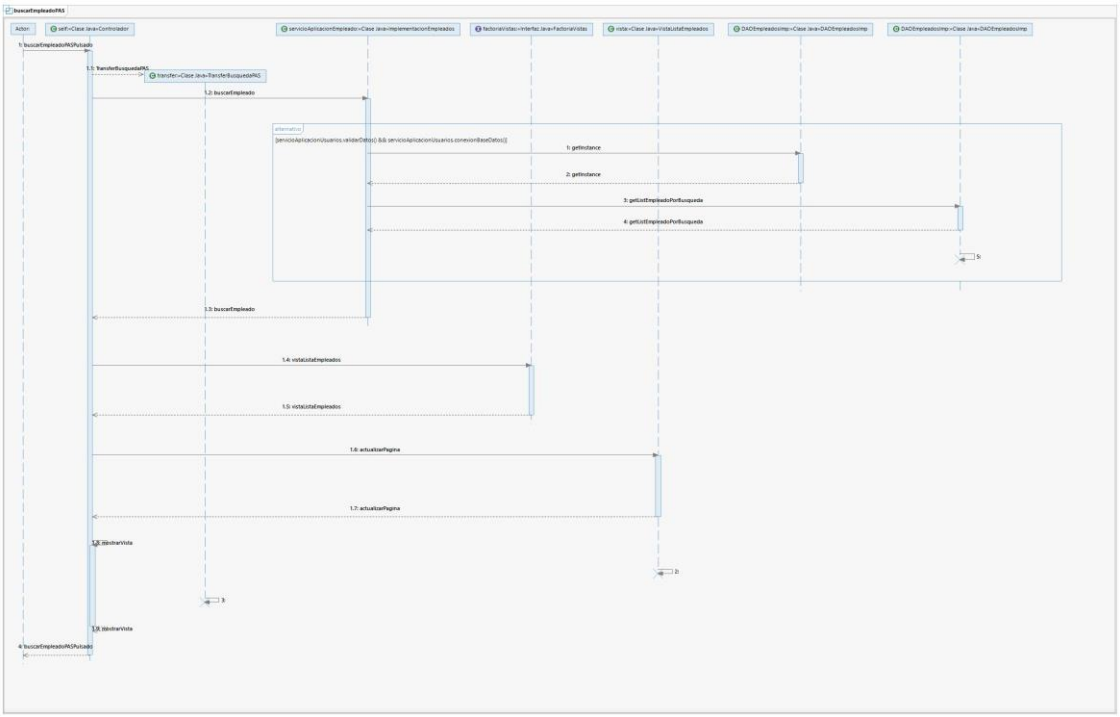
Por último el Controlador ocultará la vista volviendo así a la vista de antes de comenzar la secuencia.

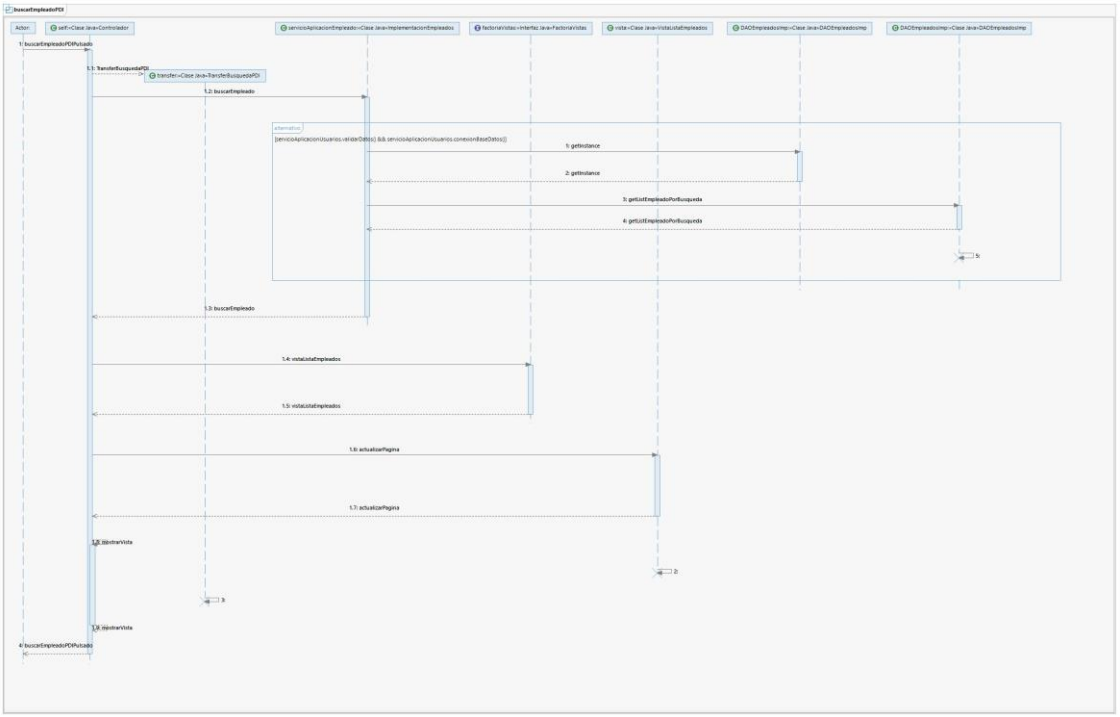
Buscar empleado

En este caso, tenemos tres diagramas de secuencias que corresponden a los casos Buscar empleado, Buscar empleado PAS y Buscar empleado PDI. Son muy similares, de forma que haremos una sola descripción para los tres.



La secuencia comienza con la llamada del actor a la acción buscar empleado. Se llama al método *buscarEmpleadoPulsado()* de *Controlador* al que se le pasan como parámetros los elementos de búsqueda (nombre, apellidos y facultad). Creamos un objeto de transferencia *TransferBusqueda*, *TransferBusquedaPDI* o *TransferBusquedaPAS* según el tipo de filtro que haya elegido el usuario y lo pasamos como parámetro en el método *buscarEmpleado()* del servicio de aplicación *ImplementaciónEmpleados*. Este método se encarga de, una vez validados los datos introducidos y comprobada la conexión con la base de datos, acceder a la instancia de *DAOEmpleadosImp* gracias a *getInstance()* y pedirle que nos devuelva una lista con empleados filtrada según los parámetros introducidos en el método. Una vez que tenemos la lista, creamos una *VistaListaEmpleados* y llamamos a *actualizarPagina()* para que la muestre.



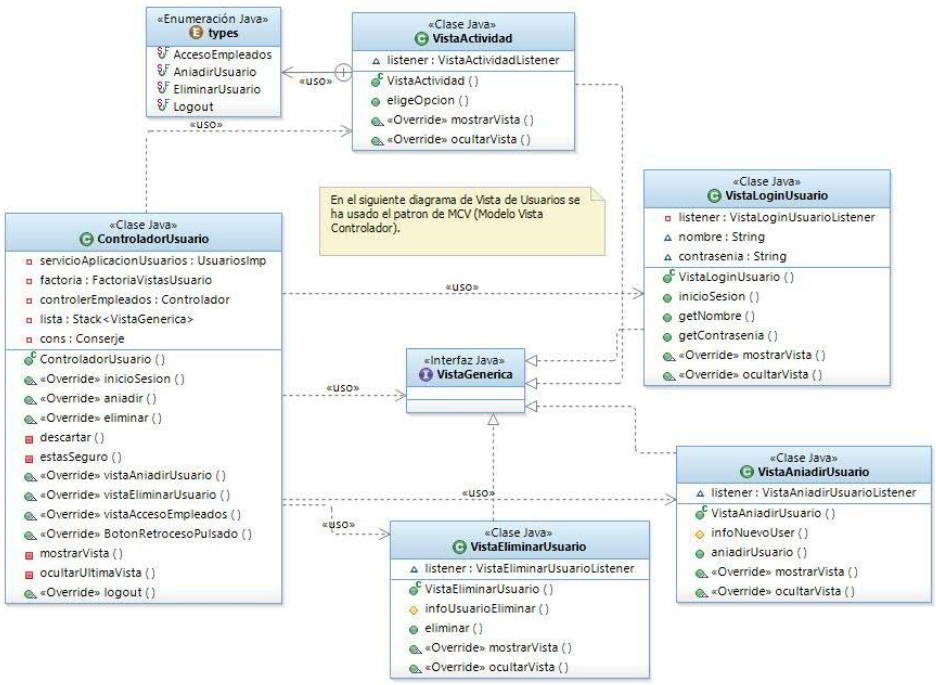


Subsistema Usuario

La división a gran escala es muy similar a la del subsistema de empleados. De nuevo, dividiremos el sistema en tres capas, utilizando la misma arquitectura multicapa. Por ello, tendremos una capa de presentación que será la encargará de la visualización de la aplicación, la capa de negocio cuya tarea es la gestión interna de la aplicación y de su correspondiente lógica, y la capa de integración que gestionará la base de datos.

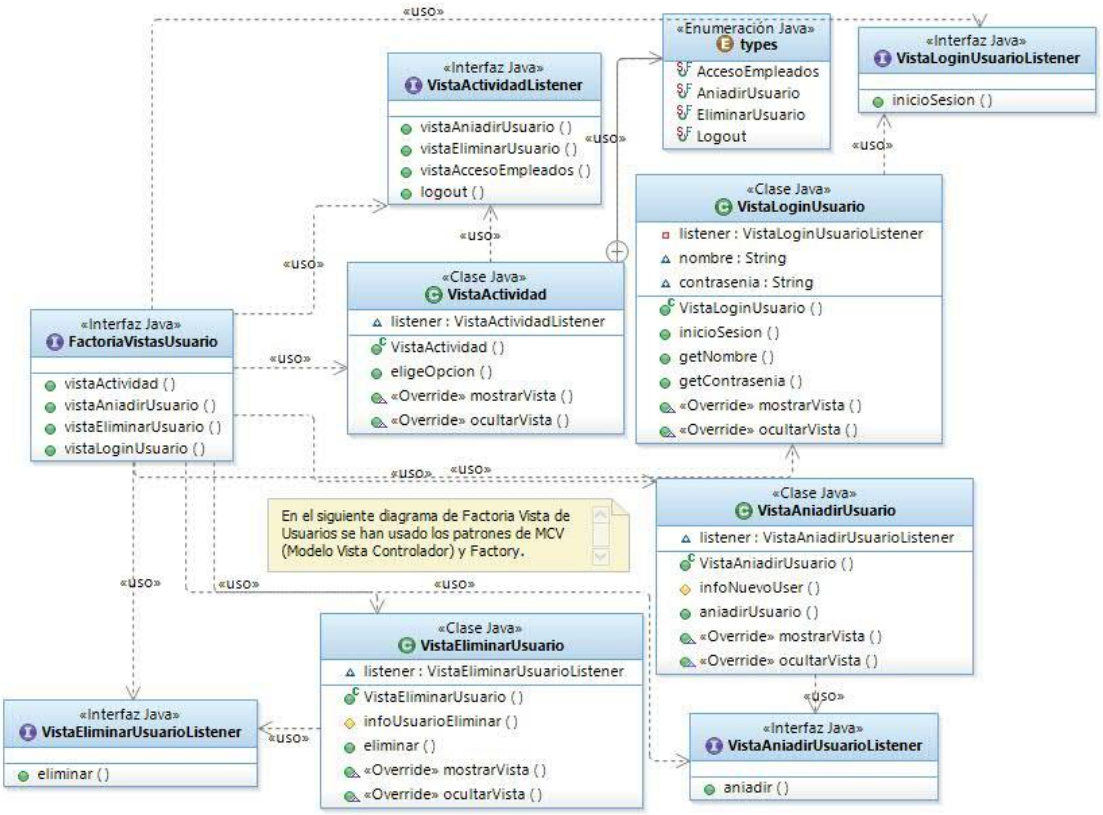
Capa de presentación

Para visualizar nuestro subsistema, utilizaremos un patrón Modelo-Vista-Controlador. Para ello utilizamos la interfaz ya mencionada antes, *VistaGenérica*, que implementarán nuestras vistas. Cada vista tiene una instancia de un *listener* que nos avisará cuando un usuario interaccione con nuestro sistema. Esta función, como ya hemos mencionado antes, cumple un propósito similar a la del Patrón Observador.



Por otro lado, la función del controlador consiste en manejar estas vistas, ocultandolas y mostrandolas cuando sea necesario, y respondiendo a las acciones del usuario sobre el

sistema, ya que implementa todas las interfaces *listener* mencionadas antes. El controlador también tiene una instancia de una factoría, cuyo cometido es crear las vistas que se le dicen. Es un claro ejemplo del Patrón Factoría.



Así, nuestras vistas serán abstractas y nuestra factoría nos proporcionará una vista cuando sea pedida. Es decir, nuestra factoría tiene un método correspondiente a cada tipo de vista. A continuación, describiremos la funcionalidad de cada una de las vistas ya mencionadas:

- *VistaAniadirUsuario*: permite añadir un nuevo usuario, mostrando un formulario a rellenar. Cabe destacar que dicho formulario es el referido en el caso de uso “Añadir Usuario” del documento “Especificación de requisitos”.
- *VistaEliminarUsuario*: permite eliminar un usuario del sistema.
- *VistaLoginUsuario*: permite iniciar sesión en la aplicación.

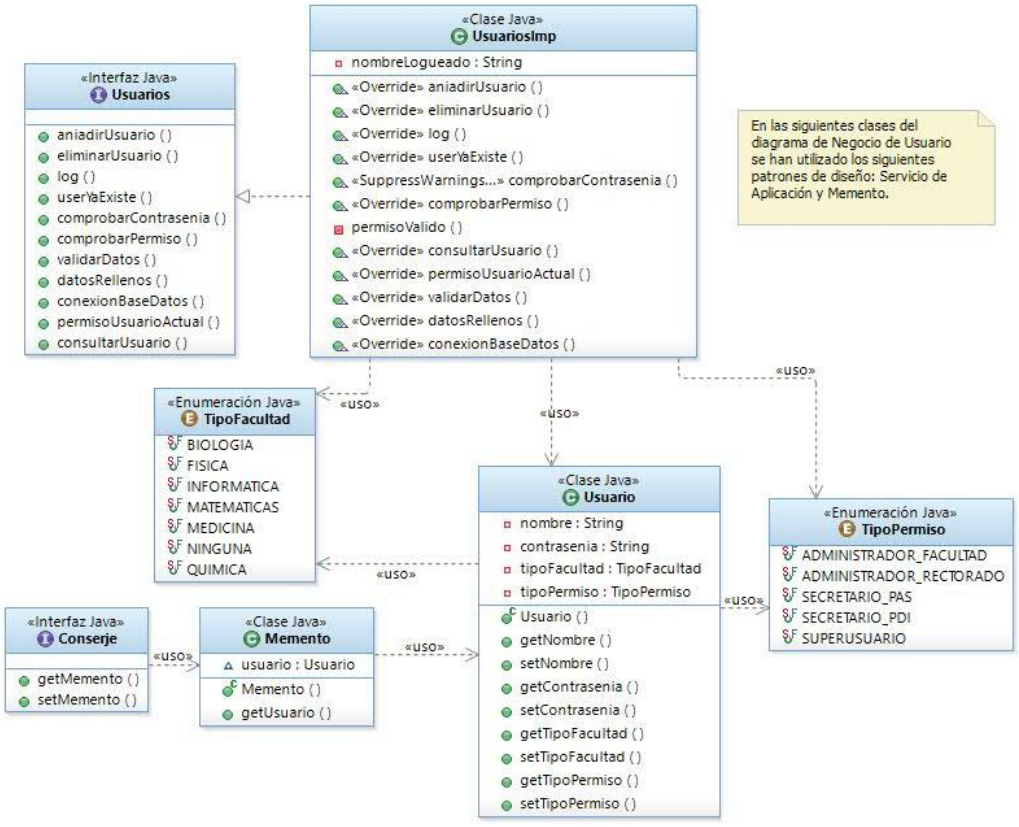
- *VistaActividad*: Especie de menú que proporciona una serie de opciones que el usuario puede elegir como: ver base de datos, eliminar usuario o añadir usuario.

Los nombres de las interfaces de los *listener* tienen la siguiente forma: "<nombre de la vista> *Listener*".

Capa de negocio

En esta capa, al igual que en el *subsistema empleados*, desarrollamos la *lógica* de nuestra aplicación. Utilizaremos el patrón de *Servicio de Aplicación*, que consiste en tener un objeto cuya función principal sea la de procesar las peticiones de la vista y comunicar a la capa de integración.

Tenemos una subdivisión en esta capa*: Lógica*, *reglas* y *objetos transferencia* (*Transfers*).



Lógica

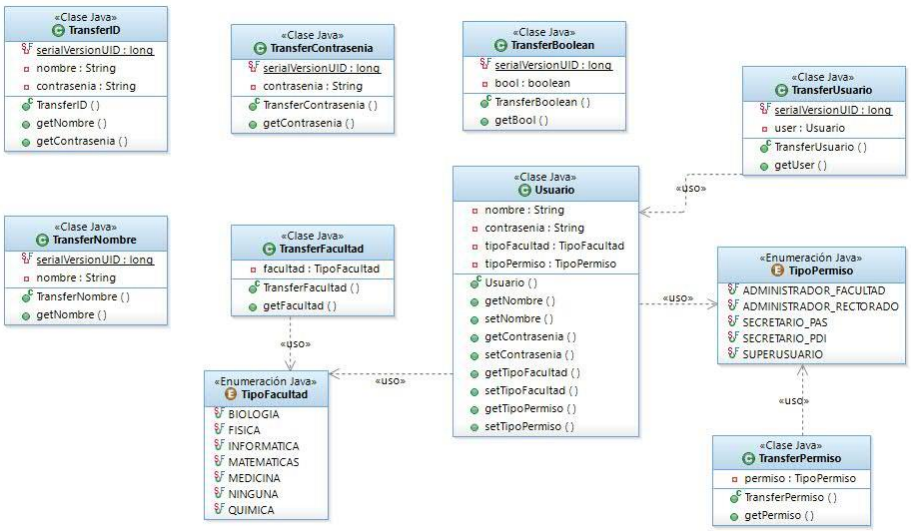
Es la parte que implemente el patrón de *Servicio de Aplicación*. Está compuesta por una interfaz "*Usuarios*" y una clase que la implementa, "*UsuariosImp*". Esta clase es usada para aislar la implementación de los métodos y tiene como función principal la de interactuar con la capa de integración y con la lógica.

Reglas

Constituye el modelo de nuestra aplicación y contiene las siguientes clases:

- *TipoFacultad* : (Matemáticas, física, química, biología, informática, medicina, ninguna).
- *TipoPermiso*: define si el permiso del usuario es superuser, Admin-Rectorado, Admin-Facultad, Secretaría-PDI o Secretaría-Pas
- *Usuario*: definirá a un usuario.
- *Conseje*: forma parte del Patrón Memento junto a la clase "Memento", es el encargado de guardar y devolver mementos para descartar eliminaciones de usuarios en la base de datos.
- *Memento*: como ya se ha puntualizado, forma parte del Patrón Memento y posee básicamente un objeto Usuario.

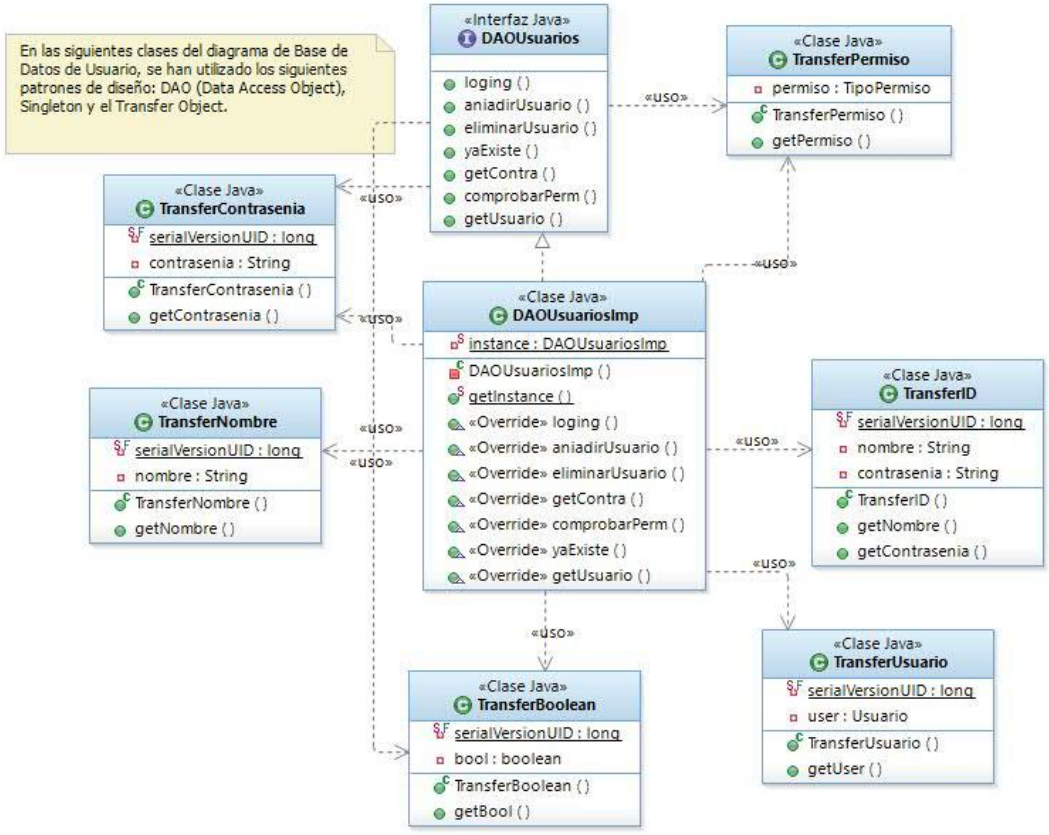
Objetos transferencia



Estos objetos son usados por todo el módulo para intercambiar información entre las diferentes capas. Existen los siguientes tipos: *TransferBoolean*, *TransferContrasenia*, *TransferFacultad*, *TransferID*, *TransferNombre, TransferPermiso* y *TransferUsuario*. Su función principal es encapsular la información de un dato o conjunto de datos.

La diferenciación en varios tipos de objetos transferencia viene dada por la necesidad de los distintos métodos de recibir objetos concretos en lugar de un usuario completo.

Capa de integración



En esta capa se realizará el intercambio de datos con la unidad de almacenamiento de información. Dicha unidad puede ser de cualquier tipo.

Hemos utilizado el Patrón de diseño DAO (“Data Access Object”), que nos permite transferir datos específicos sin la necesidad de saber los detalles del almacenamiento que haya detrás.

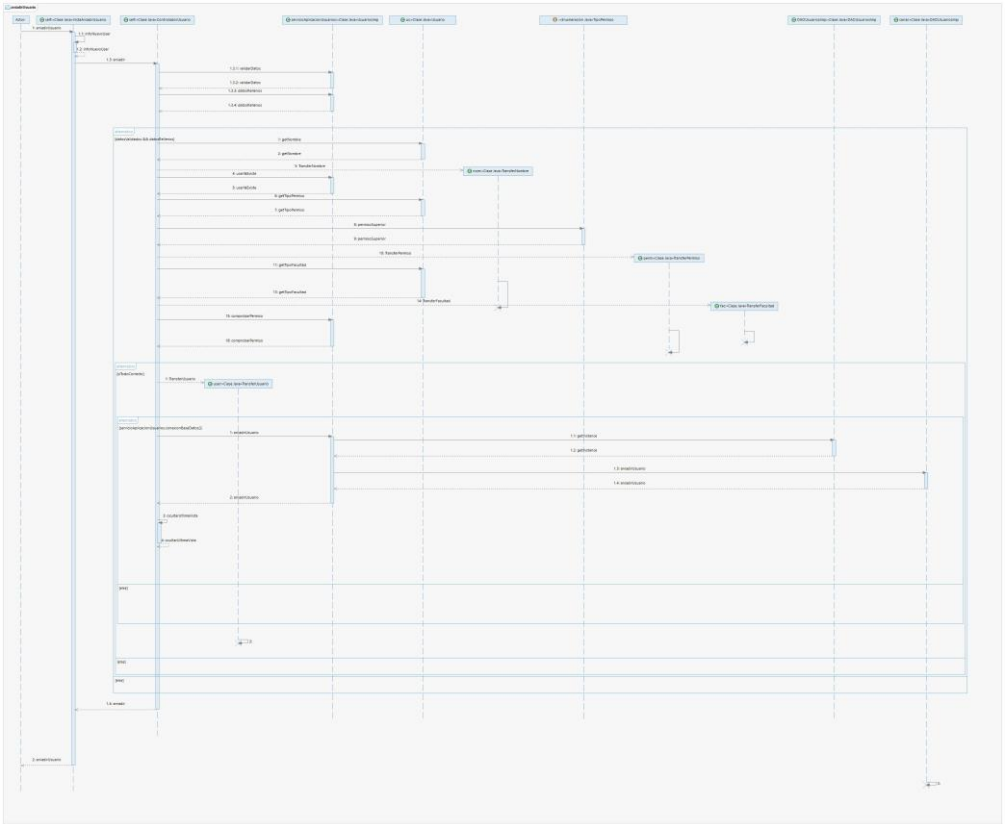
Como consecuencia, la utilización de este Patrón aportará seguridad en la capa de negocio ante cambios en la organización o en la estructura de los datos en la base de datos.

Así, la capa de integración se compone básicamente de una interfaz “DAOUsuarios”, que proporciona métodos específicos donde devolverá objetos de transferencia. Esta interfaz será implementada por la clase “DAOUsuariosImp”. Para evitar que circulen un número grande de instancias de este tipo por nuestra aplicación y evitar conexiones simultáneas de varios objetos distintos a la base de datos, hemos implementado el Patrón Singleton. De esta forma, privatizamos el constructor y sólo se crea una única instancia de DAOUsuariosImp, que es compartida por todos los objetos que la necesitan.

Subsistema usuario: diagramas de secuencia

En esta sección, haremos un recorrido por los distintos casos de uso que forman parte de este subsistema, describiéndolas mediante diagramas de secuencia.

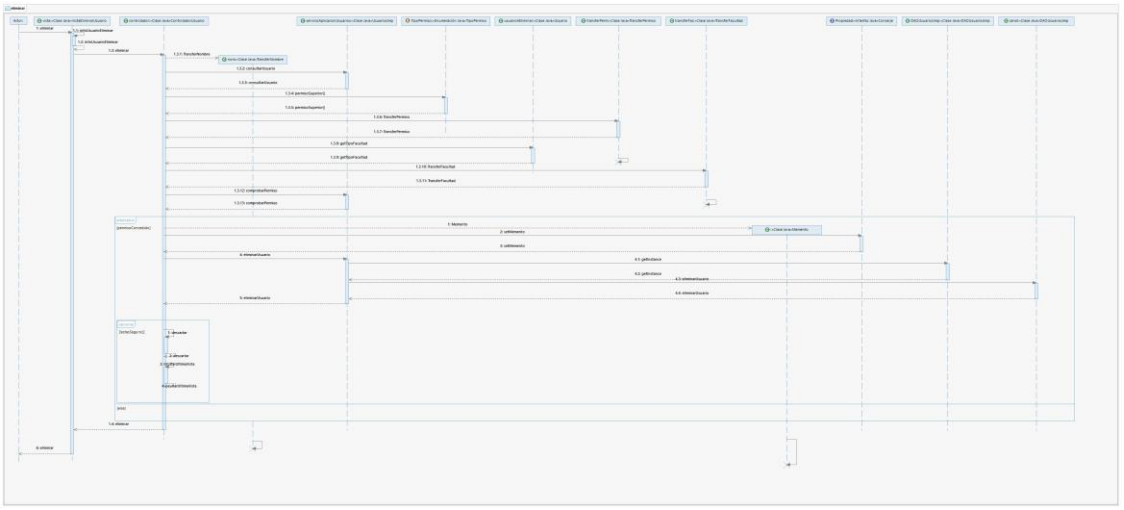
Añadir usuario



El diagrama desarrolla los casos de uso *Añadir usuario*, con sus subcasos *Cuenta de administración de rectorado*, *Cuenta de administración de facultad* y *Cuenta de secretaría* (a su vez con los subcasos *Cuenta de secretaría PAS* y *Cuenta de secretaría PDI*).

La secuencia comienza con la llamada del actor a la acción crear un nuevo usuario. Se añade la vista correspondiente (*VistaAniadirUsuario*) en la que el actor elige el tipo de usuario a crear (de rectorado, de facultad, secretario, etc.) y rellena los campos correspondientes. Se crea un usuario con los datos proporcionados y se envía a *ControladorUsuario* llamando a su método *aniadir*(). Este llama a *UsuariosImp* que hace distintas comprobaciones al Usuario recibido. Una vez que todo es correcto (los datos introducidos son correctos y completos, el usuario no existe ya y el actor tiene permiso para crearlo), creamos un objeto de transferencia *TransferUsuario* y llamamos al método *aniadirUsuario*() de *UsuariosImp* con el argumento creado *TransferUsuario*. Este método a su vez llama a *aniadirUsuario* de *DAOUsuariosImp* que lo añade a la base de datos.

Eliminar usuario

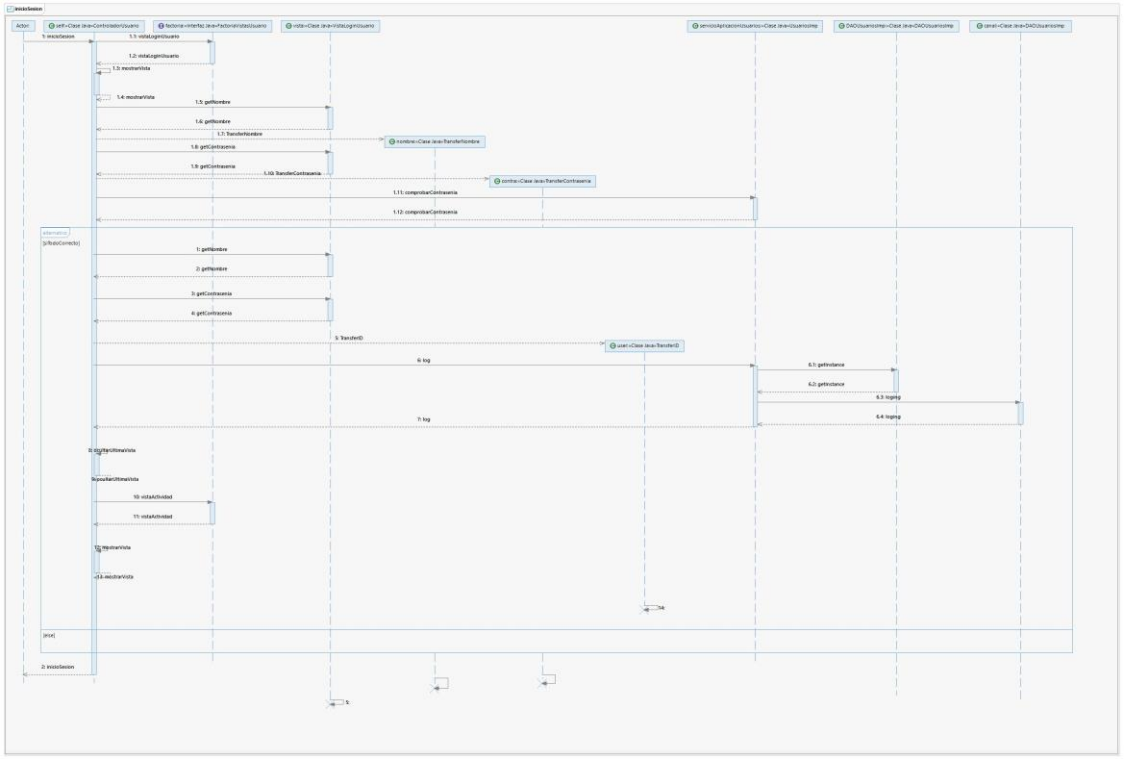


El presente diagrama desarrolla el caso de uso *Eliminar Usuario*.

La secuencia comienza con la llamada del actor a la acción eliminar un usuario. Se añade la vista correspondiente (*VistaEliminarUsuario*) en la que el actor elige el usuario a eliminar. Se envía un *String* con el nombre del usuario a *ControladorUsuario* llamando a su método *eliminar*(). Este llama a *UsuariosImp* que hace distintas comprobaciones al nombre recibido. Una vez que todo es correcto (el usuario existe y el actor tiene permiso para eliminarlo), creamos

un objeto de transferencia *TransferNombre* con el nombre de usuario y lo pasamos como parámetro del método *eliminarUsuario*() de *UsuariosImp*. Este a su vez llama a eliminarUsuario de *DAOUsuariosImp* que lo elimina de la base de datos.

Log in



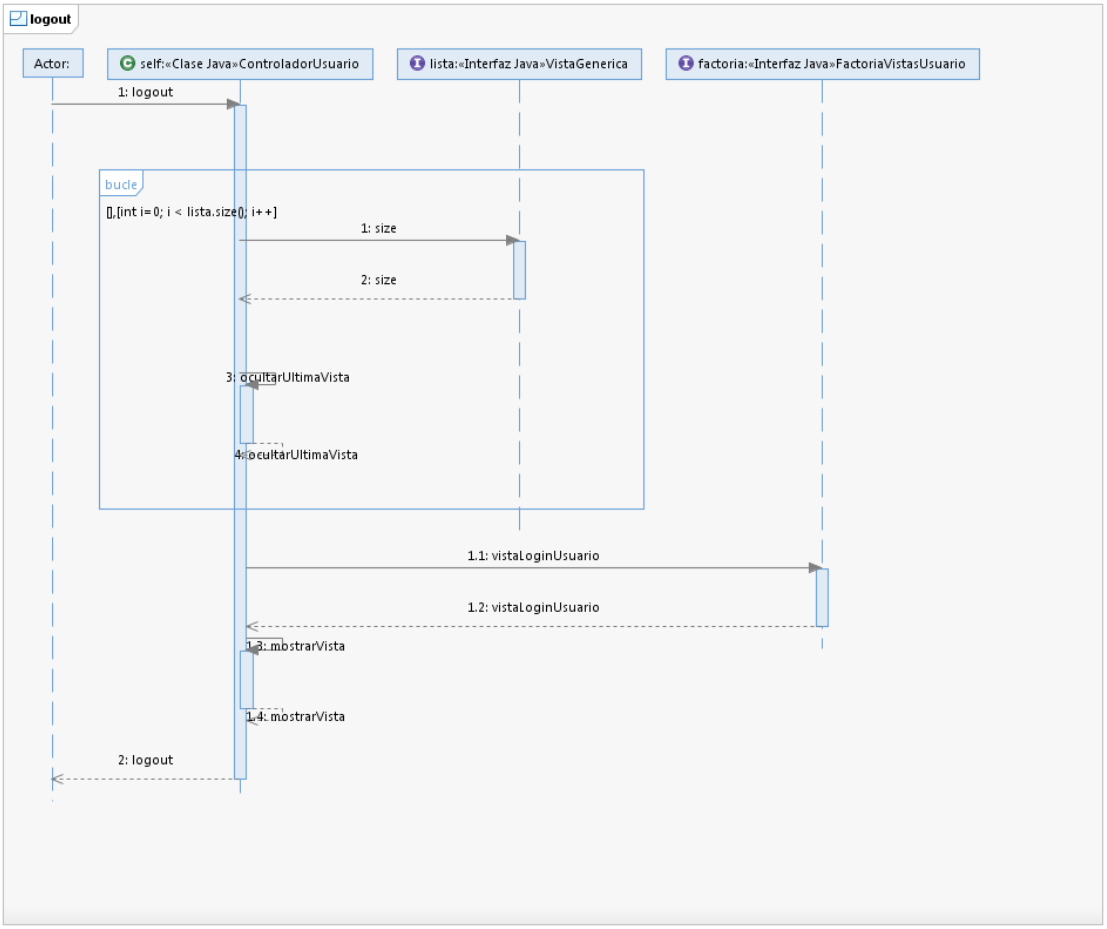
El presente diagrama desarrolla el caso de uso Log in.

La secuencia comienza con la llamada del actor a la acción iniciar sesión o login. Se llama al método *inicioSesion*() de *ControladorUsuario*. Este llama en primer lugar a *FactoriaVistasUsuario* que crea la vista donde el actor introducirá sus credenciales.

Se obtienen dichos credenciales de la vista creada y a continuación se llama a *UsuariosImp* que comprueba la contraseña. Si esta es correcta creamos un objeto de transferencia *TransferId* que pasamos como argumento del método *log*() de *UsuariosImp* que llama a *loging*() de *DAOUsuariosImp* que realiza el login del usuario.

Por último ocultamos la vista creada para la introducción de credenciales y creamos la vista principal del usuario logueado.

Logout



El presente diagrama desarrolla el caso de uso Log out.

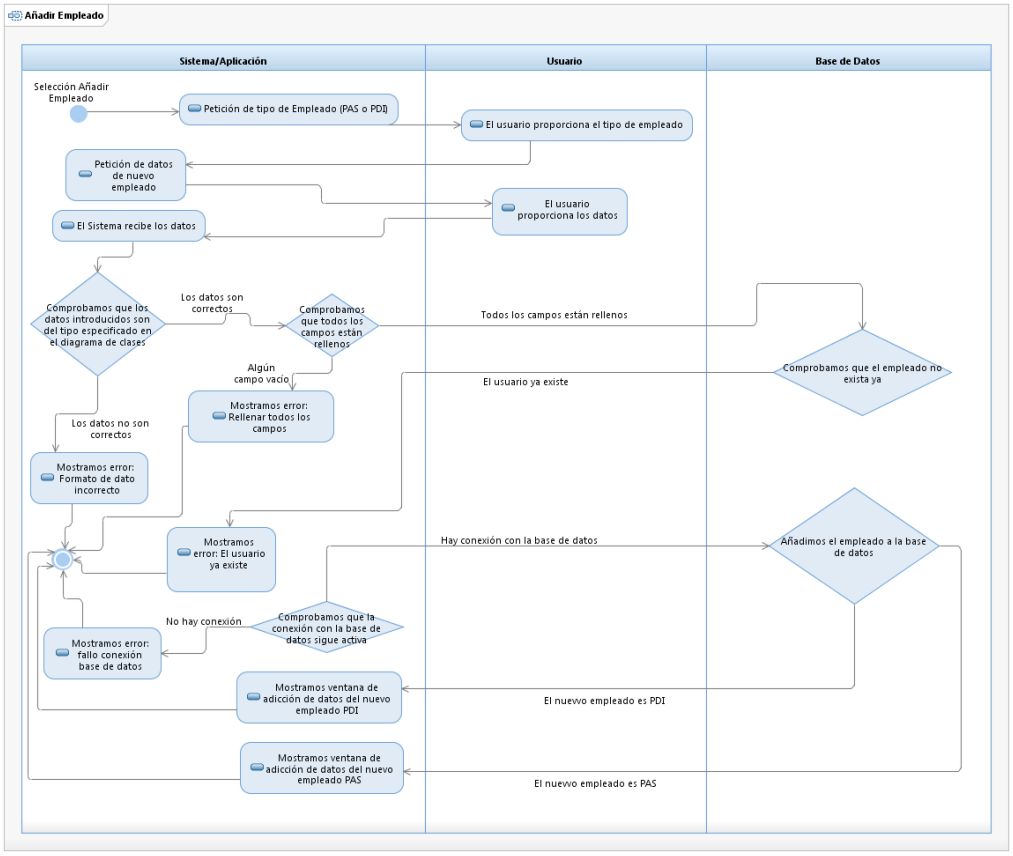
La secuencia comienza con la llamada del actor a la acción cerrar sesión o logout. Se llama al método *logout*() de *ControladorUsuario*. Este simplemente elimina todas las vistas creadas hasta el momento y crea una nueva ventana de login para un futuro usuario.

Diagramas de actividades

Por último en este apartado vamos a añadir los siguientes 5 diagramas de actividades, localizados tanto en el subsistema usuario como en el de empleado.

Añadir empleado

En el diagrama de actividades siguiente corresponde con el caso de uso de *Añadir empleado*.

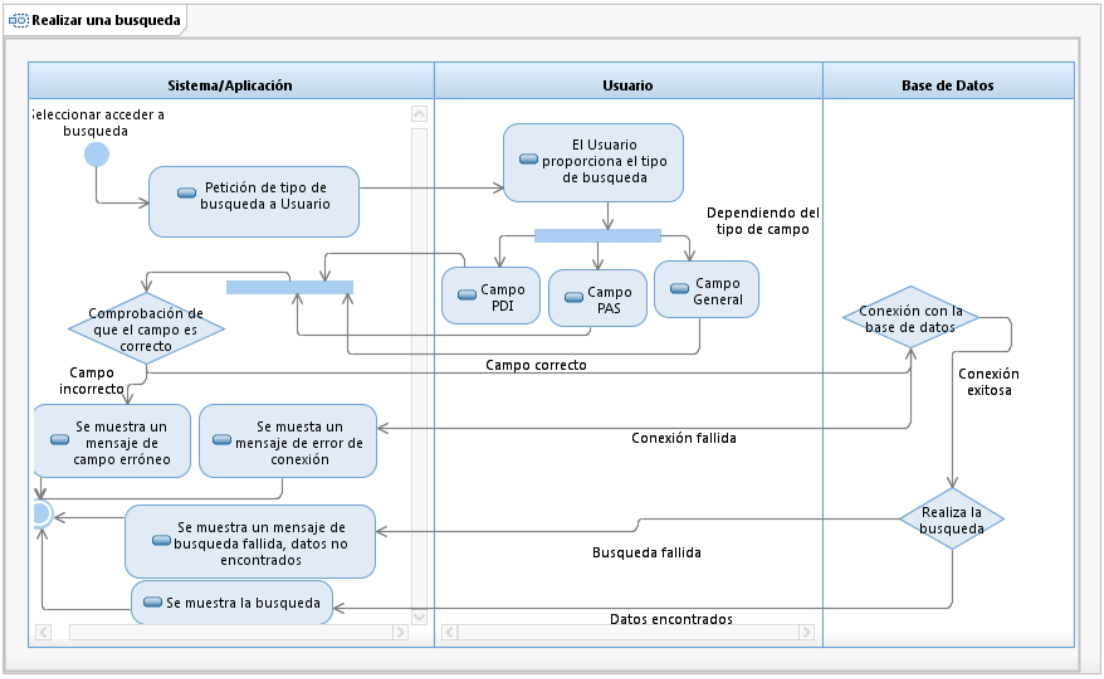


El sistema acto seguido de que un usuario haya seleccionado la opción de añadir un empleado el sistema le pide al usuario, que introduzca si desea que el empleado sea de PAS o de PDI. Entonces el usuario escribe dicho dato y el sistema lo recibe y le pide los datos correspondientes a ese nuevo empleado. Inmediatamente el sistema recibe esos datos y comprueba si son correctos o no.

Si no lo son mostrará un mensaje de error y finalizará esa tarea, pero si los datos si son correctos continua y comprueba que todos los campos esten rellenos, si no lo están mensaje de error y finalización, de estarlo continuará con otra comprobación mas. En esta, el sistema comprueba si el usuario existe o no,de no ser así mensaje de error y finalización, pero de existir se comprueba si hay conexión de la base de datos y finalmente si se consigue conectar con la base de datos, esta misma creará el nuevo empleado.

Realizar búsqueda

En el diagrama de actividades siguiente corresponde con el caso de uso de *Realizar una búsqueda*.



El sistema, cuando el usuario haya seleccionado la opción de realizar una búsqueda, le pide al usuario que introduzca el parametro de busqueda porque deseara buscar. Entonces el usuario escribe dicho dato y el sistema lo recibe, comprobando si el campo que ha solicitado es correcto. Si no, lanza un mensaje de error y acaba la ejecución de esta tarea, pero si es correcto se comprueba si hay conexión de la base de datos. Si no se consigue establecer esta conexión, se mostrará un mensaje de error y finaliza la ejecución. De ser la conexión correcta se comienza a realizar la busqueda y en este caso si se encuentra algún dato, la busqueda habrá tenido éxito; sino, habrá sido fallida y mostrará un mensaje de error de busqueda fallida.

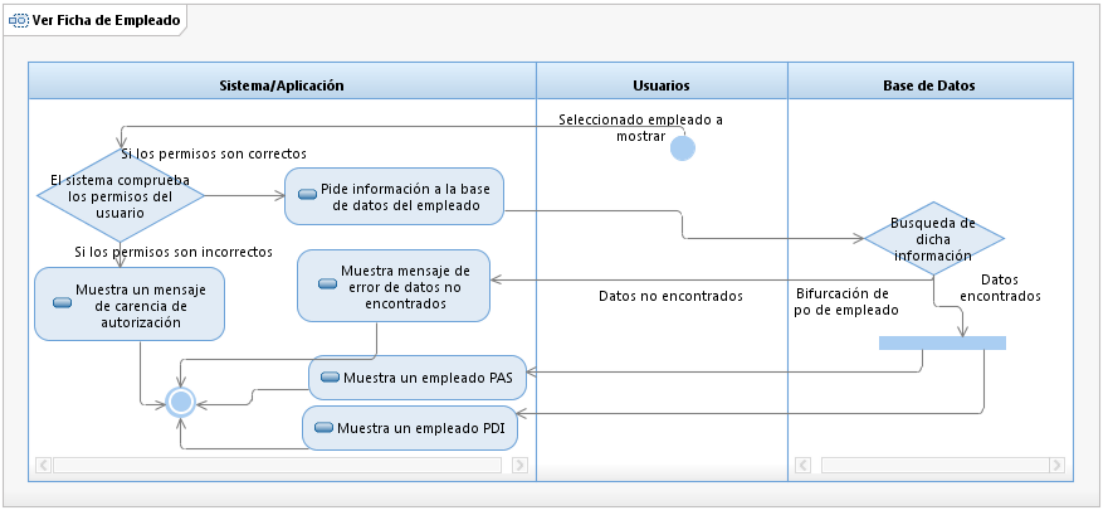
Ver ficha de empleado

Este diagrama de actividades corresponde con el caso de uso de *Ver ficha de empleado*.

El sistema, cuando el usuario haya seleccionado la opción de ver la ficha de un empleado, comprueba si los permisos del usuario son correctos. De no serlo, se producirá un mensaje de

error y se saldrá de la ejecución de la tarea pero si son correctos el sistema pedirá la información del empleado a la base de datos.

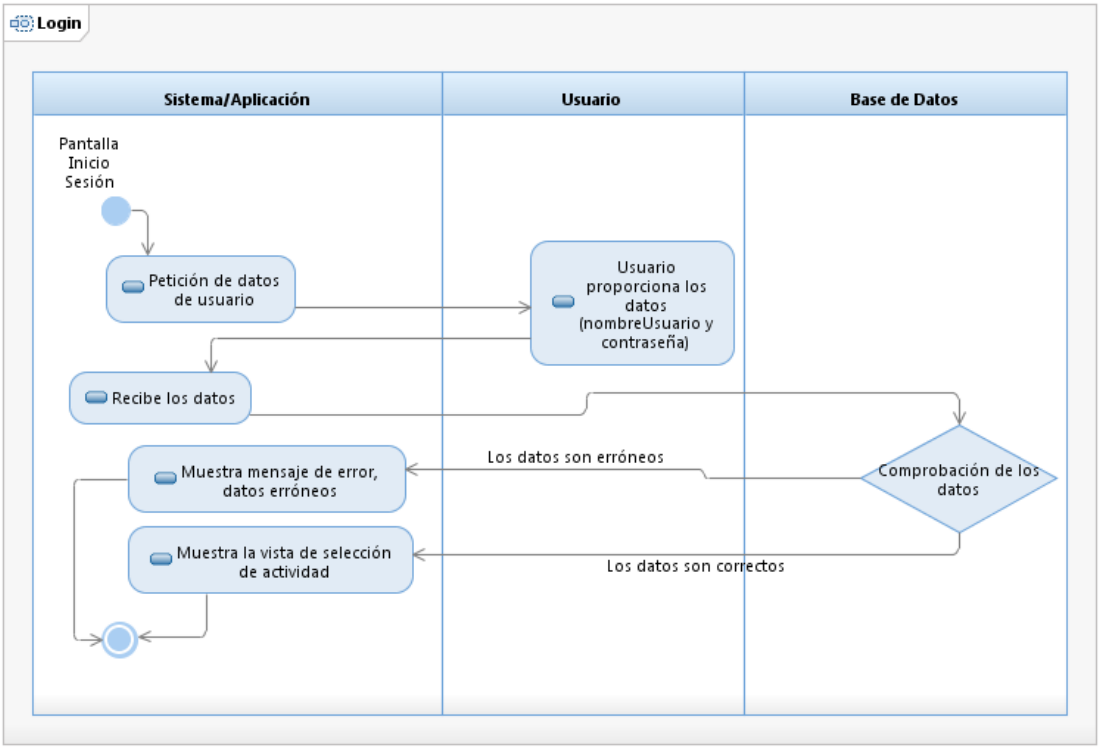
Si la base de datos no encuentra el empleado, fallará y mostrará mensaje de error en caso contrario el sistema mostrará o un empleado PAS o un empleado PDI.



Login

El diagrama de actividades siguiente corresponde con el caso de uso de *Login*.

El sistema muestra la pantalla de inicio de sesión y le pide al usuario el nombre y la contraseña. Tras leer esos datos, comprueba si son correctos con la base de datos. Si son correctos muestra la pantalla de selección de actividad, pero de lo contrario muestra un mensaje de error y vuelve a la pantalla de inicio de sesión.

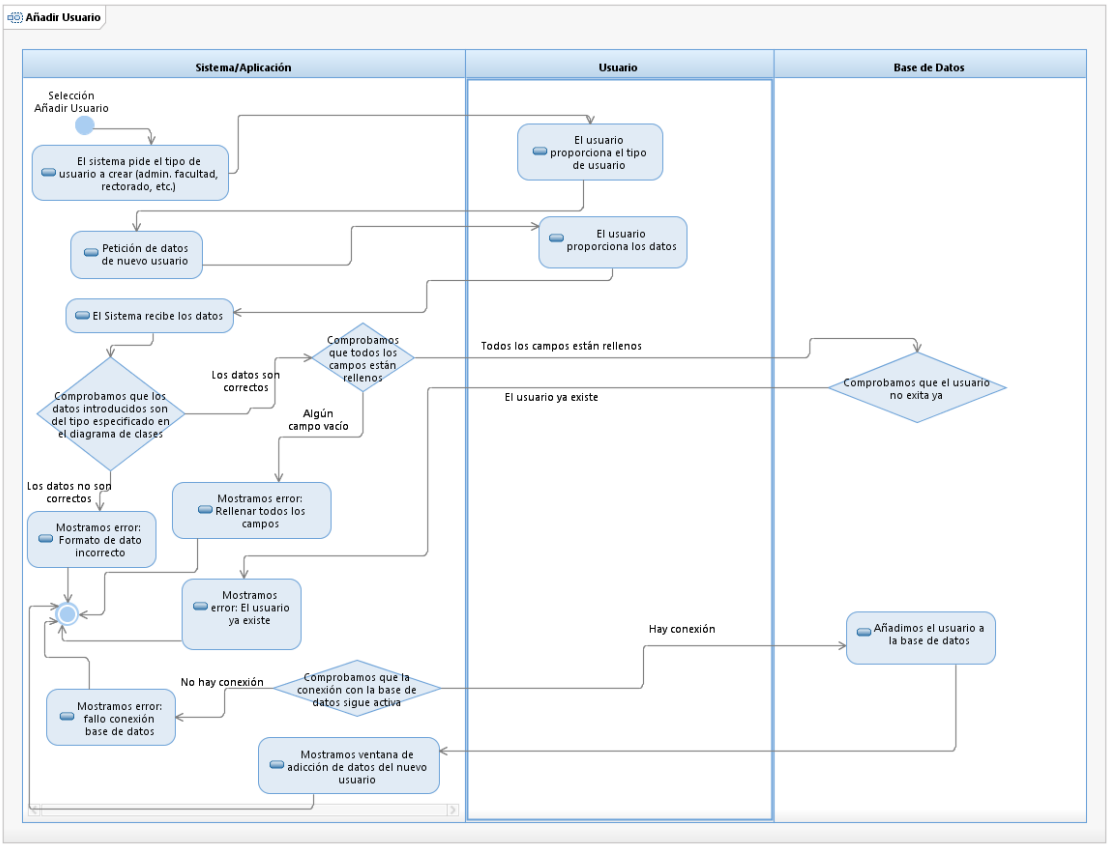


Añadir usuario

El diagrama de actividades siguiente corresponde con el caso de uso de *Añadir usuario*.

Cuando el usuario selecciona la opción de añadir otro usuario, el sistema le pide al usuario que introduzca que tipo de usuario desea crear. Entonces el usuario escribe dicho dato y el sistema pide a la base de datos los datos correspondientes a ese nuevo empleado. Inmediatamente el sistema recibe esos datos y comprueba si son correctos o no.

El sistema comprueba si esos datos son correctos o no, si todos los campos están rellenos y si el usuario existe o no. Si cualquiera de estas comprobaciones falla, el sistema mostrará un mensaje de error y finalizará esa tarea. Finalmente, si todas las comprobaciones son correctas, el sistema añadirá el nuevo usuario.


Añadir Usuario
Sistema/Aplicación
Usuario
Base de Datos
Selección Añadir Usuario
El sistema pide el tipo de usuario a crear (admin, facultad, rectorado, etc.)
El usuario proporciona el tipo de usuario
Petición de datos de nuevo usuario
El usuario proporciona los datos
El Sistema recibe los datos
Comprobamos que todos los campos están rellenos
Todos los campos están rellenos
Comprobamos que los datos introducidos son del tipo especificado en el diagrama de clases
Los datos son correctos
El usuario ya existe
Comprobamos que el usuario no exita ya
Algún campo vacío
Los datos no son correctos
Mostramos error: Rellenar todos los campos
Mostramos error: Formato de dato incorrecto
Mostramos error: El usuario ya existe
Hay conexión
Añadimos el usuario a la base de datos
No hay conexión
Comprobamos que la conexión con la base de datos sigue activa
Mostramos error: fallo conexión base de datos
Mostramos ventana de adicción de datos del nuevo usuario

Pruebas

En este apartado trataremos 5 pruebas para 5 casos de uso distintos: “Login”, “Añadir Empleado”, “Añadir Usuario”, “Ver Ficha de Empleado” y “Realizar una búqueda”.

Para cada caso de prueba seguiremos este esquema:

1. Introducción
2. Validación de datos
   - 2.1. Descripción
   - 2.2. Condiciones de ejecución
   - 2.3. Entrada
   - 2.4. Validaciones
   - 2.5. Resultado esperado

Caso de Prueba: Login

1. Introducción

Este artefacto cubre el conjunto de pruebas realizadas sobre el Caso de Uso Login.

2. Validación de datos

2.1. Descripción

La aplicación comprueba que el nombre y la contraseña introducidas por el usuario son correctas.

2.2. Condiciones de ejecución

El usuario se encuentra en la pantalla de inicio de sesión.

2.3. Entrada

El usuario introduce nombre y contraseña.

2.4. Validaciones

La aplicación conecta con la base de datos verificando la existencia o no del nombre; y, en caso de que exista, procede a comprobar si la contraseña introducida por el usuario se corresponde con la contraseña real del usuario.

2.5. Resultado esperado

Si la validación anterior tiene resultado positivo, el usuario inicia sesión y se le muestra la vista de selección de actividad. Si el resultado es negativo, la aplicación muestra un error, y espera un nuevo inicio de sesión.

Caso de Prueba: Añadir Empleado

1. Introducción

Este artefacto cubre el conjunto de pruebas realizadas sobre el Caso de Uso Alta Empleado.

2. Validación de datos

2.1. Descripción

La aplicación comprueba los permisos que tiene el usuario que va a añadir el empleado y los datos del empleado a añadir.

2.2. Condiciones de ejecución

El usuario ha iniciado la sesión, ha introducido el tipo de empleado que va a ser y se encuentra en el formulario ***EmpleadoPAS*** o ***EmpleadoPDI***.

2.3. Entrada

El sistema proporciona la información del usuario que desea dar el alta.
El usuario primero introduce el tipo de empleado que va a ser: PDI o PAS. Después, introduce los datos del nuevo empleado rellenando el formulario ***EmpleadoPAS*** o ***EmpleadoPDI***.

2.4. Validaciones

Tras comprobar que los tipos de datos introducidos por el usuario son correctos, la aplicación comprueba si los permisos del usuario que desea dar el alta son los adecuados para realizar esta operación, que la conexión con la base de datos siga establecida y que el empleado exista o no. El resultado de esta comprobación será positiva si los permisos son los adecuados, la conexión se mantiene establecida y el empleado no existe.

2.5. Resultado esperado

Si la validación anterior tiene un resultado positivo, se da de alta el nuevo empleado. Si es negativo, se muestra un mensaje de error. En ambos casos se vuelve a la pantalla anterior.

Caso de Prueba: Añadir Usuario

1. Introducción

Este artefacto cubre el conjunto de pruebas realizadas sobre el Caso de Uso Alta Usuario.

2. Validación de datos

2.1. Descripción

La aplicación comprueba los permisos que tiene el usuario que va a añadir el nuevo usuario y los datos del usuario a añadir.

2.2. Condiciones de ejecución

El usuario ha iniciado la sesión y se encuentra en el formulario ***usuario***.

2.3. Entrada

El sistema proporciona la información del usuario que desea dar el alta.
El usuario introduce los datos del nuevo usuario rellenando el formulario ***usuario***

2.4. Validaciones

Tras comprobar que los tipos de datos introducidos por el usuario son correctos, la aplicación comprueba si los permisos del usuario que desea dar el alta son los adecuados para realizar esta operación, que la conexión con la base de datos siga establecida y que el usuario exista o no. El resultado de esta comprobación será positiva si los permisos son los adecuados, la conexión se mantiene establecida y el usuario no existe.

2.5. Resultado esperado

Si la validación anterior tiene un resultado positivo, se da de alta el nuevo empleado. Si es negativo, se muestra un mensaje de error. En ambos casos se vuelve a la pantalla anterior.

Caso de Prueba: Ver Ficha de Empleado

1. Introducción

Este artefacto cubre el conjunto de pruebas realizadas sobre el Caso de Uso Ver Ficha de Empleado.

2. Validación de datos

2.1. Descripción

La aplicación comprueba los permisos del usuario que desea visualizar un empleado.

2.2. Condiciones de ejecución

El usuario ha iniciado la sesión y está visualizando la lista de empleados de la aplicación.

2.3. Entrada

El sistema proporciona la información del usuario que desea dar el alta.
El usuario selecciona el empleado a visualizar.

2.4. Validaciones

La aplicación comprueba si los permisos del usuario son los adecuados para realizar esta operación y que la conexión con la base de datos siga establecida. El resultado de esta comprobación será positiva si los permisos son los adecuados y la conexión se mantiene establecida.

2.5. Resultado esperado

Si la validación anterior tiene un resultado positivo, se muestra la información del empleado seleccionado. Si es negativo, se muestra un mensaje de error y se muestra de nuevo la lista de empleados.

Caso de Prueba: Realizar una búsqueda

1. Introducción

Este artefacto cubre el conjunto de pruebas realizadas sobre el Caso de Uso Realizar Busqueda.

2. Validación de datos

2.1. Descripción

La aplicación comprueba los permisos que posee el usuario para ver los empleados de la busqueda que desea realizar.

2.2. Condiciones de ejecución

El usuario ha iniciado la sesión y está visualizando la lista de empleados de la aplicación.

2.3. Entrada

El sistema proporciona la información del usuario que esta logeado
El usuario introduce el tipo de parametro y de empleado a buscar

2.4. Validaciones

La aplicación comprueba que el tipo de empleado introducido sea el correcto, que los permisos del usuario son los adecuados para realizar esta operación y que la conexión con la base de datos siga establecida. El resultado de esta comprobación será positiva si el tipo de empleado es PDI, PAS o GENERAL, si los permisos son los adecuados y si la conexión se mantiene establecida.

2.5. Resultado esperado

Si la validación anterior tiene un resultado positivo, se muestra el listado de empleados filtrado por el tipo de empleado y el tipo de parametro de busqueda.. Si es negativo, se muestra un mensaje de error y se vuelve a pedir al usuario que elija el tipo de datos adecuado.