

## ТЕМА 12

План лекции:

1. Модуль *Web Modeler* в *Rational Rose*.
2. Последовательность создания web-приложения в *Rational Rose*.
3. Особенности создания Web-приложений в *Rational XDE*.

### 12.1 Модуль *Web Modeler* и его инструменты в *Rational Rose*

Возможности среды разработки *Rational Rose* не ограничиваются созданием модели программной системы и генерированием по ним исходного кода приложения. Подключаемые модули позволяют использовать *Rational Rose* еще и для создания модели баз данных и разработки структуры Web сайтов. Это логично, поскольку приложение, работающее на Internet сервере, в принципе ничем не отличается от обычной программы. И даже простой Web сайт с линейной структурой страниц, который не использует средства *Java* или *ASP (Active Server Pages)* можно представить как приложение. Web сайту присущи все атрибуты настольного приложения: взаимодействие с пользователем, получение, хранение и передача информации, окна, меню.

Структура Web сайта также разрабатывается, затем кодируется и тестируется и, наконец, выставляется для всеобщего использования. Далее сайт сопровождается, добавляются и удаляются разделы, исправляются ошибки, расширяется информационное наполнение и функциональность.

Internet приложение моделируется на основе диаграммы классов. В *Rational Rose* для создания Web приложений включен модуль *Web Modeler*. Для того чтобы получить к нему доступ, нужно выполнить следующие действия: *Add-Ins => Add-Ins Manager => Rose Web Modeler = ON*, после чего в меню *Tools* появится новый пункт *Web Modeler*. *Web Modeler* позволяет моделировать Web приложения, создавать модель по существующему программному коду, а затем снова выполнять генерацию *jsp*, *asp*, *html* файлов согласно изменениям в модели. *Web Modeler* позволяет разработчику все внимание уделять бизнес логике приложения и связям, беря на себя детали реализации.




Меню *Web Modeler* состоит из двух пунктов: *User Preferences* (установки пользователя) и *Reverse Engineer a New Web Application*.

Пункт *User Preferences* предназначен для изменения установок модели, принятых по умолчанию для конкретного пользователя, что позволяет нескольким пользователям работать с одной Web моделью, но со своими настройками. Эти установки позволяют контролировать построение модели по коду, создание кода по модели, генерировать структуру каталога для ссылок, связей и HTML форм. Установки позволяют настроить действия генератора для тегов HTML и скриптов. Возможен выбор между удалением или установкой операции и тегов в роли комментариев, что необходимо для предохранения полученного кода от обработки внешними приложениями.

Пункт *Reverse Engineer a New Web Application* позволяет создавать модель по готовому коду приложения. При его активизации запускается

мастер Reverse Engineer, который помогает осуществить использование готового кода для создания модели. Мастер позволяет задать платформу ASP или JSP (Java Server Pages) и каталог, в котором находится исходный код. После того как мастер закончит работу в текущей модели, образуется структура выбранных классов, в которой отражаются связи, атрибуты и стереотипы.

Для построения Web приложения используется диаграмма классов, и их следует создавать в Web нотации. Для этого необходимо выбрать следующие пункты *Menu:Tools=>Options=>Notation=>Default Language=Web Notation*. При этом классы имеют специальные стереотипы, которые влияют на изображения класса на диаграмме. Для моделирования Web приложения

Rational Rose предоставляет ряд стереотипов классов:  – *Server page*,  – *Client page*,  – *HTML Form*. Рассмотрим их подробнее.

Стереотип *Server page* отображает скрипты, выполняемые на стороне сервера. Это может быть ввод данных и обработка запросов. Для уменьшения сложности Web приложения проектировщик может моделировать функциональность серверной части приложения при помощи стереотипа *Server page*. Web сервер выполняет переданный ему код и возвращает форматированные данные клиенту.

Стереотип *Client page* позволяет показать использование скриптов на стороне клиента, кроме того, он отражает простые *HTML* страницы, текст, графику, мультимедиа объекты. Страницы могут комбинировать статический *HTML* текст и динамические страницы, созданные при помощи *JavaScript* и *VBScript*. Динамические страницы могут управляться программными событиями, содержать переменные, подпрограммы и функции.

*HTML* формы позволяют пользователю взаимодействовать с Web-приложением. При помощи форм происходит обработка ввода пользователя и передача его серверу. Web серверы принимают запросы, сформированные посредством форм, обрабатывают и возвращают *HTML* страницы клиенту. Когда создается модель Web приложения, формы используются для объединения полей ввода на клиентских страницах и не содержат собственных операций. Атрибуты в формах могут быть представлены только как поля ввода.

Для связей страниц и форм *Web Modeler* предоставляет дополнительные стереотипы, которые позволяют отразить характер связей. Доступ к стереотипам связей можно получить с помощью контекстного меню связи в разделе спецификации. Рассмотрим имеющиеся стереотипы.

- *Link* показывает гиперссылку одной страницы на другую и может быть двунаправленной.

- *Submit* определяет, что форма взаимодействует со страницей на сервере и передает ей данные.

- *Build* показывает, что страница создается сервером.

- *Redirect* определяет передачу управления одной серверной странице от другой и используется для страниц ASP.
- *Includes* означает, что одна страница включается в другую.
- *Forward* похожа на связь *Redirect*, только используется для страниц JSP.

## 12.2 Построение Web-приложения в Rational Rose

После того как рассмотрены стереотипы, с их помощью можно создавать простые Web-приложения. Предположим, необходимо построить приложение, состоящее из главной страницы, которая создается сервером и включает меню. На этой же странице находится форма ввода, которая позволяет отправлять сообщения. Форма ввода будет взаимодействовать со страницей сервера, которая и будет непосредственно отправлять запросы. Описанное приложение приведено на рис. 12.1.

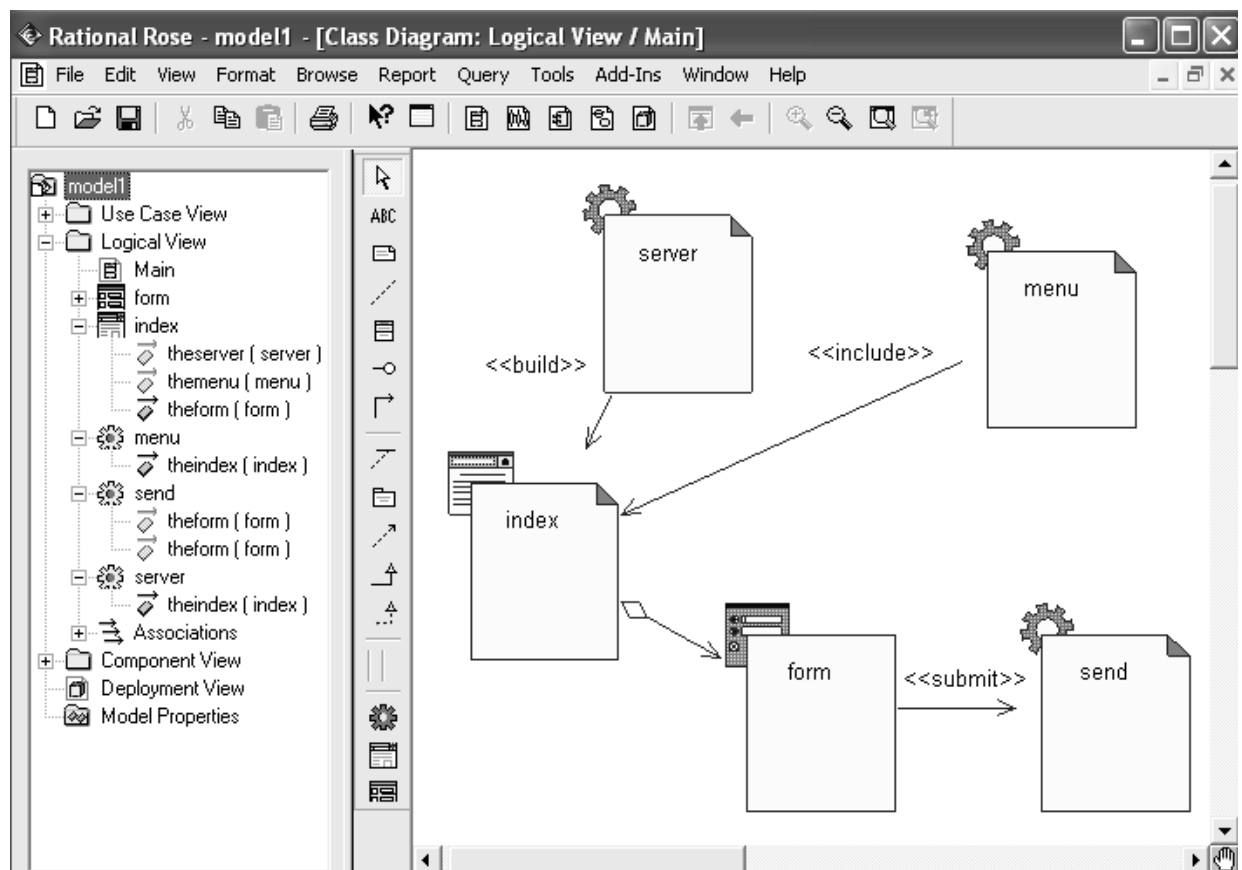


Рисунок 12.1 – Web модель

Для разработки приложения необходимо пройти несколько шагов.

1. Установка языка моделирования, как было описано ранее.
2. Из контекстного меню *Logical View* выбрать *Web Modeler* => *New* => *Virtual Directory Name* и задать каталог, в котором будет находиться исходный текст, сгенерированный на основе модели приложения.
3. Добавить необходимые элементы в модель посредством строки инструментов диаграммы классов.

4. Установить свойства и добавить атрибуты.
5. Создать исходный текст.

### 12.3 Особенности создания Web-приложений в Rational XDE

В *Rational XDE* для создания структуры системы, ориентированной на работу в Web, используется диаграмма классов, в которой при создании архитектуры приложения учитываются ограничения реализации Web-приложения. В спецификации *UML* не предусмотрен отдельный тип диаграмм для выполнения этой задачи. Вполне достаточно диаграммы классов с дополнительными стереотипами для работы с Web-элементами. На рис. 12.2 приведены инструменты для разработки структуры Web-приложения. Набор инструментов существенно расширился по сравнению с программой *Rational Rose*. Это объясняется тем, что *Rational Rose* поддерживает разработку только ASP-страниц, а *Rational XDE* предназначена для ASP.NET. Кроме стандартных Web-инструментов, на панель вынесены значки, которые создают целые группы элементов.

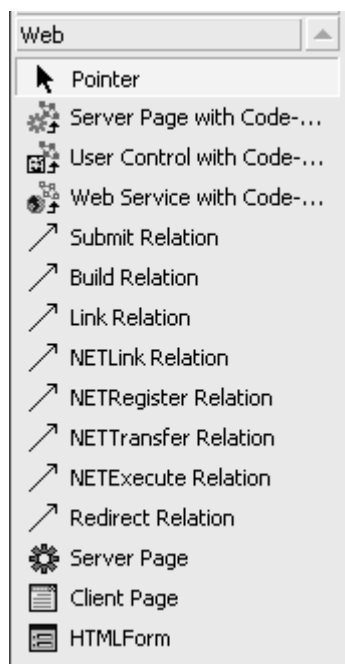


Рисунок 12.2 – Панель Web-окна Toolbox

Для построения Web-модели можно использовать новое или существующее приложение, создать в нем новую модель, а затем выполнить синхронизацию проекта с моделью. Для этого в контекстном меню проекта необходимо выбрать пункт *Synchronize*. В случае, если модель еще не создана, *Rational XDE* создает новую модель с таким же названием, как у проекта, и проставляет все необходимые ссылки. После синхронизации *Rational XDE* открывает новую диаграмму, готовую к работе.

Рассмотрим возможности *Rational XDE* по созданию Web-модели.

Значок *Client Page* позволяет создать на диаграмме отображение простых страниц HTML, не имеющих собственного поведения. Обычно такие страницы предоставляют пользователям определенную, заранее заданную информацию. Страницы *Client Page* так же, как и классы, могут содержать атрибуты и операции, которые добавляются посредством контекстного меню элемента, после чего код обновляется автоматически или вручную.

Элемент *Link Relation* позволяет отобразить связи между страницами в том случае, когда на одной странице есть ссылка на другую. Rational XDE не знает, куда добавлять созданную ссылку, и вставляет ее в конец файла.

Значок *HTML Form* позволяет отобразить формы ввода, которые присутствуют на страницах HTML. Форма не может существовать сама по себе, она включается на страницу при помощи агрегирования. Поэтому для ее разработки начинают с создания страницы, на которой будет находиться форма. Сначала форма соединяется со страницей связью *Link*, а затем посредством пункта *Properties Window* из контекстного меню связи значение свойства *UML=>Kind* изменяется на *Agregation*. Для того чтобы код *Page1* изменился, необходимо в окне Model Explorer отбуксировать элемент *Form1* в элемент *Page1*. После чего в результате генерации кода в код страницы добавятся строки обработки формы. Для добавления полей в форму можно воспользоваться пунктом ее контекстного меню *Add Web*.

На рис. 12.3 показаны клиентские страницы *Page1* и *Page2*, а также *Form1*. Все элементы соединены вышеописанными связями. Под диаграммой приведен автоматически сгенерированный код.

В случае необходимости отразить обработку данных, передаваемых из формы клиентской или серверной странице, используется значок связи *Submit Relation* (отношение предоставления).

Создание клиентских страниц вручную происходит достаточно редко: только в случае разработки статичного приложения. Поскольку основная логика приложения должна работать на сервере сети, Web-приложение создает клиентские страницы динамически по запросам пользователей. Для этого используются *Server Page*, которые и реализуют генерацию страниц для пользователя, что отображается при помощи связи *Build Relation*. Таким образом, *Server Page* являются связующим звеном между классами приложения и их визуальным отображением.

На рис. 12.4 приведена серверная страница и сгенерированный по ней исходный код.

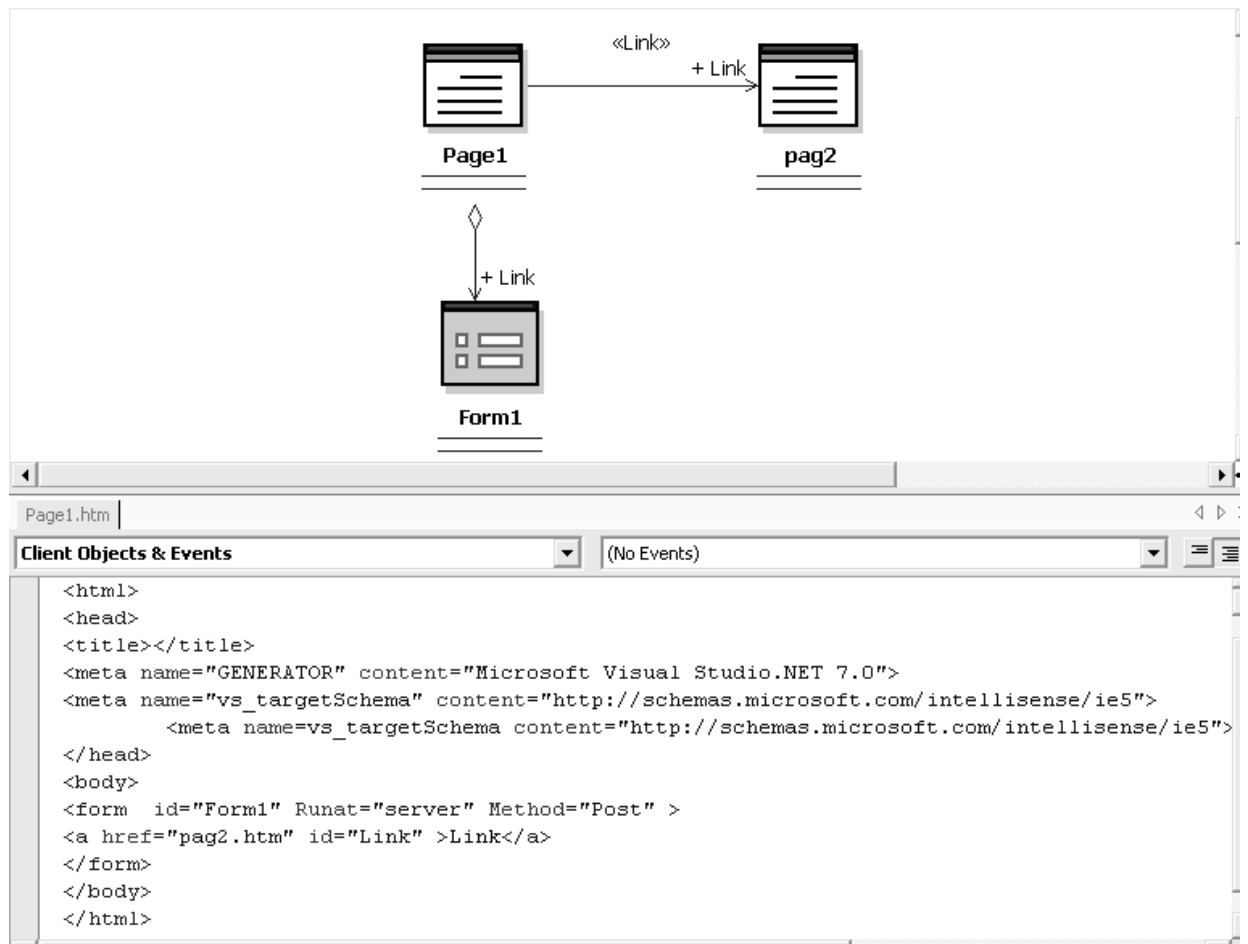


Рисунок 12.3 – Web-модель и сгенерированный по ней код

При помощи значка *Server Page with Code-Behind* создается набор элементов (рис. 12.5), связанных между собой и содержащих необходимые элементы для создания ASP.NET приложения.

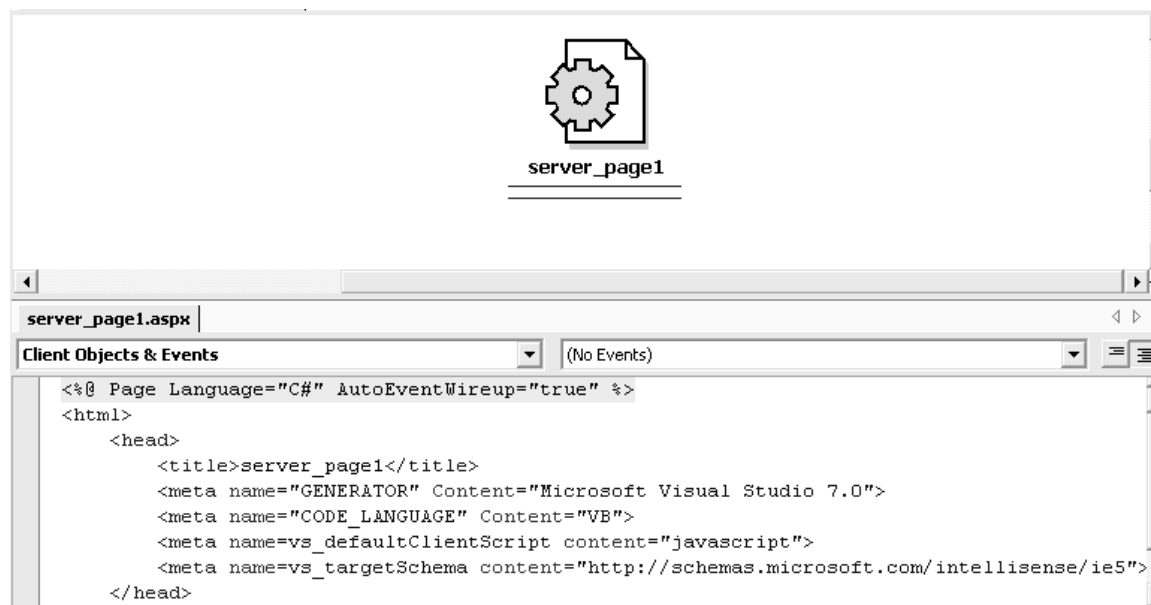


Рис. 12.4 – Серверная страница и сгенерированный по ней код

Значок *User Control with Code-Behind* позволяет отражать создание пользовательских элементов управления вместе с классом их обработки. На диаграмме создаются два элемента: страница пользовательского элемента управления и класс, от которого выполняется наследование.

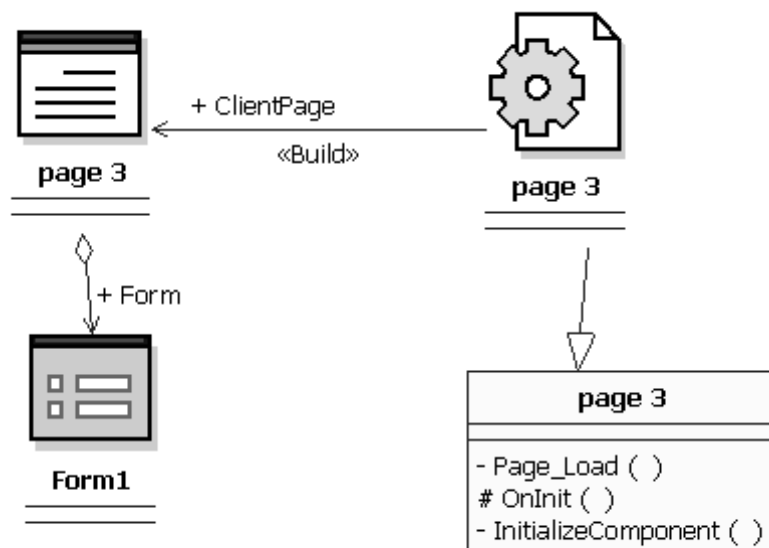


Рис. 12.5 – Пример Server Page with Code-Behind

Значок *Web Service with Code-Behind* позволяет отражать создание сервисов, которые предоставляют информацию приложениям вместе с классом их обработки.

Для отражения связей между страницами ASP используется значок *NETLink Relation*. Если форма, расположенная на ASP странице, использует элементы управления, созданные пользователем, то значок *NETRegister Relation* позволяет отразить связи между страницей ASP и элементом управления пользователя.

Для отражения передачи управления другой странице используется значок *NETTransfer Relation*. При генерации кода создается директива *Transfer*, которая позволяет передавать управление другой странице с сохранением доступа к внутренним объектам исходной страницы.

Значок *NETExecute* позволяет отразить передачу управления другой странице, но при генерации кода создается директива *Execute*, позволяющая не только передать управление с сохранением доступа к внутренним объектам, но и по завершении вернуть управление вызывающей странице.

Для отражения простой переадресации с одной страницы на другую используется связь при помощи значка *Redirect Relation*. При этом не сохраняется доступ к внутренним объектам, как это происходит при использовании связей *NETTransfer* и *NETExecute*. Такая переадресация используется при необходимости активизации страницы, чье изображение зависит от установленного языка или возможностей браузера.