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 1](#_Toc452075517)

[Modelo de Análisis 2](#_Toc452075518)

[Modelo de Diseño 6](#_Toc452075519)

[Pruebas 23](#_Toc452075520)

# 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 |

# 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 vista 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 sus permisos.

Estos 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.

#Añadir diagrama clases puestas vacias

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.
* Horas\_de\_trabajo: número de horas de trabajo designadas a ese empleado.
* Fecha\_Fin: 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 subsistemaas ya mencionados en la sección de diseño, y pasaremos a describir las partes que lo componen. Para ello utilizaremos diagramas de clases, de secuencia, y de actividades.

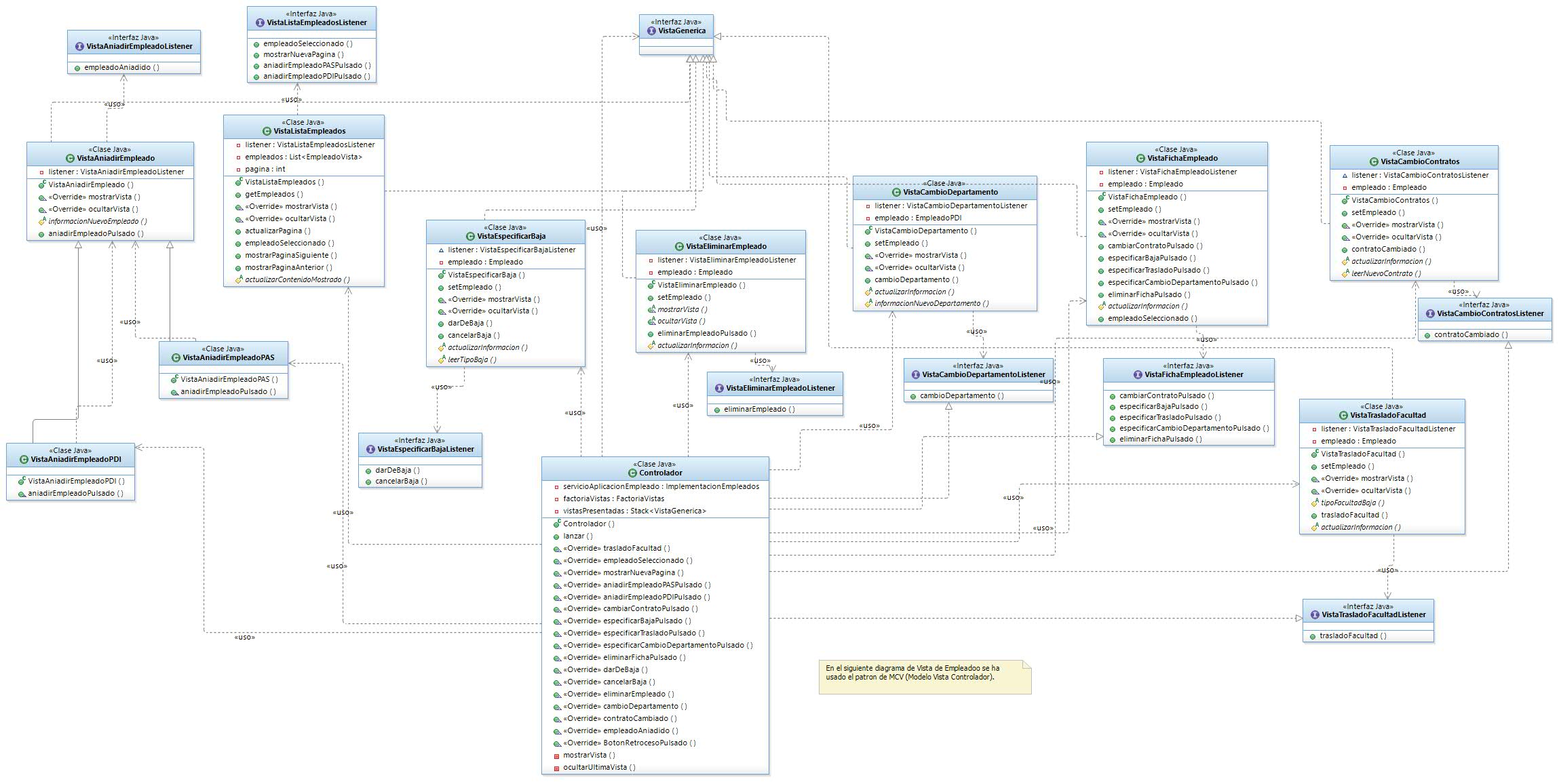
## Subsistema Empleado

Dividiremos el sistema entres capas, utilizando la arquitectura multicapa. Tendremos una capa de presentación(encargada de ls visualización de la aplicación), la capa ne 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. 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 poir 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.

Es interesante observar que hemos implementado el ***Patrón Factoria*** para crear nuestras vistas. En efecto, todas nuestras vistas son abstractas para no limitarno a una implementación concreta, y estar abiertos a distintas posibles interfaces. De esta forma, el controlador solicita la vista apropiada a una factoria abstracta que proporcionara la vistra adecuada según el tipo de interfacz en el que estemos(ventana, consola…).

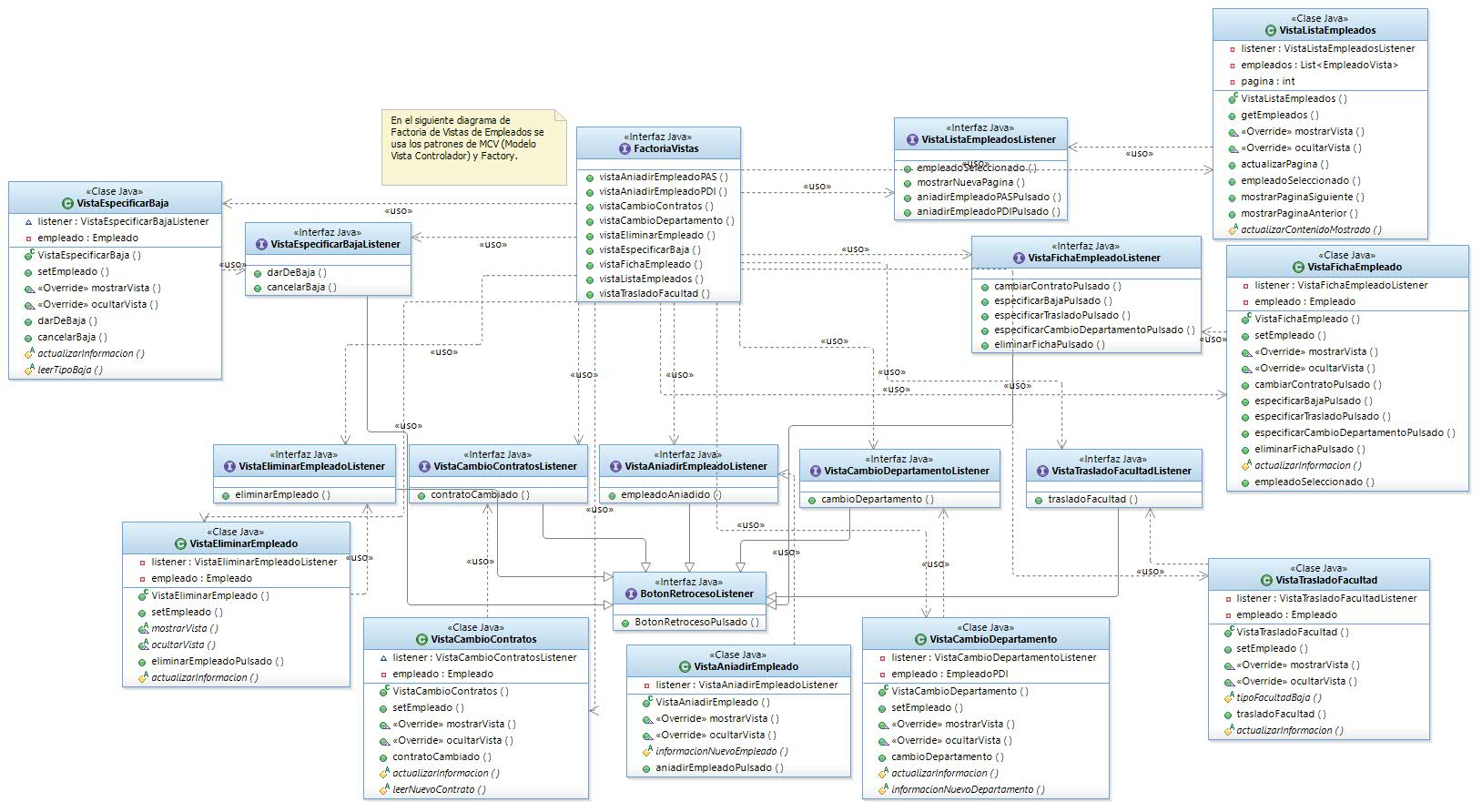


Como cada una de las vistas se refiere a un caso de uso, cuando hablamos de implementar el formulario asociado a una acción, este tiene los campos que ya enumeramos en el caso de uso correspondiente(ver documento *Especificacion\_de\_requisitos*).

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

* *VistaAniadirEmpleado*: contiene los componentes comunes a para añadir un empleado de cualquier tipo.
* *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 la interfaz de usuario para cambiar el contrato de un empleado.
* *VistaCambioDepartamento*: permite cambiar el departamento de un empleado.
* *VistaEliminarEmpleado*: muetra la opción al usuario 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”.*



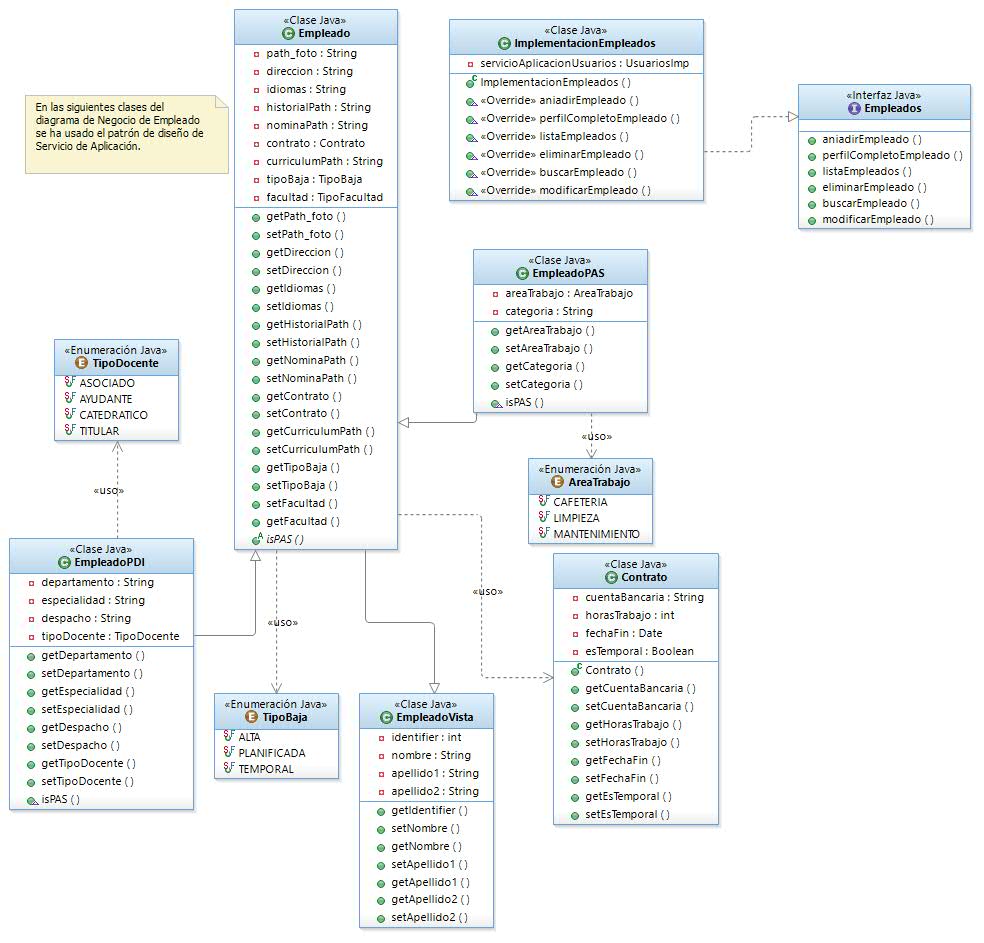
La clase *FactoríaVistas* implementa la factoria ya comentada, y tiene un método correspondiente a cada tipo de vista que se puede instanciar.

La interfaz *BotonRetrocesoListener* es la parte común de todas las interfaces de cada uno de los tipos de vistas proporcionando elementos como salir, retroceder…, en general volver a la vista antrerior.

### Capa de negocio

En esta capa desarrollaremos la lógica de nuestra aplicación, que será la encargada de procesar las peticiones de la vista (que comprobará, y que en caso de ser validas)y comunicará 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, los situaremos en esta capa debido a que esta es la que más los representa.

### 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.

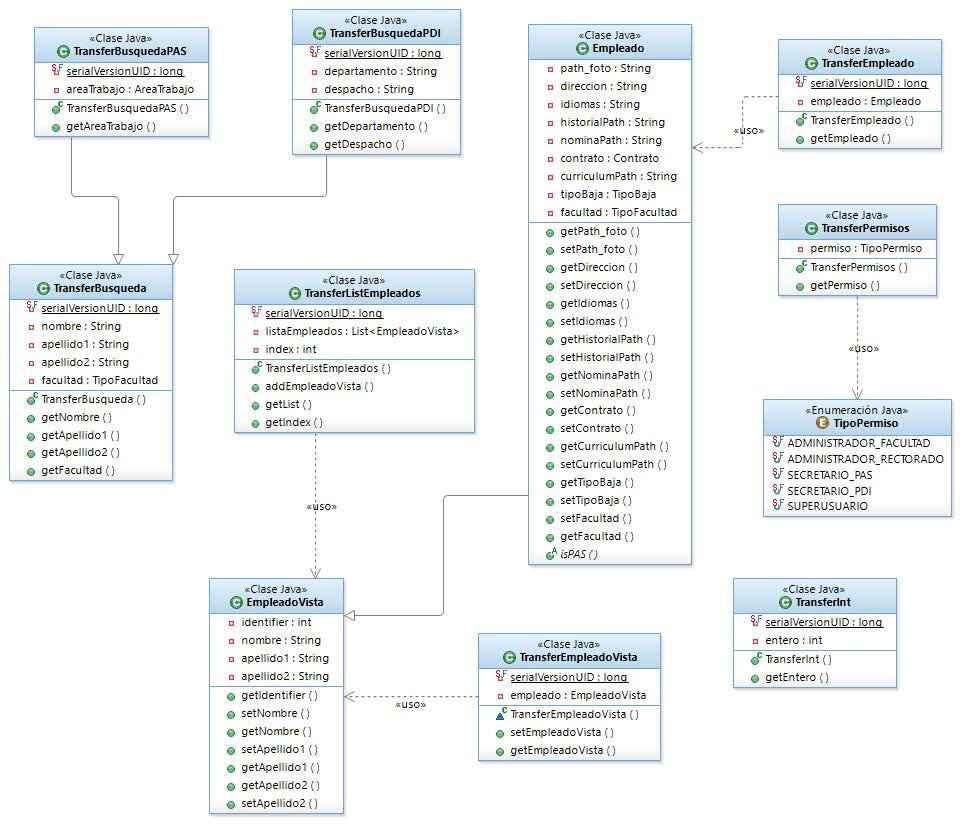
* Reglas

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, catedráticco y titular)
* Objetos de transferencia

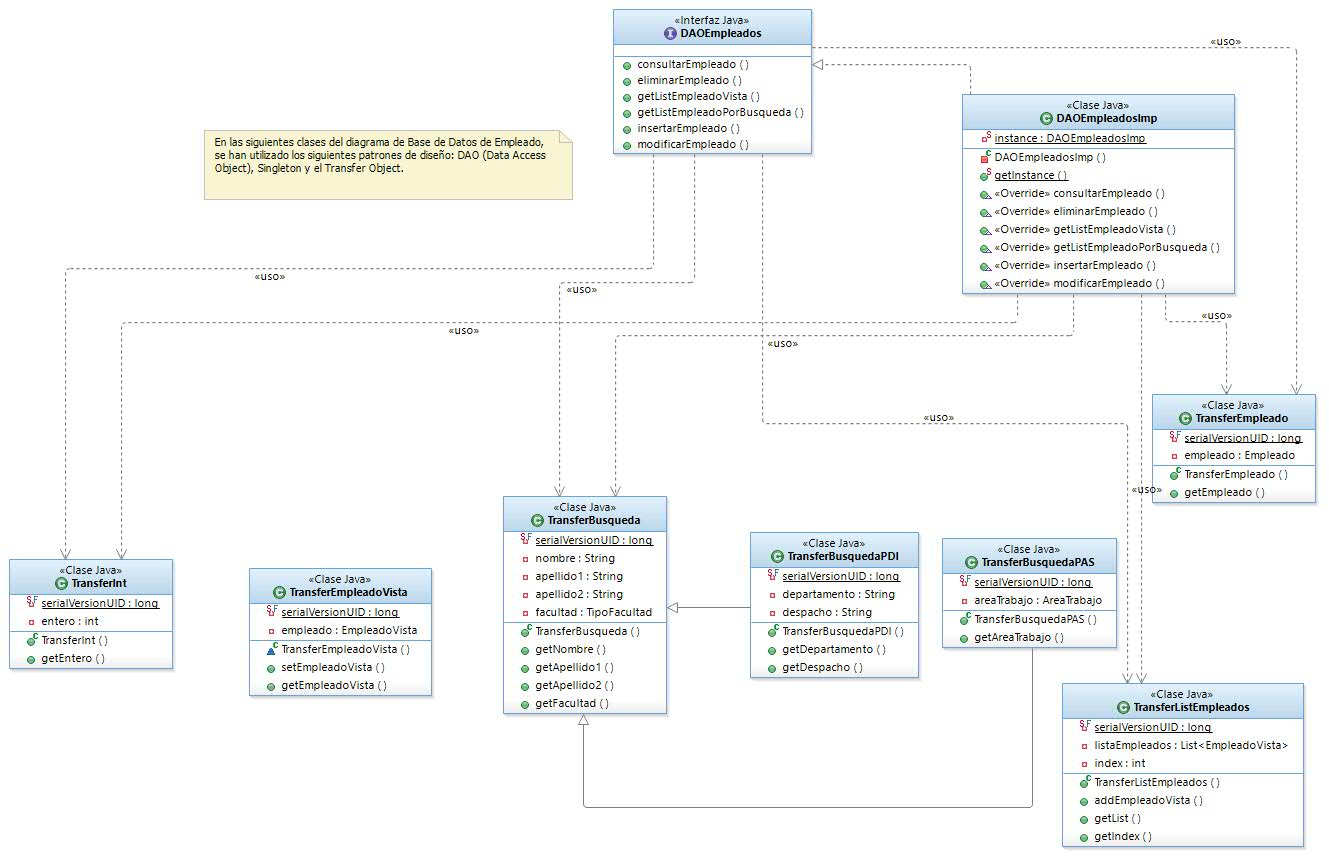
Los objetos de transferencia son usados por todo el módulo para transferir información entre distintas capas. Tenemos los siguientes tipos: *TransferBusqueda*, *TransferBusquedaPAS*, *TransferBusquedaPDI*, *TransferEmpleado*, *TransferEmpleadoVista*, *TransferInt*, *TransferListEmpleados*, *TransferPermisos*, 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 se adjunta a continuación:



### Capa de integración

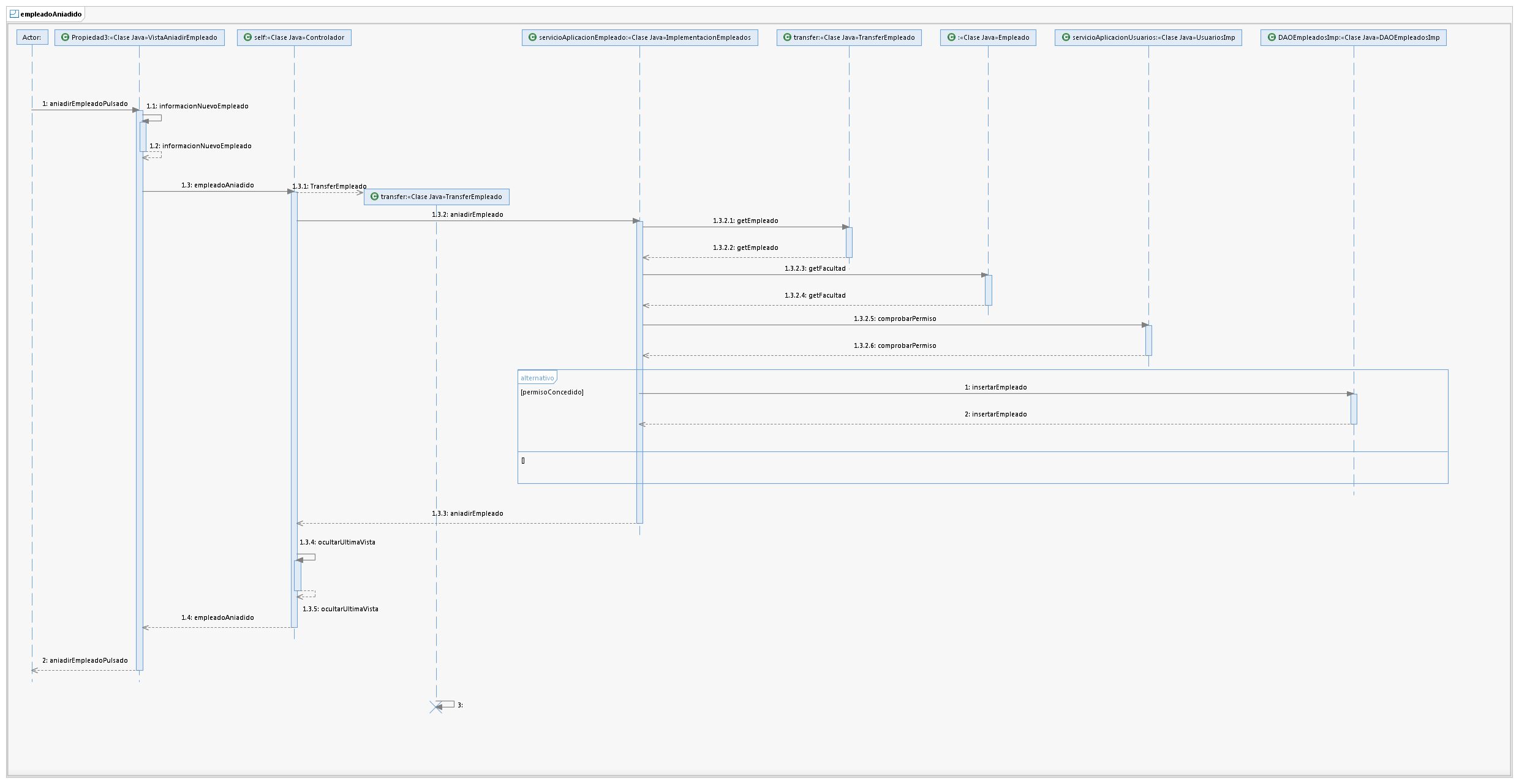
En esta capa se realizará el intercambio de datos con una unidad de almacenamiento de información(independientemente de que sea física o digital, tales como bases de datos, documentos…).



Hemos utilizado el patrón de diseño *DAO* (“Data Access Object”), que permite proveer operaciones de datos específicas, sin exponer los detalles del almacenamiento que haya detrás.

A continuación pasaremos a hacer un recorrido por las distintos casos de uso de nuestra aplicación, describiendolas mediante 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 transfer empleado. Este transfer empleado lo obtendrá el ControladorEmpleado. Este controlador llamará a su método aniadirEmpleado,

el cual obtendra la facultad del trasnfer empleado mediante el metodo getFaculad el cual es posible gracias a haber obtenido el empleado completo con getEmpleado.

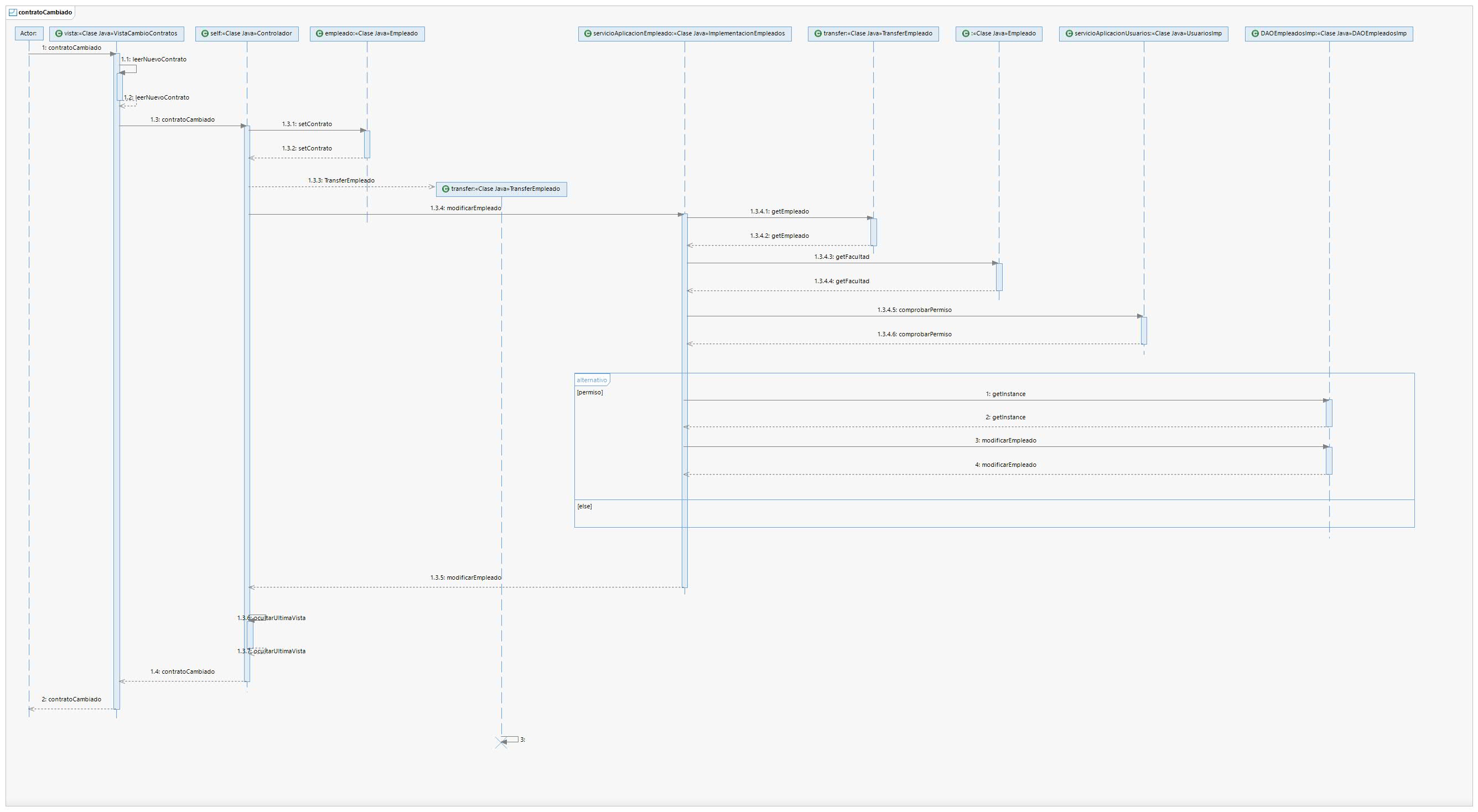
Esta facultad es un dato el cual el serviciodeAplicaciónEmpleado de la capa de negocio necesita para llamar a comprobacionPermisos con dos argumentos uno de TipoPermiso y otro de la facultad,

que determinará si el usuario posee los permisos necesarios para crearlo. En el caso de que el usuario si posea los permisos 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 ControladorEmpleado mostrara un mensaje de error y no se realizará la acción de

añadir empleado. Finalmente el ControladorEmpleado ocultará la vista de añadir empleado y se recuperará la vista anterior en la que comenzó.

### Cambiar contrato



El siguiente 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 ControladorEmpleado que modificara el contrato del objeto empleado del que se quiera cambiar el contrato con setContrato

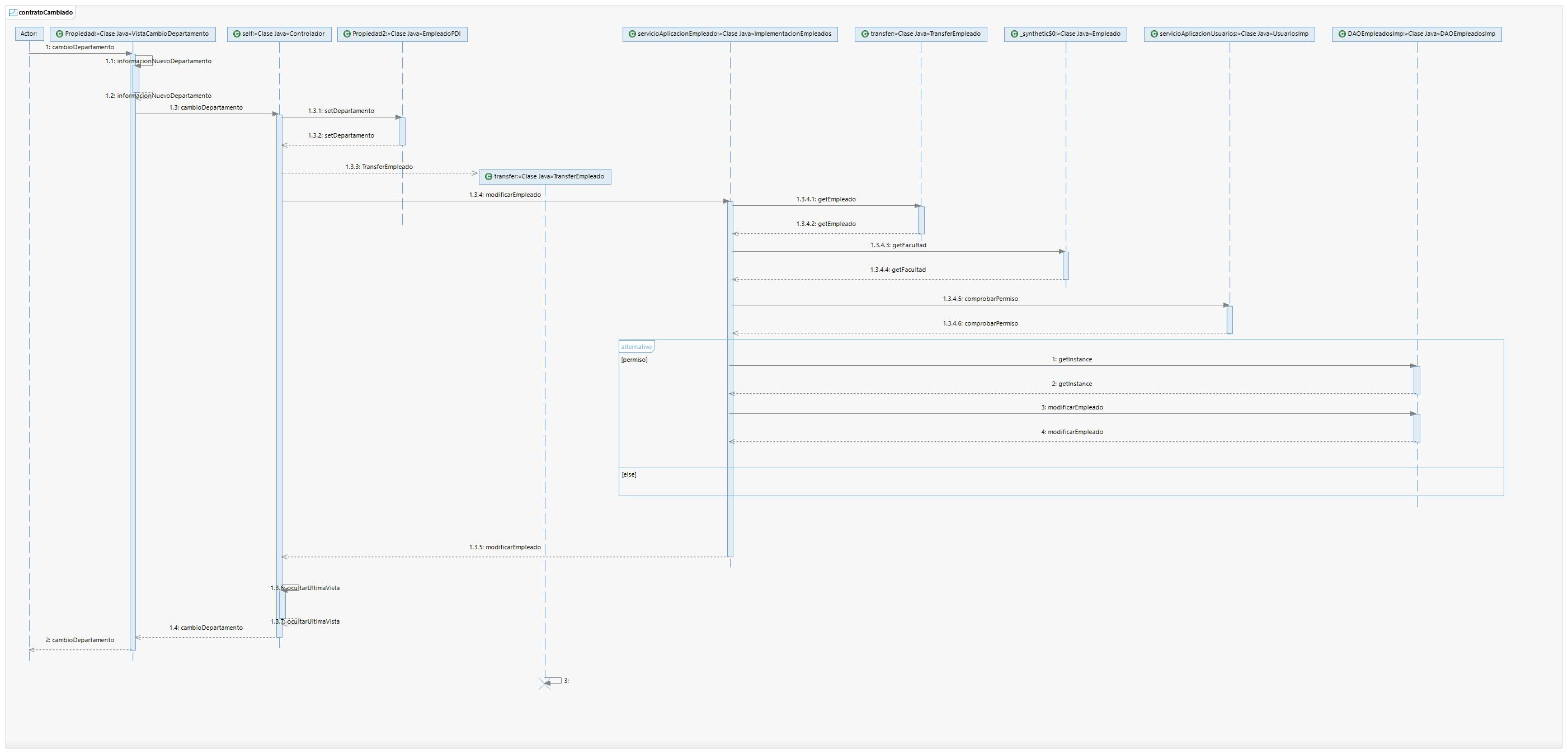
y creará un nuevo TransferEmpleado que le pasará a la capa de negocio que mediante su clase ImplementaciónEmpleados obtendra la facultad del empleado primero con getEmpleado y luego con getFacultad,

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. Por último el ControladorEmpleado

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 ControladorEmpleado que modificara el departamento del objeto empleado del que se quiera cambiar con setDepartamento.

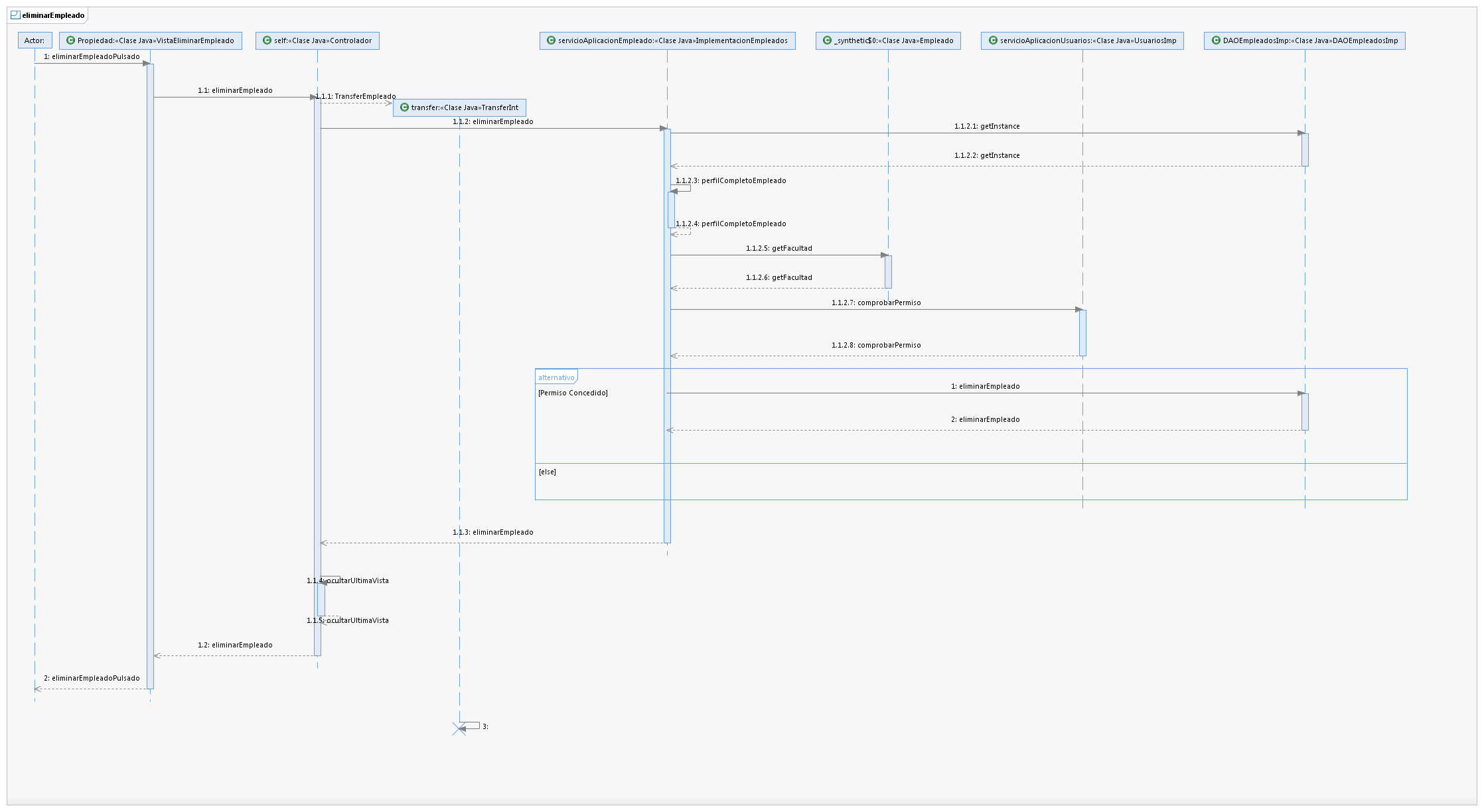
Este ControladorEmpleado creará un nuevo TransferEmpleado que le pasará a la capa de negocio que mediante su clase ImplementaciónEmpleados obtendra la facultad del empleado primero con getEmpleado y luego con getFacultad,

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.

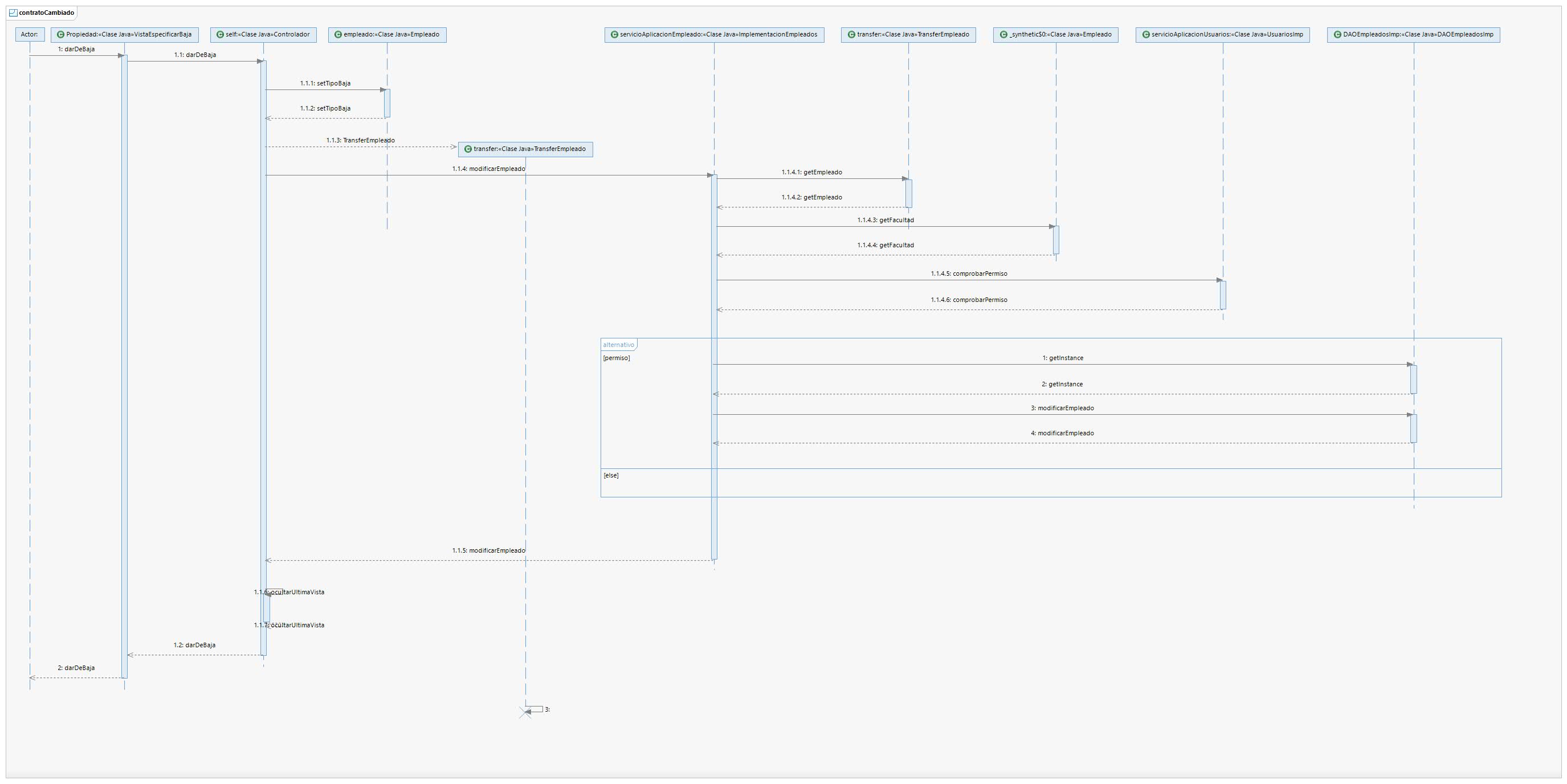
Si el usuario posee los permisos necesarios entonces el servicioAplicación de Empleado llamará al DAOEmpleados y modificará el empleado de la base de datos cambiando así el departamento del empleado.

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

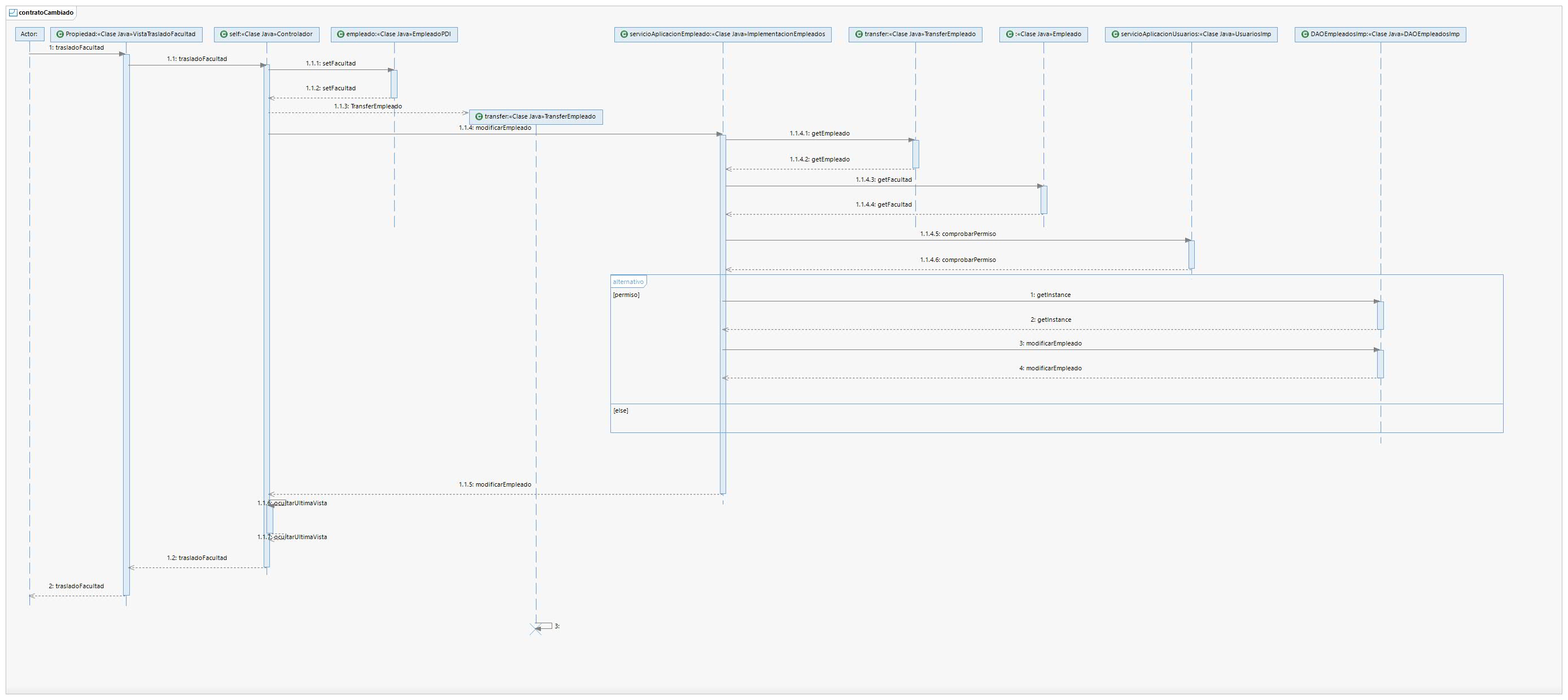
### Eliminar empleado



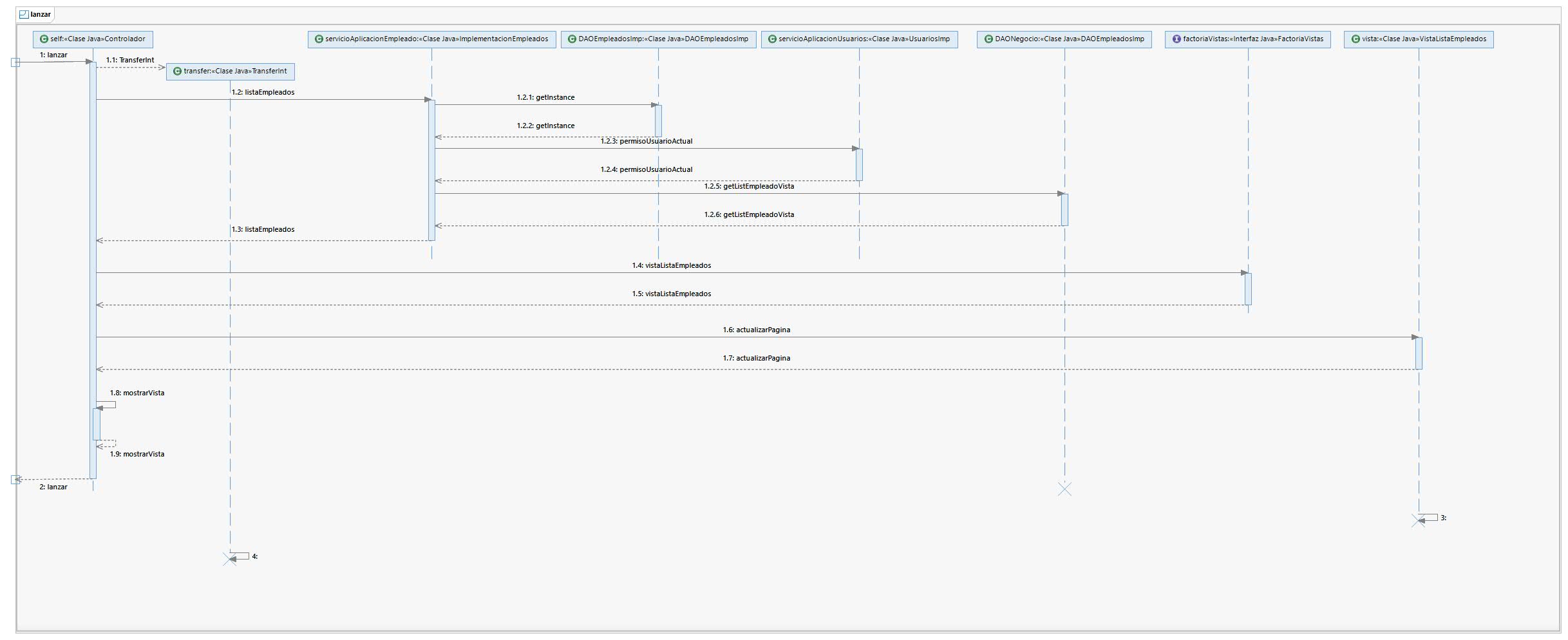
### Especificar baja



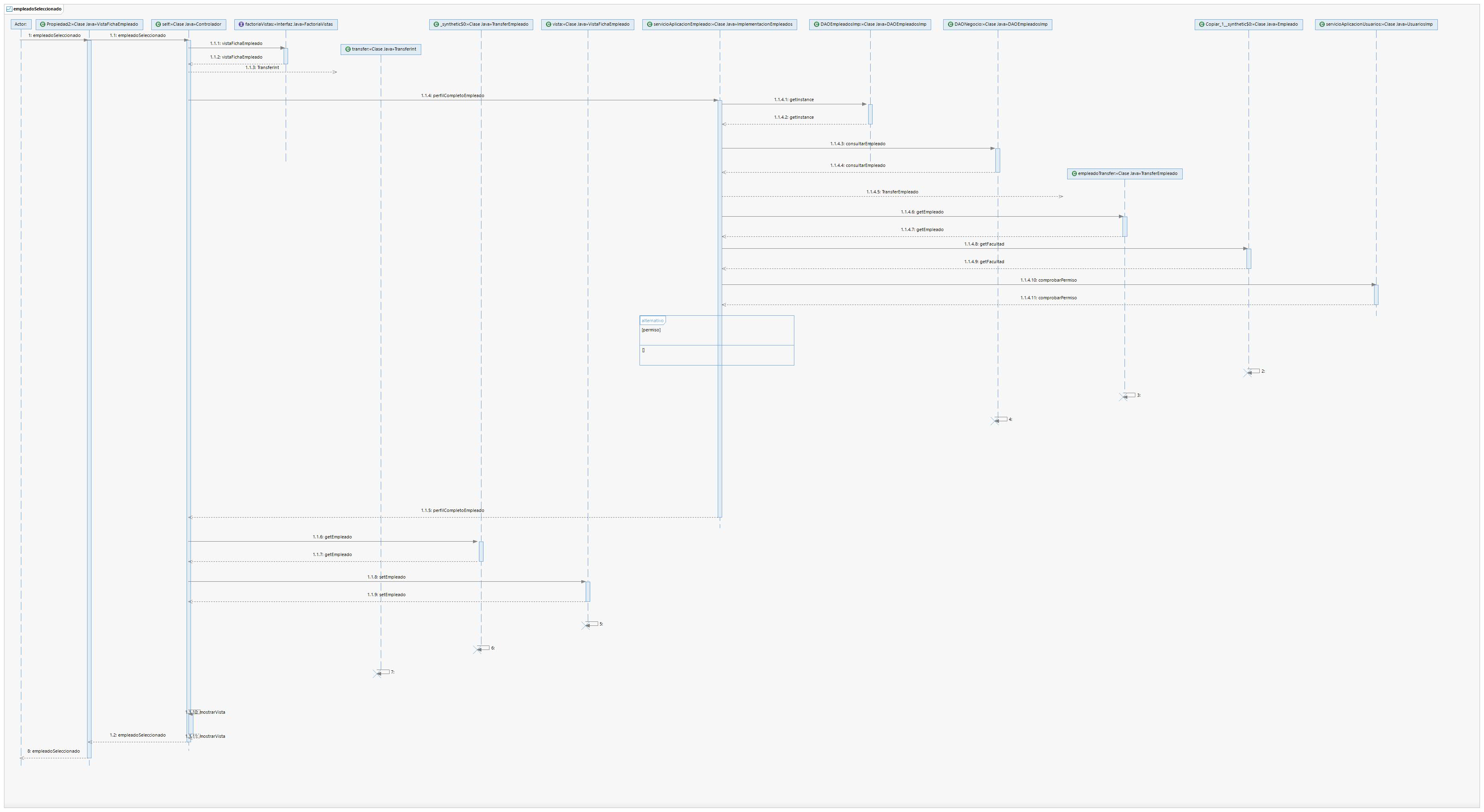
### Especificar traslado de facultad



### Mostrar lista empleados



### Ver ficha empleado



## Subsistema Usuario

De igual manera que en el subsistema de Empleado. 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 división principal 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*”.

# 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
   1. Descripción
   2. Condiciones de ejecución
   3. Entrada
   4. Validaciones
   5. Resultado esperado

### Caso de Prueba: Login

1. Introducción

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

1. Validación de datos
   1. Descripción

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

* 1. Condiciones de ejecución

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

* 1. Entrada

El usuario introduce nombre y contraseña.

* 1. 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.

* 1. 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.

1. Validación de datos
   1. Descripción

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

* 1. 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***.

* 1. 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***.

* 1. 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.

* 1. 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.

1. Validación de datos
   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.

* 1. Condiciones de ejecución

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

* 1. 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***

* 1. 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.

* 1. 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.

1. Validación de datos
   1. Descripción

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

* 1. Condiciones de ejecución

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

* 1. Entrada

El sistema proporciona la información del usuario que desea dar el alta.

El usuario selecciona el empleado a visualizar.

* 1. 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.

* 1. 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.

1. Validación de datos
   1. Descripción

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

* 1. Condiciones de ejecución

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

* 1. Entrada

El sistema proporciona la información del usuario que desea dar el alta.

El usuario introduce el tipo de empleado a visualizar.

* 1. 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.

* 1. Resultado esperado

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