Construcción de Software
Ingeniero en Informática
Facultad de Informática
Universidad de Murcia

Modelado con UML del caso de estudio

"La Mega Subasta Pública"[1]

[1] El presente trabajo ha sido elaborado por Juan José Gálvez García y Pablo González de la Peña Albacete.

Tabla de contenidos

Introducción

Hemos aplicado un proceso software basado en UML a la especificación de requisitos de un servicio de subasta pública: La Mega Subasta Pública. El proceso está caracterizado por ser iterativo e incremental y basado en casos de uso. El proceso no se ha seguido enteramente, sino que nos hemos centrado en algunos aspectos de las fases de Modelado de Requisitos, Modelo de Análisis y Modelo del Diseño.

- Del Modelado de Requisitos se extrae el modelo de casos de uso y el modelo conceptual.
- En el Modelo de Análisis se obtiene un diseño preliminar del sistema a partir de los casos de uso. En esta fase obtenemos los diagramas de secuencia del sistema que nos ayudan a descubrir las operaciones básicas del mismo. Los contratos permiten especificar de forma detallada el efecto de estas operaciones y las colaboraciones muestran el comportamiento de las mismas, como interacción de objetos. En esta fase se obtiene también el Diagrama de Clases del Análisis.
- En el diseño se obtiene el Diagrama de Clases del Diseño, se define la arquitectura del sistema y se contempla el uso de patrones, entre otras cosas. En esta fase nos limitamos a comentar la ventaja que se podría obtener de usar ciertos patrones de diseño y mostramos un primer listado de la implementación del sistema (código fuente en Java) obtenido directamente de las colaboraciones.

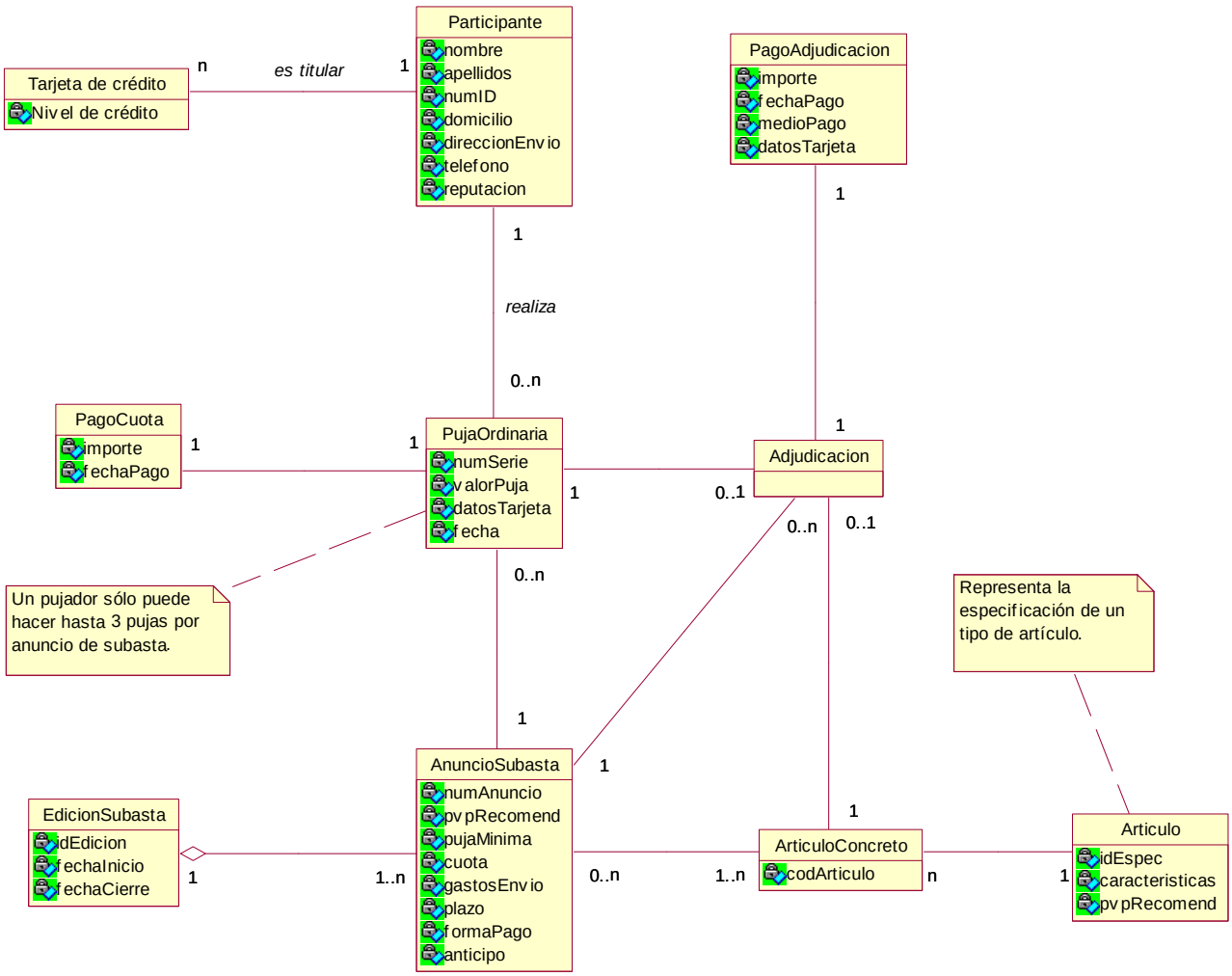
La estructura del documento es la siguiente:

Primero se muestra el diagrama Conceptual y el de Casos de Uso. A continuación, de cada CdU se deriva un apartado que contiene:

- Especificación textual del CdU.
- Diagrama de secuencia del sistema del CdU.
- Contratos junto con los diagramas de colaboración de los mismos. En todos ellos se ha intentado aplicar patrones GRASP.

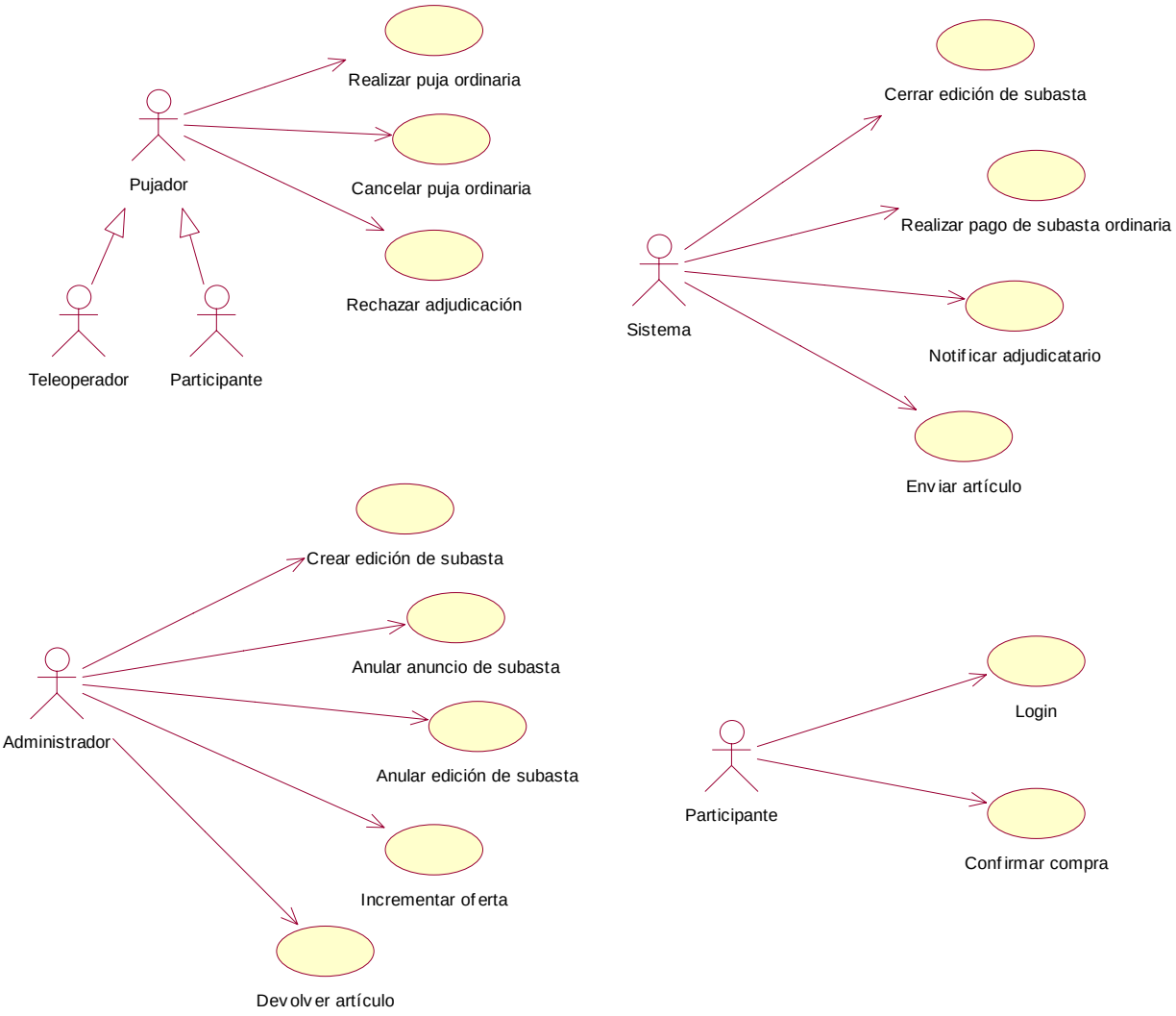
Por último se muestra el diagrama de Clases, comentario acerca del uso de patrones de diseño y el listado de código fuente.

Diagrama Conceptual



Nota del profesor. Observa que se podría haber introducido un concepto "Pago" que generalizara "PagoCuota" y "PagoAdjudicación", así como un concepto "Administrador" que representara el actor que aparece en el modelo de casos de uso y que entre otras cosas crea una edición de subastas o un anuncio de subasta. También se podría introducir un concepto "Banco" o "GestoraDeMediosDePago", que se encargaría de tramitar los pagos con tarjeta. Incluso se podría haber establecido una generalización entre "Tarjeta de crédito" y sus especializaciones: VISA, MasterCard, 4B y Tarjeta Punto Oro, con el objetivo de dotar de más información al modelo.

Diagrama de Casos de Uso



Nota del profesor. Podemos acotar la práctica centrando el trabajo en los siguientes casos de uso: Realizar puja ordinaria, Cancelar puja ordinaria, Cerrar edición de subasta, Realizar pago de subasta ordinaria y Crear edición de subasta.

Aspectos a tener en cuenta al leer los contratos y colaboraciones

De la fase de análisis hemos obtenido un diagrama de clases que varía ligeramente del diagrama conceptual. Este diagrama de clases ha surgido paralelamente a la escritura de las colaboraciones. Por ello, al leer los contratos y colaboraciones pueden aparecer clases que no estén en el diagrama conceptual, en concreto controladores, catálogos y nuevas clases relacionadas con el pago. Por ello, lo mejor es acudir al diagrama de clases para ver las diferencias.

Respecto al pago, hemos visto conveniente considerar que la clase Pago contiene un número variable de transferencias. Pago y Transferencia son clases abstractas. Las subclases de Pago definen un nuevo tipo de pago (por ejemplo PagoCuota) que incluye a su vez algún número de transferencias de cualquier tipo que sea conveniente implementar en el sistema (por ejemplo una transferencia que transfiera dinero de una tarjeta de cliente a una cuenta de la Mega Subasta). Esto facilita introducir nuevas formas de pago y de transferencias en el sistema. El pago lo realiza una clase llamada GestorPago que recibe un objeto Pago y lleva a cabo todas las transferencias especificadas en el mismo. Esto facilita diseñar la colaboración ya que GestorPagos tiene un único método que recibe un objeto Pago.

Use Case UC1: Crear edición de subasta

Stakeholders:

- Administrador: Desea que la lectura de datos sea correcta.
- Proveedor: Desea que el anuncio refleje fielmente la información proporcionada por él.
- Participante: Desea que la descripción del artículo se ajuste a la realidad, así como la fotografía. Que los datos mostrados en el anuncio sean correctos.

Actor: Administrador
Precondiciones: El Administrador está identificado y autenticado en el sistema.
Postcondiciones: Se creó una nueva edición de subasta con un conjunto de anuncios de subasta.
Ver contratos para más detalles.

Escenario Principal (o Flujo Básico):

1. El Administrador quiere crear una edición de subasta.
2. El Administrador introduce la fecha de inicio y cierre de la edición.
3. El Sistema registra la nueva edición, le asigna un número de edición y solicita la introducción de nuevos anuncios de subasta en la misma.
   Para cada subasta que el Administrador desea crear se realizan los pasos 4-12:
4. El Administrador crea una nueva subasta ordinaria.
5. El Sistema solicita al Administrador el artículo a subastar y presenta una lista de artículos disponibles.
6. El Administrador elige el artículo a subastar y el número de artículos.
7. El Administrador introduce (en cualquier orden) el valor de la puja mínima, la cuota de participación, los gastos de envío y el plazo de entrega.
8. El Sistema valida los datos.
9. El Sistema presenta al Administrador los datos introducidos con el IVA calculado al valor de puja mínima, a la cuota de participación y a los gastos de envío.
10. El Administrador guarda los cambios.
11. El Sistema registra la nueva subasta y asigna un número a la subasta.
12. El Sistema establece el estado de los artículos subastados a “En subasta”.
13. El Administrador guarda los cambios.

Nota del profesor. Observa cómo en la línea 12 se dice que el sistema debe establecer el estado de los artículos subastados a “En subasta”, pero en el modelo conceptual no hay ningún atributo que permita reflejar esto. Hay una inconsistencia entre el modelo de casos de uso y el modelo conceptual que sería preciso eliminar.

Extensiones (o Flujos Alternativos):

2-12a. El Administrador cancela la creación de la edición.
1. El Sistema cancela la edición junto con todas las subastas introducidas y no guarda los cambios.

6a. El Sistema detecta que el artículo elegido no está disponible en la cantidad solicitada.

1. El Sistema informa al Administrador de que la cantidad solicitada de artículos a subastar no está disponible y le permite elegir un artículo distinto o un número menor de artículos.

6-12a. El Administrador cancela la creación de subasta.
1. El Sistema cancela la subasta y no guarda los cambios.

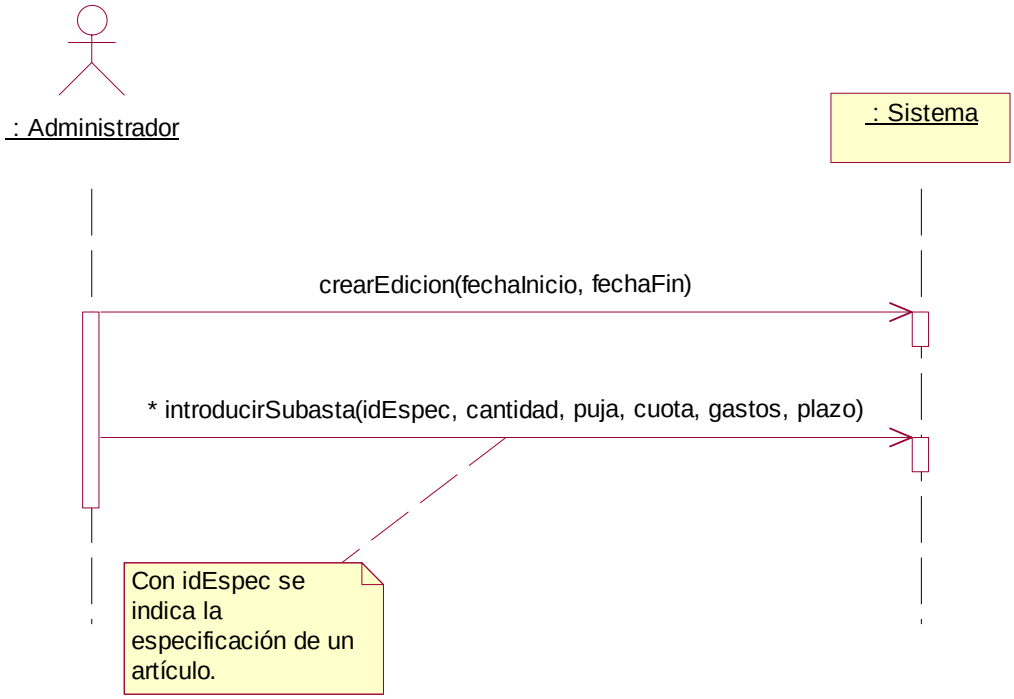
7a. El Administrador desea elegir una forma de pago.
1. El Administrador selecciona una forma de pago dentro de las posibles formas de pago.
2. El Sistema determina el valor del anticipo a partir de la forma de pago y pregunta al Administrador si desea modificar el valor del anticipo.
3. El Administrador no modifica el valor del anticipo.
   3a. El Administrador desea modificar el valor del anticipo.
      1. El Administrador introduce un valor de anticipo.

Requisitos Especiales:

- Un artículo no está disponible si está siendo subastado (su estado es “En subasta”).

Nota del profesor. Ten en cuenta el comentario anterior sobre la consistencia entre el modelo conceptual y el modelo de casos de uso.

Diagrama de secuencia del sistema



Contratos y colaboraciones

Contrato: crearEdicion

Operación: crearEdicion(fechaInicio: Fecha, fechaFin: Fecha)
Referencias: Casos de Uso: Crear edición de subasta
Controlador: ControladorAnuncios
Precondiciones:

- El Administrador está identificado y autenticado en el sistema.

Postcondiciones:

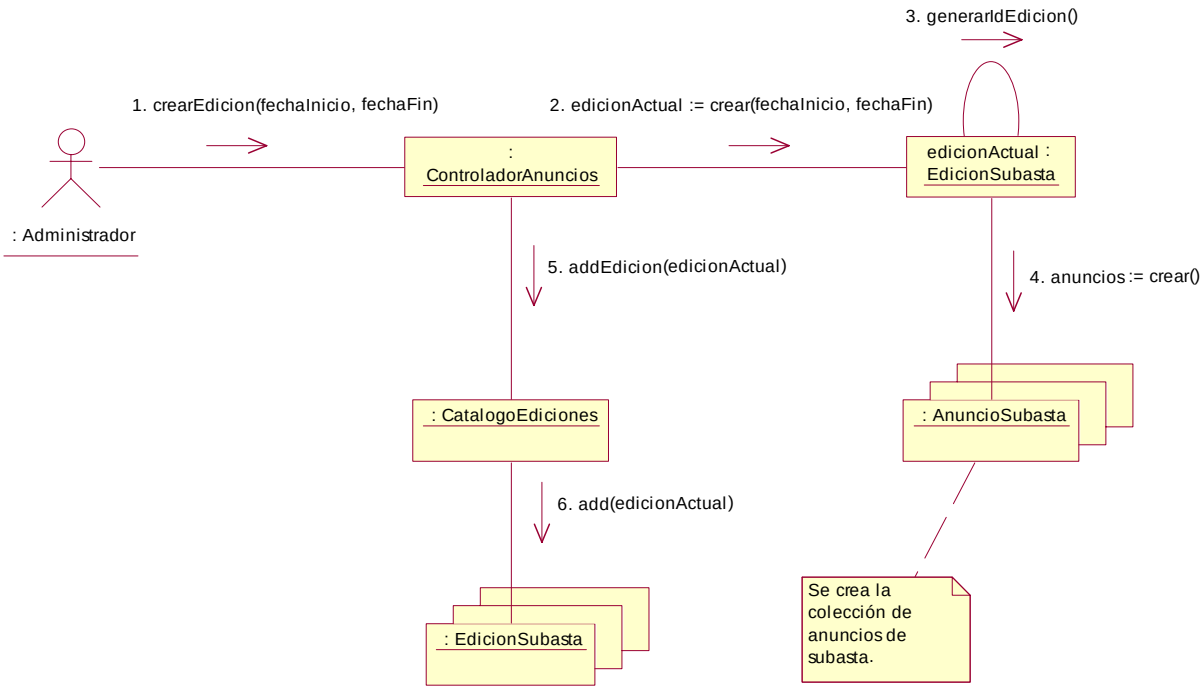
Nota del profesor. Observa el uso del pasado.

- Se creó una instancia edicionActual de EdicionSubasta.
- Se inicializó edicionActual.idEdicion
- edicionActual.fechaInicio = fechaInicio
- edicionActual.fechaCierre = fechaFin

Nota del profesor. Cuidado con la creación de colecciones, que se puede olvidar.

- Se creó una colección 'anuncios' para objetos de tipo AnuncioSubasta y se asoció con edicionActual.
- Se asoció edicionActual con el CatalogoEdiciones del ControladorAnuncios (se insertó en el catálogo).

Nota del profesor. Observa que se ha olvidado indicar un valor para edicionActual.estado, que es un atributo que aparece en el modelo de clases del análisis.



Contrato: introducirSubasta

Operación: introducirSubasta(idEspec: idEspec, cantidad: integer, puja: tipoDinero, cuota: tipoDinero, gastos: tipoDinero, plazo: intervalo)
Referencias: Casos de Uso: Crear edición de subasta
Controlador: ControladorAnuncios
Precondiciones:

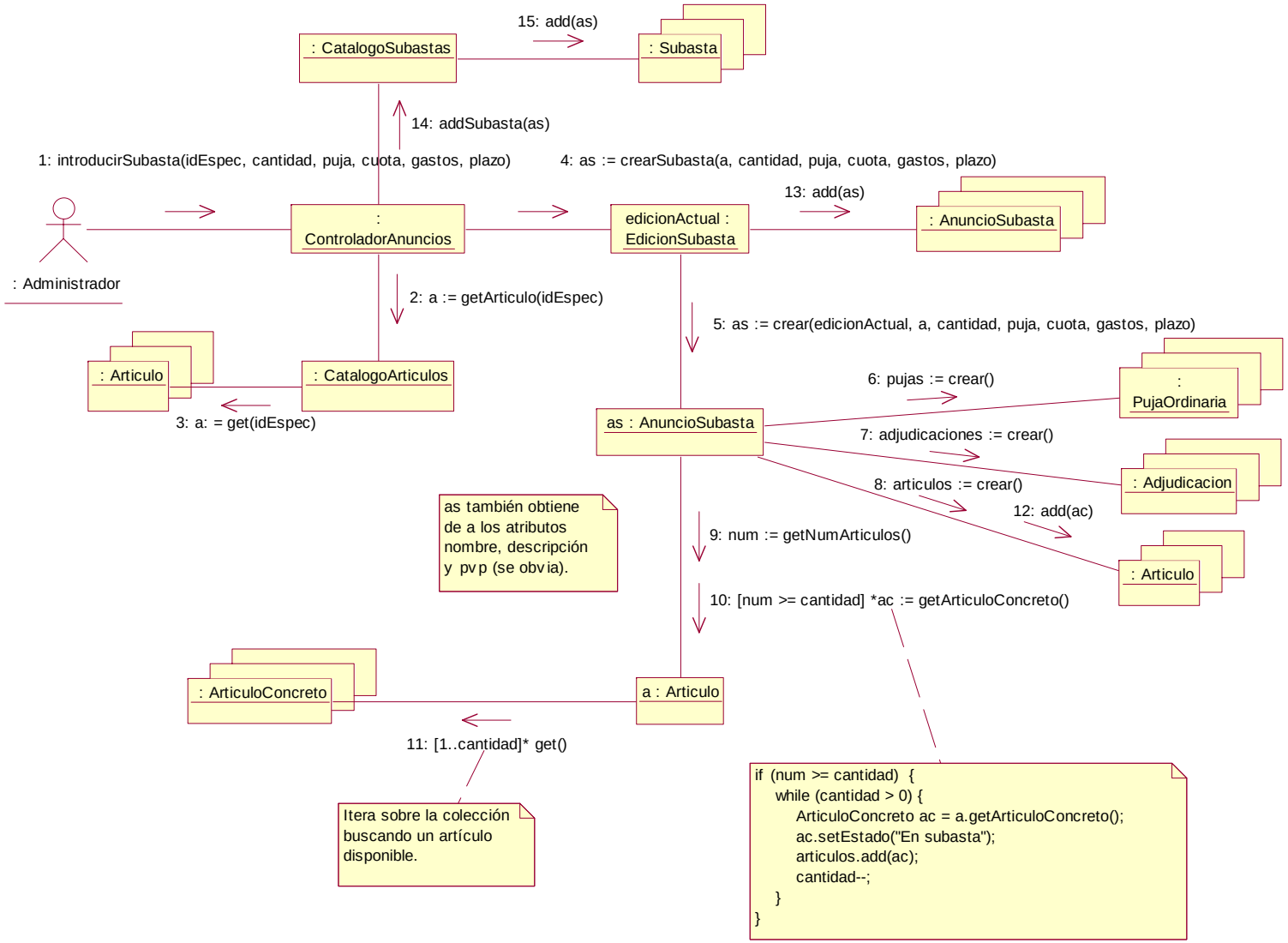
- Se está creando una edición de subasta (ControladorAnuncios tiene asociada una edicionActual)

Postcondiciones:

- Se creó una instancia as de AnuncioSubasta.
- Se inicializó as.numSubasta
- Se asoció as con edicionActual.
- Se asoció as con 'cantidad' instancias de ArticuloConcreto basándose en idEspec.
- Los atributos as.nombreArticulo, as.descArticulo y as.pvpRecomend se inicializaron de acuerdo con los atributos del Articulo a cuyo a.idEspec es idEspec.

Nota del profesor. Entiendo que "as" tenga su precio recomendado, porque el precio del artículo en cuestión puede cambiar con posterioridad a la creación de la subasta, pero veo menos claro que sea necesario copiar el nombre del artículo y la descripción del mismo.

- as.pujaMinima = puja
- as.cuota = cuota
- as.gastosEnvio = gastos
- as.plazo = plazo
- as.formaPago y as.anticipo tomaron valores por defecto.
- Se creó una colección 'pujas' para objetos de tipo PujaOrdinaria y se asoció con as.
- Se creó una colección 'adjudicaciones' para objetos de tipo Adjudicacion y se asoció con as.
- Se asoció as con la colección de AnuncioSubasta de edicionActual (se insertó en la colección).
- Se asoció as con el CatalogoSubastas del ControladorAnuncios (se insertó en el catálogo).


15: add(as)
: CatalogoSubastas
: Subasta
14: addSubasta(as)
1: introducirSubasta(idEspec, cantidad, puja, cuota, gastos, plazo)
4: as := crearSubasta(a, cantidad, puja, cuota, gastos, plazo)
13: add(as)
: ControladorAnuncios
edicionActual : EdicionSubasta
: AnuncioSubasta
: Administrador
2: a := getArticulo(idEspec)
5: as := crear(edicionActual, a, cantidad, puja, cuota, gastos, plazo)
: Articulo
: CatalogoArticulos
3: a: = get(idEspec)
6: pujas := crear()
: PujaOrdinaria
as : AnuncioSubasta
7: adjudicaciones := crear()
: Adjudicacion
8: articulos := crear()
12: add(ac)
: Articulo
as también obtiene de a los atributos nombre, descripción y pvp (se obvia).
9: num := getNumArticulos()
10: [num >= cantidad] *ac := getArticuloConcreto()
: ArticuloConcreto
a : Articulo
11: [1..cantidad]* get()
Itera sobre la colección buscando un artículo disponible.
if (num >= cantidad) {
while (cantidad > 0) {
ArticuloConcreto ac = a.getArticuloConcreto();
ac.setEstado("En subasta");
articulos.add(ac);
cantidad--;
}
}

Use Case UC2: Realizar puja ordinaria

Stakeholders:
- La Mega Subasta
- Pujador

Actor: Pujador
Precondiciones:
Postcondiciones: Una puja ordinaria es creada.
Escenario Principal (o Flujo Básico):

1. El Pujador decide pujar en un anuncio de subasta.
2. El Sistema comprueba que el Pujador ha entrado (log in).
3. El Sistema comprueba que el Pujador no ha pujado tres veces en el anuncio.
4. El Sistema pide el valor de la puja y los datos de la tarjeta de crédito con el que el Pujador efectuará el pago.
5. El Pujador introduce el valor de la puja y los datos de su tarjeta de crédito.
Nota del profesor. Parece poco funcional que el pujador tenga que introducir los datos de la tarjeta de crédito cada vez que desee pujar.
6. El Sistema pide confirmación para crear la puja.
7. El Pujador acepta.
8. El Sistema se comunica con la Entidad de Crédito y carga el valor de la cuota de participación del anuncio de subasta en la tarjeta especificada por el Pujador.
Nota del profesor. "Entidad de Crédito" no aparece en el modelo conceptual.
9. El Sistema registra el pago de la cuota realizado (importe y fecha de pago).
10. El Sistema registra la nueva puja (le asigna un número de serie y almacena el valor de la puja, los datos de la tarjeta y la fecha de realización).
11. El Sistema notifica al usuario.

Extensiones (o Flujos Alternativos):

2-6a. El Pujador cancela la puja.
1. El Sistema aborta la creación de la puja.

2a. El Sistema comprueba que el Pujador no ha entrado.
1. El Sistema solicita al Pujador los datos personales.
2. El Pujador introduce los datos personales.

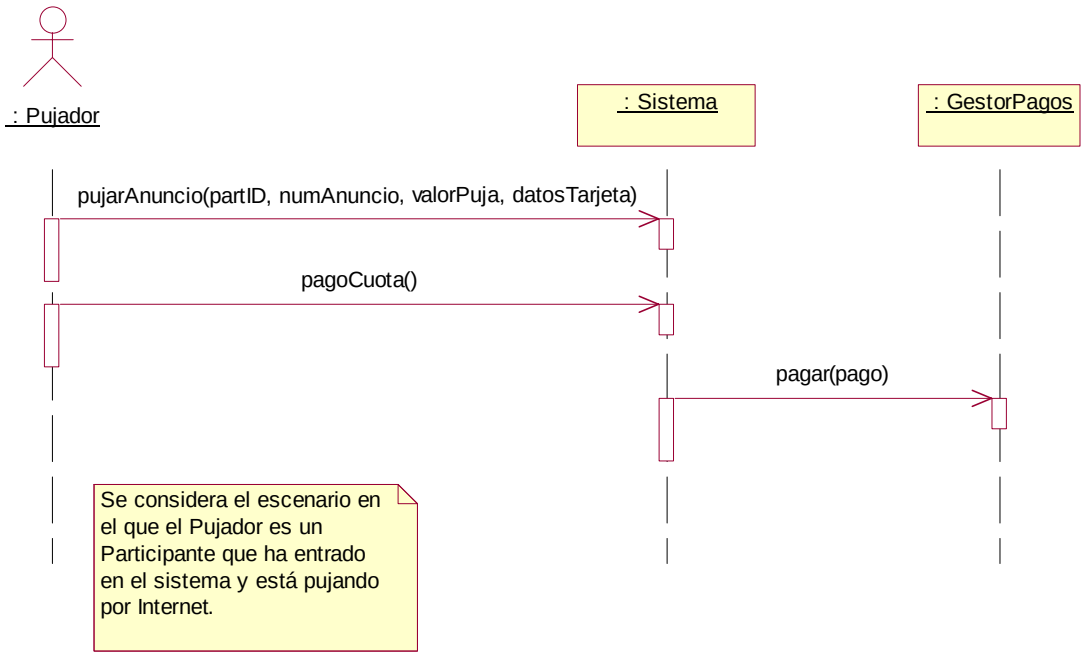
3a. El Pujador ha realizado 3 pujas en este anuncio.
1. El Sistema informa al Pujador de que no se pueden realizar más de tres pujas por anuncio de subasta y aborta la creación de la puja

5a. El Pujador introduce un valor de puja inferior a la puja mínima.
1. El Sistema rechaza el valor de la puja y solicita uno nuevo.

7a. El Pujador declina.
1. El Sistema aborta la creación de la puja.

8a. El Sistema detecta un fallo en la comunicación con la Entidad de Crédito.
1. El Sistema indica un error al Pujador y aborta la creación de la puja.

Diagrama de secuencia del sistema



Nota del profesor. La instancia de "GestorPagos" sólo debería aparecer en el DSS si fuera un subsistema externo.

En este diagrama de secuencia se observan 2 operaciones de sistema. Sin embargo, en realidad consideramos que son sólo una y que por lo tanto se puede reflejar su comportamiento en una única colaboración. Hemos decidido separarlo en dos colaboraciones para mayor claridad en los diagramas.

Contratos y colaboraciones

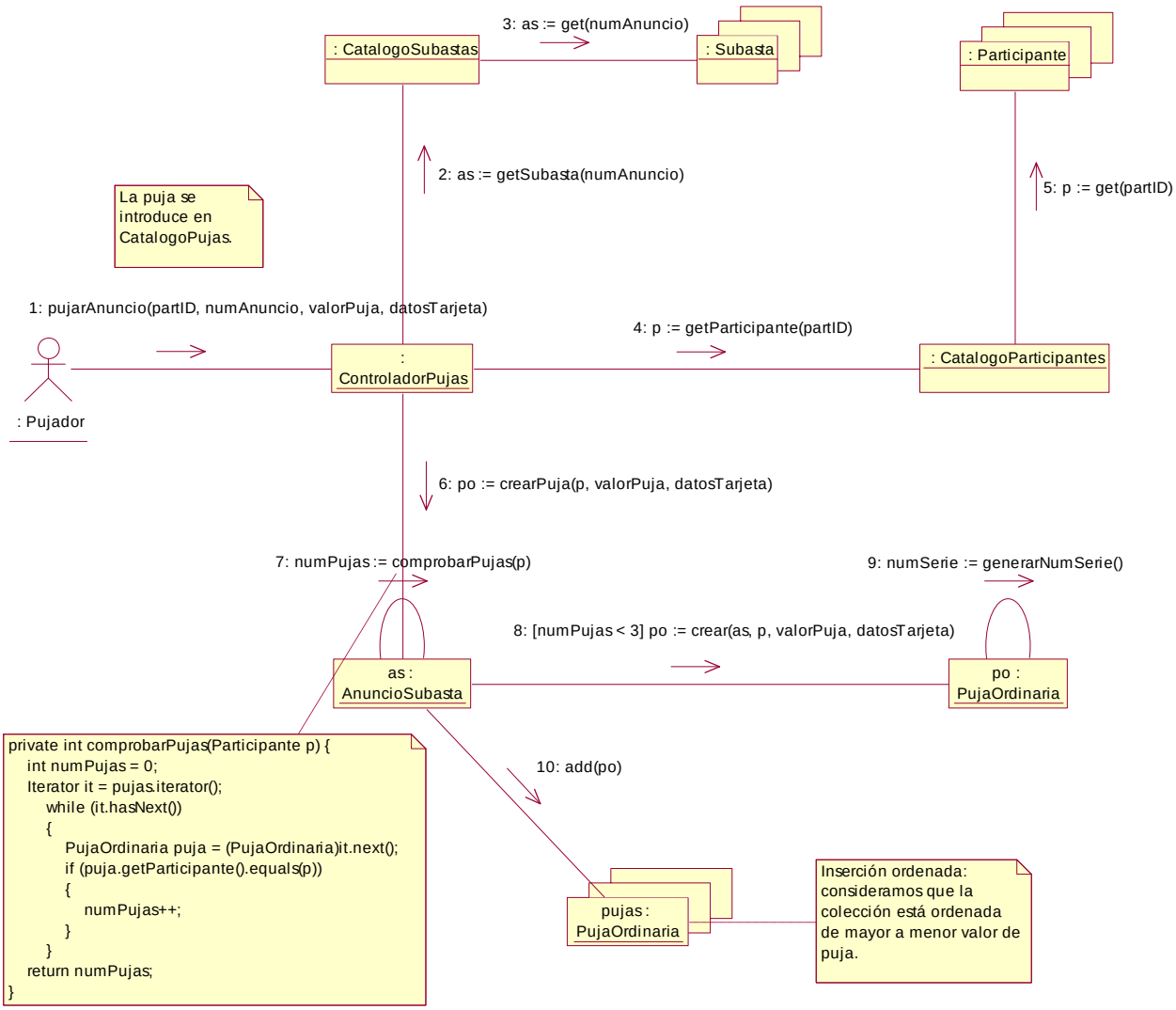
Contrato: pujarAnuncio

Operación: pujarAnuncio(partID: NumID, numAnuncio: NumSubasta, valorPuja: real, datosTarjeta: DatosTarjeta)
Referencias: Casos de Uso: Realizar puja ordinaria
Controlador: ControladorPujas
Precondiciones:
Postcondiciones:

- Se creó una instancia po de PujaOrdinaria.
- po.numSerie se inicializó.
- po.valorPuja = valorPuja
- po.datosTarjeta = datosTarjeta
- po.fecha = fechaActual
- po se asoció con el AnuncioSubasta as cuyo numAnuncio es numAnuncio.
- po se asoció con el Participante cuyo numID es partID.
- po se asoció con la colección de pujas del AnuncioSubasta as (se insertó en la colección).

- Se asoció po con el CatalogoPujas del ControladorPujas (se insertó en el catálogo).

Nota del profesor. po.estado = ¿? Debe haber una total consistencia con el modelo de clases del análisis.



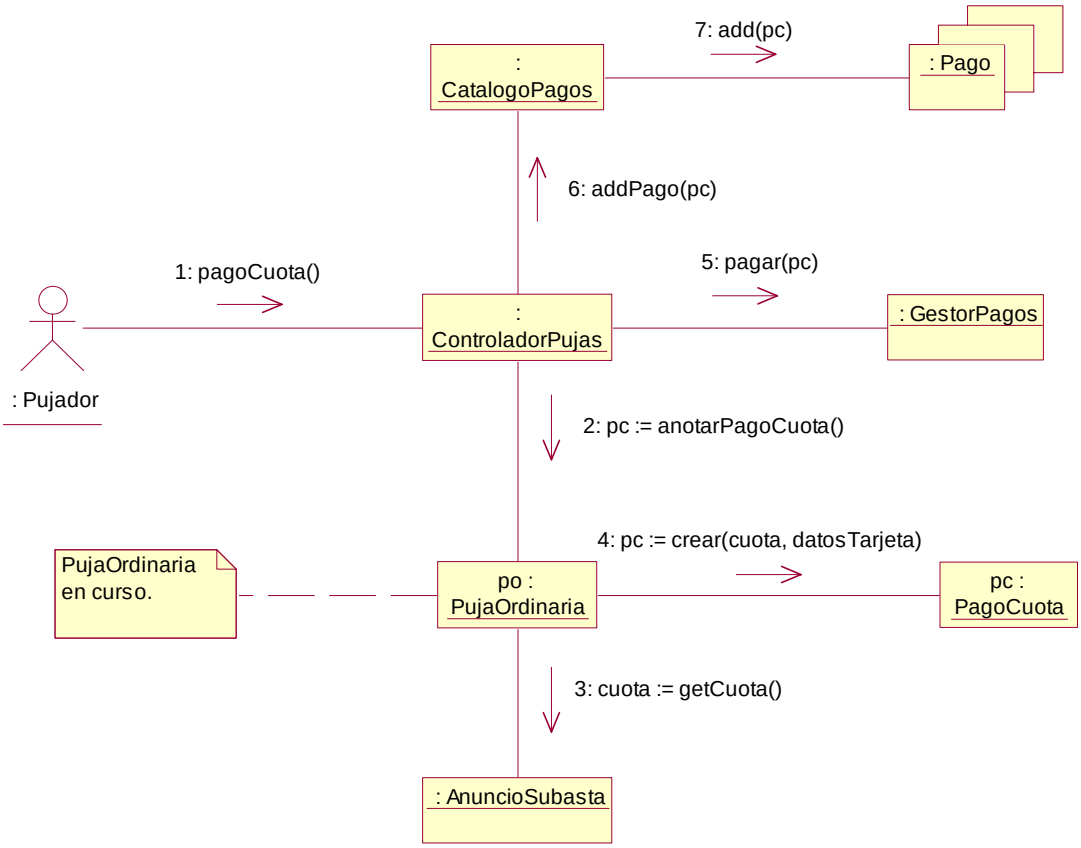
Contrato: pagoCuota

Operación: pagoCuota()
Referencias: Casos de Uso: Realizar puja ordinaria
Controlador: ControladorPujas
Precondiciones:

- Se está creando una puja ordinaria po.

Postcondiciones:

- Se creó una instancia pc de PagoCuota.
- pc creó y se asoció con una TransferenciaMegaSubasta con tarjetaOrigen = datosTarjeta de po e importe = el valor de la cuota (guardado en el AnuncioSubasta as asociado con la PujaOrdinaria po).
- Se asoció la instancia po de PujaOrdinaria con la instancia pc de PagoCuota.
- Se realizó el pago especificado en pc a través del GestorPagos.
- Se asoció el PagoCuota pc con el CatalogoPagos del ControladorPujas (se insertó en el catálogo).



Use Case UC3: Realizar pago de subasta ordinaria

Stakeholders:

- La Mega Subasta
- Pujador (todos los adjudicatarios).

Actor: Sistema
Precondiciones: Hay una o más pujas adjudicatarias de la subasta ordinaria para la que se van a realizar las operaciones de pago.
Postcondiciones: El Sistema ha llevado a cabo las operaciones de pago necesarias al acabar una subasta ordinaria.

Escenario Principal (o Flujo Básico):

1. El Sistema comienza las operaciones de pago al día hábil siguiente al del cierre de una subasta ordinaria.
2. El Sistema comprueba que el medio de pago del anuncio de subasta es el medio normal por tarjeta de crédito.
3. El Sistema accede a la lista de adjudicaciones de la subasta ordinaria.
   Para cada adjudicación el Sistema realiza los pasos 4-7:
4. El Sistema calcula el importe final a partir del valor de la puja adjudicataria y los gastos de envío del anuncio de subasta.
5. El Sistema obtiene los datos de la tarjeta a partir de la puja adjudicataria.
6. El Sistema se comunica con la Entidad de Crédito y carga el importe en la tarjeta.

Nota del profesor. Ya hemos dicho que "Entidad de Crédito" no aparece en el modelo conceptual.

7. El Sistema registra la transacción y el pago realizado, guardando el valor del importe final y la fecha de pago.

Extensiones (o Flujos Alternativos):

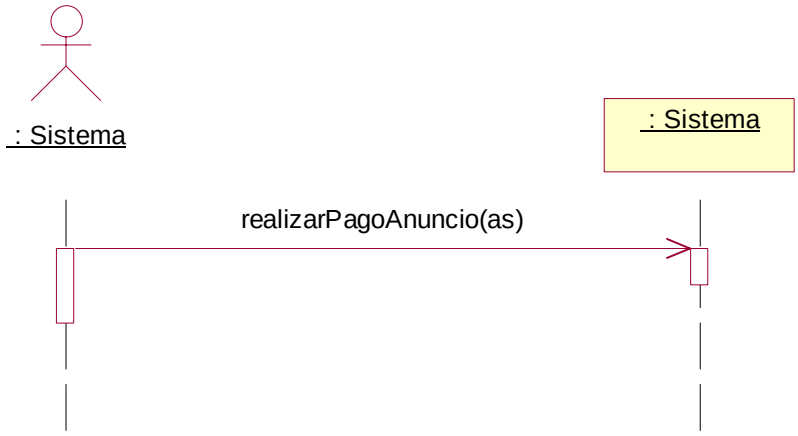
2a. El Sistema comprueba que el medio de pago no es el medio normal de pago por tarjeta de crédito.

1. El Sistema accede a la lista de adjudicaciones de la subasta ordinaria.
2. El Sistema obtiene el valor del anticipo del anuncio de subasta.
   Para cada adjudicación el Sistema realiza los pasos 3-5:
3. El Sistema obtiene los datos de la tarjeta a partir de la puja adjudicataria.
4. El Sistema se comunica con la Entidad de Crédito y carga el importe del anticipo en la tarjeta.
   4a. El Sistema detecta un fallo en la comunicación con la Entidad de Crédito.
   1. El Sistema registra el error y finaliza.
5. El Sistema realiza "algo" teniendo en cuenta la forma de pago distinta (no se conocen otras formas de pago).

6a. El Sistema detecta un fallo en la comunicación con la Entidad de Crédito.

1. El Sistema registra el error y finaliza.

Diagrama de secuencia del sistema



Contratos y colaboraciones

Contrato: realizarPagoAnuncio

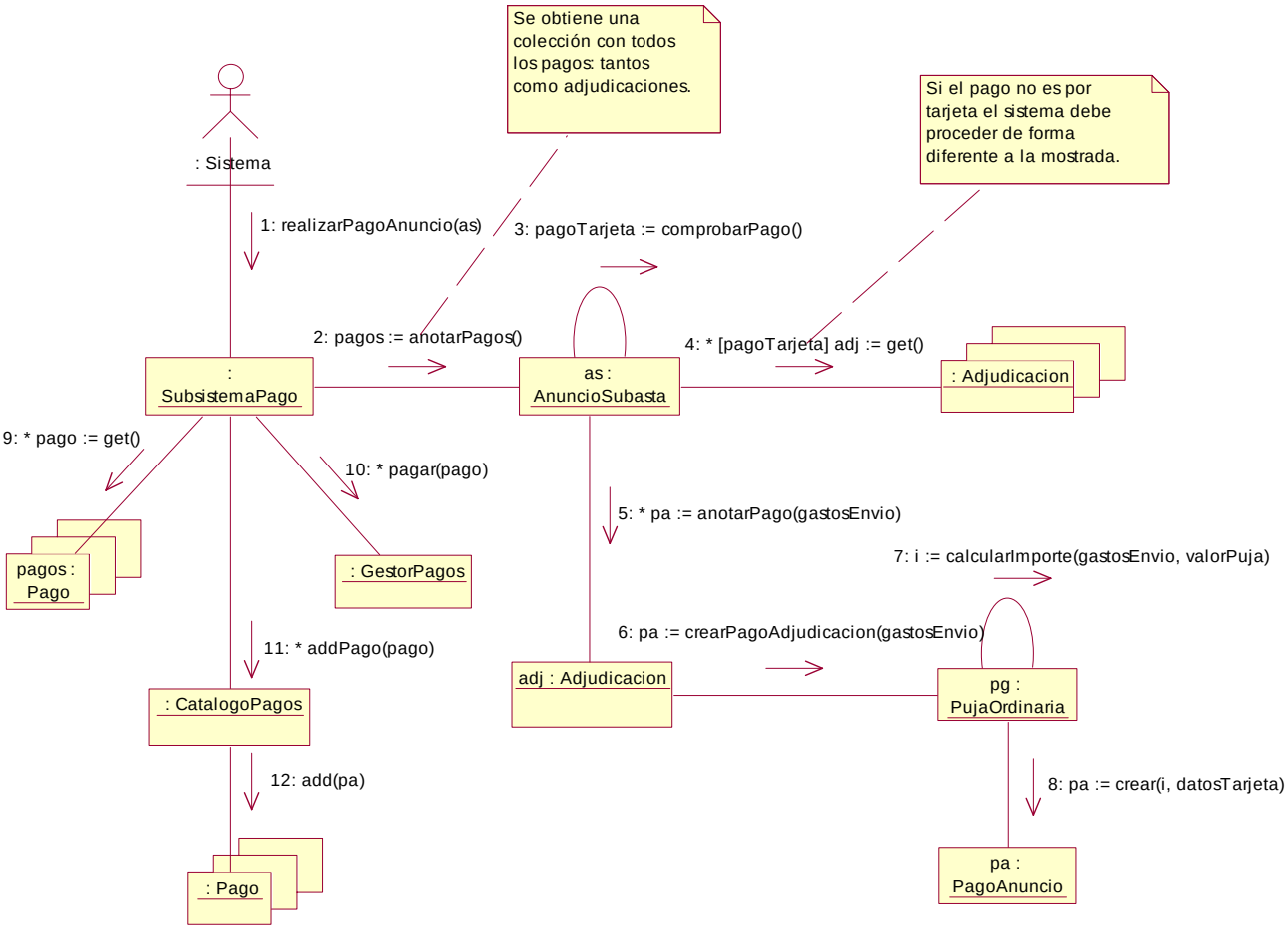
Operación: realizarPagoAnuncio(as: AnuncioSubasta)
Referencias: Casos de Uso: Realizar pago de subasta ordinaria
Controlador: SubsistemaPago
Precondiciones:

- La subasta as está cerrada.

Postcondiciones:

Para cada Adjudicacion a (siendo pg la PujaOrdinaria asociada a a):

- Se creó una instancia pa de PagoAnuncio.
- pa creó una TransferenciaMegaSubasta con tarjetaOrigen = datosTarjeta de la PujaOrdinaria pg e importe = valor de la puja pg + gastos envío
- pa.fechaPago = fecha actual
- Se asoció la PujaOrdinaria pg con el PagoAnuncio pa.
- Se realizó el pago especificado en pa a través del GestorPagos.
- Se añadió pa al CatalogoPagos del SubsistemaPago.


Se obtiene una colección con todos los pagos: tantos como adjudicaciones.
Si el pago no es por tarjeta el sistema debe proceder de forma diferente a la mostrada.
: Sistema
1: realizarPagoAnuncio(as)
3: pagoTarjeta := comprobarPago()
2: pagos := anotarPagos()
4: * [pagoTarjeta] adj := get()
: SubsistemaPago
as : AnuncioSubasta
: Adjudicacion
9: * pago := get()
10: * pagar(pago)
5: * pa := anotarPago(gastosEnvio)
7: i := calcularImporte(gastosEnvio, valorPuja)
pagos : Pago
: GestorPagos
6: pa := crearPagoAdjudicacion(gastosEnvio)
11: * addPago(pago)
adj : Adjudicacion
pg : PujaOrdinaria
: CatalogoPagos
12: add(pa)
8: pa := crear(i, datosTarjeta)
: Pago
pa : PagoAnuncio

Use Case UC4: Cerrar edición de subasta

Stakeholders: La Mega Subasta, los Pujadores.

Actor: Sistema
Precondiciones: La fecha de cierre de una edición de subasta ha vencido.
Postcondiciones: Se cierran los anuncios de subasta de la edición. Se emite la lista de resultados.

Escenario Principal (o Flujo Básico):

1. El Sistema comprueba que la fecha de cierre de una edición de subasta ha vencido y procede a cerrarla.
2. El Sistema obtiene los anuncios de subasta de la edición de subasta.
   Para cada anuncio de subasta el Sistema realiza los pasos 3-6:
3. El Sistema establece el estado de la subasta a "Cerrado".
4. El Sistema obtiene una lista de las Pujas ganadoras (las de mayor valor y tantas como artículos subastados).
5. El Sistema registra las adjudicaciones asociando para cada una la Puja ganadora, el anuncio de subasta y uno de los artículos subastados.
6. El Sistema establece el estado de los artículos subastados a "Adjudicado".
7. El Sistema emite la lista de resultados de la edición de subasta.

Extensiones (o Flujos Alternativos):

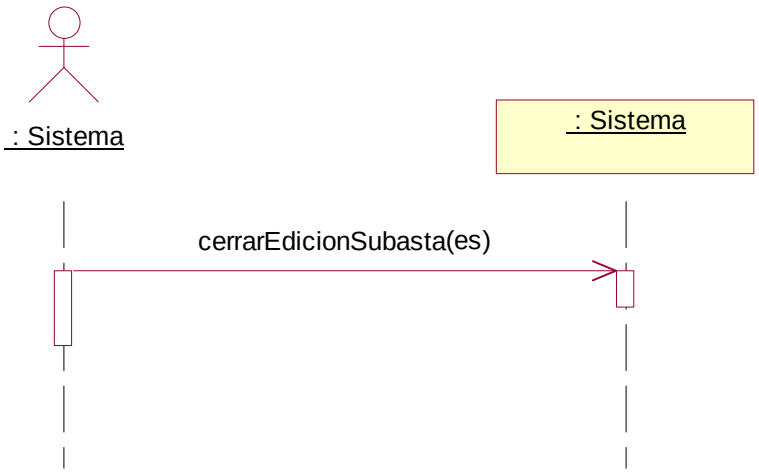
4a. Hay 2 pujas de igual valor.
1. El Sistema ordena las pujas de igual valor dando mayor prioridad a la más antigua.

4b. No existen pujas ganadoras.
1. El Sistema continúa con el siguiente anuncio de subasta.

5a. No se han adjudicado todos los artículos.
1. El Sistema marca el mismo número de artículos como "Adjudicados" que pujas ganadoras haya.

Nota del profesor. Observa como en esta práctica los campos de la plantilla de casos de uso que están vacíos no se han puesto. Es preferible que la plantilla aparezca completa aunque algunos campos queden vacíos.

Diagrama de secuencia del sistema



Contratos y colaboraciones

Contrato: cerrarEdicionSubasta

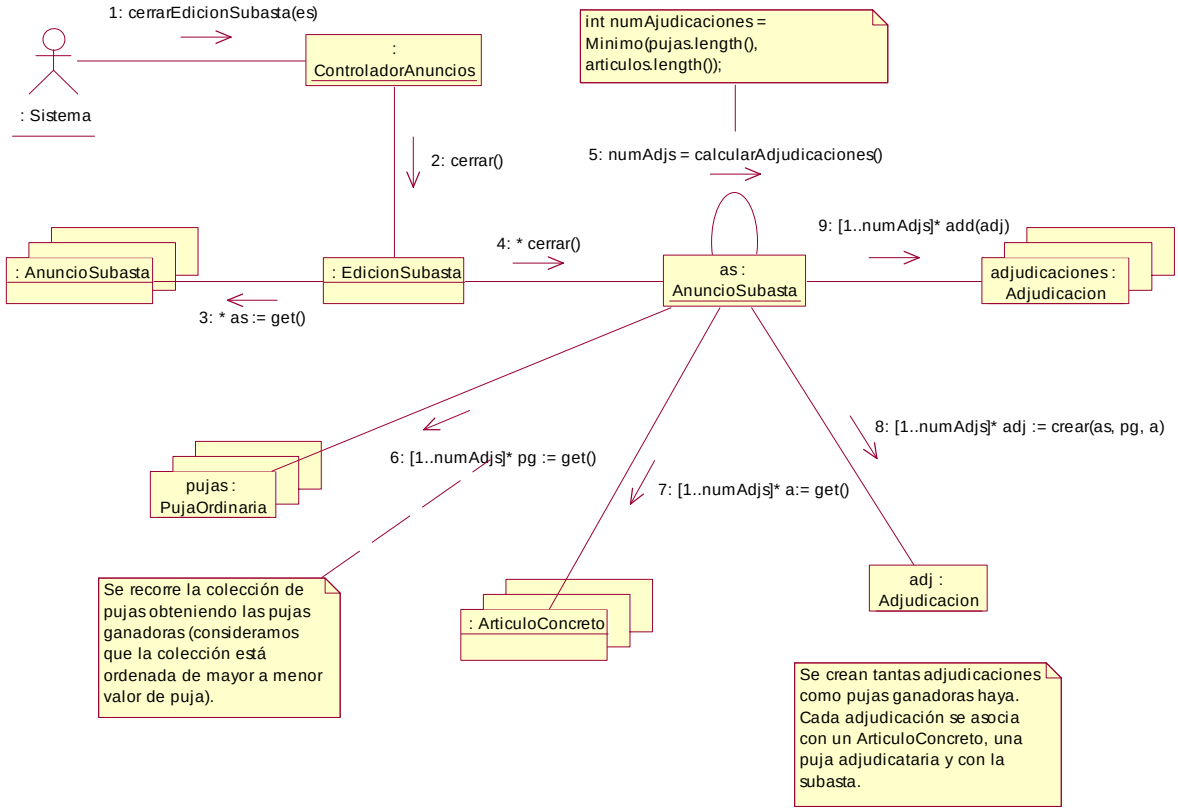
Operación: cerrarEdicionSubasta(es: EdicionSubasta)
Referencias: Casos de Uso: Cerrar edición de subasta
Controlador: ControladorAnuncios
Precondiciones:

- La fecha de cierre de la edición de subasta ha vencido.

Postcondiciones:

Para cada AnuncioSubasta as de la EdicionSubasta es:

- Se crearon n instancias adj de Adjudicacion (n = número de pujas adjudicatarias de as).
- Para cada Adjudicacion adj:
  - adj se asoció con as, la PujaOrdinaria adjudicataria y un ArticuloConcreto.
  - adj se asoció a la colección de objetos de tipo Adjudicacion de as.


1: cerrarEdicionSubasta(es)
: Sistema
: ControladorAnuncios
int numAjudicaciones = Minimo(pujas.length(), articulos.length());
2: cerrar()
5: numAdjs = calcularAdjudicaciones()
: AnuncioSubasta
: EdicionSubasta
4: * cerrar()
as : AnuncioSubasta
9: [1..numAdjs]* add(adj)
adjudicaciones : Adjudicacion
3: * as := get()
6: [1..numAdjs]* pg := get()
7: [1..numAdjs]* a:= get()
8: [1..numAdjs]* adj := crear(as, pg, a)
pujas : PujaOrdinaria
: ArticuloConcreto
adj : Adjudicacion
Se recorre la colección de pujas obteniendo las pujas ganadoras (consideramos que la colección está ordenada de mayor a menor valor de puja).
Se crean tantas adjudicaciones como pujas ganadoras haya. Cada adjudicación se asocia con un ArticuloConcreto, una puja adjudicataria y con la subasta.

Use Case UC5: Cancelar puja ordinaria

Stakeholders:
El Sistema
El Pujador

Actor: Pujador
Precondiciones: El Pujador ha realizado al menos una puja relacionada con la subasta.
Postcondiciones: La puja ha sido cancelada.

Escenario Principal (o Flujo Básico):

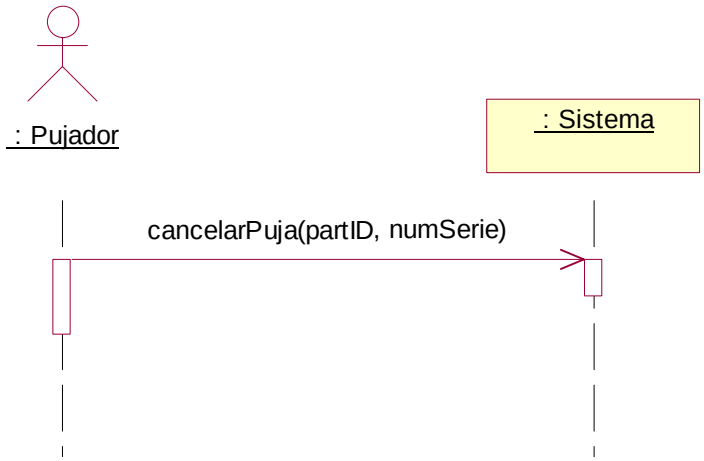
1. El Pujador decide cancelar una puja ordinaria.
2. El Sistema comprueba que el Pujador ha entrado (*log in*).
3. El Sistema pide al Pujador que seleccione la puja que desea cancelar.
4. El Pujador elige una de las pujas.
5. El Sistema comprueba que la subasta no ha concluido.
6. El Sistema pide confirmación para cancelar la puja.
7. El Pujador acepta.
8. El Sistema marca la Puja como “Cancelada”.
9. El Sistema notifica al usuario que ha cancelado la puja.

Extensiones (o Flujos Alternativos):

5a. La subasta ha terminado y la puja es adjudicataria.
1. El Sistema comprueba que el Pujador tiene más Pujas adjudicatarias.
1a. El pujador no tiene más pujas adjudicatarias.
1. El Sistema informa al Pujador que la operación es imposible.

5b. La subasta había terminado y la puja no es adjudicataria.
1. El Sistema notifica al usuario que la operación es imposible.

Diagrama de secuencia del sistema



Contratos y colaboraciones

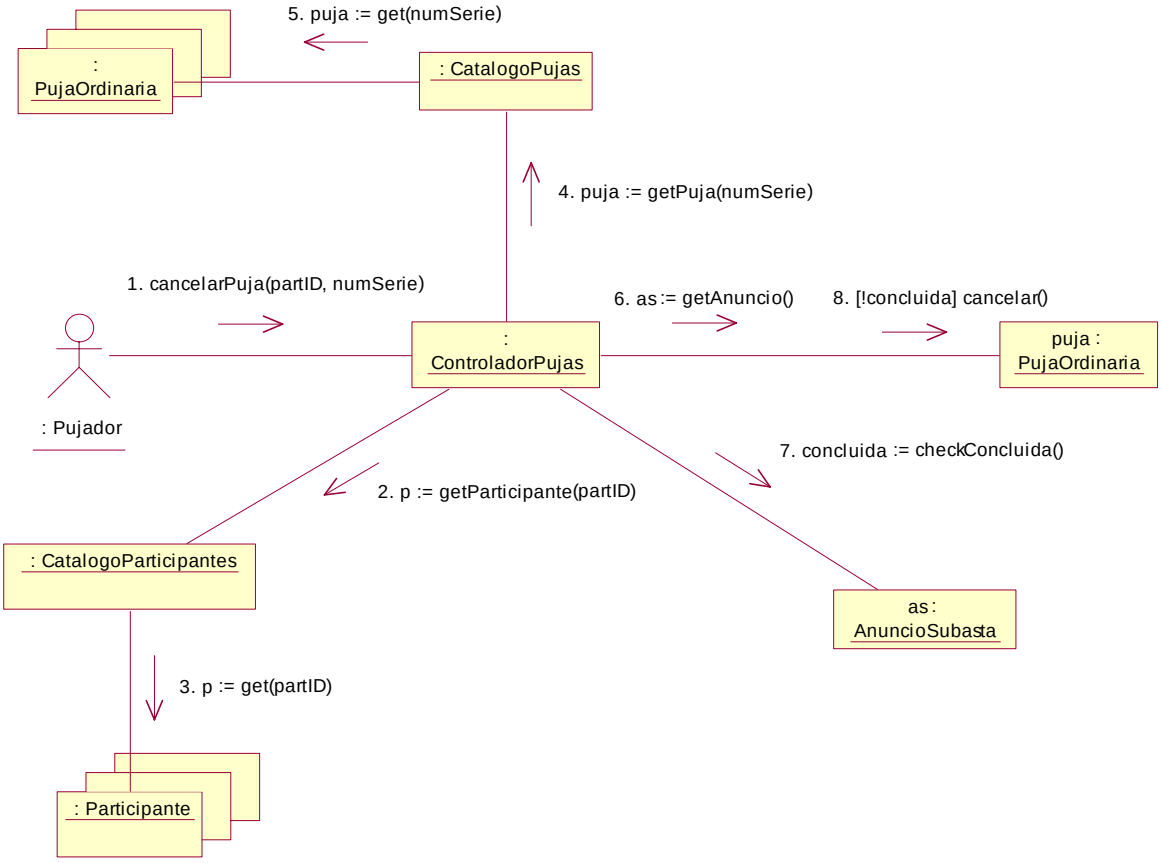
Contrato: cancelarPuja

Operación: cancelarPuja(partID, numSerie)
Referencias: Casos de Uso: Cancelar puja ordinaria
Controlador: ControladorPujas
Precondiciones:

- El anuncio de subasta en el que se ha realizado la puja identificada por “numSerie” no ha concluido.

Postcondiciones:

- La puja tomó el estado “Cancelada”.


5. puja := get(numSerie)
: PujaOrdinaria
: CatalogoPujas
4. puja := getPuja(numSerie)
1. cancelarPuja(partID, numSerie)
6. as := getAnuncio()
8. [!concluida] cancelar()
: ControladorPujas
puja : PujaOrdinaria
: Pujador
7. concluida := checkConcluida()
2. p := getParticipante(partID)
: CatalogoParticipantes
as : AnuncioSubasta
3. p := get(partID)
: Participante

Patrones de Diseño

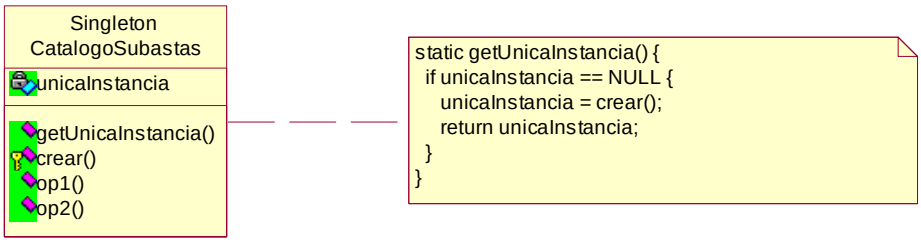
Durante el desarrollo de la práctica, se han identificado algunos patrones que se podrían aplicar a distintas partes del diseño resultante.

Singleton

Es uno de los patrones que tiene una implementación más simple y además es de identificación más directa. Consideramos que podríamos utilizar este patrón en:

- Los catálogos: es obvio que no es recomendable la aparición de varios catálogos haciendo referencia al mismo concepto y la utilización de este patrón nos proporcionaría un acceso seguro a un único catálogo de cada tipo.
- Los controladores: de forma similar a lo comentado anteriormente.
- También vemos conveniente usar este patrón para el uso de clases Contador que se encargarían de generar los números de identificación, números de serie, etc. de clases identificadas unívocamente con estos números. Por ejemplo, un contador de subastas de tal manera que cada vez que se creara una subasta ésta llamaría al contador para obtener su número de subasta.

Ejemplo de aplicación:



De este modo conseguimos tener una única instancia de Single CatalogoSubatas, a la que se accede y se obtiene a través del método de clase *getUnicaInstancia()*. Este método llama al de creación, que es protegido.

Iterator

El uso de este patrón guarda una relación muy directa con el propio lenguaje (Java) en el que se ha realizado el código de la aplicación, su uso ha sido muy extendido y ampliamente utilizado cuando se ha debido realizar el recorrido sobre los objetos de una colección. Este patrón permite abstraernos fácilmente de los detalles de implementación de las colecciones. Hacemos referencia al uso de iteradores externos. Mirando un poco más allá, podríamos implementar un iterador interno con una condición (un test) en la lista de pujas de la subasta ordinaria, donde se notifica solamente a los vencedores de la subasta.

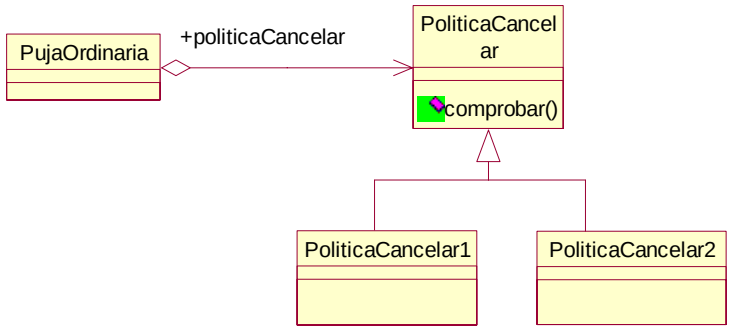
Strategy

Se ha percibido la posibilidad de utilizar este patrón en el cálculo de las cuotas que se cargan a los clientes. Es el uso más clásico de este patrón, gracias a ello, podríamos incluir diferentes tipos de cálculo de cuotas sin realizar modificaciones en el código. Si además de ello, introducimos el uso del patrón *Abstract Factory*, podríamos delegar toda la responsabilidad en la creación de las distintas políticas de precios y dotarlo de mayor flexibilidad. Podríamos pensar en la extensión del negocio a diferentes países y podríamos crear familias de cuotas relacionadas con las distintas legislaciones vigentes.

Este patrón también nos permitiría variar otras funcionalidades tal como la condición que determina si una puja se cancela o no. Si la Puja o la Subasta tuviera un

método que fuera *comprobarCancelar(Participante p)* que comprueba si el Participante puede cancelar la puja, podríamos encapsular este método en un objeto usando el patrón Strategy y variarlo cuando fuera necesario.

Ejemplo:



Al método *comprobar()* se le podría pasar el Participante y una referencia a la Puja, por ejemplo.

Otro uso del patrón Strategy podría ser para variar la condición por la que un Pujador puede realizar una PujaOrdinaria. Según la especificación, sólo puede hacerlo si ha realizado menos de 3 pujas en ese anuncio de subasta. Sin embargo, esta condición podría variar e incluso hacerse más compleja. Por lo que podría hacerse algo similar al caso anterior.

Del mismo modo, sería conveniente usar este patrón para el cálculo de la comisión al realizar el pago de las subastas de usuario.

State

Este patrón podría haberse utilizado, ya que hay objetos que poseen un estado y en base a ese estado se comportan de manera diferente. Por ejemplo: si una edición de subasta ha comenzado no se pueden introducir nuevas subastas en esa edición. Sin embargo, el número de estados de estos objetos es reducido y no se cree que vaya a aumentar en un futuro, con lo que no se forman grandes estructuras CASE en el código, sino meras comparaciones if-else en algunos lugares (ver código).

Proxy

El uso de este patrón es algo más sutil que el del resto, pensemos en la recuperación de un objeto persistente de nuestra base de datos, como por ejemplo AnuncioSubasta; puede tener muchas pujas asociadas o múltiples artículos asociados. Si necesitamos solamente un par de datos de dicho objeto, ¿por qué asumir el costo asociado a recuperar dichas pujas si no se van a referenciar?. Al cargar dicho objeto, podríamos asociarle proxies que se encargaran de cargar los objetos asociados si fuera necesario.

Para utilizar este patrón, habría que estudiar si su uso resultaría beneficioso dependiendo de requisitos de eficiencia y del método que se utilice para almacenar la información del programa.

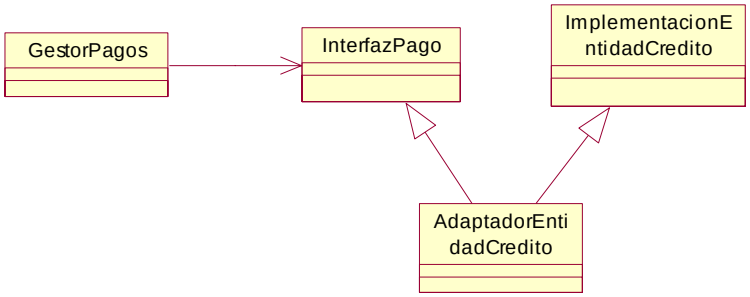
Observer

Una aplicación posible de este patrón a la práctica sería:
En la implementación de la interfaz gráfica, el propio lenguaje en el que se ha realizado la preimplementación, proporciona el modelo de delegación de eventos para la gestión de los elementos como los botones, o cuando se realice el cambio sobre algún elemento del dominio, éste debe notificar a la GUI.

Adapter

Es de suponer que las diferentes entidades de crédito con las que tenga relación la empresa, no nos proporcionarán la interfaz deseada, o al menos no todas nos ofrecerán la misma forma de comunicación con ellos. Además el uso de este patrón prevendría posibles cambios en las interfaces ofrecidas por las entidades (una nueva plataforma de conexión, por ejemplo), dado que reduciríamos el acoplamiento entre los sistemas y realizar cambios en un adaptador sería mucho menos "traumático" que realizarlos en una clase del propio modelo.

Ejemplo:



En este ejemplo suponemos que nuestra clase GestorPagos utiliza alguna interfaz para realizar el pago. Como la implementación que proporcione la Entidad no es probable que coincida con la que utiliza GestorPagos, usaríamos una clase adaptadora. En este ejemplo mostramos un adaptador basado en herencia.

Diagrama de Clases

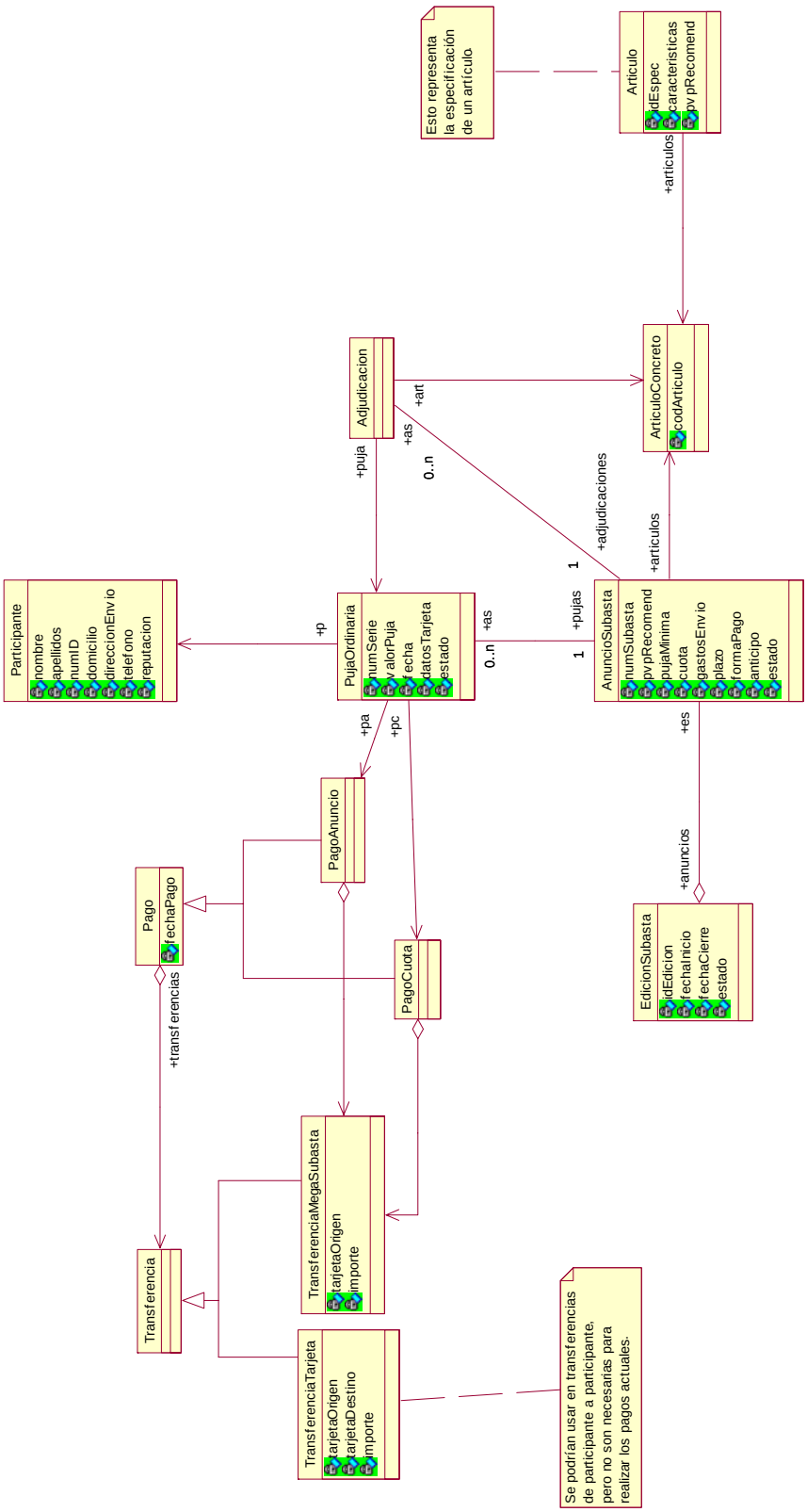
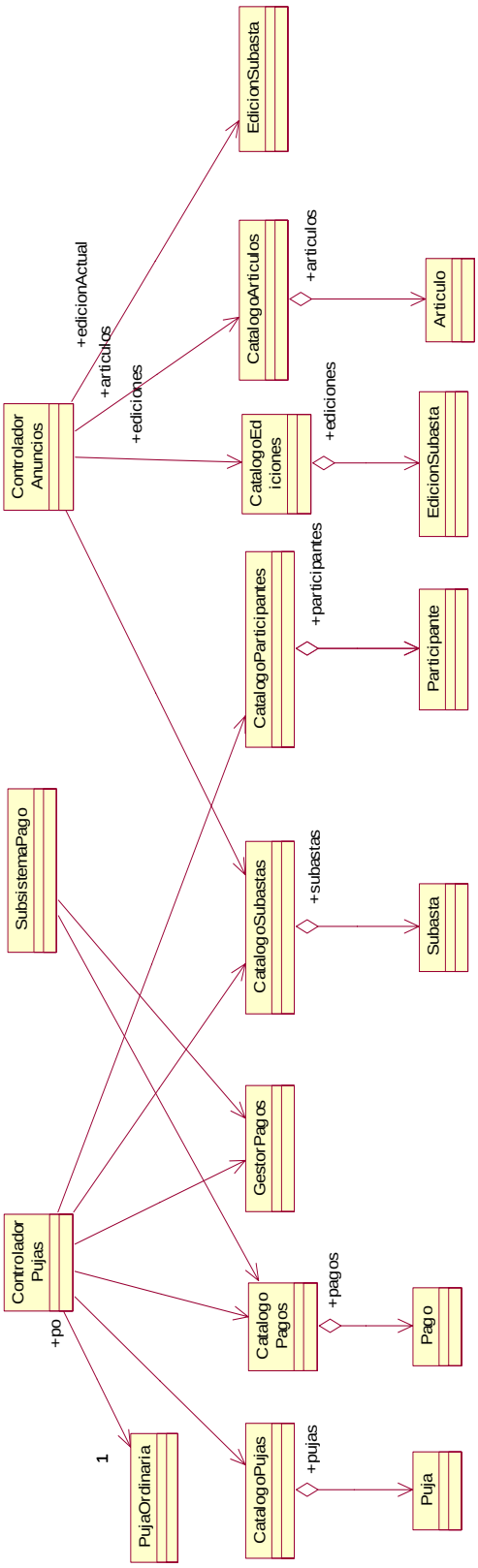


Diagrama de controladores.



Listado de código

```
   /** CONTROLADORES **/

public class ControladorAnuncios
{
   protected CatalogoArticulos articulos;
   protected EdicionSubasta edicionActual;
   protected CatalogoEdiciones ediciones;
   protected CatalogoSubastas subastas;

   public ControladorAnuncios() {
         /* Coger instancias de los catálogos (singleton) */
   }

   public void crearEdicion(Date fechaInicio, Date fechaFin) {
         edicionActual = new EdicionSubasta(fechaInicio, fechaFin);
         ediciones.addEdicion(edicionActual);
   }

   public void introducirSubasta(int idEspec, int cantidad, float puja, float
cuota, float gastos, int plazo) {
         Articulo a = articulos.getArticulo(idEspec);
         AnuncioSubasta as = edicionActual.crearSubasta(a, cantidad, puja, cuota,
gastos, plazo);
         subastas.addSubasta(as);
   }

   public void cerrarEdicionSubasta(EdicionSubasta es) {
         es.cerrar();
   }

};

public class ControladorPujas
{
   protected CatalogoPujas pujas;
   protected CatalogoPagos pagos;
   protected GestorPagos gestorPagos;
   protected CatalogoSubastas subastas;
   protected CatalogoParticipantes participantes;

   public ControladorPujas() {
         /* Coger instancias de los catálogos (singleton) */
   }

   public void pujarAnuncio(int partID, int numAnuncio, float valorPuja, Tarjeta
datosTarjeta) {
         AnuncioSubasta as = subastas.getSubasta(numAnuncio);
         Participante p = participantes.getParticipante(partID);
         PujaOrdinaria po = as.crearPuja(p, valorPuja, datosTarjeta);
         pujas.addPuja(po);
         Pago pc = po.anotarPagoCuota();
         gestorPagos.pagar(pc);
         pagos.addPago(pc);
   }

   public void cancelarPuja(int partID, int numSerie) {
         Participante p = participantes.getParticipante(partID);
         PujaOrdinaria puja = pujas.getPuja(numSerie);

         AnuncioSubasta as = puja.getAnuncio();
         if (!as.concluida())
               puja.cancelar();
   }
};
```

```
public class SubsistemaPago
{
   protected CatalogoPagos pagos;
   protected GestorPagos gestorPagos;

   public SubsistemaPago() {
   }

   public void realizarPagoAnuncio(AnuncioSubasta as) {
          Collection pagos = as.anotarPagos();
          Iterator it = pagos.iterator();
          while (it.hasNext())
          {
                 Pago pago = (Pago)it.next();
                 gestorPagos.pagar(pago);
                 pagos.addPago(pago);
          }
   }

};

/** OTROS **/

public class Participante
{

   protected String nombre;
   protected String apellidos;
   protected int numID;
   protected String domicilio;
   protected String direccionEnvio;
   protected String telefono;
   protected int reputacion;

   public Participante() {
   }

   public void notificar() {
   }

};

public abstract class Pago
{
   protected Collection transferencias;
   protected fechaPago;

   public void add(Transferencia trans) {
          transferencias.add(trans);
   }

};

public class PagoAnuncio extends Pago
{
   public PagoAdjudicacion(float i, Tarjeta datosTarjeta) {
          transferencias = new Collection();
          transferencias.add(new TransferenciaMegaSubasta(i, datosTarjeta));
          fechaPago = getFechaActual();
   }
};

public class PagoCuota extends Pago
{
   public PagoCuota(float cuota, Tarjeta datosTarjeta) {
          transferencias = new Collection();
          transferencias.add(new TransferenciaMegaSubasta(cuota, datosTarjeta));
          fechaPago = getFechaActual();
   }
};
```

```
public abstract class Puja
{
   protected int numSerie;
   protected float valorPuja;
   protected Date fecha;

   public float getValorPuja(){
          return valorPuja;
   }
};

public class PujaOrdinaria extends Puja
{
   protected Tarjeta datosTarjeta;
   protected AnuncioSubasta as;
   protected Participante p;
   protected PagoCuota pc;
   protected PagoAnuncio pa;
   protected String estado;

   public PujaOrdinaria(AnuncioSubasta as, Participante p, float valorPuja,
Tarjeta datosTarjeta) {
          this.as = as;
          this.p = p;
          this.valorPuja = valorPuja;
          this.datosTarjeta = datosTarjeta;
          fecha = getFechaActual();
          numSerie = generarNumSerie();
   }

   public PagoCuota anotarPagoCuota() {
          float cuota = as.getCuota();
          pc = new PagoCuota(cuota, datosTarjeta);
          return pc;
   }

   public crearPagoAdjudicacion(float gastosEnvio) {
          float i = valorPuja + gastosEnvio;
          pa = new PagoAnuncio(i, datosTarjeta);
          return pa;
   }

   public void cancelar() {
          estado = "Cancelada";
   }
};

public class Adjudicacion
{
   protected PujaOrdinaria puja;
   protected AnuncioSubasta as;
   protected ArticuloConcreto art;

   public Adjudicacion(AnuncioSubasta as, PujaOrdinaria pujaGanadora,
ArticuloConcreto a) {
          this.as = as;
          puja = pujaGanadora;
          art = a;
   }

   public PagoAdjudicacion anotarPago(float gastosEnvio) {
          PagoAnuncio pa = puja.crearPagoAdjudicacion(gastosEnvio);
          return pa;
   }

};

public class EdicionSubasta
{
```

```
   protected int idEdicion;
   protected Date fechaInicio;
   protected Date fechaCierre;
   protected Collection anuncios;
   protected String estado;

   public EdicionSubasta(Date fechaInicio, Date fechaCierre) {
          this.fechaInicio = fechaInicio;
          this.fechaCierre = fechaCierre;
          idEdicion = generarIdEdicion();
          anuncios = new Collection();
          estado = "Abierta";
   }

   public void comenzar() {
          estado = "Comenzada";
   }

   public AnuncioSubasta crearSubasta(Articulo a, int cantidad, float puja, float
cuota, float gastos, int plazo) {

          if (estado != "Abierta") { /* ERROR: Sólo se puede insertar subastas en
estado Abierta */ }
          else {
                 AnuncioSubasta as = new AnuncioSubasta(this, a, cantidad, puja,
cuota, gastos, plazo);
                 anuncios.add(as);
                 return as;
          }
   }

   public void cerrar() {
          if (estado != "Comenzada") { /* ERROR */ }
          else {
                 Iterator it = anuncios.iterator();
                 while (it.hasNext()) {
                        AnuncioSubasta as = it.next();
                        as.cerrar();
                 }
                 estado = "Cerrada";
          }
   }

};

public abstract class Subasta
{
   protected int numSubasta;
   protected String nomArticulo;
   protected String descArticulo;
   protected String estado;
};

public class AnuncioSubasta extends Subasta
{
   protected float pvpRecomend;
   protected float pujaMinima;
   protected float cuota;
   protected float gastosEnvio;
   protected int plazo;
   protected String formaPago;
   protected float anticipo;

   protected EdicionSubasta es;
   protected Collection pujas;
   protected Collection articulos;

   public AnuncioSubasta(EdicionSubasta es, Articulo a, int cantidad, float puja,
float cuota, float gastos, int plazo) {
          this.es = es;
```

```
        pujaMinima = puja;
        this.cuota = cuota;
        gastosEnvio = gastos;
        this.plazo = plazo;
        formaPago = "Tarjeta";
        anticipo = 0;

        pujas = new Collection();
        articulos = new Collection();
        adjudicaciones = new Collection();

        int num = a.getNumArticulos();
        if (num >= cantidad) {
              while (cantidad > 0)
              {
                    ArticuloConcreto ac = a.getArticuloConcreto();
                    ac.setEstado("En subasta");
                    articulos.add(ac);
                    cantidad--;
              }
        }
        else { /** ERROR **/ }

        nombreArticulo = a.getNombre();
        descArticulo = a.getDesc();
        pvpRecomendad = a.getPVP();

        numSubasta = generarNumSubasta();
  }

  public PujaOrdinaria crearPuja(Participante p, float valorPuja, Tarjeta
datosTarjeta) {
        int numPujas = comprobarPujas(p);
        if (comprobarPujas(p)) {
              PujaOrdinaria po = new PujaOrdinaria(this, p, valorPuja,
datosTarjeta);
              pujas.add(po);
              return po;
        }
        else { /** ERROR **/ }
  }

  /* Devuelve una lista de pagos (tantos pagos como adjudicaciones haya) */
  public Collection anotarPagos() {
        Collection pagos = new Collection();
        if (formaPago == "Tarjeta") {
              Iterator it = adjudicaciones.iterator();
              while (it.hasNext()) {
                    Adjudicacion adj = it.next();
                    PagoAdjudicacion pa = adj.anotarPago(gastosEnvio);
                    pagos.add(pa);
              }
              return pagos;
        }
        else { /** Proceder de forma distinta **/ }
  }

  public void cerrar() {
        /* El número de adjudicaciones es el mínimo del número de pujas
        y el número de artículos */
        int numAjudicaciones = Minimo(pujas.length(), articulos.length());
        if (numAdjudicaciones > 0) {
              Iterator itArt = articulos.it();
              Iterator itPujas = pujas.it();
              int i = numAjudicaciones;
              while (i > 0) {
                    /* Suponemos que la colección de pujas está ordenada
empezando por las
                    de mayor valor */
                    PujaOrdinaria pg = it.next();
```

```
                        ArticuloConcreto art = itArt.next();
                        art.setEstado("Adjudicado");
                        Adjudicacion adj = new Adjudicacion(this, pg, art);
                        adjudicaciones.add(adj);
                        i--;
                  }
            }
      }

      public boolean concluida() {
            if (es.getFechaCierre() < fechaActual) return true;
            else return false;
      }

      private int comprobarPujas(Participante p) {
            int numPujas = 0;
            Iterator it = pujas.iterator();
            while (it.hasNext())
            {
                  PujaOrdinaria puja = (PujaOrdinaria)it.next();
                  /* Comprobar cuantas veces ha pujado el Participante p en esta
subasta */
                  if (puja.getParticipante().equals(p))
                  {
                        numPujas++;
                  }
            }
            return numPujas;
      }

};

public class ArticuloConcreto
{
      protected int codArticulo;
      protected String estado;

      public ArticuloConcreto() {
      }
};

public class Articulo
{
      protected int idEspec;
      protected float pvpRecomend;
      protected String nombre;
      protected String desc;

      protected Collection articulosConcretos;

      public Articulo(float pvp, String nombre, String desc) {
            this.pvpRecomend = pvp;
            this.nombre = nombre;
            this.desc = desc;
            idEspec = generarIdEspec();
            articulosConcretos = new Collection();
      }

      public int getNumArticulos() {
            return articulosConcretos.length();
      }

      public ArticuloConcreto getArticuloConcreto() {
            Iterator it = articulosConcretos.iterator();
            while (it.hasNext())
            {
                  ArticuloConcreto ac = (ArticuloConcreto)it.next();
                  if (ac.getEstado() == "Disponible")
                        return ac;
            }
```

```
            /* Lanzar error. No hay artículos disponibles */
    }
};
```