

# Seminario de Rational Rose 2001

## 1. Introducción

Existen un buen número de herramientas de modelado con UML, entre las más conocidas encontramos MagicDraw, Poseidon, IBM Rational Rose, EclipseUML, Borland Together y Visual Paradigm. Es posible descargarse versiones de prueba de estas herramientas. Rational Rose fue la primera herramienta UML disponible, creada por la empresa Rational Software, entre cuyos fundadores se encontraban los creadores de UML. En marzo de 2003, IBM adquirió Rational Software por 2.100 millones de dólares.

El objetivo del seminario es aprender el manejo básico de Rational Rose 2001 para que los alumnos puedan utilizarla para construir los diagramas necesarios en el desarrollo del trabajo práctico de la asignatura. La herramienta soporta UML 1.1. Para el próximo curso está previsto el uso de MagicDraw 11 que soporta UML 2.0. Cabe señalar que el estudio completo de una herramienta de modelado con UML no es uno de los objetivos de la asignatura.

## 2. Presentación de Rose

### Inicio

Al arrancar la herramienta no selecciones ninguno de los *frameworks* que aparecen, elige la opción *Cancelar*.

En el menú `Tools/Options...` selecciona la pestaña `Notation`, y establece como notación por defecto `Unified` y como lenguaje por defecto `Analysis`.

### Archivos de proyecto en Rose

Un modelo en Rose se almacena como un fichero `.mdl`

### Estructura de la ventana principal de Rose

En esta ventana destaca el *explorador* (browser) en la parte izquierda, desde donde se pueden crear nuevos elementos del modelo y acceder a todos los elementos del modelo actual, con una pequeña ventana de documentación debajo de ella, en la que aparece la documentación textual asociada al elemento seleccionado en el explorador.

En el canvas de la derecha se abren los diagramas que sirven de representación gráfica del modelo; entre esta ventana destaca la de log, que contiene información administrativa de los modelos (fechas de creación, actualización, etc.) junto con los errores de consistencia que se vayan produciendo.

## Las cuatro vistas de la arquitectura en Rose

Como comentamos en clase, la “Arquitectura 4+1” es una conocida arquitectura del sistema que lo organiza en cinco vistas, la vista de casos de uso que juega un papel central y las otras cuatro vistas son: la vista de diseño, vista de procesos, vista de implementación y vista de despliegue. Por defecto, en Rational Rose el explorador contiene los paquetes (carpetas) que corresponden a la “arquitectura 4+1” pero se han unido la vista de diseño y la vista de procesos en una vista denominada vista lógica.

Cada vista muestra una proyección de la arquitectura y usa un conjunto de diagramas. Por defecto, cada una de estas vistas tiene un diagrama especial, denominado Main (su icono cambia en cada vista), y una carpeta especial Associations. Cada vista se puede estructurar en paquetes, de manera que el diagrama Main suele estar formado por paquetes y sus relaciones.

Las vistas de Rose son las siguientes:

- 1) La *Vista de Casos de Uso*, *Use Case View*, que es la vista en la que se presenta el comportamiento deseado del sistema: en ella se encontrarían los modelos relacionados con la captura de requisitos. Según el proceso que hemos visto en clase, en esta vista se ubicarían el modelo del negocio, el modelo conceptual, el modelo de casos de uso del sistema y los diagramas de secuencia del sistema.
- 2) La *Vista Lógica*, *Logical View*, en la que encontraríamos los modelos que muestran el vocabulario y la funcionalidad (estructura y comportamiento) del sistema, a través de un conjunto de colaboraciones que *realizan* los casos de uso de la vista de casos de uso (colaboraciones que se modelan mediante diagramas de clases y diagramas de interacción: secuencia y colaboración). También comprende los hilos y procesos que establecen la concurrencia del sistema.
- 3) La *Vista de Componentes*, *Component View*, en la que se representa la implementación del sistema mediante componentes, la organización modular del software. Esta vista está relacionada con la gestión de la configuración del software. Los paquetes en esta vista se organizan en niveles. Un componente está relacionado con un archivo de software y un lenguaje de programación. Las clases de la vista lógica se asignarían a los componentes de la vista de componentes.
- 4) La *Vista de Despliegue*, *Deployment View*, en la que se modela la distribución o despliegue de los componentes a los nodos de procesamiento del sistema. Muestra la topología, distribución e instalación del sistema.

En el explorador tenemos también en una carpeta especial la opción Model Properties, que coincide con Tools/Options..., y que contiene la información de configuración del proyecto actual, incluyendo la apariencia de los elementos y las opciones de generación de código.

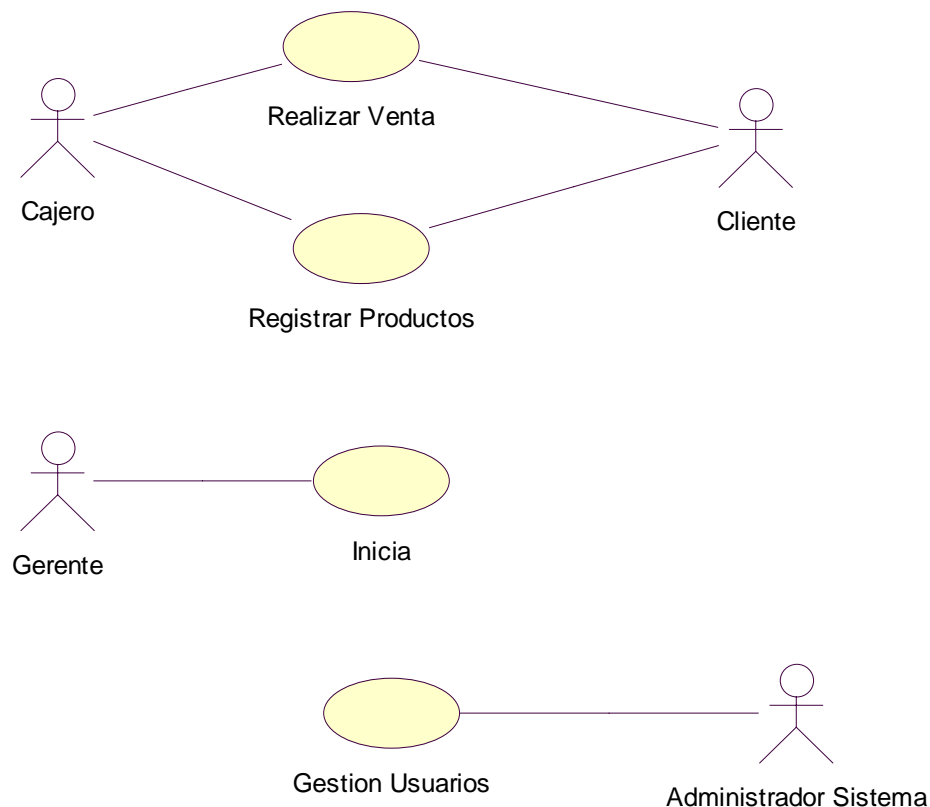
### 3. Modelado de casos de uso

Comenzamos el modelado de un sistema trabajando en la vista de casos de uso, introduciendo a través del explorador:

- los elementos del modelo de casos de uso del negocio (si consideramos conveniente realizar un modelado del negocio)
- los elementos del modelo de casos de uso del sistema

Vamos a crear el modelo de casos de uso del sistema. Para ello, primero crearemos a través del explorador los elementos del modelo de casos de uso (actores y casos de uso), y después crearemos los diagramas de casos de uso que mostrarán el modelo. También se podrían crear directamente al editar, pero conviene separar la idea de elemento de un modelo de elemento de un diagrama.

Comenzamos interactuando directamente con el explorador, para **crear los actores y casos de uso que hay en el diagrama de casos de uso de la Figura 1**. (Sólo los actores y casos de uso, no el diagrama.)



**Figura 1.** Diagrama de Casos de Uso TPV

Para crear un actor en el explorador:

1. Clic con el botón derecho del ratón sobre la carpeta Use Case View.
2. Seleccionar New:Actor. En el explorador aparece un nuevo actor denominado NewClass.
3. Selecciónalo y cámbiale el nombre por defecto por el nombre apropiado.
4. Haz doble clic sobre el actor, y observa cómo se edita como una clase, con sus atributos y operaciones, pero con el estereotipo <<actor>>.

Para documentar un actor:

1. Si la ventana de documentación no es visible, ábrela seleccionando el menú View y activando la opción Documentation.
2. Selecciona el actor en el explorador.
3. Pon el cursor en la ventana de documentación y escribe el texto que describa el actor.

Para crear un caso de uso a través del explorador, y añadirle una breve descripción textual:

Sigue los mismos pasos que para crear un actor, pero con New:Use Case.

### **Asocia al caso de uso *Realizar Venta* un fichero que lo describa utilizando una plantilla.**

Para ligar documentos con plantillas de casos de uso a casos de uso:

1. Botón derecho sobre el caso de uso, opción Open Specification/Files.
2. Botón derecho /Insert File...
3. Seleccionar el archivo correspondiente (Word, Excel, etc.).

Por comodidad, se puede hacer que un mismo archivo contenga las plantillas de varios casos de uso.

### **Crear un diagrama de casos de uso.**

Cada sistema tiene normalmente un Main Use Case Diagram, que muestra los actores junto con las principales casos de uso asociados a cada uno de ellos. Por supuesto, se pueden construir otros diagramas de casos de uso con otros objetivos, por ejemplo:

- para mostrar todos los casos de uso de un paquete
- para mostrar todos los casos de uso para un actor determinado
- para mostrar todos los casos de uso que son implementados en una iteración
- para mostrar un caso de uso y todas sus relaciones

Para crear un diagrama de casos de uso:

Con botón derecho sobre la carpeta correspondiente, `New: Use Case Diagram`.

Crea ahora un diagrama denominado *Diagrama Casos de Uso TPV*, y arrastra sobre él los actores y casos de uso de la Figura 1, los cuales has creado mediante el explorador.

Por supuesto, los actores y casos de uso se pueden crear también directamente en la ventana usando la barra de herramientas.

### **Relaciones entre casos de uso.**

Tenemos tres relaciones principales entre casos de uso en Rose: inclusión, extensión y generalización

Para crear relaciones entre casos de uso:

1. Pinchar el icono `Dependency or Instantiates`
2. Arrastrar la línea en el sentido adecuado
3. Doble clic sobre la línea para hacer visible `Specification/General`
4. Seleccionar el valor adecuado en el campo `Stereotype`

Puedes poner o quitar la flecha de la relación con botón derecho/`Navigable` sobre la relación, cerca del extremo adecuado.

### **Establece relaciones entre casos de uso.**

Crea un caso de uso *Pago con Tarjeta* y establece una relación `include` de *Realizar Venta* a este nuevo caso de uso. Cambia después la relación por una `extend`.

### **Añadir nuevas herramientas a la barra**

Es posible añadir nuevas herramientas a la barra de herramientas del editor. Para ello, coloca el cursor sobre la barra de herramientas y pincha botón derecho, selecciona `Customize`. Prueba a añadir la asociación no dirigida.

### **Borrar elementos (del diagrama o del modelo)**

Con la tecla `Supr` (botón derecho/`Edit/Delete`) se borra un elemento del diagrama actual, pero no del modelo. (Recuerda que el diagrama no es más que una “vista” o proyección del modelo.) Para borrar dicho elemento del modelo, usa botón derecho/`Edit/Delete from model`.

## 4. Modelado de clases

Primero vamos a crear un **modelo conceptual**, a partir del cual crearemos luego un **modelo de clases del análisis**.

---

### Crea el modelo conceptual para el sistema TPV.

Dentro de la vista de casos de uso, añade el diagrama de clases de la Figura 2, que denominaremos *Modelo Conceptual TPV*:

- Introduce clases
  - En el explorador, botón derecho/New:Class, o bien directamente a través de los iconos de la barra de herramientas.
- Puedes documentarlas a través de la *ventana de documentación*.

#### Colisiones de nombres.

El problema de la colisión de nombres aparece si quieres crear, por ejemplo, una clase **Gerente** en la *Vista de Casos de Uso*, donde ya tienes el actor **Gerente**, ya que los actores son clases estereotipadas. Para evitarlo, puedes poner las clases y el diagrama del modelo conceptual en un subpaquete denominado “Modelo Conceptual”.

También se pueden importar clases desde otras localizaciones. En particular, en ocasiones vemos ejemplos en los que en un diagrama de clases se ha importado un actor. Por ejemplo, puedes arrastrar el **Gerente**, que hemos definido como un actor. Observarás como aparece dibujado como un actor, debido a que tiene asociado el estereotipo <<actor>>.

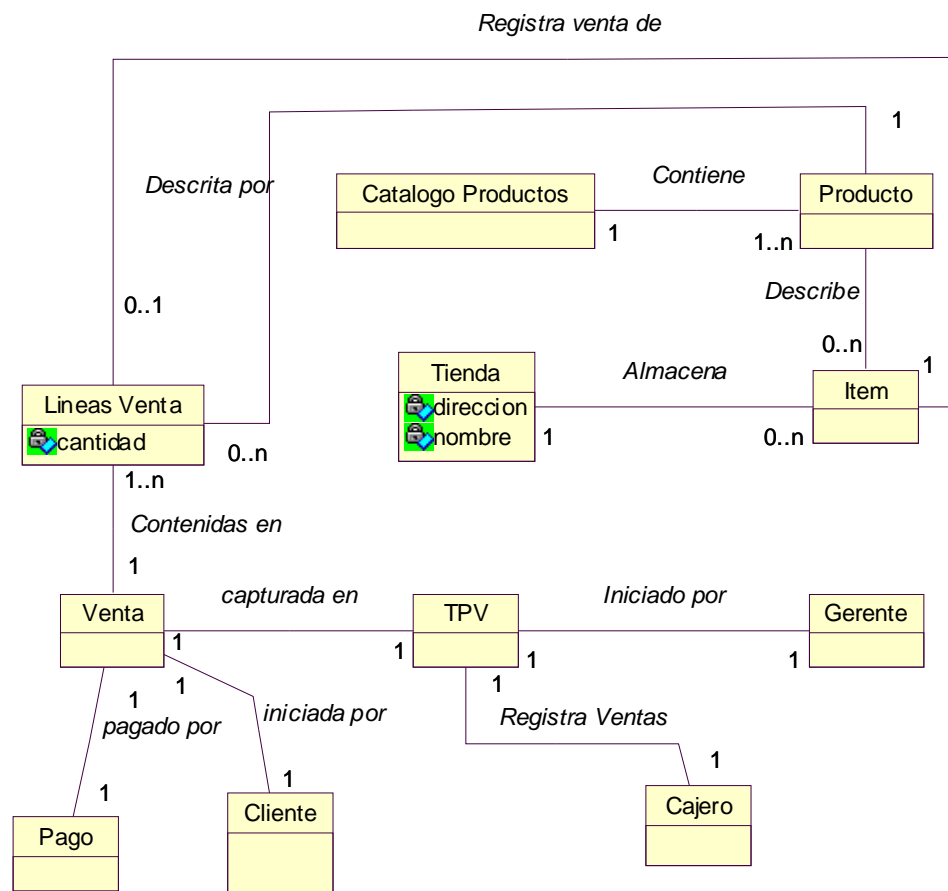
Un estereotipo se puede visualizar de varias maneras, según lo indiques con botón derecho sobre el elemento deseado en el diagrama de clases en Options/Stereotype Display. Cuando importas un clasificador desde otra vista, puedes cambiar su apariencia con esta opción.

Por ejemplo, cuando copias un actor del modelo de casos de uso al modelo conceptual, usa esta opción para que aparezca como una clase (*none*) en lugar de como una figura de actor (*icon*)

Esta aproximación es cómoda, pero hay un problema: conceptualmente el actor **Gerente** y la clase **Gerente** son elementos distintos, un actor es una clase estereotipada. Por ello, recomendamos crear un actor y una clase distintos para cada actor del que se tenga que almacenar información en el modelo.

- Establece relaciones entre clases:
  - En este ejemplo usa sólo *Undirectional Association*
- Indica el nombre y/o roles de algunas asociaciones.
- Indica las cardinalidades
  - Opción *Multiplicity* haciendo clic con el botón derecho cerca del extremo deseado de la relación.
- Recuerda que en este modelo no se indica la *navegabilidad* de las relaciones

- Para poner o quitar la flecha de navegabilidad, puedes usar la opción *Navigable* haciendo clic con el botón derecho cerca del extremo deseado de la relación.
- Examina el resto de propiedades de las asociaciones (como *derived*) y de los roles (como exportación: *public*, *private*, *protected*).
- Añade atributos a cada clase
  - En el explorador, sobre la clase botón derecho/*New:Attribute*.
  - Observa cómo cambia el icono asociado al atributo según sea *public*, *protected*, *private* o *implementation* (botón derecho/*Open Specification/General/Export Control*). Examina el resto de opciones de especificación de un atributo.



**Figura 2.** Modelo conceptual TPV.

- Ya que en este modelo las operaciones no son relevantes, oculta el compartimento de operaciones en cada clase
  - botón derecho sobre la clase en el diagrama, *Options/Supress Operations*.

---

## Crea el modelo de clases del análisis para el sistema TPV

Dentro de la carpeta *Logical View*, crea un diagrama de clases denominado *Clases del Analisis*.

- Incluye las clases *TPV*, *Item*, *Venta*, *Pago*, *Tienda*, *LineaVenta*, *CatalogoProducto* y *Producto*, que ya creaste durante el modelado de los requisitos, arrastrando con el ratón desde el explorador. (Observa cómo aparecen las relaciones entre clases que se habían definido en la vista de casos de uso.)
- También puedes añadir clases a un diagrama a través del menú *Query*.

La *visibilidad* de una clase (from “*vista\_de\_definición*”) indica el paquete donde está definida la clase. En Rose se puede establecer si se muestra (o no) la visibilidad de una clase:

- (para todas las clases) en *Tools/Options/Diagram*, en *Show Visibility*
- (para sola una clase) se establece con botón derecho/*Options/Show Visibility*.

Si no se desea mantener el modelo conceptual se puede crear el modelo de clases del análisis modificando directamente el diagrama de clases que lo representa.

---

En el modelo conceptual hemos introducido algunos atributos a las clases. Ahora vamos a **introducir operaciones a las clases**.

Sobre la clase *Venta*, botón derecho/*Open specification.../Operations*

Por ejemplo, define la siguiente operación en la clase *Venta*:

```
public void crearLineaVenta( Producto spec, int cantidad )
```

En *Tools/Options/Diagram* se puede indicar cómo se mostrarán las operaciones en las clases de un diagrama: sólo nombre, nombre y tipos de parámetros o completas.

---

Define *Pago* como **clase abstracta** con dos subclases *Pago Tarjeta* y *Pago Efectivo*.

- Sobre la clase *Pago*, botón derecho/*Open specification.../Detail*, activar la opción *Abstract*
- Examina el resto de opciones de la especificación de una clase: control de exportación (*public*, *protected*, *private*, *implementation*), persistencia, etc.

---

Introduce **estereotipos** para las clases (por ejemplo: *boundary*, *entity*, *control*) y observa los resultados.

- En el explorador o en el diagrama, con botón derecho/*Open Specification/General/Stereotype*.

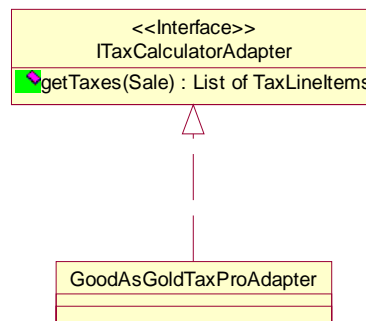


- Cambia la forma de visualizar el estereotipo con botón derecho sobre la clase en el diagrama/Options/Stereotype Display. Un estereotipo se puede mostrar de diferentes formas: *icon*, *label*, *decoration* o *none*.

---

Crea el diagrama de la Figura 3, en el que **una clase implementa una interfaz**.

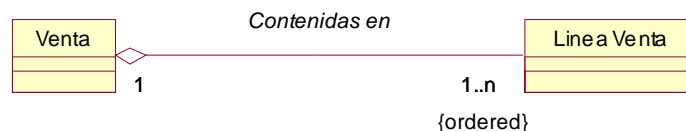
- Usa el estereotipo <<interface>> y muéstralo como *label*; utiliza una relación *realizes*.
- Para mostrar la interfaz mediante su icono, muestra el estereotipo mediante *icon*.
- Introduce la operación que aparece en la 3 y observa las distintas opciones de especificación de una operación.
- Observa que Rose propaga las operaciones de la interfaz en la clase que la implementa, si bien no las muestra. (Ocurre lo mismo con la relación de generalización.)



**Figura 3.** Una clase que implementa una interfaz.

---

En el modelo conceptual de la Figura 2, crea una **agregación** como la de la figura siguiente:



**Figura 4.** Un ejemplo de agregación.

- *Aggregation* – usando botón derecho/Open Specification.../Role Detail/Aggregate sobre una asociación.
- También puedes personalizar la barra de herramientas para incluir el icono de agregación, con el botón derecho sobre la barra.

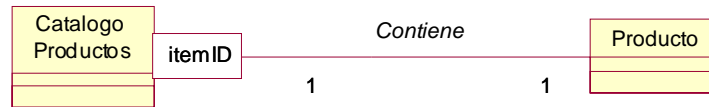
Realmente, la agregación anterior es más propiamente una **composición**:

- Para indicar la composición, botón derecho sobre la línea, cerca de LineaVenta, y en Containment of LineaVenta, seleccionar By Value.

---

En el modelo conceptual de la Figura 2, define ahora una **asociación calificada**, como la que se muestra en la siguiente figura:

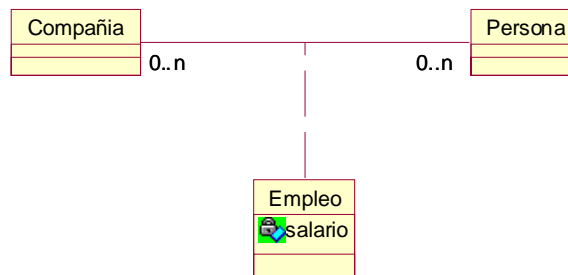
- botón derecho sobre la asociación, New Key/Qualifier



**Figura 5.** Ejemplo de asociación calificada.

---

Define una **clase asociación** como la de la siguiente figura:

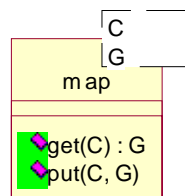


**Figura 6.** Un ejemplo sencillo de clase asociación.

- Crea la clase asociación, con sus atributos y operaciones.
- Usa el icono *Association Class* y arrástralo desde la clase asociación *Empleo* hasta la relación que modifica.

---

Crea una **clase parametrizada** como la de la siguiente figura:



**Figura 7.** Una clase parametrizada.

1. Introduce una nueva clase
  2. Sobre la nueva clase, con botón derecho/Open Specification.../General, poner como Type ParameterizedClass y pulsa OK
  3. Los argumentos formales de la clase se introducen después con botón derecho/Open Specification.../Detail, usando botón derecho/insert sobre Formal Arguments
- Las clases instanciadas se crean con el tipo *InstatiatedClass*, dándole a los parámetros *actuales* los valores adecuados. Entre la clase parametrizada y sus instancias se pone una relación de tipo *Dependency or instantiates*.

## 5. Modelado de interacciones entre objetos

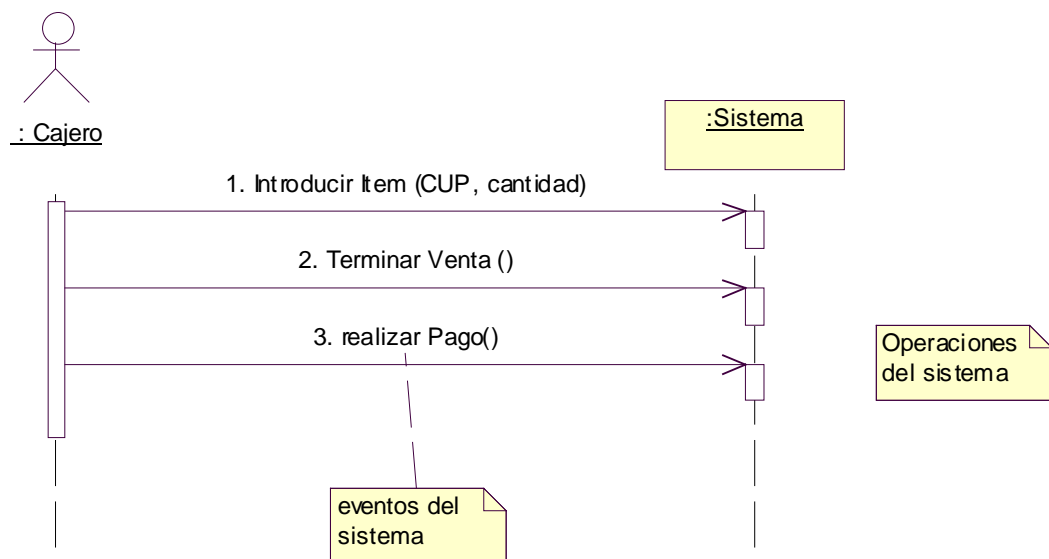
Primero vamos a crear un **diagrama de secuencia para un diagrama de secuencia del sistema** (de acuerdo al proceso de Larman visto en clase) y luego el **diagrama de secuencia** y el **diagrama de comunicación** (denominados **diagramas de colaboración** en versiones anteriores de UML) que corresponden a una de las operaciones del sistema del diagrama de secuencia de sistema.

---

**Crea un Diagrama de Secuencia del Sistema** para el caso de uso *Realizar Venta*.

Como sabemos, los diagramas de secuencia del sistema son diagramas de interacción de tipo diagrama de secuencia. Denominaremos a este diagrama **DSS1** (ver Figura 8 ).

- Crea el diagrama desde el explorador con botón derecho sobre el caso de uso /New:Sequence Diagram. De esta forma lo ligamos al caso de uso.
- Ábrelo con doble clic.
- Selecciona el actor en el explorador, y arrástralo al diagrama.
- Selecciona el icono *Object* de la barra de herramientas, e introduce un objeto anónimo y cuya clase es **Sistema**. Este objeto representa al sistema a crear. Otra alternativa hubiera sido crear la clase con el explorador y arrastrarla al editor.
- Usa el icono *Object Message* para introducir los mensajes.

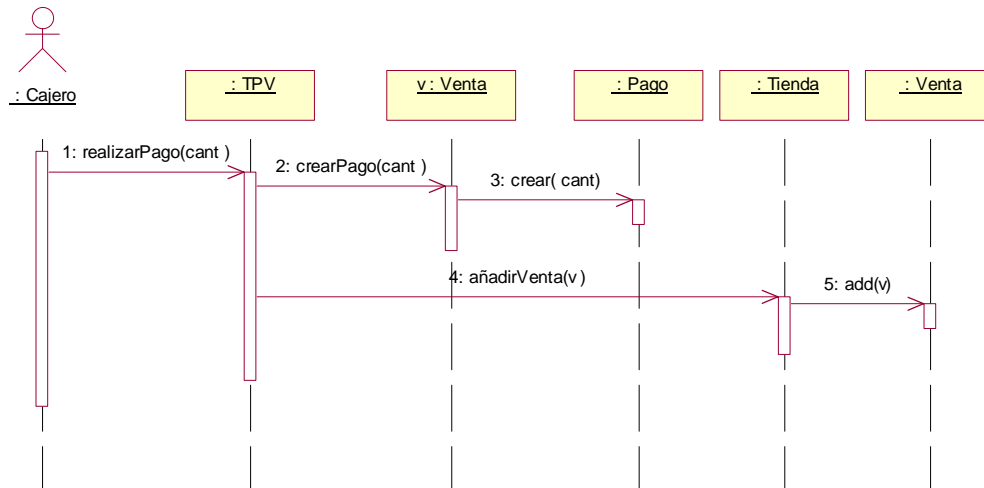


**Figura 8. Diagrama de secuencia del sistema DSS1**

---

**Crea un diagrama de secuencia** para una operación.

Crea el diagrama de secuencia de la figura siguiente.



**Figura 9.** Diagrama de secuencia *realizarPago()*

- Crea el diagrama desde el explorador con botón derecho sobre *Logical View*.
- Ábrelo con doble clic.
- Selecciona las clases en el explorador y arrástralas al diagrama en el orden en que aparecen los objetos de las líneas de vida.
- Usa el icono *Object Message* para introducir los mensajes.

Puedes quitar la numeración del diagrama (Tools/Options/Diagrams /Sequence numbering) y quitar los focos de control (Tools/Options/Diagrams/Focus of control).

---

**Transforma el diagrama de secuencia anterior en un diagrama de colaboración.**

- Se puede crear automáticamente el correspondiente diagrama de colaboración mediante Browse/Create Collaboration Diagram F5. Después tienes que reordenar los elementos del diagrama.
- Nótese que el icono de múltiples instancias de una clase no se muestra en los diagramas de secuencia, pero sí en un diagrama de colaboración.

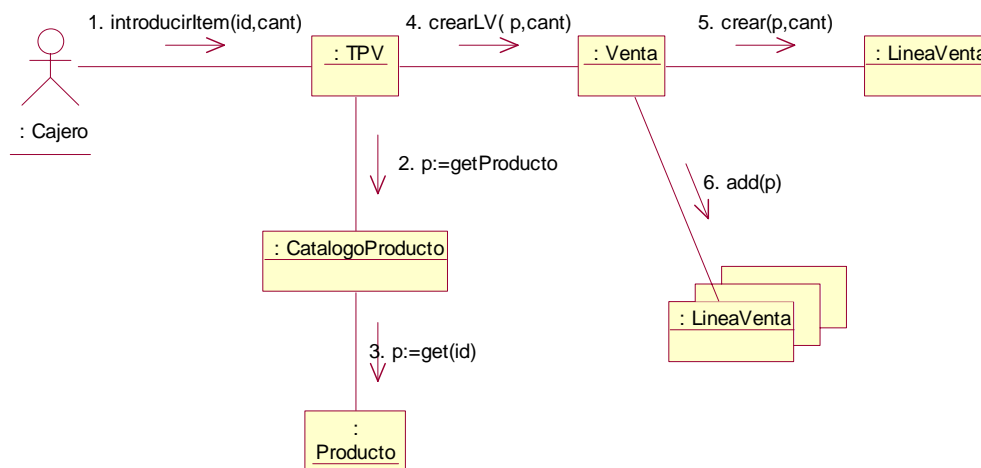
Ahora los diagramas de secuencia y de colaboración están sincronizados, y un cambio en uno se refleja automáticamente en el otro.

---

**Crea un diagrama de colaboración** para una operación de un caso de uso.

Vamos a crear el diagrama de colaboración mostrado en la Figura 10, para la operación del sistema *introducirItem*.

- En el explorador, con botón derecho sobre la vista *Logical View*, crea un diagrama de colaboración subordinado denominado *IntroducirItem*.
- Establece que las interacciones utilizarán la numeración jerárquica (Tools/Options/Diagrams/Hierarchical Messages).
- Examina las opciones de visibilidad de los enlaces (*unspecified, field, parameter, local, global*), y de compartición (*shared*).
- Examina las opciones de sincronización de los mensajes (botón derecho /Open Specification/Detail).



**Figura 10.** Diagrama de colaboración *IntroducirItem*.

Para añadir operaciones a una clase:

- Pulsa con el botón derecho sobre cualquiera de los mensajes del diagrama de colaboración, y observa como aparece la opción **<new operation>**. Con esta opción puedes crear automáticamente una operación a partir del mensaje seleccionado. La operación aparece automáticamente en la clase correspondiente.
- Pulsando el botón derecho sobre un mensaje también aparecen las operaciones que se han definido sobre la clase, para acelerar la definición de los mensajes, si se basan en métodos que ya se han definido.

¿Cómo podríamos introducir un nuevo mensaje entre, por ejemplo, el mensaje 3 y el 4? No podemos hacerlo en los diagramas de colaboración, debemos hacerlo a través del correspondiente diagrama de secuencia.

---

## ¿Diagramas de secuencia o de colaboración?

Como discutimos en clase los diagramas de secuencia destacan la ordenación temporal de los mensajes y los diagramas de colaboración la estructura de objetos que intervienen en la interacción y facilitan la distribución espacial. En el caso de Rose, una razón adicional en favor de los diagramas de colaboración es que en esta versión de la herramienta los diagramas de secuencia no soportan los multiobjetos, y los de colaboración sí.

La forma recomendable de trabajar con los diagramas de interacción es *crear* las interacciones con diagramas de secuencia (sin problemas para insertar nuevos mensajes intermedios o para establecer el orden) y generar los diagramas de colaboración a partir de ellos. En el documento de la práctica es más adecuado incluir los diagramas de colaboración.

---

**Se pueden establecer hipervínculos entre diagramas**, a través de las notas:

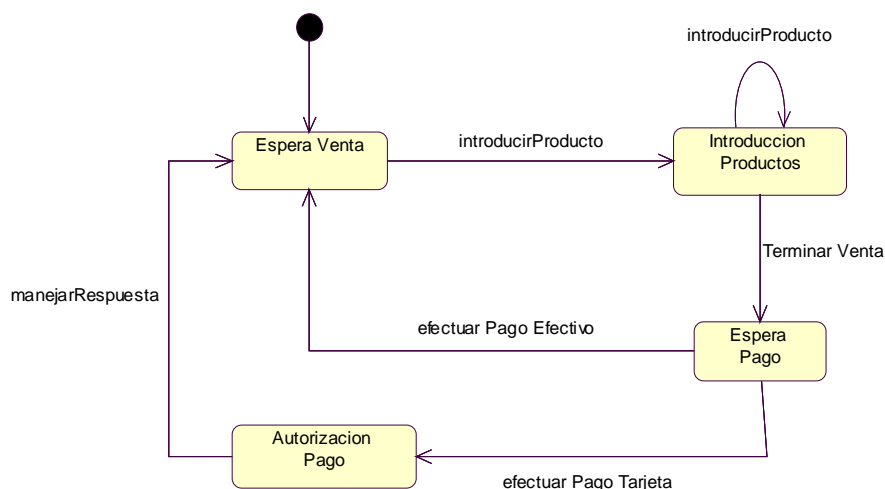
- Inserta una nota en el diagrama de secuencia del sistema DSS1 (Figura 8).
- Selecciona en el explorador el diagrama de secuencia que se quiere enlazar con DSS1 (la colaboración que acabas de crear) y arrástralo sobre la nota.
- Haciendo doble clic sobre la nota se sigue ahora el hipervínculo.

## 6. Máquinas de estado y diagramas de actividades

---

Vamos a **crear un diagrama de estados** (*statechart*) para un caso de uso. Este diagrama describe las posibles secuencias de eventos externos en el contexto del caso de uso. También podríamos crear el diagrama de estados para una clase que tuviera un comportamiento dinámico interesante.

- Para el caso de uso *Realizar Venta*, dentro de la vista de casos de uso, crea el diagrama de estados mostrado en la Figura .
  - Sobre el caso de uso en el explorador, botón derecho/New:Statechart Diagram.
  - La creación del diagrama es directa a partir de la barra de herramientas.
  - Para agregar los detalles de las transiciones entre estados (*acción*, *guarda* y/o *evento* que es enviado), usa botón derecho/Open Specification.../Detail.
  - Los detalles de los estados se introducen también con Open Specification.



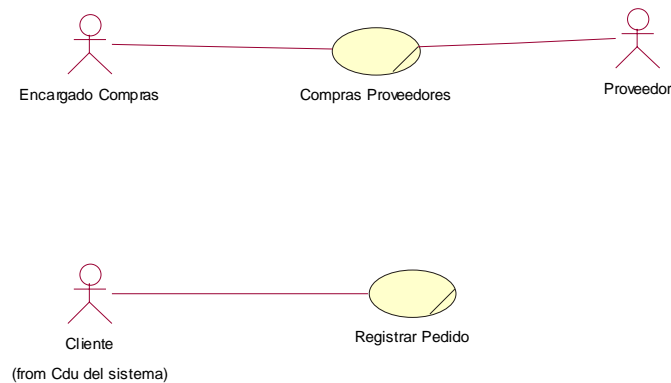
**Figura 11.** Diagrama de estados *Realizar Venta*.

---

**Crea un diagrama de casos de uso para modelar procesos de negocio (casos de uso del negocio y actores del negocio) con el nombre *Modelado del Negocio TPV* (ver Figura 12).**

Crea primero un paquete denominado *Modelo del Negocio TPV*, donde se almacene toda la información relativa a este modelo: usa botón derecho/*New: Package* sobre la carpeta *Use Case View*.

El modelo y los diagramas de casos de uso del negocio se crean de la misma manera que los casos de uso del sistema, pero se usan los estereotipos <<business use case>> y <<business actor>>.



**Figura 12.** Modelado del Negocio TPV.

---

**Crear un diagrama de actividades para el caso de uso del negocio *Registrar Pedido* (ver Figura 13)**

- El diagrama se crea con el nombre *Registrar Pedido*. La creación del diagrama es directa a partir de la barra de herramientas, salvo en algunos pequeños detalles como los que siguen:
  - La introducción de guardas asociadas a transiciones (como [SI] o [NO]) se hace con botón derecho sobre la transición/*Open Specification.../Detail*.
  - Puede que en la barra de herramientas no aparezca el icono de “Objeto”. Si es así, personaliza la barra de herramientas (con botón derecho sobre ella) para añadir este icono.
  - Para que las líneas sean rectas, usa *Format /Line/Style/Rectilinear*.

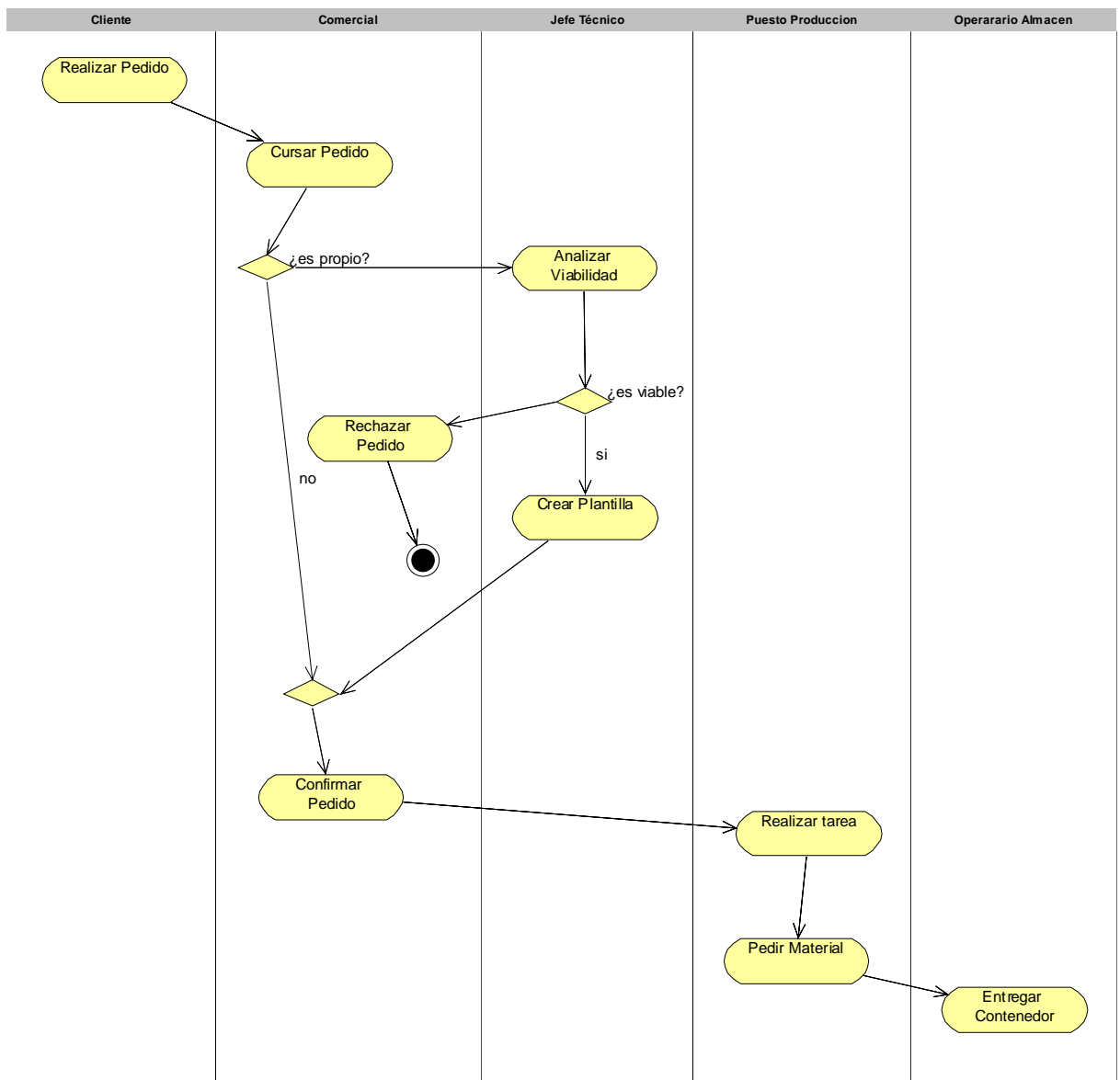


Figura 13. Diagrama de actividades para el caso de uso del negocio *Registrar Pedido*.

## 7. Generación de código

Rose permite realizar generación de código (ingeniería directa) a partir de los modelos, e ingeniería inversa a partir del código. Se pueden generar definiciones de atributos y firmas de los métodos. Rose soporta muchos lenguajes de implementación. En este seminario nos vamos a centrar en la generación de código Java a partir de componentes definidos en la *vista de componentes* de Rose. Esta vista denota la estructura física de archivos de código del sistema. Las clases de Rose se traducen a clases Java y los componentes de Rose a archivos *.java*.

Más información en la ayuda de Rose: *Forward Engineering (Code Generation) in Rose J.*



Los pasos que hay que seguir para generar automáticamente código Java a partir del modelo UML son los siguientes:

1. Asignar clases Java a componentes Java en el modelo. (Puedes asignar varias clases Java al mismo componente Java.)
  - Para crear una clase Java hay dos posibilidades:
    - a. Hacer que Java sea el lenguaje por defecto (para todas las clases):  
Tools/Options.../Notation
    - b. Hacer que una clase concreta sea considerada clase Java: basta con hacerla corresponder con un componente Java.
  - Necesitas crear un nuevo diagrama de componentes en la *vista de componentes*, o usar el denotado por *Main*.
  - Crea un componente en el diagrama de componentes. Para hacerlo componente Java:
    - a. Puedes hacer que Java sea el lenguaje por defecto, o bien...
    - b. Para asignar el lenguaje Java a un único componente, usa botón derecho/Open Specification.../General/Language
  - Asigna la clase al componente Java:
    - a. Arrastra la clase desde el explorador y suéltala sobre el componente Java. (En el explorador, el nombre del componente aparece ahora entre paréntesis después del nombre de la clase.)
    - b. Alternativamente, lo puedes hacer con botón derecho sobre el componente /Open Specification/Realizes
  - Debes tener en cuenta que:
    - a. Cada componente tiene a lo sumo una clase con visibilidad pública.
    - b. Deben coincidir el nombre del componente y el nombre de la clase con visibilidad pública del mismo.
    - c. Para cambiar la visibilidad de una clase, puedes usar sobre la misma botón derecho/Open Specification. Observa como al haber elegido *Java* como lenguaje en lugar de *Analysis*, esta opción ha cambiado. Las funcionalidades anteriores se mantienen en la opción botón derecho/Open Standard Specification.
  - IMPORTANTE: no es obligatorio que tú crees el componente. Si usas botón derecho/Generate Java directamente sobre la clase, Rose creará automáticamente, en la vista de componentes, un componente con el mismo nombre que la clase.
2. Comprobar el *classpath* (Tools/Java/Project Specification/General).
  - Rose J necesita que la variable CLASSPATH apunte a las clases de la API JDK y a tus bibliotecas de usuario. CLASSPATH es una variable del entorno del sistema que no es propia de Rose: varias herramientas y aplicaciones de Java (como la *Java Virtual Machine*) dependen del CLASSPATH del sistema para resolver sus referencias.
  - Para ingeniería inversa, Rose J requiere que CLASSPATH apunte a la biblioteca de clases JDK.
  - Usa la sección *Directories* de la ventana de classpath para añadir directorios que extienden el CLASSPATH. Esta variable

- `Directories` es guardada con el modelo Rose, pero no altera el `CLASSPATH`. Puedes usar `Directories` para indicar en qué directorio quieres que Rose J guarde el código generado.
3. Especificar las propiedades del proyecto que afectan a la generación de código (opcional).
    - `Tools/Java/Project Specification/Detail`
  4. Hacer copia de respaldo del código (opcional).
    - Ten en cuenta que cuando generas código a un archivo `.java` existente, Rose crea un backup del archivo actual usando la extensión `~jav`.
  5. Generar código Java.
    - Selecciona una o más clases o componentes y usa `Tools/Java/Generate Java` o botón derecho sobre el componente `/Generate Java`.
    - Si es la primera vez que generas código para un elemento del modelo, aparecerá un diálogo que permite relacionar paquetes y componentes a los valores del `classpath`.
    - Los errores quedan reflejados en la ventana *Rose Log* (disponible desde el menú `Window`).
    - Alternativamente, Rose J ofrece una *opción de autosincronización* que inicia la generación de código automáticamente cuando se crea o se modifica cualquier elemento Java en el modelo. Se activa esta opción desde `Tools/Java/Project Specification/Detail`.
  6. Ver y extender el código generado.

Observa que como resultado de la generación de código, Rose J marca los métodos con un comentario de la forma `@roseuid string`. Por ejemplo:

```
public class HelloWorldApp {  
    public HelloWorldApp() {}  
  
    /**  
     * @roseuid 364597B40046  
     */  
    public static void main(String[] args) {
```

Rose J usa *roseuid* en el proceso posterior de ingeniería inversa. Se recomienda dejar la etiqueta en su sitio, pero no es obligatorio. (Si se borra el comentario, la documentación de Rose dice que no se perjudicará de forma significativa el posible proceso de ingeniería inversa posterior.)

---

Experimenta con los pasos anteriores creando atributos y operaciones en una clase de la vista lógica, asignándola después a un componente Java y generando el código correspondiente.

## **Generación de informes**

Puedes publicar el modelo en HTML con la opción `Tools/Web Publisher...`