CURSO 2016 - 2017

ACTIVIDAD EVALUABLE Y CALIFICABLE

1. Portada con:

Asignatura: 71013035 – Diseño de Software

Año de la práctica:

Centro Asociado al que pertenece:

Tutor que le da asistencia:
(o grupo de tutoría Intercampus al que está adscrito)

Datos personales:
- Nombre y apellidos:
- DNI o número de expediente:
- Contacto (teléfono o correo electrónico):
- Localidad de residencia:

Datos de coste:
- Horas de dedicación al estudio de los contenidos:
- Nº de actividades no evaluables realizadas y horas de dedicación:
- Horas de dedicación para realizar esta actividad:

Juan del Rosal, 16
28040, Madrid

Tel: 91 398 89 10
Fax: 91 398 89 09

swdesign@issi.uned.es

2. El enunciado y planteamiento del caso de estudio.

El dominio del problema es un **sistema de gestión para la venta de billetes de avión en una agencia de viajes** (e-Vuela).

El caso de estudio es similar al punto de venta (PdV) del libro de la asignatura y, como ahí, el trabajo se centrará, exclusivamente, en la lógica del negocio y en los servicios técnicos mínimos necesarios para implementar esa lógica en la aplicación. Es decir, aunque exista una capa de presentación, no debe tener en cuenta la interfaz de usuario, la presentación o cómo se capturan los eventos. Céntrese en la lógica de la interacción entre el usuario-agencia y su funcionamiento.

Se pretende que la aplicación tenga flexibilidad para ser utilizada tanto por un *operador* en un terminal de la propia agencia de viajes, por un *cliente* desde un dispositivo móvil o por un *administrador*, para realizar labores de supervisión y mantenimiento. Cada uno de estos perfiles determina las funcionalidades disponibles para un *usuario* concreto. Por tanto, la utilización de esta aplicación se restringe a usuarios registrados, con sus datos de acceso, personales y, en el caso de clientes, de facturación, almacenados en el sistema. En cualquier momento se puede registrar o dar de baja un nuevo *cliente* y cualquier usuario puede editar sus propios datos.

A parte de la interacción debida al mantenimiento de una cuenta (alta, baja, modificación de los datos del usuario, etc.), los servicios de la aplicación se originan cuando un usuario solicita o consulta un viaje en avión, con un número de plazas (pasajeros), entre un origen y un destino, para una fecha y condiciones de uso determinadas (necesidades especiales del viajero, facturación de equipaje, equipajes especiales, etc.). Con esos datos, la aplicación obtiene una *lista de itinerarios* mediante un sistema (externo) de planificación que, a su vez, consulta a las compañías aéreas que operen en cada punto de los itinerarios. Un itinerario consiste en una secuencia de uno o más vuelos entre el origen, las escalas intermedias (si las hay) y el lugar de destino. Cada itinerario contiene (referido a un único pasajero):

- Los datos de cada vuelo, con la siguiente información:
  1. Número de vuelo.
  2. Fecha y hora de salida (local del origen).
  3. Fecha y hora de llegada (local del destino).
  4. Aerolínea.
  5. Lugar de origen del vuelo.
  6. Lugar de destino del vuelo.
  7. Duración del vuelo.
- Duración total del viaje.
- Número de plazas (pasajeros).
- Coste total del viaje por pasajero. En el coste se incluyen los precios de cada vuelo así como las tasas (de aeropuerto, etc.) que correspondan.

- Condiciones de uso: facturación de equipaje incluido en el precio y otra información pertinente al respecto.

El resultado de la consulta (lista de itinerarios), se puede almacenar en su perfil, recuperar, actualizar y ordenar según la duración del viaje o su precio total. Con una lista de itinerarios, el usuario también puede seleccionar uno o más itinerarios y **reserva**rlos. Al hacer la reserva se '*bloquean*', durante un tiempo (24 h), las plazas solicitadas para los vuelos de ese itinerario (¡¡Aún no se compran!!). Cada reserva queda reflejada en la lista de itinerarios y, transcurrida su vigencia, desaparece (se '*libera*') al recuperar o actualizar la lista de itinerarios.

Tras hacer una reserva, si está vigente, un usuario puede **comprar** los billetes del viaje de la siguiente manera:

1. Registrando al *cliente*, especialmente sus datos de facturación; o editándolos o confirmándolos si ya estaba registrado.
2. Introduciendo los datos, necesarios para los vuelos, de cada viajero.
3. El *cliente* realiza el pago del importe completo: de todos los vuelos, de todas las tasas, de todos los viajeros...
4. La aplicación emite los billetes de todos los vuelos del itinerario, nominales para cada viajero.

Los detalles y simplificaciones admitidas son:

- Un usuario sólo puede manejar su propia información. Un *operador*, además, puede compartir las listas de itinerarios con cualquier *cliente* o emitir billetes. Un *administrador*, además, puede realizar cualquier operación y manejar la información de cualquier usuario con los otros dos perfiles.
- Se obviará la relación entre las compañías aéreas y la agencia de viajes, que capacita a esta última para '*bloquear*' un determinado número de pasajes (reserva) en un vuelo. No obstante, dichas compañías aéreas son las únicas que manejan cualquier elemento relacionado con los vuelos. Es decir, si bien el planificador (externo) será el encargado de obtener, por parte de las compañías, la información de los vuelos, serán ellas las únicas que emitan los billetes de sus vuelos, previo pago y nominales para cada viajero, cuando lo reclame la agencia.
- Las fechas y horarios son siempre locales. En el cómputo temporal se ignorarán desfases por usos horarios, cambio de año, etc.
- En la reserva y compra, no se contempla la opción de seleccionar asiento.
- Los precios se presentan y manejan en la divisa local de la geolocalización en la que opera el usuario. Aunque se ignorarán los procedimientos de conversión de divisa, la aplicación se debe poder usar en cualquier país.
- En el coste de los vuelos, se considerará el precio por la facturación de única maleta estándar por viajero estándar. Es decir, no se considerarán los sobrecostes por exceso de equipaje o condiciones especiales que queden fuera de la tarifa establecida en el itinerario. Así mismo, la comisión por los servicios (según las políticas de precios y descuentos de la agencia de viajes) estará incluida en el precio final del viaje.

- En la **compra**, el pago se realiza **a** la agencia de viajes. En la emisión de los billetes, se ignorará el procedimiento por el que la agencia, a su vez, reparte el pago de los distintos vuelos y tasas entre las compañías aéreas incluidas en el itinerario.
- Si la compra se realiza a través de un *operador*, el pago (del cliente a la agencia) podría ser en cualquiera de las modalidades (efectivo, tarjeta, etc.). Sin embargo, si es el *cliente* el que opera a través de un dispositivo móvil, sólo se considera el pago por medios electrónicos (tarjeta, portales de pago seguro, etc.)

3. **El enunciado de cada cuestión y las respuestas.** Para cada cuestión, incluirá los desarrollos, listados, diagramas y las argumentaciones que estime necesarios.

RECOMENDACIONES: Después de tratar de entender el planteamiento anterior del caso de estudio, léase todos los enunciados de las cuestiones siguientes, hasta el final. Esté atento a algunos indicios que le pueden ayudar para el enfoque de sus respuestas. Es posible que se le pida que describa o relacione algún elemento; lo cual podría hacerle sospechar que debería contar con él.

PREGUNTAS GUÍA: Estas preguntas no forman parte del ejercicio. Su objetivo es ayudar a que dirija sus conclusiones adecuadamente.

¿Qué hace, exactamente, este **software**?

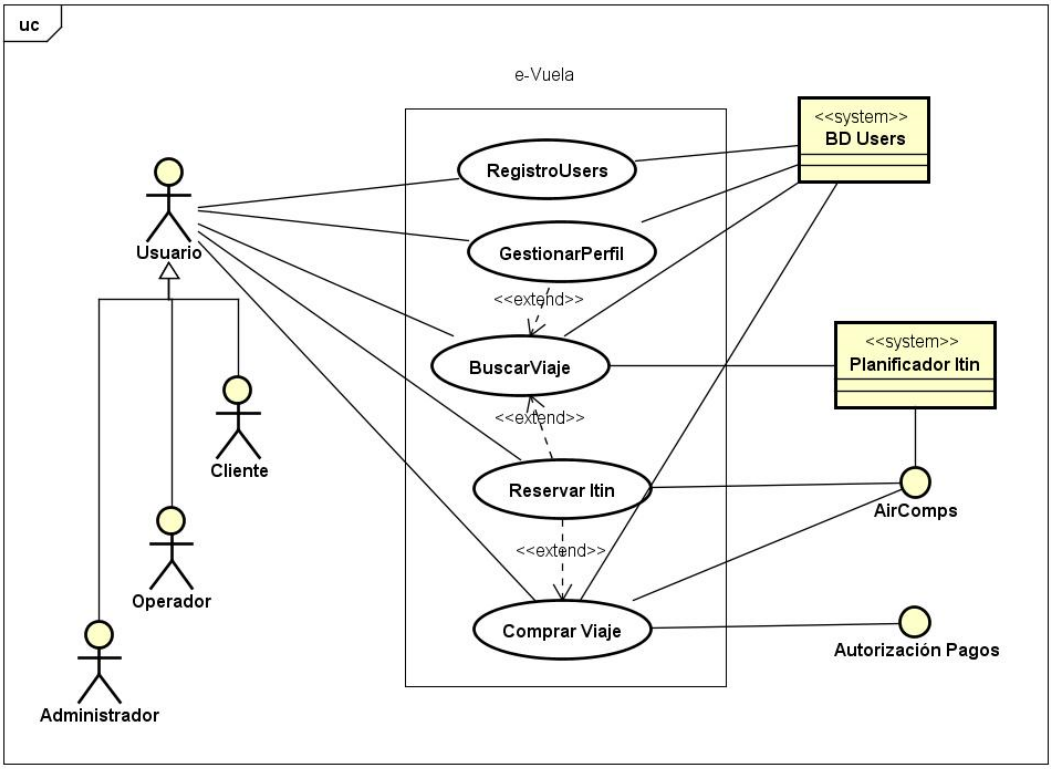
¿De dónde provienen los estímulos? Es decir ¿quiénes son los actores y cuál debe ser su comportamiento (estímulos), frente al que debe reaccionar el software?

Una vez '*aislado*' el software y comprendido el objetivo primordial de su funcionamiento ¿qué secuencia de operaciones debe realizar, como reacción a los estímulos, para que el funcionamiento sea el deseado (Caso de Uso)?

En el comportamiento del software, la situación es parecida: un objeto reacciona ante un estímulo que proviene de otro objeto o del exterior (un actor). Esa es, precisamente, la tarea de asignar responsabilidades. Por tanto, la pregunta es: ¿qué objeto atiende un estímulo y cómo reacciona ante él? Nunca hay que perder de vista que, en definitiva, **estamos hablando de código** y que, esos estímulos, se traducen en llamadas a métodos. Por consiguiente, es inaceptable una invocación a un método inexistente o aquellas llamadas en las que se solicita una reacción que es imposible realizar porque no se dispone de la información necesaria (falta de parámetros, etc.) o ésta es inaccesible.

Sección 1. Evaluación de ***Casos de Uso***

1. (0'5 puntos) En relación al software de e-Vuela <u>considerado en el caso de estudio</u>, identifique al menos 4 casos de uso primarios y sus actores correspondientes. Represente los resultados en un diagrama de casos de uso de UML.



El interés principal del diagrama anterior es:

1. Identificar las primeras operaciones y funciones principales de la aplicación (o el módulo que se está estudiando).
2. Identificar los actores principales: aquellos actores cuyo objetivo principal, al utilizar la aplicación, nos dará su funcionalidad *esencial*, los casos de uso principales. Aunque un usuario genérico está especializado en 3 tipos, el sentido del funcionamiento de la aplicación (el *negocio*) es satisfacer los objetivos fundamentales del *Cliente*:
    a. Hacer una consulta para buscar un viaje.
    b. Comprar un viaje.

    Las otras especializaciones de Usuario tendrán, además de las anteriores, otros objetivos fundamentales que, seguramente, darán lugar a casos de uso principales.

    En cualquier caso los actores principales serán los que generan los estímulos necesarios (o eventos, *deben* ser capaces de ello) para que la aplicación responda con el comportamiento esperado (el *contenido* del caso de uso).
3. El análisis del enunciado puede llevar a detectar otras operaciones completas que, aunque no justifiquen en sí el funcionamiento de la aplicación, pueden responder a otros objetivos importantes para los actores principales:
    a. Registrar un usuario o mantener los datos de una cuenta. Normalmente esta información (datos de identificación, preferencias, medios de pago, etc.) se asimila con el término '*Perfil* de la cuenta'. Pero, en el enunciado, el almacén de consultas de viajes aparece asociado a él, por lo que se asumirá que dicho almacén es un componente del perfil.
    b. Guardar o recuperar una consulta en el perfil de usuario (Mantener o Gestionar Perfil).
    c. Reservar un itinerario.
    d. Que un Operador *obtenga* el itinerario de un cliente, o suplante su identidad, para realizar otras operaciones en su nombre.
    e. Mantenimiento de los sistemas de información (B.B. de D.D.) precios, paquetes promocionales y de la aplicación en general.

    Sin perder de vista los dos casos de uso detectados en el punto 2, se puede discutir si estas operaciones son completas, independientes, casos de uso secundarios o primarios.

En este enlace se pueden encontrar unas anotaciones sobre las relaciones de inclusión y extensión entre los casos de uso: "UML use case EXTEND vs INCLUDE relationships.pdf" (obtenido de http://www.uml-diagrams.org)

4. Las conclusiones anteriores facilitan analizar qué tipo de información (datos) se va a manejar en cada una de las operaciones identificadas, qué información es razonable que se maneje dentro de la aplicación (o de la *sesión* correspondiente a los casos de uso o las operaciones que se están manejando), cuál va a provenir de servidores de información externos o, enfocado desde otra perspectiva, qué actores secundarios o 'de apoyo' serán necesarios para que colaboren con los casos de uso identificados y provean dicha información.

Con todo lo anterior se tiene un mapa bastante aproximado del sistema estudiado, de sus operaciones principales, de los actores principales, que generan los estímulos para que se realicen, y de los secundarios y los de apoyo, que colaboran con el sistema para aportar la información adicional necesaria para completar dichas operaciones principales.

2. (1 punto) Escriba el caso de uso <<*ProcesarCompra*>> en un formato completo (se recomienda la variante '*en dos columnas*') y estilo esencial. Incluya tanto el escenario principal de éxito como 2 extensiones o flujos alternativos que pudieran ser frecuentes. Suponga que el escenario de partida para este caso de uso es el de un usuario registrado que ya está identificado en el sistema, que ya había realizado varias consultas para un viaje, para un mínimo de 2 viajeros, y las había almacenado en su perfil. Dicho usuario decide realizar la compra correspondiente a un itinerario, que ya había reservado, de una de las listas de itinerarios que ha recuperado de su perfil (el flujo básico de acciones comienza con la orden de compra del itinerario reservado y termina con la obtención de los billetes). En este flujo principal, considere sólo el pago por medios electrónicos, sin tener en cuenta las políticas de precios, descuentos o promociones de la agencia de viajes. No escriba un encabezamiento demasiado elaborado del caso de uso (es decir, omita *propósito*, *resumen*…); en su lugar, afronte directamente el transcurso típico de los acontecimientos.

Caso de uso: **ProcesarCompra**

Formato completo (variante 'a dos columnas'), estilo esencial.

Evolución típica de los acontecimientos

Acciones del actor (normalmente el Cliente)	Respuesta del sistema
1. El caso de uso comienza cuando el cliente selecciona la operación '*Comprar*' sobre un itinerario reservado.	2. El sistema comprueba la vigencia de la reserva y solicita los datos de cada viajero (necesarios para la emisión de los billetes).
3. Para cada viajero, el cliente facilita los datos pedidos (por ejemplo, rellenando un formulario).	4. El sistema recoge la información de cada viajero, necesaria para que las aerolíneas emitan los billetes.

Se repiten los pasos 3 a 4 hasta que el usuario finalice la entrada.

5. El sistema presenta un resumen con la información de los usuarios, descripción de los vuelos y el precio total con los impuestos calculados.
6. El cliente paga, aportando la información de su tarjeta de crédito.
7. El sistema da opción a que el cliente modifique la información de pago de su perfil.
8. El sistema envía la petición de autorización del pago al sistema externo del Servicio de Autorización de pagos y solicita la autorización del pago.
9. El sistema recibe la aprobación del pago y lo notifica al cliente.
10. El sistema registra el pago a crédito, que incluye la aprobación del pago.
11. El sistema presenta el mecanismo de entrada para la firma del pago a crédito.
12. El cliente introduce la firma del pago.
13. El sistema se pone en contacto con cada aerolínea del itinerario, paga la cantidad que corresponda a los vuelos de esa compañía, para todos los viajeros, y solicita la emisión de los billetes.
14. El sistema registra la venta completa (incluido el pago a las aerolíneas) y envía la información de la venta y el pago al sistema de Contabilidad externo.
15. El sistema transfiere todos los billetes al cliente y presenta un recibo.
16. El cliente recoge los billetes, el recibo y termina la operación.

Alternativas

- 3 a 5 El cliente desea modificar de algún viajero. Se da opción de rectificar la entrada deseada, comenzar desde el 1er viajero o cancelar la compra.
- 8 El sistema detecta un fallo en el diálogo con el sistema de autorización. El sistema solicita un modo de pago alternativo al cliente. De otro modo, se cancela la operación.
- 9 El sistema recibe una denegación de la autorización del pago. El sistema solicita un modo de pago alternativo al cliente. De otro modo, se cancela la operación.
- 13 El sistema detecta un fallo en el diálogo con la aerolínea. Si no se puede obtener algún billete, se cancelan los billetes ya emitidos (para restitución del importe), se hace una devolución del pago al cliente, se registra el error en la operación y se cancela toda la venta.

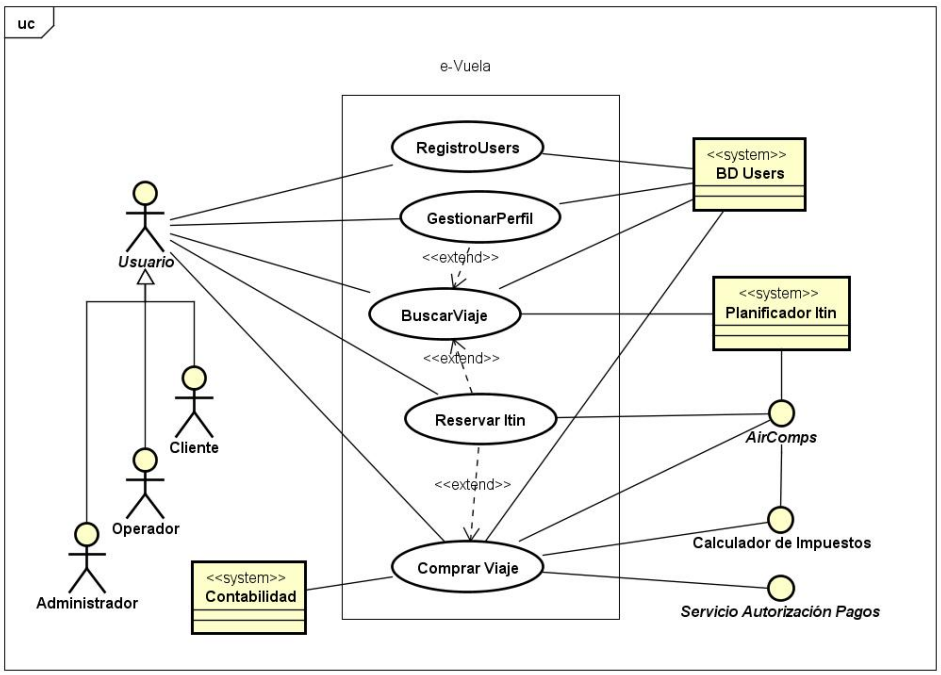
Este apartado está orientado a pensar:

- **QUÉ** tiene que hacer la aplicación (el '*negocio*').
- Qué **pasos** fundamentales debe dar (el software) **para hacerlo**.

En este proceso ya se empieza a identificar la información que se puede necesitar para que la aplicación realice las operaciones de cada paso, si los datos concretos se pueden obtener o deducir de la información que maneja el sistema o, por el contrario, requiere la intervención de algún actor.

Otra vez, cuando se piensa en cómo se va a realizar esto, se concluye que el software de la aplicación sólo debe manejar la información que aporte el cliente y la estrictamente necesaria para realizar las operaciones requeridas (sólo las del caso de uso). Algunas de ellas deben ser versátiles o sí necesitan un volumen de información importante; por lo que no parece razonable que se procese localmente en la sesión. Por ejemplo, la conexión al Servicio externo de Autorización de Pagos o a las Aerolíneas requiere un adaptador que sirva de interfaz para adecuar los formatos de representación de la información y el protocolo de comunicación con cada una. No parece viable que dicha colección de adaptadores (polimorfismo) residan en la sesión. Parece más lógico que la conexión se establezca desde la Agencia de Viajes (equivalente a 'Tienda' en PdV). De igual forma, parece que se va a utilizar cierta información del cliente que, en principio, se asocia como depositada en su perfil: el itinerario, los datos de pago… Típicamente, esa información reside en una base de datos (la de usuarios) que es externa a la lógica del negocio que se está modelando.

Tras lo anterior, un primer refinamiento del diagrama de casos de uso podría ser éste:



Sección 2. Evaluación del ***Modelado Conceptual***

3. (2 puntos) En relación al caso de uso anterior <<*ProcesarCompra*>>, construya un Modelo de Dominio y represéntelo en notación UML. Represente los objetos conceptuales, las asociaciones y los atributos.

Esta sección es importante porque el Modelo es el antecedente del Diagrama de Clases, casi el objetivo final.

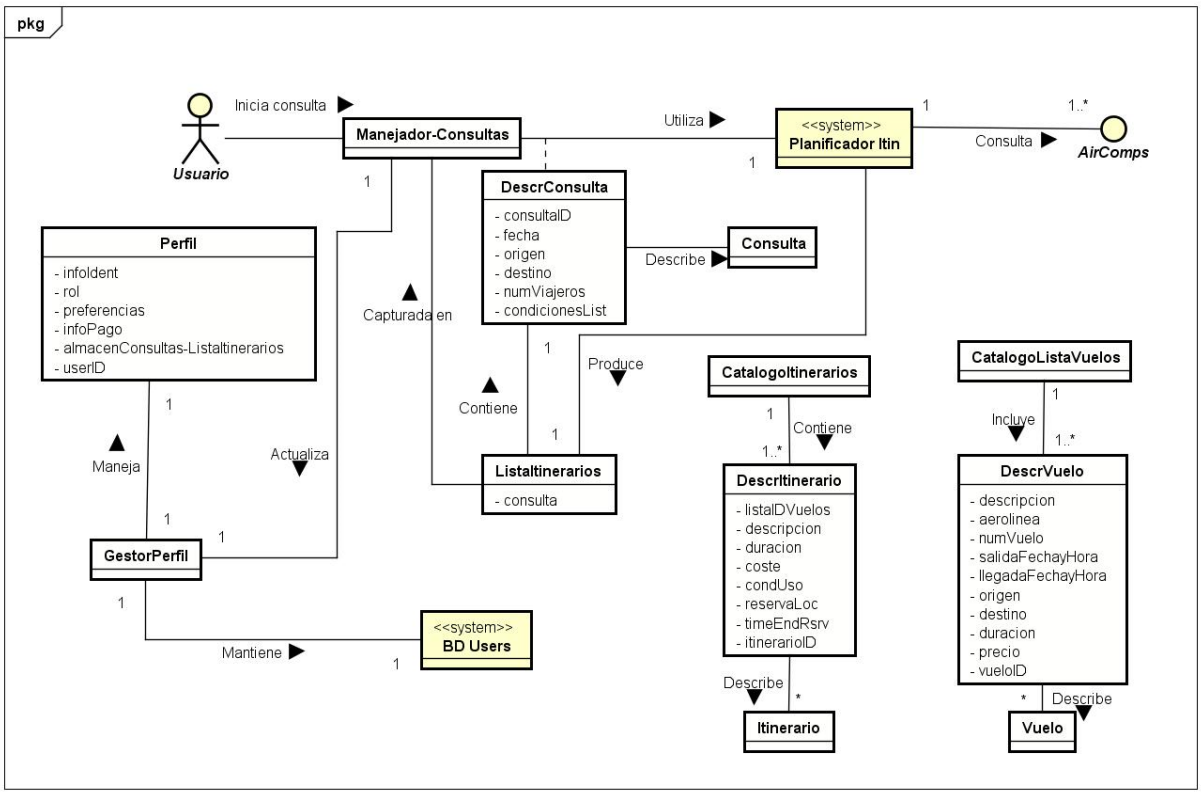
Se trata de asignar, las operaciones identificadas en la escritura del caso de uso, a determinados *elementos* u *objetos*. Como quiera que los principios GRASP (especialmente los 5 primeros) están orientados a mantener una '*privacidad*' en las operaciones, de forma que sólo manejan la información '*local*' (la del objeto en el que están), hay que colocar, la información que se ha identificado como necesaria para una operación, en el mismo objeto en el que está dicha operación. Es la única manera de obtener independencia funcional y, con ello, la flexibilidad que se busca para el funcionamiento del software.

Siguiendo con la idea central de lo anterior, se podría poner el ejemplo PdV del texto de la asignatura. En la escritura del caso de uso de Procesar Venta, se llega a conclusión de que una operación fundamental es registrar los datos de una venta. Por tanto, a la hora de construir el Modelo (y, luego, el diseño de clases) hay que considerar un elemento 'Registro', que recoja el registro de las ventas, y otro, 'Venta', con los datos de la venta. ¿Y cómo Registro captura o registra los datos de una venta? Pues actúa de controlador, recogiendo la información del actor y buscando el resto, y se la provee a Venta para que realice su labor.

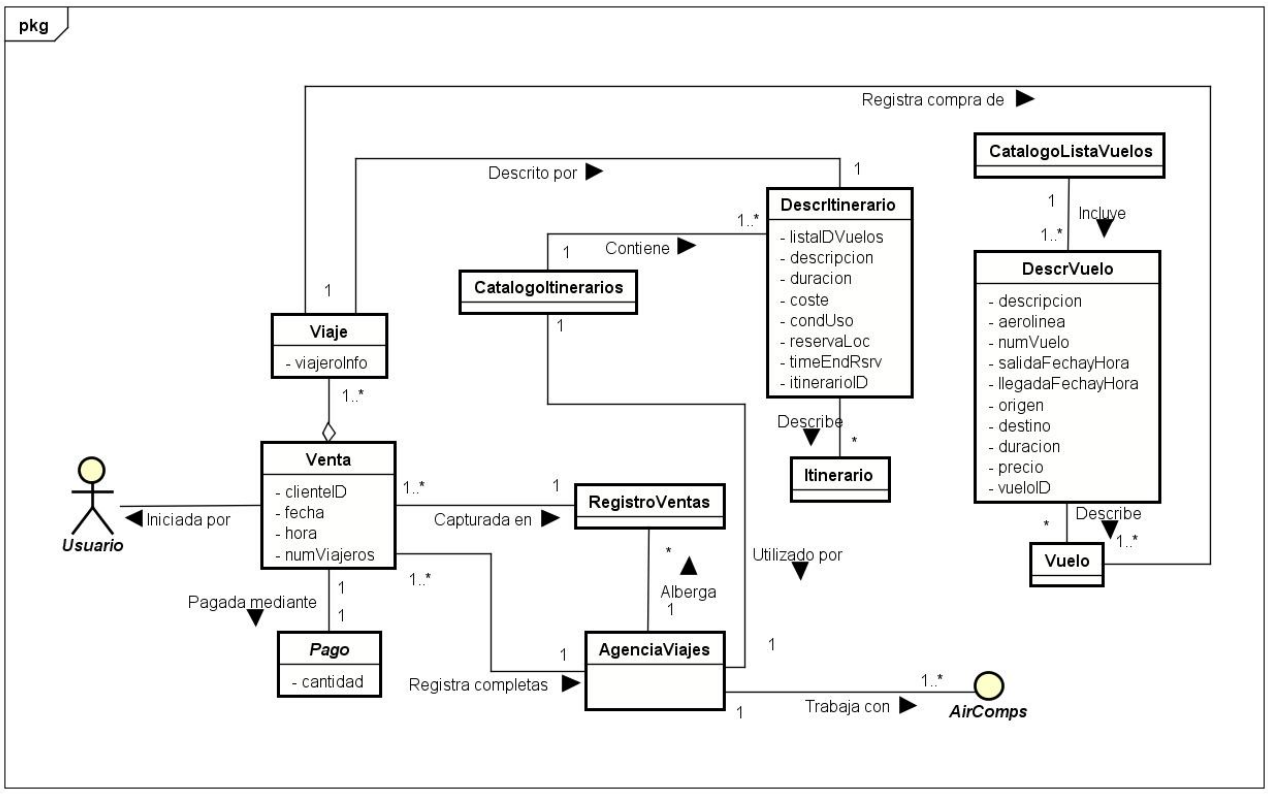
Este caso podría ser similar: es necesario registrar los datos de la **venta de n viajes, operación que consiste en obtener los billetes de los vuelos de cada viajero y pagar por ello**. Esta es la primera conclusión importante del caso de uso.

Con poco que se profundice en un ejercicio, se tiende hacia un planteamiento complicado. Siempre es necesaria una labor de síntesis y abstracción que puede ser difícil al principio. Así, en este caso, aunque el caso de uso está restringido a una situación '*irrealmente*' reducida, la aplicación del procedimiento enunciado anteriormente de "*seguir el hilo de dónde proviene la información que se necesita*" lleva a otra conclusión importante: si el objeto de las '*ventas*' son los '*viajes*' (de cada viajero), éstos consisten, a su vez, en un conjunto de vuelos. Es decir, siguiendo con la analogía con PdV, hay un desdoblamiento anidado en el equivalente a la '*línea de venta*' (cada viaje).

Para comprender mejor la estructura de lo que se está manejando, puede ser útil pensar en cómo se generan los itinerarios: las consultas. Una aproximación superficial al modelo de dominio de ese caso de uso, podría dar este resultado:



A partir de aquí, de la escritura del caso de uso, las analogías con PdV y la comprensión de los catálogos como mecanismo importante para restringir el manejo de la información a la estrictamente necesaria en cada operación, se puede aventurar el siguiente diagrama para el caso de uso de la pregunta:

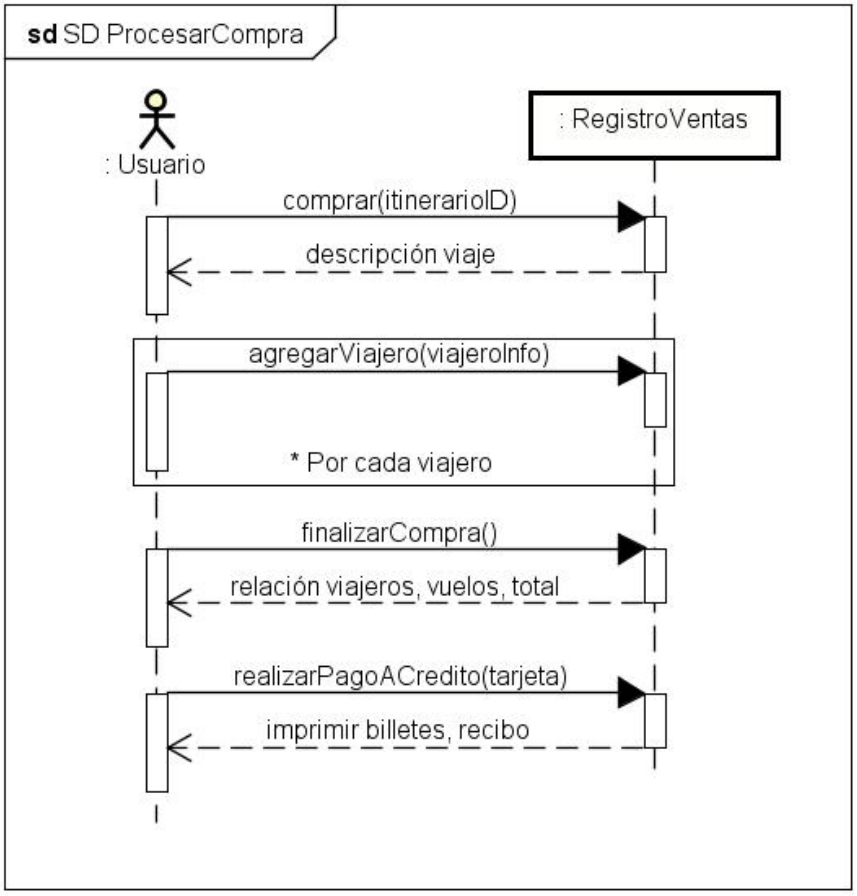


Para los siguientes pasos, se ve que hay más complejidad. La compra del viaje significa la compra de los billetes y, su emisión, se hace tras comprar los vuelos a las aerolíneas. Todo apunta a que, en el proceso de venta, también hay que considerar una especie de pago a los proveedores.

Sección 3. Evaluación de los ***Eventos del Caso de Uso***

Antes de comenzar a asignar responsabilidades (diseño), se suele complementar la comprensión del comportamiento deseado para el sistema (como en el Modelo de Dominio que se acaba de elaborar) con la evaluación de los Eventos del Sistema. Para ello se construye un Diagrama de Secuencia del Sistema (DSS) en el que se analizan los eventos o estímulos que se originan en los actores y cómo reacciona el sistema. Es decir, cuál es el comportamiento del sistema software (tomándolo como un único objeto o elemento), en cuanto a su salida (la reacción '*externa*' hacia el actor), cuando recibe un evento (estímulo) de algún actor.

Podría ser éste:

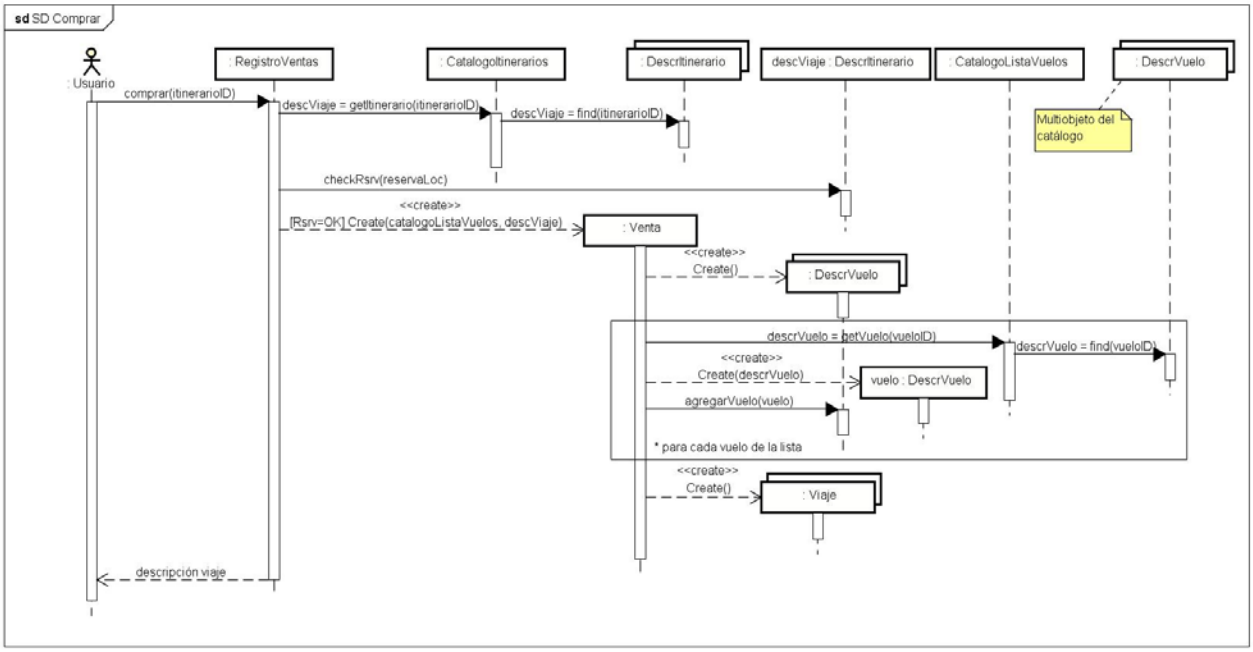


4. (1’5 puntos) Circunscrito al caso de uso anterior <<*ProcesarCompra*>>, construya un Diagrama de Secuencia (diagrama de interacción DS) en UML. Represente los actores y los eventos de los componentes del sistema para este caso de uso.

   NOTA: se pide un diagrama de secuencia en el que represente el paso de mensajes entre los actores y los objetos, **NO** del Sistema (DSS). Por tanto, represente las líneas de tiempo de los objetos identificados en el modelo en lugar de la del *sistema global*.

   A continuación se desglosa el DS anterior en cada una de las operaciones. Se podría dar por correcto un diagrama de secuencia similar al anterior (si induce a que la escritura de los contratos de las operaciones elegidas sea coherente y adecuada para los diagramas de colaboración posteriores) o similar a una unión simplificada de los siguientes:

   IniciarCompra

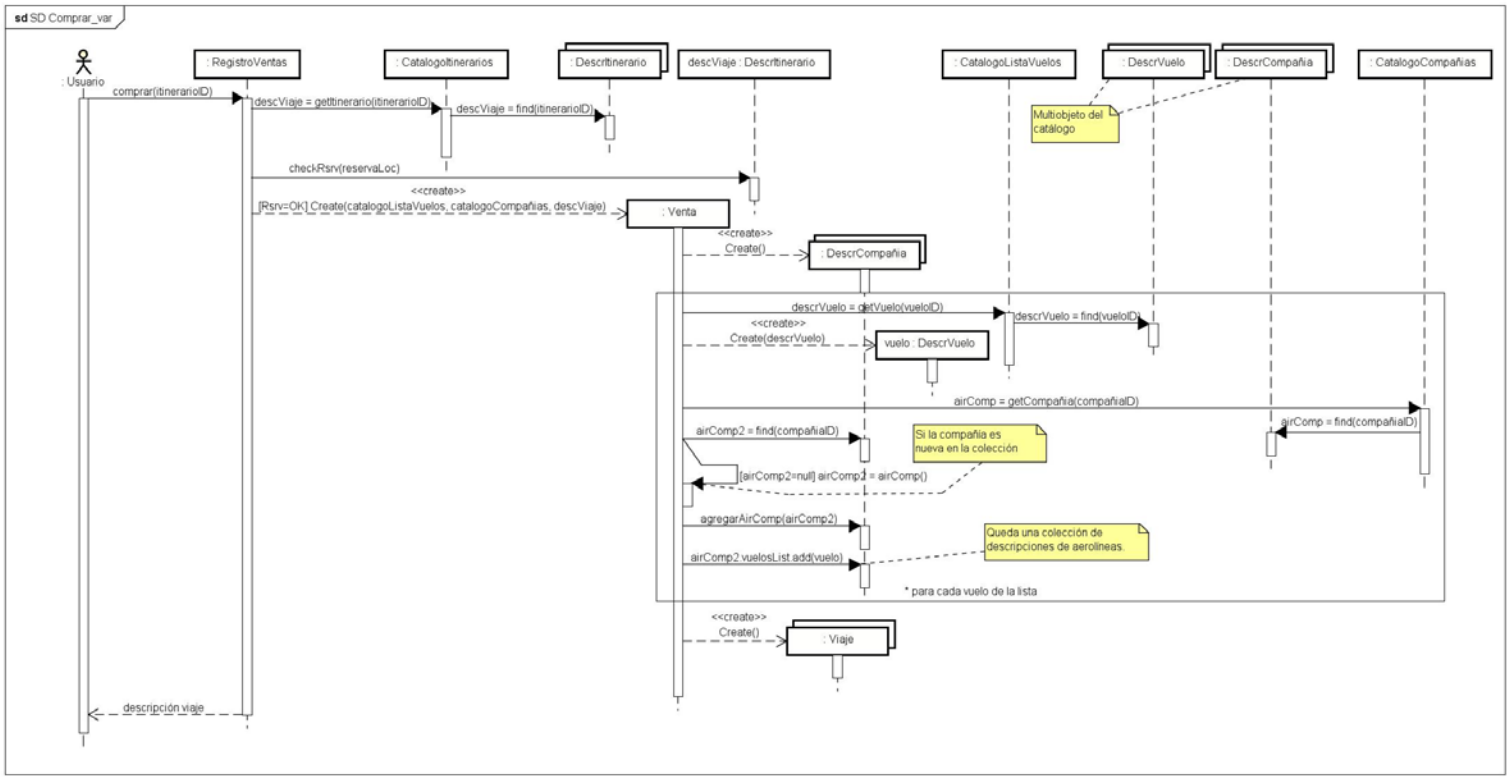


   Nótese:

   - Un actor ‘*humano*’ no puede procesar datos. Sólo puede invocar una acción (en el software) o aportar una información que, en general, no puede ser utilizada directamente por el software. Por ejemplo, un cliente no conoce el código con el que se maneja, internamente, un itinerario.
   - Las ‘*flechas*’ entrantes en una clase son invocaciones a una operación. Significa que esa clase tiene esa operación, conoce y maneja la información de los tipos de los argumentos que acompañan a la invocación, así como del valor que devuelve.
   - Se sigue el criterio de agrupar la información, crear las estructuras de datos y los objetos en cuanto se disponga de los datos necesarios para ello y *se necesiten*, intentando minimizar las acciones necesarias para la realización de las operaciones principales.

En este sentido, a partir de la selección del ID del itinerario y la comprobación de la vigencia de su reserva, se puede crear la Venta (*v*) y la colección vacía del equivalente en PdV a las líneas de venta: Viaje. Sin embargo, en este caso, la información distintiva de cada viaje es la información del viajero (necesaria para producir sus billetes) y lo que se '*vende*' es una lista de vuelos (común para todos los viajeros). Por tanto es mucho más eficiente y desacoplado crear una colección para la información de los viajeros y otra para la información de los vuelos.

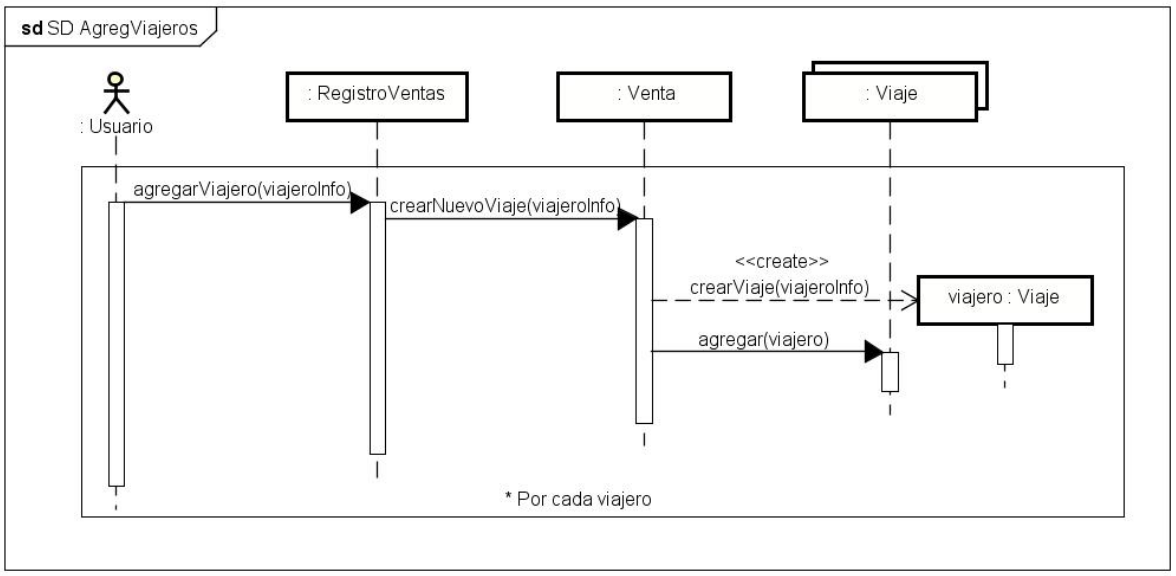
Más adelante, cuando se elabora la operación de la obtención de billetes, se cae en la cuenta de que éstos son los *artículos* que se obtienen en la compra y que quién los provee son las compañías aéreas. Por ello, si se piensa que la obtención de los billetes significa una conexión a la aerolínea y la transmisión de la información de sus vuelos, convendrá reordenarlos por compañías aéreas:

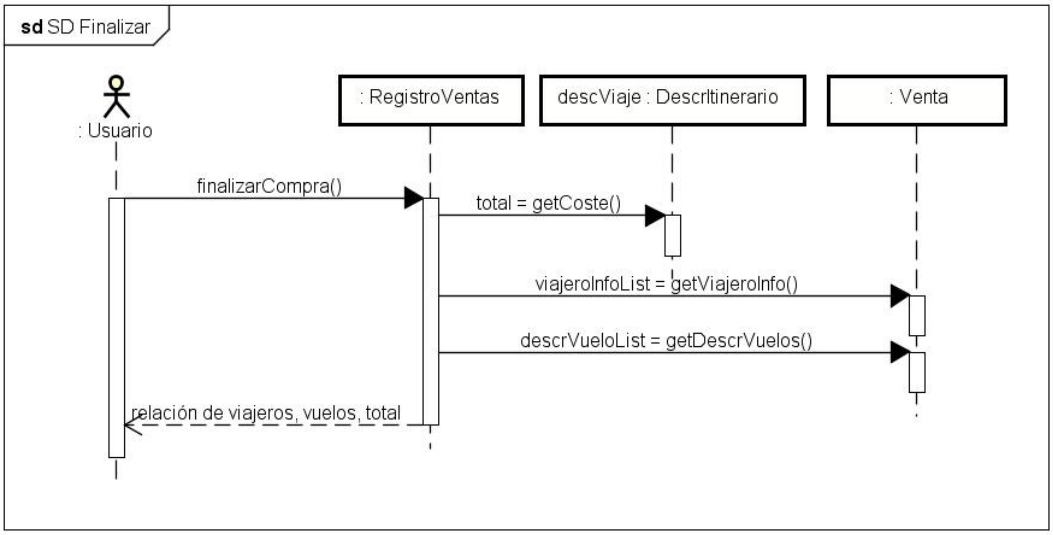


Nótese que la emisión de los billetes requiere la información del vuelo, no su ID de uso interno, por lo que es esa la información que se agrega en la lista de vuelos de la compañía.

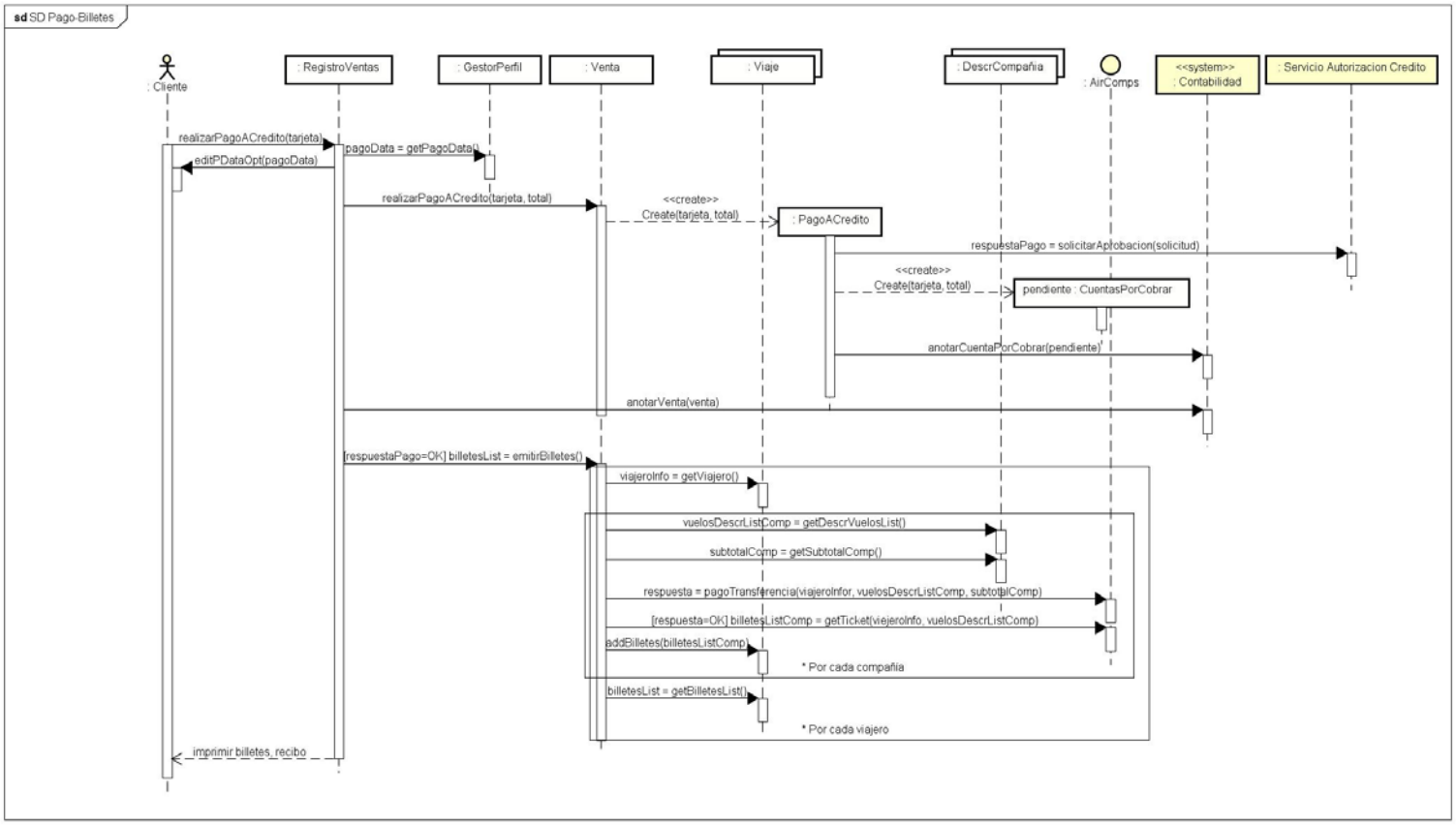
Para esta reordenación, se ha hecho uso de otro catálogo, el de compañías con las que trabaja la agencia de viajes. Habría que actualizar el modelo de dominio con estos nuevos objetos.

AgregarViajero:





RealizarPago e ImprimirBilletes:



Se puede mejorar mucho la eficiencia de la obtención de los billetes anidando los bucles de forma inversa: a cada compañía se le envía la información de todos los viajeros, la lista con la información de todos sus vuelos y se le paga esa cantidad de una vez.

La escritura de los contratos de las operaciones principales consiste en transcribir los DS anteriores **mediante una sintaxis específica** (de pseudocódigo) para las precondiciones y postcondiciones:

Precondiciones (situación previa a la ejecución de la operación):

- "Existencia de instancias y asociaciones".

Postcondiciones (situación posterior a la ejecución de la operación):

- "Creación de instancias (*nombre*) o asociaciones".
- "Modificación en el valor de los atributos o en el estado de los objetos".

A partir de los anteriores DS, con la ayuda de los contratos escritos para las operaciones, la obtención de los diagramas de colaboración correspondientes es inmediata (preguntas 5 y 6).

A partir de aquí, con un refinamiento previo del modelo de dominio, extrayendo de él los objetos implicados en el registro de la compra y obtención de los billetes, con sus relaciones, agregando a cada clase los métodos, obtenidos como mensajes en los diagramas anteriores, y estableciendo las asociaciones

adicionales que se deducen de la escritura de los contratos de las operaciones, se obtendría el diagrama de clases del diseño (DCD) pedido.

En general, los resultados así obtenidos podrían ser *aceptables*. Sin embargo hay determinados conceptos, fundamentales en esta asignatura, que no se han tratado con una profundidad *definitiva*. Debido a la persistencia de experiencias anteriores en los resultados, merece la pena tratarlos y reflejarlo en este ejemplo. Explicado de una manera *gruesa* y poco rigurosa:

1. El primer objetivo en el desarrollo de una aplicación es que funcione como lo plantea el cliente (satisfacción de la lógica del negocio solicitada por el cliente –en este caso, el enunciado—).
2. El objetivo de esta asignatura, supeditado al anterior, es que dicha aplicación sea, además, flexible y fácilmente mantenible. Es decir, que facilite los cambios en su comportamiento (algunos, los que puedan ser de interés para el negocio del cliente). Es evidente que el comportamiento de la aplicación está definido por el código y, a su vez, el diseño es la especificación para el funcionamiento del código; por lo que el objetivo se alcanza a través del diseño.
3. La principal manera de obtener flexibilidad y mantenibilidad es exigiendo **independencia funcional** en los diseños. Esto significa que las responsabilidades (funciones) que desempeña un elemento de un diseño sean independientes de las de otro. Un diseño también es una representación del trabajo colaborativo de sus elementos. Por tanto, aquí, el diseño consiste en asignar convenientemente roles de comportamiento organizativo (Controlador, Experto-Creador, Polimorfismo, etc… fachada, factoría, adaptador, observador, proveedor de servicios…) y responsabilidades (funciones) que se conjuguen correctamente con la ocultación (privacidad), el encapsulamiento, la cohesión alta, la generalización-especialización… y, sobre todo, el **acoplamiento bajo**; para conseguir la *colaboración independiente* de los elementos (objetos) de la aplicación.

Además hay otros conceptos importantes que, aunque se insiste en ellos de manera destacada en toda la asignatura, parecen ser ignorados persistentemente:

- **El Controlador**. Es un rol de comportamiento que consiste en aceptar las peticiones de acción, los eventos o estímulos y los dirige hacia el objeto capacitado para reaccionar con el comportamiento adecuado (respuesta) a la operación solicitada. Siempre debe haber uno *de cara* a la fuente de los estímulos principales (en principio, para atender al actor principal) del caso de uso. Puede no ser único, puede repartirse ese comportamiento con otros, o puede asumir, además, otros roles o funciones (por ejemplo, registro en PdV); pero debe existir desde el primer momento en el que se representa en funcionamiento del software (diseño).
- **El Registro**. Hay una gran cantidad de operaciones, realizadas con software, que representan una transacción. En general, una transacción

es una acción en la que se involucran al menos dos partes y en la que se intercambia algo entre ellas. La imagen más obvia es la de la transacción comercial; en la se intercambia un producto o servicio generalmente por dinero. También en la mayoría de los casos, hay un interés muy alto, de todas las partes implicadas, porque quede constancia fiable de lo que se intercambia y en qué circunstancias. La figura encargada de esto (dejar constancia fiable del intercambio) es el registro.

A la hora de modelar un escenario (modelo de dominio o de la lógica del negocio) puede haber innumerables situaciones (no sólo en las transacciones) en las que exista ese interés por dejar constancia de algún hecho; y debe existir el registro que lo haga o el objeto que haga esa función, se llame como se llame. Esta argumentación nos vuelve a dirigir hacia la reflexión más importante del análisis:

¿Cuál es la operación principal que define el caso de uso?

En el caso de PdV es una transacción comercial en la que se intercambian unos artículos por dinero.

¿Y cuál es en el caso de uso de e-Vuela? Parece ser, también, una transacción comercial en la que se intercambian viajes por dinero. ¿Seguro? ¿En qué consiste el objeto que se intercambia y cuáles son las circunstancias del cambio? Los viajes, en realidad, son vuelos en avión; representados por unos billetes. El Cliente obtiene sus billetes pero la Agencia de Viajes, a su vez, los compra a las Aerolíneas con las que trabaja y le cobra, al Cliente, una cuantía adicional por sus gestiones. Se trata de una transacción comercial con intermediario (la agencia de viajes), o una doble transacción, que varía notablemente el planteamiento del modelo del dominio y, por consiguiente, cómo se resuelve el diseño.

Lo anterior se podría enunciar como un corolario de algún principio del análisis:

> Al modelar un caso de uso definido principalmente por una transacción, es muy recomendable hallar los objetos finales del intercambio, las partes implicadas y las circunstancias en las que se produce.

Bromas aparte, si se da esta situación, como ocurre en un número muy elevado de casos, el registro (o un objeto que cumpla esas funciones) debe existir y reflejarse desde el primer momento en el que se representa la lógica del negocio (modelo de dominio).

- **El par Catálogo-InformaciónDeObjeto**. Se asume que la privacidad, ocultación y encapsulamiento de los objetos consiste en que sólo pueden manejar su propia información (son Expertos) y viceversa: la información de sus atributos es la estrictamente necesaria para que realicen las operaciones que se les encomienda. Por otro lado, se busca la cohesión y el acoplamiento bajo (independencia funcional) que consiste en que las operaciones que realiza un objeto sean independientes de las que realizan otros. Es decir, que no requieran la información de otro objeto. Pero, para

que haya colaboración entre las operaciones, es muy frecuente que éstas utilicen alguna información de otro objeto. La respuesta inmediata, **y errónea**, es incluir al objeto como componente. Si sólo es necesario un atributo del objeto ¿qué sentido tiene incluir todo el objeto? Especialmente cuando existen varias alternativas para usar un objeto u otro (listas, colecciones o multiobjetos), esto lleva a diseños monolíticos, inconexos y fuertemente acoplados en los que todos los objetos están *unos dentro de otros*, como las *muñecas rusas*.

La solución a esto puede ser el uso de un grado de indirección y el **catálogo** es un ejemplo, como **mecanismo de desacoplo**, similar. De ahí su gran importancia.

La idea consiste en utilizar una referencia al objeto, un identificador (articuloID), no todo él en la colección. Además, es necesario un mecanismo por el que, cuando sea necesario y a partir de esa referencia, dé acceso (Catálogo) a la información que, estrictamente, se requiera del objeto de la lista (EspecificacionDelProducto).

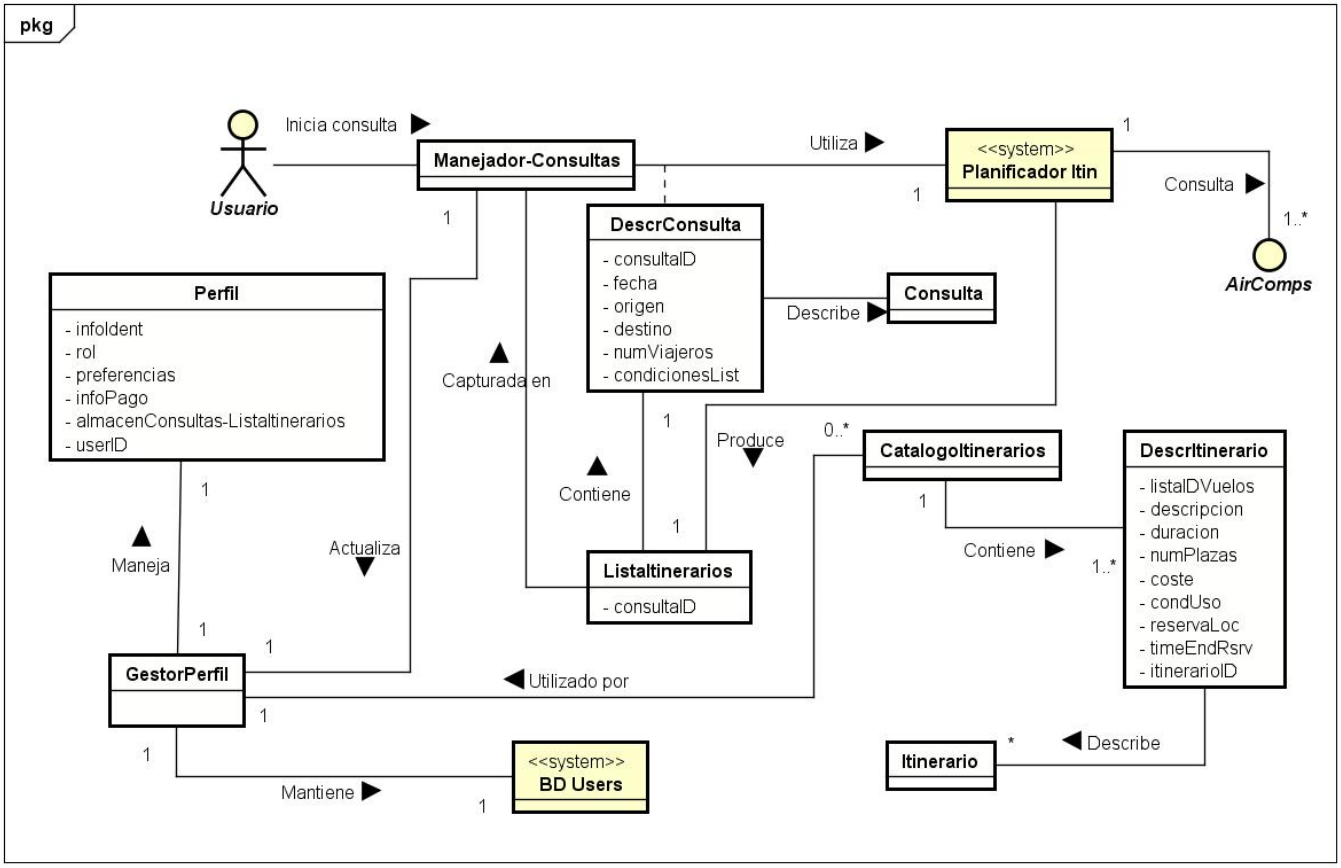
El catálogo es una especie de índice, de directorio, que contiene una colección de parejas {clave-de-búsqueda, información-requerida}. Hay que resaltar que 'información-requerida' se mantiene en otra clase (EspecificacionDelProducto) y no se corresponde con todo el contenido del objeto (Articulo), sólo con la que se va a usar en ese otro objeto.

En nuestro escenario, una consulta contiene los datos de búsqueda y un listado de los itinerarios encontrados. Pero en este listado no está la información de cada itinerario, sólo su identificador; al contrario que en la base de datos de usuarios, que sí estará, junto con otra abundante información adscrita al perfil del usuario. En el caso de uso, un itinerario contiene una lista de vuelos pero, en cada uno de los elementos de esa lista, no están los datos del vuelo (necesarios para emitir los billetes), sólo su identificador. Un vuelo tiene información sobre la aerolínea pero no toda la que se pueda usar en el caso de uso, sólo su identificador.

La excepción está en Billetes, cuya cardinalidad es numViajeros X numVuelos, y en el hecho de que cada aerolínea emite los billetes de **sus** vuelos. Para empezar, será conveniente agrupar los vuelos (comunes a todos los viajeros) por compañías aéreas. Además, para evitar la multiplicidad de todos los viajeros con todos los vuelos, sería más económico no desplegar el producto cartesiano y mantener separadas las dos listas (numViajeros **+** numVuelos). Como la información que necesitan las aerolíneas para emitir los billetes es, precisamente, la especificación de los elementos de cada una de esas listas ("información del viajero" y "datos del vuelo"), es el único caso en el que los mencionados elementos no serán los identificadores (*referencias a…*) de los objetos sino, directamente, su contenido.

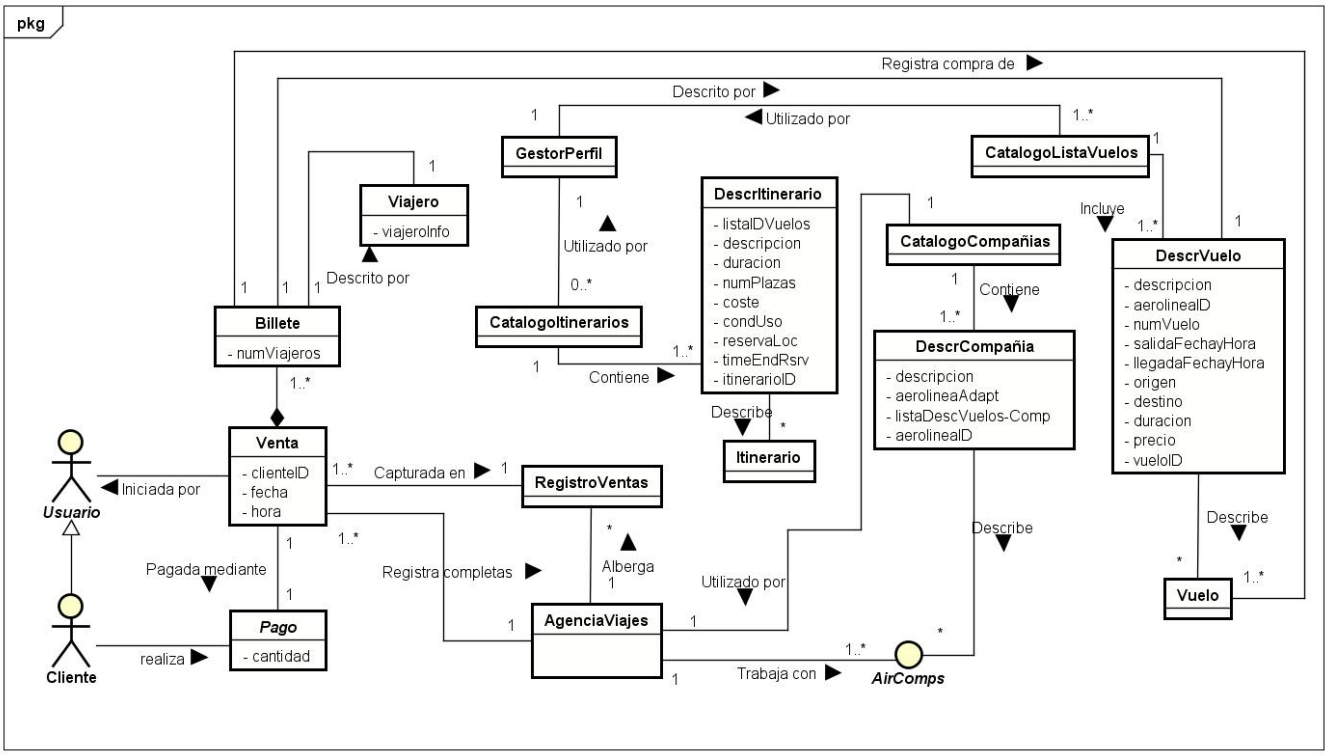
Con estas conclusiones, se vuelve a hacer un refinamiento desde el inicio del análisis del caso de uso.

La aproximación al modelo de la consulta quedaría:

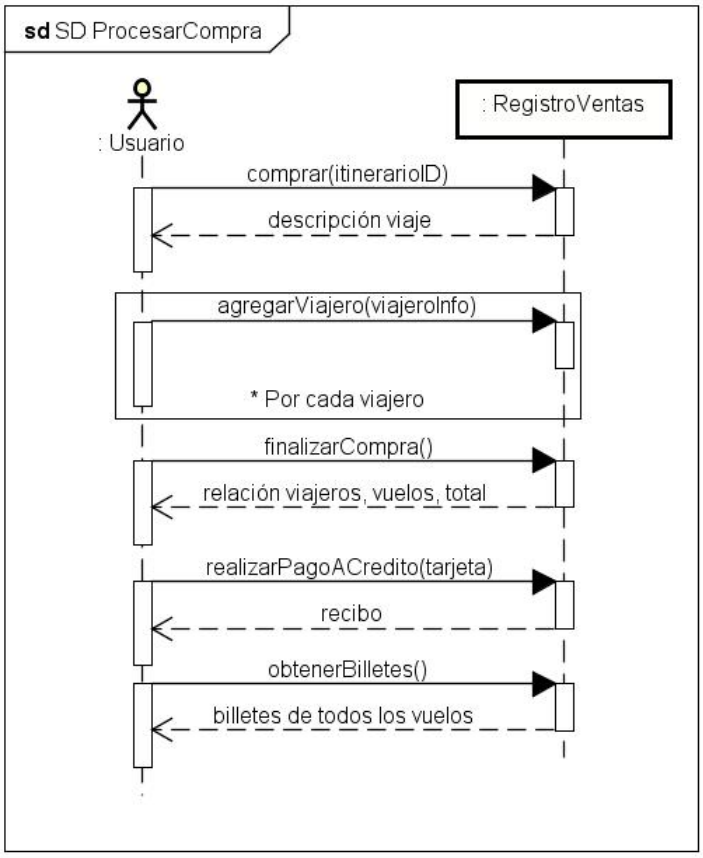


Además de reflejar la información que se maneja en un itinerario, se ha querido expresar la lógica del inicio del caso de uso. Cada consulta tiene asociada una lista de itinerarios cuya información se guarda, asociada al perfil del usuario, en la base de datos de usuario. GestorPerfil es el experto en esta información y quien tiene la responsabilidad de manejarla. Será el controlador principal de la sesión del usuario quien, al seleccionar éste un itinerario para su compra, invocará a GestorPerfil que, a su vez, consultará a la base de datos de usuarios (mediante un adaptador), extraerá la información DescrItinerario de los itinerarios de esa consulta y construirá el catálogo de itinerarios de esa consulta; con el que podrá obtener la información de ese itinerario. De igual forma construirá el catálogo con las especificaciones de todos los vuelos del mencionado itinerario. Tras esto, pasará el control al controlador del caso de uso (como se verá en el DS de la operación de inicio de la compra).

El modelo de dominio *de trabajo* inicial del caso de uso quedaría:

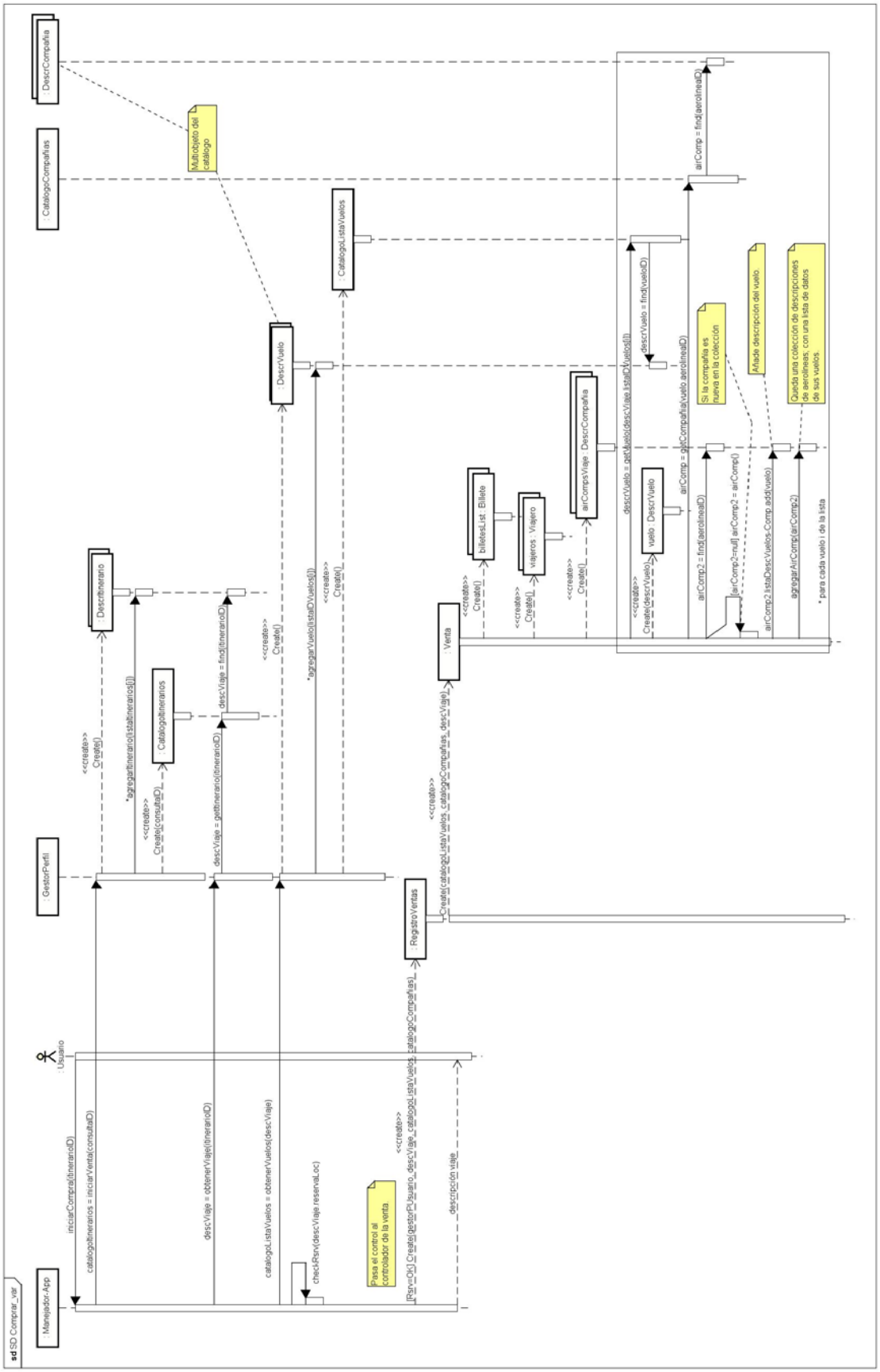


Para el DS global, se obvia la etapa de inicio y paso del control al registro de ventas:

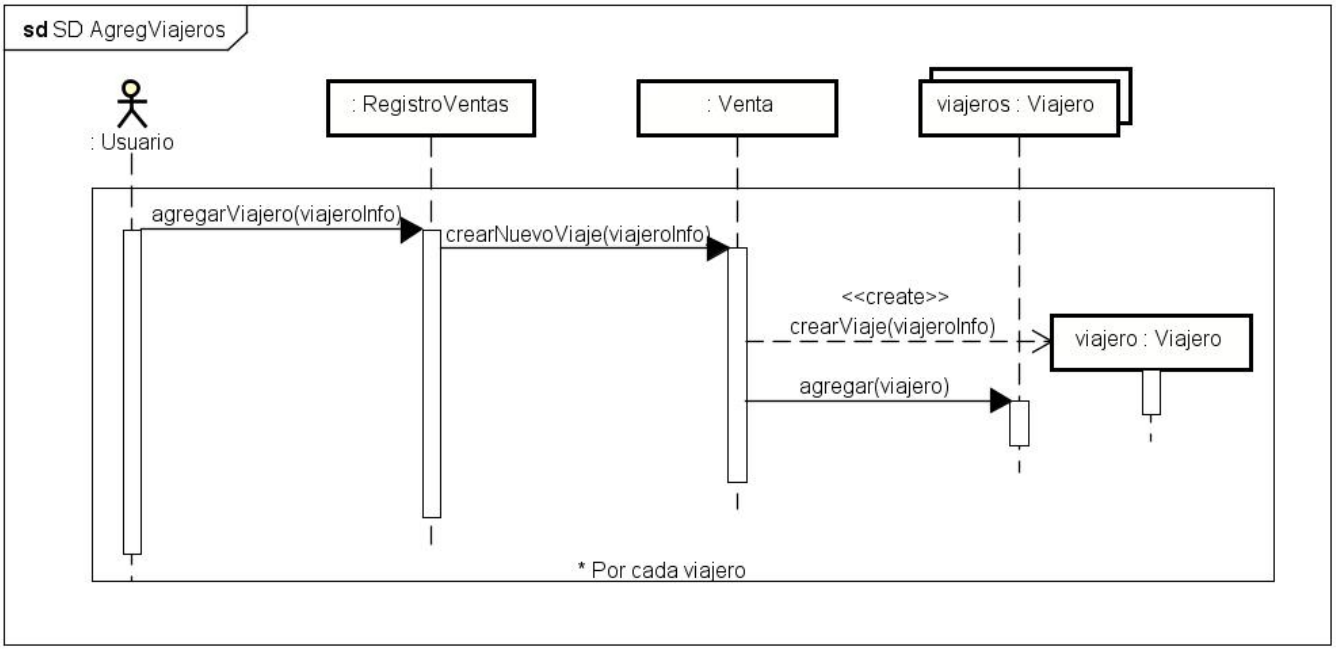


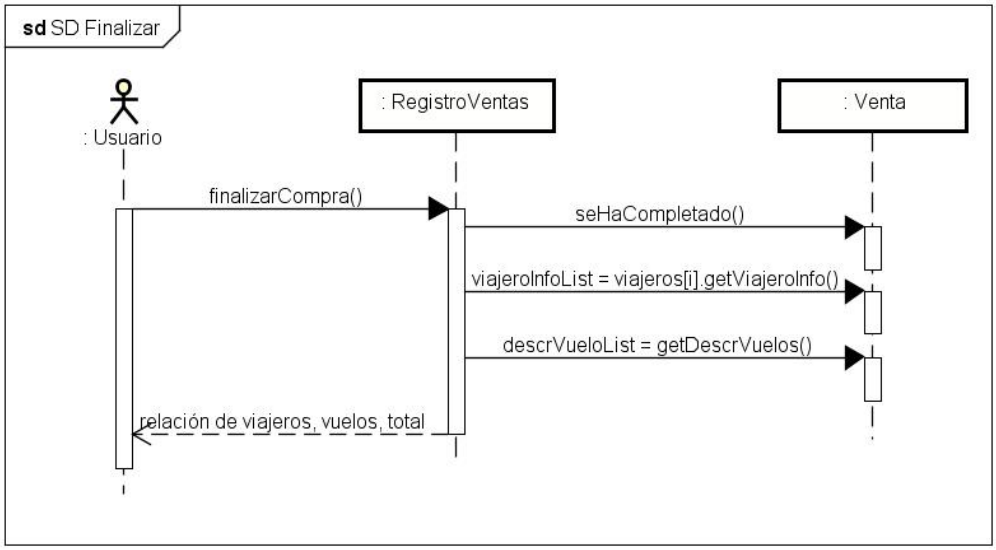
La descomposición en las operaciones principales produciría estos DS parciales:

IniciarCompra

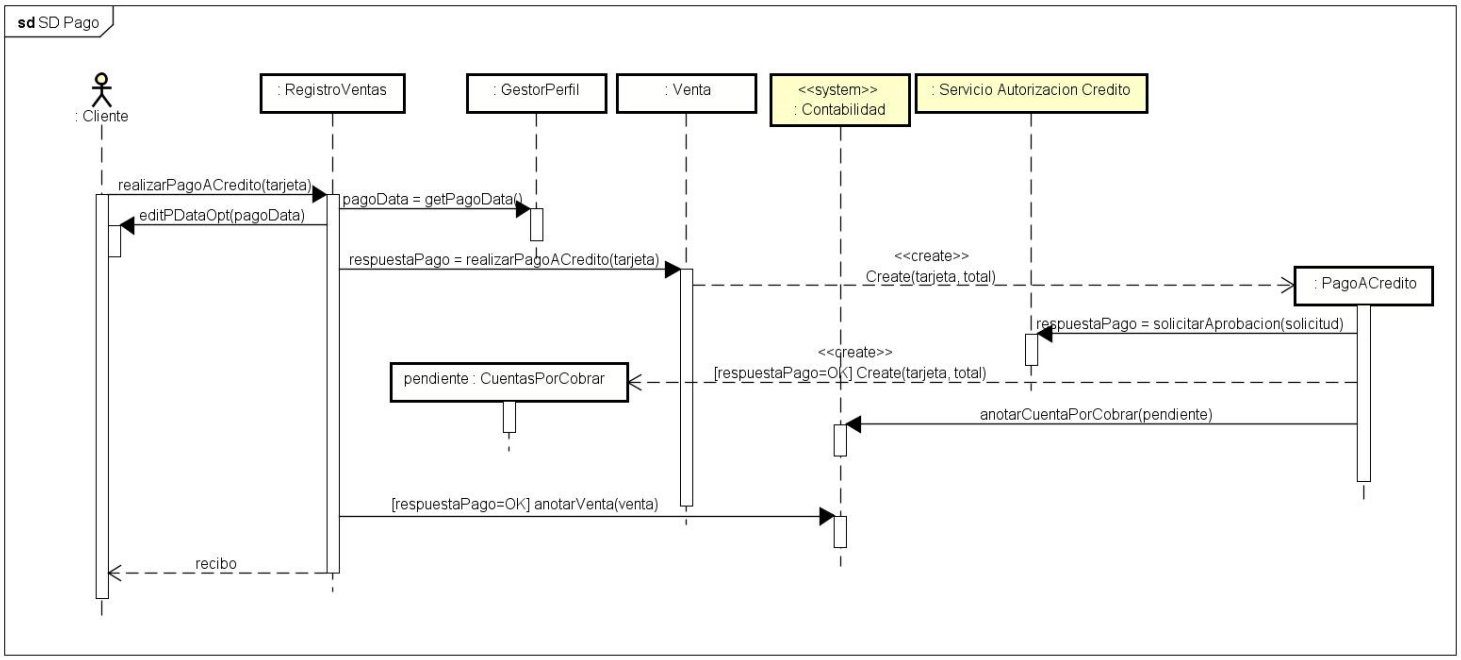


AgregarViajeros y Finalizar:

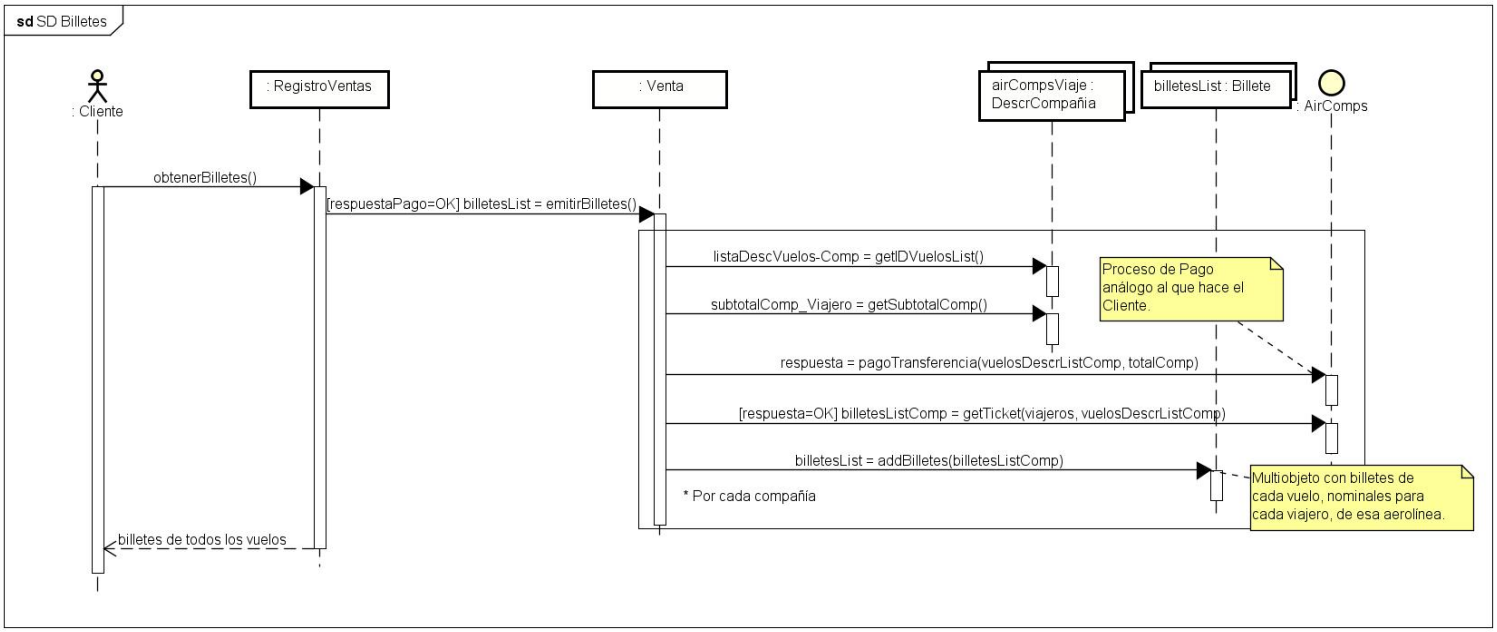




RealizarPago:

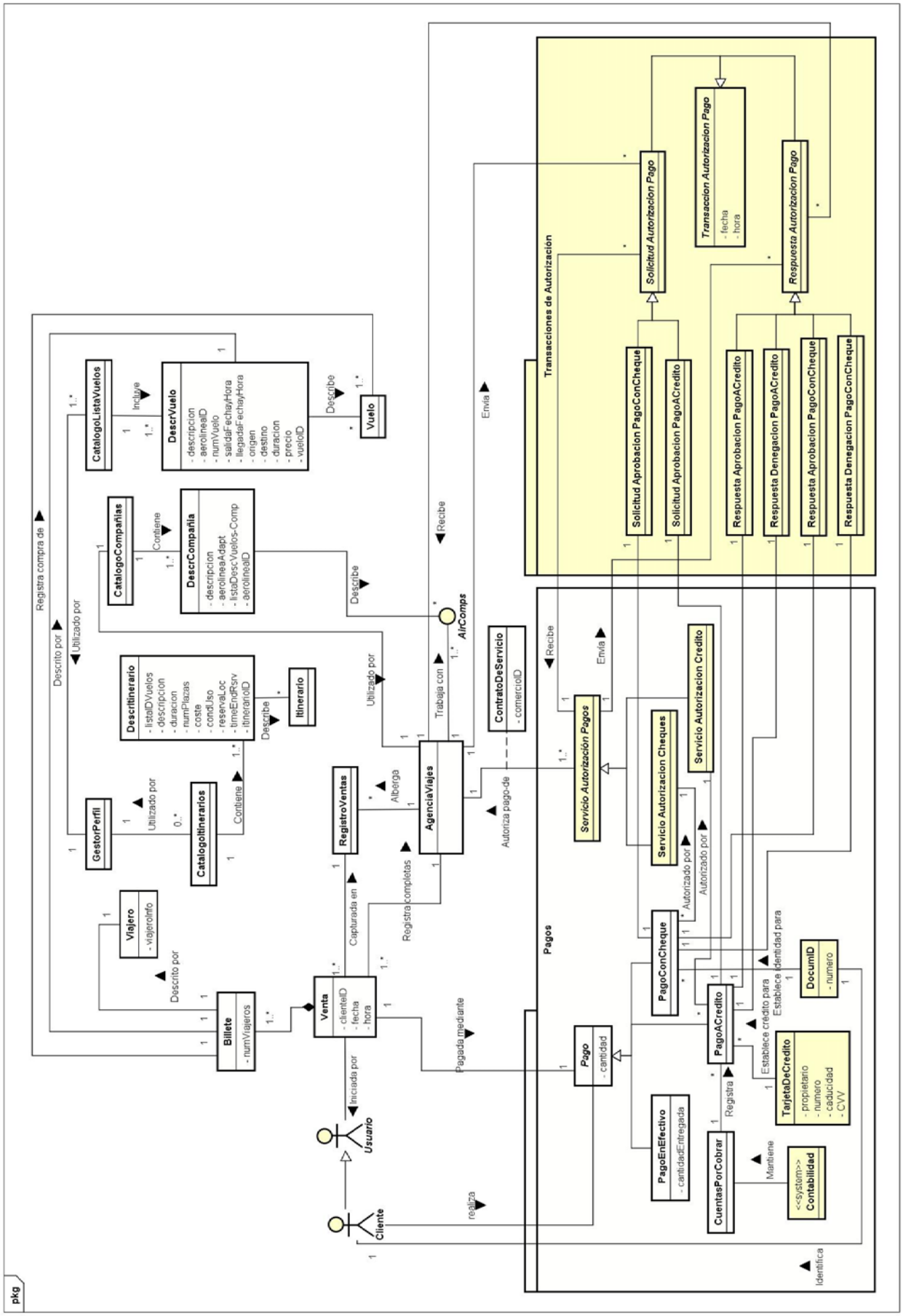


ObtenerBilletes:



En el último, se ha omitido la interacción para la comprobación de la firma en la tarjeta del cliente. También se ha obviado otro proceso de pago repetitivo, análogo al del cliente, pero que ocurre entre la agencia de viajes y cada una de las aerolíneas a las que se les contrata algún vuelo para este viaje.

Para entender la operación de pago, quizás convenga revisar el modelo de dominio en el que se despliegan los paquetes de *Pagos* y *Transacciones de Autorización*:


pkg
Transacciones de Autorizacion
Solicitud Autorizacion Pago
Transaccion Autorizacion Pago
- fecha
- hora
Respuesta Autorizacion Pago
Solicitud Aprobacion PagoConCheque
Solicitud Aprobacion PagoACredito
Respuesta Aprobacion PagoACredito
Respuesta Denegacion PagoACredito
Respuesta Aprobacion PagoConCheque
Respuesta Denegacion PagoConCheque
Pagos
Servicio Autorizacion Pagos
Servicio Autorizacion Cheques
Servicio Autorizacion Credito
Pago
- cantidad
PagoConCheque
PagoACredito
PagoEnEfectivo
- cantidadEntregada
TarjetaDeCredito
- propietario
- numero
- caducidad
- CVV
DocumID
- numero
CuentasPorCobrar
<<system>>
Contabilidad
Usuario
Cliente
Venta
- clienteID
- fecha
- hora
Billete
- numViajeros
Viajero
- viajeroInfo
GestorPerfil
CatalogoItinerarios
RegistroVentas
AgenciaViajes
AirComps
ContratoDeServicio
- comercioID
CatalogoListaVuelos
DescrVuelo
- descripcion
- aerolineaID
- numVuelo
- salidaFechayHora
- llegadaFechayHora
- origen
- destino
- duracion
- precio
- vueloID
Vuelo
CatalogoCompañias
DescrCompañia
- descripcion
- aerolineaAdapt
- listaDescVuelos-Comp
- aerolineaID
DescrItinerario
- listaIDVuelos
- descripcion
- duracion
- numPlazas
- coste
- condUso
- reservaLoc
- timeEndRsrv
- itinerarioID
Itinerario
Registra compra de
Descrito por
Utilizado por
Incluye
Describe
Contiene
Utilizado por
Contiene
Describe
Utilizado por
Trabaja con
Recibe
Envia
Alberga
Autoriza pago-de
Recibe
Envia
Capturada en
Registra completas
Iniciada por
Pagada mediante
Autorizado por
Establece crédito para
Establece identidad para
Registra
Mantiene
realiza
Identifica

A partir de este DS, especifique los contratos de **dos** de las siguientes operaciones: 'IniciarCompra' (inicio de la compra del itinerario reservado), '*AgregarViajero*' (agregar la información de cada viajero para poder emitir los billetes), 'RealizarPago' (pago con tarjeta de todo el viaje, con todos los billetes) o '*ImprimirBilletes*' (emisión e impresión de todos los billetes, de todos los vuelos y de todos los viajeros, tras comprobar el pago).

Contrato CO1: iniciarCompra

Operación:	comprar(itinerarioID)
Referencias cruzadas:	Caso de Uso: ProcesarCompra.
Precondiciones:	– Hay una sesión activa, **registroVentas**, con una instancia de **gestorPerfilUsuario**, **descViaje**, **catalogoListaVuelos** y **catalogoCompañias**. – Se ha comprobado la vigencia de la reserva en virtud del atributo reservaLoc de **descViaje**.
Postcondiciones:	– Se creó una instancia de Venta **v**. – **v** se asoció con **registroVentas**. – Se creó una colección vacía (multiobjeto) de Billetes, **billetesList** y se asoció con **v**. – Se creó una colección vacía (multiobjeto) de Viajero, **viajeros** y se asoció con **v**. – Se creó una colección de DescrCompañia, **airCompsViaje**, se asignó valor a su atributo **listaDescVuelos-Comp** en función de los valores obtenidos de **catalogoListaVuelos** y de **catalogoCompañias** y se asoció con **v**.

Contrato CO2: agregarViajeros y finalizar

Operación:	agregarViajero(viajeroInfo)
Referencias cruzadas:	Caso de Uso: ProcesarCompra.
Precondiciones:	– Hay una Venta, **v**, en curso. – Hay una colección vacía (multiobjeto) de Viajero, **viajeros**.
Postcondiciones:	– Se creó una instancia de Viajero, **viajero**. – viajero.**viajeroInfo** pasó a ser viajeroInfo. – **viajero** se asoció con **viajeros**. – **v.esCompleta** pasó a ser verdad**.**

Contrato CO3: realizarPago

Operación:	realizarPagoACredito(tarjeta)
Referencias cruzadas:	Caso de Uso: ProcesarCompra.
Precondiciones:	– Hay una Venta, **v**, en curso.
Postcondiciones:	– Se creó una instancia de PagoData, **pagoData** y se inicializaron sus atributos en función de los datos obtenidos por **gestroPerfilUruario**. – Se creó una instancia de PagoACredito, **pago**. – **pago** se asoció con la venta actual **v**. – Se creó una instancia de TarjetaDeCredito, **tarjeta**; y se inicializaron sus atributos con los valores de **pagoData**. – **tarjeta** se asoció con **pago**. – Se creó una SolicitudPagoACredito, **solicitud-pago**. – **pago** se asoció a **solicitud-pago**. – Se creó una instancia de CuentasPorCobrar, **pendiente**. – **pendiente** se asoció con el servicio externo de Contabilidad.

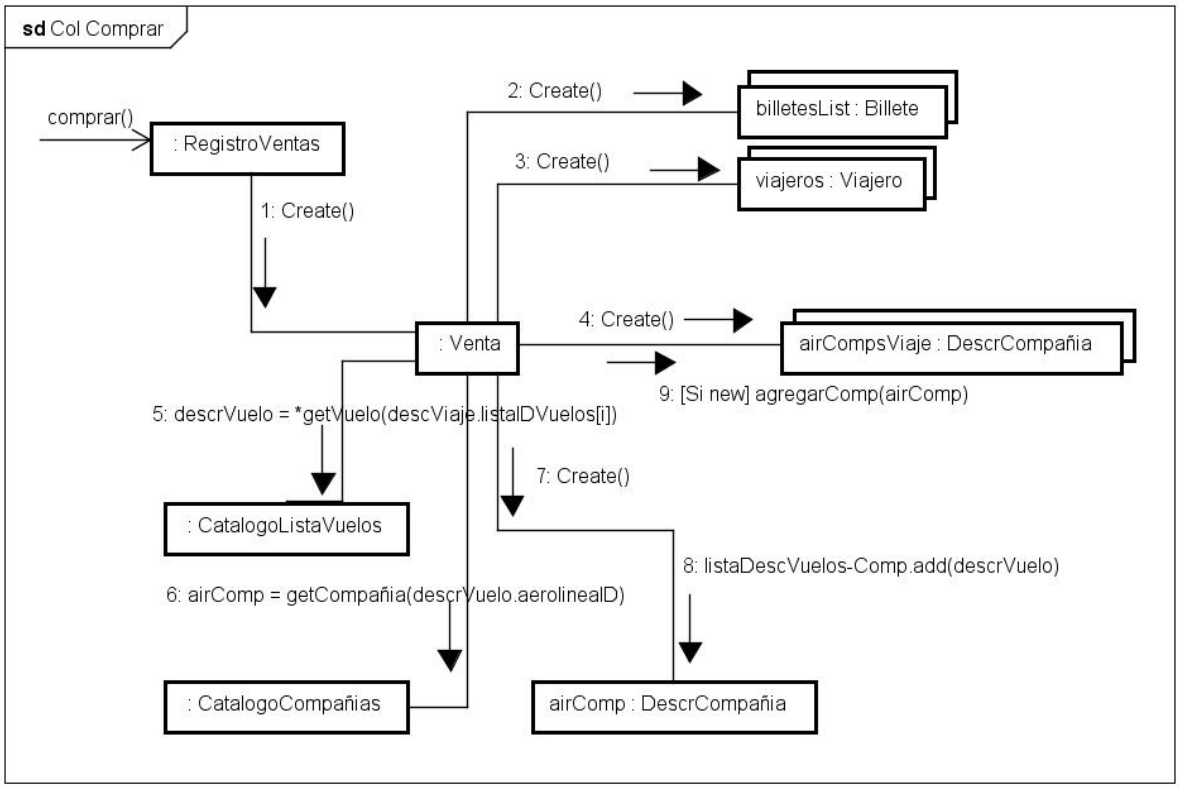
Contrato CO4: obtenerBilletes

Operación:	obtenerBilletes()
Referencias cruzadas:	Caso de Uso: ProcesarCompra.
Precondiciones:	– Hay una Venta, **v**, en curso.
Postcondiciones:	– Se creó una instancia de Pago, **pagoAirComp**. – **pagoAirComp** se asoció con la venta actual **v**. – Se creó una SolicitudPago, solicitud-pagoAirComp. – solicitud-pagoAirComp se asoció con pagoAirComp. – Se creó una instancia de CuentasPagadas, **pagada**. – **pagada** se asoció con el servicio externo de Contabilidad. – Se creó una instancia de Billete, **billete**. – **billete** se asoció con la venta actual, **v**. – Se creó una SolicitudEmision, **solicitud-billete**. – **solicitud-billete** se asoció a **billete**. – **v** se asoció con AgenciaViajes como una venta completa.

Sección 4. Evaluación de la ***Asignación de Responsabilidades*** y ***Diseño de Colaboraciones***

5. (2 puntos) A partir del contrato de la operación <<*se omite la operación A*>> que haya indicado en el punto 4, complete el diagrama de colaboración en UML. Consigne cada mensaje con los patrones GRASP (Experto, Creador, etc.) o cualquier otro que lo justifique. Si añade responsabilidades no explicitadas en el contrato (porque crea que es importante señalarlas), explíquelas brevemente.

IniciarCompra()



6. (2 puntos) A partir del contrato de la operación <<*se omite la operación B*>> que haya indicado en el punto 4, complete el diagrama de colaboración en UML. Consigne cada mensaje con los patrones GRASP (Experto, Creador, etc.) o cualquier otro que lo justifique. Si añade responsabilidades no explicitadas en el contrato (porque crea que es importante señalarlas), explíquelas brevemente.

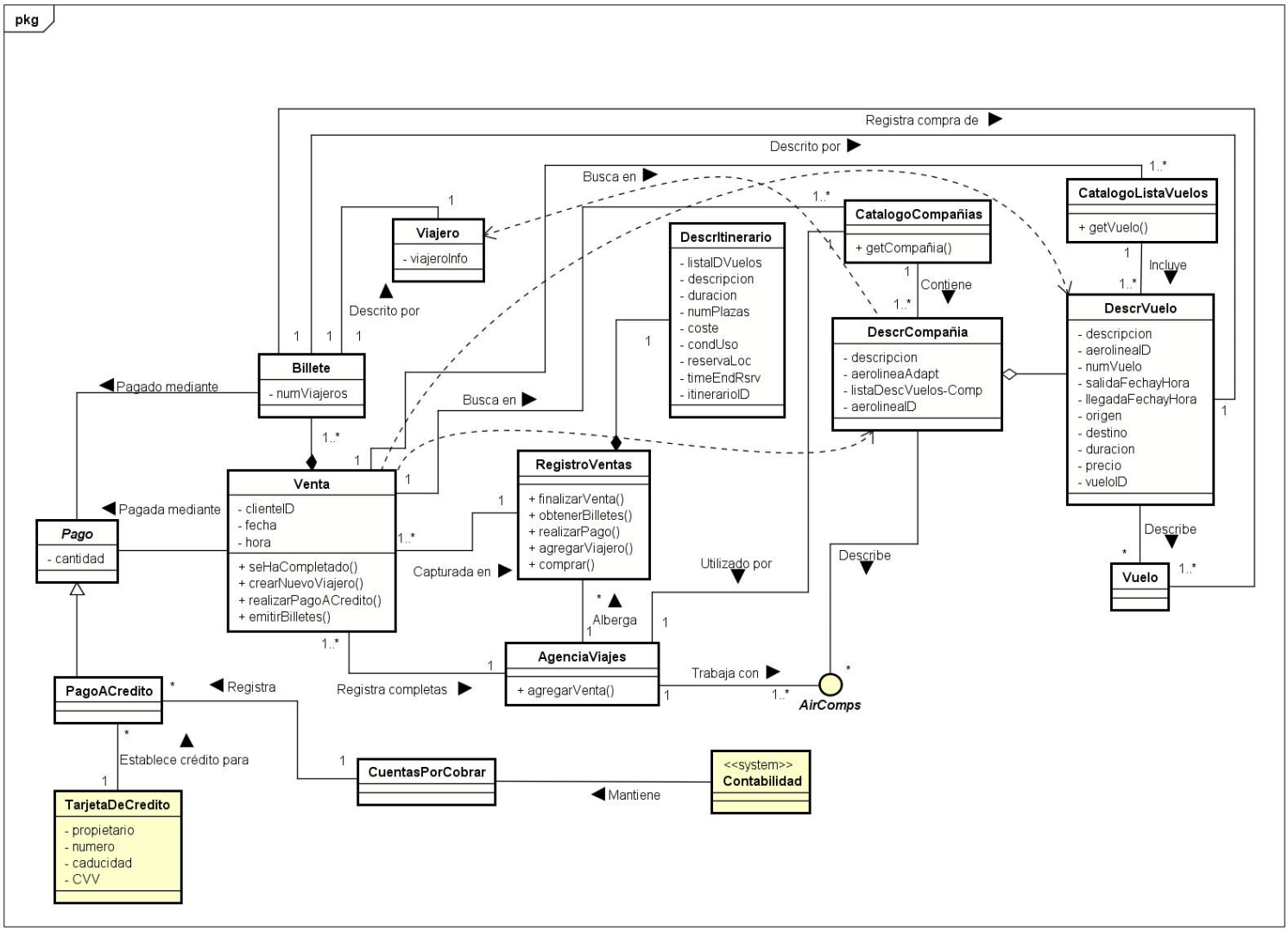
   De igual forma se haría para el resto. *agregarViajeros()* y *finalizar()*:

   realizarPago():

   obtenerBilletes():

Sección 5. Evaluación de los ***Diagramas de Clases*** de diseño

7. (0'5 puntos) Elabore un diagrama de clases para el caso de uso que se está tratando <<*ProcesarCompra*>> (DCD), centrado en la clase cuya responsabilidad es el registro de la compra de la reserva seleccionada y de la emisión de billetes, según se ha descrito en el caso de uso. Represente los nombres de todos sus atributos, asociaciones (con la navegabilidad) y métodos.



Sección 6. Evaluación de la ***Transformación del Diseño en Código***

8. (0’5 puntos) A partir de los anteriores diagramas de clases y colaboraciones, elabore y defina la clase que haya definido, en el desarrollo anterior, como responsable de la compra de la reserva seleccionada y de la emisión de billetes en el caso de uso <<*ProcesarCompra*>>. Incluya las definiciones de todas las variables que la componen (miembros), pero escriba solamente la definición completa del cuerpo para el método (o métodos) principal/es o más significativo/s: <<*se omite el método*>>. Ignore los pequeños detalles de sintaxis -el objetivo es evaluar la capacidad fundamental para transformar el diseño en código-. Utilice la sintaxis de Java.