3.3 Introducción al software Rational Rose

El software Rational Rose de Rational Software Corporation proporciona el soporte para dos aspectos elementales de la ingeniería de software: modelado orientado a objetos y desarrollo iterativo controlado.

En el proyecto se ha usado el software Rose para realizar el modelado del sistema, en ningún momento para el desarrollo, y ese aspecto es el que se tratará aquí.

El software Rose soporta diagramas UML, entre otros tipos, como OMT y Booch. UML es el lenguaje de modelado usado para el sistema.

Esta sección es una pequeña introducción al uso del software Rose para su uso en desarrollo de modelos UML.

Si al arrancar la aplicación se carga un modelo, aparecerá una pantalla similar a la de figura 3.3.1.

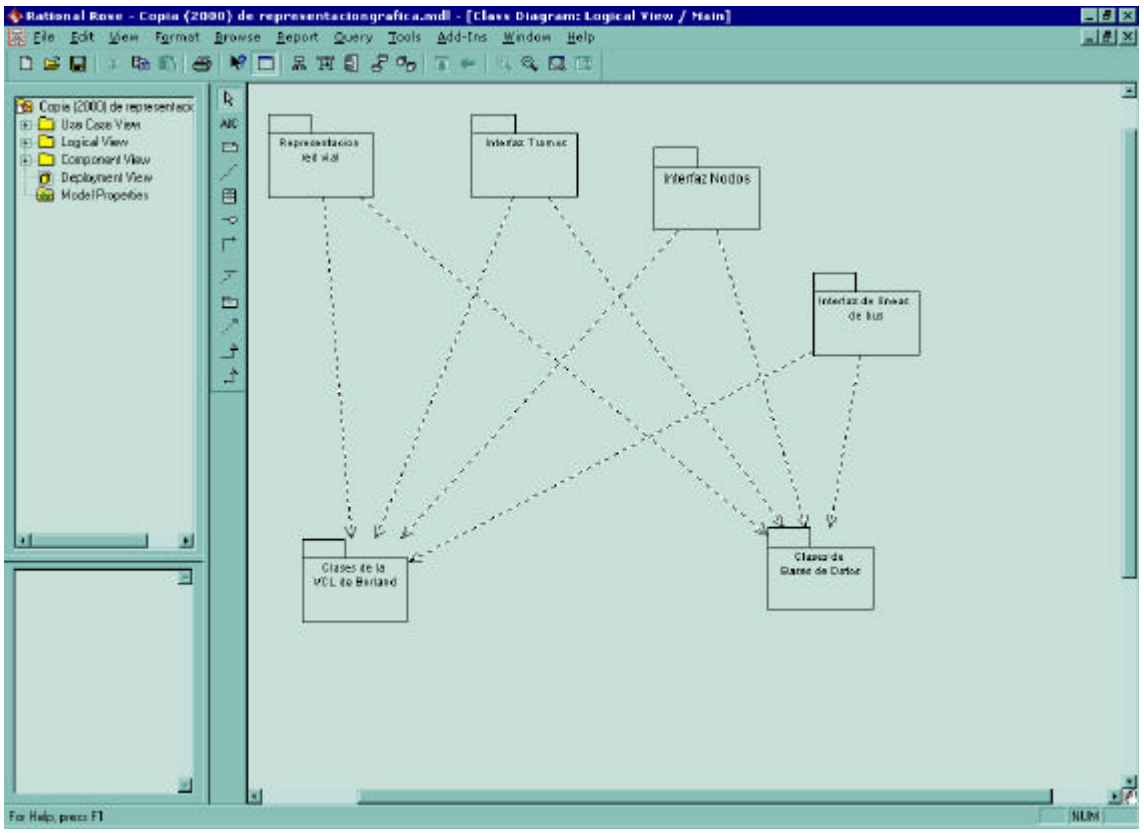

Figura 3.3.1: Vista general inicial

Los modelos se guardan en un archivo de extensión .mdl. Todos los diagramas, ya sean de clases, de objetos, de secuencias, de casos de uso, se guardan en un único archivo .mdl.

Por defecto se muestra, dentro de la vista lógica (Logical View), el diagrama de clases. En la parte izquierda de la pantalla hay un navegador que contiene las carpetas Use Case View, Logical View, Component View y Deployment View. Debajo de cada una

de estas carpetas pueden aparecer subcarpetas si se agrupan los elementos del modelo en paquetes. En este caso, cada paquete sería una subcarpeta.

Bajo la carpeta Use Case View aparecen todos los actores y casos de uso definidos para el modelo. Además aparecen todos los diagramas de casos de uso y de secuencia que se hayan creado.

Bajo la carpeta Logical View aparecen todas las clases y relaciones definidas además de los distintos diagramas de clases que se tengan.

Si se hace doble click sobre el nombre de un diagrama, éste se muestra en la pantalla, en la parte derecha. Una vez que se tiene activo un diagrama, se pueden añadir elementos al mismo usando la barra de herramientas que está a la derecha del navegador de vistas.

Para añadir a un diagrama un nuevo elemento basta con hacer click sobre su icono en la barra de herramientas y a continuación hacer de nuevo click en el lugar donde se desea crear dicho elemento. Si el elemento ya existe, puede arrastrarse desde el navegador hacia el diagrama. Si se hace doble click sobre cualquier elemento en el navegador o en el diagrama en el que aparece aparece un ventana con sus características.

Pueden visualizarse varios diagramas simultáneamente, pero sólo uno puede estar activo. Los iconos que aparecen en la barra de herramientas varían según el tipo de diagrama que esté activo. Por ejemplo, si el diagrama activo es un diagrama de clases, no aparecerá ningún actor, pero sí aparecerán clases y todas las relaciones posibles entre ellas, etcétera. En definitiva, todos los elementos que se pueden incluir en cada uno de los diagramas son los que aparecen cuando ese diagrama se encuentra seleccionado.

Diagrama de clases

Los diagramas de clases incluyen clases y las relaciones entre ellas. Opionalmente pueden contener paquetes. En caso de contener paquetes, haciendo doble click sobre el icono del paquete sobre el diagrama se accede al subdiagrama bajo dicho paquete.

Clases

Cuando se crea una clase y se hace doble click sobre ella aparece el menú de características. El campo Name contiene el nombre de la clase. En el campo Stereotype se puede especificar el estereotipo deseado, si es necesario.

En el campo Export Control se define la visibilidad fuera del paquete donde se han definido.

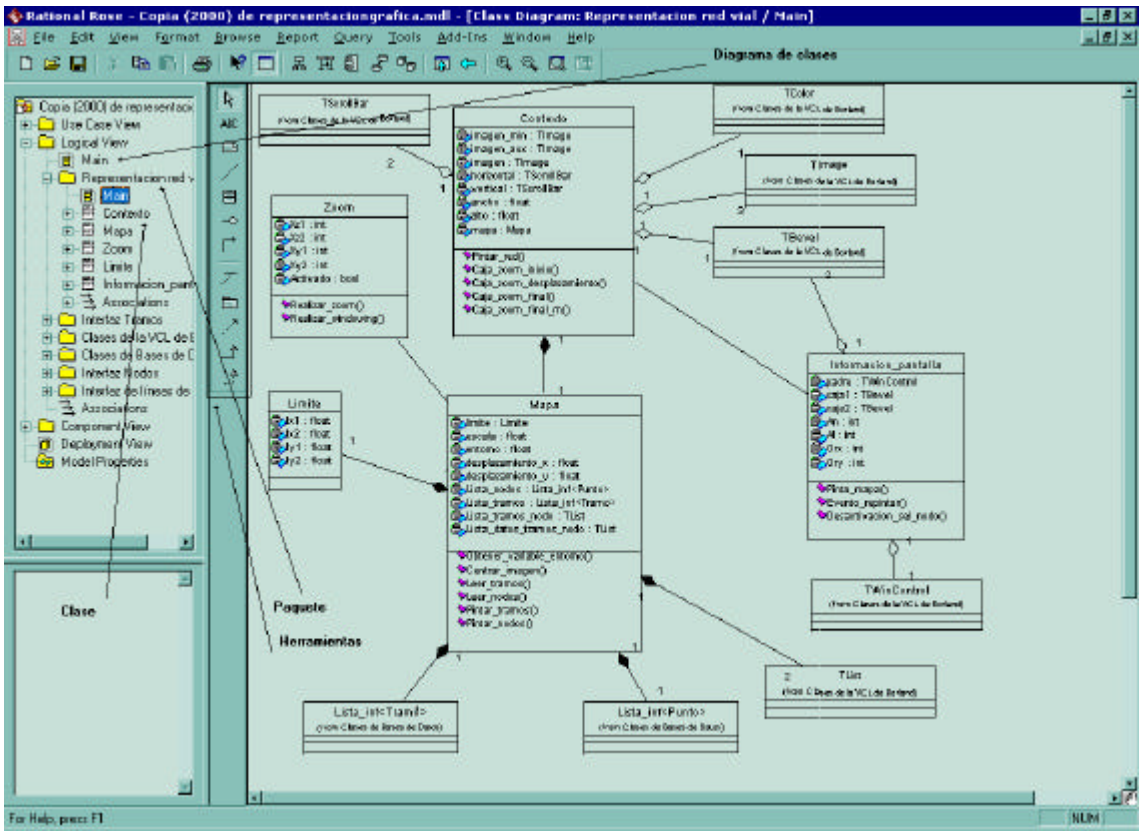

Figura 3.3.2: Diagrama de clases

Las otras dos pestañas más interesante son Atributes y Operations.

En Atributes puede especificarse cada atributo de la clase, y en Operations las operaciones. Una vez creados los atributos u operaciones, un doble click sobre ellos hace que se despliegue un menú con las características del atributo u operación elegido.

Tanto en Atributes como en Operations aparece una casilla Show inherited, que se usa para mostrar los atributos u operaciones heredados.

Para crear un nuevo atributo u operación basta con hacer pulsar el botón derecho del ratón sobre el espacio en blanco.

Al hacer doble click sobre un atributo se despliega un menú con sus características. La característica más importante es Export Control, que controla la visibilidad del atributo. También es importante, en la pestaña Detail, especificar el valor Containment. Puede ser de tres tipos: by value, by reference o unspecified. La opción by value indica que el objeto se contiene físicamente. La opción by reference indica que lo que se contiene es un puntero al objeto. Finalmente la opción unspecified es no especificado.

En cuanto a las operaciones, en la pestaña General se puede especificar el valor de clase que devuelve en el campo Return Class. También en esta pestaña se especifica la visibilidad en el campo Export Control.

En la pestaña Detail se pueden especificar los argumentos que recibe cada operación. También se puede definir la concurrencia de la operación. Lo normal es que sea Sequential, es decir, sólo puede ser ejecutada por un hilo de control.

En la pestaña Semantics se puede especificar las acciones que realiza la operación.

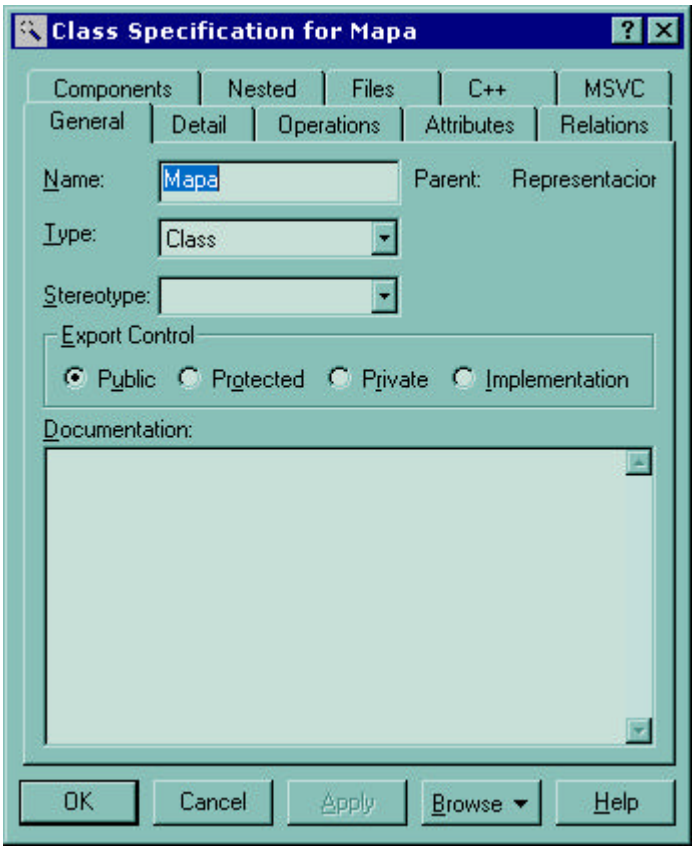

Figura 3.3.3: Solapa General de una clase

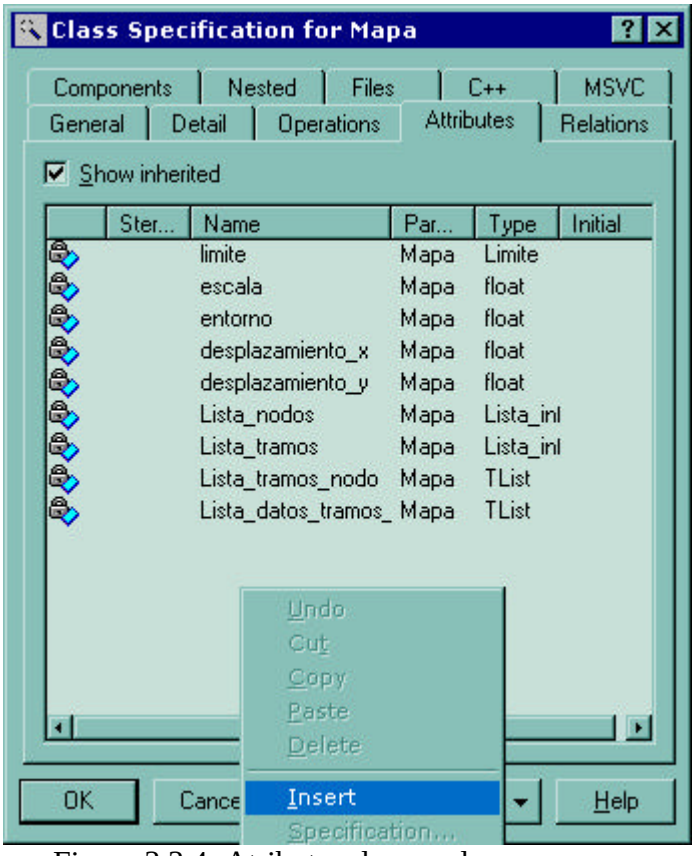

Figura 3.3.4: Atributos de una clase

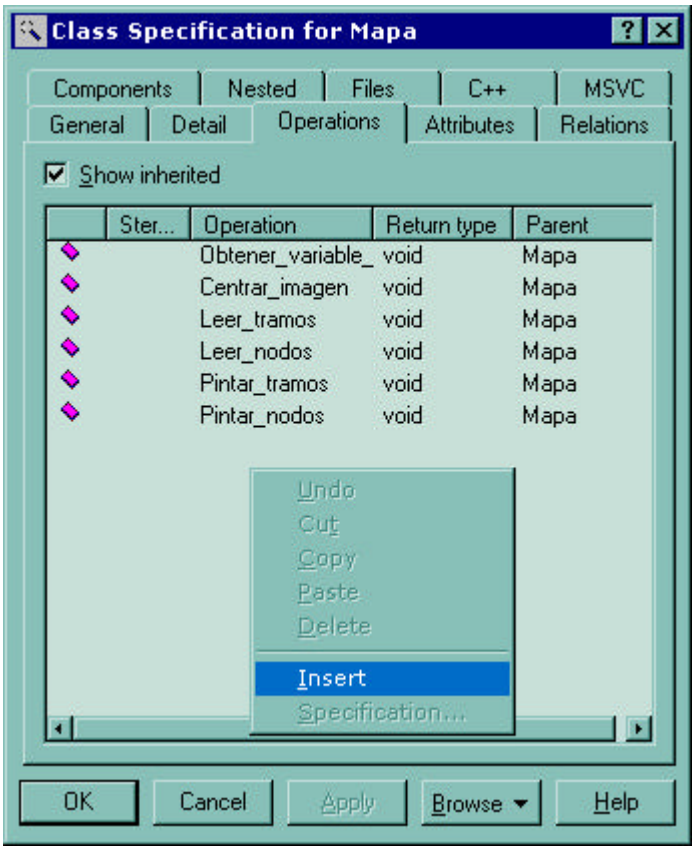

Figura 3.3.5: Operaciones de una clase

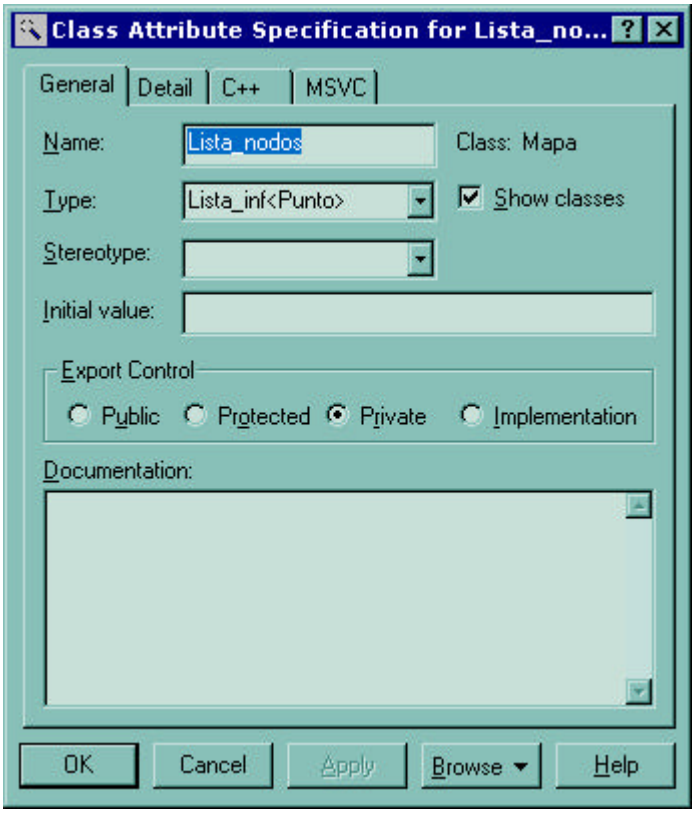

Figura 3.3.6: Solapa general de un atributo de una clase

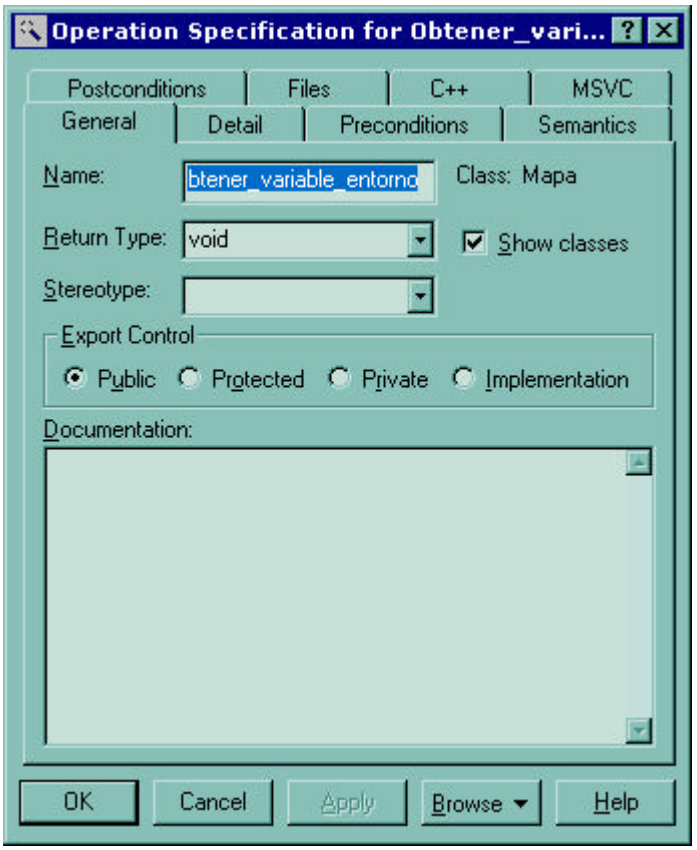

Figura 3.3.7: Solapa General de una operación de una clase

Asociaciones

Haciendo doble click sobre la asociación se despliega una ficha de características. En la pestaña General se puede especificar el nombre y estereotipo, así como el rol de cada uno de los extremos.

En las otras pestañas hay más parámetros de configuración, pero no suelen usarse demasiado.

Dependencias

Se pueden especificar los campos Name, Export Control y Cardinality from y Cardinality to. Estos dos últimos sirven para definir la cardinalidad en los distintos sentidos.

Agregación y composición

Lo interesante es que para ambas se debe crear una agregación. Según se seleccione la opción By value o By reference será composición o agregación.

Diagrama de casos de uso

Los diagramas de casos de uso incluyen casos de uso, actores y las asociaciones entre ellos. Opcionalmente, de nuevo, pueden contener paquetes.

Un actor no es más que una clase con el estereotipo Actor.

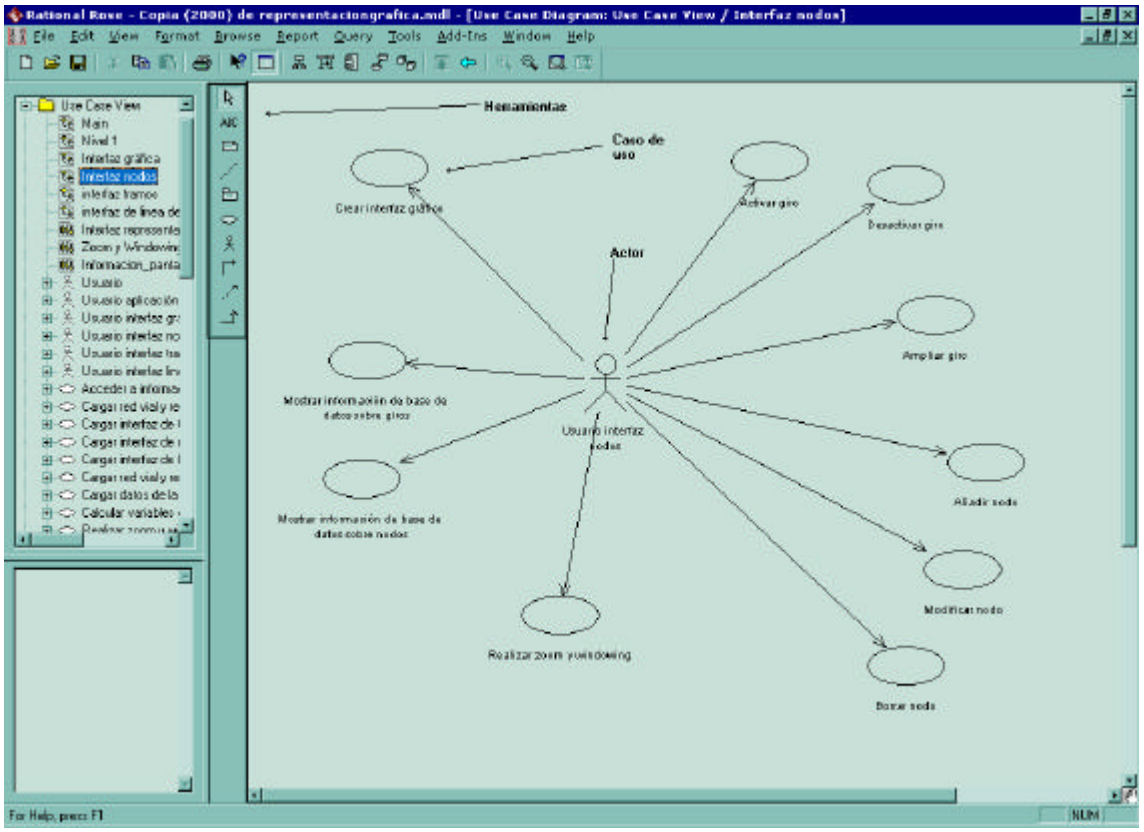

Figura 3.3.8: Diagrama de casos de uso

Casos de uso

En las características del caso de uso, al que se accede haciendo doble click sobre él en el diagrama o en el navegador, aparece en primer lugar la solapa General. En ella el campo principal es Name. Todos los casos de uso tienen un nombre, normalmente descriptivo de las acciones que engloba.

En la solapa Diagrams se muestran todos los diagramas que posee el caso de uso. Al pulsar sobre el espacio en blanco con el botón derecho del ratón se abre un menú desplegable con los posibles diagramas que se pueden añadir. Un uso lógico sería insertar el diagrama de secuencias que define su implementación.

En la solapa Relations aparecen las relaciones con otros casos de uso y actores.

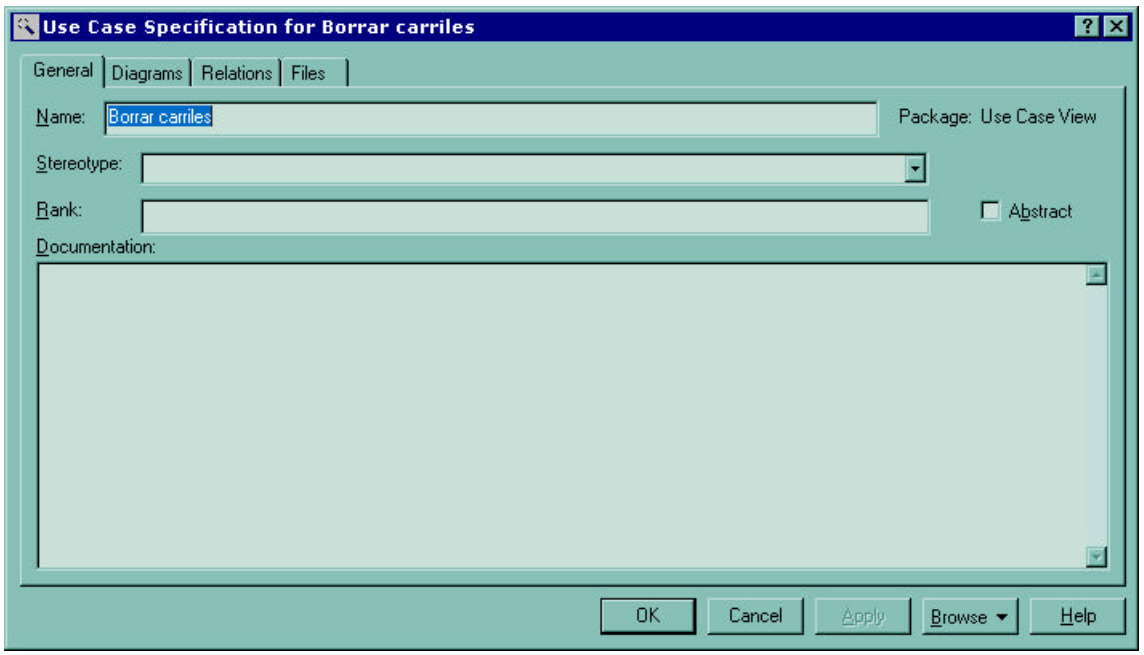

Figura 3.3.9: Solapa General de un caso de uso

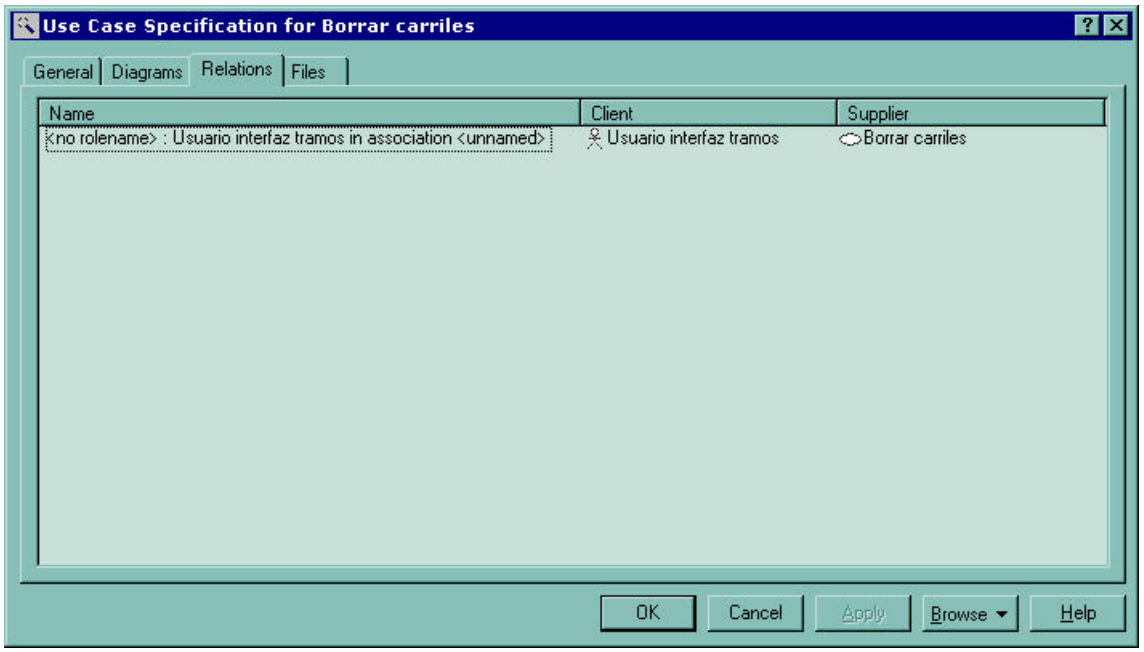

Figura 3.3.10: Solapa Relations de un caso de uso

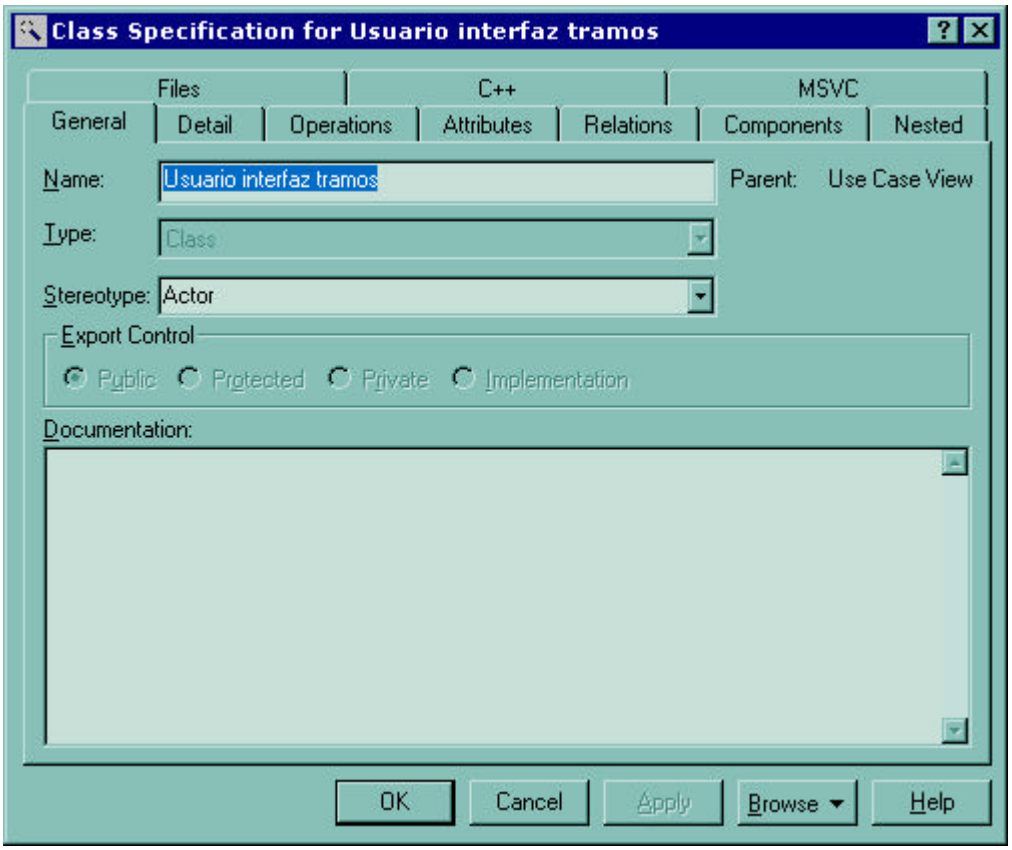

Figura 3.3.11: Solapa General para un actor

Diagrama de secuencias

Estos diagramas definen la implementación de los casos de uso. En estos diagramas aparecen objetos (instancias de clases) y los mensajes que se pasan entre ellos. Los mensajes más comunes entre objetos son los que tienen como fin crear o destruir otro objeto, que tendrían el estereotipo create o destroy, y los que solicitan la ejecución de una operación por parte de otro objeto, así como los que devuelven el resultado de la realización de una operación.

Los objetos son instancias de las clases y los mensajes con instancias de las asociaciones.

Cuando se accede a las características de un objeto sólo aparece una solapa, General. En ella se puede especificar el nombre del objeto y la clase a la que pertenece. Además se puede especificar la persistencia. La persistencia puede ser:

- Transient: el objeto existe después de la terminación del programa
- Static: El objeto existe durante la ejecución del programa
- Transient: El objeto se crea y destruye dinámicamente durante la ejecución del programa.

El foco de control puede habilitarse o deshabilitarse en Tools->Options->Diagram->Display->Focus of control.

Cuando se accede a las características del mensaje se tienen varias solapas. Una de ellas es General, donde sólo se puede especificar el nombre. En la solapa Detail se puede

especificar la sincronización (Synchronization). Este apartado sirve para controlar la concurrencia. El valor más normal es Simple, que indica que se tiene un único hilo de control.

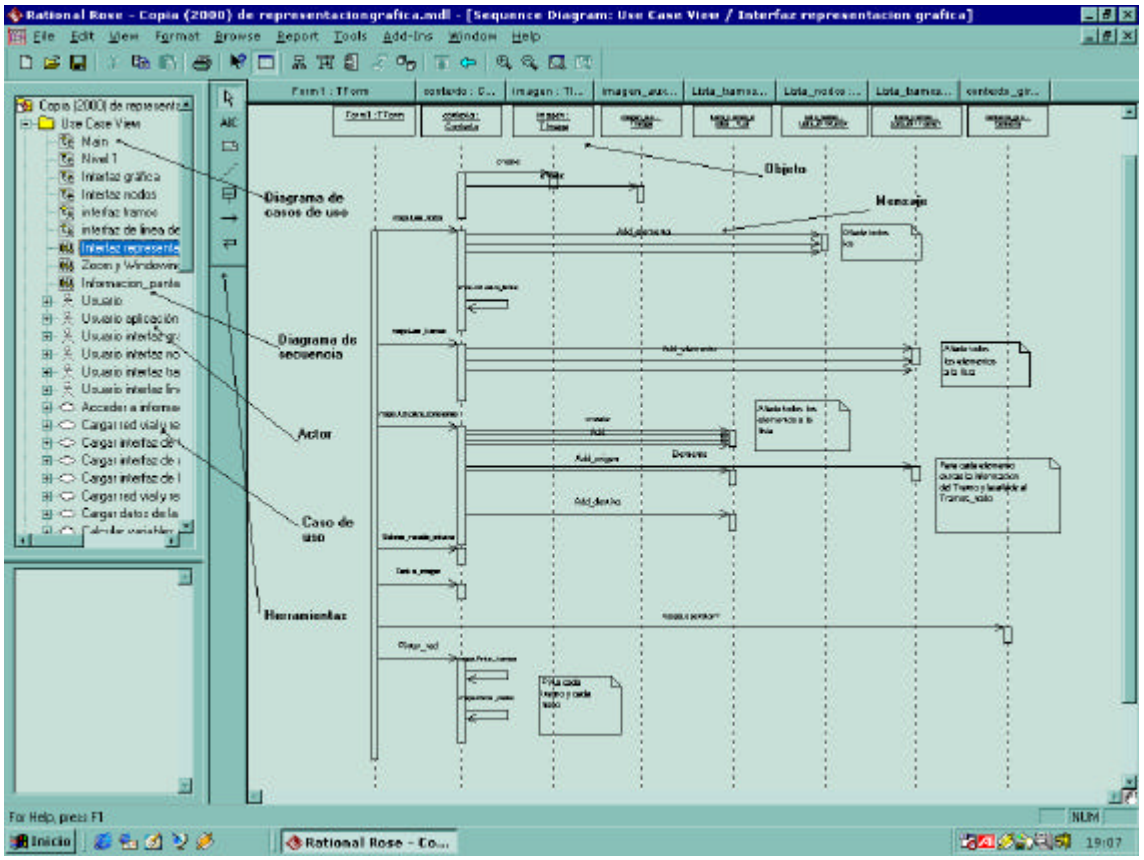

Figura 3.3.12: Diagrama de secuencia

La frecuencia especifica si el mensaje es enviado a intervalos irregulares (Aperiodic) o a intervalos regulares (Periodic).

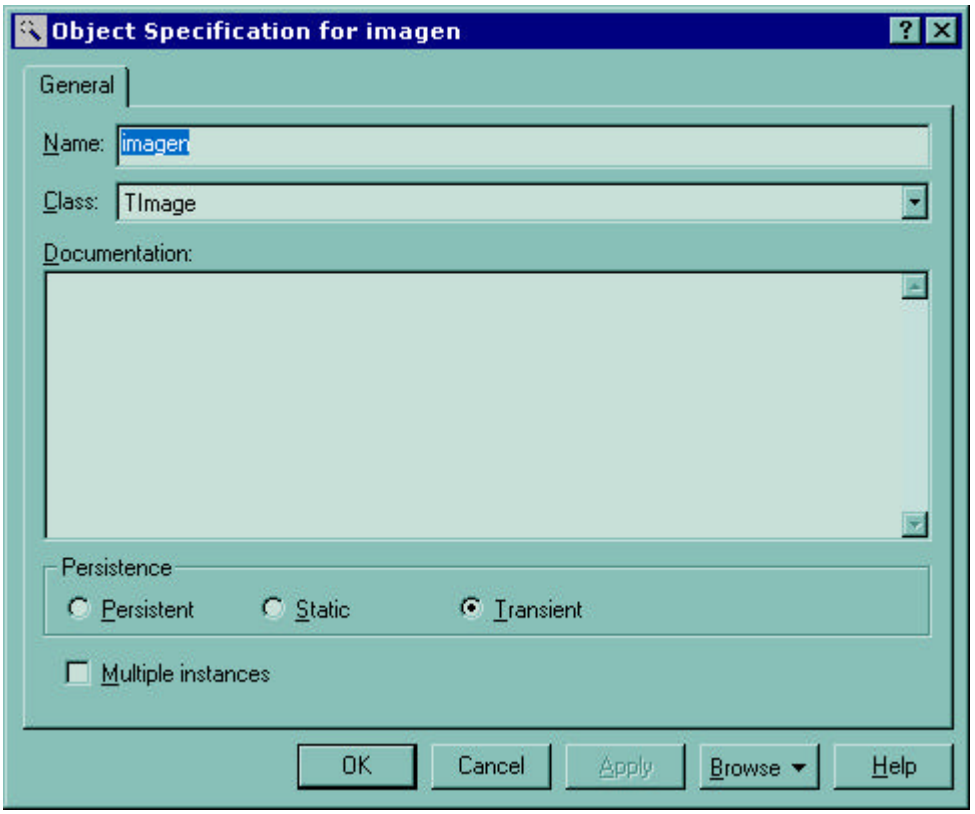

Figura 3.3.13: Solapa General para un objeto

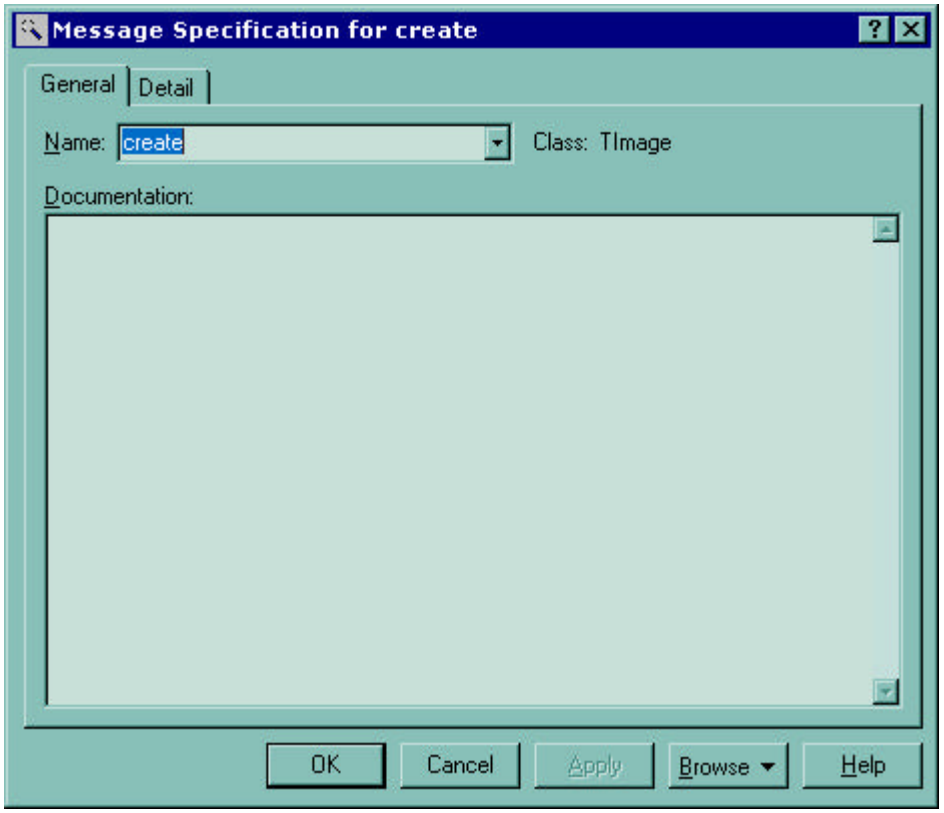

Figura 3.3.14: Solapa General para un mensaje

Generación de código a partir de modelos

El software Rational Rose permite la generación de código C++ con los datos de un modelo.

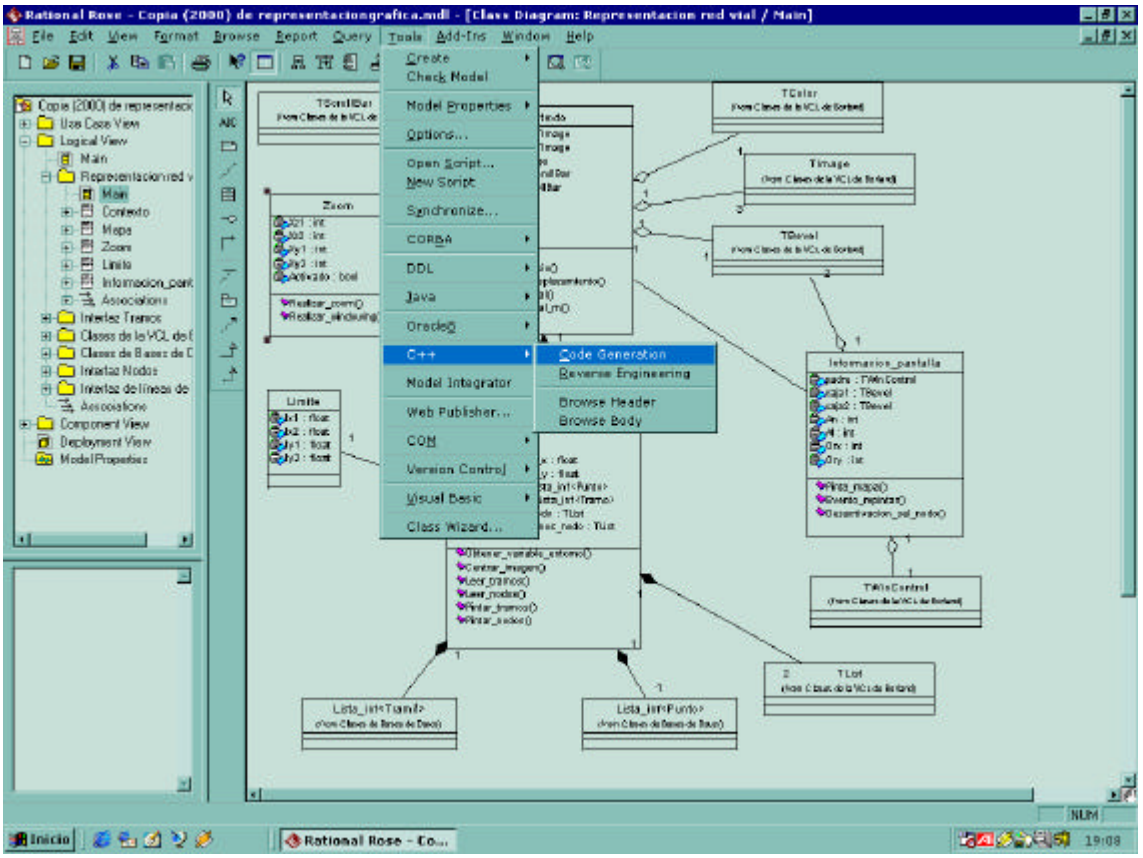

Figura 3.3.15: Generación de código

Para generar código, en primer lugar hay que seleccionar la clase o clases para las que se quiere generar el código. Seleccionar un paquete equivale a seleccionar todas las clases incluidas en ese paquete.

A continuación hay que seleccionar la herramienta de generación de código como se muestra en la figura 3.3.15.

Rational Rose genera ficheros de cabecera (.h) donde plasma las declaraciones de las clase. Esto incluye atributos y operaciones, incluso las operaciones típicas que suelen usarse para devolver o actualizar el valor de cada uno de los atributos.

Rose también genera ficheros de código (.cpp), pero en ellos no plasma código fuente, sino que incluye el esqueleto de las funciones miembro. No es, por tanto, muy interesante.

A continuación se adjunta el fichero generado por Rational Rose para una de las clases del modelo, la clase Zoom:

```
//## begin module%1.2%.codegen_version preserve=yes
//    Read the documentation to learn more about C++ code
generator
//   versioning.
```

```
//## end module%1.2%.codegen_version

//## begin module%3AFFF02C02A8.cm preserve=no
//     %X% %Q% %Z% %W%
//## end module%3AFFF02C02A8.cm

//## begin module%3AFFF02C02A8.cp preserve=no
//## end module%3AFFF02C02A8.cp

//##    Module:    Zoom%3AFFF02C02A8;    Pseudo    Package
specification
//##  Source  file:  C:\Archivos  de  programa\Rational\Rose
2000\Rprsntcn\Zoom.h

#ifndef Zoom_h
#define Zoom_h 1

//##      begin      module%3AFFF02C02A8.additionalIncludes
preserve=no
//## end module%3AFFF02C02A8.additionalIncludes

//## begin module%3AFFF02C02A8.includes preserve=yes
//## end module%3AFFF02C02A8.includes

// Mapa
#include "Rprsntcn\Mapa.h"
//##    begin    module%3AFFF02C02A8.additionalDeclarations
preserve=yes
//## end module%3AFFF02C02A8.additionalDeclarations


//## begin Zoom%3AFFF02C02A8.preface preserve=yes
//## end Zoom%3AFFF02C02A8.preface

//## Class: Zoom%3AFFF02C02A8
//## Category: Representacion red vial%3AFFEE6C010E
//## Persistence: Transient
//## Cardinality/Multiplicity: n

class Zoom
{
  //##     begin     Zoom%3AFFF02C02A8.initialDeclarations
preserve=yes
  //## end Zoom%3AFFF02C02A8.initialDeclarations

  public:
    //## Constructors (generated)
      Zoom();

      Zoom(const Zoom &right);
```

```
    //## Destructor (generated)
      ~Zoom();

    //## Assignment Operation (generated)
      Zoom & operator=(const Zoom &right);

    //## Equality Operations (generated)
      int operator==(const Zoom &right) const;

      int operator!=(const Zoom &right) const;


    //## Other Operations (specified)
      //## Operation: Realizar_zoom%3B0D53910140
      void  Realizar_zoom  (Contexto  origen  =  default,
Contexto destino = default);

      //## Operation: Realizar_windowing%3B0D542200DC
       void Realizar_windowing (Contexto origen = default,
Contexto destino = default);

    //##  Get  and  Set  Operations  for  Associations
(generated)

      //##      Association:      Representacion      red
vial::<unnamed>%3B0D51360122
      //## Role: Zoom::<the_Mapa>%3B0D51360384
      const Mapa * get_the_Mapa () const;
      void set_the_Mapa (Mapa * value);

    // Additional Public Declarations
      //## begin Zoom%3AFFF02C02A8.public preserve=yes
      //## end Zoom%3AFFF02C02A8.public

  protected:
    // Additional Protected Declarations
      //## begin Zoom%3AFFF02C02A8.protected preserve=yes
      //## end Zoom%3AFFF02C02A8.protected

  private:
    //##  Get  and  Set  Operations  for  Class  Attributes
(generated)

      //## Attribute: Xz1%3AFFF3C100AA
      const int get_Xz1 () const;
      void set_Xz1 (int value);

      //## Attribute: Xz2%3AFFF3D10096
      const int get_Xz2 () const;
      void set_Xz2 (int value);
```

```
      //## Attribute: Xy1%3AFFF3D200BE
      const int get_Xy1 () const;
      void set_Xy1 (int value);

      //## Attribute: Xy2%3AFFF3D3015E
      const int get_Xy2 () const;
      void set_Xy2 (int value);

      //## Attribute: Activado%3AFFF3EA0244
      const bool get_Activado () const;
      void set_Activado (bool value);

    // Additional Private Declarations
      //## begin Zoom%3AFFF02C02A8.private preserve=yes
      //## end Zoom%3AFFF02C02A8.private

  private: //## implementation
    // Data Members for Class Attributes

      //##  begin  Zoom::Xz1%3AFFF3C100AA.attr  preserve=no
private: int {U}
      int Xz1;
      //## end Zoom::Xz1%3AFFF3C100AA.attr

      //##  begin  Zoom::Xz2%3AFFF3D10096.attr  preserve=no
private: int {U}
      int Xz2;
      //## end Zoom::Xz2%3AFFF3D10096.attr

      //##  begin  Zoom::Xy1%3AFFF3D200BE.attr  preserve=no
private: int {U}
      int Xy1;
      //## end Zoom::Xy1%3AFFF3D200BE.attr

      //##  begin  Zoom::Xy2%3AFFF3D3015E.attr  preserve=no
private: int {U}
      int Xy2;
      //## end Zoom::Xy2%3AFFF3D3015E.attr

      //##      begin      Zoom::Activado%3AFFF3EA0244.attr
preserve=no  private: bool {U}
      bool Activado;
      //## end Zoom::Activado%3AFFF3EA0244.attr

    // Data Members for Associations

      //##      Association:      Representacion      red
vial::<unnamed>%3B0D51360122
      //##     begin     Zoom::<the_Mapa>%3B0D51360384.role
preserve=no  public: Mapa { -> RHN}
      Mapa *the_Mapa;
```

```
      //## end Zoom::<the_Mapa>%3B0D51360384.role

    // Additional Implementation Declarations
      //##      begin      Zoom%3AFFF02C02A8.implementation
preserve=yes
      //## end Zoom%3AFFF02C02A8.implementation

};

//## begin Zoom%3AFFF02C02A8.postscript preserve=yes
//## end Zoom%3AFFF02C02A8.postscript

// Class Zoom

//## Get and Set Operations for Class Attributes (inline)

inline const int Zoom::get_Xz1 () const
{
  //## begin Zoom::get_Xz1%3AFFF3C100AA.get preserve=no
  return Xz1;
  //## end Zoom::get_Xz1%3AFFF3C100AA.get
}

inline void Zoom::set_Xz1 (int value)
{
  //## begin Zoom::set_Xz1%3AFFF3C100AA.set preserve=no
  Xz1 = value;
  //## end Zoom::set_Xz1%3AFFF3C100AA.set
}

inline const int Zoom::get_Xz2 () const
{
  //## begin Zoom::get_Xz2%3AFFF3D10096.get preserve=no
  return Xz2;
  //## end Zoom::get_Xz2%3AFFF3D10096.get
}

inline void Zoom::set_Xz2 (int value)
{
  //## begin Zoom::set_Xz2%3AFFF3D10096.set preserve=no
  Xz2 = value;
  //## end Zoom::set_Xz2%3AFFF3D10096.set
}

inline const int Zoom::get_Xy1 () const
{
  //## begin Zoom::get_Xy1%3AFFF3D200BE.get preserve=no
  return Xy1;
  //## end Zoom::get_Xy1%3AFFF3D200BE.get
}
```

```
inline void Zoom::set_Xy1 (int value)
{
  //## begin Zoom::set_Xy1%3AFFF3D200BE.set preserve=no
  Xy1 = value;
  //## end Zoom::set_Xy1%3AFFF3D200BE.set
}

inline const int Zoom::get_Xy2 () const
{
  //## begin Zoom::get_Xy2%3AFFF3D3015E.get preserve=no
  return Xy2;
  //## end Zoom::get_Xy2%3AFFF3D3015E.get
}

inline void Zoom::set_Xy2 (int value)
{
  //## begin Zoom::set_Xy2%3AFFF3D3015E.set preserve=no
  Xy2 = value;
  //## end Zoom::set_Xy2%3AFFF3D3015E.set
}

inline const bool Zoom::get_Activado () const
{
  //##      begin      Zoom::get_Activado%3AFFF3EA0244.get
preserve=no
  return Activado;
  //## end Zoom::get_Activado%3AFFF3EA0244.get
}

inline void Zoom::set_Activado (bool value)
{
  //##      begin      Zoom::set_Activado%3AFFF3EA0244.set
preserve=no
  Activado = value;
  //## end Zoom::set_Activado%3AFFF3EA0244.set
}

//## Get and Set Operations for Associations (inline)

inline const Mapa * Zoom::get_the_Mapa () const
{
  //##      begin      Zoom::get_the_Mapa%3B0D51360384.get
preserve=no
  return the_Mapa;
  //## end Zoom::get_the_Mapa%3B0D51360384.get
}

inline void Zoom::set_the_Mapa (Mapa * value)
{
  //##      begin      Zoom::set_the_Mapa%3B0D51360384.set
preserve=no
```

```
  the_Mapa = value;
  //## end Zoom::set_the_Mapa%3B0D51360384.set
}

//## begin module%3AFFF02C02A8.epilog preserve=yes
//## end module%3AFFF02C02A8.epilog


#endif
```

Puede observarse como crea el constructor y el destructor y las declaraciones necesarias para todas las funciones. Pero además crea funciones inline, no sólo las declaraciones, para devolver el valor de cada uno de los atributos o actualizarlos, como por ejemplo, *get_Activado* y *Set_Activado*.

Generación de modelos a partir de código fuente

Rational Rose también permite la operación inversa, es decir, generación de los modelos a partir del código fuente. Para ello, debe crearse un proyecto donde hay que incluir todos los archivos del código, tanto .h como .cpp.

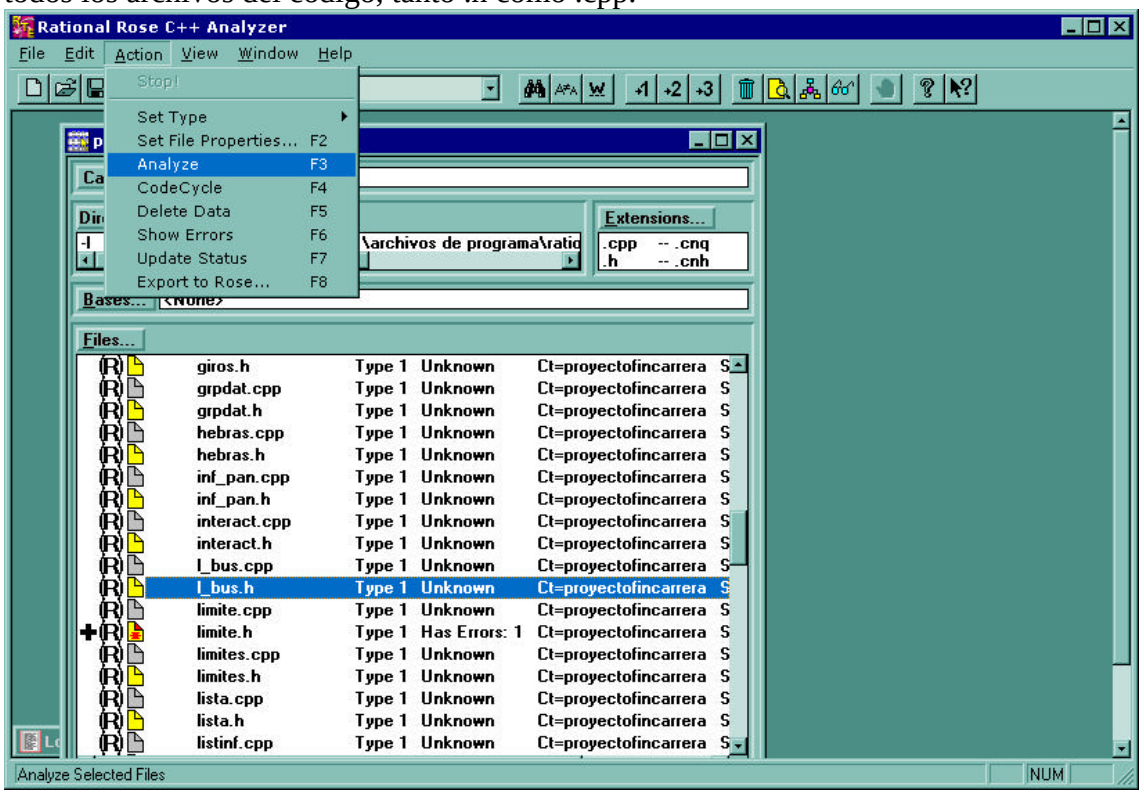

Figura 3.3.16: Analizador de código

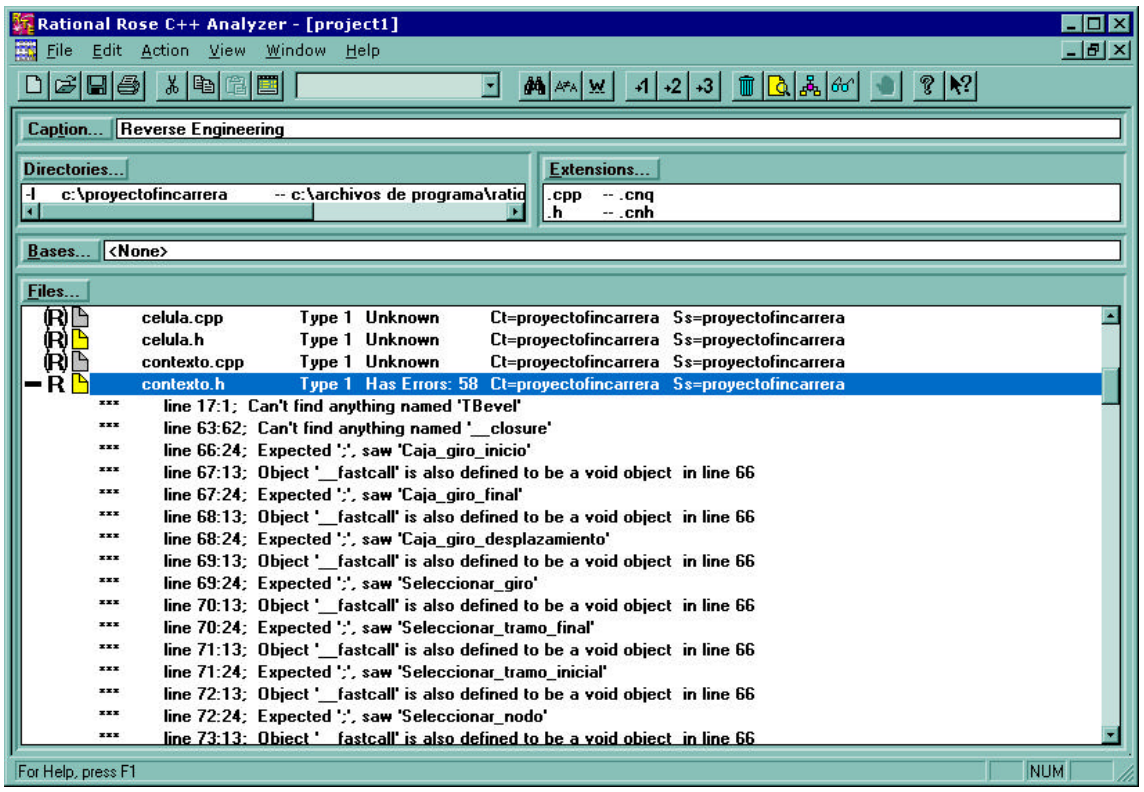

Figura 3.3.17: Problemas del generador de código con el generado con Borland C++ Builder

Una vez creado dicho proyecto, basta con seleccionar el fichero (.h o .cpp) que se desea trasladar al modelo para que Rose lo genere.

Un problema que puede aparecer si se usan entornos de desarrollo como Microsoft Visual C++ o Borland C++Builder, o cualquier biblioteca de clases, que Rose no reconocerá muchas clases, ya que no se declaran en el proyecto.

En el caso de la aplicación, por ejemplo, la clase *TImage* no es reconocida por el analizador de código de Rose, ya que pertenece a la VCL de Borland y no se ha declarado en ningún proyecto en el código de la aplicación.

Este problema lo resuelve Rose para el caso de los entornos de Microsoft, ya que incluye ficheros de proyecto donde se definen estas clases y pueden incluirse en el proyecto desarrollado por el usuario, con lo que se eliminan los problemas de no definición de dichas clases. Sin embargo, no incluye los proyectos necesarios para el código desarrollado con los entornos de Borland.

Para mostrar un ejemplo de generación de modelo, se puede usar un fichero que contenga clases cuyos atributos no pertenezcan a la VCL. Se elige para este fin el fichero limite.h, donde se define la clase *Limite.*

Aún así, se produjeron fallos en el proceso de generación del modelo, al no reconocer tampoco los tipos *bool* o *boolean*. Al eliminar los atributos de este tipo de la definición de la clase *Limite* se consigue por fin que genere el modelo.

Cuando se abre el fichero .mdl generado por Rose se encuentra lo que se muestra en las figuras 3.3.18 y 3.3.19.

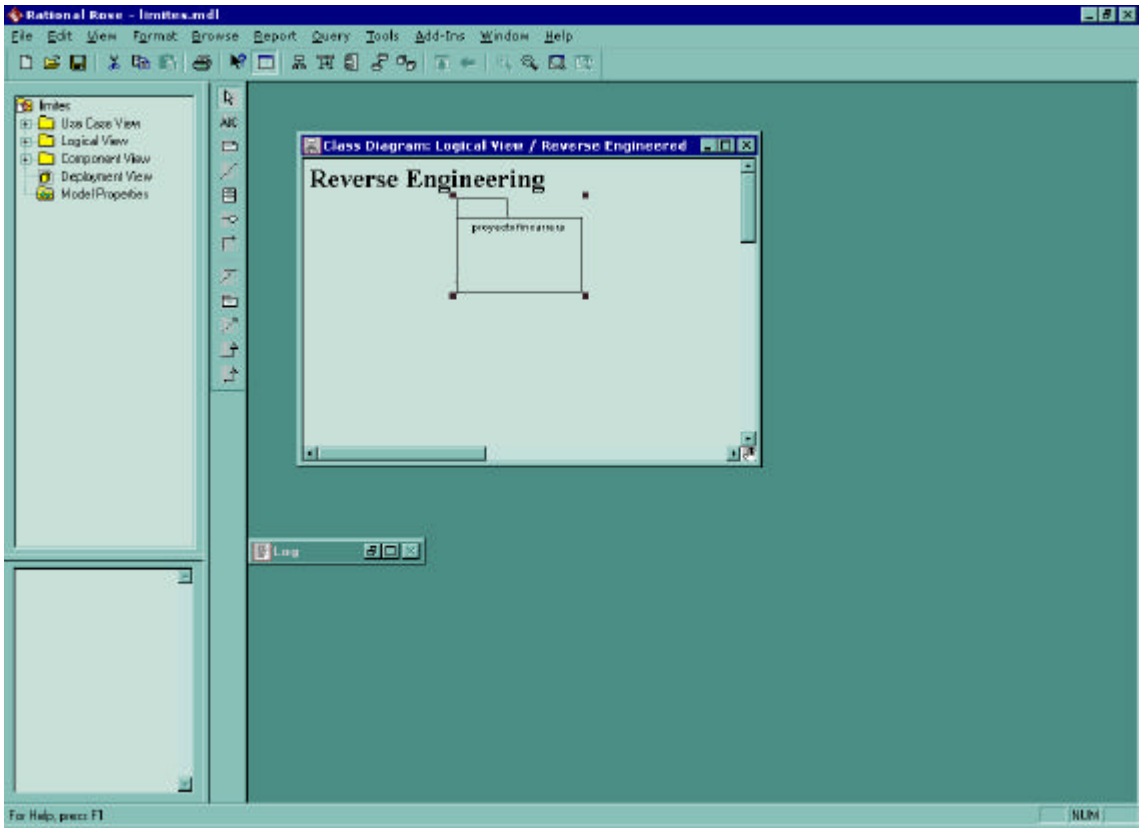

Figura 3.3.18: Paquete del modelo generado por Rose

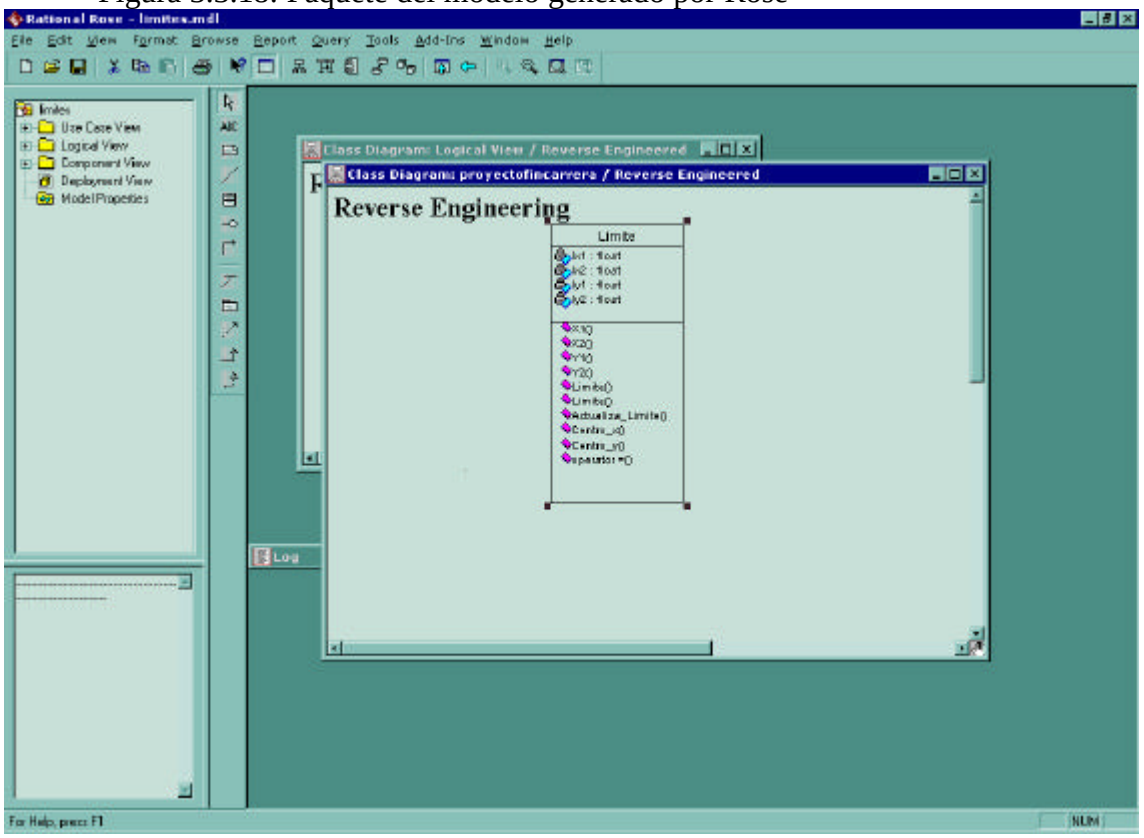

Figura 3.3.19: Clase Limite del modelo generado por Rose