

PRIORIZACION DE CASOS DE USO

Se elabora a través de encuestas a los usuarios del sistema evaluando criterios de priorización

Criterios de Priorizacion Casos de Uso	CP1	CP2	CP3	CP4	CP5	$\sum_{j=1}^n P_j$
CU1	P11	P12	P13	P14	P15	
CU2	P21	P22	P23	P24	P25	
CU3	P31	P32	P33	P34	P35	
CU4	P41	P42	P43	P44	P45	

Criterios de Priorización

CP1 = Caso de uso Base

CP2 = Mejorar la imagen de la institución

CP3 = Incrementar el nivel de satisfacción de los usuarios del sistema

CP4 = Incrementar el nivel de satisfacción de los clientes

CP5 = Proceso importante

NIVEL DE IMPACTO	PUNTAJE
Muy Bajo	1
Bajo	2
Medio	3
Alto	4
Muy alto	5

Criterios de	CP1	CP2	CP3	CP4	CP5	$\sum_{j=1}^n P_j$	$\sum_{j=1}^n P_j / 5$	PRIORIDAD
--------------	-----	-----	-----	-----	-----	--------------------	------------------------	-----------

Priorizacion Casos de Uso								
CU1	2.5	4	3	3.5	4	17.0	3.4	2º
CU2	3	3	4	5	4	19.0	3.8	1º
CU3	4	2.5	3	4	3	16.5	3.3	3º
CU4	3	3	3	4.5	3	16.5	3.3	4º

FASE II : ELABORACION

2.1 Modelo de Análisis.

1º Diagrama de Clases de Análisis

Se terminan los diagramas de clases parciales de cada caso de uso (Requerimientos) y luego se autogenera el diagrama de clases general debiendo contener atributos y tipos de datos.

EJEMPLO: SISTEMA DE VENTAS

Ver ejemplo en Rational Rose

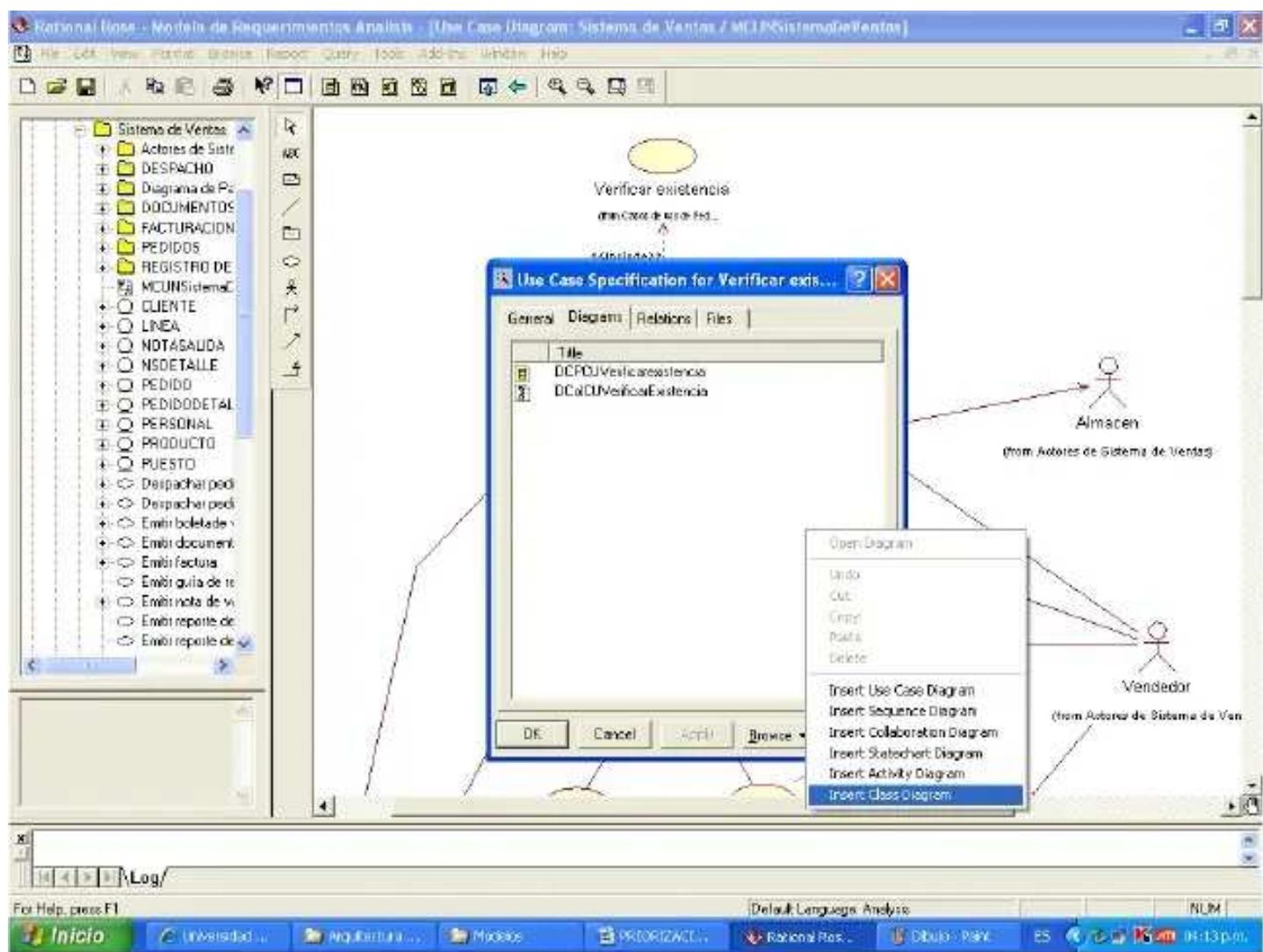


Diagrama Parcial de Clase para CU Seleccionar producto

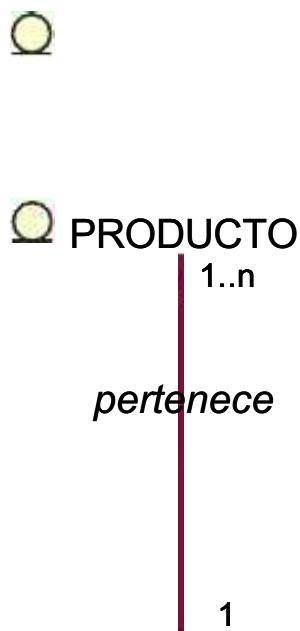


Diagrama Parcial de Clase para CU Atender Pedido

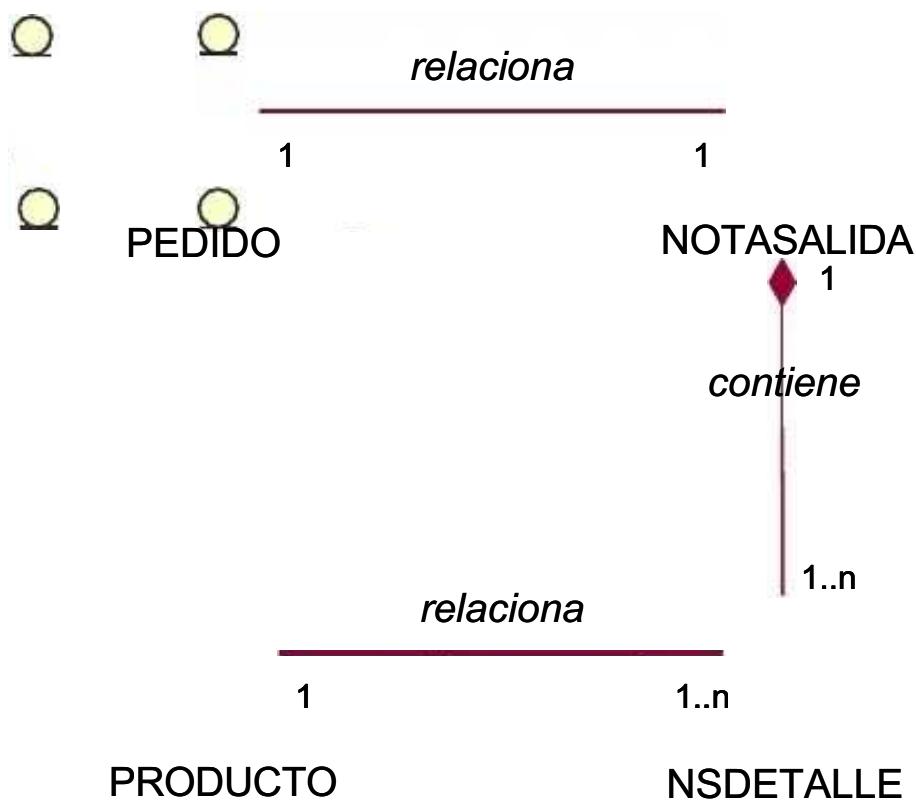


Diagrama Parcial de Clase para CU Emitir Nota de Salida

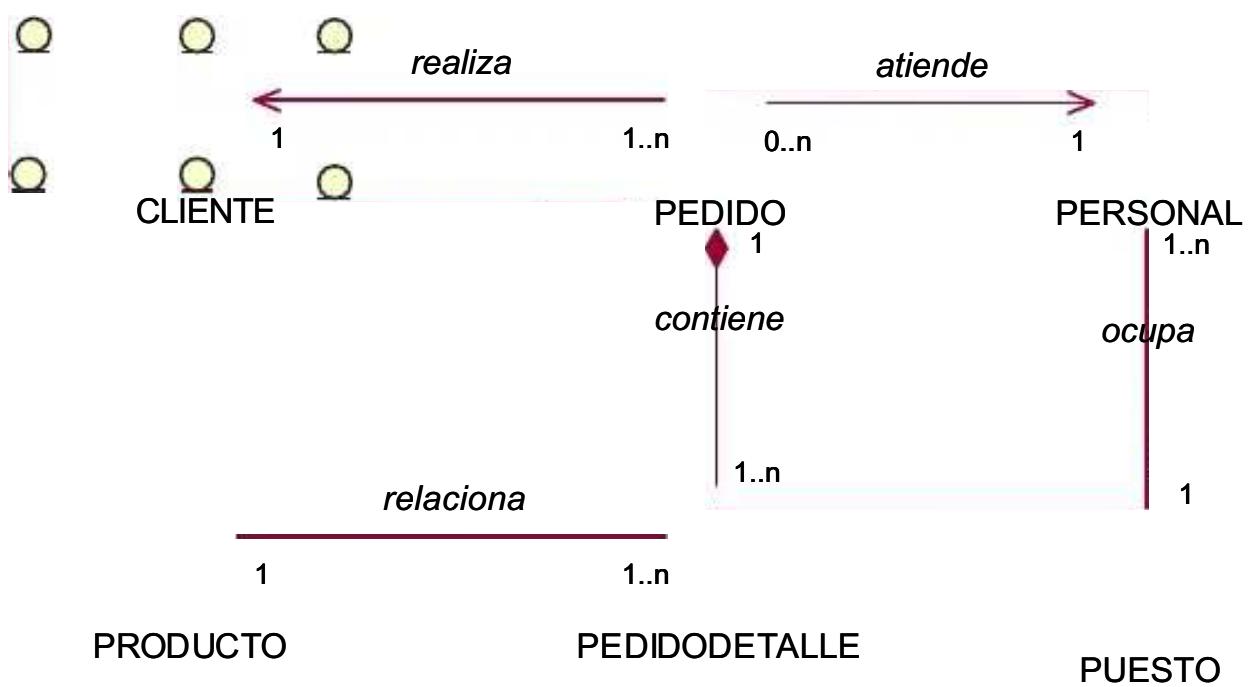


Diagrama Parcial de Clase para CU Registrar venta

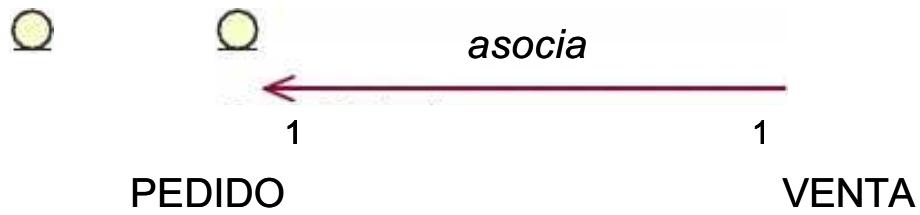


Diagrama Parcial de Clase para CU Registrar venta al contado



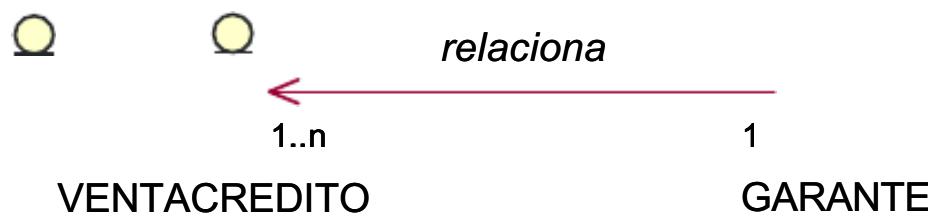
VENTACONTADO

Diagrama Parcial de Clase para CU Registrar venta al crédito



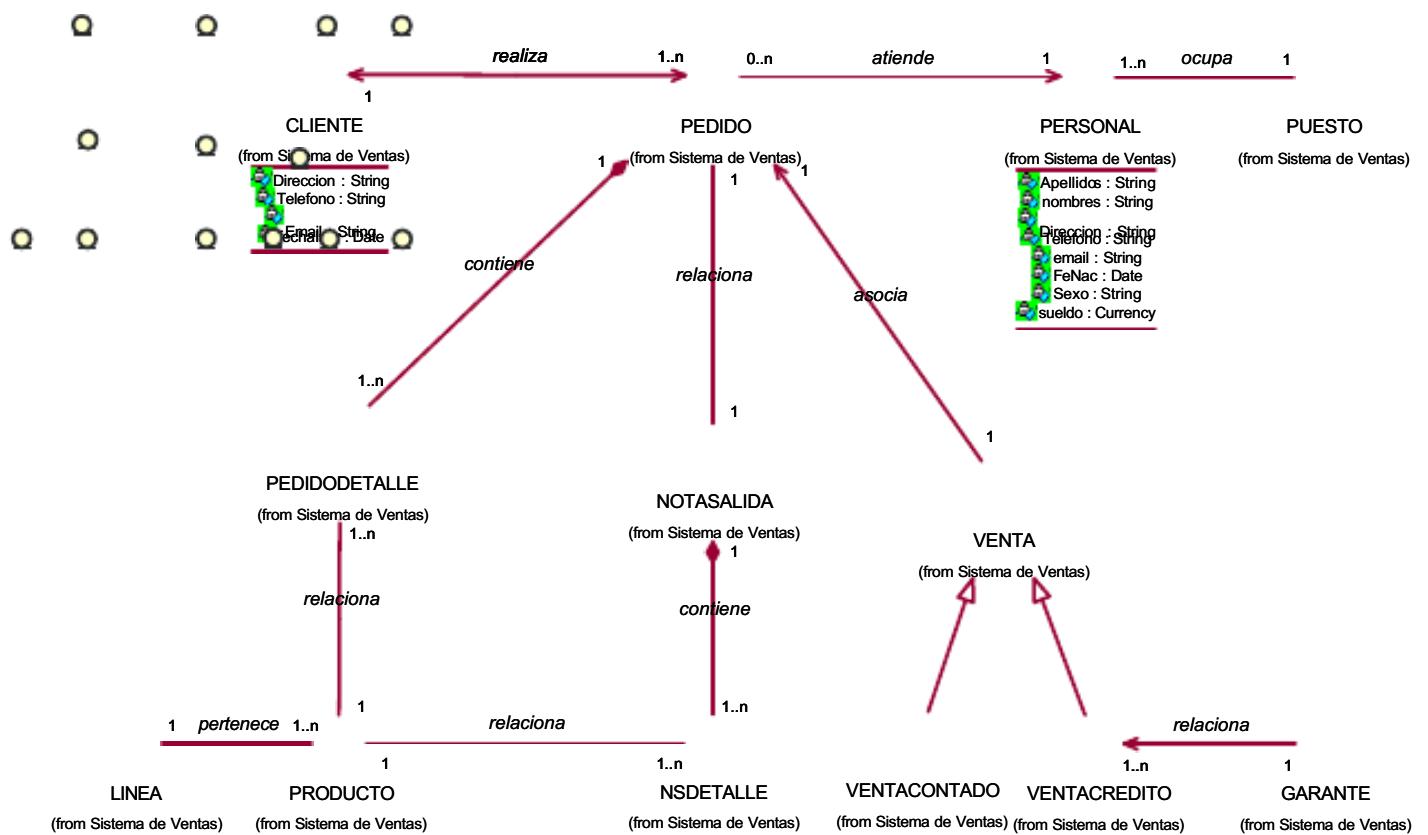
VENTACREDITO

Diagrama Parcial de Clase para CU Registrar Garante



..... Lo mismo para todos los casos de uso

Luego se autogenera el Diagrama de Clases general arrastrando todas las entidades y agregando atributos y tipos de datos. El diagrama se refina a partir del diagrama anterior obtenido

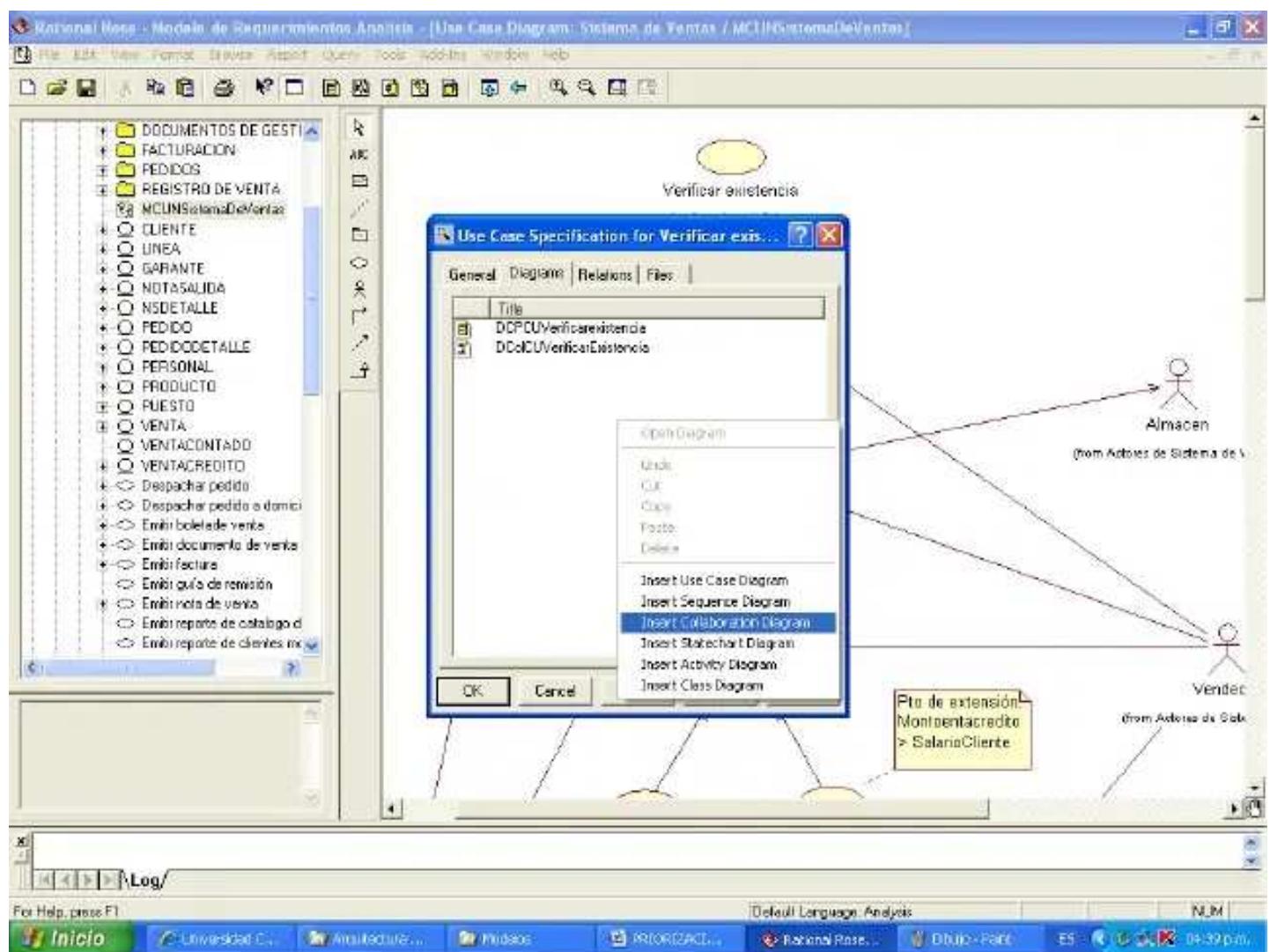


2º Diagramas de Colaboración

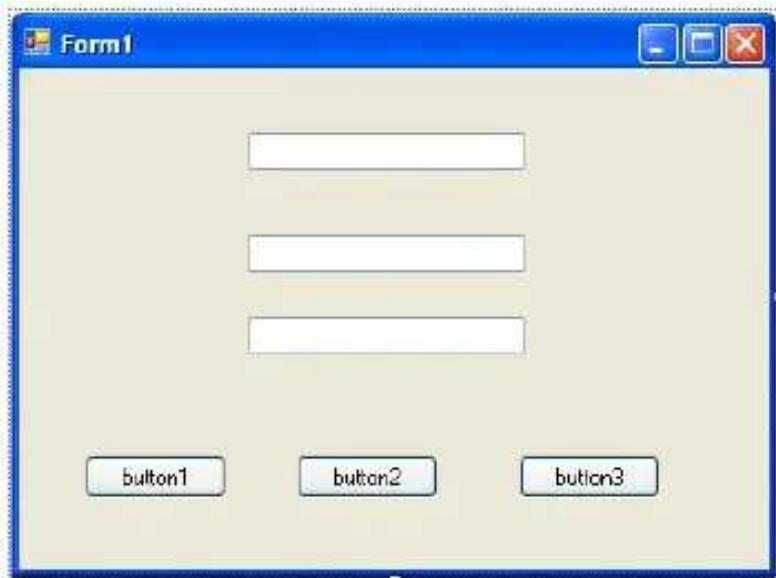
Se elabora por cada caso de uso, partiendo de un diseño de pantalla previo. El objetivo es determinar las funciones y/o procedimientos (con o sin parámetros) que activan los controles (botones), los cuales deberán ser agregados a cada clase en diseño

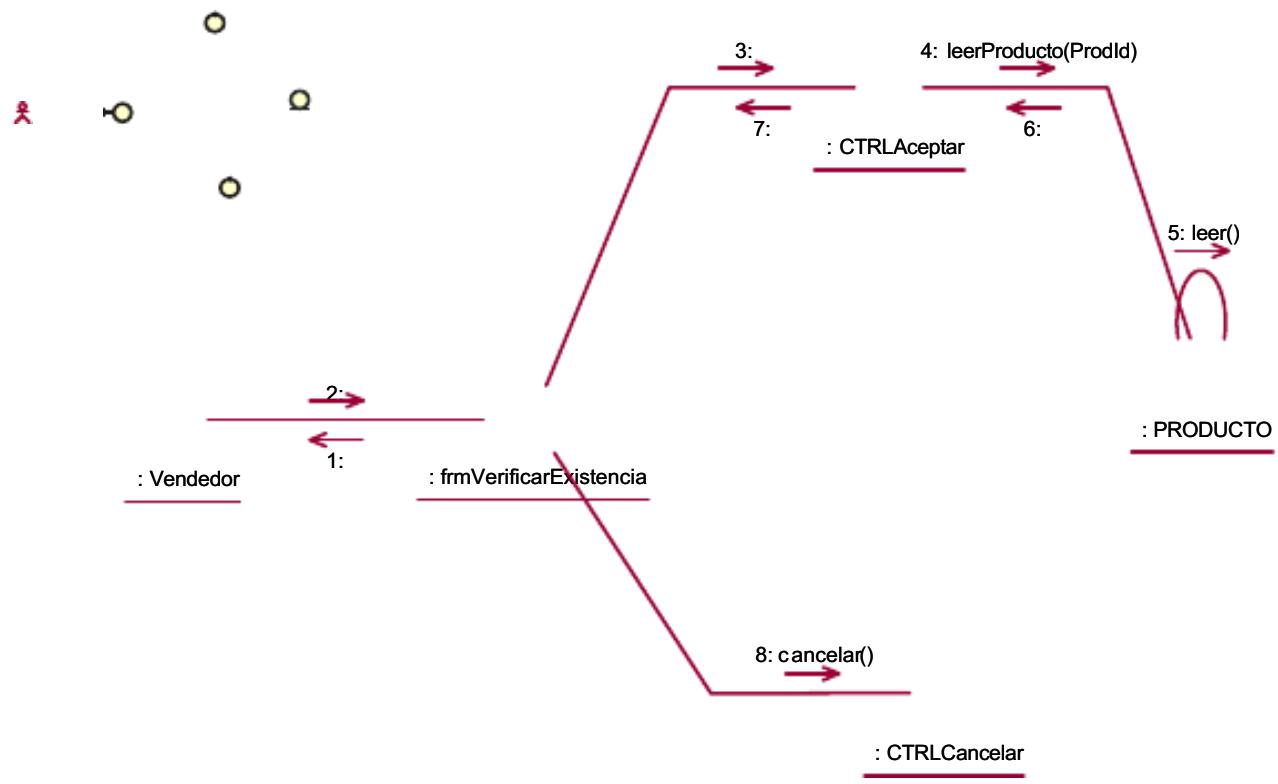
EJEMPLO: SISTEMA DE VENTAS

Se elaboran los diagramas de colaboración para todos los casos de uso



Diseño de Pantalla



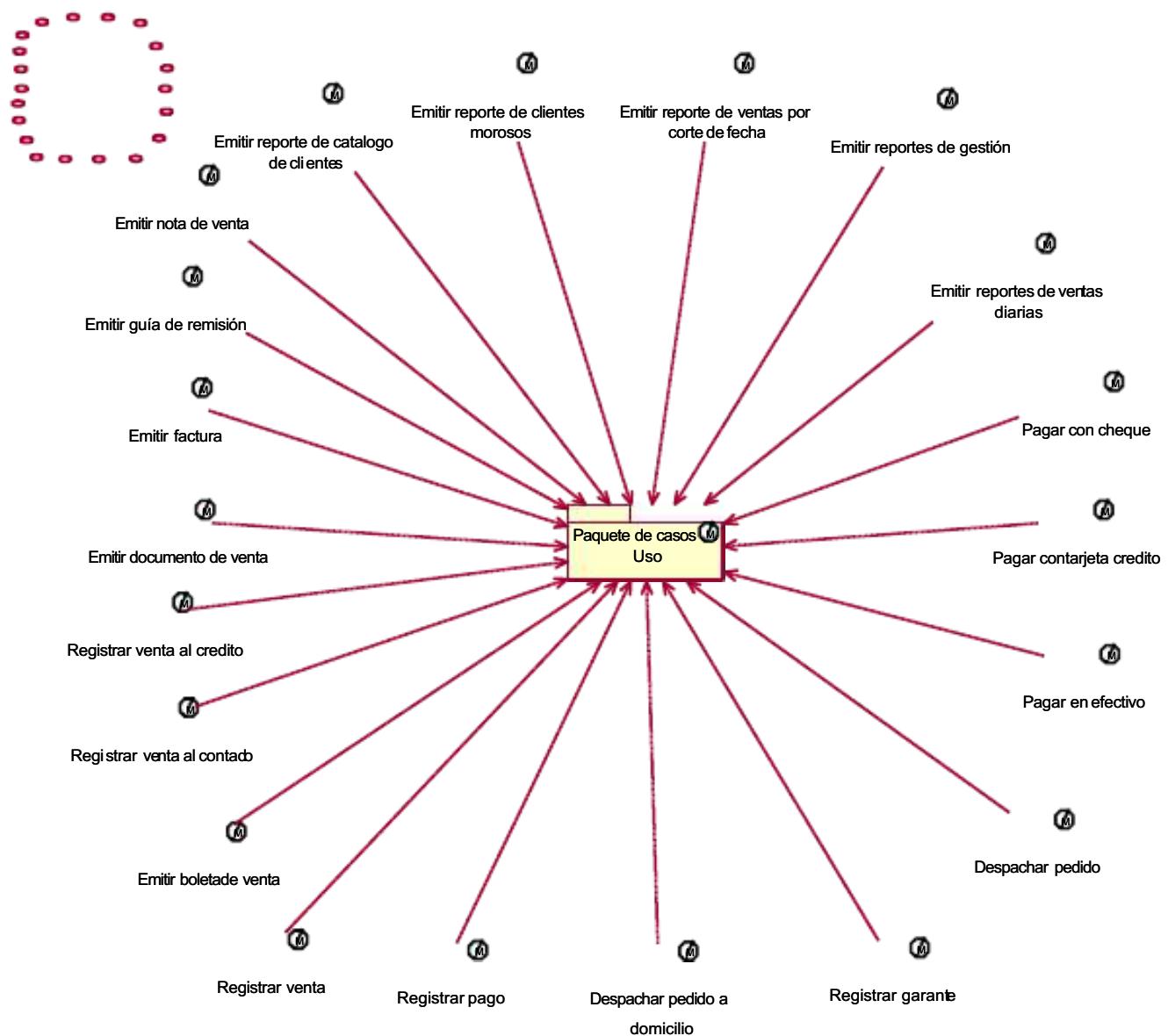
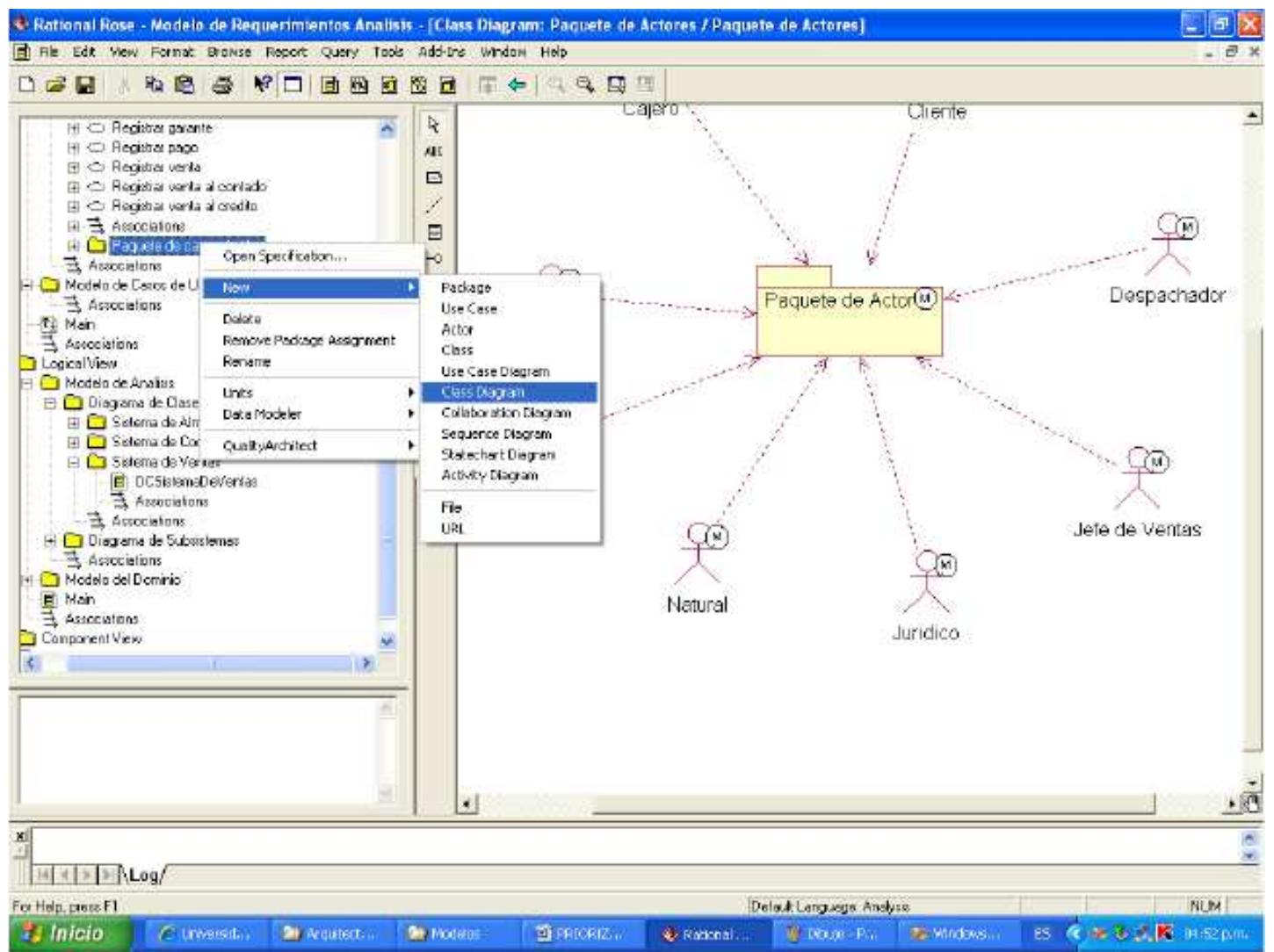


3º Diagramas de Paquetes

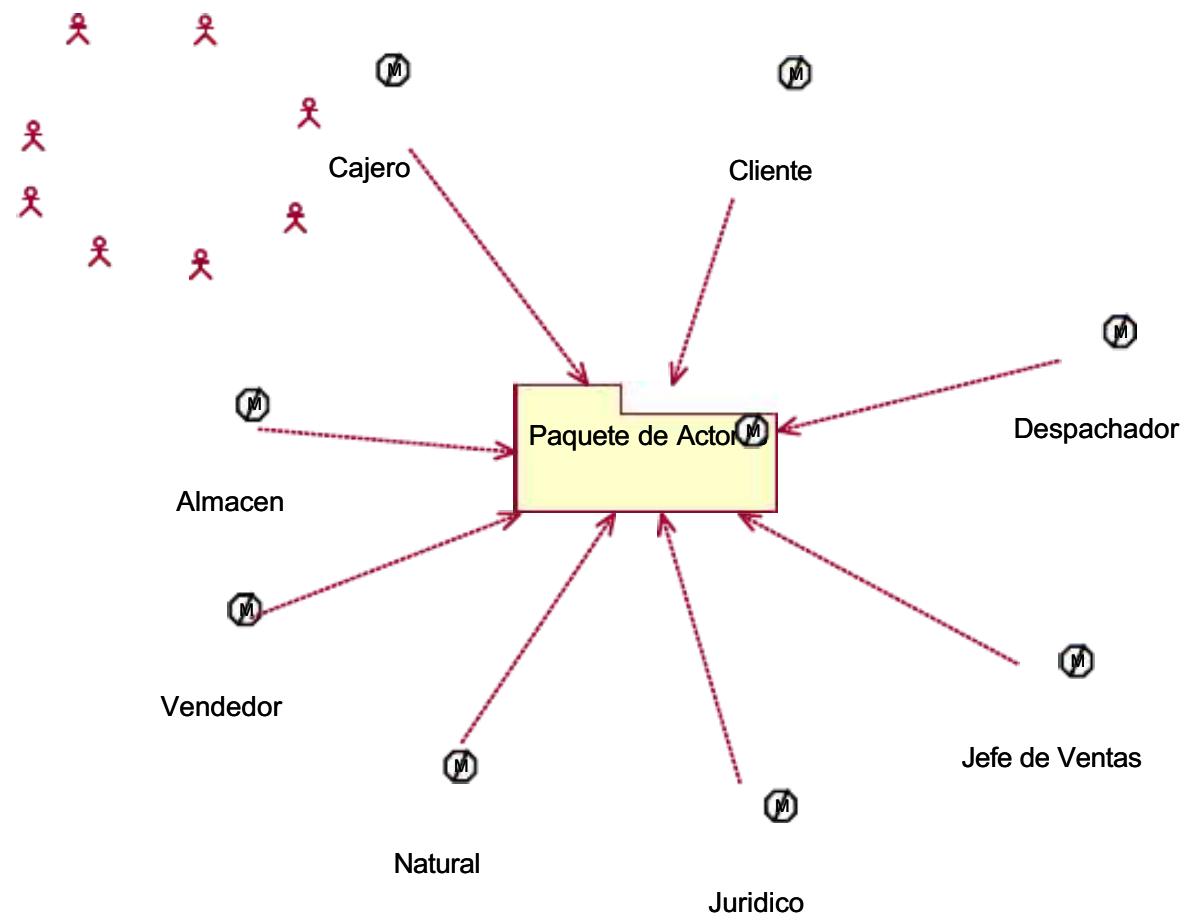
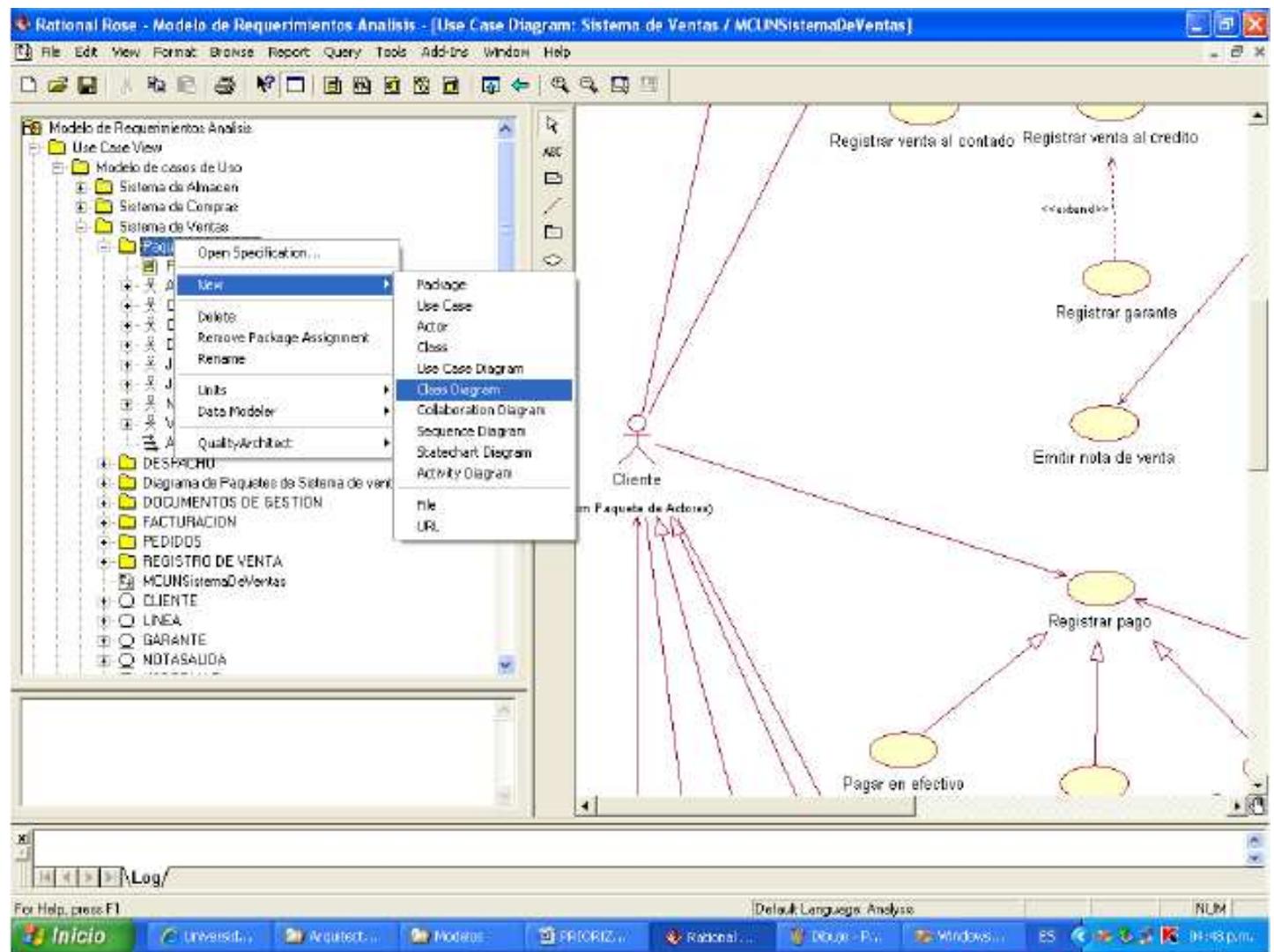
Diagramas de paquetes de casos de uso, de actores, de fichas (boundaries), de controles

EJEMPLO: Ver en Rational Rose

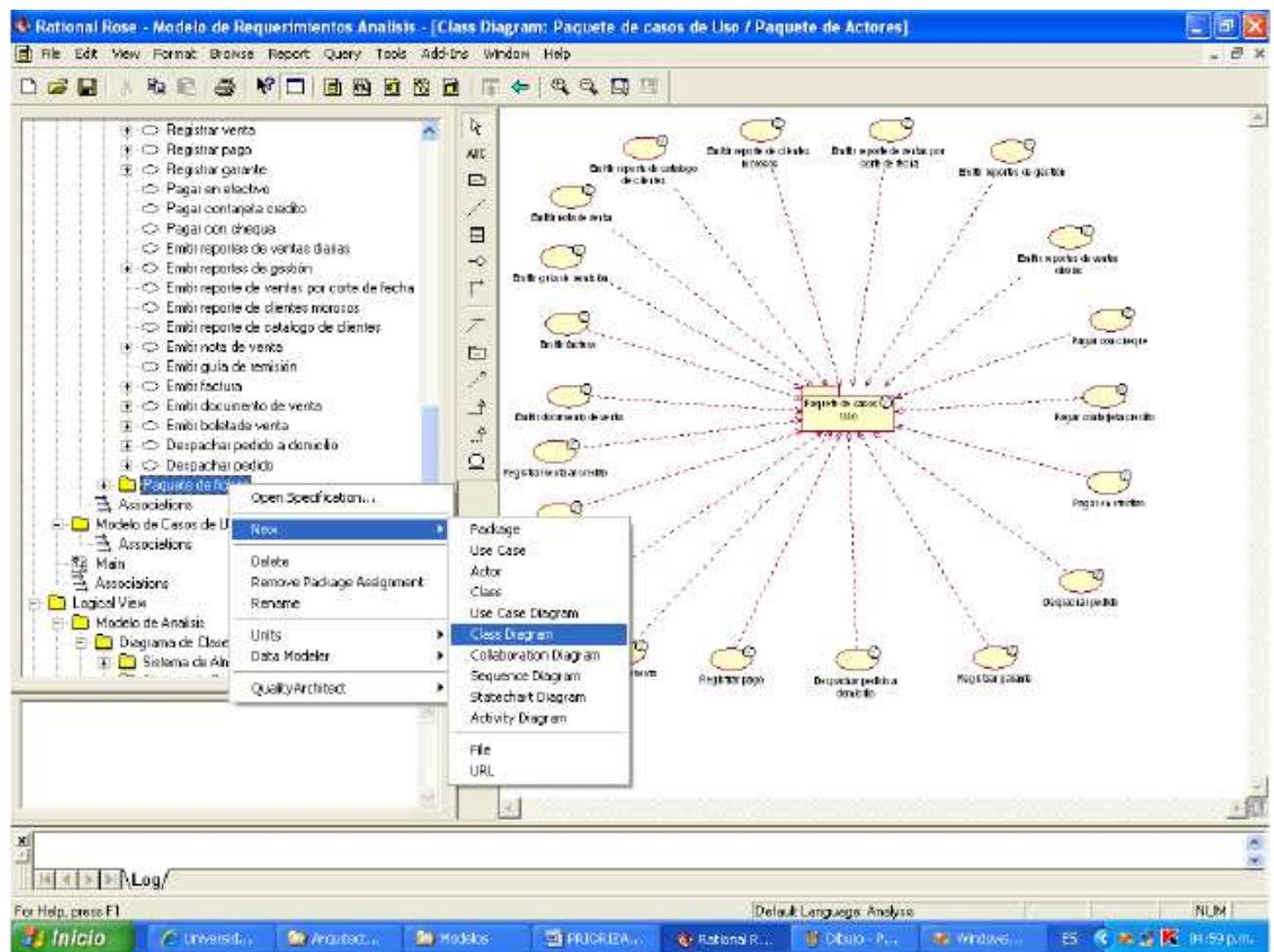
A. Paquetes de casos de Uso



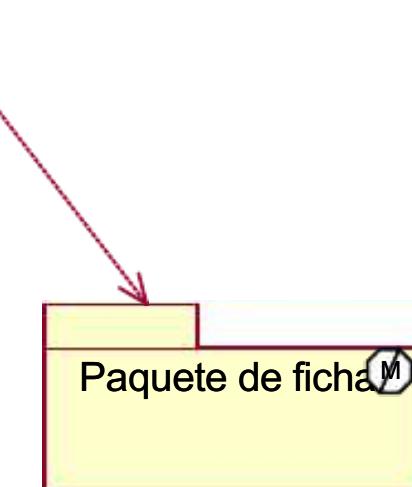
B. Paquetes de Actores



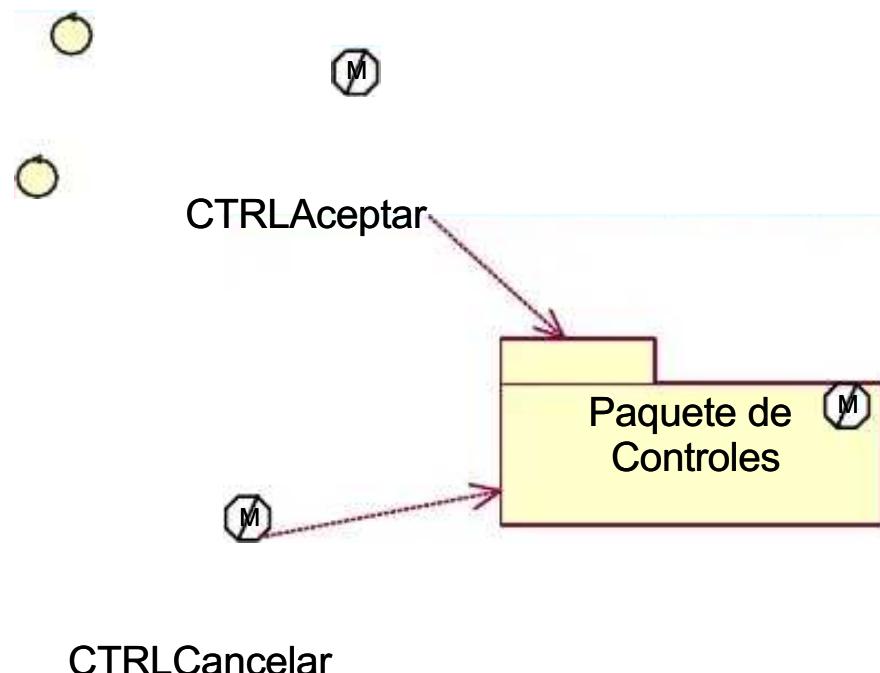
C. Paquete de Fichas



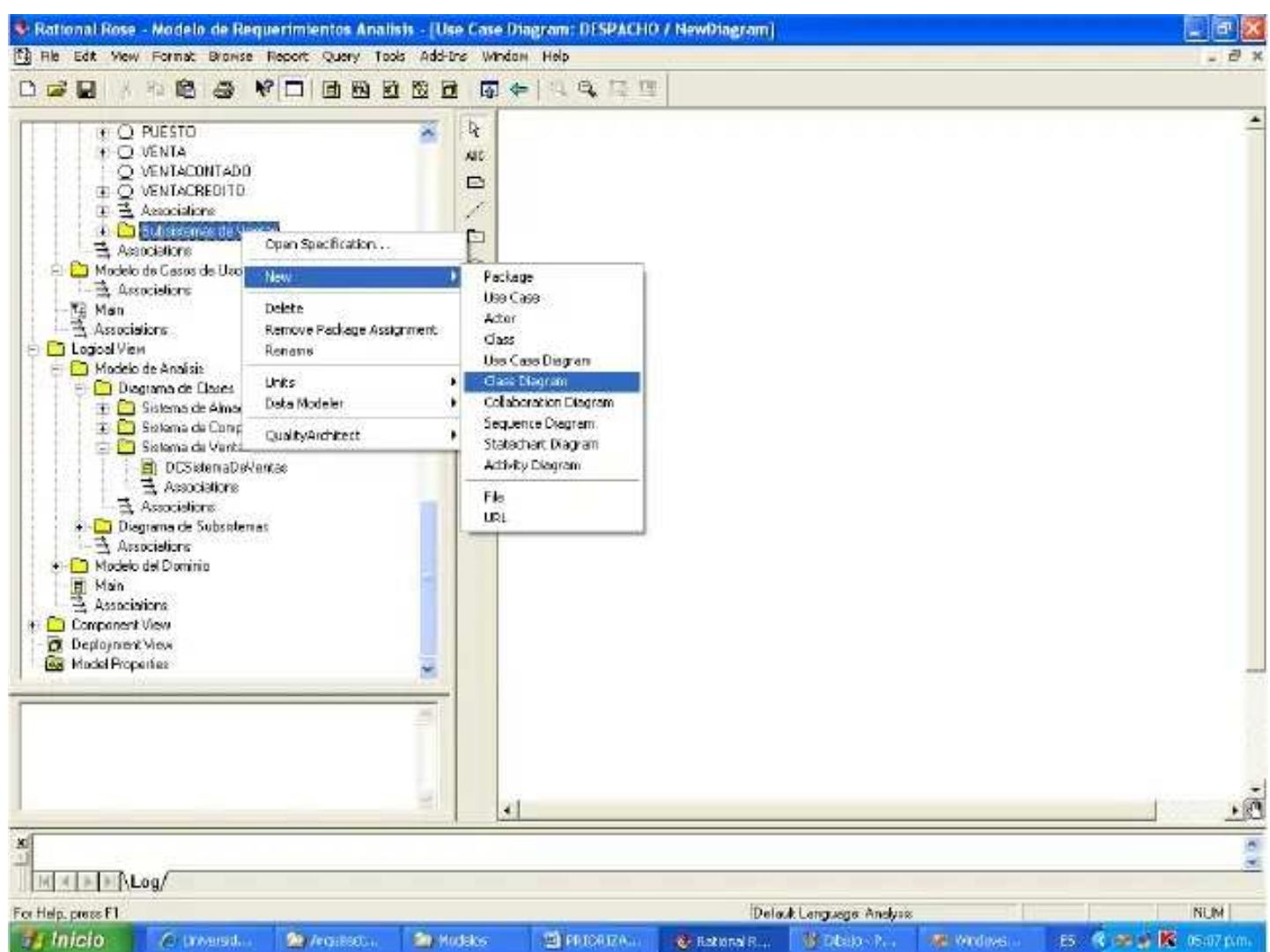
frmVerificarExistencia

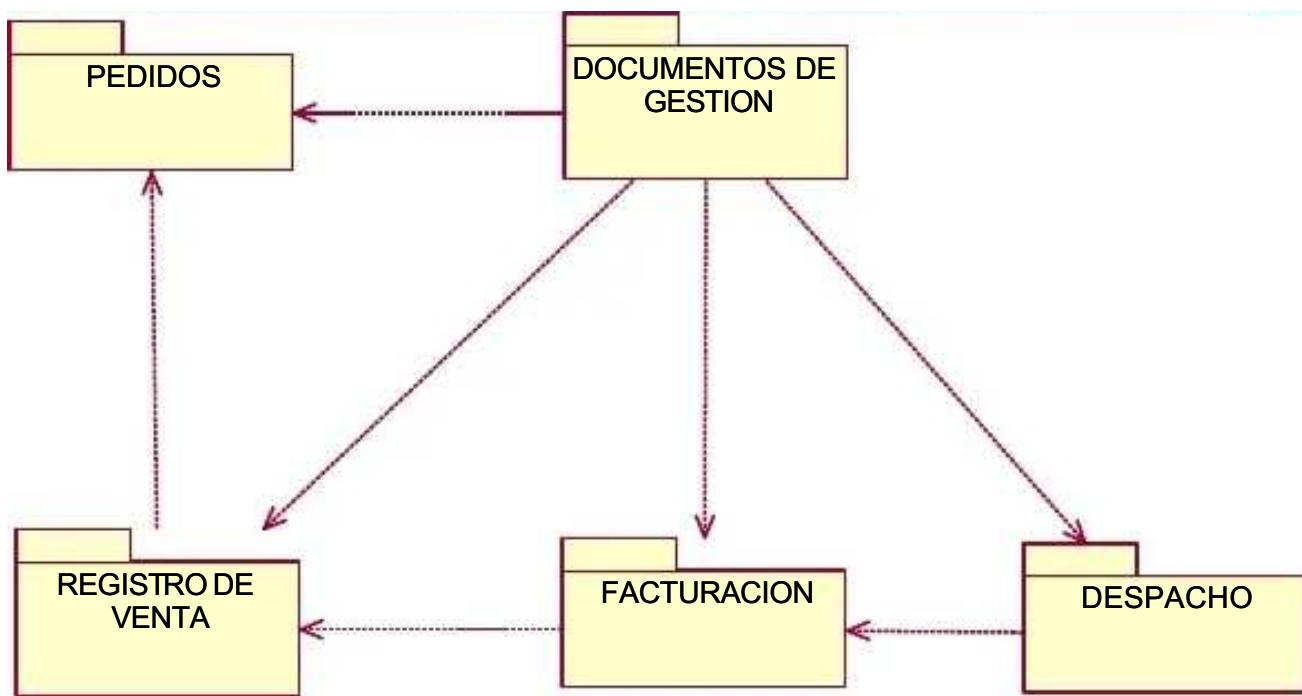


D. Paquete de Controles



4º Diagrama de Paquetes Del Sistema de ventas

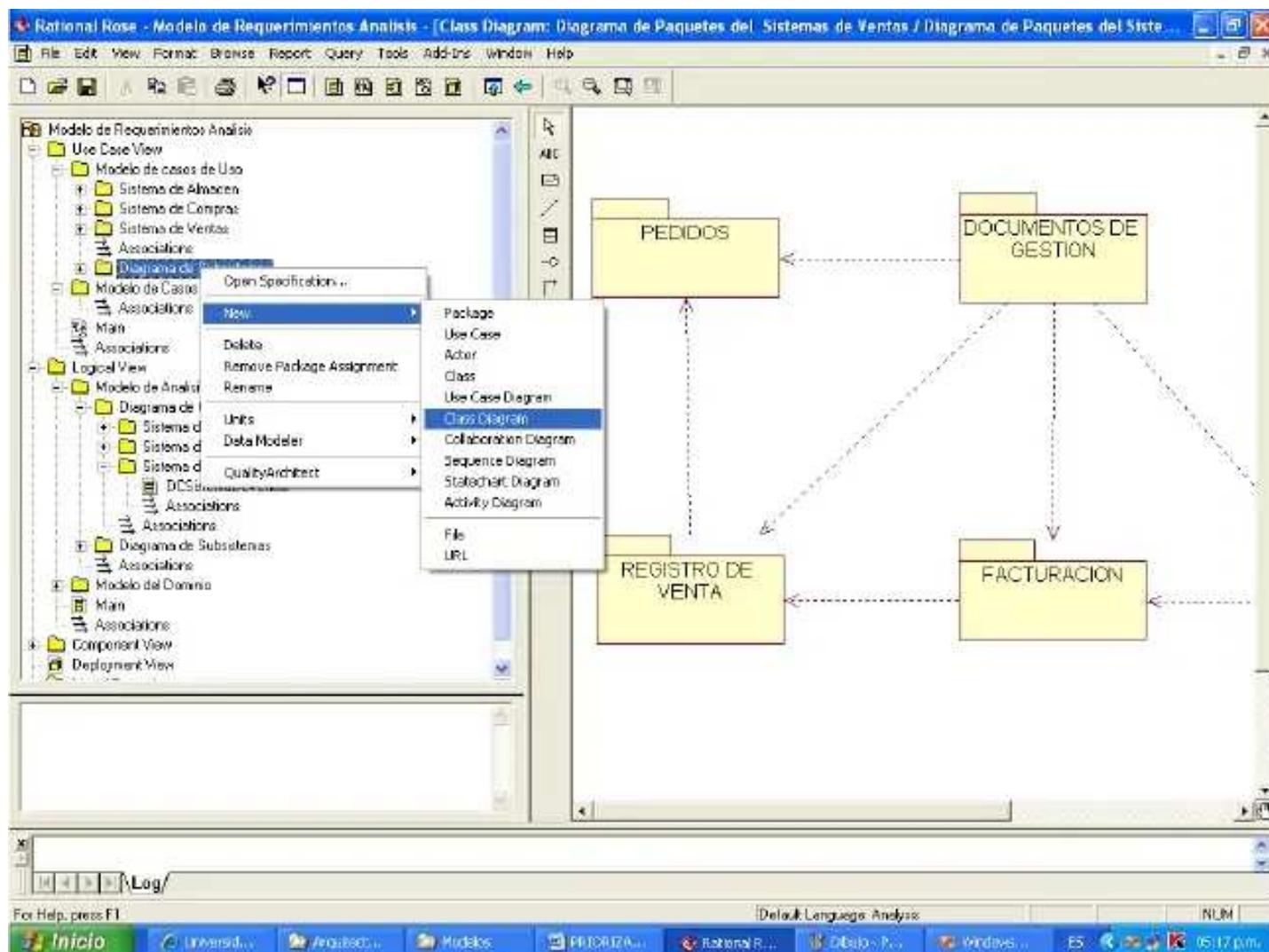


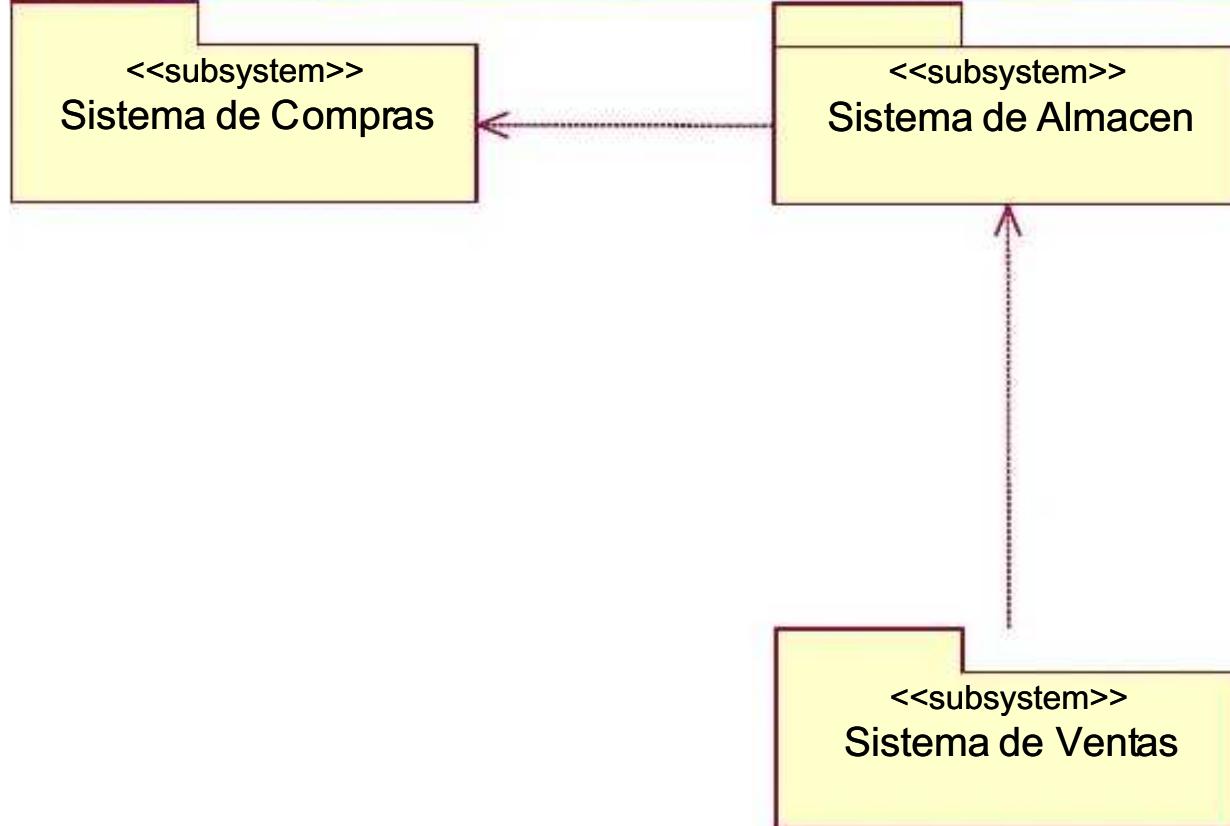
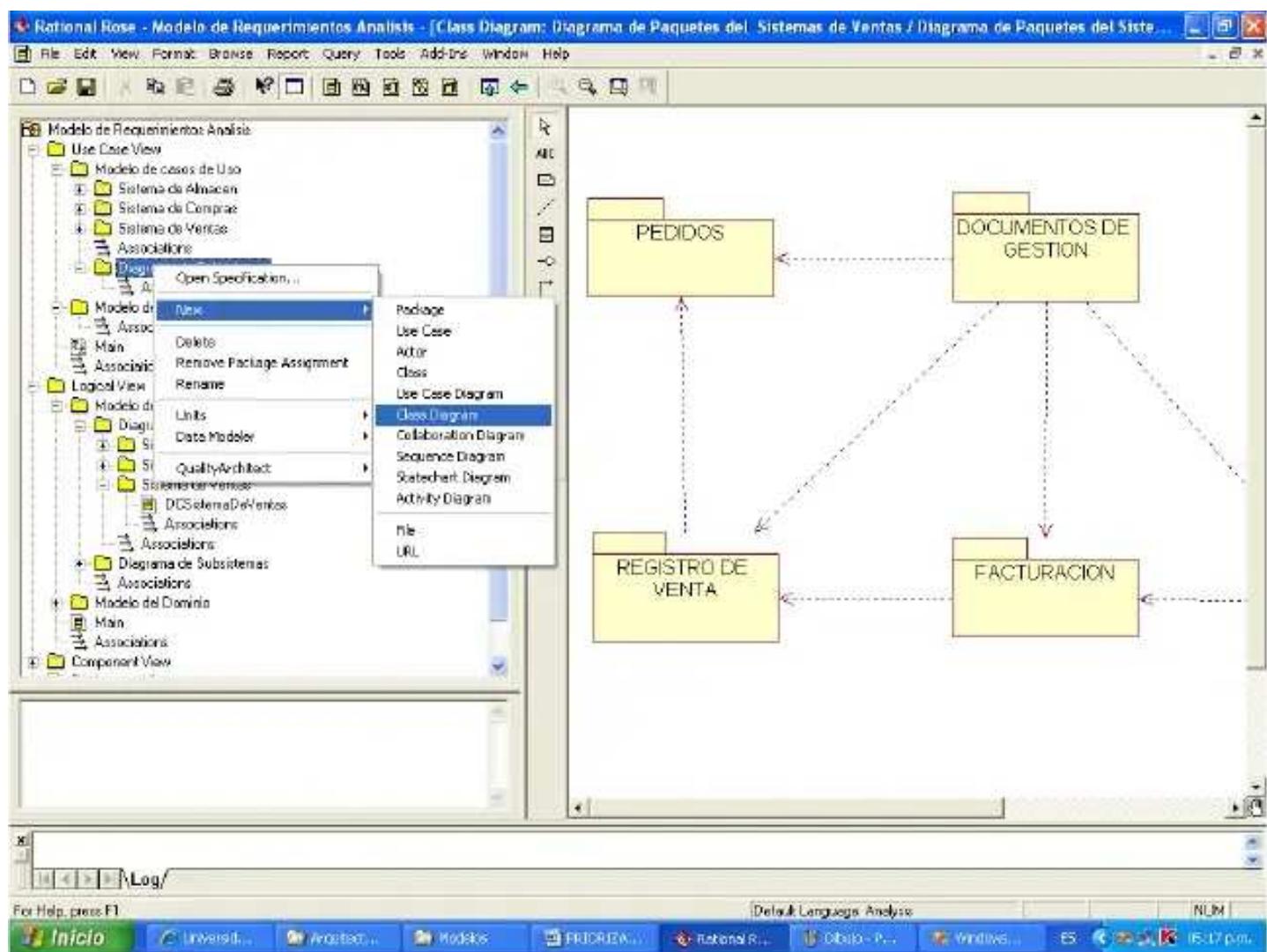


5º Diagrama de Subsistemas

Se elabora en Vista Lógica en el modelo de análisis

Ver ejemplo en Rational Rose

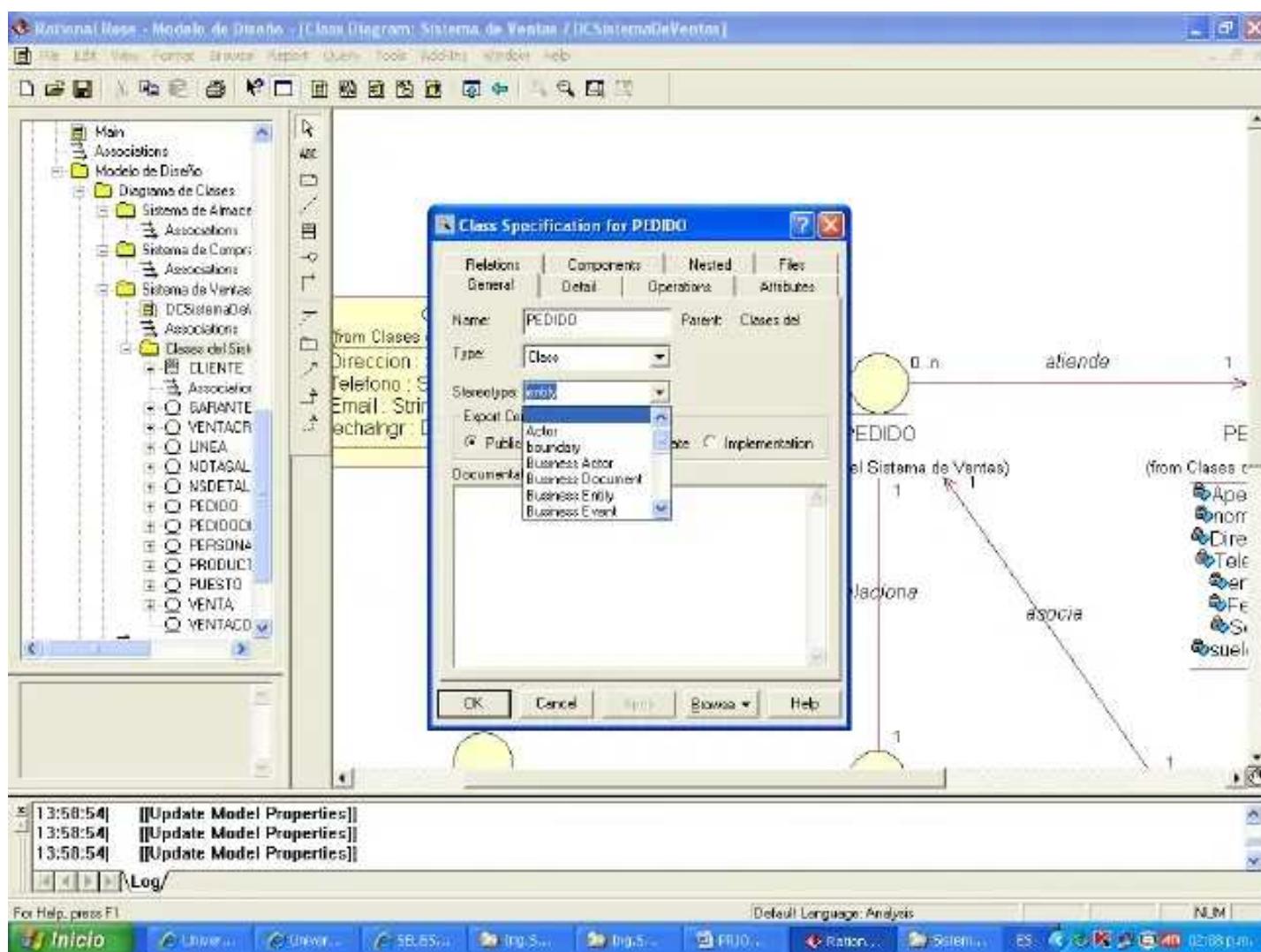




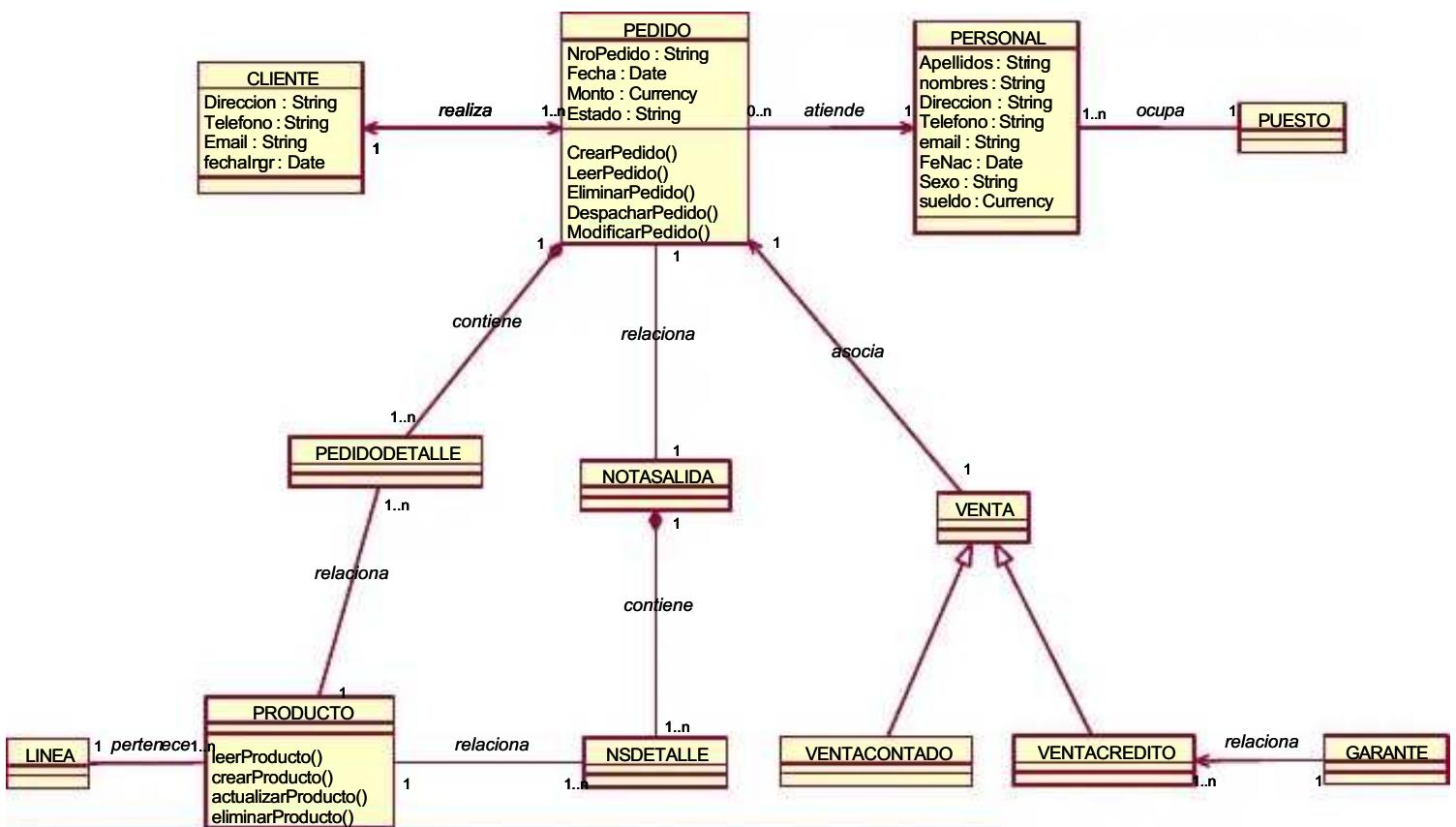
2.2 Modelo de Diseño

1º Diagrama de clases de diseño (Ejemplo Sistema de ventas)

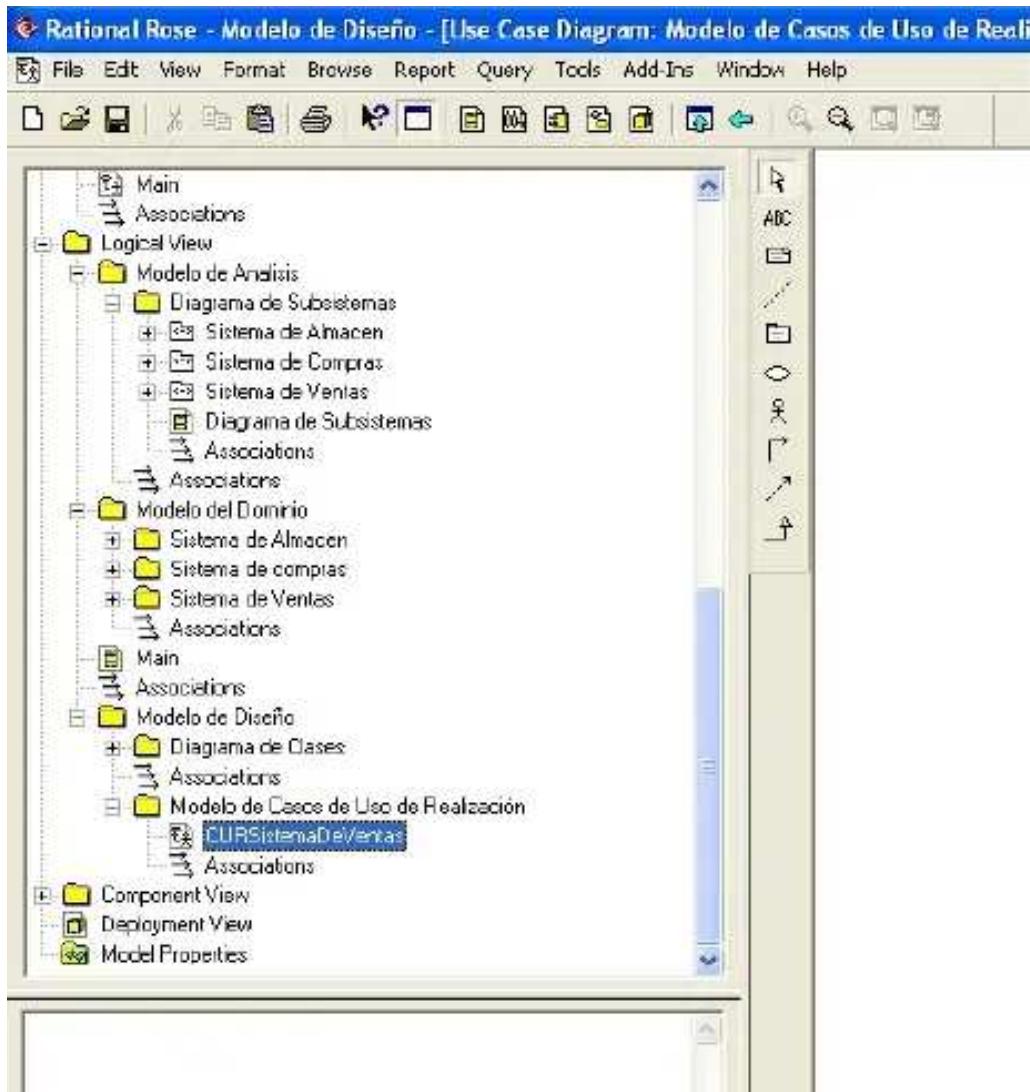
- Renombrar el modelo de Análisis como Modelo de Diseño el Rational Rose.
- Crear en Logical View un paquete llamado Modelo de Diseño
- Dentro de la carpeta Modelo de Diseño en el Paquete diagrama de Clases del Sistema de Ventas creamos el paquete Clases del Sistema de Ventas y arrastramos todas las entidades, reubicándolas.
- Convertir las entidades en clases propiamente.



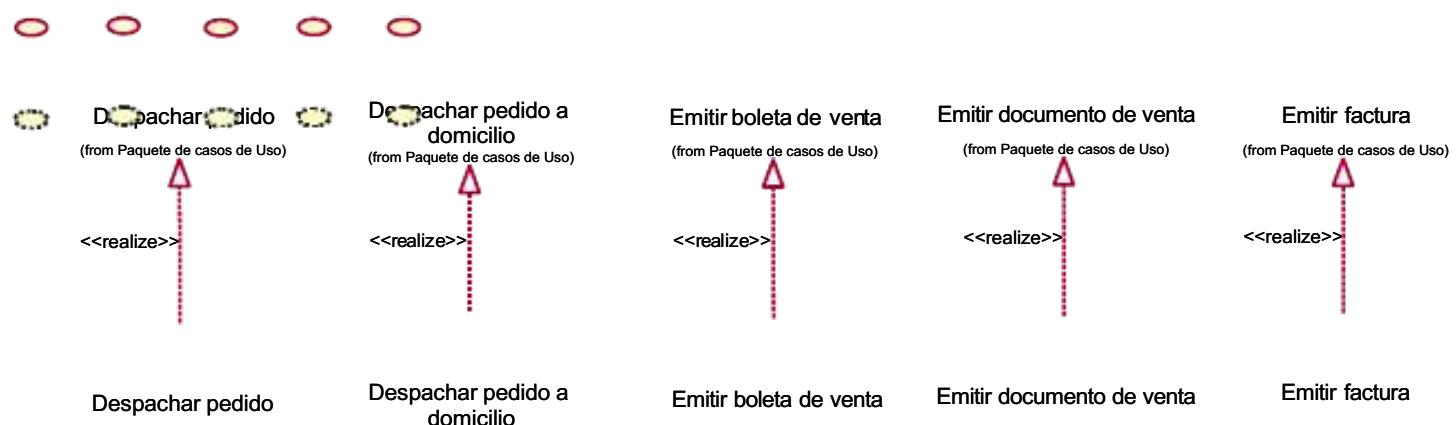
- Agregar los métodos y parámetros a cada clase en base a los diagramas de colaboración



2º Casos de Uso de Realización (Ejemplo Sistema de Ventas)

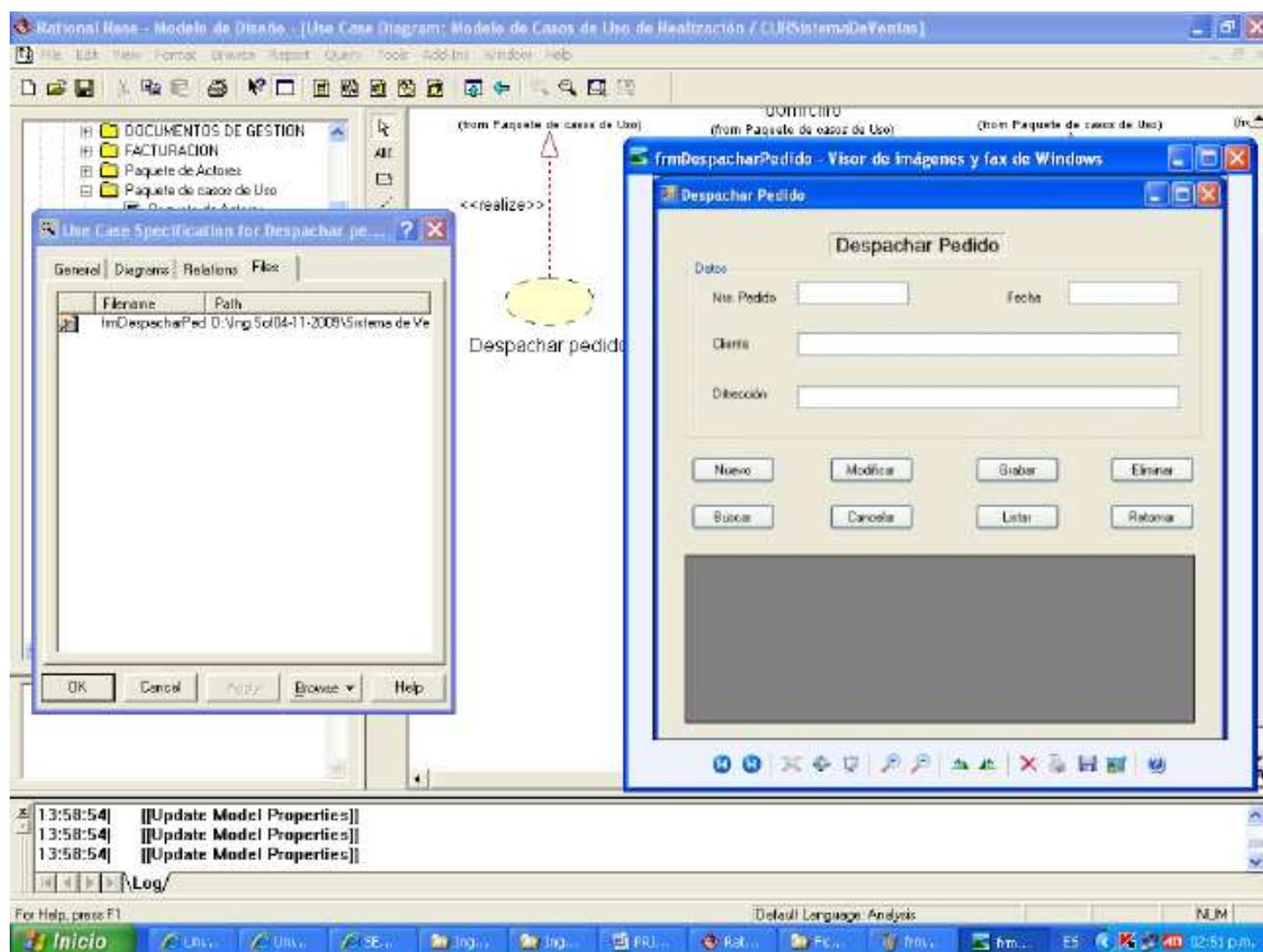


Creado el diagrama de casos de uso de realización se arrastran al editor todos los casos de uso y se eliminan con DEL todo tipo de relación, quedando individualizados y luego se agregan los casos de realización con el mismo nombre de cada caso de uso que dando emparejados.



Cada caso de uso de realización se documenta con:

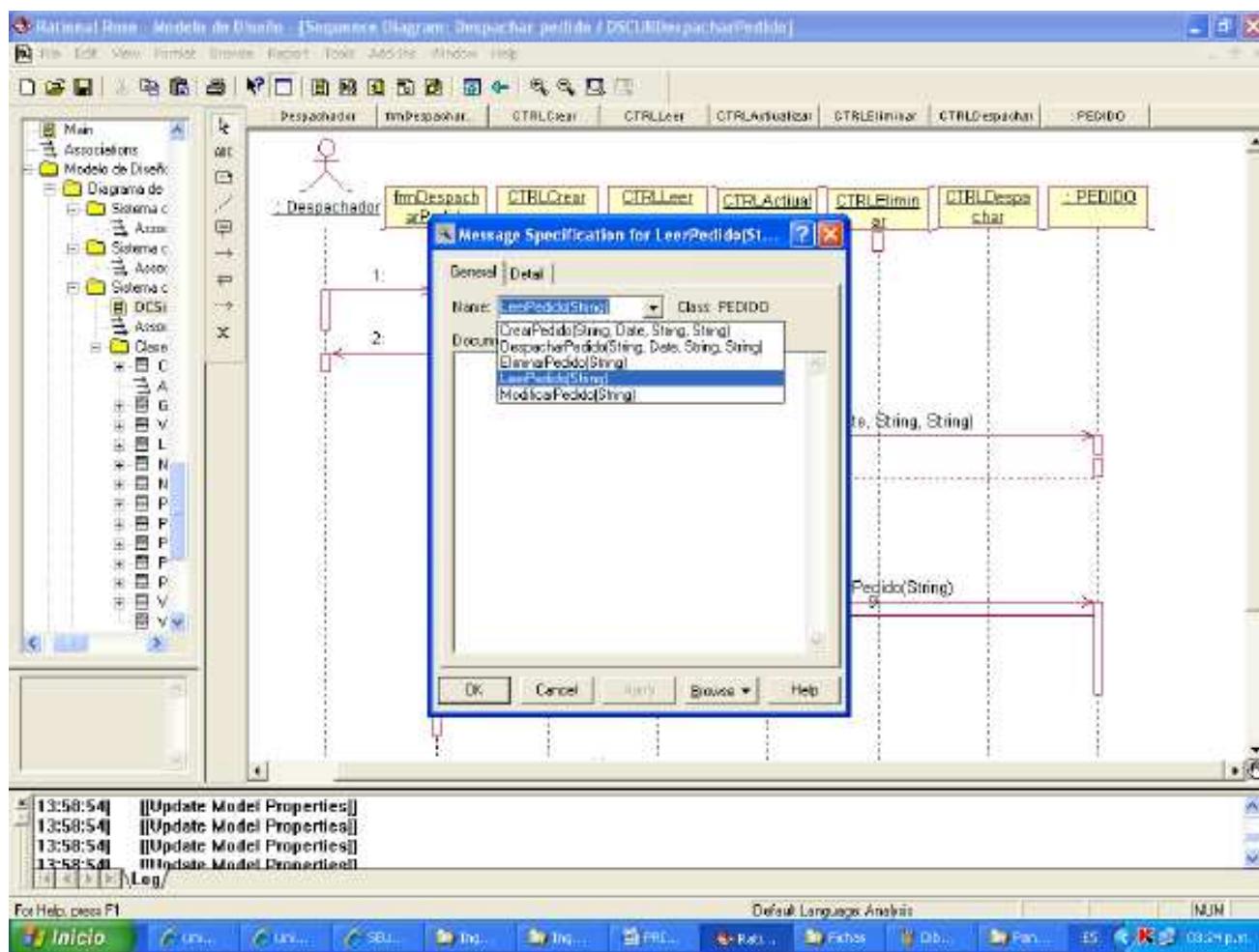
i) Diseño de la ficha (formulario)



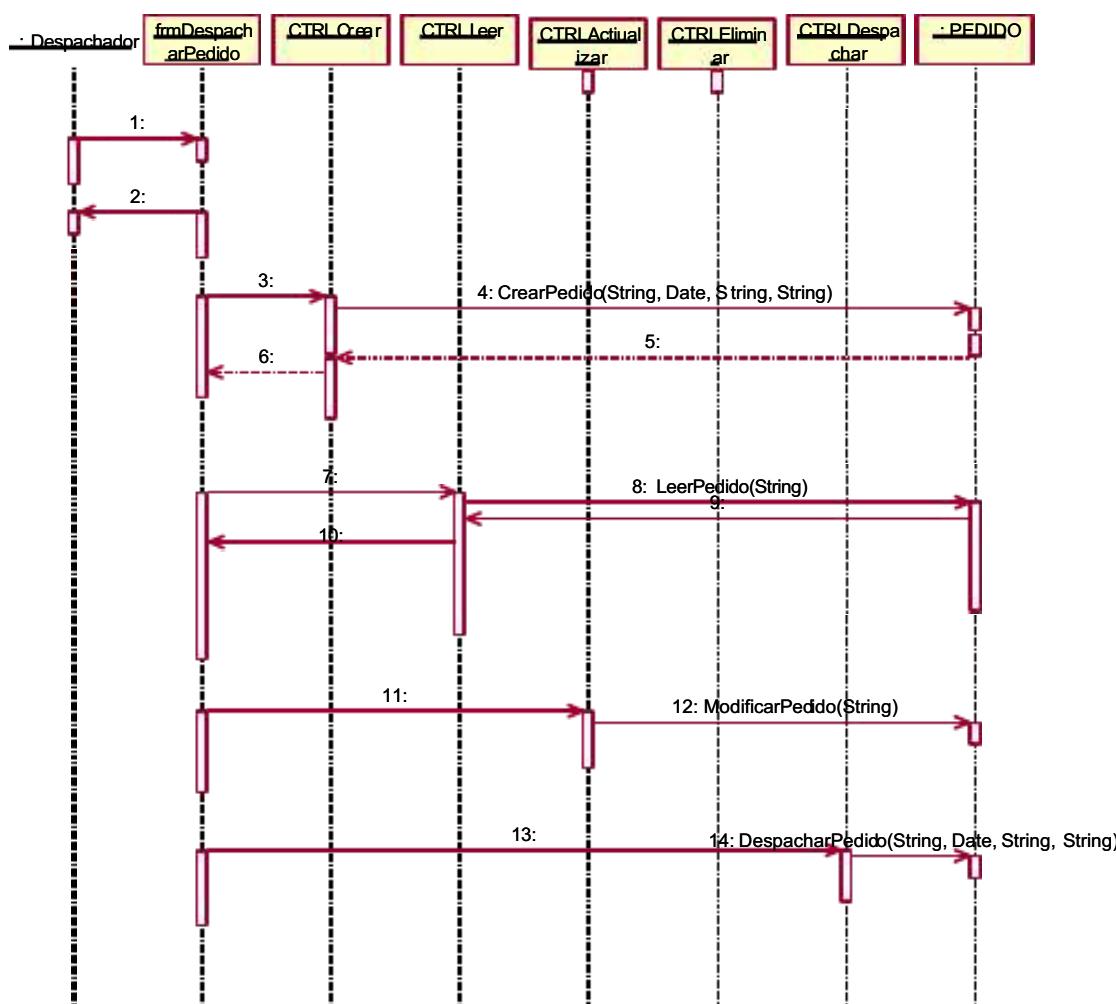
ii) Diagrama de secuencia

Sirven para comprobar que todas las clases de diseño tienen todos los métodos que se obtuvieron de los diagramas de colaboración. Los métodos de los controles a las clases solo se SELECCIONAN, no se escriben.

Si no hay un método en el combo box del diagrama de secuencia, debe agregarse a la clase en el diagrama de clases y repetir la operación.

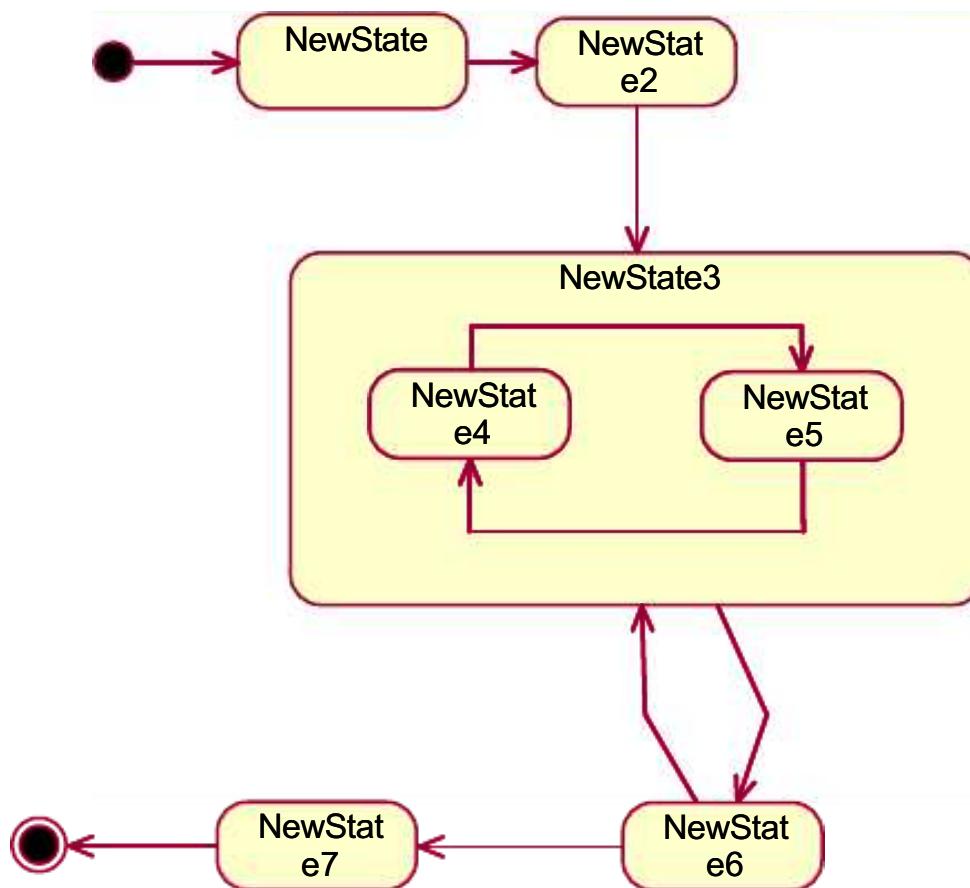
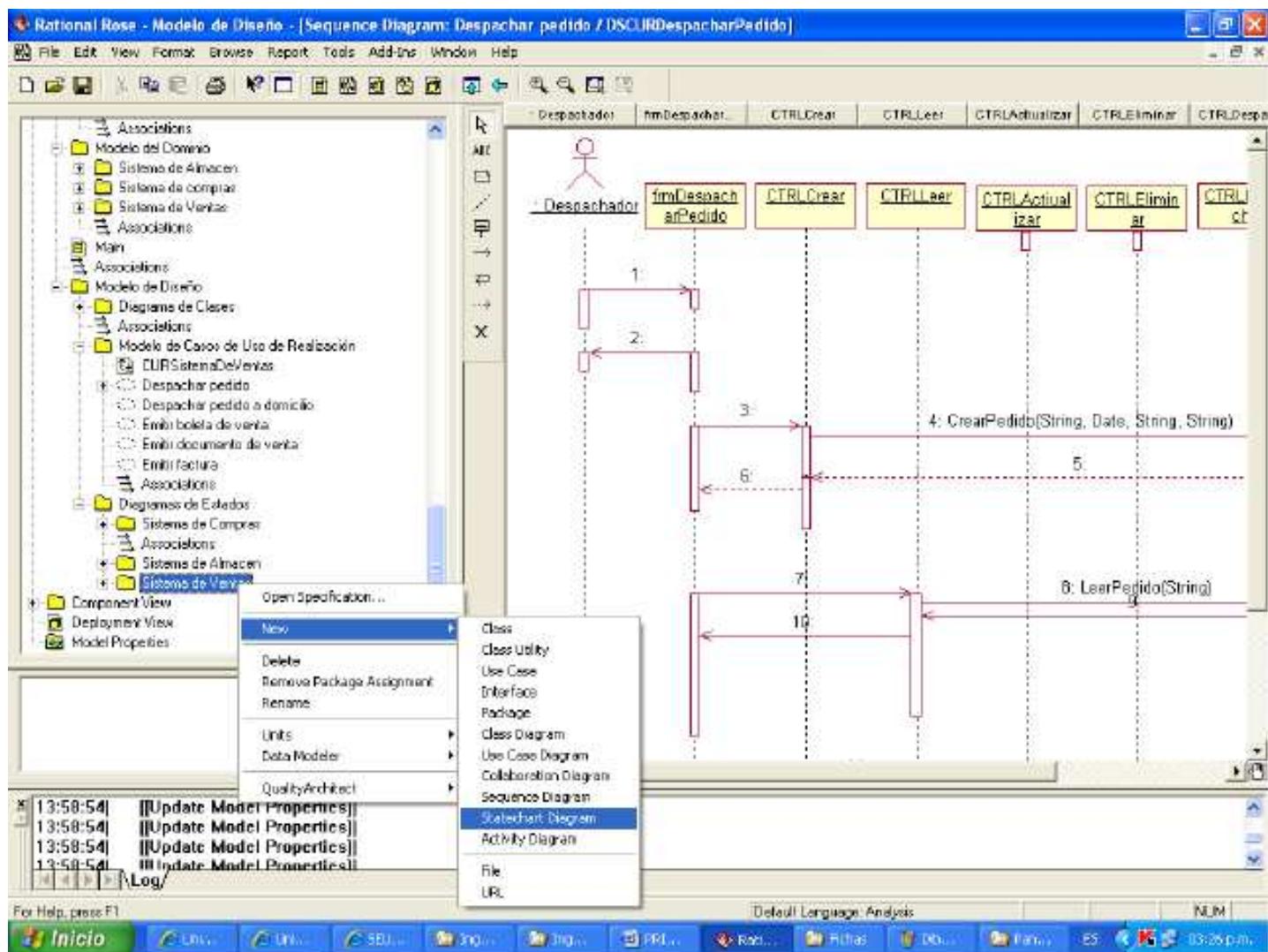


iii



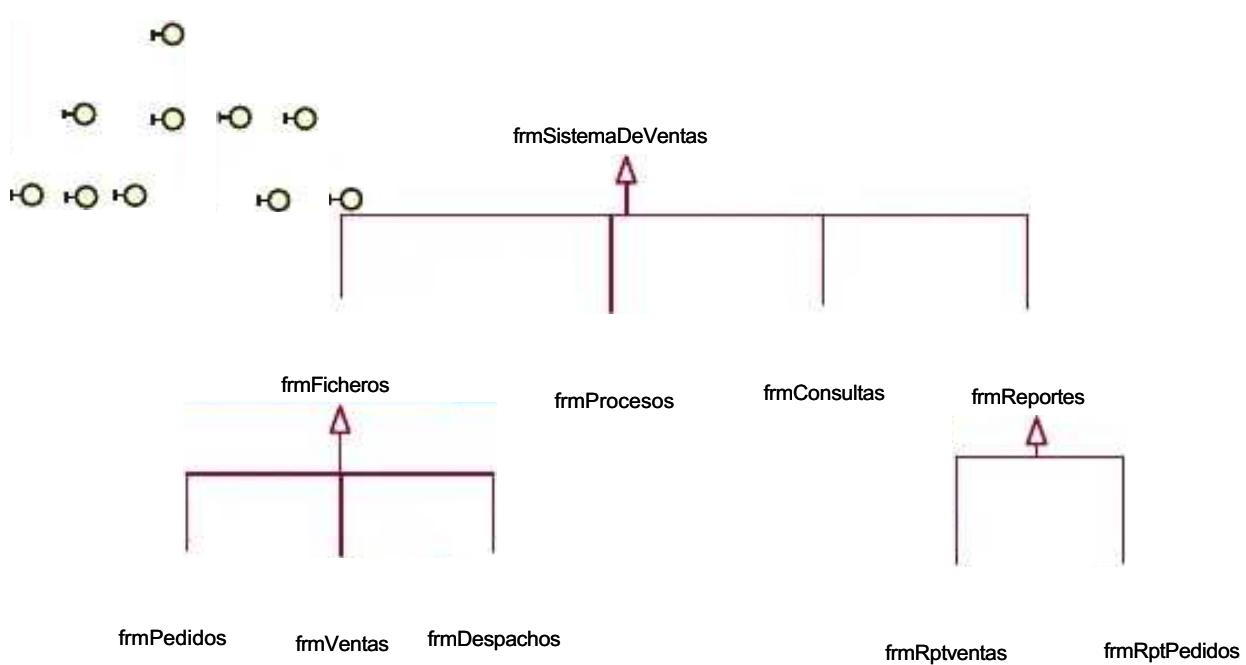
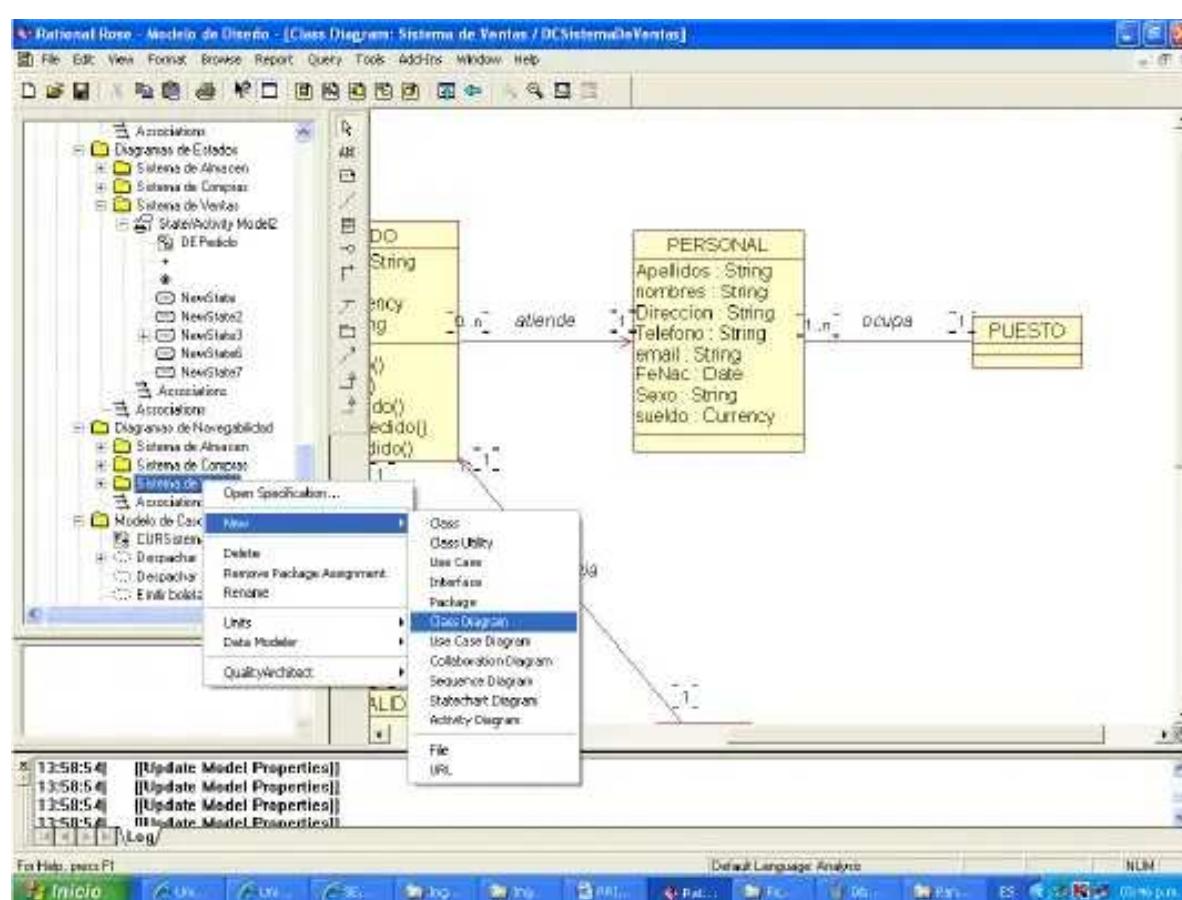
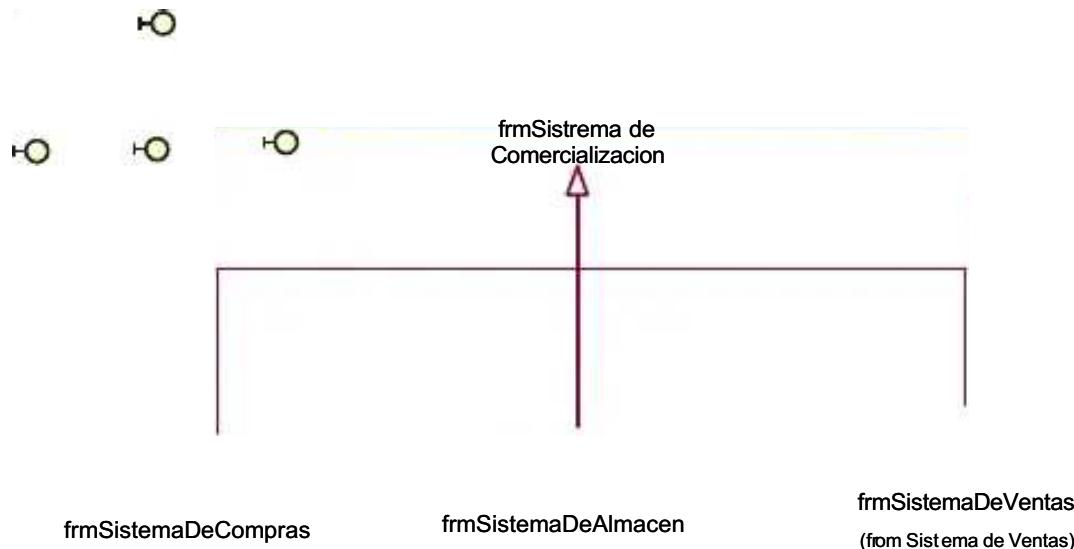
iii) Diagrama de actividades (opcional)

3º Diagramas de estados para las clases que lo requieran



4º Diagrama de Navegabilidad

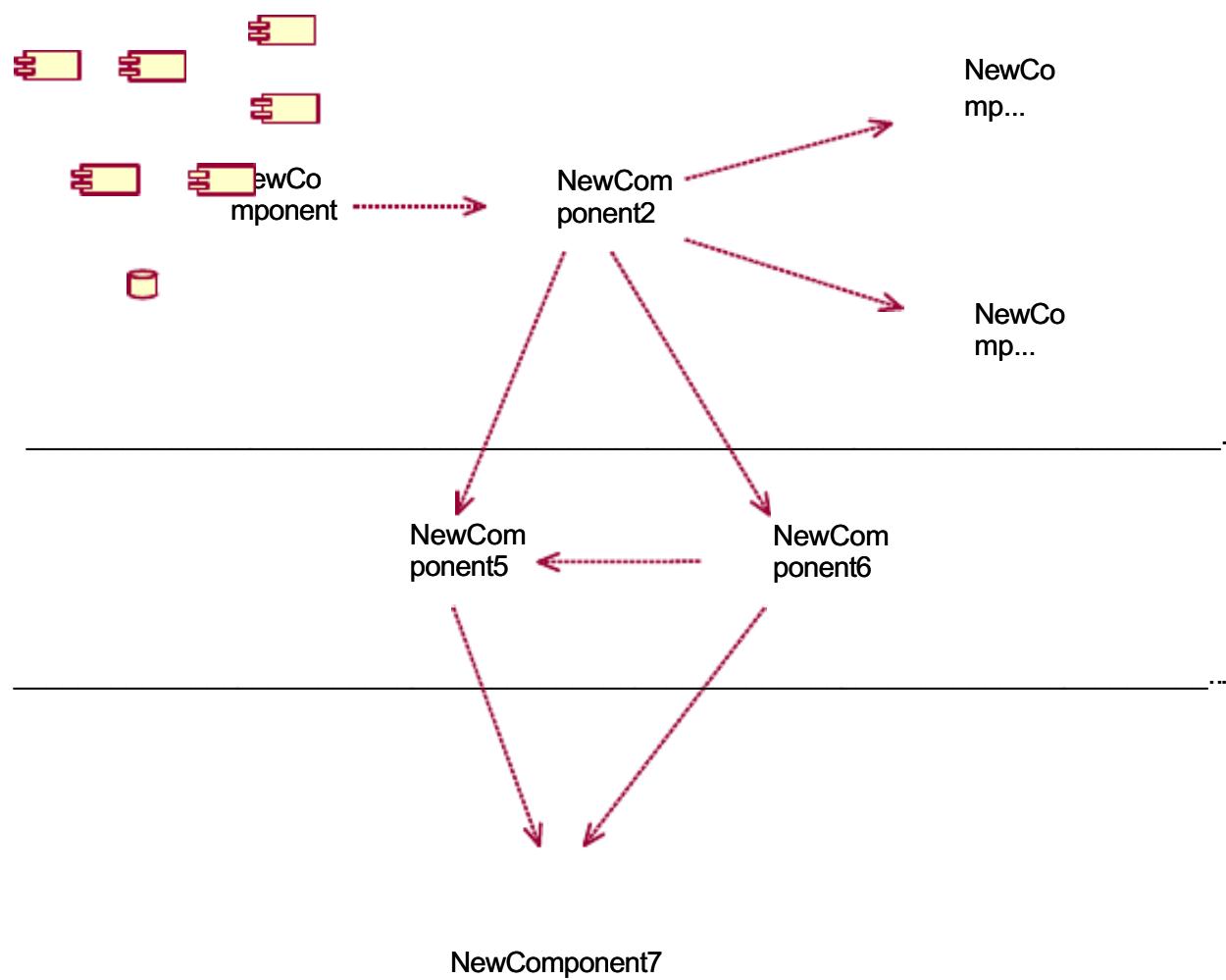
Representa el menú desplegable del sistema de información



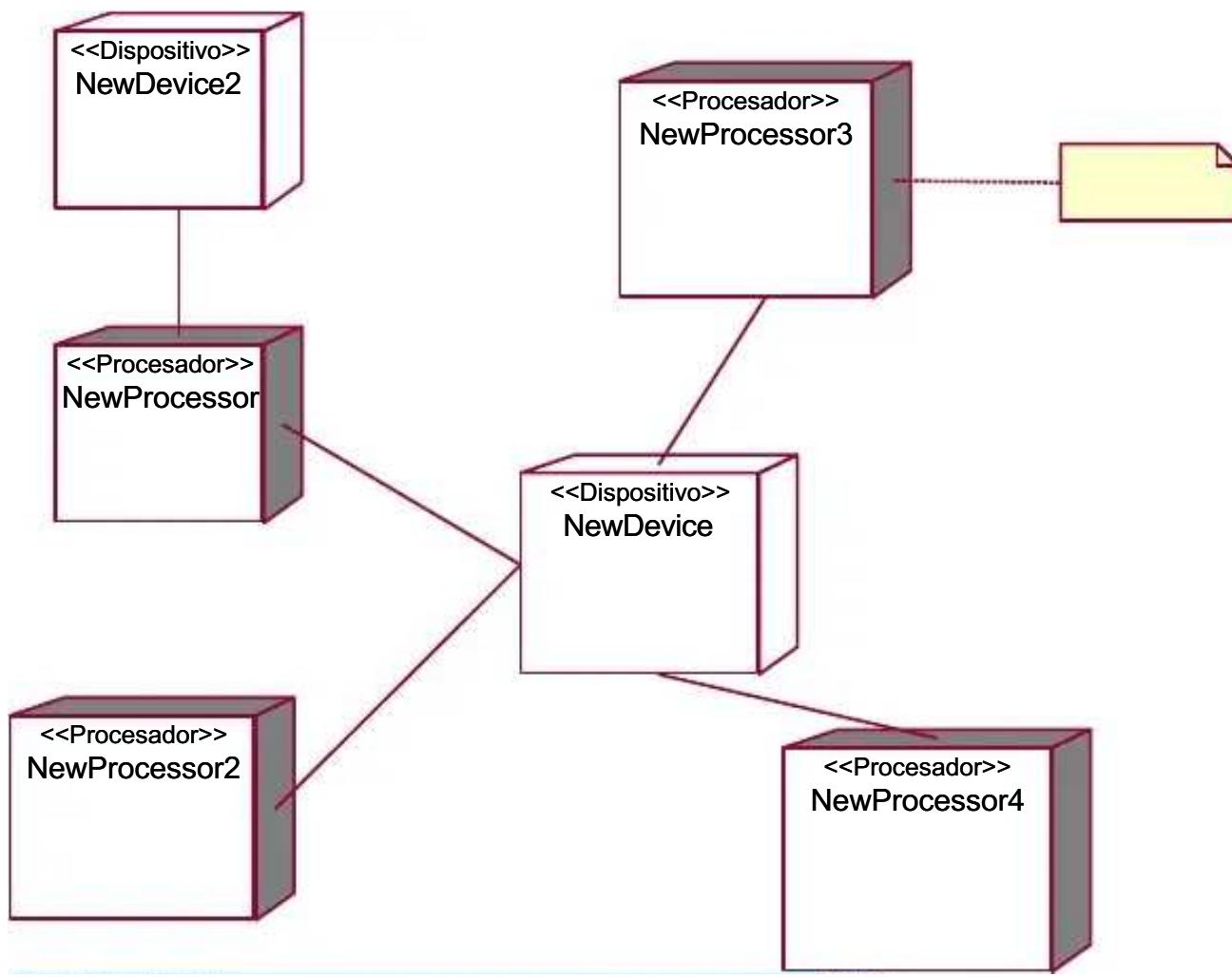
FASE III : CONSTRUCCION

3.1 Modelo de Desarrollo

1º Diagrama de componentes



2º Diagrama de despliegue

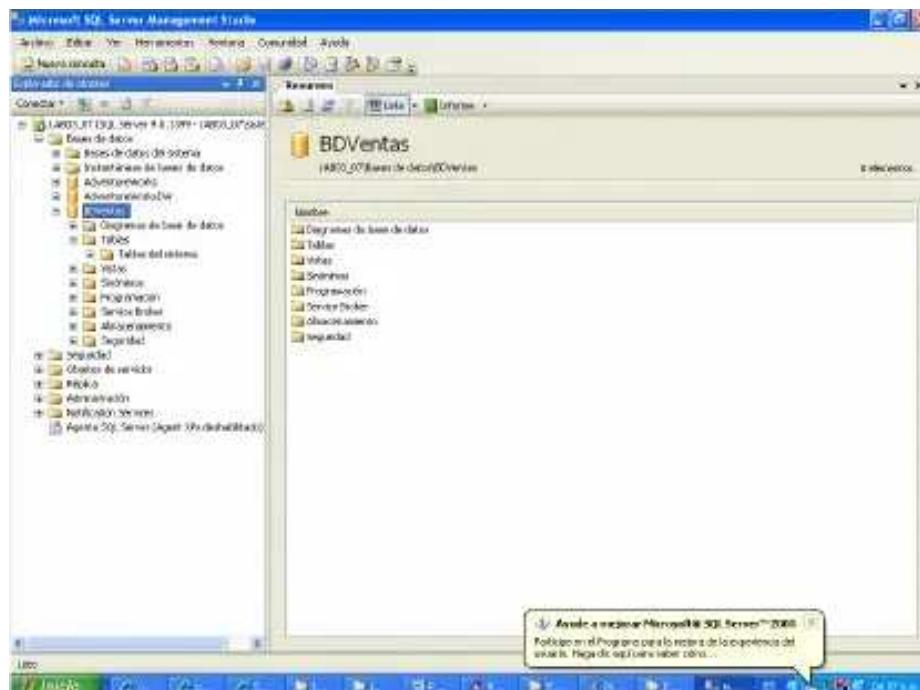


3º Modelo de Datos (Rational Rose)

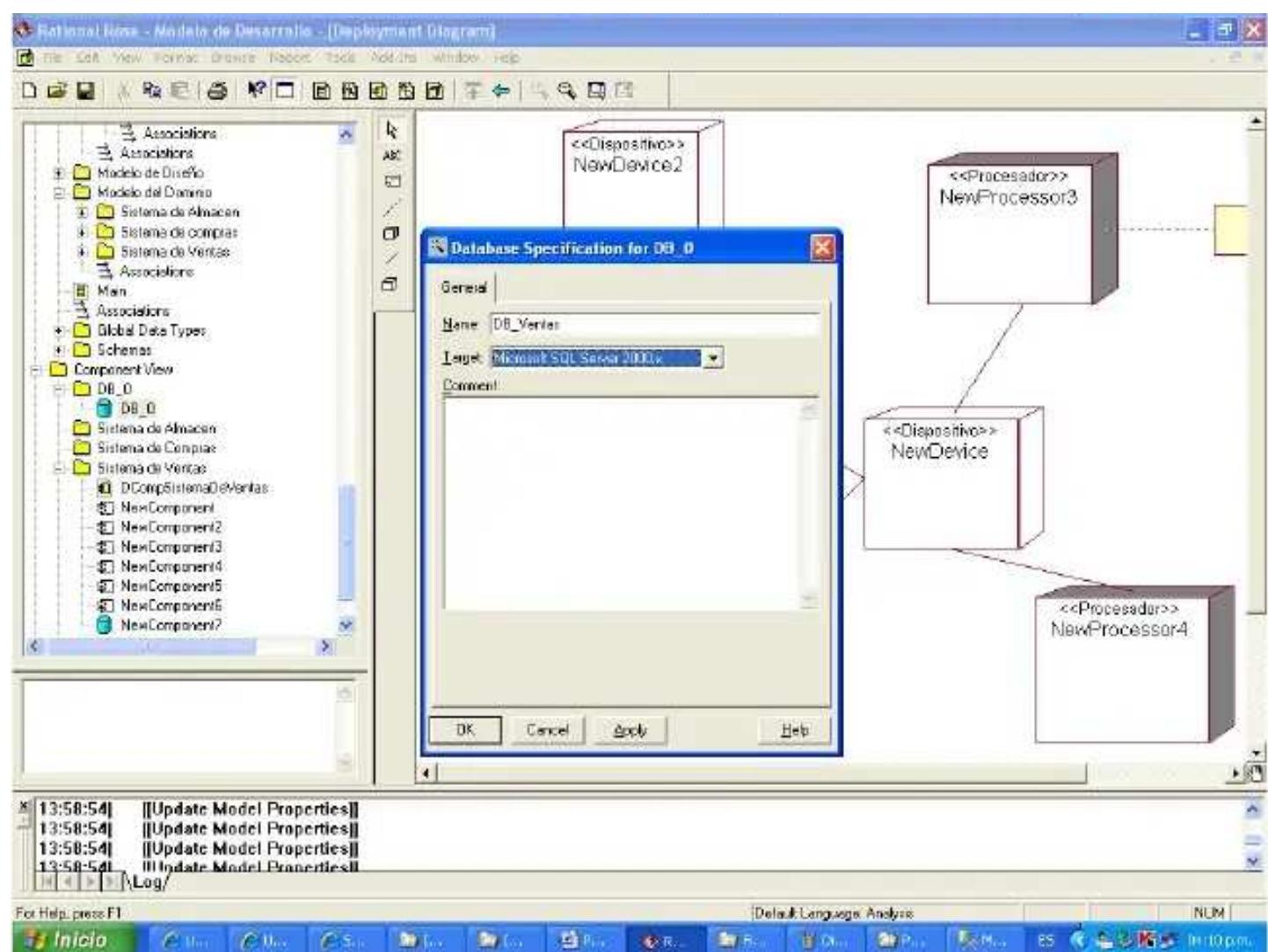
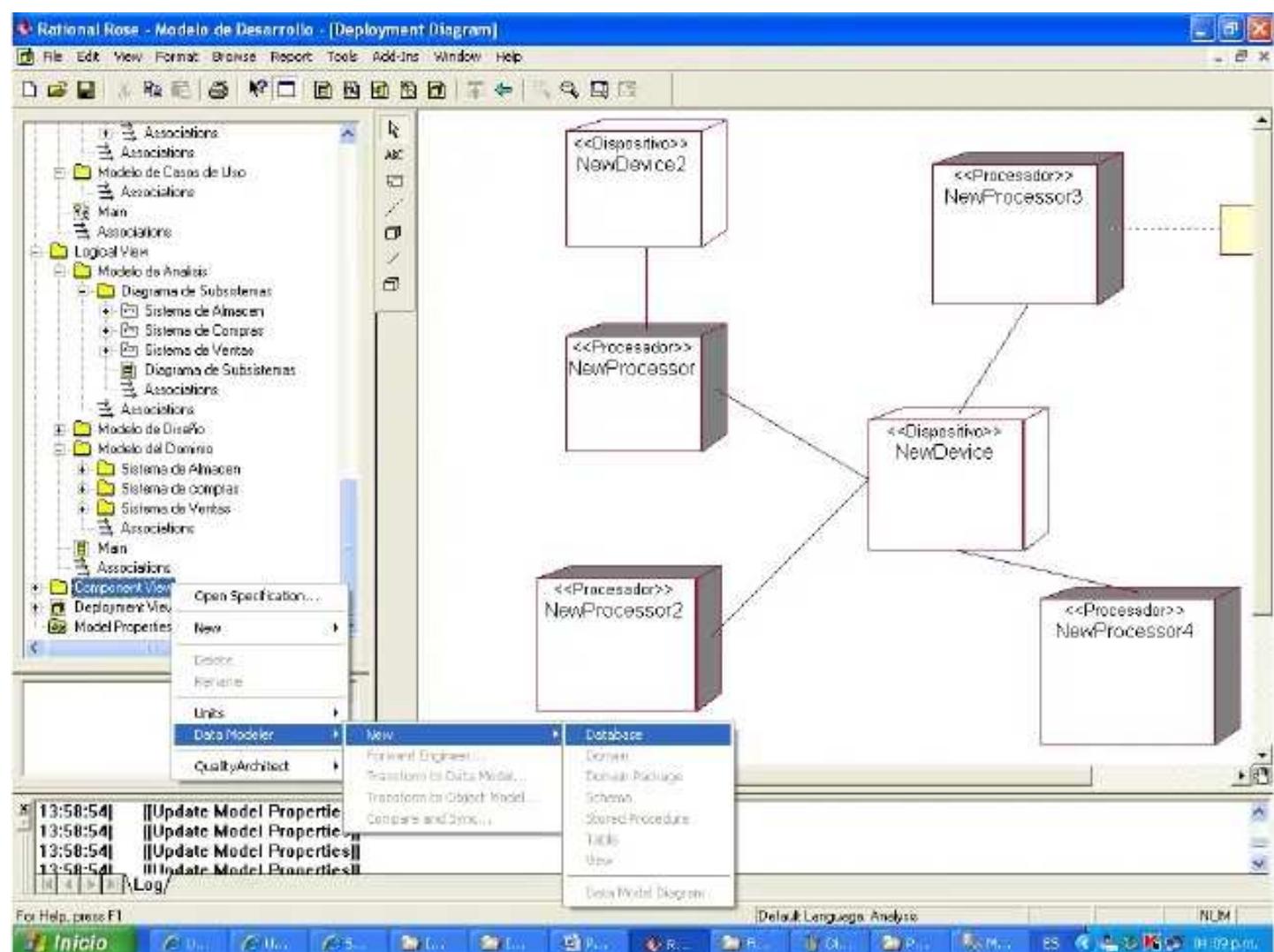
El diagrama de clases se convierte en un modelo de datos, es decir, se pierden los métodos convirtiéndose las Clases en Tablas.

PASOS:

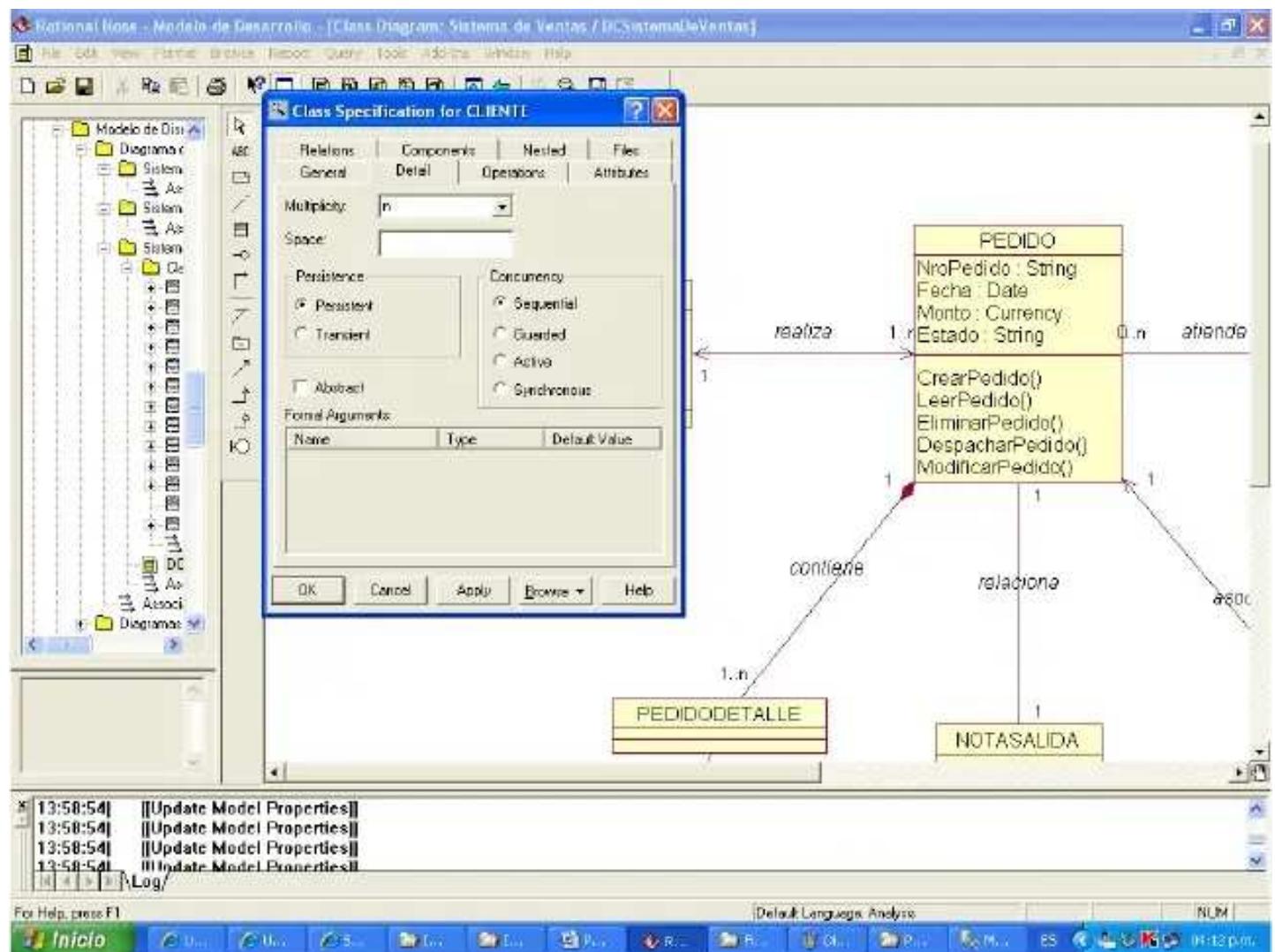
- Seleccionar un gestor de Bases de Datos (SQL Server) y crear una base de datos vacía (BDVentas).



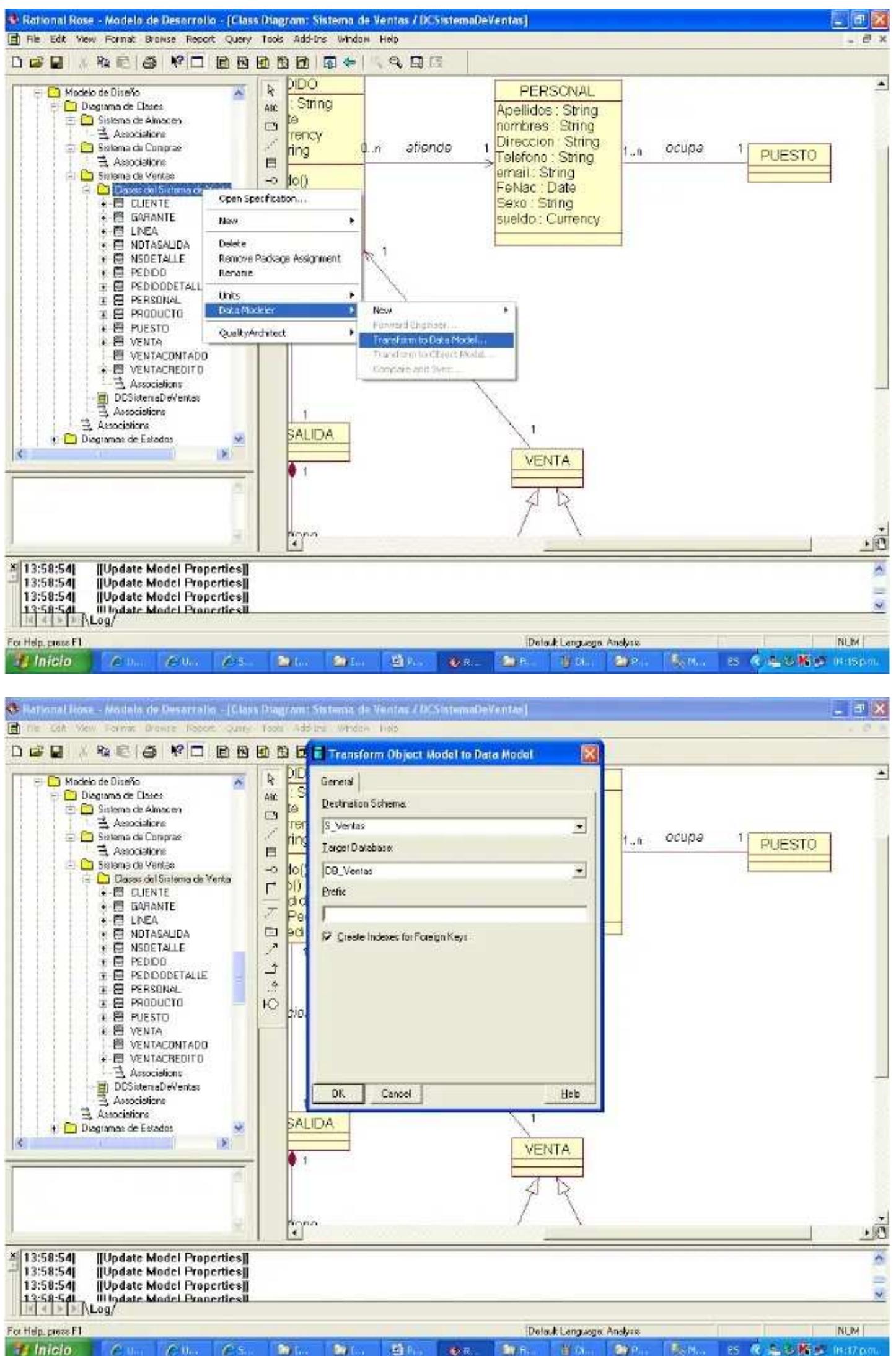
- ii. Seleccionar en el Rational Rose el Gestor de base de datos a donde se ha de migrar el modelo de datos.



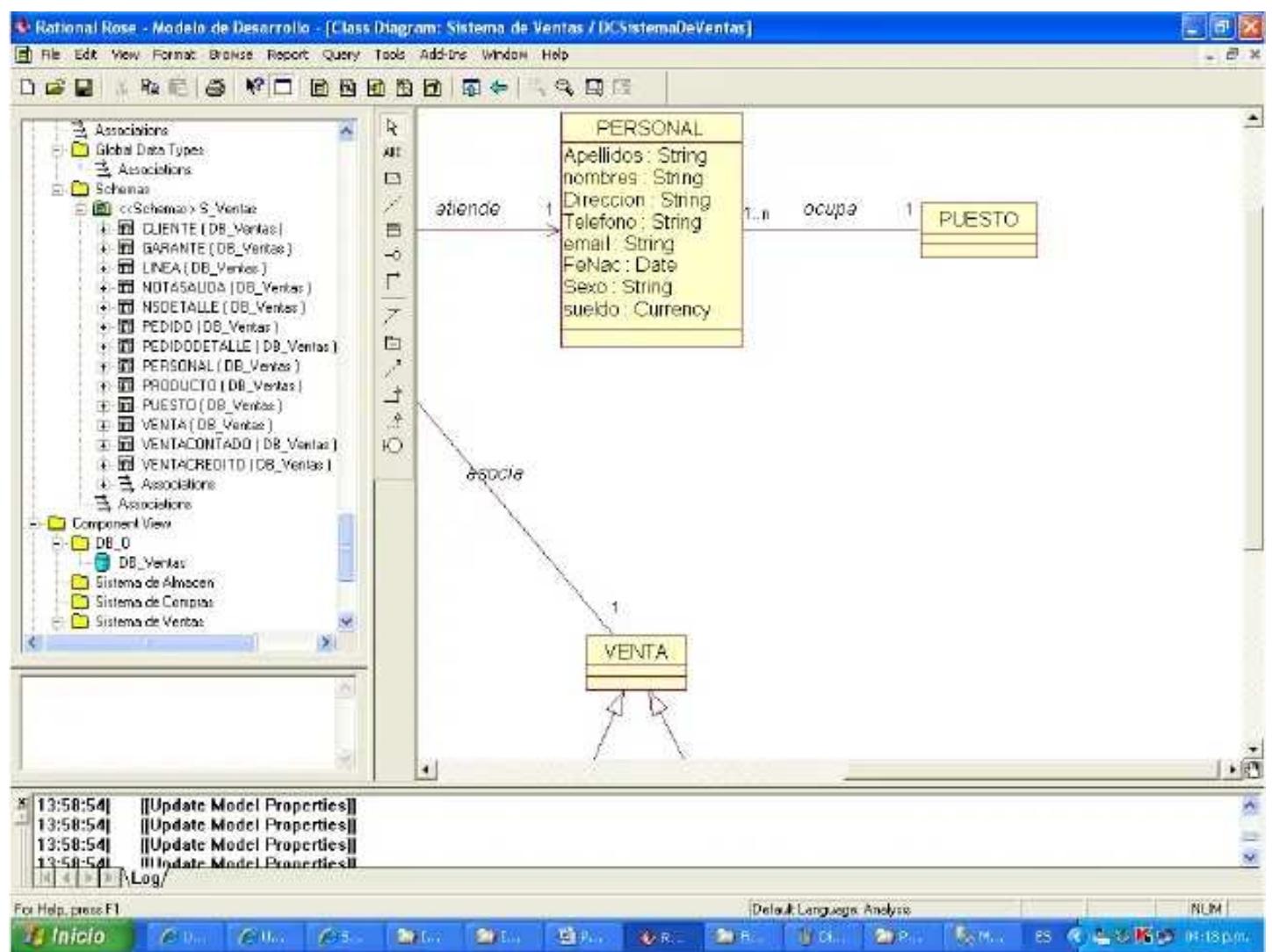
- iii. Convertir todas las clases en PERSISTENTES en el diagrama de clases del Rational Rose. Si se omite alguna clase y no se convierte como persistente, NO SE GENERA en el Modelo de Datos.



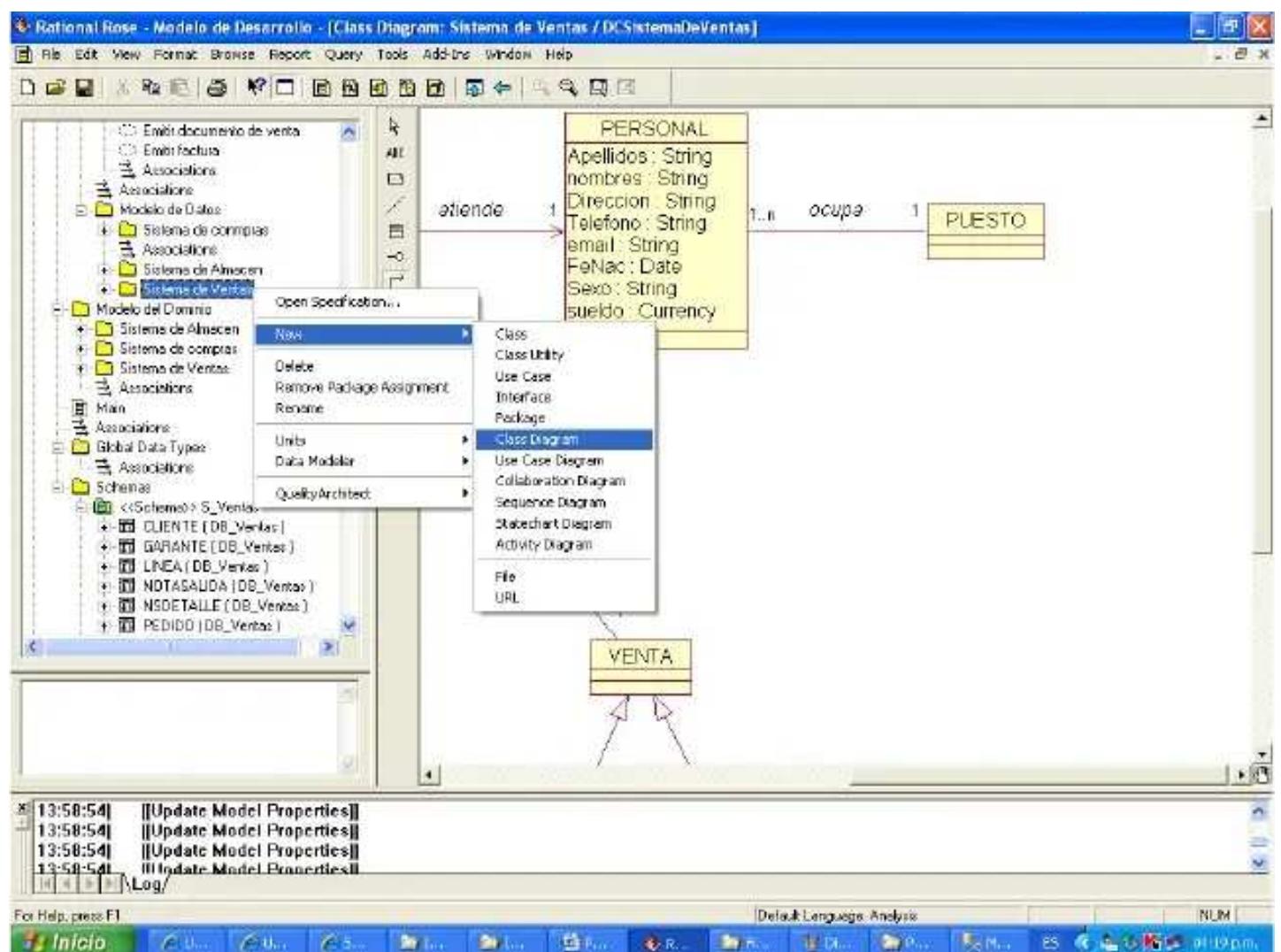
- iv. Convertir el diagrama de Clases a un Modelo de Datos



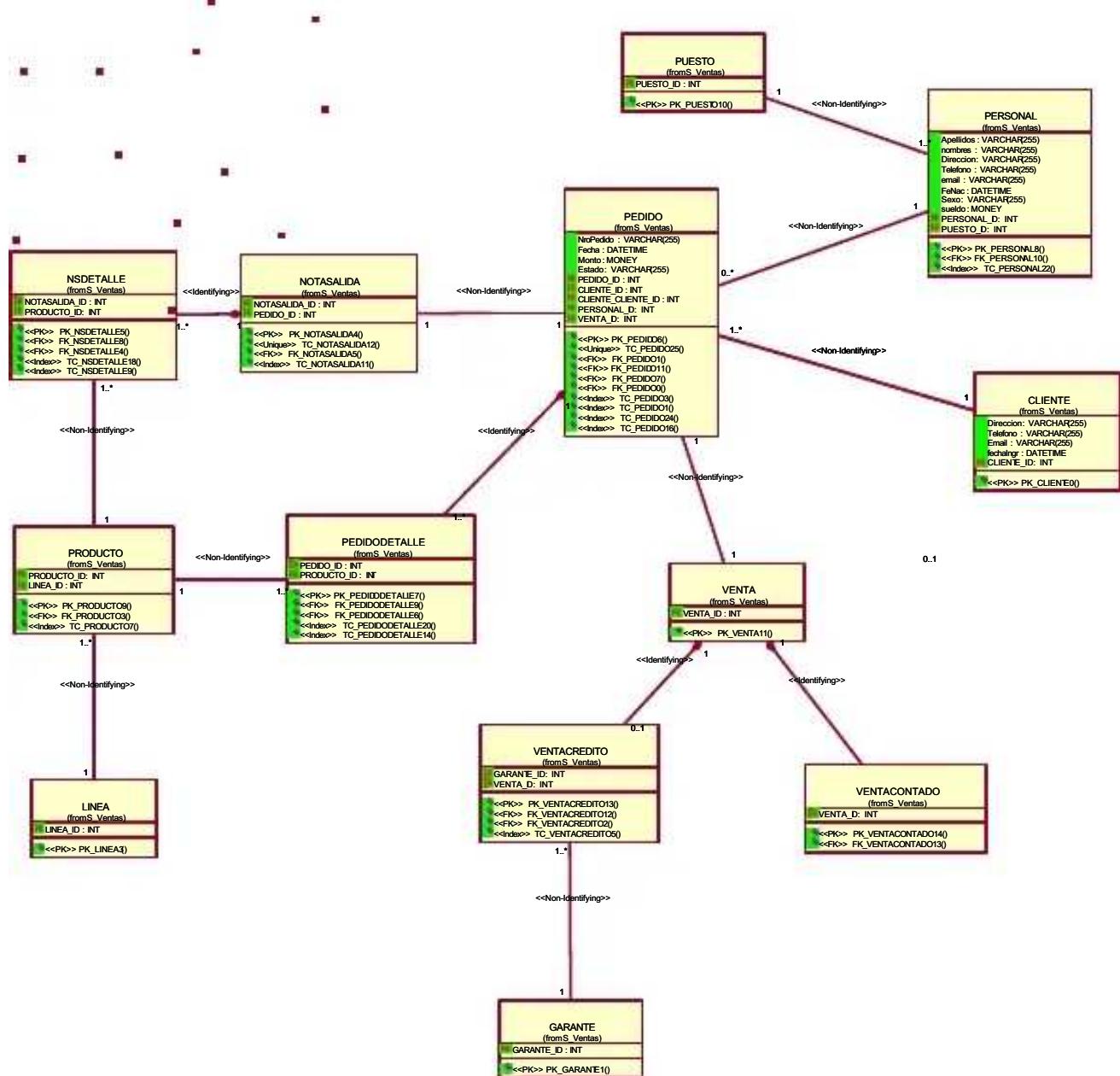
Se crean las tablas en:



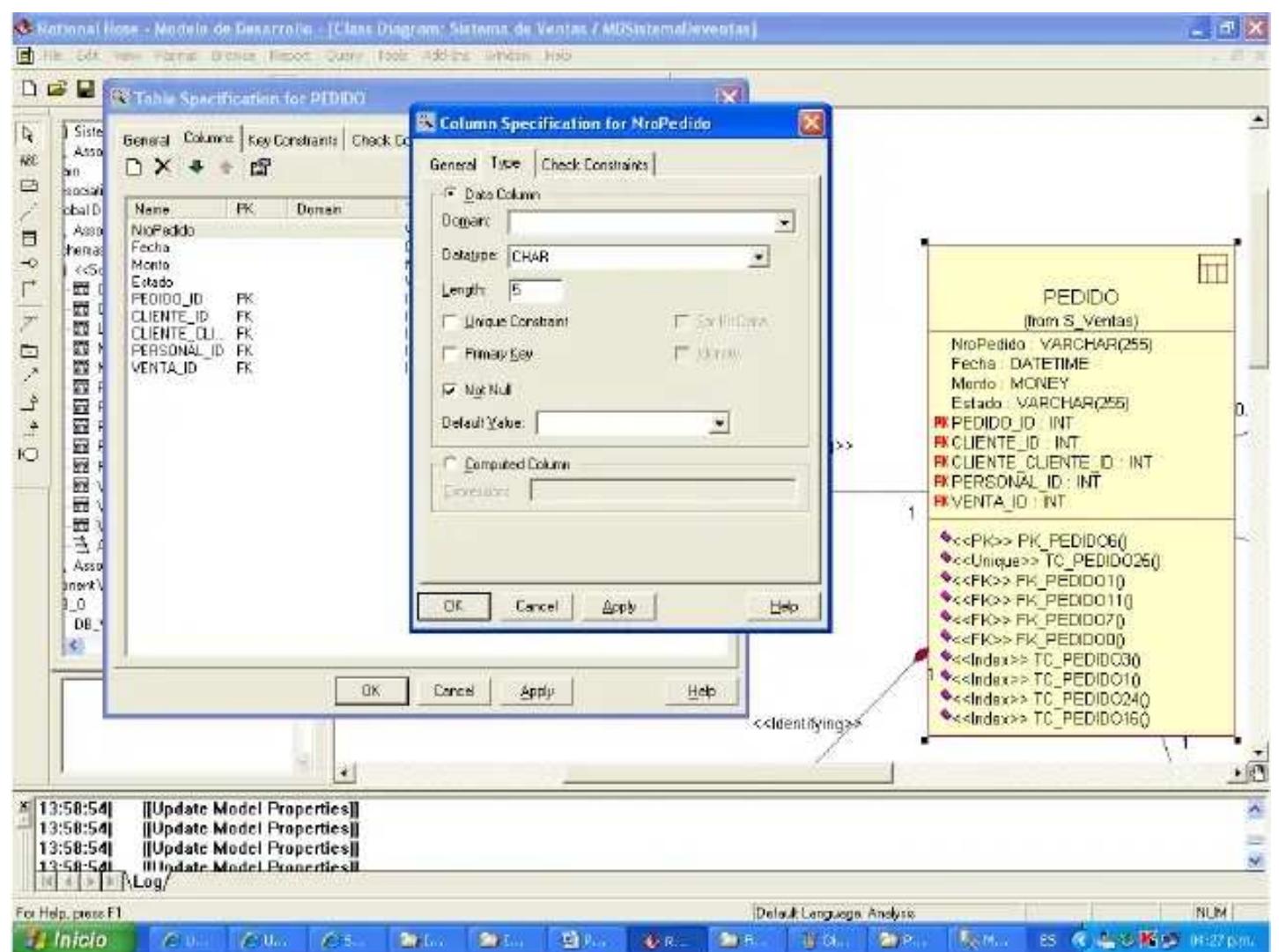
V. Generar el Modelo de datos



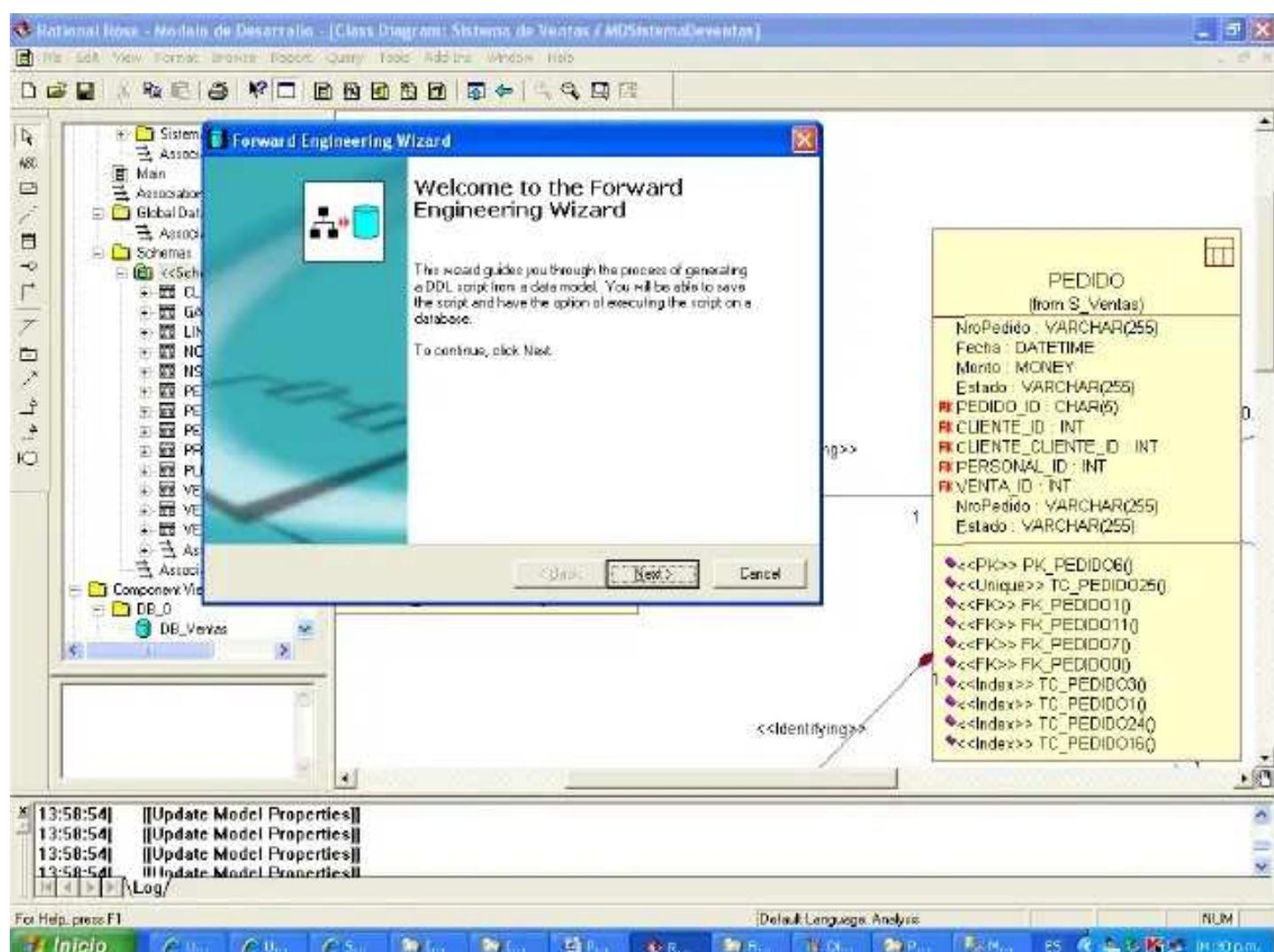
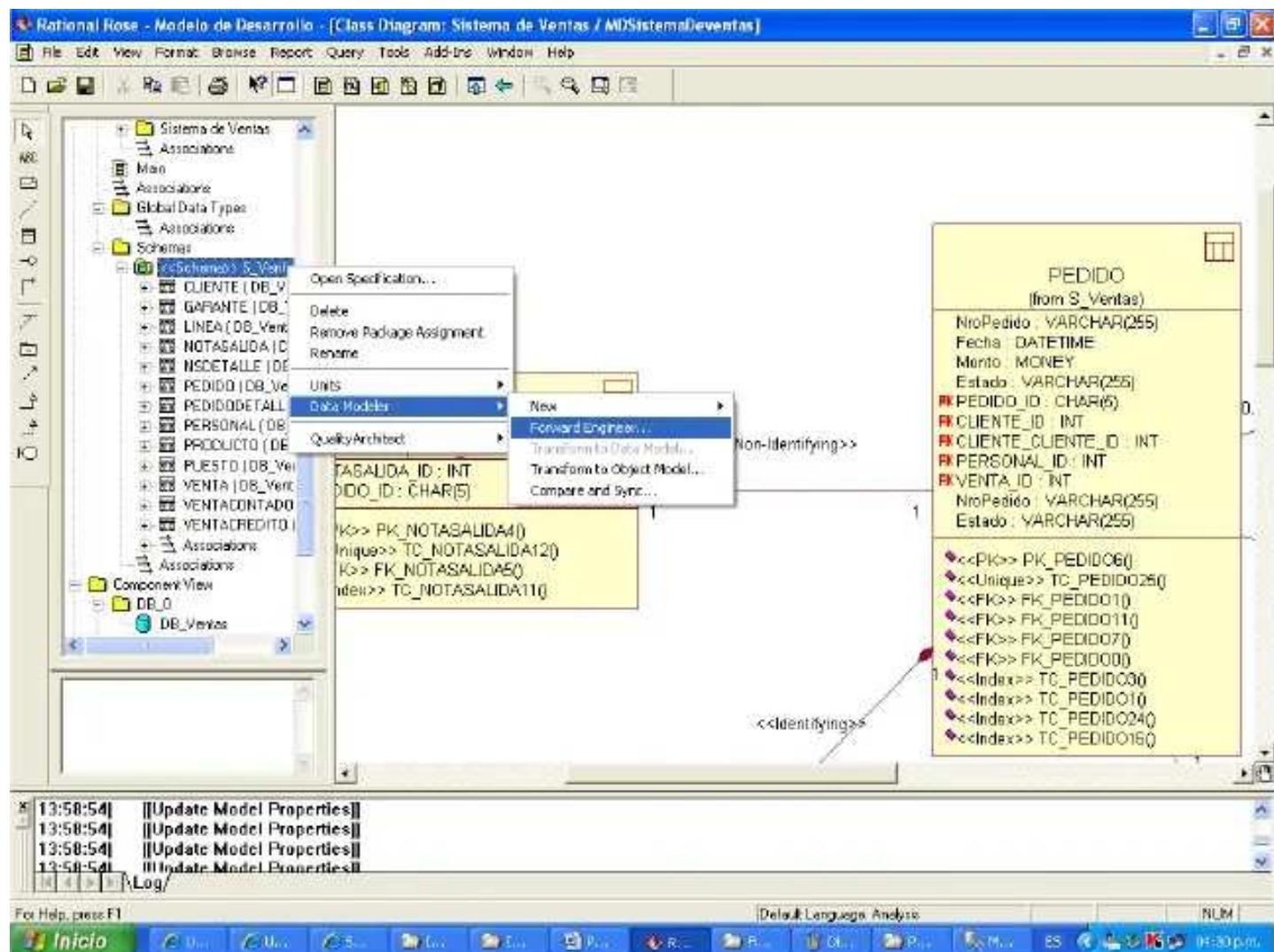
Dar nombre y doble click. Arrastrar al editor todas las tablas generadas, autogenerándose las claves primarias y claves foráneas

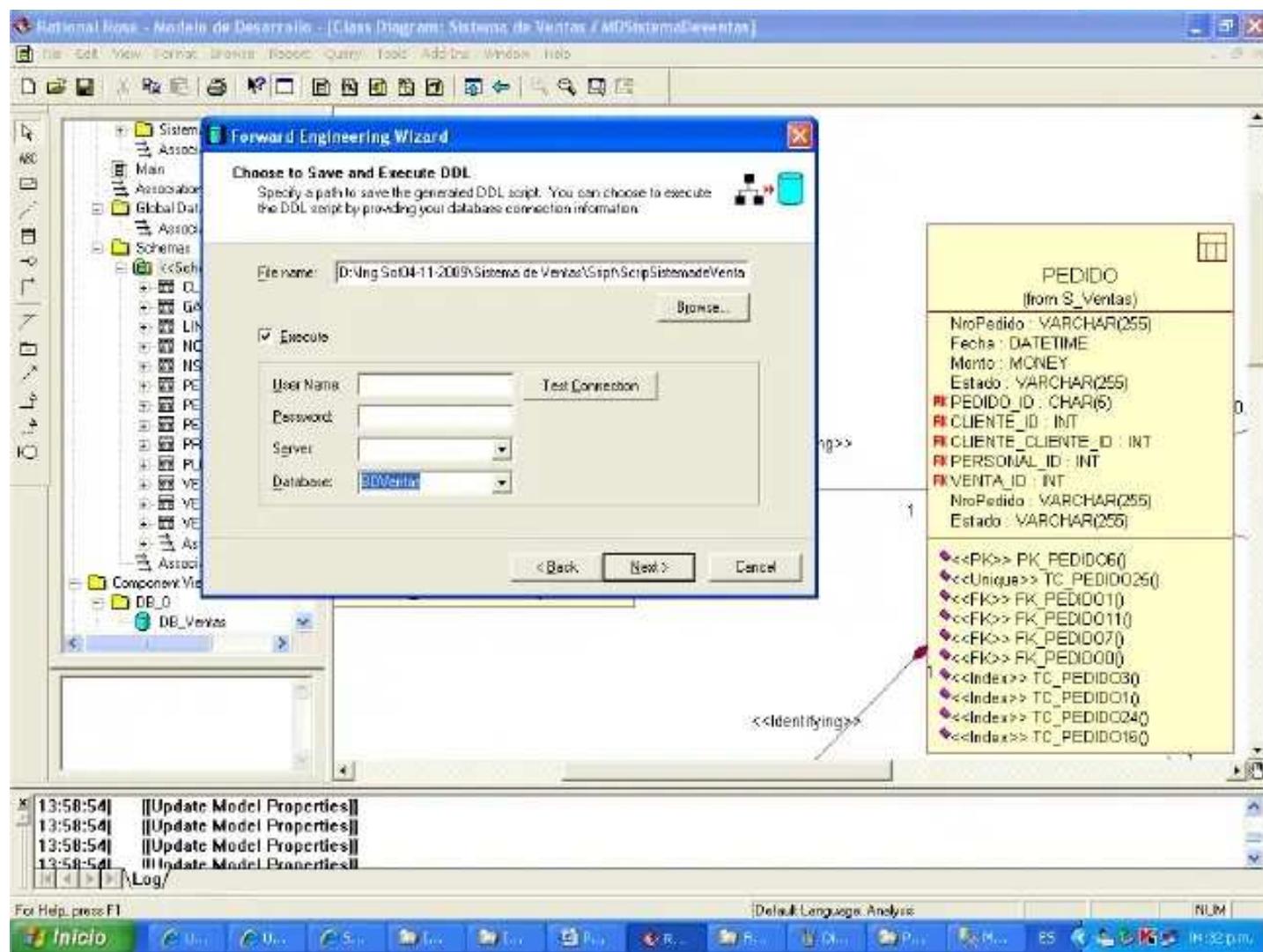
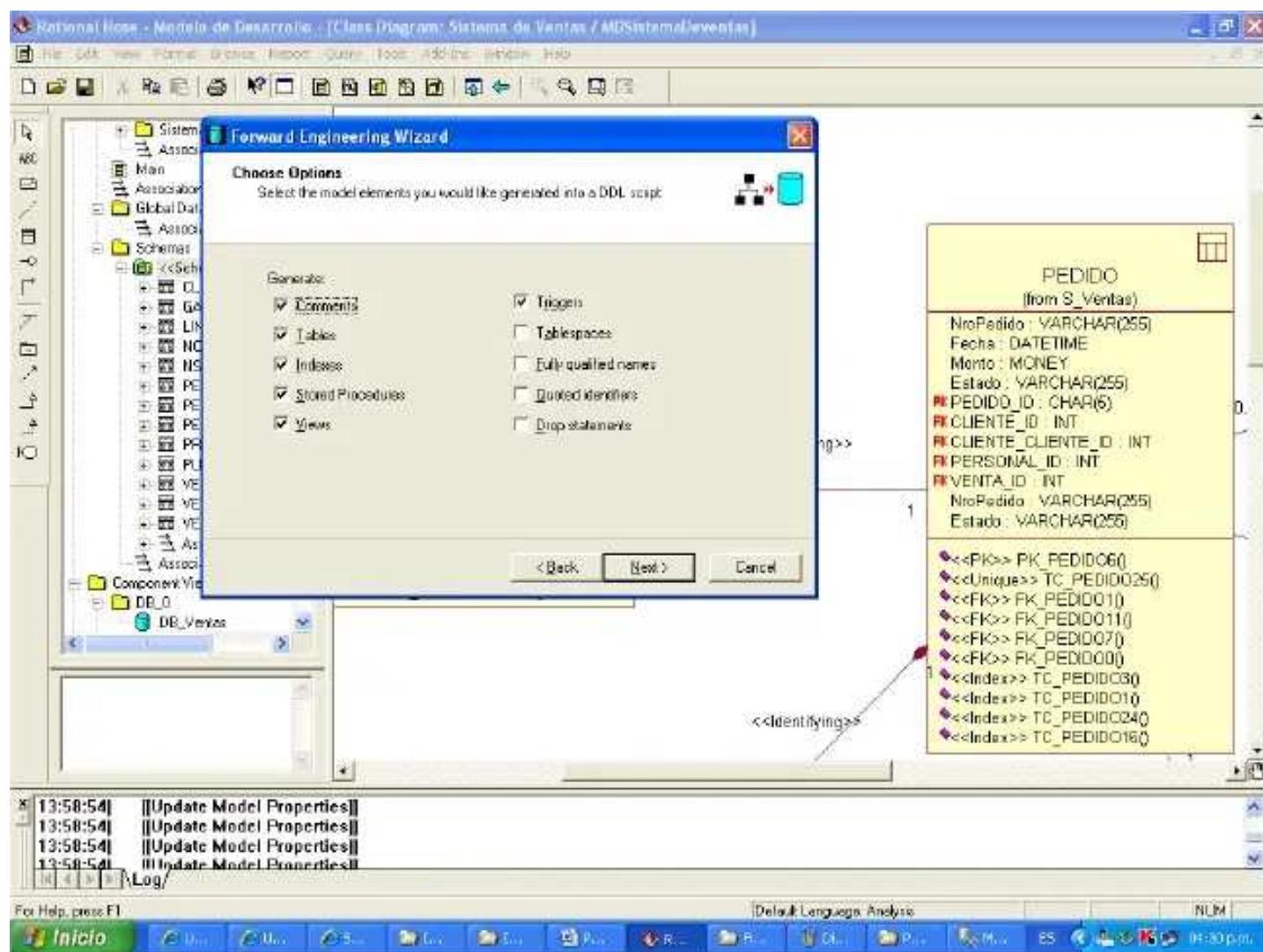


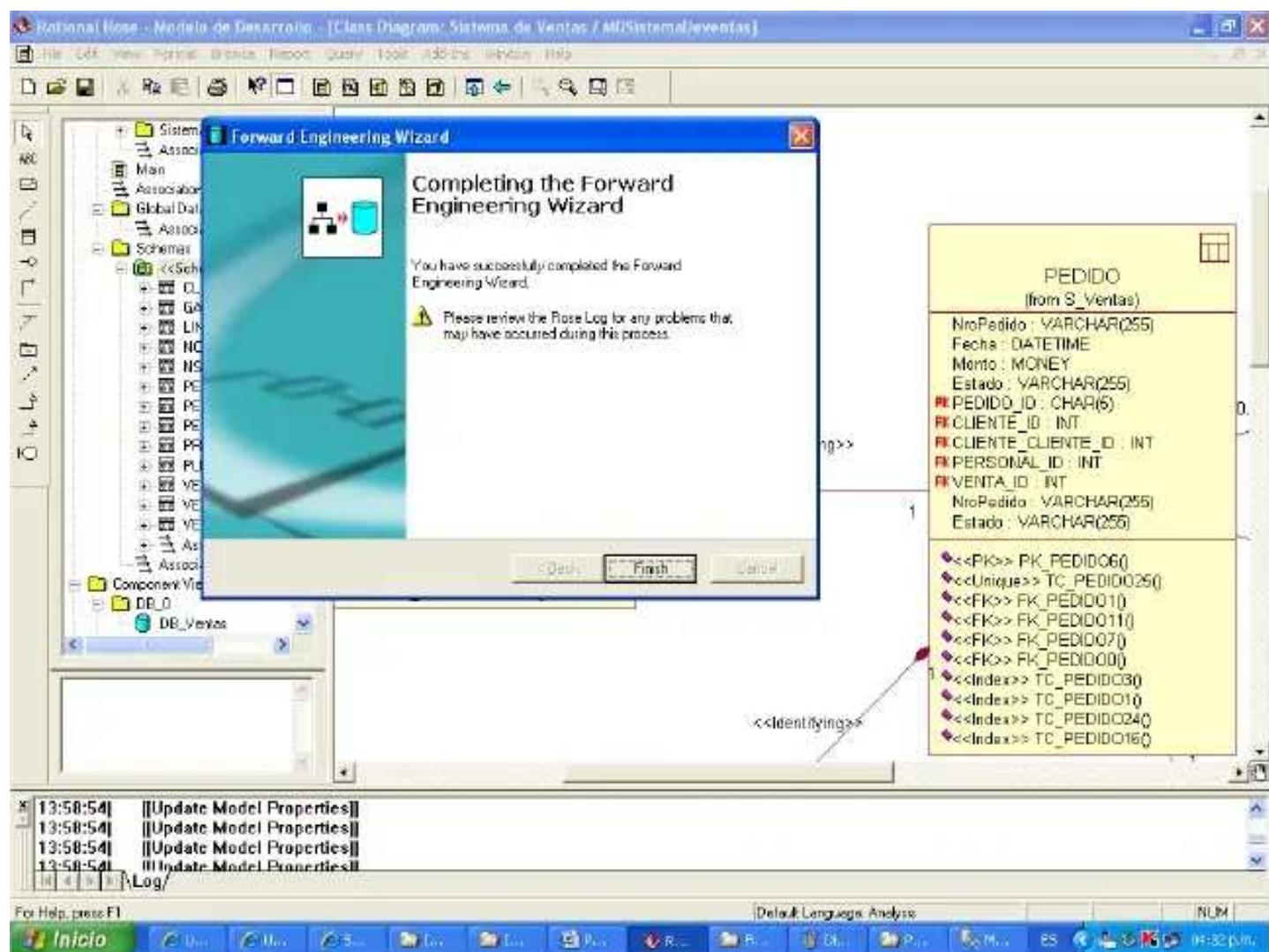
- vi. Si se desea se pueden tipos y longitudes de los atributos de las tablas en el Rational Rose



vii. Migrar el Modelo de datos (Rational Rose) al Gestor de Base de Datos (SQL Server)





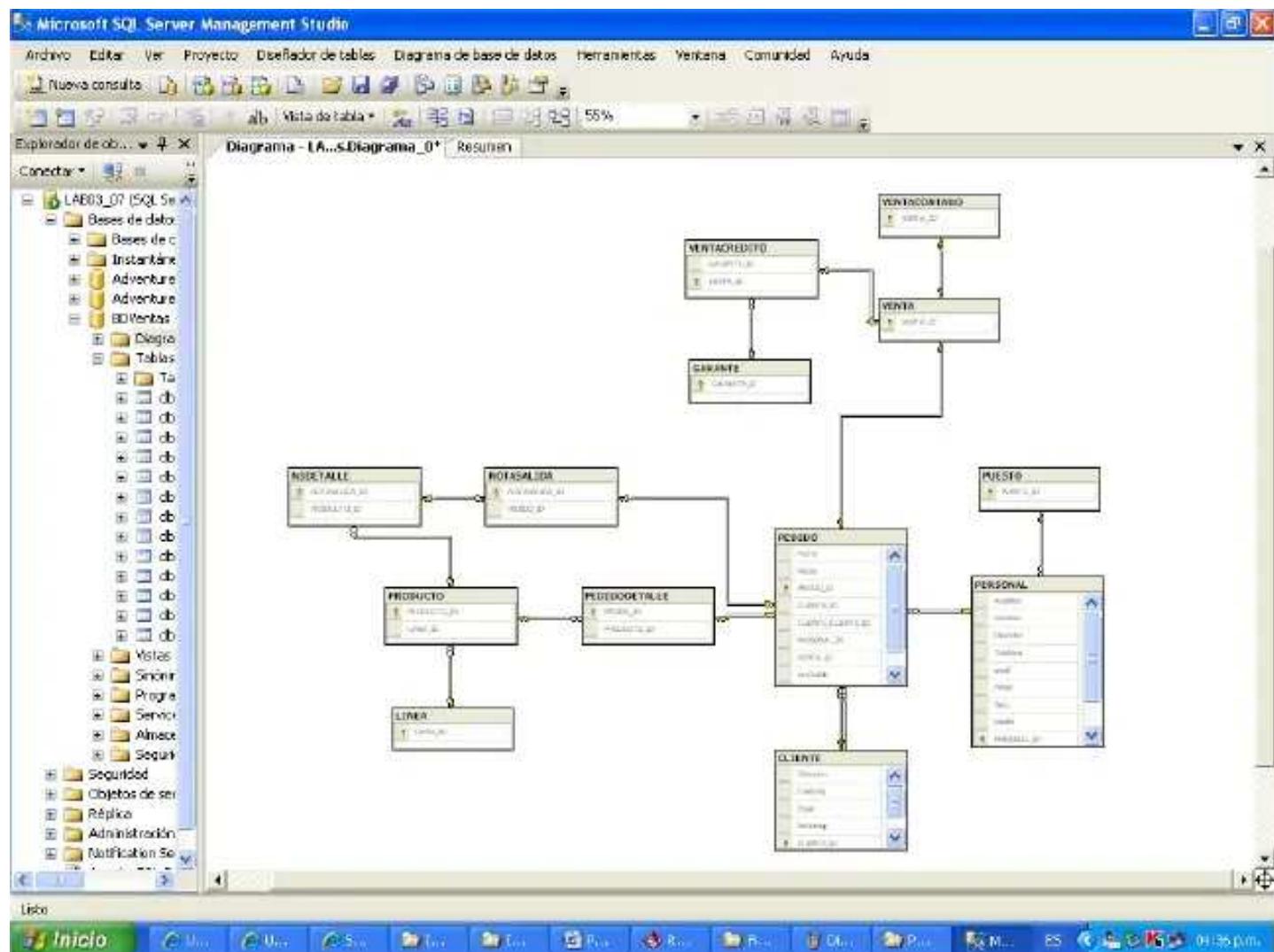


Migran las Tablas al SQL Server en la base de datos vacía creada

Nombre	Esquema	Creado
CLIENTE	dbo	04/12/2009
GARANTE	dbo	04/12/2009
LINEA	dbo	04/12/2009
NOTASALIDA	dbo	04/12/2009
NSDETALLE	dbo	04/12/2009
PEDIDO	dbo	04/12/2009
PEDIDODETALLE	dbo	04/12/2009
PERSONAL	dbo	04/12/2009
PRODUCTO	dbo	04/12/2009
PUESTO	dbo	04/12/2009
VENTA	dbo	04/12/2009
VENTACONTADO	dbo	04/12/2009
VENTACREDITO	dbo	04/12/2009

4º Diagrama de E-R (SQL Server)

Se crea en SQL Server el diagrama de Entidad-Relación



Se puede generar a partir del Script

```
CREATE TABLE PRODUCTO (
    PRODUCTO_ID INT IDENTITY NOT NULL,
    LINEA_ID INT NOT NULL,
    CONSTRAINT PK_PRODUCTO9 PRIMARY KEY NONCLUSTERED (PRODUCTO_ID)
)

GO
```

```
CREATE TABLE CLIENTE (
```

```
Direccion VARCHAR ( 255 ) NOT NULL,  
Telefono VARCHAR ( 255 ) NOT NULL,  
Email VARCHAR ( 255 ) NOT NULL,  
fechalngr DATETIME NOT NULL,  
  
CLIENTE_ID INT IDENTITY NOT NULL,  
CONSTRAINT PK_CLIENTE0 PRIMARY KEY NONCLUSTERED (CLIENTE_ID)  
)
```

GO

```
CREATE TABLE PEDIDO (   
Fecha DATETIME NOT NULL,  
Monto MONEY NOT NULL,  
PEDIDO_ID CHAR ( 5 ) NOT NULL,  
CLIENTE_ID INT NOT NULL,  
CLIENTE_CLIENTE_ID INT NOT NULL,  
PERSONAL_ID INT NOT NULL,  
VENTA_ID INT NOT NULL,  
NroPedido VARCHAR ( 255 ) NOT NULL,  
Estado VARCHAR ( 255 ) NOT NULL,  
CONSTRAINT PK_PEDIDO6 PRIMARY KEY NONCLUSTERED (PEDIDO_ID),  
CONSTRAINT TC_PEDIDO25 UNIQUE NONCLUSTERED (VENTA_ID)  
)
```

GO

```
CREATE TABLE GARANTE (
```

```
    GARANTE_ID INT IDENTITY NOT NULL,  
    CONSTRAINT PK_GARANTE1 PRIMARY KEY NONCLUSTERED (GARANTE_ID)  
)
```

```
GO
```

```
CREATE TABLE VENTACREDITO (  
    GARANTE_ID INT NOT NULL,  
    VENTA_ID INT NOT NULL,  
    CONSTRAINT PK_VENTACREDITO13 PRIMARY KEY NONCLUSTERED (VENTA_ID)  
)
```

```
GO
```

```
CREATE TABLE PEDIDODETALLE (  
    PEDIDO_ID CHAR ( 5 ) NOT NULL,  
    PRODUCTO_ID INT NOT NULL,  
    CONSTRAINT PK_PEDIDODETALLE7 PRIMARY KEY NONCLUSTERED (PEDIDO_ID)  
)
```

```
GO
```

```
CREATE TABLE VENTACONTADO (  
    VENTA_ID INT NOT NULL,  
    CONSTRAINT PK_VENTACONTADO14 PRIMARY KEY NONCLUSTERED  
(VENTA_ID)  
)
```

```
GO
```

```
CREATE TABLE PUESTO (
    PUESTO_ID INT IDENTITY NOT NULL,
    CONSTRAINT PK_PUESTO10 PRIMARY KEY NONCLUSTERED (PUESTO_ID)
)
```

```
GO
```

```
CREATE TABLE NSDETALLE (
    NOTASALIDA_ID INT NOT NULL,
    PRODUCTO_ID INT NOT NULL,
    CONSTRAINT PK_NSDETALLE5 PRIMARY KEY NONCLUSTERED (NOTASALIDA_ID)
)
```

```
GO
```

```
CREATE TABLE PERSONAL (
    Apellidos VARCHAR ( 255 ) NOT NULL,
    nombres VARCHAR ( 255 ) NOT NULL,
    Direccion VARCHAR ( 255 ) NOT NULL,
    Telefono VARCHAR ( 255 ) NOT NULL,
    email VARCHAR ( 255 ) NOT NULL,
    FeNac DATETIME NOT NULL,
    Sexo VARCHAR ( 255 ) NOT NULL,
    sueldo MONEY NOT NULL,
    PERSONAL_ID INT IDENTITY NOT NULL,
```

```
PUESTO_ID INT NOT NULL,  
CONSTRAINT PK_PERSONAL8 PRIMARY KEY NONCLUSTERED (PERSONAL_ID)  
)
```

```
GO
```

```
CREATE TABLE VENTA (  
VENTA_ID INT IDENTITY NOT NULL,  
CONSTRAINT PK_VENTA11 PRIMARY KEY NONCLUSTERED (VENTA_ID)  
)
```

```
GO
```

```
CREATE TABLE NOTASALIDA (  
NOTASALIDA_ID INT IDENTITY NOT NULL,  
PEDIDO_ID CHAR ( 5 ) NOT NULL,  
CONSTRAINT PK_NOTASALIDA4 PRIMARY KEY NONCLUSTERED  
(NOTASALIDA_ID),  
CONSTRAINT TC_NOTASALIDA12 UNIQUE NONCLUSTERED (PEDIDO_ID)
```

```
)
```

```
GO
```

```
CREATE TABLE LINEA (  
LINEA_ID INT IDENTITY NOT NULL,  
CONSTRAINT PK_LINEA3 PRIMARY KEY NONCLUSTERED (LINEA_ID)  
)
```

```
GO
```

```
CREATE INDEX TC_PRODUCTO7 ON PRODUCTO (LINEA_ID)
```

```
GO
```

```
CREATE INDEX TC_PEDIDO3 ON PEDIDO (CLIENTE_CLIENTE_ID)
```

```
GO
```

```
CREATE INDEX TC_PEDIDO1 ON PEDIDO (CLIENTE_ID)
```

```
GO
```

```
CREATE INDEX TC_PEDIDO16 ON PEDIDO (PERSONAL_ID)
```

```
GO
```

```
CREATE INDEX TC_PEDIDO24 ON PEDIDO (VENTA_ID)
```

```
GO
```

```
CREATE INDEX TC_VENTACREDITO5 ON VENTACREDITO (GARANTE_ID)
```

```
GO
```

```
CREATE INDEX TC_PEDIDODETALLE14 ON PEDIDODETALLE (PEDIDO_ID)
```

```
GO
```

```
CREATE INDEX TC_PEDIDODETALLE20 ON PEDIDODETALLE (PRODUCTO_ID)
```

```
GO
```

```
CREATE INDEX TC_NSDETALLE9 ON NSDETALLE (NOTASALIDA_ID)
```

```
GO
```

```
CREATE INDEX TC_NSDETALLE18 ON NSDETALLE (PRODUCTO_ID)
```

```
GO
```

```
CREATE INDEX TC_PERSONAL22 ON PERSONAL (PUESTO_ID)
```

```
GO
```

```
CREATE INDEX TC_NOTASALIDA11 ON NOTASALIDA (PEDIDO_ID)
```

```
GO
```

```
ALTER TABLE PRODUCTO ADD CONSTRAINT FK_PRODUCTO3 FOREIGN KEY (LINEA_ID)  
REFERENCES LINEA (LINEA_ID)
```

```
GO
```

```
ALTER TABLE PEDIDO ADD CONSTRAINT FK_PEDIDO1 FOREIGN KEY  
(CLIENTE_CLIENTE_ID) REFERENCES CLIENTE (CLIENTE_ID)
```

```
GO
```

```
ALTER TABLE PEDIDO ADD CONSTRAINT FK_PEDIDO0 FOREIGN KEY (CLIENTE_ID)  
REFERENCES CLIENTE (CLIENTE_ID)
```

```
GO
```

```
ALTER TABLE PEDIDO ADD CONSTRAINT FK_PEDIDO7 FOREIGN KEY (PERSONAL_ID)  
REFERENCES PERSONAL (PERSONAL_ID)
```

```
GO
```

```
ALTER TABLE PEDIDO ADD CONSTRAINT FK_PEDIDO11 FOREIGN KEY (VENTA_ID)  
REFERENCES VENTA (VENTA_ID)
```

```
GO
```

```
ALTER TABLE VENTACREDITO ADD CONSTRAINT FK_VENTACREDITO2 FOREIGN KEY  
(GARANTE_ID) REFERENCES GARANTE (GARANTE_ID)
```

```
GO
```

```
ALTER TABLE VENTACREDITO ADD CONSTRAINT FK_VENTACREDITO12 FOREIGN KEY  
(VENTA_ID) REFERENCES VENTA (VENTA_ID)
```

```
GO
```

```
ALTER TABLE PEDIDODETALLE ADD CONSTRAINT FK_PEDIDODETALLE6 FOREIGN KEY  
(PEDIDO_ID) REFERENCES PEDIDO (PEDIDO_ID)
```

```
GO
```

```
ALTER TABLE PEDIDODETALLE ADD CONSTRAINT FK_PEDIDODETALLE9 FOREIGN KEY  
(PRODUCTO_ID) REFERENCES PRODUCTO (PRODUCTO_ID)
```

```
GO
```

```
ALTER TABLE VENTACONTADO ADD CONSTRAINT FK_VENTACONTADO13 FOREIGN KEY  
(VENTA_ID) REFERENCES VENTA (VENTA_ID)
```

```
GO
```

```
ALTER TABLE NSDETALLE ADD CONSTRAINT FK_NSDETALLE4 FOREIGN KEY  
(NOTASALIDA_ID) REFERENCES NOTASALIDA (NOTASALIDA_ID)
```

```
GO
```

```
ALTER TABLE NSDETALLE ADD CONSTRAINT FK_NSDETALLE8 FOREIGN KEY  
(PRODUCTO_ID) REFERENCES PRODUCTO (PRODUCTO_ID)
```

GO

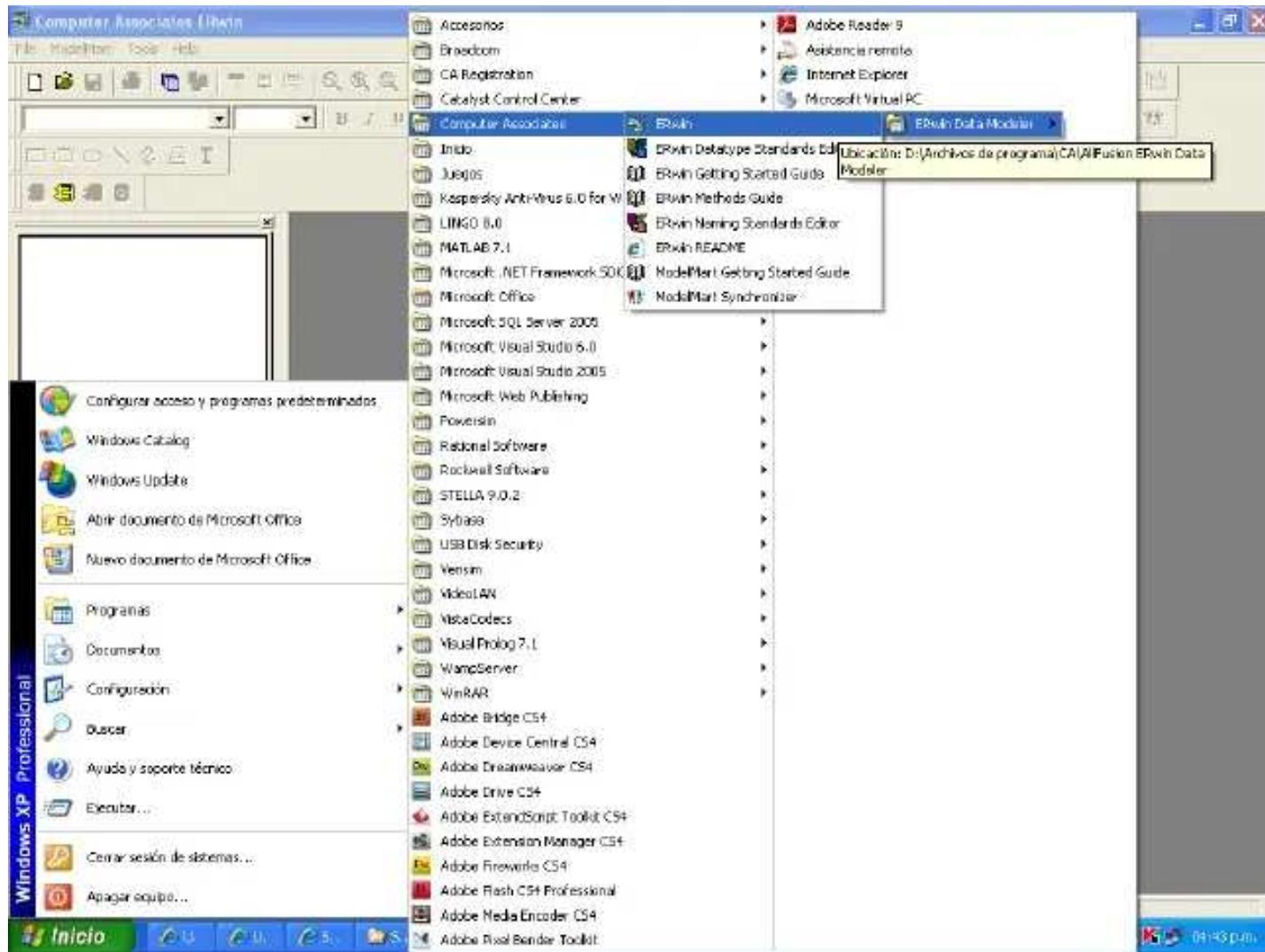
```
ALTER TABLE PERSONAL ADD CONSTRAINT FK_PERSONAL10 FOREIGN KEY (PUESTO_ID)
REFERENCES PUESTO (PUESTO_ID)
```

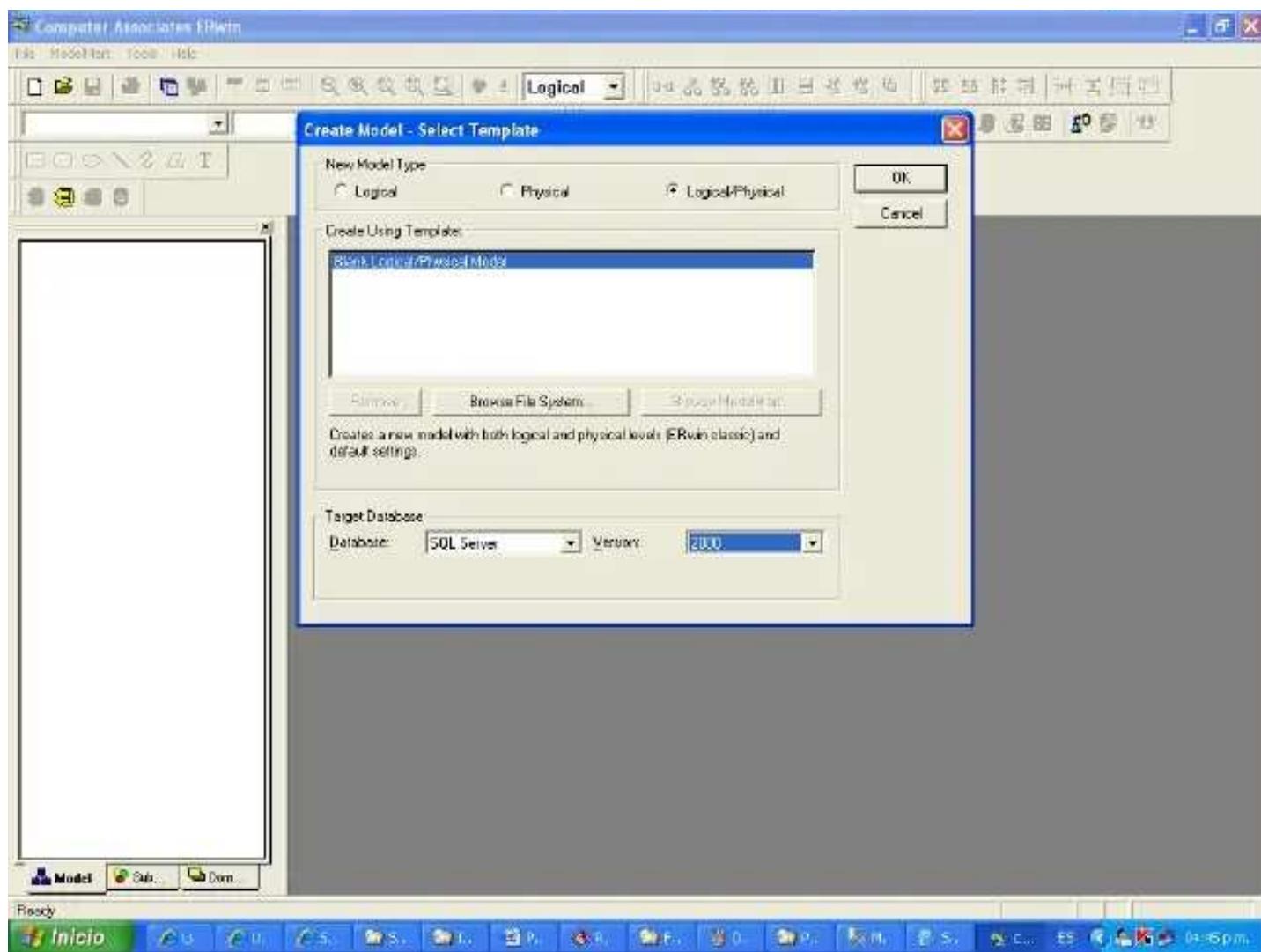
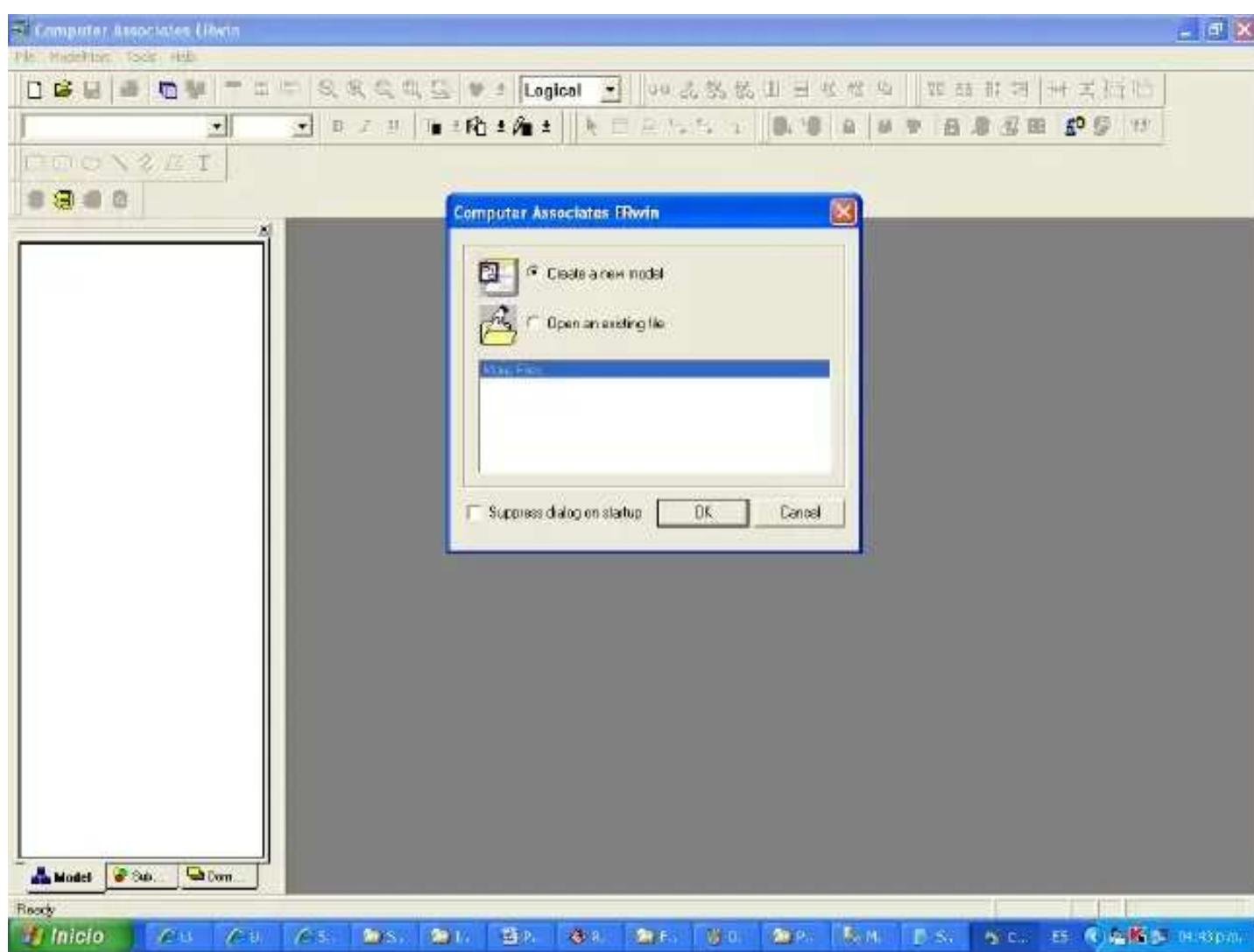
GO

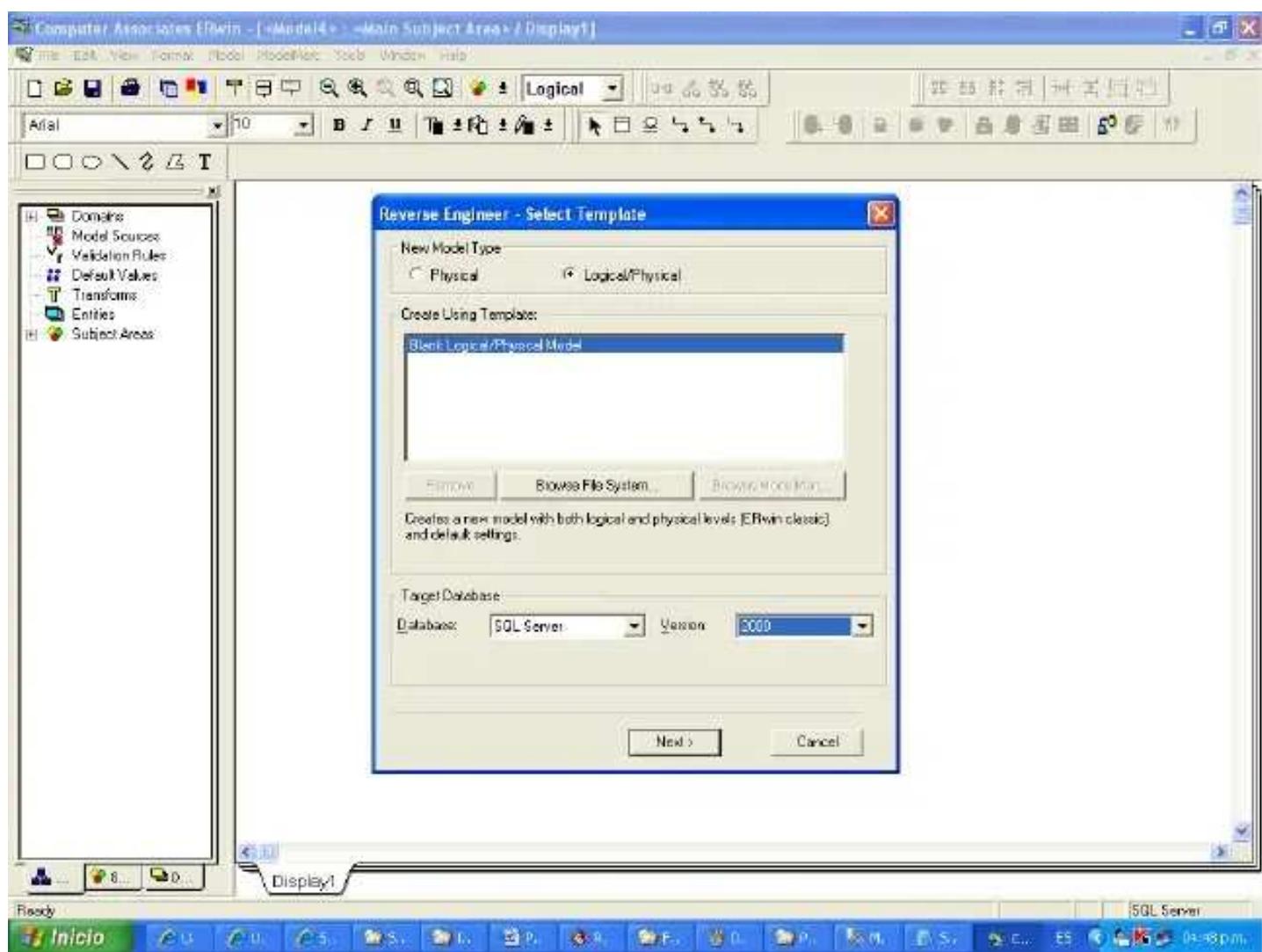
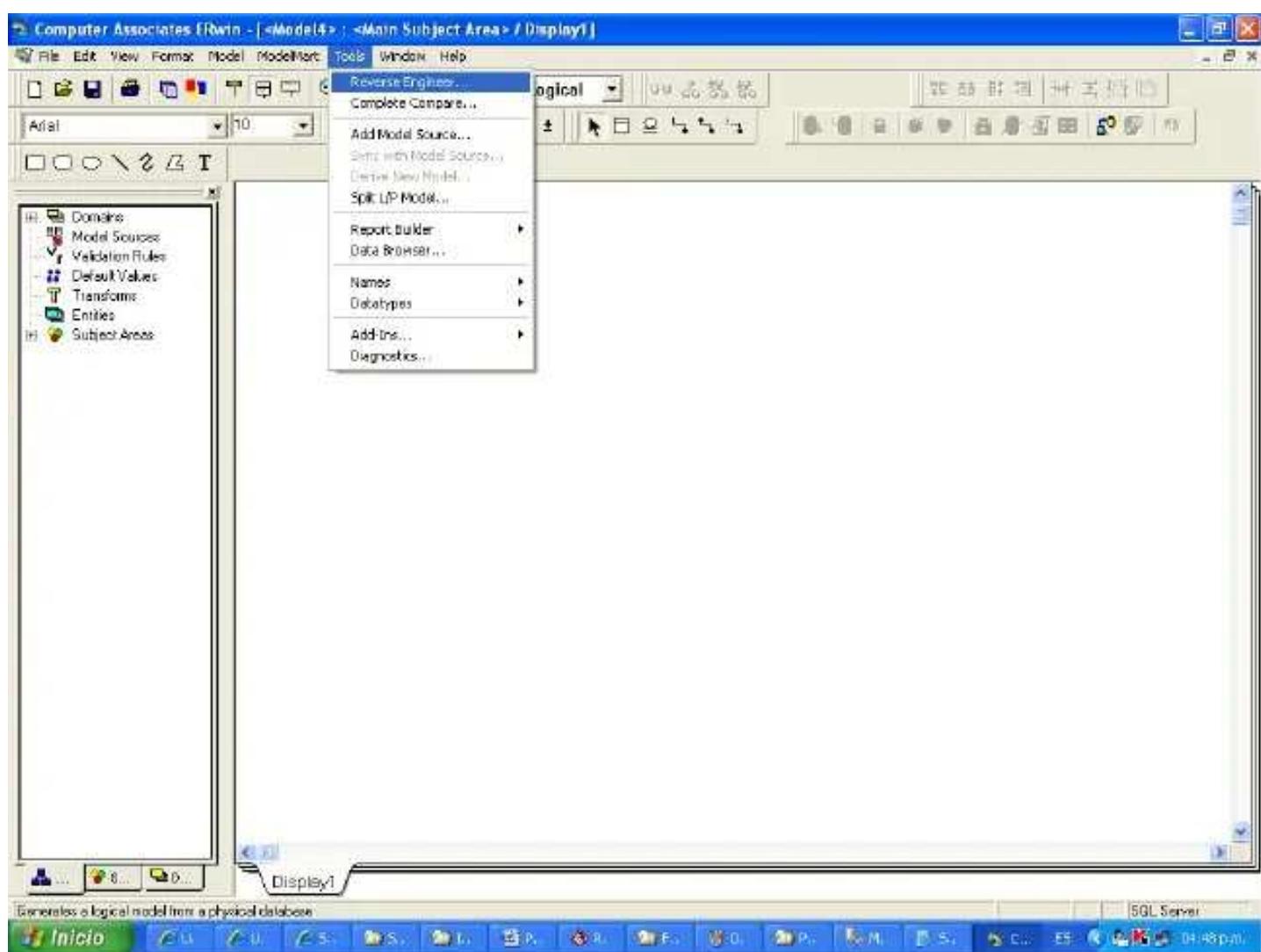
```
ALTER TABLE NOTASALIDA ADD CONSTRAINT FK_NOTASALIDA5 FOREIGN KEY
(PEDIDO_ID) REFERENCES PEDIDO (PEDIDO_ID)
```

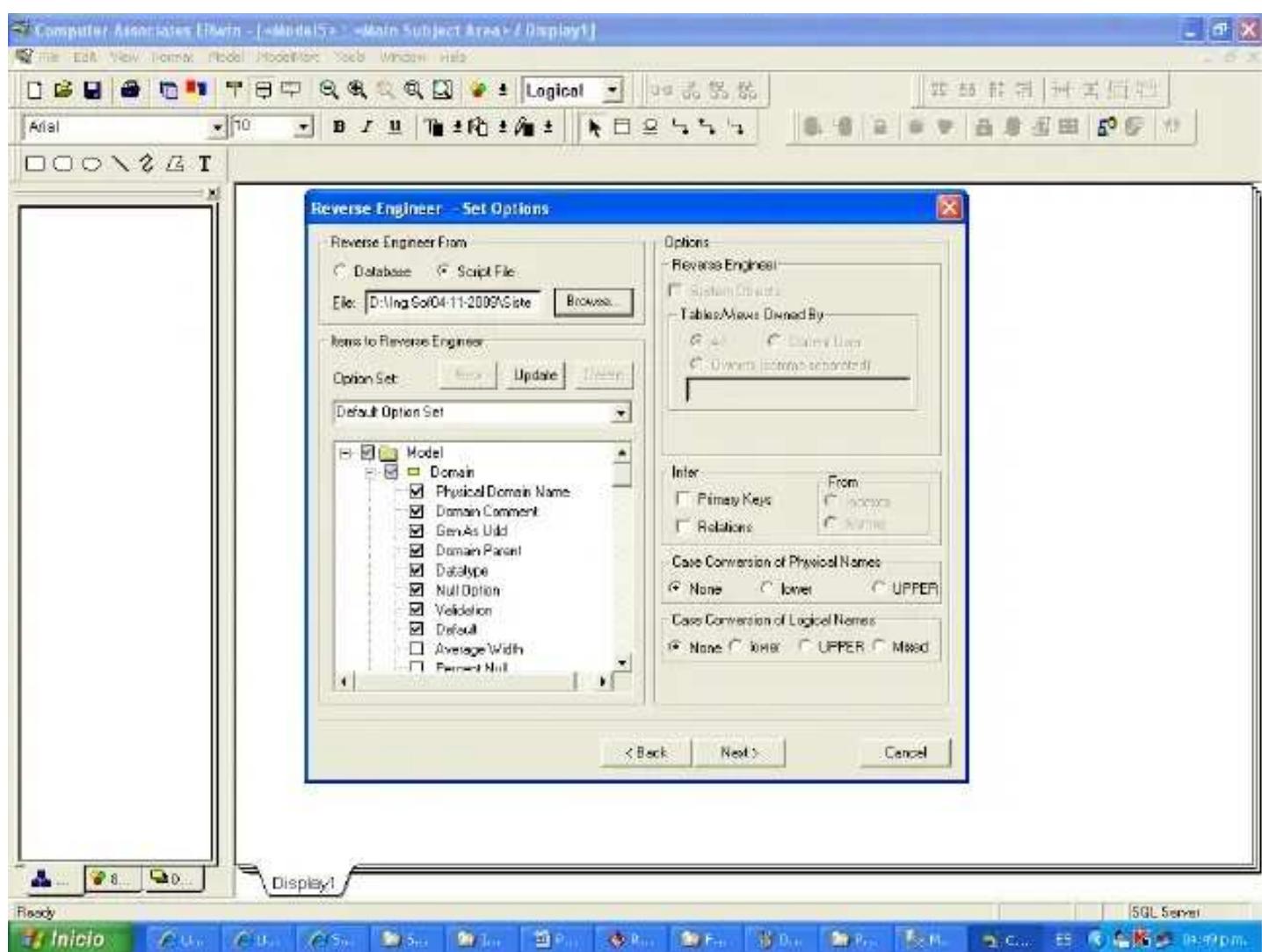
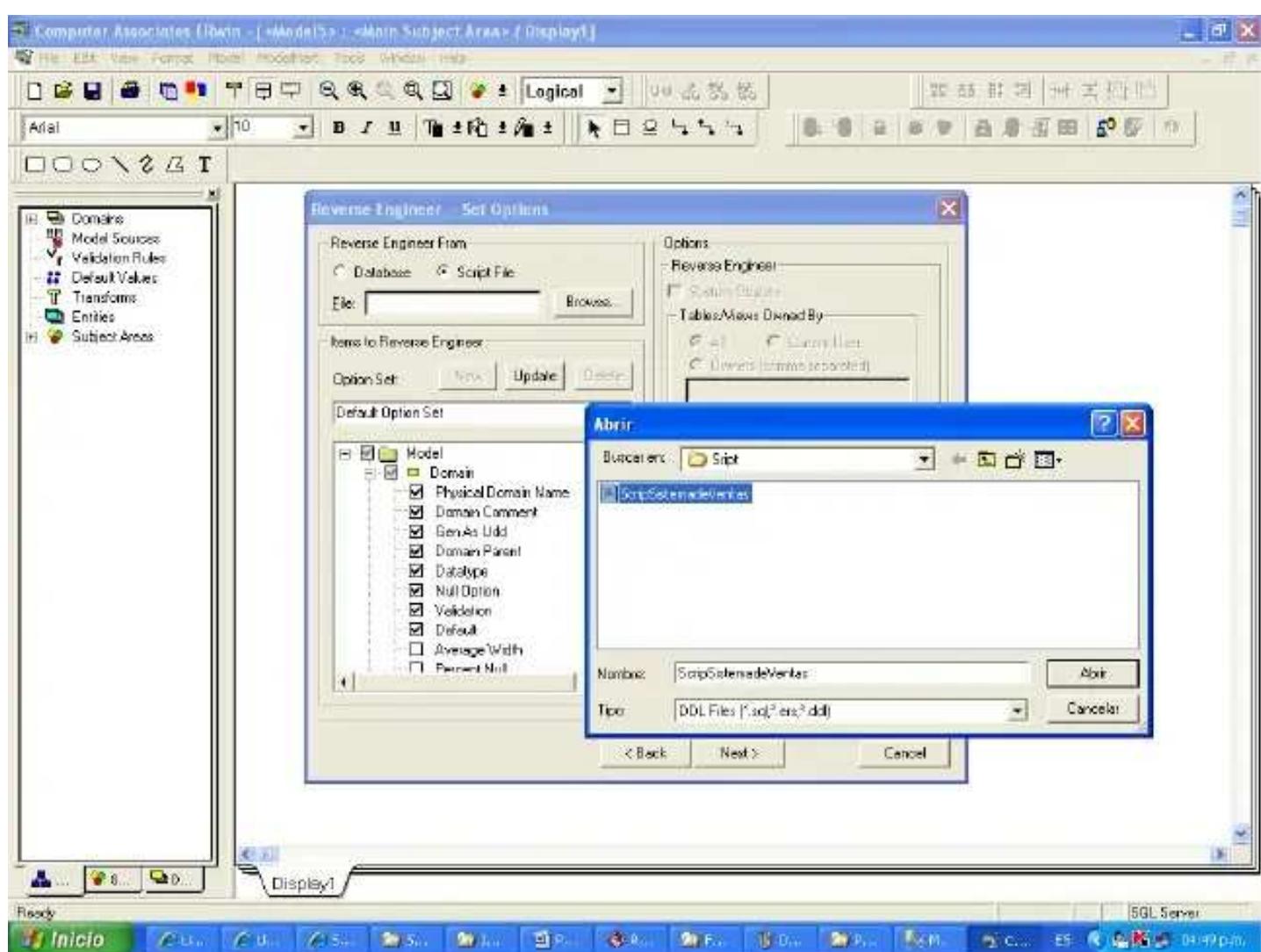
GO

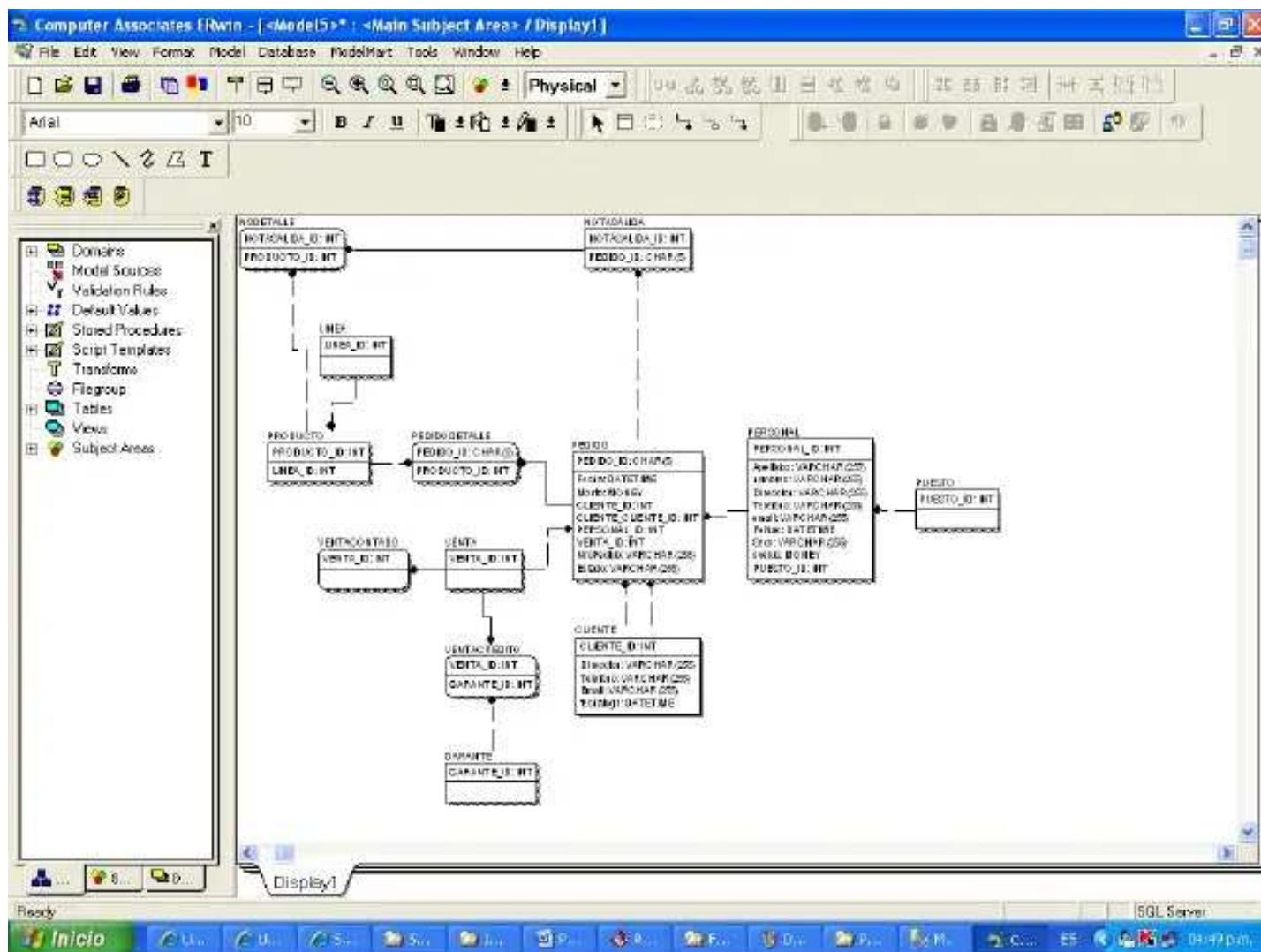
5º Diagrama de Entidades Lógico (ERWIN)



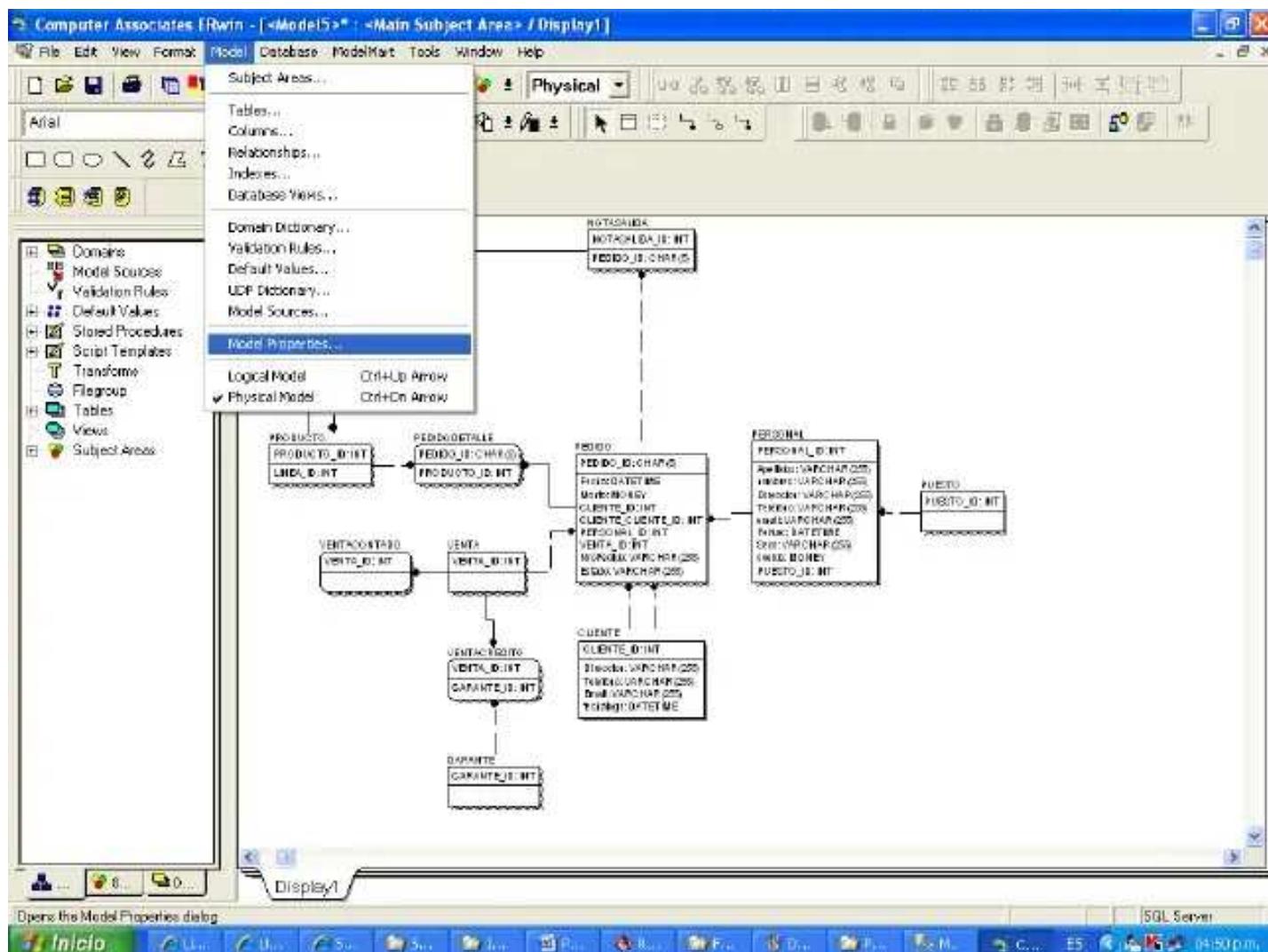


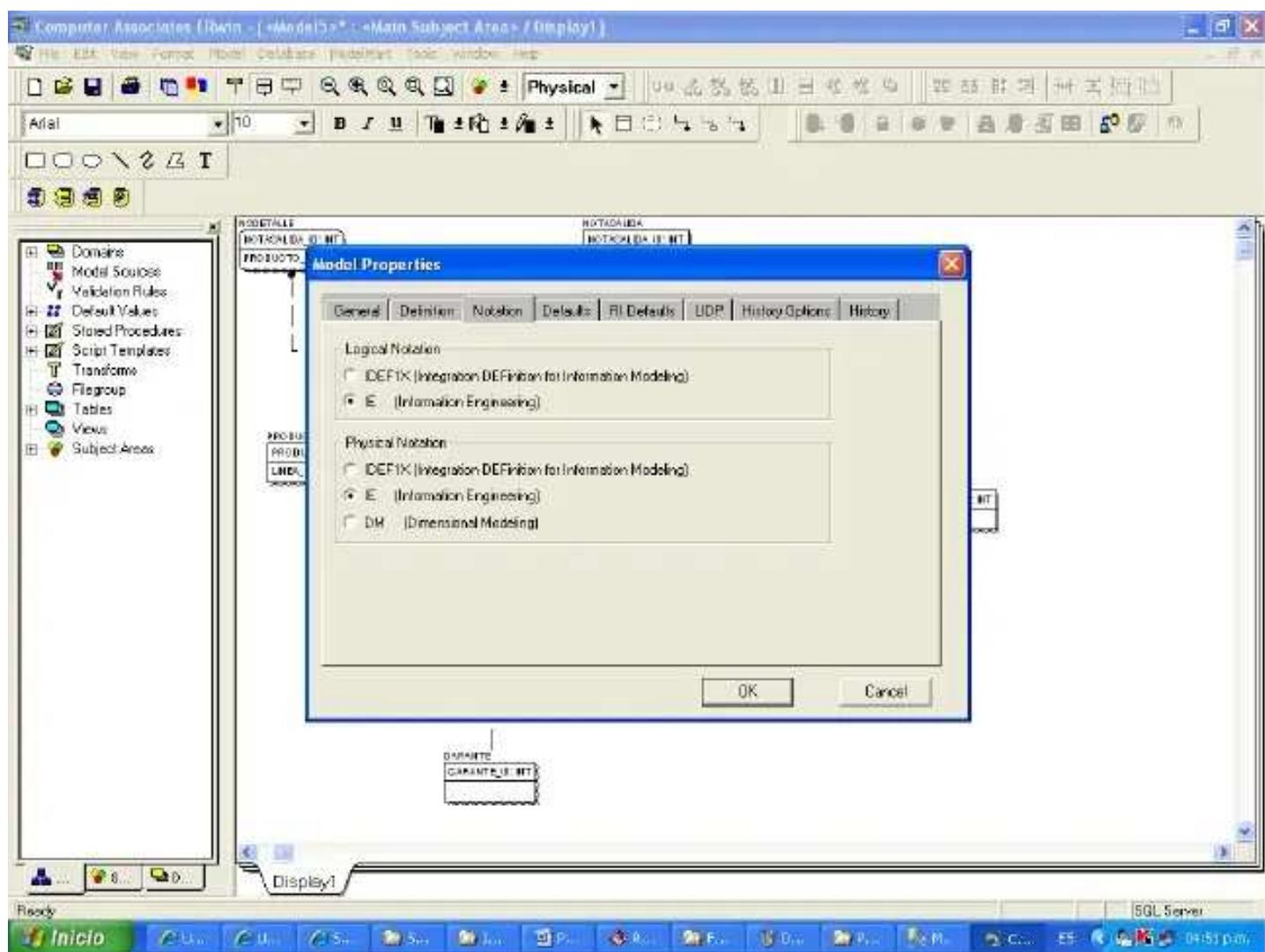




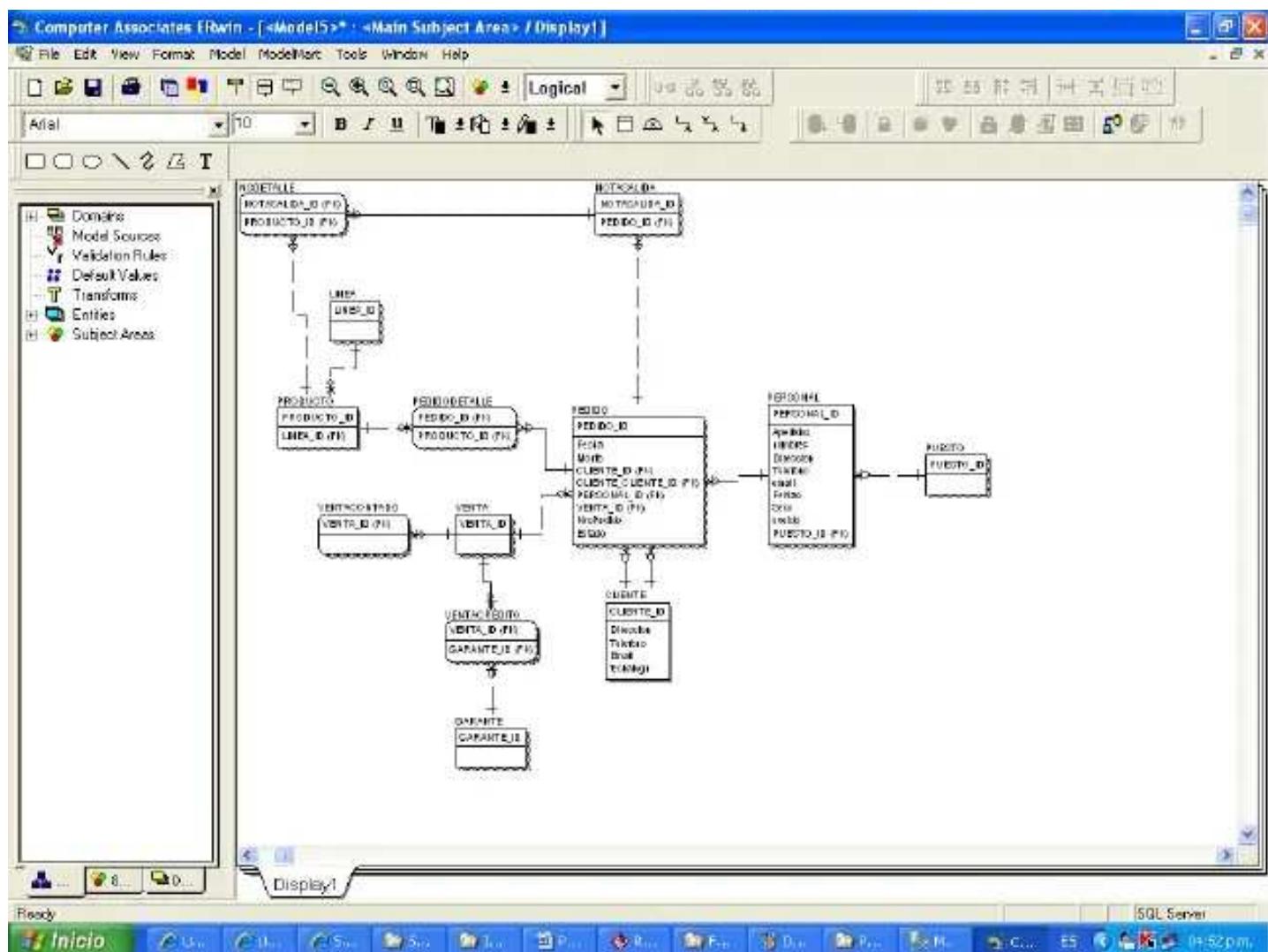


Cambiar los diagramas a la notación de Ingeniería de la Información (James Martin)

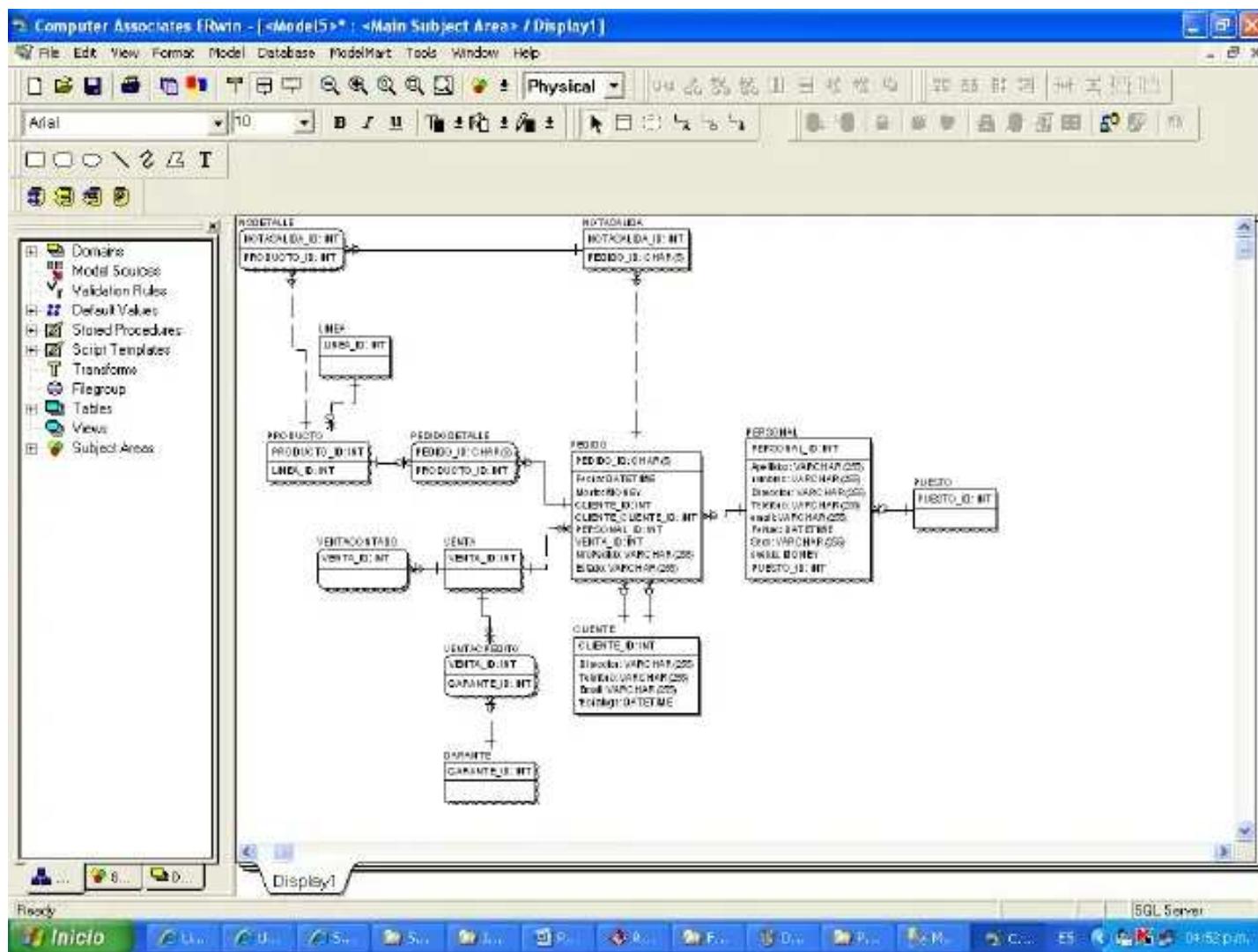




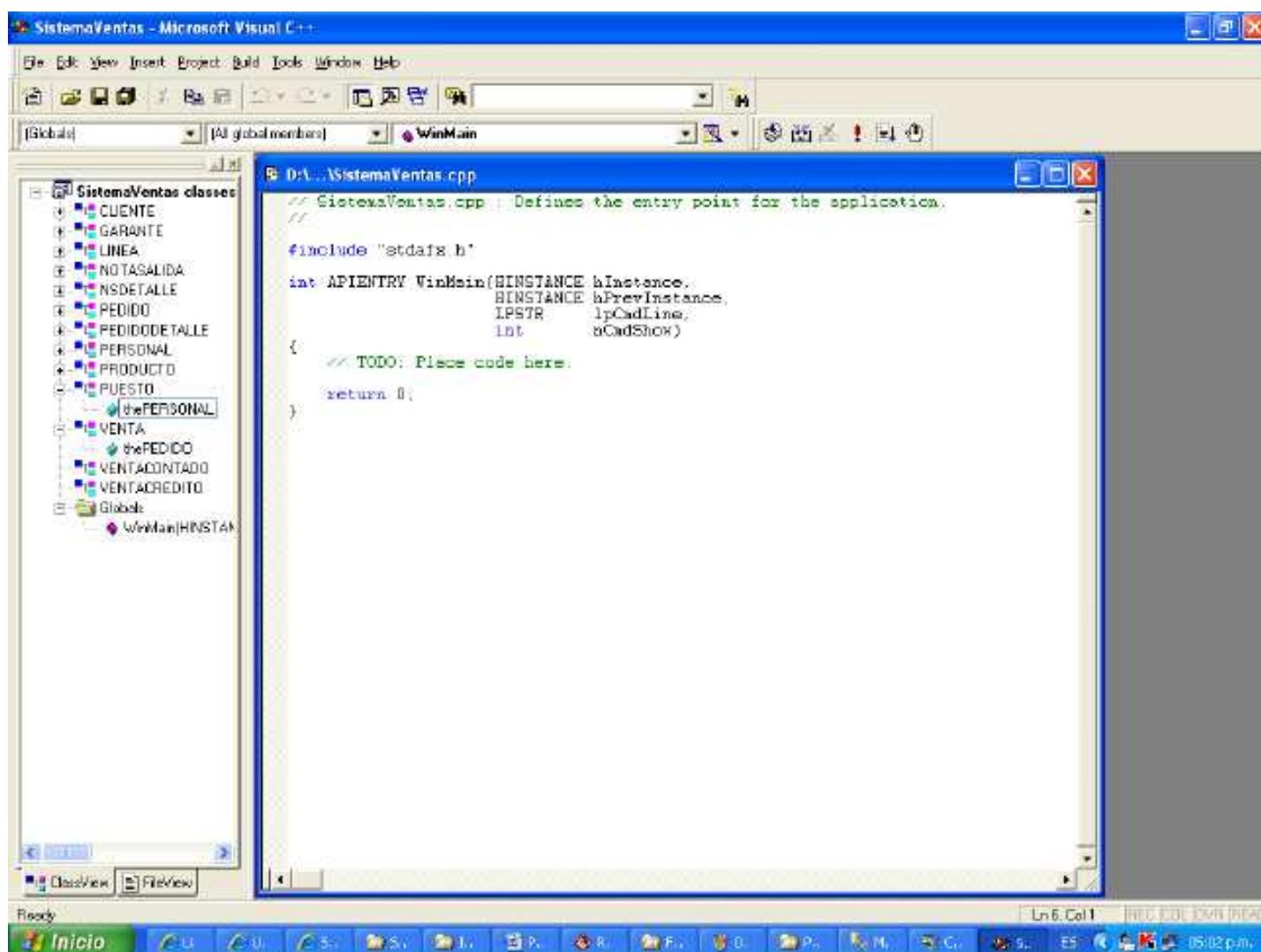
MODELO LOGICO DE ENTIDADES



6º Diagrama de Entidades Físico (ERWIN)



7º Generación de código (Ejemplo Visual C++)



FASE IV: TRANSICION

4.1 Modelo de Pruebas

1º Casos de uso de pruebas

Se seleccionan los casos de uso de procesos (donde hay cálculos, por ejemplo, generar factura), para probar que funcionan sin errores

2º Prueba de la caja blanca

Prueba de la Caja Blanca

La prueba de la caja blanca es un método de diseño de casos de prueba que usa la estructura de control del diseño procedimental para derivar los casos de prueba. Mediante los métodos de prueba de la caja blanca el ingeniero del software puede derivar casos de prueba que:

- Garanticen que se ejercitan al menos una vez todos los *caminos independientes de cada módulo*
- Se ejercitan todas las *decisiones lógicas en sus caras verdaderas y falsas*
- Se ejecutan todos los bucles en sus límites y con sus límites operacionales
- Se ejercitan las estructuras de datos internas para asegurar su validez.

En estas encrucijadas se puede exponer una pregunta razonable: "¿Por qué gastar tiempo y energía probando y preocupándose de las minuciosidades lógicas cuando podemos gastar mejor el esfuerzo asegurando que se han alcanzado los requerimientos del programa?". La respuesta se encuentra en la naturaleza misma de los defectos del software

- *Los errores lógicos y las suposiciones incorrectas son inversamente proporcionales a la probabilidad de que se ejecute un camino del programa.* Los errores tienden a producirse en nuestro trabajo cuando diseñamos o implementamos funciones, condiciones o controles que se encuentran fuera de lo normal. El procedimiento habitual tiende a hacer más comprensible, mientras que el procesamiento de “casos especiales” tiende a caer en el caos.
- *A menudo creemos que un camino lógico tiene pocas posibilidades de ejecutarse cuando, de hecho, se puede ejecutar de forma regular.* El flujo lógico de un programa a veces no es nada intuitivo, lo que significa que nuestras suposiciones intuitivas sobre el flujo de control y los datos nos pueden llevar a tener errores de diseño que solo se descubren cuando comienza la prueba del camino.
- *Los errores tipográficos son aleatorios.* Cuando se traduce un programa a código fuente de un lenguaje de programación, es muy probable que se den algunos errores de escritura. Muchos serán descubiertos por los mecanismos de comprobación de sintaxis, pero otros permanecerán indetectados hasta que comience la prueba. Es igual de probable que haya un error tipográfico en un oscuro camino lógico que en un camino principal.

Cada una de estas razones nos da un argumento para llevar a cabo las pruebas de caja blanca. La prueba de caja negra, sin tener en cuenta como sea de completa, puede pasarse los tipos de errores que acabamos de señalar. Como estableció Beizer: “Los errores se esconden en los rincones y se aglomeran en los límites”. Es mucho más fácil de descubrirlos con la prueba de caja blanca.

Pruebas del Camino Básico.

La prueba del camino básico es una técnica de prueba de la caja blanca propuesta inicialmente por Tom McCabe. El método del camino básico permite al diseñador de casos de prueba derivar una medida de complejidad lógica de un diseño procedural y usar esa medida como guía para la definición de un conjunto básico de caminos de ejecución. Los casos de prueba derivados del conjunto básico garantizan que durante la prueba se ejecuta por lo menos una vez cada sentencia del programa.

Notación de grafo de flujo

Antes de considerar el método del camino básico se debe introducir una sencilla notación para la representación del flujo de control, denominada *grafo de flujo*. El grafo de flujo representa el flujo de control lógico mediante la notación ilustrada en la figura 2.1. Cada construcción estructurada tiene un correspondiente símbolo en el grafo de flujo.

Para ilustrar el uso de un grafo de flujo, consideraremos la representación del diseño procedural de la figura 2.2.

Aquí, se usa un diagrama de flujo para representar la estructura de control de un programa.

Cuando en un diseño procedural se encuentran condiciones compuestas, la generación del grafo de flujo se hace un poco más complicada. Una condición compuesta se da cuando aparecen uno o más operadores booleanos (OR, AND, NAND, NOR lógicos) en una sentencia condicional.

Prueba de Bucles

Los bucles son la piedra angular de la mayoría de los algoritmos implementados en software. Y por ello debemos prestarle normalmente un poco de atención cuando llevamos a cabo la prueba del software.

La prueba de bucles es una técnica de prueba de caja blanca que se centra exclusivamente en la validez de las construcciones de bucles. Se pueden definir cuatro clases diferentes de bucles: Bucles simples, Bucles concatenados, Bucles Anidados, y bucles no estructurado.

Además de un análisis del camino básico que aísla todos los caminos de un bucle, se recomienda un conjunto especial de pruebas adicionales para cada tipo de bucles. Estas pruebas buscan errores de inicialización, errores de indexación o de incremento y errores en los límites de los bucles.

Bucles Simples: A los bucles simples se les debe aplicar el siguiente conjunto de pruebas, donde n es el número máximo de pasos permitidos por bucle.

- Saltar totalmente el bucle.
- Pasar una sola vez por el bucle.
- Pasar dos veces por el bucle.

- Hacer m pasos por el bucle con $m < n$.
- Hacer $n-1, n$ y $n+1$ pasos por bucle.

Bucles Anidados: Si extendiéramos el enfoque de prueba de los bucles simples a los bucles anidados, el numero de posibles pruebas crecería geométricamente a medida que aumenta el nivel de anidamiento. Esto nos llevaría a un numero impracticable de pruebas. Beizer sugiere un enfoque que ayuda a reducir el número de pruebas:

- Comenzar en el bucle más interior. Disponer todos los demás bucles en sus valores mínimos.
- Llevar a cabo las pruebas de bucles simples con el bucle mas interno mientras se mantiene los bucles exteriores con los valores mínimos para sus parámetros de iteración. Añadir otras pruebas para los valores fuera de rango o para valores excluidos.
- Progresar hacia afuera, llevando a cabo las pruebas para el siguiente bicle, pero manteniendo todos los demás bucles exteriores en sus valores mínimos y los demás bucles anidados con valores “típicos”.
- Continuar hasta que se hayan probado todos los bucles.

Bucles Concatenados: los bucle concatenados se pueden probar mediante el enfoque anteriormente definido para los bucles simple, mientras que cada uno de los bucles sea independiente del resto. Por ejemplo, si hay dos bucles concatenados y se usa el contador del bucle 1 como valor inicial del bucle 2, entonces los bucles *no* son independientes. Cuando los bucles no son independientes, se recomienda usar el enfoque aplicado para los bucles anidados.

Bucles No Estructurados: Siempre que sea posible, esta clase de bucles se deben rediseñar para que se ajuste a las construcciones de la programación estructurada.

3º Prueba de la caja negra

Prueba de la Caja Negra

Los métodos de prueba de la caja negra se centran en los requerimientos funcionales del software. O sea, la prueba de la caja negra permite al ingeniero del software derivar conjuntos de condiciones de entrada que ejerciten completamente todos los requerimientos funcionales de un programa. La prueba de la caja negra *no* es una alternativa a las técnicas de prueba de la caja blanca. Mas bien se trata de un enfoque complementario que intenta descubrir diferentes tipos de errores que los métodos de la caja blanca.

La prueba de la caja negra intenta encontrar errores de las siguientes categorías:

- Funciones incorrectas o ausentes
- Errores de interfaz
- Errores en estructura de datos o en acceso a base de datos externas
- Errores de rendimiento
- Errores de inicialización y de terminación.

A diferencia de la prueba de la caja blanca, que se lleva a cabo previamente en el proceso de prueba, la prueba de la caja negra tiende a ser aplicadas en posteriores fases de prueba. Ya que la prueba de la caja negra intencionadamente ignora la estructura de control, concentra su atención en el dominio de la información. Las pruebas se diseñan para responder a las siguientes preguntas:

- ¿Cómo se prueba la validez funcional?
- Qué clase de entrada compondrán unos buenos casos de prueba?

- ¿Es el sistema particularmente sensible a ciertos valores de entrada?
- ¿De qué forma están aislados los límites de una clase de datos?
- ¿Qué volúmenes y niveles de datos tolerará el sistema?
- ¿Qué efectos sobre la operación del sistema tendrán combinaciones específicas de datos?

Mediante las técnicas de prueba de la caja negra se derivan un conjunto de casos de prueba que satisfacen los siguientes criterios:

- Casos de prueba que reducen, en un coeficiente que es mayor que uno, el numero de casos de prueba adicionales que se deben diseñar para alcanzar una prueba razonable,
- Casos de prueba que nos dicen algo sobre la presencia o ausencia de clases de errores asociados solamente con la prueba en particular que se encuentra disponible.

4º Elaboración de Manuales

Se elaboran los siguientes manuales del sistema:

- i) De Instalación
- ii) Del Sistema
- iii) De errores
- iv) De ayuda

5º capacitación de los usuarios del sistema

Antes de poner en marcha o servicio el sistema de información se capacita a los usuarios del mismo.

4.2 Modelo de Implantación

1º Implantación del sistema (Puesta en marcha o servicio)

Poner en servicio el sistema de información creado, el cual debe estar sin errores.

2º Mantenimiento del Sistema

Una vez puesto en servicio del sistemas de información se realiza el mantenimiento del mismo de acuerdo a los nuevos requerimientos de los usuarios.

FIN