Разработка приложений ECO в Borland C#Builder и Borland Delphi 8 for the Microsoft .Net Framework

Автор: ***Евгений Даниленко***
EADSOFT.COM
Источник: ***RSDN Magazine #3-2004***

Опубликовано: *06.11.2004*
Версия текста: *1.0*

HRSampleDelphi.zip – пример на Delphi
HRSampleCSBuilder.zip – пример на C#

Процесс разработки ECO приложений

Процесс разработки ECO приложений состоит из трех этапов:

1. Построения модели;
2. Реализации бизнес правил;
3. Построение пользовательского интерфейса;

Весь процесс разработки будет показан на примере приложения, облегчающего учет персонала компании. Приложение будет построено на базе одной и той же модели в средах C#Builder и Delphi.

Построение модели

При построении модели подразумевается, что разработчик имеет представление о предметной области, о бизнес-сущностях процесса, который ему предстоит автоматизировать, и о взаимосвязях между бизнес-сущностями. В этой статье не будет рассматриваться процесс детерминирования бизнес-процессов и бизнес-сущностей.

Чтобы начать разработку приложения, нужно вызвать соответствующий мастер из главного меню сред C#Builder (см. рис 1) или Delphi 8 (см. рис 2).

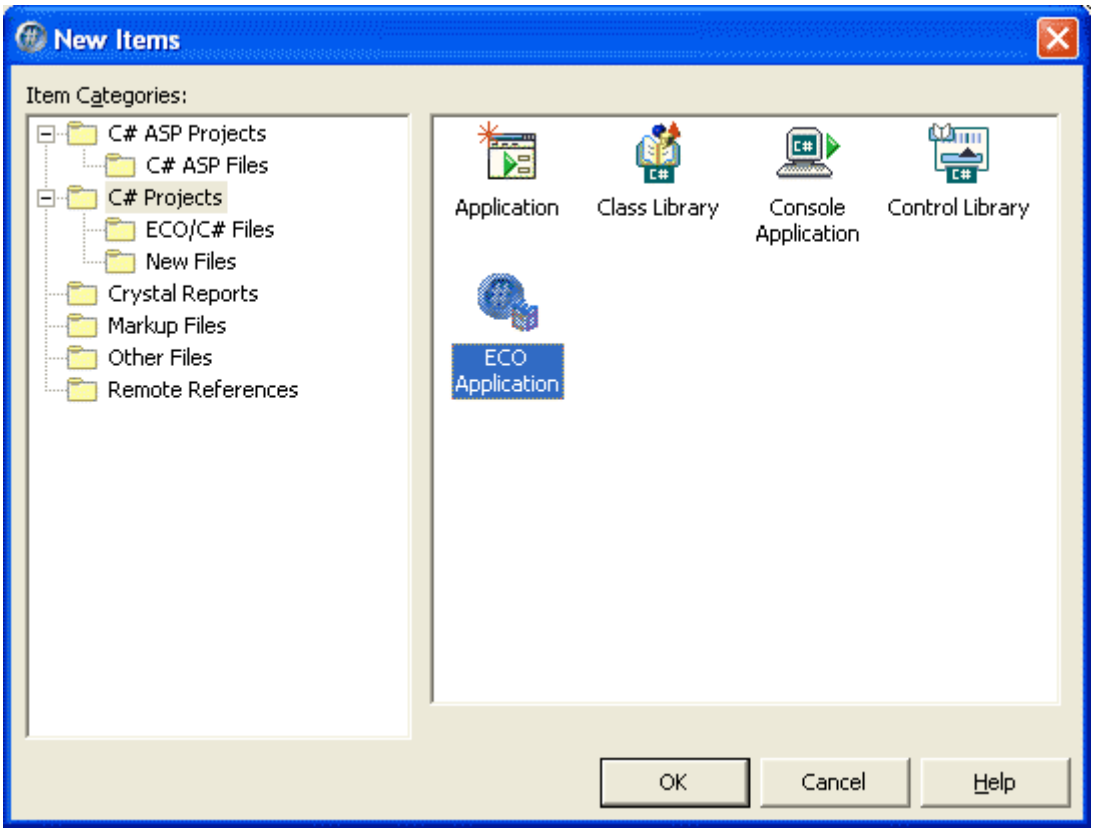

Рисунок 1. Выбор создания приложения в C#Builder

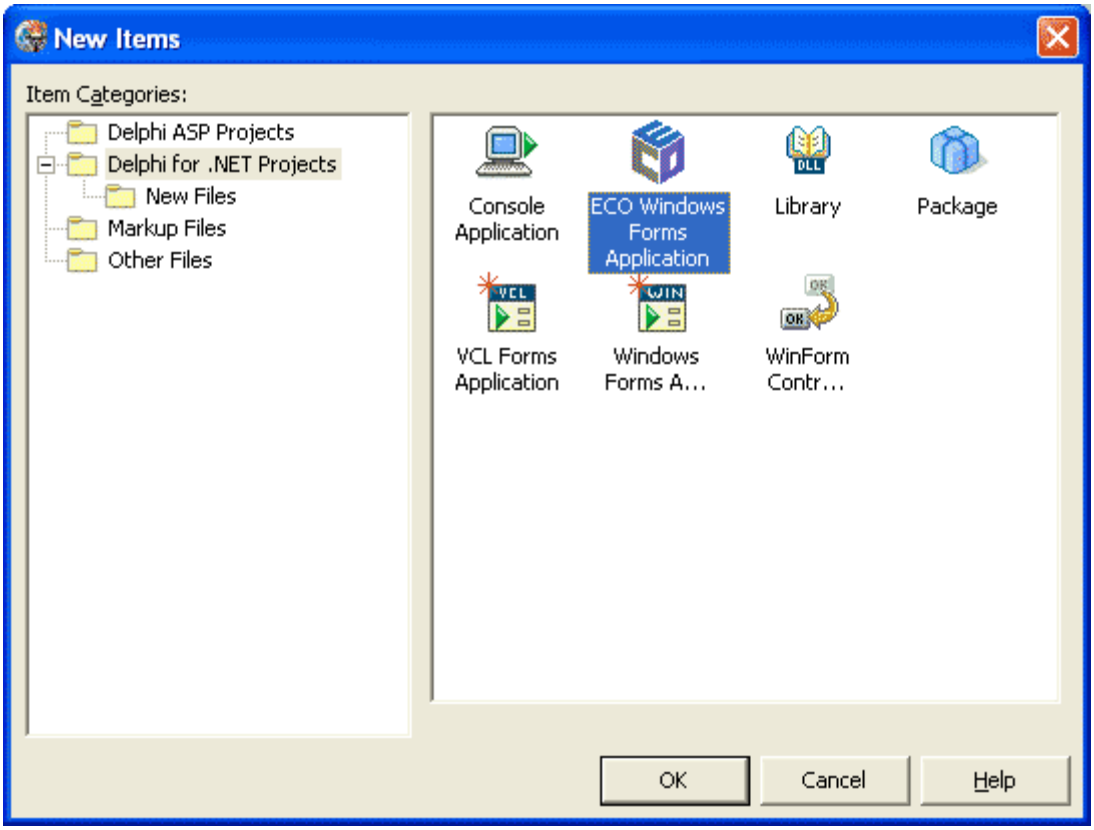

Рисунок 2 Выбор создания приложения в Delphi 8

После этого в обеих средах будет дан запрос на выбор пути, где будет сохранен проект (рис 3).

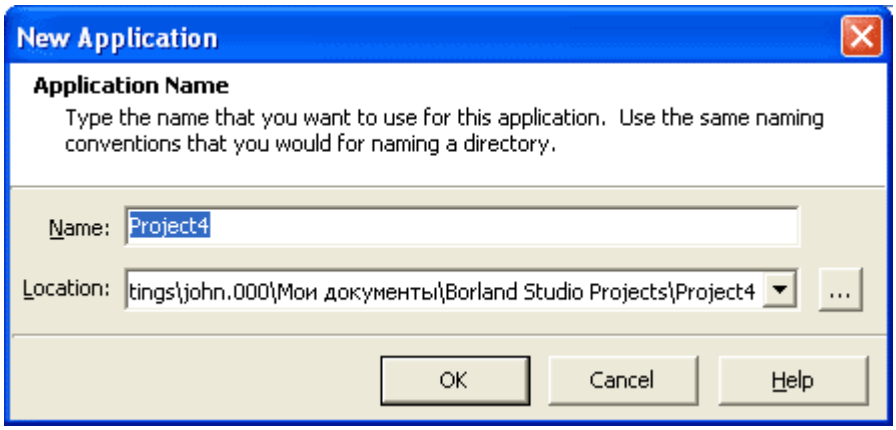

Рисунок 3. Запрос пути создания проекта.

Далее среды создают минимальный набор файлов, содержащих все необходимые элементы ECO-приложения. Набор файлов у обеих сред разный, хотя есть и общие элементы (таблица 1).

	EcoSpace	Бизнес - сущности	Главная форма приложения	ECO-форма
C#Builder	EcoSpace.cs	CoreClasses.cs	WinForm.cs	EcoWinForm.cs
Delphi	HRSampleEcoSpace.pas	CoreClassesUnit.pas	WinForm.pas	

Таблица 1. Файлы, создаваемы мастерами ECO-приложения.

Все создаваемые файлы приложения можно просмотреть в Project Manager (C#Builder – рисунок 4, Delphi – рисунок 5).

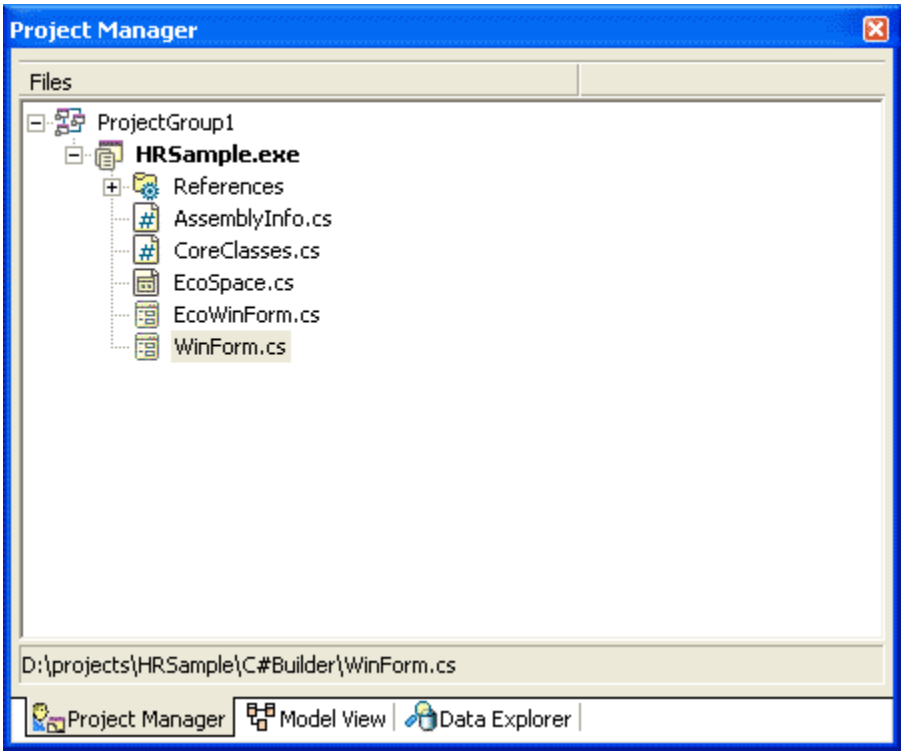

Рисунок 4. Вид Project Manager в C#Builder.

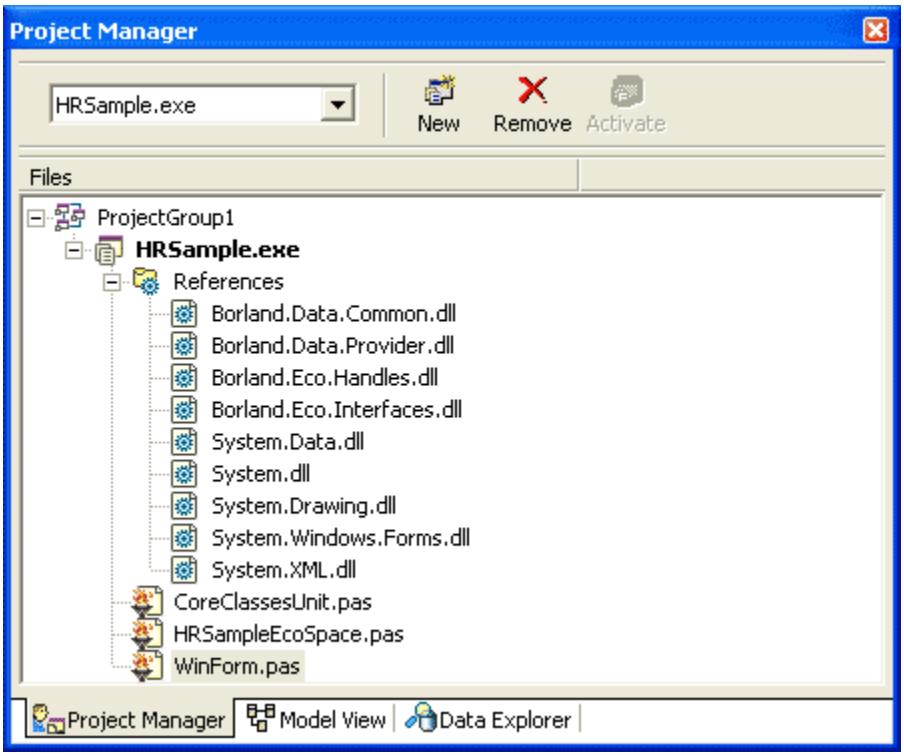

Рисунок 5. Вид Project Manager в Delphi 8.

В обеих средах построение модели происходит в окне модели, для этого надо выбрать Model View в главном меню среды. Следует обратить внимание, что в модели приложения отображаются не только диаграммы бизнес-сущностей, но и все классы, определенные в приложении: это классы форм и т.д. Все эти классы отображаются по-разному при просмотре моделей. Классы, интересующие нас, отображаются внутри пакетов, в данном случае в CoreClassPackage для C#Builder (рисунок 6) и CoreClass в Delphi 8. (рисунок 7).

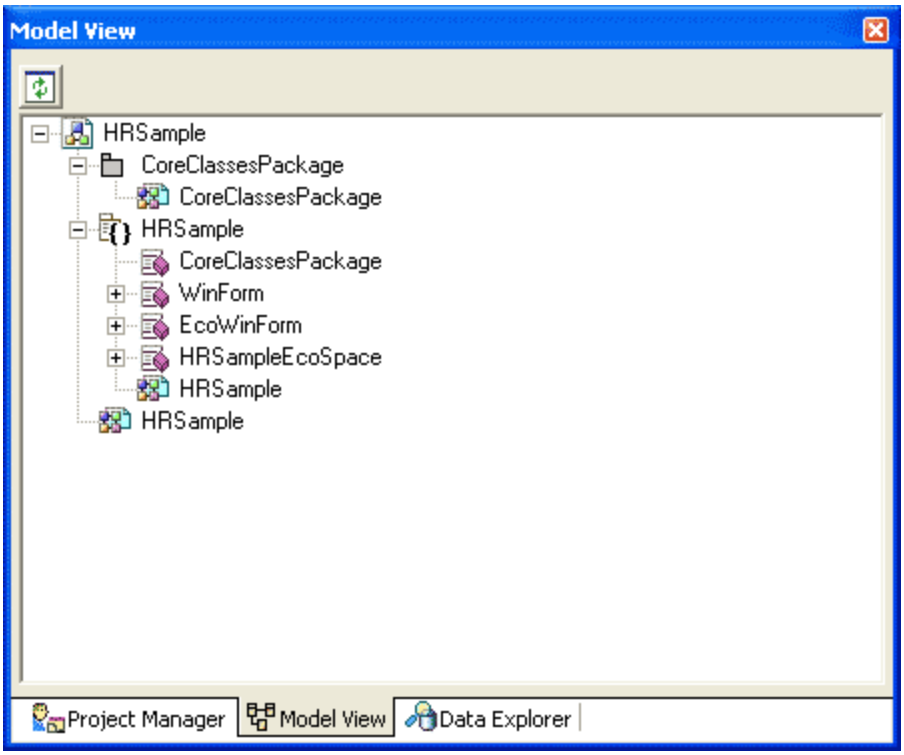

Рисунок 6. Вид окна просмотра модели в C#Builder.

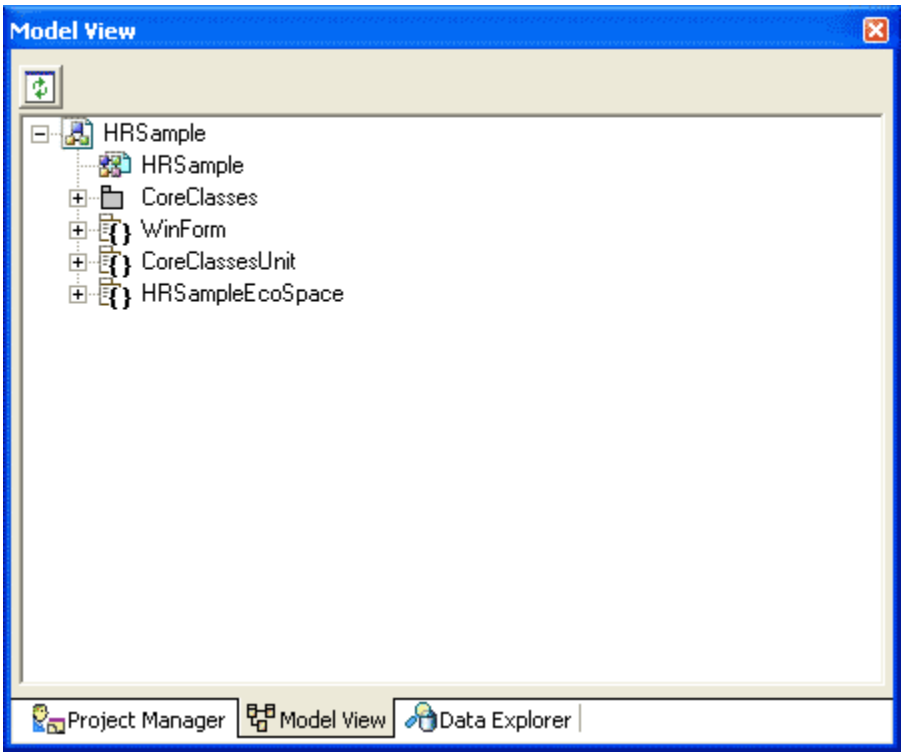

Рисунок 7. Вид окна просмотра модели в Delphi 8.

Итак, все готово для того, чтобы начать строить модель бизнес-сущностей приложения. На этой модели мы определим атрибуты сущностей и связи между ними. Для этого вызовем окно построения модели, выбрав в контекстном меню пункт Open Diagram (Рисунок 8).

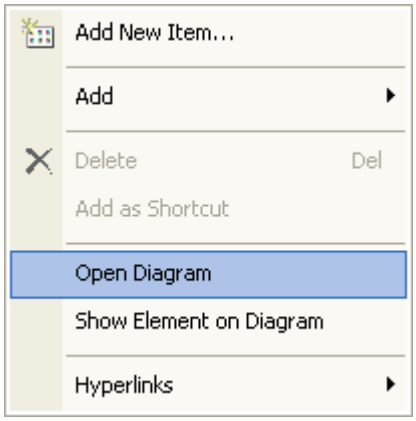

Рисунок 8. Контекстное меню окна просмотра модели

Начнем с базового класса нашей модели, именно от этого класса будут наследоваться все классы бизнес сущностей нашего приложения. Для построения моделей существует линейка инструментов, представленная на рисунок 9.

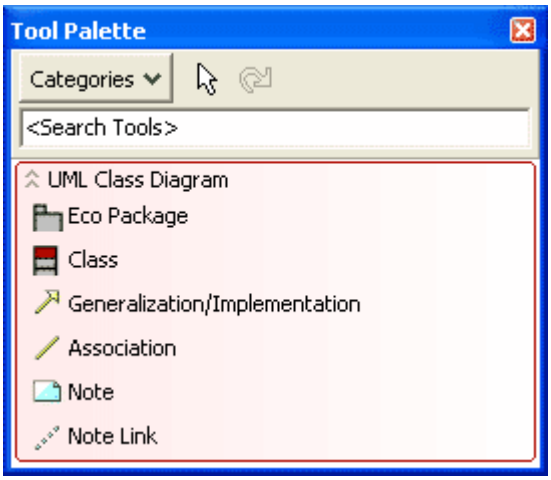

Рисунок 9. Панель инструментов построения модели

Поместив изображение класса в окно отображения модели, можно сразу переименовать его. Наш базовый класс будет называться BaseClass. (рисунок 10).

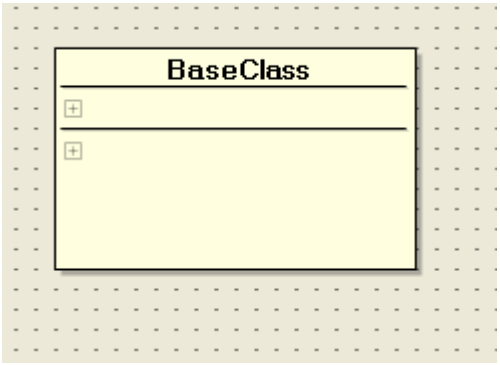

Рисунок 10. Пустой базовый класс.

В Object Inspector будет отображаться свойства класса в модели. В C#Builder (рисунок 11) набор свойств класса несколько меньше, чем в Delphi 8 (рисунок 12).

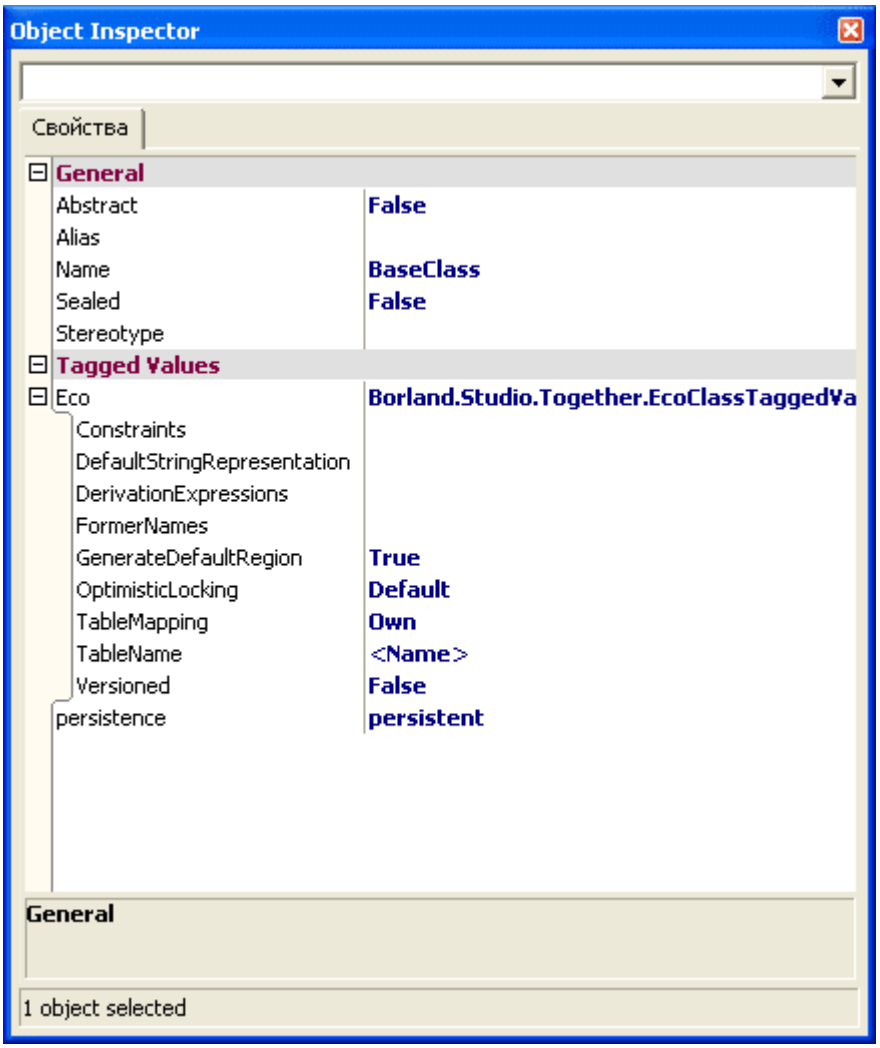

Рисунок 11. Набор свойств класса модели в C#Builder.

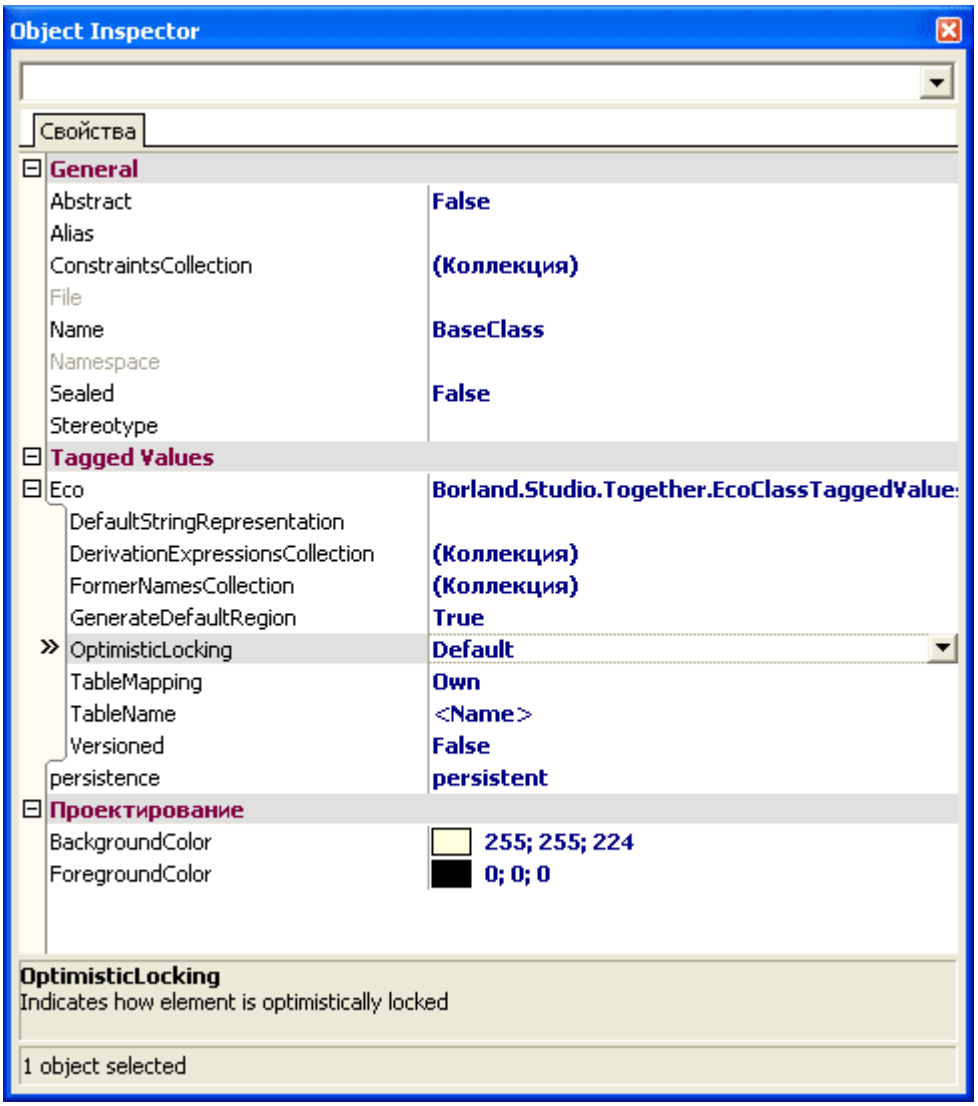

Рисунок 12. Набор свойств класса модели в Delphi 8.

Свойства класса позволяют управлять поведением класса в дальнейшем, указывают на место хранения в базе данных и т.д. Остановимся подробнее на этих свойствах (Таблица 2).

Наименование	C#Builder	Delphi 8	Описание
Abstract	+	+	Показывает, является ли класс абстрактным
Alias	+	+	Псевдоним. Если это свойство заполнено, то на диаграмме отображается псевдоним вместо названия
ConstraintsCollection		+	Ограничения класса.
Constrains	+		Ограничения класса.
File		+	Имя файла
Name	+	+	Название класса

NameSpace		+	Наименование пространства имен
Sealed	+	+	Показывает, является ли класс «герметичным»
Stereotype	+	+	Указывает на стереотип класса
Eco.DefaultStringRepresentation	+	+	Указывает на представление класса в виде строки по умолчанию
Eco.DerivationExpressionsCollections		+	Определяет выражения формул для класса
Eco.DerivationExpressions	+		Определяет выражения формул для класса
Eco.FormerNames	+		Бывшие имена класса. Используются для выражений
Eco.FormerNamesCollection		+	Бывшие имена класса. Используются для выражений
Eco.GeneralDefaultRegion	+	+	Показывает, генерирует ли класс автоматически описание региона
Eco.OptimisticLocking	+	+	Определяет стратегию оптимистического блокирования
Eco.TableMapping	+	+	Определяет, в какой таблице данных будет храниться класс
Eco.TableName	+	+	Определяет имя таблицы, в которой будут храниться экземпляры класса. Если это свойство пусто, то название таблицы соответствует названию класса.
Eco.Versioned	+	+	Определяет, применяется ли к классу версионное хранение
Persistence	+	+	Показывает, является ли класс сохраняемым.
BackgroundColor		+	Цвет фона элемента
ForegroundColor		+	Цвет изображения элемента.

Таблица 2. Список свойств класса в модели.

Теперь необходимо определить атрибуты сущности. В нашем случае атрибутами сущности будут дата создания – created и дата модификации – modified. Чтобы создать атрибут, нужно выбрать из контекстного меню класса на диаграмме пункт Add|Atribute (рисунок 13).

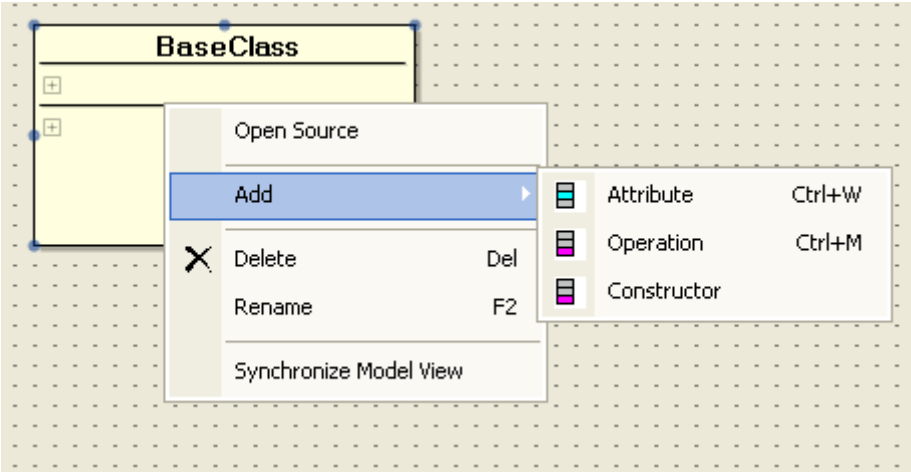

Рисунок 13. Добавление нового атрибута в класс.

Итак, у нас есть базовый класс BaseClass, имеющий два атрибута. Теперь необходимо создать оставшиеся бизнес-сущности и указать, от каких классов будут наследоваться классы-наследники. В нашем случае нужно добавить два класса: Department и Person, имеющих один общий атрибут – название или имя. Для этого имеет смысл завести свой базовый класс, который и будет содержать этот атрибут. Так же мы помним, что у нас есть базовый класс. Мы добавляем класс BaseNameClass с атрибутом Name и, используя инструмент Generalization/Implementation из палитры инструментов, устанавливаем наследование. Далее добавляем классы Department и Person, устанавливая связи наследования аналогичным образом. После этого мы получим модель, показанную на рисунке 14.

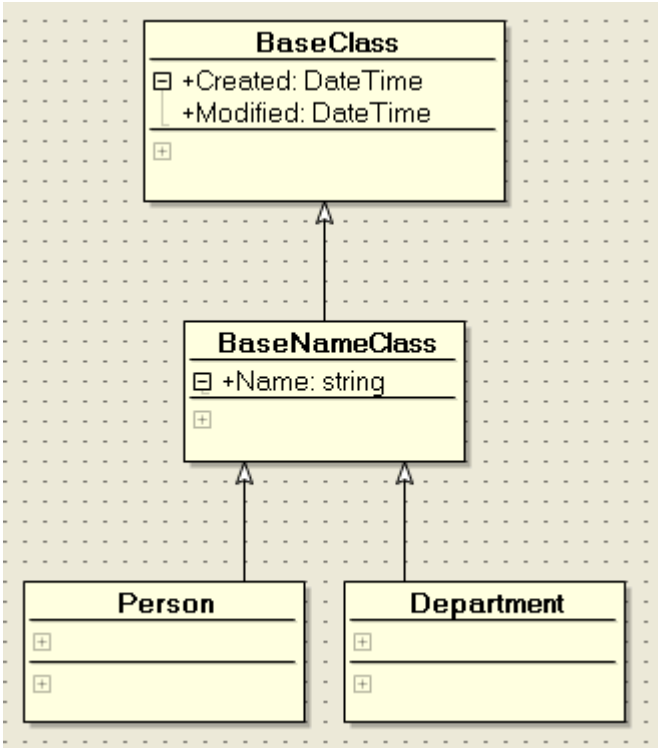

Рисунок 14. Модель с базовыми классами.

Далее нужно определить сущности "Должность" (Position) и "Заработная плата" (Salary), обращая внимание на то, от каких классов проводить наследование. Внешний вид диаграммы модели показан на рисунке 14. Стоит обратить внимание, что пока на нашей модели описаны только сами сущности, но не их поведение и связи, т.е. бизнес-логика.

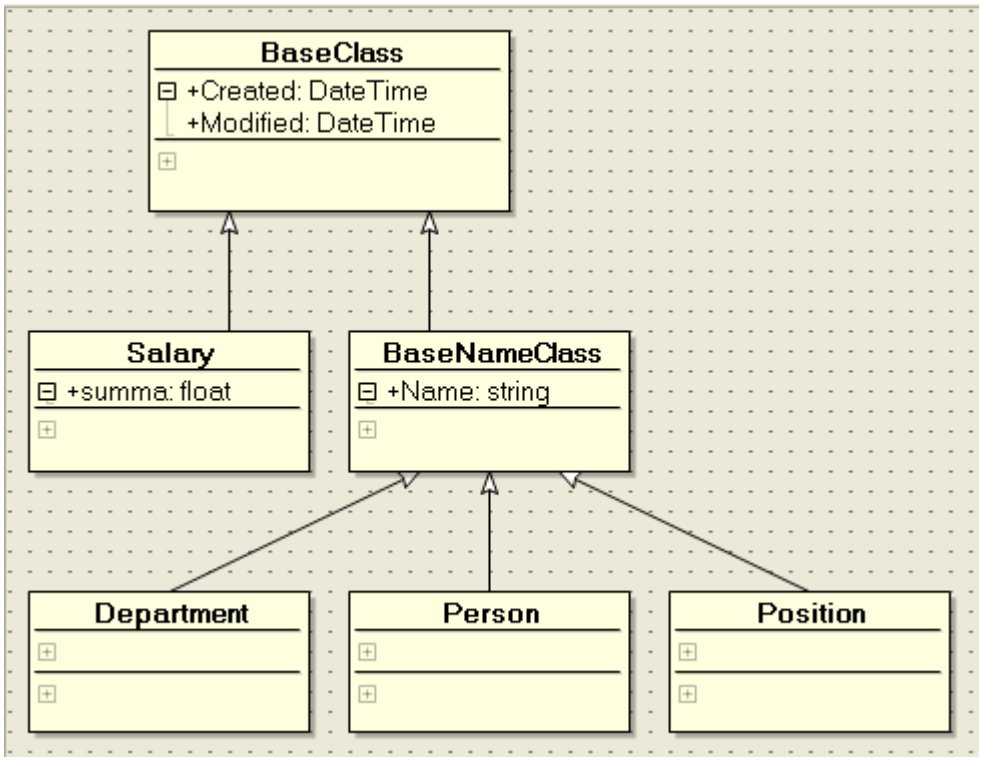

Рисунок 15. Модель со всеми сущностями приложения.

Необходимо отметить, что не только класс в модели имеет набор свойств, но и атрибуты класса также имеют набор свойств, определяющий их поведение и правила хранения. Используя их, можно гибко настраивать правила хранения сущностей в базе данных. Рассмотрим свойства атрибутов сущностей.

Наименование	C#Builder	Delphi	Описание
Alias	+	+	Псевдоним. Если свойство заполнено, то в модели вместо названия отображается псевдоним.
InitialValue	+	+	Начальное значение. Присваиванется в виде OCL выражения.
Name	+	+	Имя атрибута.
Stereotype	+	+	Стереотип атрибута.
Type	+	+	Тип данных атрибута.
Visibility	+	+	Видимость атрибута в классе.
derived	+	+	Показывает, является ли атрибут вычисляемым.
AllowNULL	+	+	Показывает, разрешено ли значение NULL для атрибута.

ColumnName	+	+	Имя колонки в таблице данных. Если свойство пустое, то в качество имени поля используется имя атрибута.
DefaultDBValue	+	+	Значение по умолчанию в базе данных.
DelayedFetch	+	+	Показывает, будет ли атрибут автоматически доставлен из БД при загрузке экземпляра класса.
DerivationOCL	+	+	OCL выражение, используемое для вычисления значения атрибута.
DerivedSettable	+	+	Указывает, может ли вычисляемоае значение быть установленным. Т.е. установленным программным путем, без OCL.
FormerNames	+		Бывшие имена класса. Используются для выражений.
FormerNamesCollection		+	Бывшие имена класса. Используются для выражений.
Length	+	+	Длина значения.
PMapper	+	+	Persistence mapper, используемый для атрибута.
persistence	+	+	Показывает, является ли класс сохраняемым.
BackgroundColor		+	Цвет фона элемента.

Таблица 3

Построение бизнес логики

Бизнес-логика может быть построена с использованием двух механизмов: описанием связей между сущностями и описанием операций над сущностями. В нашем случае мы опишем связи между сущностями прямо в модели. Для этого используется инструмент Association из панели инструментов. В нашем приложении существует несколько связей:

- между Department и Person. В отдел входит несколько работников.
- между Person и Position. Каждый работник занимает какую-то должность.
- между Person и Salary. Каждый работник получает некую заработную плату.

Итак, мы получили модель следующего вида (рисунок 16)

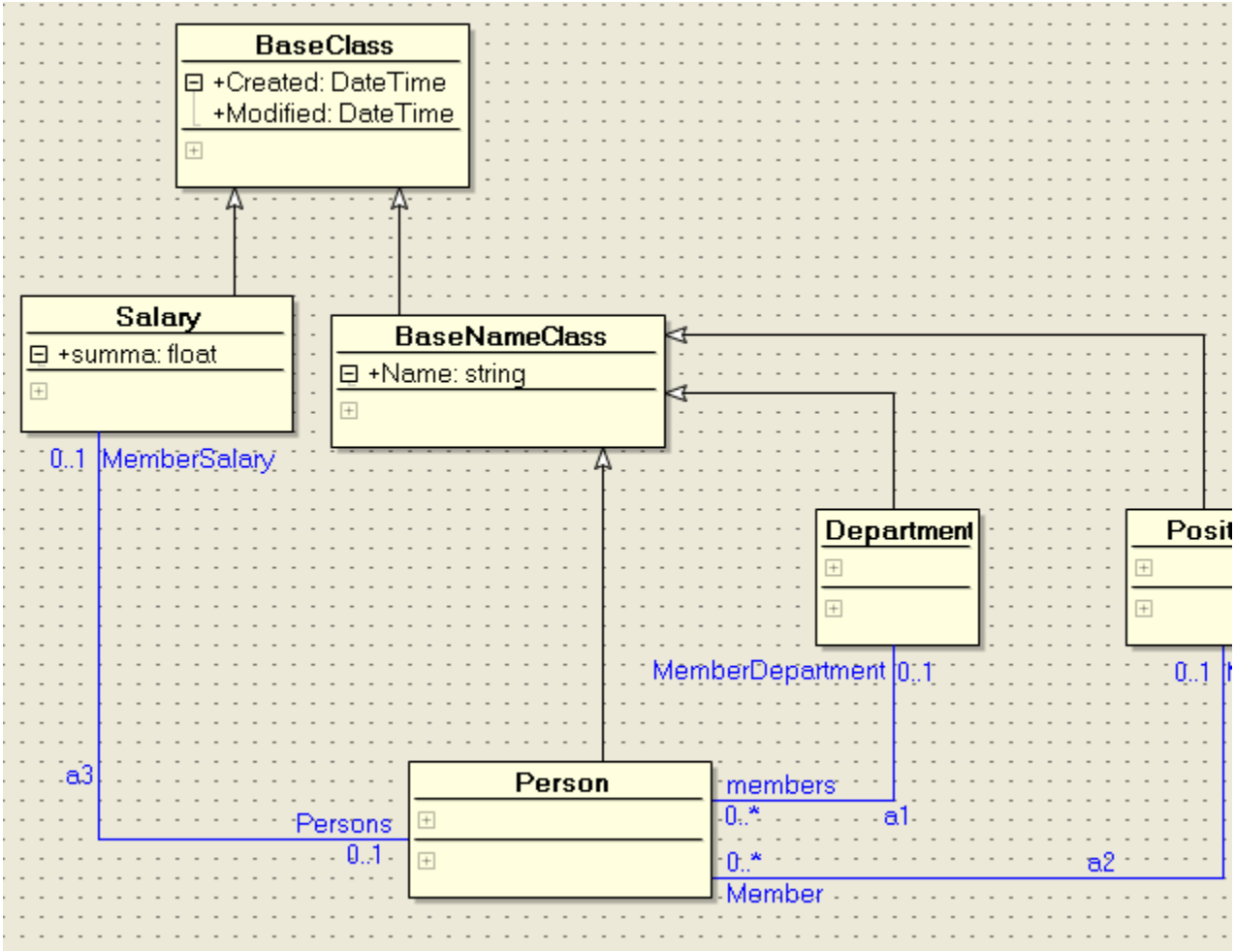

Рисунок 16. Модель с установленными связями.

Каждая связь имеет свой набор свойств, отображаемых в Object Inspector (рисунок17).

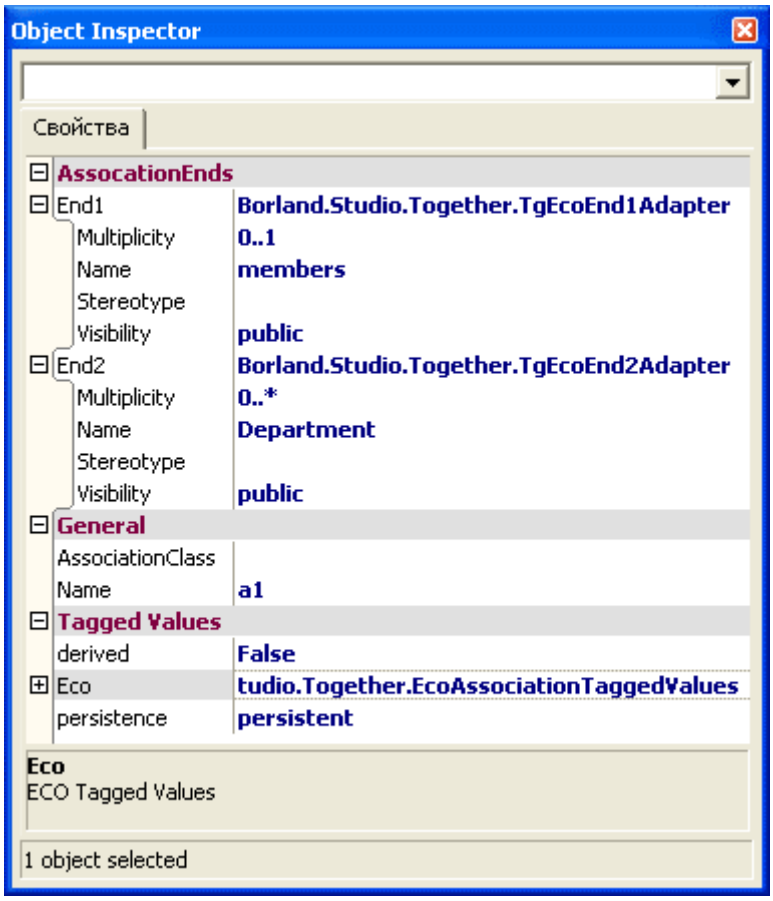

Рисунок 17. Свойства связей

Остановимся на наиболее часто употребляемых и важных (Таблица 4).

End1	Первое окончание связи
Multiplicity	Тип отношения
Name	Наименование окончания
Stereotype	Стереотип окончания
Visibility	Видимость окончания
End2	Второе окончание связи
Multiplicity	Тип отношения
Name	Наименование
Stereotype	Стереотип окончания
Visibility	Видимость окончания
Name	Имя отношения

Таблица 4

Итак, свойства End1 и End2 показывают два окончания связи, у каждого из них есть имя Name, под которым они будут видны в OCL-выражениях, и тип отношения. Тип отношения бывает четырех видов:

- 0..1 – отношение 0 или 1;
- 0..* - отношение 0 или множество;
- 1..1 – отношение «один к одному»;
- 1..* - отношение «один ко многим».

Таким образом, построив модель, можно перейти к выбору средства хранения данных приложения.

Здесь надо остановиться на одной очень полезной и важной особенности ECO. Для модели ECO совершенно безразлично, где будут храниться данные. Обращение к данным идет через слой Persistence, который берет на себя все заботы по сохранению данных, генерации структуры базы данных и т. д. Persistence может хранить данные в следующих хранилищах:

- XML
- MS SQL 2000/MSDE
- Базы данных, к которым есть драйвер Borland Data provider (BDP) - Interbase, Oracle, DB2 или Informix.

Чтобы указать, где будут храниться данные, нужно указать Persistence Mapper. Для этого необходимо перейти в окно EcoSpace (в C#Builder – EcoSpace.cs, в Delphi 8 - HRSampleEcoSpace.pas) и положить соответствующий компонент в EcoSpace. Мы будем использовать в качестве сервера БД Borland Interbase, поэтому нам необходим PersistenceMapperBdp и BdpConnection. В PersistenceMapperBdp1 значение свойства Connection устанавливаем в BdpConnection1, а свойства PersistenceMapper в EcoSpace – в PersistenceMapperBdp1. После этого нужно настроить PersistenceMapperBdp1 на использование соответствующего драйвера BDP, используя мастер, который показывается при выделении компонента PersistenceMapperBdp1 (рисунок 18).

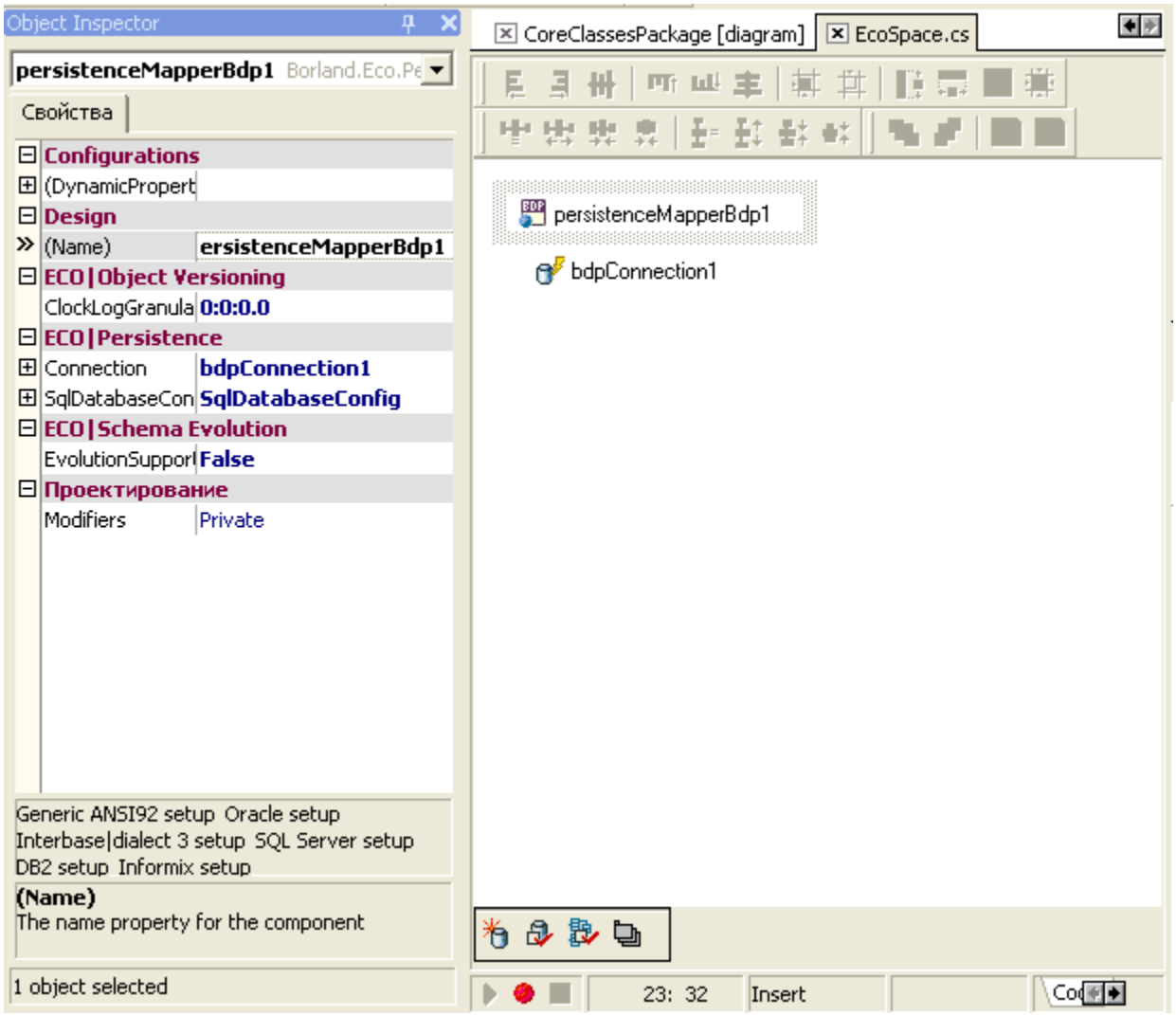

Рисунок 18 EcoSpace с установленным PersistenceMapper.

Все готово для того, чтобы на основании построенной модели сгенерировать структуру БД. Следует помнить, что после любого изменения модели, требующего изменений структуры БД, необходимо откомпилировать приложение. Чтобы сгенерировать БД, необходимо выбрать инструмент Create Database Schema в нижней части окна EcoSpace. В дальнейшем, при изменении модели необходимо использовать инструмент Evolve database. В этом случае ECO, по возможности, старается изменить схему базы данных таким образом, чтобы внесенные данные не пропали. Еще одно важное замечание – необходимо стараться не называть сущности и атрибуты именами, которые повторяют зарезервированные слова SQL базы данных. В случае если по каким-то причинам все-таки необходимо использовать такие имена для сущностей, то обязательно нужно изменить свойства TableName для сущности и ColumnName для атрибута сущности.

Если все готово и схема базы данных сгенерировалась успешно, в область сообщений будет выведено сообщение (рисунок 19).

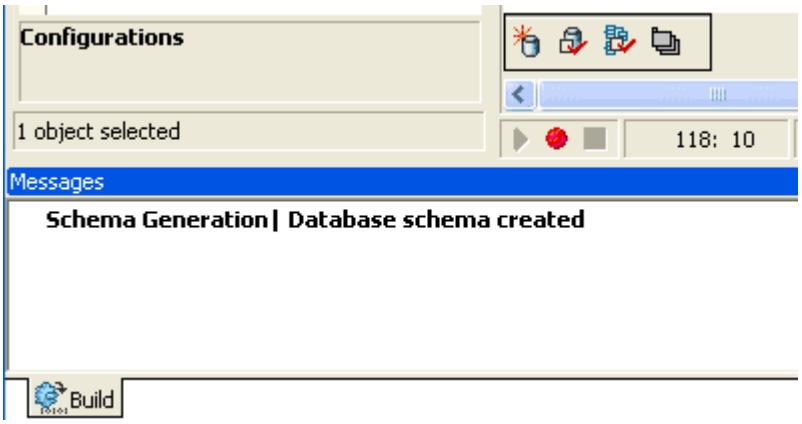

Рисунок 19. Сообщение об успешном создании схемы базы данных.

Построение пользовательского интерфейса

Построение пользовательского интерфейса включает не только собственно пользовательский интерфейс, но и частичную реализацию бизнес-правил.

Построение пользовательского интерфейса в Borland C#Builder

Построение интерфейса в нашем случае означает отображение и редактирование данных приложения. Для простоты мы будем отображать данные, используя компоненты DataGrid. Итак, нам нужно построить интерфейс, в котором будут отображаться отделы, работники отделов, их должности и зарплаты. Главное окно приложения выглядит следующим образом (рисунок 20):

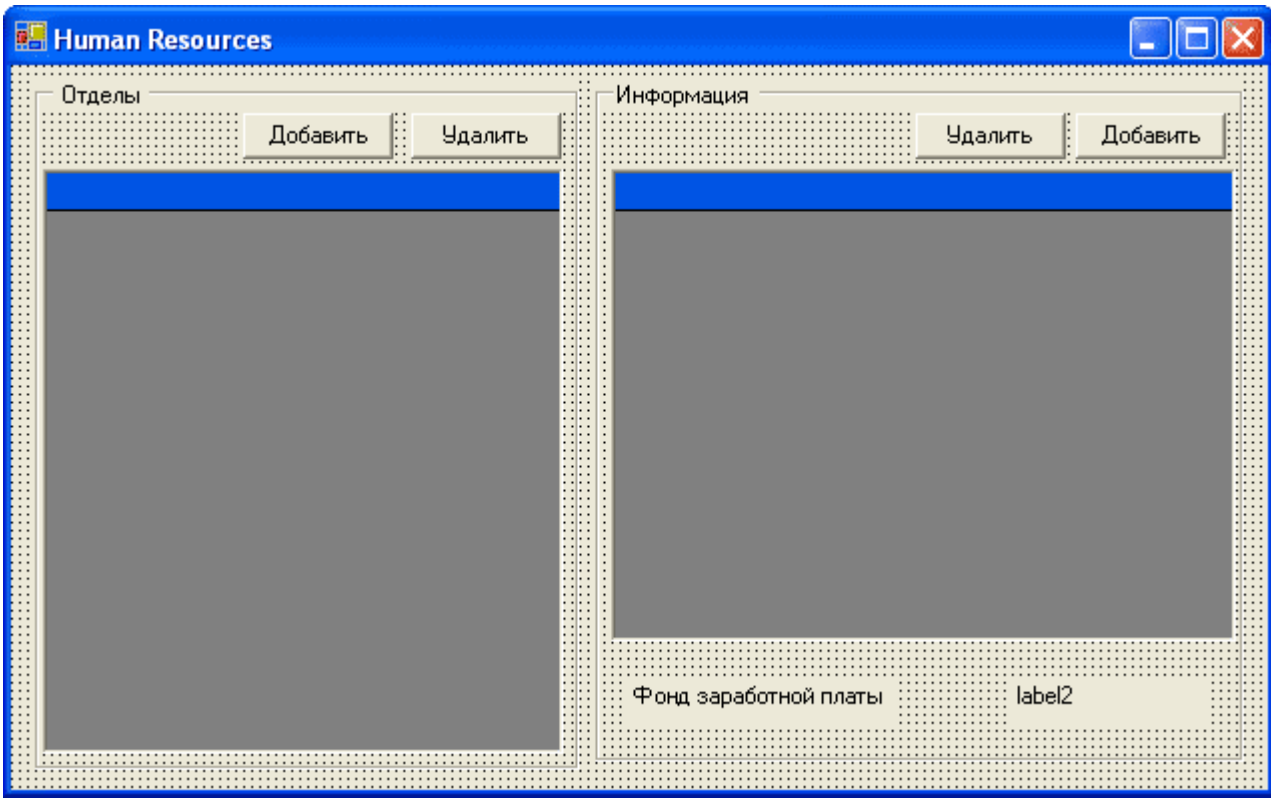

Рисунок 20. Внешний вид главного окна приложения

Связывание данных с элементами управления осуществляется с помощью компонентов ExpressionNandle, которые в данном случае выступают в качестве источника данных. Вообще говоря, ExpressionHandle выполняет OCL-выражение и отдает его результаты в качестве данных. Исходя из архитектуры OCL, связь с моделью должна осуществляться, используя EcoSpace, за что отвечает компонент ReferenceHandle. Итак, структура получения данных из базы данных следующая: Persistence (EcoSpace (ReferenceHandle (ExpressionHandle (визуальный элемент. Разработчику нет необходимости заботиться о том, чтобы на каждой форме присутствовал ReferenceHandle, мастер создания EcoEnabled Form делает это сам. Нам нужно отобразить список отделов и обеспечить добавление и удаление экземпляров класса Department. Для этого на форму помещаем компонент ExpressionHandle, устанавливаем значение свойства RootHandle в rhRoot, далее вызываем OCL-редактор (рисунок 21), чтобы заполнить свойство Expression, где задаем нужное нам выражение. В данном случае это будет Department.allInstances. Данное выражение показывает, что нам нужны все экземпляры класса Department. Далее – просто подставляем в соответствующем гриде ehDepartment в качестве DataSource. После этого в dgDepartments становятся видны все атрибуты сущности Department. Чтобы настроить вид dgDepartments, нужно выполнить стандартную процедуру определения стиля грида, т.е. заполнить TableStyles.

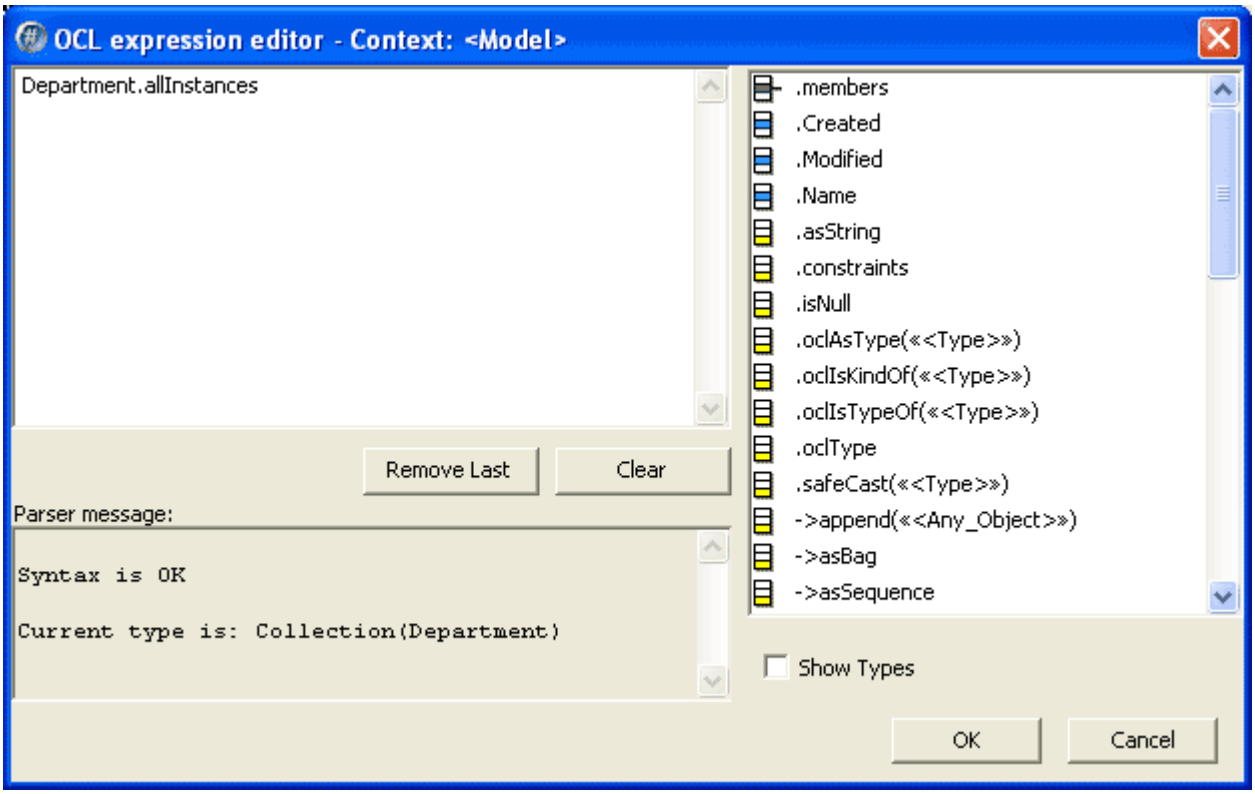

Рисунок 21. Редактор OCL выражений

Теперь нужно реализовать добавление и удаление экземпляров класса Department. Для этого потребуется написать некоторое количество кода. Добавление класса осуществляется очень просто. Этот код мы поместим в метод обработки события нажатия на кнопку btnAddDepartment:

```
private void btnAddDepartment_Click(object sender, System.EventArgs e)
{
  Department dpt = new Department(EcoSpace);
  dpt.Created = DateTime.Now;
}
```

Стоит обратить внимание на то, что при добавлении нового экземпляра Department мы одновременно устанавливаем значение атрибута Created.

Чтобы получить возможность удалять экземпляры Department, нам нужно добавить еще один компонент на форму, впоследствии он понадобится для того, чтобы осуществить связь master-detail между Department и Person. Это компонент CurrencyManagerHandle, назовем его chDepartments. Назначение этого компонента – отслеживание текущего экземпляра класса в визуальном объекте. Для него требуются следующие установки (Таблица 5)

Свойство	Значение
RootHandle	ehDepartment
BindingContext	dgDepartments

Таблица 5

Для удаления элемента Department необходимо написать код обработки события

нажатия на кнопку btnDelDepartment. Вот он:

```
private void btnDelDepartment_Click(object sender, System.EventArgs e)
{
  if (cmDepartment.Element.AsObject != null)
  {
    ((Department)cmDepartment.Element.AsObject).AsIObject().Delete();
  }
}
```

Теперь нужно отобразить информацию о работниках отдела. Для этого опять понадобится ExpressionHandle. Стоит обратить внимание, что, поскольку мы выбираем работников отдела, то в качестве RootHandle мы используем cmDepartments. Устанавливаем значения свойств:

Свойство	***значение***
RootHanlde	cmDepartments
Expression	Members
Columns[1].Name	Name
Columns[1].Expression	Name
Columns[2].Name	Salary
Columns[2].Expression	MemberSalary.Summa
Columns[3].Name	Position
Columns[3].Expression	MemberPosition.Name

Таблица 6.

Необходимо обратить внимание на заполнение свойства Columns, так как необходимо отображать не только атрибуты сущности Person, но и атрибуты связанных сущностей Salary и Position.

Далее указываем ehMembers в качестве DataSource для dgMembers.

Обработка удаления экземпляра Person происходит аналогично удалению Department, но для этого нужно добавить еще один CurrencyManagerHandle, который будет «следить» за экземплярами Person.

```
private void btnDelMember_Click(object sender, System.EventArgs e)
{
if (cmMember.Element.AsObject!=null)
  {
    ((Person)cmMember.Element.AsObject).AsIObject().Delete();
  }
}
```

Добавление нового экземпляра Person можно сделать несколько сложнее, через отдельную форму, где будет можно выбрать зарплату и должность из справочника. Для начала создадим новую Eco Enabled-форму, а в ней - два справочника: зарплат и

должностей, введем туда также строку ввода фамилии, имени и отчества работника. (рисунок 22). Назовем эту форму wfNewPerson.

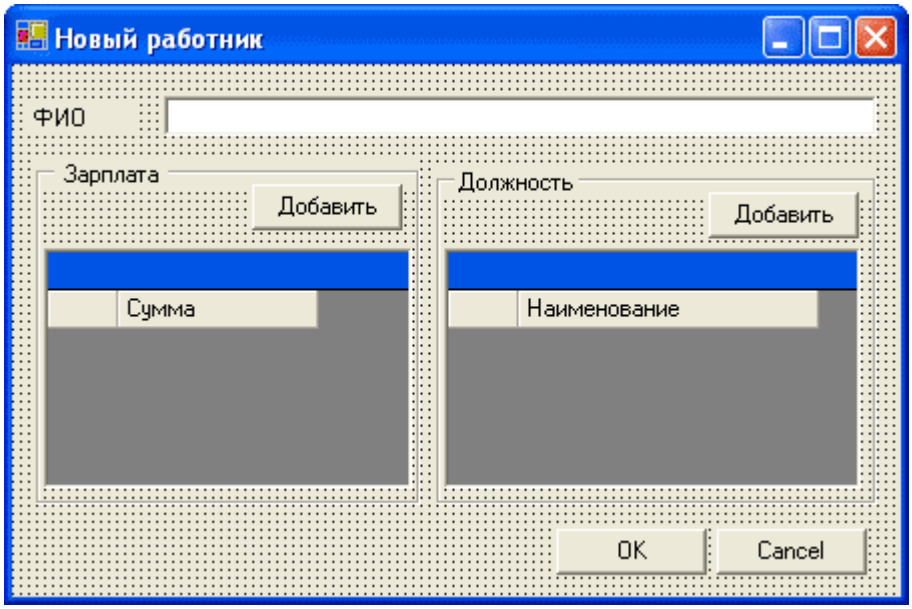

Рисунок 22. Форма добавления нового работника

Используем два ExpressionHandle, два CurrencyManagerHandle и один ReferenceHandle (значения свойств приведены в таблице 7).

Компонент	***Свойство***	***Значение***
ehSalary	RootHandle	rhRoot
	Expression	Salary.allInstances
ehPosition	RootHandle	rhRoot
	Expression	Positions.allInstances
cmSalary	RootHandle	ehSalary
	BindingContext	dgSalary
cmPosition	RootHandle	ehPosition
	BindingContext	dgPosition

Таблица 7

Описываем добавление экземпляров Salary и Position:

```
private void btnAddSalary_Click(object sender, System.EventArgs e)
{
        new Salary(EcoSpace);
}

private void btnAddPosition_Click(object sender, System.EventArgs e)
{
  new Positions(EcoSpace);
}
```

Необходимо добавить три поля в класс формы (это связано с правилами видимости), а также присвоение этим полям значений при нажатии на кнопку OK.

```
public string MemberName;
public Salary slry;
public Positions ps;

private void button2_Click(object sender, System.EventArgs e)
{
        MemberName = textBox1.Text;
  slry = (Salary)cmSalary.Element.AsObject;
  ps = (Positions)cmPosition.Element.AsObject;
}
```

Итак, у нас все готово, чтобы написать код, который добавляет нового работника в отдел. Вот он:

```
private void button3_Click(object sender, System.EventArgs e)
{
        wfNewPerson fNewP = new wfNewPerson(EcoSpace);
        if (fNewP.ShowDialog()==DialogResult.OK)
        {
                Person member = new Person(EcoSpace);
                member.MemberDepartment=(Department)cmDepartment.Element.AsOb
                member.Name = fNewP.MemberName;
                member.MemberSalary = fNewP.slry;
                member.MemberPosition = fNewP.ps;
        }
}
```

Осталось сделать подсчет и отображение фонда заработной платы отдела. Это можно сделать несколькими способами. Мы добавим вычисляемый атрибут в сущность Department. Для этого добавляем атрибут TotalSummary в сущность Department, устанавливаем свойство derived в True, и в свойство DerivationOCL записываем выражение OCL, которое будет вычислять сумму зарплат сотрудников отдела:

```
if members->IsEmpty then 0 else members.MemberSalary.summa->sum endif
```

Модель будет выглядеть так (рис 23)

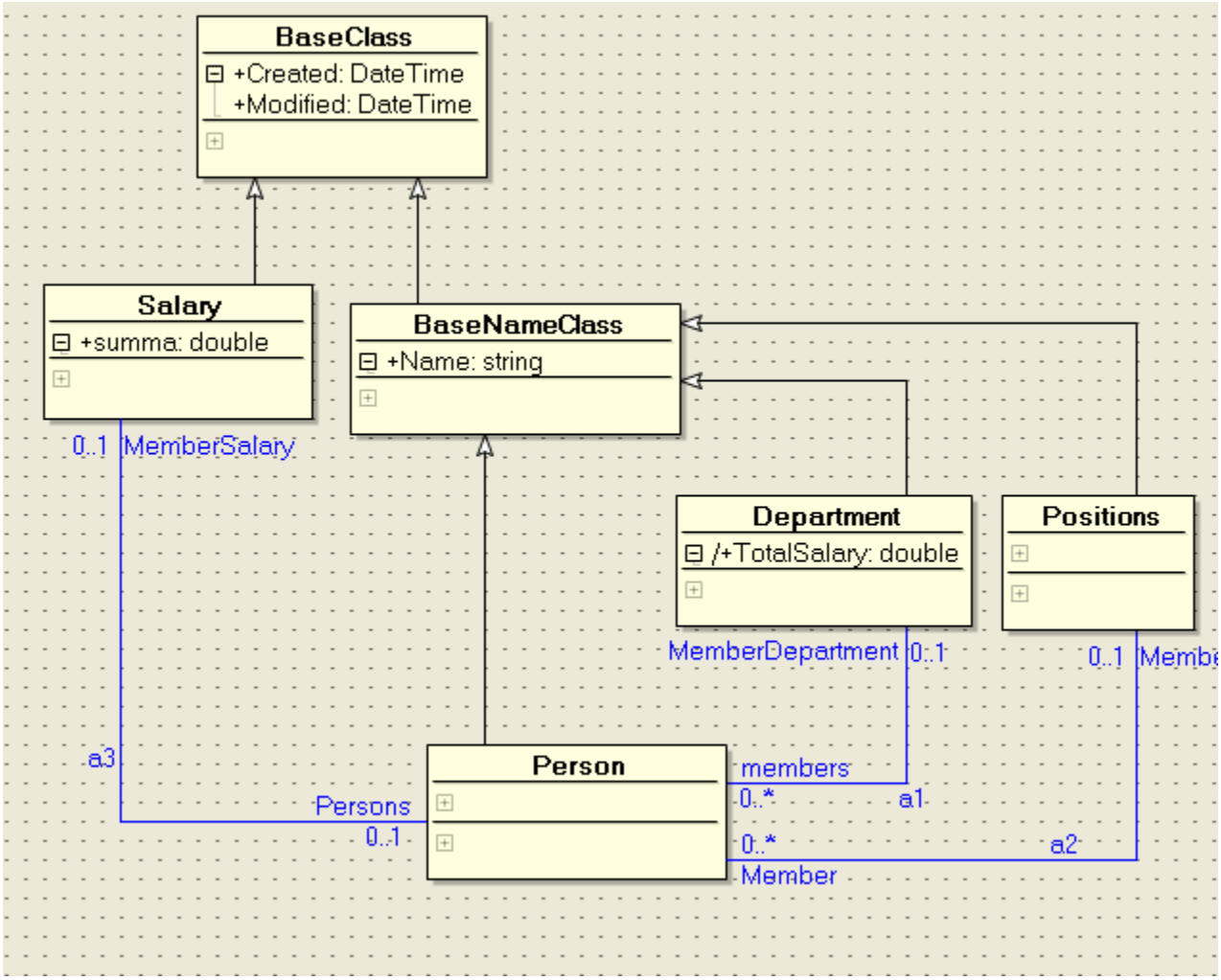

Рисунок 23.

Далее устанавливаем DataBindings.Text у надписи, которая будет отображать сумму зарплат в ehDepartment – TotalSalary.

В принципе приложение готово, но, как можно заметить, введенные данные не сохраняются. Это очень легко поправить. Для этого, например, в обработке события закрытия главного окна, можно вписать код сохранения данных:

```
private void WinForm_Closing(object sender, System.ComponentModel.CancelEvent
{
  EcoSpace.UpdateDatabase();
}
```

На этом можно считать приложение полностью законченным.

Создание пользовательского интерфейса в Delphi 8 for Microsoft.NET

В целом, построение интерфейса пользователя в Delphi осуществляется аналогичным образом. Отличие в том, что становится возможным использование таких компонентов как EcoAutoFormExtender, EcoActionExtender, EcoDragDropExtender, AcoListActionExtender. Применение их еще более упрощает работу разработчика,

освобождая его от написания рутинных конструкций. Все эти компоненты добавляются в форму мастером создания новой EcoEnabled-формы.

Итак. У нас есть модель приложения и сделана привязка к данным. Мы строим точно такую же форму, как и при использовании C#Builde, но не занимаемся созданием кода, который отслеживает текущий объект в каждом DataGrid-е и добавляет/удаляет экземпляры объектов. Мы переложим эти задачи на ECO. Для этого необходимо обратится к компоненту EcoListAction. Изменим его свойства на нужные нам (рисунок 24).

Рисунок 24. Изменение свойств EcoListAction

Теперь мы используем этот компонент, чтобы сообщить кнопкам на форме, какую функцию они будут выполнять. Изменим свойства кнопок (рисунок 25). Мы изменяем Currency manager и EcoListAction, что позволяет нам не писать никакого кода для добавления экземпляра класса Department.

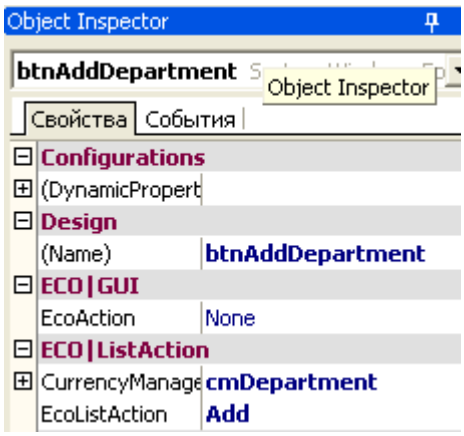

Рисунок 25. Изменение свойств кнопок.

Аналогичным образом поступаем с другими кнопками, кроме кнопки добавления

нового работника – обработку этого действия мы выполним сами, аналогично тому, как это делалось в C#Builder.

```
procedure TWinForm.btnAddMember_Click(sender: System.Object; e: System.EventA
var
  frmNewMember : frmNewPerson;
  member : Person;
begin
  frmNewMember:=frmNewPerson.Create(fEcoSpace);
  if (frmNewMember.ShowDialog=System.Windows.Forms.DialogResult.OK) then
  begin
    member:=Person.Create(fEcoSpace);
    member.Name:=frmNewmember.nm;
    member.MemberSalary:= frmNewMember.slry;
    member.MemberPosition := frmNewMember.pstn;
    member.MemberDepartment := cmDepartment.Element.AsObject as Department;
  end;
end;
```

Распространение приложений ECO

Распространение приложений ECO не отличается от распространения любых .NET приложений. Естественно, для их работы необходим .NET runtime, а также сборки, отвечающие за доступ к данным и за работу ECO. Эти сборки нужно поместить либо рядом с исполняемым файлом приложения, либо в GAC (Global Assembly Cache) той машины, на которой будет работать приложение.

При использовании BDP потребуются сборки, перечисленные в таблице 8.

Используемая СУБД	***Сборки, DLL***	***Назначение***
IBM DB/2	Borland.Data.Common.dll	GAC
	Borland.Data.Db2.dll	GAC
	bdpdb2.dll	System
	Borland.Data.Provider.dll	GAK
Interbase	Borland.Data.Common.dll	GAC
	Borland.Data.Provider.dll	GAC
	Borland.Data.Interbase.dll	GAC
	bdpint.dll	System
MS SQL 2000/MSDE	Borland.Data.Common.dll	GAC
	Borland.Data.Provider.dll	GAC
	Borland.Data.Mssql.dll	GAC
	bdpmss.dll	System
Oracle	Borland.Data.Common.dll	GAC
	Borland.Data.Provider.dll	GAC

	Borland.Data.Oracle.dll
	bdpora.dll

Таблица 8.

Сборки, отвечающие за работу ECO: Borland.Eco.Core.dll, Borland.Eco.Handles.dll, Borland.Eco.Interfaces.dll, Borland.Eco.Ocl.ParserCore.dll, Borland.Eco.Persistence.dll. Эти сборки также лучше поместить в GAC.

Заключение

Данная статья не претендует на всеобъемлющий охват такой большой темы, как ECO, это всего лишь первое вступление в эту интересную тему. Больше материалов по применению ECO можно найти на Borland Developers Network (http://bdn.borland.com) и на сайте C#Builder.ru (http://www.csbuilder.ru). Там же можно найти исходные тексты примеров, рассматриваемых в этой статье. Пример C#Builder (http://www.csbuilder.ru/getfileid.aspx?id=52), пример Delphi (http://www.csbuilder.ru/getfileid.aspx?id=53) .

Эта статья опубликована в журнале RSDN Magazine #3-2004. Информацию о журнале можно найти здесь