RECOMENDACIONES PARA LAS EVALUACIONES DE LA ASIGNATURA

1. PRESENTACIÓN

El objeto de estudio en esta asignatura eleva el horizonte sobre el proceso global del desarrollo de software, trasciende sustancialmente *la programación* y sus requisitos estrictamente funcionales para incorporar otros, de índole cualitativo, con los que mejorar el producto final. Tras cursar, en años anteriores, varias asignaturas enfocadas directamente a la programación, se parte del supuesto de que el estudiante es capaz de construir una sencilla aplicación en java o algún otro lenguaje OO. Si es así, es muy importante que no olvide que ya tiene esa destreza.

En otras asignaturas, también anteriores, se le han mostrado algunos aspectos ingenieriles directamente aplicables a cómo hacer ese desarrollo del producto software (Ingeniería de Software y otras estrechamente relacionadas con la ingeniería), a cómo organizar las tareas y con qué criterios.

Esta asignatura consiste en ejercitar la práctica para aplicar aquellos aspectos ingenieriles a mejorar su eficacia en la codificación de las aplicaciones y su destreza en la programación para obtener mejores aplicaciones, con menor coste de desarrollo y mayores prestaciones en su explotación: código flexible, comprensible y funcionalmente independiente.

A la hora de desarrollar, desde el principio, un sistema software es necesario adoptar varios puntos de vista, ubicar y parcelar esos enfoques, profundizar en el conocimiento del comportamiento deseado, de las interacciones entre sus partes y sus implicaciones, tomar decisiones sobre qué simplificaciones se pueden abordar (con el conocimiento de qué repercusiones puede tener hacerlas) y afrontar, progresiva e iterativamente, la construcción de cada parte.

De la misma manera que un diseño es una especificación del funcionamiento de un componente software, de su estructura y características, tanto estáticas como funcionales, y cuya granularidad puede llegar hasta la estructura de datos o hasta la función; una arquitectura también es un diseño, pero se centra en los aspectos organizativos entre los componentes. Describe el comportamiento del sistema o subsistema estudiado, en función de cómo se organizan sus componentes y las interacciones que se producen entre ellos.

Así, asumiendo su capacidad para elaborar elementos de código que satisfacen los requisitos funcionales (esto ya es un diseño), **el objetivo de la asignatura es aprender a hacer que dicho código, además, sea funcionalmente independiente (elementos con acoplamiento débil y cohesión alta), flexible y comprensible**.

Juan del Rosal, 16
28040, Madrid

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

swdesign@issi.uned.es

Por un lado, es imprescindible que estas cualidades impregnen todos los aspectos y perspectivas del producto, desde la presentación y E/S hasta el manejo de la información persistente, desde la organización arquitectónica hasta el funcionamiento detallado de cada elemento del código. Por otro lado, la obtención de este objetivo obliga a replantearse cómo se obtienen los requisitos funcionales, cómo hay que manejarlos y qué consideraciones hay que tener en cuenta para construir el código.

Por las razones anteriores, **la evaluación de esta asignatura se centra en el diseño detallado. Es decir, en la especificación del comportamiento (hasta el nivel del código) de un componente o un grupo reducido de ellos; generalmente asociado a un caso de uso principal de la aplicación.**

2. DELIMITACIÓN DEL OBJETIVO DE ESTUDIO EN LA ASIGNATURA

Con la perspectiva del enfoque de la asignatura, presentado en el punto anterior, la elaboración de la evaluación se plantea de forma natural y coherente con la trayectoria que se sigue tanto en el programa como en el texto base: desde el ámbito global del sistema o subsistema estudiado, y su relación con el entorno, hasta el detalle codificado de cada operación y caso de uso.

2.1. DIAGRAMA DE CASOS DE USO

Así, los Diagramas de Casos de Uso son el resultado de una primera valoración general que permitirán aventurar cuáles son las funcionalidades más importantes para el sistema estudiado (los propósitos de su funcionamiento, los Casos de Uso Primarios), cuál es la delimitación de dicho sistema y qué tipo de relaciones e interacciones más notables se producen entre él y otros sistemas.

Aquí, lo principal es encontrar esa '*situación*' de la aplicación, en relación a los actores y los subsistemas de apoyo, y cuáles son los objetivos fundamentales de su uso por parte de los actores principales (Casos de Uso 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 https://www.uml-diagrams.org/).

2.2. EL CENTRO DE ATENCIÓN EN LA ASIGNATURA. ARQUITECTURA GENÉRICA DE UN CASO DE USO.

Nótese que, mientras en el libro o en el desarrollo del programa de la asignatura jamás se pierde esa perspectiva global, macroscópica o arquitectónica, del proceso de construcción (aunque se sumerja en el detalle de su codificación), en el caso de la evaluación, de los ejercicios, de la PEC o del trabajo final, existen fuertes restricciones temporales, de espacio documental e incluso de coste del propio desarrollo, que obligan a adoptar otra estrategia.

Por este motivo, en las pruebas de evaluación (una vez que se ha ubicado el objeto de estudio, mediante los Diagramas de Casos de Uso), toda la elaboración y el trabajo se centran en el análisis, diseño y codificación de **un único caso de uso primario**. Evidentemente, sin perder de vista sus interrelaciones con los otros casos de uso o con los otros componentes del sistema global. De ahí proviene la importancia de comprender, correctamente, *ese* caso de uso principal; en sí mismo y en su relación con los otros dentro del sistema global.

Partiendo de que el centro de atención está situado en ese caso de uso, vamos a construir una abstracción que consiste en un esquema genérico, mayoritariamente válido para cualquier sistema estudiado.

Hay una poderosa razón para hacer esto: si el objetivo es que todo el código sea funcionalmente independiente, la única manera es comenzar por identificar y aislar el *motor que mueve* cada funcionalidad, lo esencial del comportamiento que se quiere implementar (el código que constituye lo que denominamos el 'Negocio'), de todo lo demás.

Con ello, volvemos a alinearnos con el planteamiento del libro en las páginas 28 y 29. Es una estructura en 3 capas:

- **Capa de Presentación**. En general, es la capa donde se producen las interacciones con lo que se consideran *los actores externos principales* que manejan el sistema. Está compuesta por los elementos software encargados de adecuar y transformar la información de manera que sea útil o manejable a ambos lados de la capa. Por constituir una interfaz entre la lógica de la aplicación y los elementos externos a ella, dependiendo de cuál sea la delimitación que se establezca para el subsistema estudiado (la lógica del negocio mencionada o el caso de uso objetivo) y la naturaleza de los componentes externos con los que interacciona, puede ser de gran complejidad y algunos de sus elementos podrían confundirse con los de la capa de Servicios Técnicos. Otra de las simplificaciones de la asignatura es que el ámbito de esta capa se restringe a la **Interfaz de Usuario**. Es decir, aquellos elementos que encapsulan la interacción, estrictamente, entre la Lógica del Negocio y los Actores dotados de voluntad propia (elementos que no son software, ni hardware automatizado, cuyas reacciones y comportamiento no son predecibles o, si acaso, responden a objetivos de uso determinados por su propia voluntad). Se refiere a que esta capa implementa la interacción entre los Actores humanos (generalmente) y la propia implementación software de la Lógica del Negocio, del Caso de Uso.

- **Capa de la Lógica del Negocio**, **Lógica de la Aplicación** o **Dominio del Negocio**. Está constituida por los elementos software que implementan el comportamiento deseado para el caso de uso y que colaboran, necesariamente, tanto con la gobernanza del sistema global como con otros casos eventualmente. Es, precisamente, **el objetivo de trabajo en cualquier evaluación**: el análisis del comportamiento deseado, de las interacciones (a través de las otras 2 capas) con los elementos externos y que son necesarias para obtener ese comportamiento; y la construcción (diseño y codificación) de los objetos software que gestionan esas interacciones y que se comportan según los requisitos establecidos en el enunciado para ese caso de uso. En la Figura 1 sería el bloque azul del centro.

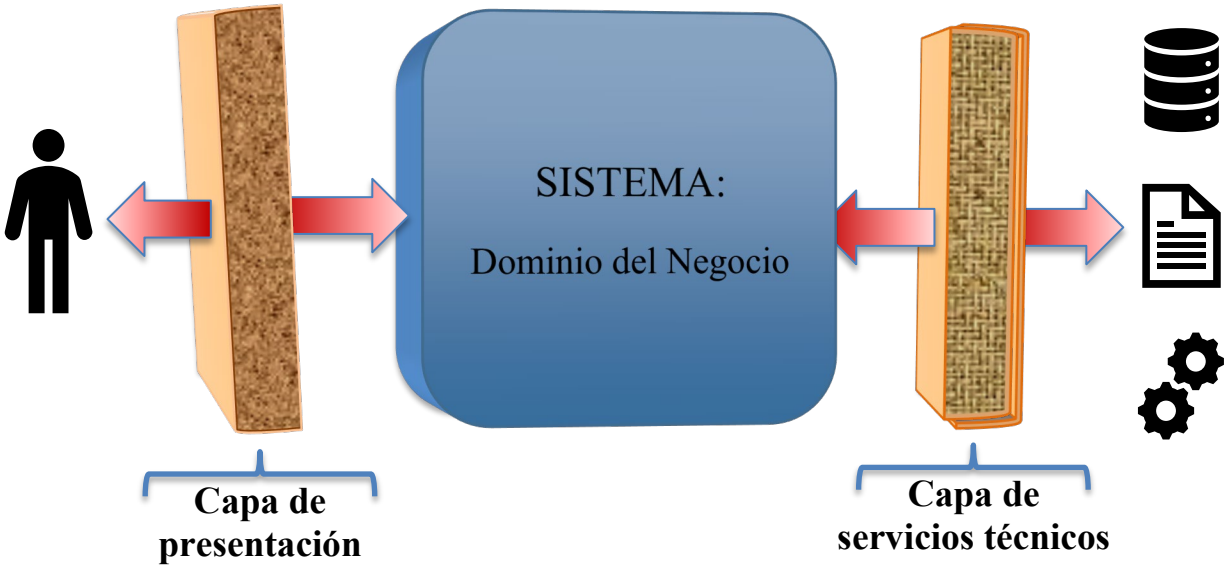


Figura 1. Estructura en capas del sistema software de estudio. El código del Caso de Uso se puede extender por todas las capas, pero, en la evaluación, sólo se va a implementar lo que corresponde al Dominio del Negocio, a lo esencial de su funcionalidad.

- **Capa de Servicios Técnicos**. Está formada por los elementos software que **no** constituyen, directamente, un componente *conceptual* representativo en el comportamiento de la lógica del negocio, sino que sirven de apoyo, en la consecución de dicho comportamiento, o actúan como mediadores para posibilitarlo.

  Frecuentemente, una funcionalidad descrita o implícita en la lógica del negocio no tiene una representación conceptual, en esa capa, que pueda albergar dicha responsabilidad, sino que requiere otros objetos conceptuales que provienen de algún servicio de apoyo técnico. Por ejemplo, en la lógica del caso de uso ProcesarVenta no hay ninguna indicación explícita de la necesidad para manejar globalmente los datos de la transacción Venta realizada (es decir, hacer persistentes esos datos en el Registro para poderlos manejar un componente declarado intencionadamente como externo, Contabilidad) y, sin embargo, es obvio que esa necesidad existe y es fundamental para éste (junto con el mecanismo de persistencia), para otros casos de uso, para otros componentes del sistema global y para la aplicación entera, en interés del negocio del cliente. Este objeto conceptual, con la responsabilidad de registrar persistentemente los datos de la Venta (o de actualizar el Inventario), proviene de esta capa de Servicios Técnicos y de apoyo; pero, por ser una parte fundamental en el comportamiento del caso de uso de una Venta, se analizará y se implementará (en los objetos software que sean necesarios) en la capa del Dominio del Negocio (como se muestra en el capítulo 30 del libro, en la página 422 y siguientes).

  En esa misma situación están algunos adaptadores, conectores polimórficos o interfaces que intercambian información y servicios con los actores y sistemas de apoyo; y, aunque provengan de esta capa, son elementos fundamentales para la creación o gestión de objetos o estructuras de información que resultan indispensables para la realización de la funcionalidad del caso de uso (consulta a BB.DD, creación de catálogos, etc.). Por este motivo se incluyen en el Dominio del Negocio.

Otra manera más técnica de representarlo podría ser la de la Figura 2.

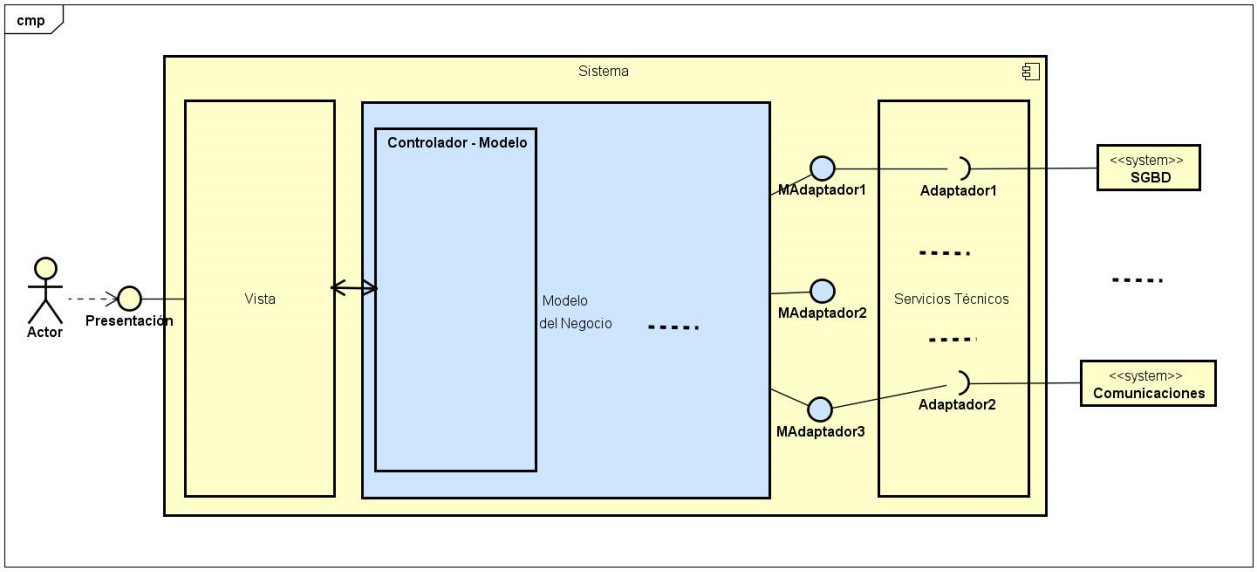


Figura 2. Otra representación de la arquitectura genérica del Caso de Uso.

Entiéndase que todo lo que se ha descrito, sobre una arquitectura genérica y universalmente válida para cualquier caso de uso, no tiene nada que ver con la arquitectura *efectiva* de la aplicación, con la manera en que se organizan sus componentes para colaborar y obtener así, conjuntamente, la funcionalidad y las prestaciones del sistema global; que puede ser muy variada. Es decir, esta estructura no es el estilo arquitectónico MVC (Modelo – Vista – Controlador).

De hecho, con un mínimo aumento de las funciones que realice, la organización de su comportamiento suele necesitar incluir, en la arquitectura, un componente o una capa arquitectónica (la capa de aplicación o de Gobierno del sistema) que gestione la manera en la que se accede o se posibilita cada caso de uso (por ejemplo: inicio de la aplicación, acceso de un usuario con un rol determinado, etc.). Como se verá a continuación, conviene tener esto muy en cuenta para delimitar el comportamiento del caso de uso sometido a estudio y para determinar qué procesos y qué elementos de información son antecedentes y apoyos necesarios para su inicio y su desarrollo; y de dónde provienen dichos antecedentes. Es decir: qué funcionamiento se debe haber producido en la aplicación y qué artefactos deben estar disponibles, con anterioridad a su inicio, para que el flujo de éxito, del caso de uso estudiado, sea posible.

> Por tanto, en la construcción del software que nos ocupa, en el análisis, diseño y codificación de este caso de uso, **sólo nos ocuparemos del comportamiento de *lo que debe estar dentro* de ese bloque azul**: de qué datos e información se necesita manejar y cómo se tiene que manipular para conseguir el resultado esperado, de qué interacciones se producen, de qué actor proviene cada estímulo y qué

3. LA ESCRITURA DEL CASO DE USO

La escritura del caso de uso consiste en la descripción, en lenguaje natural, de la secuencia de acontecimientos que se producen en el escenario de utilización de la aplicación que se plantea, desde su inicio hasta su finalización completa. El relato incluye tanto la secuencia principal de éxito (todo discurre directamente, sin inconvenientes y con las alternativas obvias, hacia el éxito en el cumplimiento del objetivo de uso), como los flujos alternativos que, previsiblemente, se puedan producir.

Aunque el relato sea en lenguaje natural, para que sea útil en el análisis y el posterior diseño, es necesario elaborarlo de una manera particular:

1. **Hay que determinar el inicio y la situación final de éxito en el caso de uso**. Aunque esta labor se refiere a un caso de uso primario, las limitaciones mencionadas para los trabajos de evaluación finales, o la PEC, frecuentemente inducen a hacer el desarrollo sobre *algo* que no es un caso de uso primario. Para reducir la complejidad o la extensión del desarrollo, a menudo, se enuncian escenarios de uso que, en realidad, deberían ser operaciones de un caso de uso primario. En este punto, no se pretende que se diferencie entre casos de uso primarios y los que no lo son (como en el diagrama de casos de uso); así que asúmalo, axiomáticamente, dedíquese a comprender qué inicia el caso de uso, qué

se pretende con esa utilización del sistema y cuándo se ha llevado a término (según se indica en el enunciado de la pregunta).

Por descontado, esa simplificación tendrá sus consecuencias, pero será en la elaboración del modelo de dominio y, posteriormente, en desarrollo del diseño. Forzar la delimitación del caso de uso seguramente exigirá la existencia previa de objetos y estructuras de información que se han obtenido mediante procesos que, aún, no se han tenido en cuenta.

2. El relato se plantea como una **secuencia de estímulo — respuesta**. Como su propio nombre indica, la perspectiva del relato es la del usuario; y lo que se pretende describir es qué tiene que hacer el usuario para obtener el resultado de su objetivo de uso.

   Cada **estímulo** se origina en el actor principal. El formato '*a dos columnas*' favorece la visualización contrapuesta del planteamiento. Pero, si no lo usa, en ningún caso puede eludir dicho planteamiento de «*estímulo del actor*» vs «*respuesta del sistema*» (la otra columna). De esta forma, el inicio del caso de uso es el primer estímulo del actor, el que desencadena toda la secuencia, y casi siempre debería enunciarse: «El caso de uso comienza cuando el 'actor principal' '*hace algo*' —el estímulo—…». Otra cuestión es cuándo se producen los estímulos: el actor generará un estímulo únicamente en el caso de que quede bien sentado que esa acción es la única vía para conseguir su objetivo.

   En cuanto a la **respuesta**, es la descripción de la reacción del sistema en correspondencia al estímulo recibido. Se trata, siempre, de sistemas causales: no puede haber reacción sin estímulo. En coherencia con el punto de vista del uso por parte del actor, la descripción de la reacción del sistema debería limitarse a lo que *le ofrece* al usuario para continuar el recorrido hacia la consecución de su objetivo. Pero estamos en un momento delicado del análisis en el que sabemos el resultado deseado (por el actor) pero no cómo llegar a él. Sin entrar en detalles técnicos sobre estructuras de datos o descomposiciones funcionales (el *cómo*, que decidiremos en el diseño), aquí se trata de meditar qué secuencia de operaciones debe realizar el sistema (y en qué orden) en aras de conseguir el resultado final o hasta que requiera otro estímulo, en un paso intermedio hacia él.

   El estilo esencial consiste en presentar la naturaleza de la información o de las acciones que se manejan en lugar de comprometerse con su contenido concreto o los detalles técnicos o tecnológicos de cómo se hace o se gestiona algo: «…El usuario solicita acceso al sistema y se identifica…» no «…El usuario introduce su tarjeta en el lector y el sistema consulta la base de datos con sus credenciales…». En este sentido, ni el relato de los estímulos ni el de las operaciones realizadas por el sistema, en respuesta a ellos, deben contener referencias a cómo lo hacen o a sus detalles técnicos; más allá de la secuencia de pasos u operaciones para hacerlo.

3. Los **flujos alternativos** son bifurcaciones en el relato de la línea principal de éxito cuando sucede alguna contingencia probable en ella. Son líneas completas y, por lo tanto, se deben enumerar todos los pasos: desde la

bifurcación y su condición, hasta su término o su reconexión con la línea principal de éxito.

4. EL MODELO DE DOMINIO

El modelo de dominio es la **descripción del funcionamiento del caso de uso**, derivado del relato desarrollado en el punto anterior.

Mientras que en la escritura del caso de uso se ha meditado y concluido qué operaciones son necesarias para alcanzar el objetivo del caso de uso, en qué secuencia temporal se deben producir y qué estímulos requieren por parte del/los actor/es, en esta representación se plantea el análisis de qué información se requiere para hacer esas operaciones y cómo se agrupa para que sea manejable.

El modelo de dominio **debe reflejar el funcionamiento** del caso de uso. Para ello, se deben construir unas clases de objetos (objetos *conceptuales*) que puedan realizar las acciones que se han descrito en el flujo principal de éxito elaborado, en la pregunta anterior, para este caso de uso. Es decir:

1. Que puedan recibir los estímulos generados por los actores.
2. Que puedan realizar las operaciones indicadas en la escritura del caso de uso.
3. Que, en el caso de que requiera algún dato o información adicional para realizar alguna acción correspondiente al rol que se le ha asignado, un objeto pueda obtenerlo de otro o a través de un actor de apoyo.

También se representan los actores principales y los estímulos (*eventos del sistema*) dirigidos a los objetos que, según la lógica del funcionamiento del caso de uso[1], se encargan de recibirlos.

Se trata de elaborar una representación con **objetos a los que se asigna un rol funcional**; pero no se hacen explícitas las funciones o los métodos que va a realizar cada uno (porque no se ha decidido aún, se hará en el diseño); sino que, pensando en la funcionalidad o rol que va a desempeñar, **se le asigna la información, datos o componentes** (atributos, que sí se representan) **que necesita para llevarla a cabo**. Por lo tanto, la tarea de elaboración del Modelo de Dominio va orientada a decidir qué objetos son los representantes candidatos para realizar las acciones que se han expresado en la escritura del caso de uso (su rol funcional) y qué información (los atributos) necesitan para hacerlo.

[1] Nótese que **no** se está refiriendo a qué objeto software va a ser el encargado de recibir y gestionar el evento (un `controlador`); si no al objeto conceptual al que, según el relato del caso de uso, va dirigido el estímulo. Por ejemplo, en PdV, el `Cliente` *`inicia`* una nueva `Venta`; que es lo obvio e inmediato según la escritura del caso de uso. Más adelante, en el diseño, se decidirá que el `controlador` (el que recibe los estímulos de los actores principales) será el mismo que el `Registro` (con el rol funcional de '*dejar constancia de la transacción*'), como estrategia para la inmediatez en la obtención de los datos que va a registrar. Por este motivo, en la implementación que se deriva del diseño, es el `Registro` el que recibe la orden de *`iniciar`* (*`crear`*) una nueva `Venta` por parte del `Cliente` – `Cajero`.

Un buen ejemplo de asignación de un rol funcional a un objeto conceptual es el del Registro:

- **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.

  Nótese, sin embargo, que el rol funcional del registro es coyuntural y está asociado a este caso de uso. El invariante que debe aparecer en cualquier caso de uso es el rol del **controlador**: objeto único (y específico del caso de uso) que gobierna y organiza todo el funcionamiento. En la especificación funcional (diseño detallado) el controlador también es el único interlocutor entre el actor principal y el resto de las instancias. En este ejemplo, al rol del controlador se le ha agregado la funcionalidad del registro.

  A la hora de modelar un escenario (el 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 un *registro* 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.

  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. Si se da esta situación, la de una transacción de la que interesa dejar constancia, 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).

Tras la decisión de qué objetos son necesarios, la reflexión sobre el balance y el equilibrio de las operaciones que realiza cada uno (y de los datos o atributos que requieren) nos puede llevar a derivar en otros objetos con los que se relacionan. En esta distribución, **siempre hay que tener presente que un objeto sólo puede manejar la información de sus atributos**; el resto no la conoce. Por ejemplo, en PdV, una Venta está formada por ítems vendidos (LineaDeVenta); cada uno de los cuales está definido por la cantidad del producto y su precio (algo fundamental que necesitamos conocer). La lógica del negocio nos indica que, en una venta, la transacción que hay que registrar es la lista de ítems vendidos (cantidad – ArticuloID – Precio) y el precio total. La reflexión sobre cómo se articula esta operación (la de obtener el precio de un producto) nos lleva a la conclusión de que habrá que consultar algún almacén

de información (posiblemente un SGBD, sistema—actor de apoyo externo) del que, a través de un adaptador (para el desacoplo), obtendremos únicamente la caracterización que se necesita para ese producto: el objeto EspecificacionDelProducto.

Otro ejemplo de deducción para un objeto conceptual, en función de la información que se necesita manejar y su procedencia, es el del Catálogo—InformaciónDeObjeto:

- **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, en la implementación del software, se busca la cohesión alta y el acoplamiento débil (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* (matrioshkas).

  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.

  Aquí se están manejando varias cuestiones:

  1. La operación que se está evaluando en el caso de uso maneja un subconjunto dentro de una colección de objetos (un subconjunto, seleccionable, de un multiobjeto). El mecanismo para acceder a la información objetivo de cada objeto seleccionado es el catálogo. En este caso, **en el modelo de dominio se debe modelar un mecanismo de acceso a cada uno de los elementos de información utilizados en esa operación del caso de uso**.
  2. La información objetivo de cada objeto, la que se va a manejar en la operación del caso de uso, ni es el objeto ni es toda la información del objeto que pueda existir en el almacén de información o en la base de datos. En el caso de PdV, es EspecificacionDelProducto, que consiste en articuloID, descripción y **precio**. La repercusión en el modelo de dominio es que se debe crear un objeto que represente la información específica que se está manejando en la operación del caso de uso y a la que se accede a través del Catálogo, único que maneja el multiobjeto contenido en él. La repercusión en el diseño será la elaboración de la especificación para construir esas clases, para construir sus métodos de manera que se comporten según el mecanismo diseñado. Además, si se requiere construir el Catálogo *dentro* del caso de uso, será tarea del diseñador interponer un

adaptador que, por un lado, *conozca* el contenido que se desea obtener para esas otras dos clases (Catálogo y multiobjeto InformaciónDeObjeto) y, por el otro, sea capaz de consultar al SGBD externo y obtener dicho contenido. Así se evita que el SGBD tenga la obligación de conocer la estructura de dichas 2 clases y se desacopla del caso de uso.

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

El catálogo es una especie de índice, un directorio, que contiene una colección de parejas {clave-de-búsqueda, información-requerida} y, por tanto, el multiobjeto. 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, de él, en la operación.

En definitiva, el modelo de dominio explica el funcionamiento del caso de uso mediante objetos a los que se les asigna un rol funcional y la información que será necesaria para realizar esa función; vinculados entre sí, mediante relaciones, en la medida en que expliquen dicho funcionamiento o el origen de la información que requiera y para representar las interacciones con los actores principales y los actores y sistemas de apoyo externo que sean necesarias.

En el modelo de dominio, las relaciones entre objetos conceptuales son bilaterales, suelen ser de composición y se refieren, casi exclusivamente, a la información que contienen ('contiene'), a sus componentes ('albergado en', 'capturado en') o a su procedencia ('describe'), nunca a acciones o a las operaciones que realizan los objetos ('usa', 'depende'). Es imprescindible expresar la cardinalidad de todas las relaciones.

Por último, una caracterización, por contraposición, del modelo de dominio; '*en negativo*':

- Los nombres de los objetos conceptuales **no** son verbos correspondientes a las acciones que realizan, si no que se refieren a la entidad de información sobre la que realizan las operaciones correspondientes al rol funcional que se les asigna. Es decir, generalmente, son sustantivos (Venta, Registro, Catálogo, etc.).
- Los nombres de las relaciones entre objetos **tampoco** son los verbos que se identificarán con las invocaciones a los servicios solicitados en el objeto relacionado, sino que se refieren a la interacción con el actor, a la ubicación de la información necesaria para efectuar la operación o el lugar donde se realiza (Inicia, Contiene, Registra-venta-de, Registra-completas, etc.). De igual forma, **tampoco** son los argumentos transmitidos en esa supuesta invocación de un servicio (*llamada a función*), **ni** los datos o información que comparten los objetos.
- El diagrama del modelo de dominio **no** es un diagrama Entidad – Relación (DER). El modelo de dominio no tiene la intención de representar las entidades de información ni las relaciones existentes entre esos datos, si

no es para explicar cómo se desenvuelven los acontecimientos en el caso de uso. Por razones similares, tampoco es un diagrama de flujo (DFD), de estados (DTE) o de clases de diseño (DCD).

- En coherencia con este diagrama, en las clases de los objetos conceptuales, **no se representan** sus métodos u operaciones, aunque sí sus atributos, para significar qué información y qué datos va a manejar, cada objeto, de forma privada. La especificación de estos atributos no necesita ser exhaustiva ni exacta, puesto que se pretende que dé una idea (ésta sí debe ser exacta) de cómo se procesa la información en este caso de uso. Por el mismo motivo, **tampoco** se indican, aquí, los tipos de esos atributos.

5. EL DIAGRAMA DE SECUENCIA Y LA ESCRITURA DE LOS CONTRATOS DE LAS OPERACIONES

Esta pregunta se inscribe en la «Sección 3. Evaluación de los ***Eventos del Caso de Uso***». Para hacer hincapié en la delimitación al caso del uso, no al sistema global, nótese que, a diferencia del libro, se ha rebautizado como Eventos del Caso de Uso, en lugar de Eventos del Sistema.

Antes de comenzar a asignar responsabilidades (diseño), se suele complementar la comprensión del comportamiento deseado para el sistema (ampliando la obtenida del Modelo de Dominio que se acaba de elaborar) con la evaluación de los Eventos del Sistema. (entiéndase aquí *Sistema* como el software correspondiente al modelo de dominio que se acaba de modelar para este caso de uso, la parte del *bloque azul* de la Figura 1 y Figura 2). 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, una *caja negra*), en cuanto a su salida (la reacción '*externa*' hacia el actor), cuando éste recibe un evento (estímulo).

Esto está muy bien, y sería suficiente, para ayudar a determinar cuáles son las operaciones importantes del caso de uso: normalmente, las que se inician por un estímulo del actor principal. Pero esta sección es la puerta de entrada hacia el diseño, la que nos emboca hacia la construcción de la especificación del código. Y hay un detalle *ejecutivo* que nos puede inducir una paradoja: la escritura de los contratos de las operaciones se hace con una sintaxis específica (se verá más adelante) mediante la que se define, unívocamente, qué instancias concretas se crean, cambian de estado, cuáles de sus atributos se modifican, qué asociaciones o dependencias se establecen entre las clases, etc. Es decir, una pre—especificación para el código; por lo que ya deberíamos haber pensado en él.

La paradoja se produce cuando se proviene de la secuencia en cascada, que disciplina para no pasar a la especificación del diseño hasta que estén completamente definidos los requisitos del análisis. Por el contrario, el ciclo de vida iterativo—evolutivo no sólo lleva a seleccionar los casos de uso más importantes de la aplicación y a obtener un prototipo temprano codificado; si no que, en cada caso de uso, también impone tomar, en el análisis, sólo los requisitos más importantes, elaborarlos parcialmente y, lo que aquí es más

importante: hacerlo con la vista puesta, simultáneamente, en el diseño y la codificación que se van a hacer. Esto corrobora que el planteamiento de toda la asignatura está dirigido a la programación.

De acuerdo, tanto el libro como este texto tienen una intención pedagógica y, hasta aquí, con las simplificaciones de restringir la atención a un único caso de uso (frecuentemente también simplificado e incompleto), a un subconjunto de sus requisitos más importantes, se ha ido indicando qué hacer en cada etapa. Pero se ha pasado por alto la *mirada estrábica* que impone el Proceso Unificado: **durante el análisis y la construcción del modelo de dominio, ya deberíamos estar pensando cómo se construyen esas clases, como van a ser sus métodos y cómo se van a utilizar sus atributos en las invocaciones a esos métodos**.

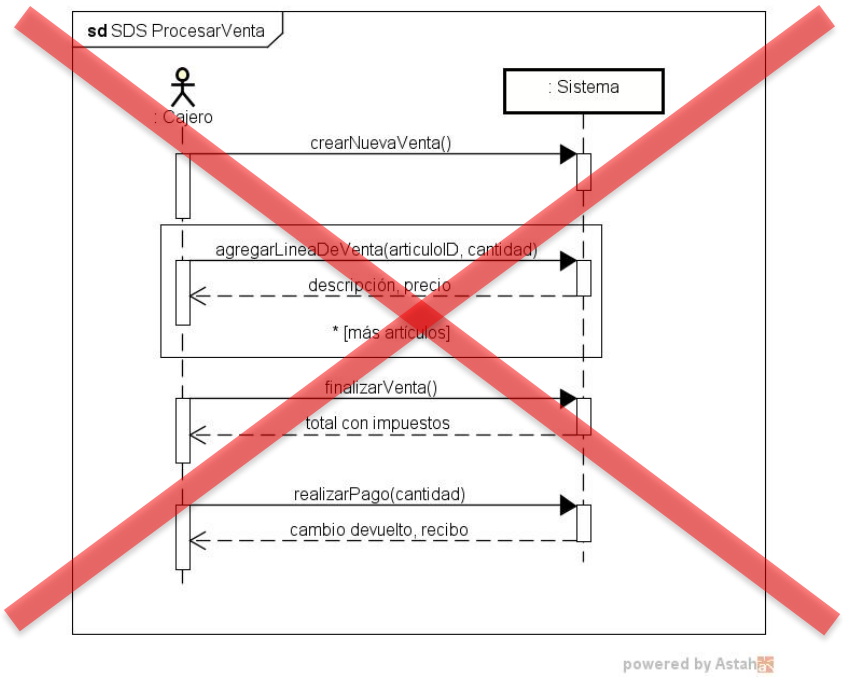
De haberlo hecho así, no debería haber ningún problema en dar el paso y aventurar una especificación del diseño (en una especie de pseudocódigo) al escribir los contratos de las operaciones.

Por otro lado, hacerlo así también requiere entrenamiento y disciplina para obtener unas destrezas que, en principio, no se pueden presuponer en el estudiante. Precisamente, los ejercicios prácticos, PEC, etc. son para obtenerlas. Además, en la secuencia de preguntas de las evaluaciones (PEC y exámenes), hasta esta cuestión, no queda constancia de ese trabajo *paralelo* en la elaboración del diseño ni cómo se ha estado pensando en la viabilidad y la coherencia del código que iba a producir; ambos fundamentales para que la especificación del diseño sea correcta y también coherente con todo lo demás.

Pues bien, si de lo que se está hablando no se ha hecho hasta ahora, es el momento de hacerlo; para que no haya dificultad en la escritura correcta de los contratos de las operaciones y esto facilite, enormemente, la elaboración de los diagramas de interacción, el DCD y la finalización con éxito en la codificación. ¿Es éste el punto crítico en la elaboración de los ejercicios?[2]

> Así que, en esta pregunta, **no se pone la interacción entre el actor principal y el sistema como un único bloque—objeto que represente el software del caso de uso**; si no que **hay que desglosar esa interacción, entre los distintos objetos del modelo de dominio**: cuál recibe los eventos (generalmente el controlador), cómo se desencadenan las acciones hacia los otros objetos del sistema, cómo se recogen sus respuestas y cómo se presentan los resultados o las indicaciones al actor para que genere un nuevo evento y se

[2] Personalmente he de confesar que *hago trampa*: tras escribir el caso de uso y esbozar el modelo de dominio, me vuelco en la elaboración, exhaustiva y detallada, del diagrama de secuencia; totalmente equivalente a los diagramas de colaboración y, en parte, al código. Es aquí donde articulo los estímulos con la creación de instancias, con el desglose de invocaciones a los métodos necesarios para reproducir el comportamiento deseado y con el paso de los parámetros correspondientes a la información que necesitan manejar. Tras comprobar el funcionamiento y su coherencia, los hallazgos de aquí los paso al diagrama del modelo de dominio.



5.1. DIAGRAMAS DE SECUENCIA

Una vez aclarado cómo no hay que hacer el DS en esta pregunta, se insiste en que ya hay que pensar en el diseño, en la especificación del código.

El diagrama de secuencia tiene una sintaxis gráfica específica con la que se intenta representar la secuencia temporal en la que se producen cada una de las acciones en las que se desglosa el caso de uso (estímulos, respuestas, reacciones, etc.), qué objeto inicia cada acción, qué información se maneja y cómo se transforma o manipula en cada una:

1. Los objetos (incluido el actor principal) se representan con el nombre de la clase software que simbolizan aunque, en realidad, son las instancias de los objetos que intervienen en el flujo principal del caso de uso. Para indicar esto, se suele subrayar el nombre de la clase y se puede detallar más añadiendo el nombre concreto de la instancia (lv:LineaDeVenta). Los nombres se inscriben en una caja (rectángulo) y, si se trata de un multiobjeto, el rectángulo se *desdobla*.
2. Todos los objetos (incluido el actor principal) tienen una línea temporal, vertical, que progresa desde la base del rectángulo que contiene la etiqueta de su denominación (arriba, en el punto de su creación) hacia abajo; y en la que se sitúa cada ocurrencia, tanto de los eventos o estímulos, como del resto de acciones que inicia cada objeto. Aunque cada acción tiene su origen en un punto concreto de la línea temporal del objeto que la inicia, se dirige a la línea temporal del objeto destinatario según una línea horizontal; es decir, se transmite en el mismo instante (excepto los auto-mensajes). Es muy importante mantener esta ortogonalidad porque, de lo contrario, se perdería la referencia temporal y el sentido secuencial. También **es imprescindible** indicar, con una flecha en la línea, el sentido en el que se produce la acción (que, en realidad, es una llamada a una función, perteneciente al objeto destinatario, desde el objeto origen).

3. Cada evento, estímulo, acción, reacción o respuesta de los objetos (excepto hacia el actor principal *humano*, en el caso de los dos últimos), aunque inicialmente se denominan "*mensajes*" en el libro, se van a resolver como **invocaciones a funciones, a métodos**; y así se van a representar aquí. Cada método es conocido sólo dentro de su clase; por lo que, si un objeto inicia una acción *hacia* otro objeto, está invocando a un método que *está en* el objeto destinatario y, por consiguiente, *necesita* tener a esa instancia destinataria como componente del objeto invocante. El control de tipos obliga a que ocurra lo mismo con los argumentos utilizados en la invocación de cada método.

   Todos los sistemas que se estudian aquí son causales. **Ningún objeto produce una reacción sin que exista una acción o estímulo sobre él**. Incluso los estímulos producidos por los actores '*humanos*' necesariamente deben originarse porque el sistema sólo ofrece esa alternativa para llegar al objetivo del actor. También en el inicio del caso de uso se reproduce esta situación de única opción, bien porque el control del sistema (a nivel de la aplicación) o bien porque un caso de uso o proceso antecesor lo ha llevado hasta allí.

   Este orden causal se representa en el diagrama situando la reacción del objeto en un tiempo posterior, en su línea de tiempo, a la llegada del estímulo.

4. En la mayoría de los casos los estímulos y eventos principales provienen de un actor humano. Como se ha explicado en el punto 2.2 y en la Figura 1, la interacción entre el actor y el sistema software (el dominio del negocio del caso de uso) se produce a través de la interfaz de usuario y, con este punto de vista, **lo que ocurra en ella (IU) es absolutamente irrelevante para este estudio** (la IU es *opaca* o totalmente transparente, como se quiera decir). Además, por el enfoque del estudio, centrado exclusivamente en la lógica del negocio, y otras razones, conviene que la IU sea '*thin layer*' (capa delgada); es decir: sin responsabilidades de la lógica del negocio y, por tanto, muy poco acoplada con él. Esto se traduce en que la IU se limita a formatear, representar y transmitir código de información *plana*, sin ningún valor semántico; el cuál se aporta en la interpretación del actor, por un lado, y en la responsabilidad de construir, por parte del software del caso de uso, una estructura de datos útil para la función que tiene que realizar.

   Así, independientemente de la significación para él/ella, cada selección, texto escrito, lectura en algún dispositivo, click en una imagen, etc., que realice el actor, llegará al lado del sistema **como una invocación** a algún método de un objeto (normalmente, el controlador), que **incorpora, en forma de argumento, la información transmitida**. Para evitar acoplamientos, dicha información tampoco puede tener una significación directa para la aplicación, representada por una estructura de datos – objeto que pueda utilizar inmediatamente; sino que debe responsabilizarse de obtener el objeto que necesita manejar a partir de ella. Suele ser un código (articuloID), un texto (nomViajero), una lista de ellos (listaVueloID, datosViajero), con el que deberá modificar el valor de un atributo o enviar una consulta a la base de datos, a través de un adaptador, para que éste cree un catálogo o consultarlo para obtener,

a partir del código recibido como clave de búsqueda, la información del objeto que necesita manejar o, rara vez, adecuar una lista de campos para inicializar los atributos en la creación de un objeto.

Por el contrario, aunque la aplicación envíe información en los argumentos de alguna llamada al adaptador de la IU, en respuesta o indicación de espera para el actor, éste no tiene capacidad para decodificar una función o sus argumentos. Esa información la formateará y la presentará la IU y será responsabilidad del actor su interpretación y, si es el caso y le resulta conveniente, contestar con alguna acción. Por ello, **las reacciones o respuestas hacia el actor**, o la presentación de resultados, **nunca se representan, como una llamada a función**; sino que **se indica un mensaje**, simplemente, **con la naturaleza de la información que se transmite** (cambio devuelto, recibo).

5. Lo mencionado hasta aquí está demostrando que, claramente, se están realizando tareas del diseño y, por tanto, se debería pensar cómo aplicar los Principios Generales de Asignación de Responsabilidades (GRASP). También parece evidente que el primer principio que se está teniendo en cuenta es el del **acoplamiento bajo**: es una cuestión prioritaria por ser fundamental para obtener la independencia funcional en el diseño-código; uno de los objetivos de la mejora en la construcción de software que se quieren obtener en esta asignatura. Como ya se ha comentado, debería tenerse en cuenta el código que se va a generar desde el principio del desarrollo y, en concreto, desde la elaboración del modelo de dominio, al asignar roles funcionales a los objetos o al decidir qué información maneja cada objeto, se deberían tener en cuenta esos principios GRASP (Experto, Controlador, Cohesión y Acoplamiento Bajo…). Como no se ha hecho allí, aquí, en el diseño, casi todo se está justificando desde el principio de acoplamiento bajo.

   De la misma manera que se está implementando el diseño considerando los principios GRASP mencionados, en este punto hay que tomar la decisión de elegir el objeto que va a regular la secuencia del funcionamiento en el caso de uso: el **controlador**. Como en los otros casos, esta decisión debería comenzarse en la elaboración del modelo de dominio; pero, en las evaluaciones, es aquí donde se refleja.

> **El Controlador**. Es un rol de comportamiento que consiste en **recoger las peticiones de acción, los eventos o estímulos del actor** (principal) y **los dirige hacia el objeto capacitado para reaccionar con el comportamiento adecuado** (respuesta) a la operación solicitada.

   - **Siempre** debe haber, **simultáneamente, sólo un controlador** *de cara* a la fuente de los estímulos principales (en principio, para atender al actor principal) del caso de uso. En un escenario de un sistema distribuido, basado en componentes ofertantes de servicios, sí sería posible que este rol se distribuyera, uniformemente, entre los componentes; pero, aun así, se requeriría un sistema de arbitraje del tipo escucha – registro - notificación. Este tipo de escenarios no va a aparecer en la evaluación porque lo

que se pretende, principalmente, es el dominio del concepto esencial del controlador.

- El controlador suele ser un objeto *singleton*, *de fachada* o *de sesión*, creado por un nivel jerárquico superior de supervisión (por ejemplo, el control en el nivel de aplicación o el de un caso de uso precedente) y que le participa los datos e información disponibles y necesarios para que se inicie el caso de uso. Aceptada la unicidad temporal del controlador, puede no ser el único a lo largo de la línea temporal del caso de uso. Para paliar la sobrecarga de responsabilidades (u otros motivos de conveniencia), puede repartirse ese comportamiento con otros objetos. Pero, entonces, debe transferir el control al nuevo objeto y no aceptar más eventos hasta que el controlador vigente devuelva dicho control o desaparezca (generalmente esto se realiza creando un nuevo objeto controlador y transmitiéndole la información que necesita para su tarea). Por el contrario, el controlador, además de atender a los eventos que recibe el sistema software, puede asumir otros roles o funciones (por ejemplo, Registro en PdV).

En resumen: el controlador **se encarga de recibir los eventos**, **debe existir desde el primer momento** en el que se representa en funcionamiento del caso de uso (diseño) y **no puede haber, simultáneamente, más de un objeto que atienda a esos eventos**.

6. Un caso especial en las acciones que debe hacer el software, para realizar el funcionamiento del caso de uso, es la creación de instancias. Desde luego, esa responsabilidad se asigna a un objeto en los términos que indica el libro para los principios GRASP de **Creador** y **Experto**. Pero, en el DS, se representa de una manera especial; mediante un mensaje «*Create*», originado en la línea de tiempo del objeto Creador y que, horizontalmente, da lugar a la aparición del nuevo objeto y de su línea de tiempo. Anteriormente a ese punto, el objeto o la instancia no existen. La creación únicamente exige la ejecución de su constructor, pero no es un estímulo para que haga nada más. Como cualquier objeto, para que la instancia creada tenga una reacción, necesita recibir una invocación desde otro objeto que la conozca.

En el libro de la asignatura se ilustran un buen número de mecanismos para operaciones comunes (búsqueda de un dato, modificación de un atributo, creación de instancias, construcción de multiobjetos, bucles, condiciones y *guardas*, valores de retorno, etc.) y cómo se utiliza esta sintaxis para representarlos. Para el lector, no debería suponer excesiva dificultad aplicar su bagaje de conocimiento algorítmico para elaborar el diseño del funcionamiento del caso de uso y expresarlo con esta sintaxis. Sin embargo, aunque no se sea especialmente rígido con la mencionada sintaxis de UML, **si no se siguen las pautas y el formalismo que se han indicado** en este epígrafe, **lo que se represente en el DS** (**y de aquí en adelante, con las otras representaciones**) **no tiene ninguna validez como especificación para el diseño o para la codificación**; porque es completamente ambiguo o carente de un significado unívoco y comúnmente aceptado. Ya que se trabaja, que sirva para algo.

5.2. LOS CONTRATOS DE LAS OPERACIONES Y SU ESCRITURA

Una operación es una agrupación de uno o más estímulos o eventos y las reacciones que produce en el sistema software. Es lo que se podría denominar, utilizando el término anacrónico, una abstracción funcional.

Al igual que con el caso de uso, primario o no, se podría alimentar la controversia sobre qué es una operación principal para el caso de uso y cuándo no es principal. Dentro de la evaluación, igual que entonces, es una tarea estéril. Relativizando para el caso de uso de estudio, y sus límites, una operación es principal cuando la agrupación *evento – sucesión de reacciones y respuestas que completan el objetivo de ese evento* producen la máxima cobertura en la descripción del funcionamiento del caso de uso. Entiéndase: *crearNuevaVenta()* no sería operación principal si tomamos la agrupación de reacciones de inicialización hasta el bucle *agregarLineaDeVenta(articuloID, cantidad)* (que sí sería principal, al igual que *finalizarVenta()* –porque calcula el coste, los impuestos, etc.— o *realizarPago(cantidad)* —porque articula los procedimientos de las distintas modalidades de pago—); como tampoco lo sería la agrupación que contiene todo el caso de uso.

En la evaluación, o se especifican las dos operaciones de estudio, o hay que seleccionarlas de forma que den una cobertura *casi* completa al funcionamiento del caso de uso; o, al menos, que cada una tenga una entidad apreciable en cuanto a la importancia del objetivo que la colaboración entre las acciones del software pretende alcanzar, dentro del funcionamiento del caso de uso.

La escritura de los contratos de las operaciones es *parecida* a la del flujo básico de éxito en el caso de uso. Tiene cuatro bloques: "denominación", "referencias cruzadas", "precondiciones" y "postcondiciones". Lo importante es que esta representación pretende formar una imagen exacta de la situación del software, antes y después de la operación. Es decir, describe la evolución del software, su funcionamiento, mediante la comparación entre una situación anterior y otra posterior. La utilidad de la descripción del funcionamiento de la operación depende de la exactitud con que se representen esas dos situaciones. De ahí, lo extremadamente importante que resulta utilizar un grado de formalismo adecuado para esas representaciones.

1. Denominación: nombre del evento que produce la operación. Como se ha indicado para el DS, será el nombre del método invocado, con todos sus argumentos, si los hubiera.
2. Referencias cruzadas: nombre del caso de uso al que pertenece y si tiene relación con casos de uso o procesos anteriores o posteriores. Si es necesario, porque afecta al funcionamiento de la operación, también se describirá la situación del software a la que ha conducido la ocurrencia de esos casos de uso o procesos anteriores; con la misma sintaxis que las precondiciones.
3. Precondiciones: se enumera la existencia de todas las instancias que se vayan a ver afectadas, de alguna manera, por lo que ocurra en la descripción de las postcondiciones.
4. Postcondiciones: en pretérito indefinido (*ocurrió*…), con los nombres específicos de los elementos software afectados y (a ser posible) en la secuencia temporal o lógica en que se producen, se describen todas las

variaciones que se han producido como consecuencia de ejecutar la operación:

- Creación de instancias.
- Modificación del valor de los atributos o su inicialización.
- Modificación del estado de los objetos.
- Creación o formación de asociaciones. Por ejemplo, porque se modifica el valor del atributo *tal* en virtud del contenido del objeto *cual* o del valor de su atributo $talcual$. O porque el objeto $talcual$ pasa a ser un componente del objeto A.

Se insiste en que, **si no se sigue este formalismo en la expresión, no tiene ninguna validez como especificación para el diseño o para la codificación.**

Como se ve, la escritura de los contratos de las operaciones (y la sintaxis que se emplea en ellas) obliga a tener notablemente avanzada la especificación del diseño y a haber tomado un buen número de decisiones previas en cuanto a la organización de los objetos y a las funcionalidades o responsabilidades que se asigna a cada uno. Esto es lo que ha impulsado que, en lugar de representar un DSS con los eventos que recibe el sistema global, se haya *recargado* esta pregunta con una elaboración bastante más detallada y formal del diseño. En contrapartida, con el trabajo realizado aquí, están casi resueltas, también, las dos preguntas siguientes: los diagramas de colaboración.

6. DIAGRAMAS DE COLABORACIÓN

Los diagramas de colaboración, aunque sean fragmentos por operaciones, expresan exactamente lo mismo que el diagrama de secuencia elaborado en 5.1 Por ello, si se ha simplificado el DS únicamente con los eventos que dan lugar a las operaciones, el controlador, sus reacciones más notables y los objetos que intervienen en ellas, es probable que se haya eludido gran parte de las reflexiones, conclusiones y decisiones a las que hemos llegado allí; y habrá que hacerlas ahora. Además de que los diagramas de colaboración son un desglose, por operaciones, del DS completo anterior, estos dos diagramas difieren, ligeramente, en su sintaxis gráfica.

1. En estos diagramas la representación es horizontal, centrándose en el intercambio de mensajes (invocaciones a métodos) entre los objetos que intervienen en la operación.
2. El diagrama parte del evento que inicia la operación y que llega al controlador desde el actor que lo genera.
3. Siempre que exista algún intercambio de mensajes entre dos objetos (llamada a función) se establece un vínculo entre ellos, representado por una línea.
4. Cuando un objeto realiza una llamada a un método de otro objeto, el texto de la invocación correspondiente (sintaxis de código) se sitúa en el espacio adyacente a la línea de vínculo que los une, **con una flecha orientada hacia el propietario del método**.

5. Los objetos y los multiobjetos, en sí, se representan igual que en el DS. En este diagrama, los objetos carecen de línea temporal. Esta información, **imprescindible**, se suple **numerando, con un índice, cada uno de los mensajes**; en el mismo orden que la secuencia temporal en la que se producen. Si alguna invocación da lugar a varias acciones, en las que se desglosa, también se indica el orden temporal, en el que se suceden, mediante un subíndice.

Con estos elementos y las mismas pautas utilizadas en el DS, se construye el diagrama de colaboración de una operación. Lógicamente, se podrían unir los diagramas de todas las operaciones y obtener uno global para el caso de uso, como se ha hecho en el DS; pero se amontonarían los mensajes en torno a las líneas de vínculo, haciéndolo bastante menos comprensible. Por el contrario, permite indicar con más facilidad los roles que desempeñan los objetos (principios GRASP de Controlador, Experto, Creador y otros de comportamiento; los demás en menor medida, porque se identifican peor o no están localizados en un objeto concreto).

7. DIAGRAMA DE CLASES DE DISEÑO

Poco queda por hacer. Durante la elaboración del DS (punto 5.1) y los diagramas de colaboración (punto 6), es **imprescindible modificar el modelo de dominio** con los hallazgos y las conclusiones **que afecten a la lógica de la aplicación**; **no así** con los objetos estrictamente software obtenidos (fabricación pura o auxiliares para la *mecánica* de la implementación) ni con las asociaciones encontradas ahí.

Es fundamental que exista coherencia entre todas las representaciones, desde la escritura del caso de uso, modelo de dominio, DS, los contratos de las operaciones, sus diagramas de colaboración, DCD, hasta el código final.

Generalmente se pide el DCD centrado en el objeto sobre el que se realiza la operación fundamental que define al caso de uso. Se construye así:

1. Desde el modelo de dominio, se transfiere a este diagrama ese objeto (por ejemplo, la Venta), todos los objetos involucrados en la operación, con sus atributos, las relaciones que existían en el modelo de dominio y sus cardinalidades. Se excluyen los objetos que no intervienen, los actores (principales o de apoyo), adaptadores e interfaces que actúen de conexión con el *exterior* de este *núcleo* del caso de uso o no intervengan en él.
2. Se añaden las clases software adicionales que se puedan haber encontrado en el desarrollo del DS, o los diagramas de colaboración, para las operaciones desarrolladas. Si es preciso, se ajustan las relaciones y sus cardinalidades.
3. Se incorporan los métodos, que implementan la funcionalidad del caso de uso, a cada una de las clases del diagrama. Se excluyen los *getters* y *setters* que no sean relevantes en el funcionamiento del caso de uso (que no aparezcan en el DS).

4. Se añaden las asociaciones obtenidas en la escritura de los contratos de las operaciones.

En este enlace se pueden encontrar unas anotaciones sobre el significado y la notación, en UML, de las diferentes relaciones entre objetos: "Relaciones en UML.pdf" (obtenido de Vaughn Vernon: "Understanding UML Class Relationships" y http://www.uml-diagrams.org).

8. ESCRITURA DEL CÓDIGO

También debería estar prácticamente resuelto. Todos los métodos significativos se han tenido que describir, ya, en el DS y en los diagramas de colaboración; con sus argumentos, y sus tipos, definidos y adecuados. Bastaría con trasladarlos aquí y darles un formato de definición se clase. Faltaría por decidir qué métodos son los más representativos de la ejecución del caso de uso; de acuerdo con todo lo anterior.

Por ejemplo, en PdV, la Venta se captura en Registro que, además, es el controlador de sesión. Por tanto nos interesa la clase **Registro** (**Controlador**), sus atributos fundamentales (CatalogoDeProductos y Venta), los métodos más significativos (*introducirArticulo (Articulo id, int cantidad)*, *realizarPago(Dinero cantidadPagada)* y *finalizarVenta()*) y, si acaso, la clase **Venta**, sus atributos (List lineasDeVenta , Date fecha, boolean *esCompleta()* y *Pago pago*) y alguno de sus métodos (*crearLineaDeVenta (EspecificacionDelProducto espec, int cantidad)*, y *realizarPago(Dinero cantidadPagada)*).