ТЕМА 11

План лекции:

1. Создание кода класса в Rational Rose на Microsoft Visual C++.
2. Создание шаблона приложения в Rational Rose на Microsoft Visual C++.
3. Генерация исходного кода в среде Rational XDE.

11. 1 Автоматическая генерация кода класса

Прежде, чем создавать приложение, рассмотрим процесс построения кода класса на заданном языке программирования. Наиболее удобный способ – это получить код класса на основе библиотеки классов *Microsoft Foundation Class* (MFC). Для этого не нужно вручную оперировать большим количеством установок, так как в пакет встроен модуль *Model Assistant*, который позволяет изменять все необходимые установки при помощи визуальных.

Для того чтобы генератор *Rational Rose* мог создавать на основе описанной модели программный код, для каждого класса необходимо указать язык и определить компонент, в котором класс будет храниться. В случае языка VC++ открывается доступ ко всей иерархии классов библиотеки *MFC*. Ассоциация класса с VC++ происходит в результате выполнения следующих действий: *Menu=> Tools=>Visual C++=>Component Assigned Tools*, после чего открывается окно, показанное на рис. 11.1. В его правой части выбирается класс и перетаскивается на значок *VC++,* а затем подтверждаются создание *VC++* компонента и его связь с классом. Следующим открывается окно выбора проекта, где можно создать проект или выбрать уже существующий для помещения в него новый класс.

Model Assistant представляет собой окно (рис. 11.2), в котором создаются атрибуты, операции, а также задаются их свойства. В окне имеется ряд полей:

- *Preview* показывает описание класса в текущий момент..
- *Generate Code* – ключ, определяющий необходимость создания для класса исходного текста на VC++; если ключ снят, то генерация кода не происходит и класс не показывается в списке классов для обновления кода.
- *Class Type* – установка типа класса: “class”, “struct”, “union”.
- *Documentation* – произвольные комментарии для класса.

Rational Rose предоставляет возможности по интерактивной установке свойств методов класса. Если активизировать строку *calibrate* в окне на рис. 11.2, то откроется диалоговое окно (рис. 11.3), предназначенное для работы со свойствами атрибутов.

Для того чтобы получить преимущества использования инструмента *Component Assigment Tool*, необходимо создавать компоненты в окне (рис. 11.2), а не через окно Browser или в диаграмме компонентов. При этом созданные компоненты будут содержать всю необходимую информацию для

генерации кода на выбранном языке программирования. Кроме того, инструмент позволяет просмотреть классы, которые еще не назначены в компоненты.

Одна из самых важных возможностей *Rational Rose* – это *Update Code/Update Model*. Данное средство позволяет создать проект Visual C++ по разработанной модели и обновить модель по уже готовому проекту, созданному при помощи *MFC*. Предположим, уже создан проект, и известно, что после того как последний раз изменялась модель, исходный код определенного класса был исправлен, и его необходимо обновить. Для этого выбирается пункт *Update Model*, после чего появляется окно, в котором можно обновить либо все

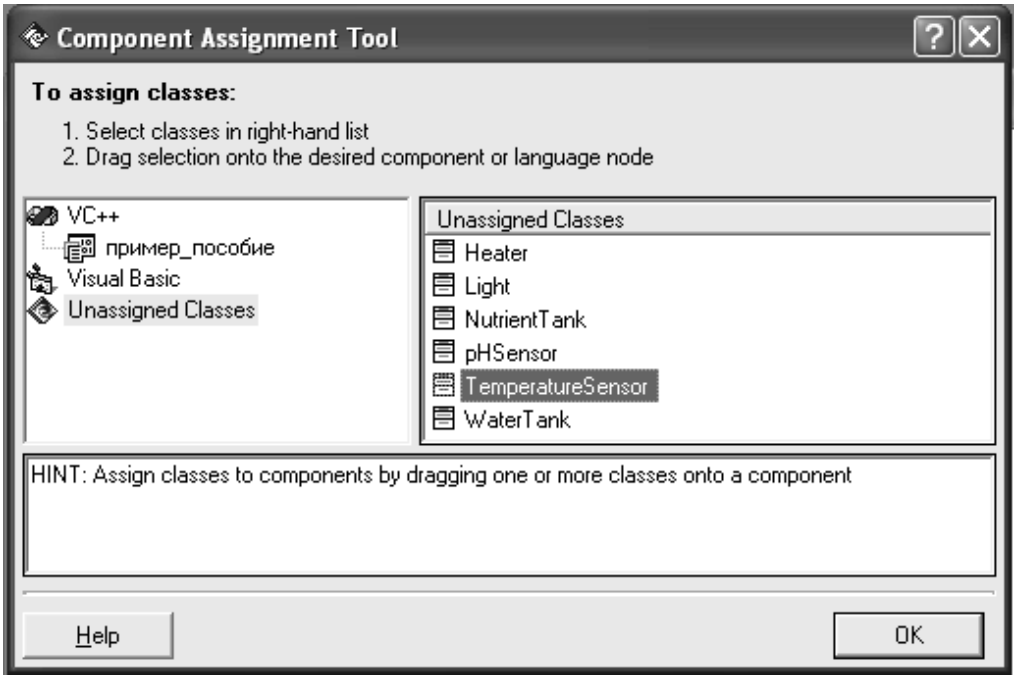

Рисунок 11.1 – Окно средства Component Assigned Tools

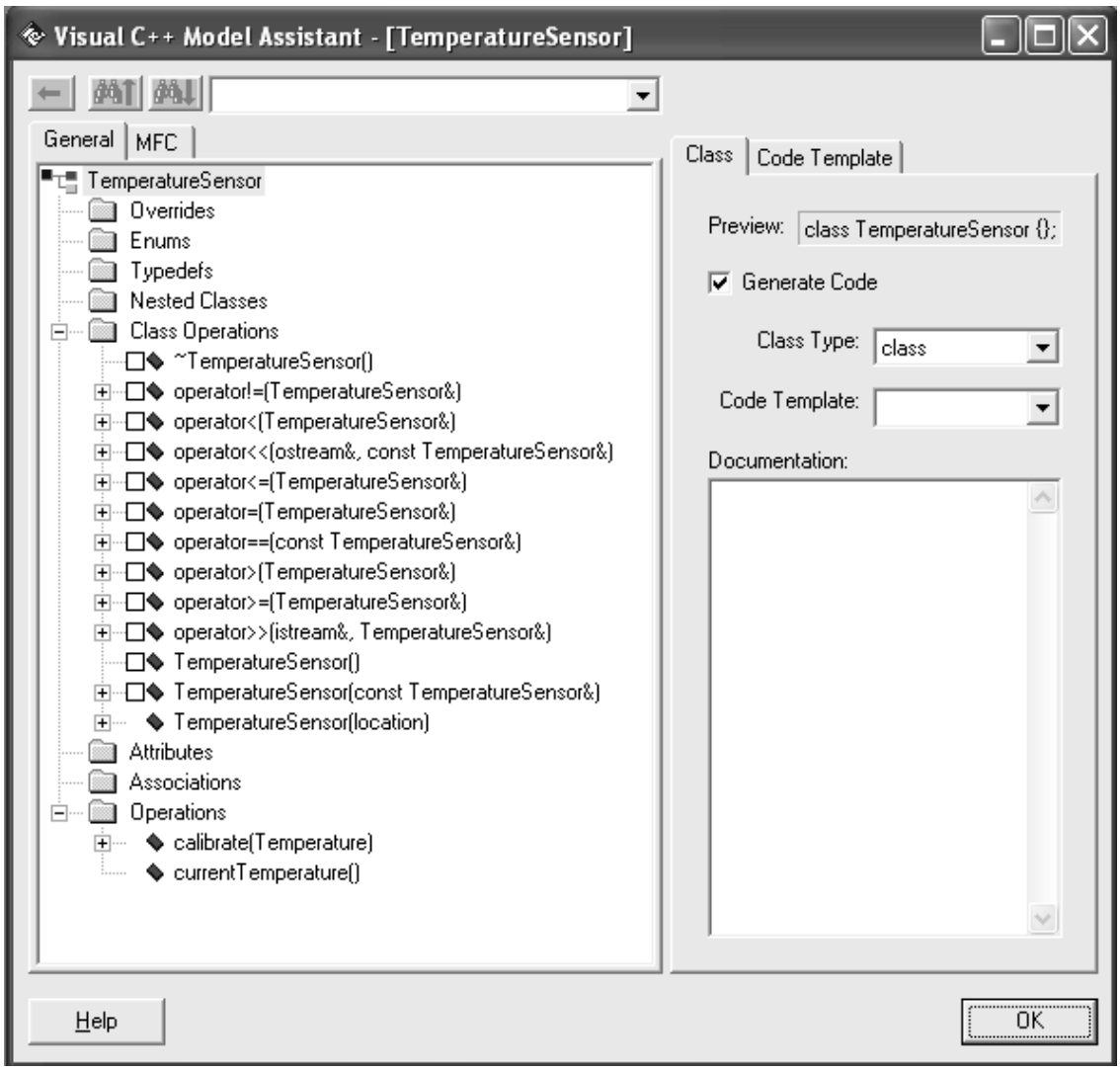

Рисунок 11.2 – Окно Model Assistant для класса TemperatureSensor

классы путем установки на них отметок, либо только выбранные, сняв с остальных отметки. Если классы модели еще не ассоциированы ни с одним про ектом *VC++*, то это можно сделать из текущего окна. Затем *Rational Rose* получает информацию из проекта *Visual C++*, для чего загружается *Microsoft Visual Studio*, и активизируется нужный проект. Если в код были внесены изменения или удалены компоненты, программа предложит выполнить такие же изменения и в модели.

В случае, когда в проект Visual C++ были добавлены классы, и они еще не отражены в модели Rational Rose, необходимо провести обновление кода при помощи функции Update Code. Процесс обновления кода по изменениям в модели происходит аналогично обновлению модели. По завершении всех действий программой будет представлен отчет о том, как прошло обновление. Если все прошло нормально, то ошибок и предупреждений быть не должно.

11.2 Построение шаблона приложения

Будем рассматривать процесс создания приложения как логическое продолжение построения кода класса. Поэтому в нашем распоряжении находятся

язык VC++ и библиотека *MFC*. Мастер создания приложений VC++ (*AppWizard*) может строить несколько типов приложений:

- *Single document* – приложение работает с одним документом.
- *Multiple document* – приложение работает с несколькими документами.
- *Dialog based* – приложение основано на окне диалога.

Для приложения, работающего с одним документом, мастер строит код следующих классов:

- главный класс приложения *C***App*;
- класс документа *C***Doc*;
- класс просмотра *C***View*;
- класс для окна «О программе» *CAboutDlg*;
- класс основного окна программы *CMainFrame*.

Все приложения *VC++ MFC* являются объектами. Поэтому приложение – это главный класс, который включает в себя все необходимые для работы классы. Соблюдая соглашение об именах, мастер создает главный класс приложения с именем проекта, прибавляя к нему в начале букву С, а в конце *App* (в нашем случае это – *C***App*, где *** – имя проекта). С****App* наследуется из библиотечного класса CWinApp. Класс документа *C***Doc*, в котором должна проходить обработка данных, наследуется из библиотечного класса *CDocument*. Класс просмотра *C***View*, отображающий данные на экране компьютера, наследуется из библиотечного класса *CEditView*. Данные отображаются в окне класса *CMainFrame*, наследуемого из библиотечного класса *CFrameWnd*.

Таким образом, на основе стандартных классов документа, предоставляемых MFC, строится приложение, в котором необходимо будет только добавить функциональность, а за отображение документа на экране отвечает библиотека.

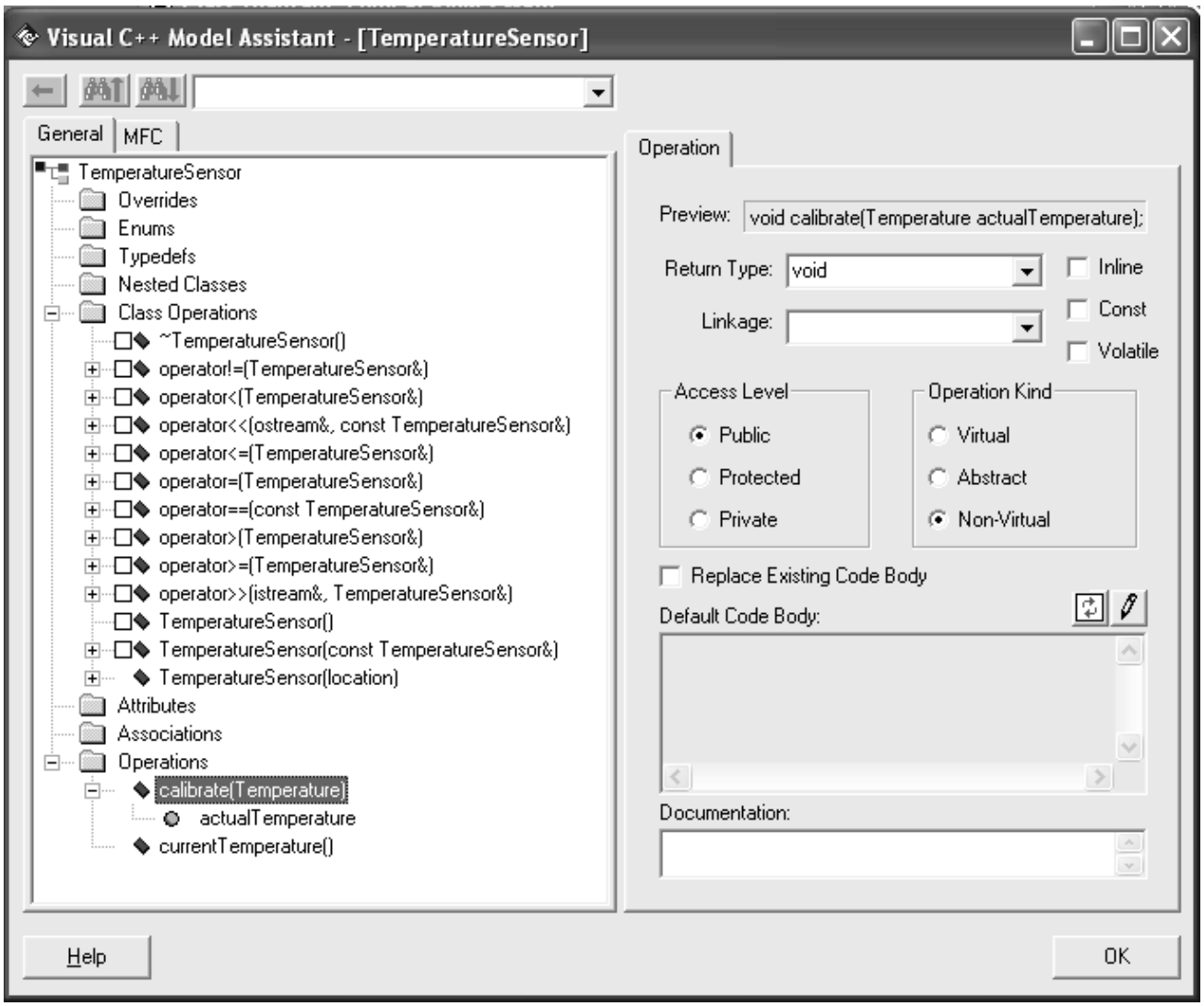

Рисунок 11.3 – Окно атрибутов операции calibrate

Ассоциация проекта *Rational Rose* с проектом *VC++* выполняется аналогично тому, как это делалось для одного класса. С помощью *Component Assigment Tool* в строку *VC++* перетаскиваются все необходимые классы. Для простоты работы все классы, для которых необходимо создание исходного кода, помещаются в один компонент, заключенный в проект ***. Библиотека классов *MFC* импортируется в модель путем следующих действий: *Menu:Tools=>Visual C++=>Quick Import MFC 6.0.* Затем в модель *Rational Rose* можно загрузить перечисленные ранее классы, созданные в VC++. Для того чтобы они появились в проекте, необходимо обновить проект по готовому коду, выполнив действия: *Menu:Tools => Visual C++=> Update Model From Code*. После обновления в модели Rational Rose и в проекте VC++ содержатся одинаковые наборы классов. Теперь можно запустить *Visual Studio*, перейти в нужный проект, откомпилировать его и получить результат работы программы.

1. 3 Генерация исходного кода в среде Rational XDE

После того как рассмотрены все диаграммы, предназначенные для проектирования системы, можно переходить к генерации исходного кода на основе диаграммы классов. Генерация по готовым диаграммам позволяет исключить рутинный труд по ручному кодированию и созданию шаблонов, что сокращает количество ошибок на стадии преобразования готовой модели в программный код. Кроме того, по любой программе, написанной при помощи среды *Microsoft .NET*, можно провести обратное проектирование и разобраться в архитектуре, представленной графически в виде иерархии классов.

Rational XDE предоставляет большое количество настроек для автоматической и ручной генерации кода приложения. Все установки по-разному влияют на генерацию кода и призваны повышать продуктивность работы.

Для работы с исходным кодом в контекстном меню класса доступна следующая группа пунктов меню:

– *Generate Code* – генерация кода по созданной модели.

– *Synchronize* – синхронизация кода и модели. При этом все изменения, внесенные в код, отражаются в модели, а изменения модели переносятся в исходный код.

– *Browse Code* – позволяет переключаться в режим просмотра созданного кода.

Для работы с одним-двумя классами этого вполне достаточно, но при работе с большими моделями удобнее использовать синхронизацию в автоматическом режиме.

При создании архитектуры классов приложения используются различные типы связей. Одни не требуют настройки, другие обладают значительным количеством свойств, влияющих на исходный код.

При создании исходного кода ассоциативные связи создают одинаковый код независимо от того, направленная это ассоциация или нет, отражает ли связь агрегацию или композицию классов (рис. 11.4). При генерации кода будет создана переменная класса, заданная в параметрах связи.

При этом направление связи не играет роли, а важно задание имени переменной. При создании связи *Directed Association* Rational XDE автоматически создает переменную класса того типа, который находится в окончании стрелки связи. Эта переменная включается в класс, из которого стрелка исходит. Установки, созданные при построении связи, несложно изменить с помощью окна *Properties*.

Для настройки параметров автоматической синхронизации модели необходимо выбрать из главного меню пункт *Tools => Options => Rational XDE => Round Trip Engineering => Synchronization Setting* (рис 11.5).

После отметки в приведенном окне пункта *Automatic Synchronization* становятся доступными следующие варианты синхронизации:

– *When Saving Model Files* – синхронизация в момент сохранения модели;

– *When Model gets Focus* – синхронизация в момент активизации модели;

– *When saving Code Files* – синхронизация происходит в момент сохранения кода после его изменения;

– *When Code gets Focus* – синхронизация происходит в момент активизации окна кода.

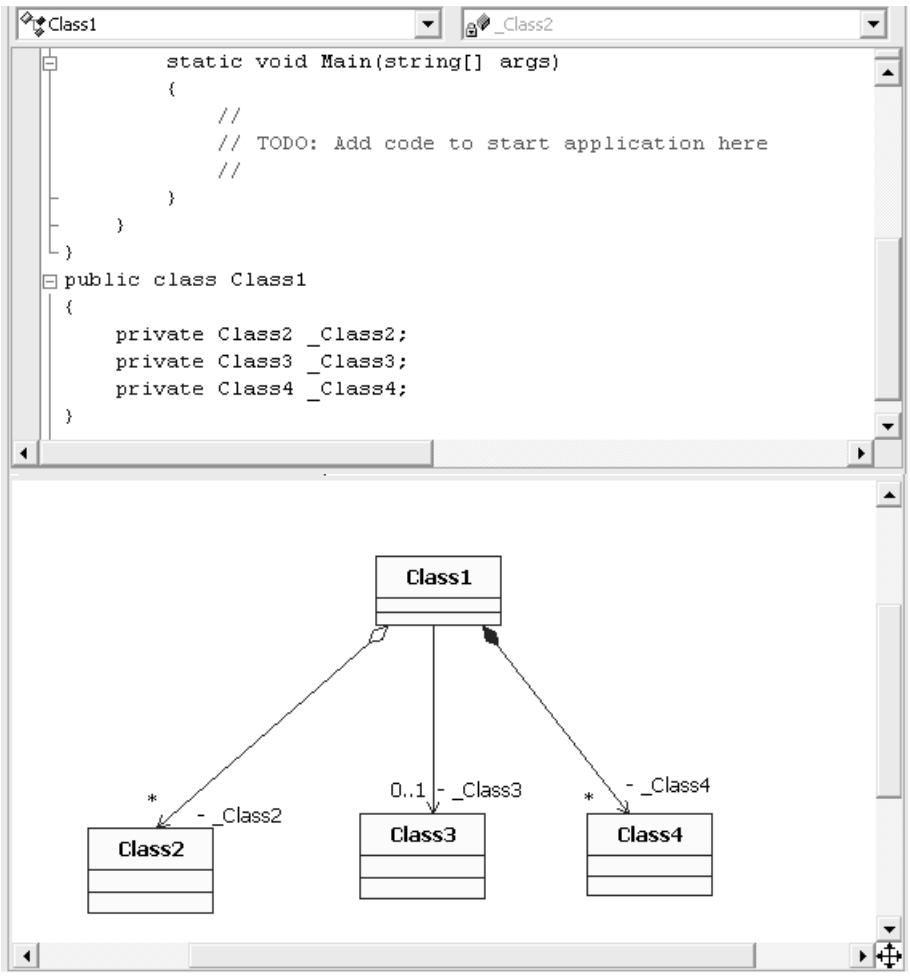

Рисунок 11.4 – Исходный код для различных типов связей

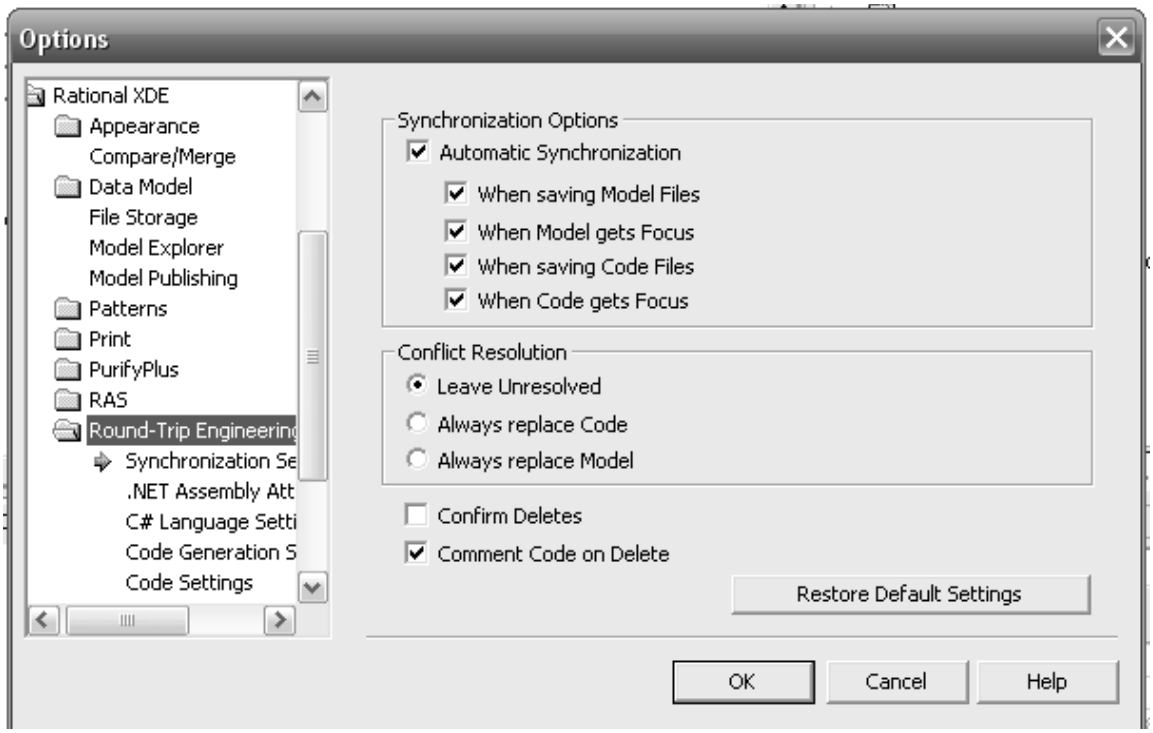

Рисунок 11.5 – Настройка синхронизации кода

Каждый вариант синхронизации удобен для своего случая. На этапе анализа и проектирования, когда основную работу по моделированию выполняет аналитик, синхронизации в момент сохранения модели позволит программистам, работающим в проекте, пользоваться самой последней версией структуры приложения. На этапе реализации, когда основная работа ложится на программистов, установка синхронизации после сохранения внесения изменений в файлы кода позволит поддерживать модель системы в актуальном состоянии. Когда же разработка закончена или создается новая версия уже работающей системы, чтобы не нарушить работу программы, синхронизация может быть вообще отключена.