УДК 681.3.06

РАСШИРЕНИЕ ВОЗМОЖНОСТЕЙ MICROSOFT VISUAL STUDIO С ИСПОЛЬЗОВАНИЕМ ТЕХНОЛОГИИ DATA DESIGNER EXTESIBILITY (DDEX)

В. А. Голуб, Т. М. Потапова

Воронежский государственный университет

В предлагаемой работе рассматривается новый подход к предоставлению разработчику необходимой функциональности Интегрированной Среды Разработки (IDE). Он заключается в интеграции в IDE пользовательского источника данных. Практическая реализация метода проведена на примере Microsoft Visual Studio.

ВВЕДЕНИЕ

На сегодняшний день Microsoft Visual Studio является исключительно широко распространенным программным продуктом. Впервые общая технология наращивания функциональности Интегрированной Среды Разработки (IDE) Visual Studio Integration and Extensibility (VSIEP) была представлена в Microsoft Visual Studio 2005. Это нововведение позволяет создавать свои расширения IDE, такие как панели управления, меню, окна, плагины, новые типы проектов, редакторы для них, расширять и переопределять существующие компоненты, к примеру, Solution Explorer, Server Explorer, Class Window, Property Window и др.

Одна из наиболее актуальных задач интеграции обусловлена тем, что современные технологии разработки программного обеспечения требуют прямого доступа из Microsoft Visual Studio к БД (для просмотра и модификации объектов, создания новых схем и т.п.). Перспективным направлением является подчасть технологии VSIEP — Data Designer Extensibility (DDEX) — расширение функциональности доступа к данным.

В качестве СУБД, для которой может использоваться прямой доступ из IDE, выбрана СУБД ЛИНТЕР [1], являющаяся единственной системой управления базами данных, которая полностью разработана в России. ЛИНТЕР подходит для работы в условиях ограниченных вычислительных ресурсов, поддерживает работу в операционных системах реального времени, обладает высокой надежностью и отказоустойчивостью. СУБД ЛИНТЕР поддерживает практически все операционные системы, что позволяет создавать на ее основе кроссплатформенные приложения.

Целью работы является рассмотрение вопросов интеграции нового источника данных (СУБД ЛИНТЕР) в интегрированную среду разработки Microsoft Visual Studio 2005—2008. В качестве интерфейса для связи с СУБД используется самый нижний уровень IntLib, что обеспечивает максимальную совместимость и встраиваемость, т.к. вообще говоря разрабатываемый провайдер данных является кроссплатформенным.

1. ТЕХНИЧЕСКАЯ РЕАЛИЗАЦИЯ

Решение поставленной задачи требует разработки и реализации двух провайдеров: так называемого DDEX провайдера (фактически UI) и провайдера данных (провайдер — это библиотека, посредством которой осуществляется доступ от объекта-композита к объекту-части [2]) System.Data.LinterClient аналогичного System.Data.SQLClient или System.Data.OracleClient, что позволило бы обеспечить возможность доступа к СУБД ЛИНТЕР из Microsoft Visual Studio через ADO.NET провайдер.

Разрабатываемый провайдер будет представлять собой связующее звено между встраиваемым элементом и Microsoft Visual Studio. Он должен отвечать следующим требованиям:

— соответствовать базовой спецификации ADO.NET 2.0 (и далее 3.0), т.е. реализовывать Base-Class-Based Provider Model, т.е. классы провайдера должны быть привязаны к существующим в ADO.NET базовым классам-шаблонам с помощью наследования;

— в качестве основного языка программирования должен быть выбран язык C#. Это требование объясняется тем, что Framework 2.0 и 3.0, для которого разрабатывался провайдер, наилучшим образом работает с управляемым кодом. В тех случаях, когда нужно использовать

внутренние функции библиотеки IntLib, написанные на C++, используется механизм маршаллинга.

— инкапсулировать эти классы в специальном классе LinterClientFactory, который в свою очередь наследуется от абстрактного класса DbProviderFactory;

— имя сборки, namespace и invariant name провайдера (то название провайдера, под которым он зарегистрирован в реестре Windows и прописан в конфигурационном системном файле machine.config, и по которому Visual Studio обращается к пользовательскому провайдеру) должны быть одинаковыми. (Вопрос инсталляции провайдера будет более подробно рассмотрен ниже.)

На рис. 1 показана схема функционального взаимодействия источника данных — ядра СУБД (Data Source) — и среды разработки — Microsoft Visual Studio (Host Layer). Это взаимодействие осуществляется с помощью промежуточного слоя — DDEX Layer, — который предоставляет сервисы для доступа к данным, и состоит из двух провайдеров — верхнего, собственно, DDEX провайдера (System.Data.LinterClientDesigner) и нижнего ADO.NET провайдера (System.Data.LinterClient).

В таблице показаны основные базовые классы и интерфейсы для ADO.NET 2.0 [2]. Содержимое таблицы следует понимать следующим образом: базовый класс Base class — это абстрактный класс, описанный в спецификации MSVS, также как и предложенный средой интерфейс Generic interface. В свою очередь, LinterClient class — это реализованный в библиотеке провайдера пользовательский класс.

Собственно, приступим к описанию Base-Class-Based модели построения провайдера.

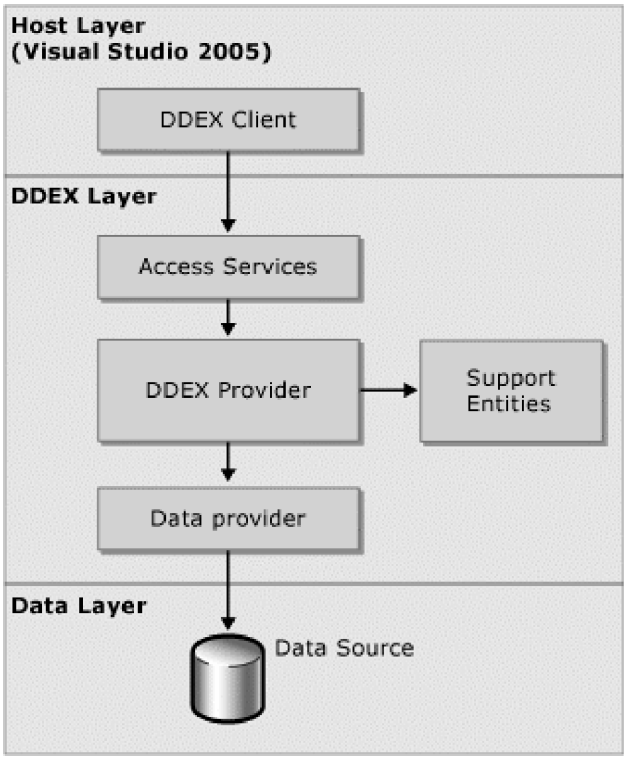

Рис. 1. Интегрирование нового источника данных в MSVS 2005 [2]

LinterClientFactory (наследуется от абстрактного DBProviderFactory) — класс-фабрика (является обязательной в провайдере данных для ADO.NET 2.0 и выше), она инкапсулирует такие классы как:

— LinterDbCommand,
— LinterDbCommandBuilder,
— LinterDbConnection,
— LinterDbConnectionStringBuilder,
— LinterDbDataAdapter,
— LinterDbParameter,
— LinterDbParameterCollection.

Таблица 1

Базовые классы и интерфейсы провайдера данных для ADO.NET 2.0

LinterClient class	Base class	Generic interface
LinterDbConnection	DbConnection	IDbConnection
LinterDbCommand	DbCommand	IDbCommand
LinterDbDataReader	DbDataReader	IDataReader/IDataRecord
LinterDbTransaction	DbTransaction	IDbTransaction
LinterDbParameter	DbParameter	IDbDataParameter
LinterDbParameterCollection	DbParameterCollection	IDataParameterCollection
LinterDbDataAdapter	DbDataAdapter	IDbDataAdapter
LinterDbCommandBuilder	DbCommandBuilder	
LinterDbConnectionStringBuilder	DbConnectionStringBuilder	
LinterDbPermission	DBDataPermission	

1. Класс LinterDbCommand

Экземпляр этого класса представляет собой SQL-запрос в формате Unicode-строки, снабженный методами для выполнения этого запроса и необходимыми свойствами.

На рис. 2 показана UML-диаграмма класса. Рассмотрим подробно основные свойства класса, заявленные в спецификации. Особо следует отметить, что в любой задаче интеграции неотъемлемой частью будет соответствие встраиваемого пакета предложенной абстрактной, интерфейсной или шаблонной модели.

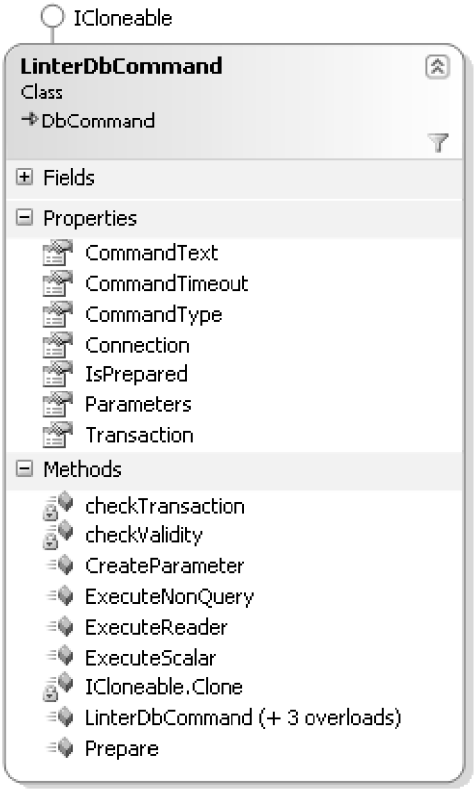

Рис. 2. UML-диаграмма класса LinterDbCommand

Свойство CommandText содержит фактически строку SQL-запроса;

CommandTimeOut — максимальное время ожидания команды;

CommandType — тип команды;

Connection — свойство readonly, содержит LinterDbConnection, соответствующее текущему состоянию подключения;

Parameters содержит коллекцию параметров для команды типа LinterDbParameterCollection;

IsPrepared — подготовлен ли запрос к исполнению (важно, если он содержит параметры, и их надо привязать к буферу SQL-запроса (коррелирует с методом Prepare));

Transaction — текущая транзакция;

Основные методы класса (используется терминология операции языка UML 2.0):

Prepare — метод подготовки команды к фактическому исполнению — выполняется трансляция запроса и привязка параметров. Его можно написать в коде программы в явном виде, как это принято у MSSQL Server, и тогда на этапе выполнения команды (при вызове одного из методов ExecuteReader, ExecuteScalar, ExecuteNonQuery) буферы параметров будут привязаны к буферу запроса, но если этого не произошло, то есть если явно не был вызван метод Prepare, и, соответственно, IsPrepared = false, то он вызовется автоматически.

CreateParameter — создание параметра для запроса. Возвращает объект типа LinterDbParameter. Затем этот параметр можно добавить в свойство Parameters — в коллекцию параметров для команды.

ExecuteReader — используется, если запрос возвращает данные (SELECT или EXECUTE хранимой процедуры, возвращающей значения)

ExecuteScalar — частный случай метода ExecuteReader, возвращает только один кортеж значений, а не множество строк.

ExecuteNonQuery — выполнение команды, которая не возвращает данные (INSERT, UPDATE, DELETE, CREATE TABLE, CREATE VIEW и т.д.)

Поддерживаемые интерфейсы — ICloneable.

Метод ICloaneble.Clone — реализация интерефейса ICloneable. Необходимо для класса LinterDbDataAdapter, который работает с порожденной командой.

2. Класс LinterDbCommandBuilder

Так же, как и остальные классы, является обязательным по спецификации Microsoft Visual Studio, и содержит функционал для автогенерации команд — как правило, для объекта LinterDbDataAdapter.

Его основные методы:

GetDeleteCommand,

GetInsertCommand,

GetUpdateCommand.

Они должны возвращать команды, текст которых сгенерирован автоматически по введенным полям для запроса, значениям и типам команд.

Также важным методом является ApplyParameterInfo — добавление информации об автоматически созданном внешним кодом параметре: его тип с точки зрения СУБД, его точность и количество знаков после запятой. Информацию о структуре класса смотрите на рис. 3.

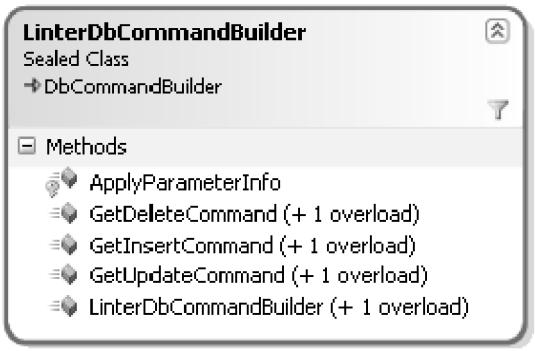

Рис. 3. UML-диаграмма класса LinterDbCommandBuilder

3 Класс LinterDbConnection

Класс представляет объект соединения с СУБД ЛИНТЕР.

Его свойства:

ConnectionString — строка соединения. Содержит Connection Information — логин, пароль, имя сервера и др.

ConnectionTimeOut — предел времени ожидания соединения.

DataBase — имя базы данных

DataSource — источник данных

DbProviderFactory — класс-фабрика провайдера, которая инкапсулирует основные классы.

IsClosed — свойство булевского типа, возвращает true, если состояние соединенения закрыто.

IsTransaction — свойство говорит о том, что запрос выполняется в рамках транзакции.

PersistSInfo — информация о Connection Information — запоминать или нет данные аутентификации и идентификации пользователя.

ServerVersion — версия сервера.

State — состояние соединения (открыто или закрыто).

Методы класса:

Open — открывает соединение, используя введенную строку ConnectionString

Close — закрывает соединение с БД.

BeginTransaction — начинает транзакцию, уровень изоляции (IsolationLevel) выбирается при соединении, по умолчанию — ReadCommited.

CreateCommand — возвращает объект LinterDbCommand, устанавливая в свойстве команды Connection текущее соединение

GetSchema — важный метод для получения метаданных — информация о строение (количество таблиц, внешние, уникальные и первичные ключи, владельцы таблиц, информация о столбцах и их типах и т.д.)

В свою очередь, метод GetSchema — доступная пользователю операция — вызывает частные методы класса, такие как: Schema_Columns, Schema_DataTypes, Schema_Foreignkeys, Schema_Indexes, Schema_Procedures, Schema_Tables, Schema_Triggers, Shema_Users, Schema_Views и др.

UML-диаграмма класса показана на рис. 4. Еще раз отмечу, что в статье описание классов в терминах UML, и потому у метода — операции — не уточняются параметры, но это не значит, что их там нет. Это сделано для того, чтобы не засорять текст избыточными данными, например, информацией о перегруженных методах с различными параметрами.

4. LinterDbConnectionStringBuilder

Как следует из названия класса, его назначение — собирать строку — объект типа LinterDbConnectionString из введенных данных ConnectionInformation.

Из рис. 5 видно, что свойства класса полностью соответствуют аналогичным свойствам класса LinterDbConnection:

идентификатор пользователя;

пароль;

имя базы данных;

имя сервера;

уровень изоляции транзакций;

кодировка вводимых запросов;

время ожидания ответа от ядра при соединении.

5. LinterDbDataAdapter

Это один из важнейших классов провайдера. Осуществление запросов к СУБД возможно с помощью механизма, о котором косвенно уже говорилось ранее: DbCommand-DbDataReader (в случае, если запрос возвращает значения), но при этом мы постоянно при каждой команде обращаемся к базе данных. Или, к примеру, если нужно сделать вставку 3000 записей, то,

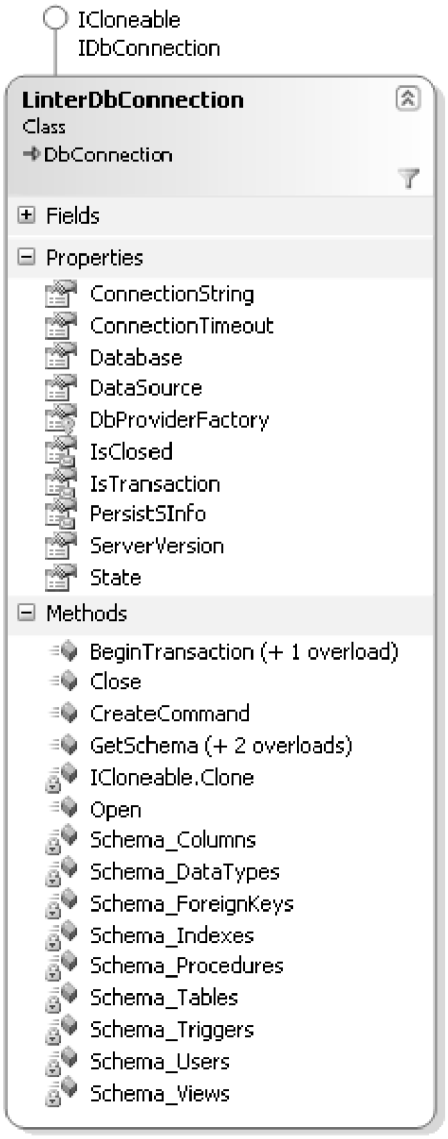

*Рис. 4.*UML-диаграмма класса LinterDbConnection

используя этот метод, мы будем добавлять их по одной. При этом соединение будет открыто всегда. Это дорого, и вообще говоря, не всегда необходимо.

Такой уровень доступа к данным ADO.NET в литературе называется связным [3], [4].

В тех случаях, когда связный уровень доступа не выгоден, или не очень удобен, есть альтернативный метод — несвязный уровень [3], [4] взаимодействия с СУБД. Этот механизм основан на абстрактном классе DbDataAdapter — в нашем случае на его потомке LinterDbDataAdapter.

В отличие от связного уровня, данные, полученные с помощью адаптера DbDataAdapter, не обрабатываются с помощью объекта чтения DBDataReader. Вместо этого для хранения данных между вызывающей стороной и источни-

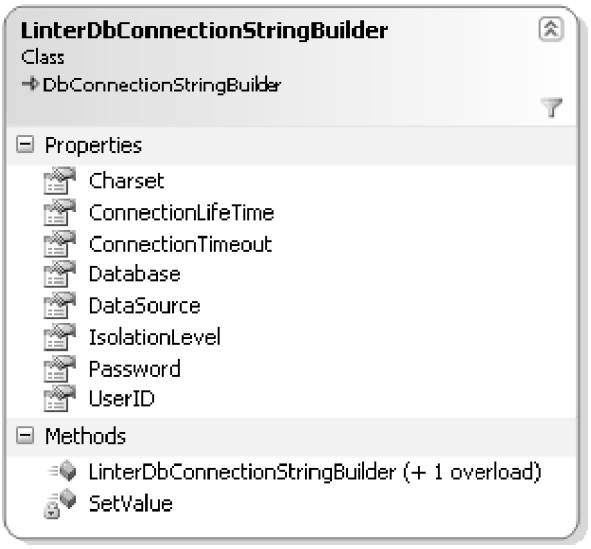

Рис. 5. UML-диаграмма класса LinterDbConnection StringBuilder

ком адаптер данных использует объект DataSet. Тип DataSet — это контейнер, используемый для любого числа объектов DataTable, каждый из которых содержит коллекцию экземпляров типа DataRow и DataColumn.

Объект адаптера данных обрабатывает соединение с базой данных автоматически. С целью расширения возможностей масштабируемости адаптеры данных сохраняют соединение открытым минимально возможное время. Как только вызывающая сторона получает объект DataSet, соединение с СУБД разрывается, и вызывающая сторона остается со своей локальной копией удаленных данных. Вызывающая сторона может вставлять, удалять и модифицировать данные DataTable, но физически база данных не будет обновлена до тех пор, пока вызывающая сторона не передаст явно объект DataSet адаптеру данных для обновления.

Адаптер данных LinterDbDataAdapter работает с четырьмя основными командами , которые хранятся в его свойствах:

DeleteCommand,
InsertCommand,
UpdateCommand,
SelectCommand.

Последняя команда используется в методе Fill. Он является непрегруженным и потому не показан на UML-диаграмме (рис. 6).

Также видно, что у класса есть два обработчика событий — RowUpdating, RowUpdated, которые обеспечивают механизм физического обновления БД.

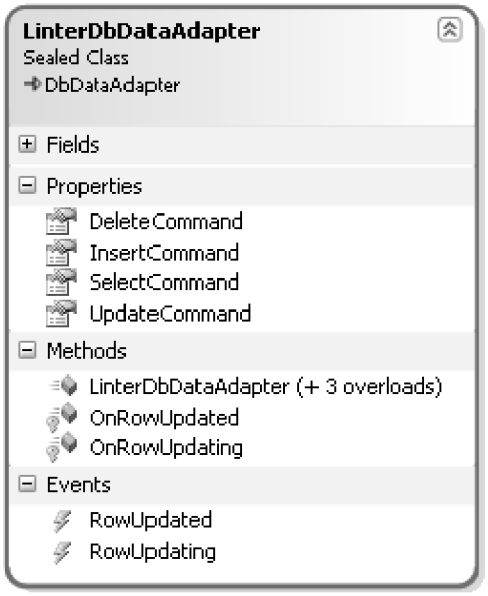

Рис.6. UML-диаграмма класса LinterDbDataAdapter

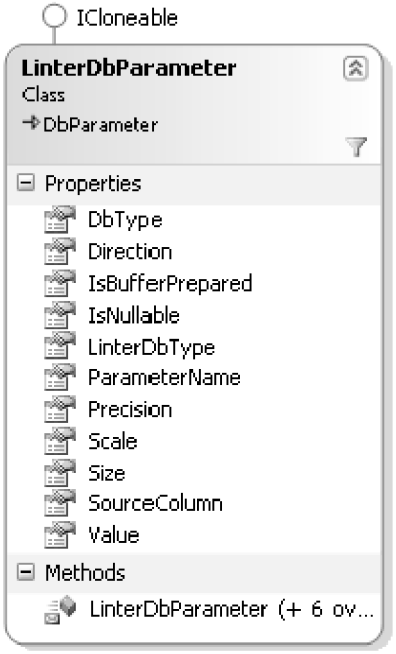

Рис. 7. UML-диаграмма класса LinterDbParameter class

6. LinterDbParameter

Класс просто описывает сущность "параметр запроса". Причем существует еще один класс провайдера — LinterDbParameterCollection, и между ними отношение композиции: LinterDbParameter является частью коллекции параметров.

Свойства класса LinterDbParameter следующие:

DbType — дотнетовский тип параметра,

Direction — направление (входной, выходной или по ссылке),

IsNullable — значение может принимать null,

LinterDbType — тип параметра в терминах СУБД,

ParameterName — имя параметра, если он именован — при задании параметра перед фактическим именем необходимо поставить префикс ":",

Precision, Scale — точность, количество знаков после запятой,

Size — размер параметра в байтах,

SourceColumn — реальная колонка таблицы, которая соответствует параметру,

Value — фактическая величина параметра запроса.

Схема класса LinterDbParameter показана на рис. 7:

7. LinterDbParameterCollection

Этот класс представляет собой список, элементы которого LinterDbParameter, причем сам список реализован посредством интерфейса IList.

Важным методом является Add — добавление параметра в коллекцию.

А также поиск по индексу или по имени параметра в коллекции — IndexOf(перегруженный метод имеет другие параметры).

Инсталляция данного интеграционного пакета, состоящего из двух провайдеров, заслуживает отдельного обсуждения. В действительности был реализован специальный инсталлятор Install.exe, который выполняет следующие функции:

— регистрирует Data Provider System.Data.LinterClient в системном реестре (в разделах DataProviders, Packages, Services, DataSources). Следует отметить, что в свое время это было сложной задачей, т.к. документации по данному вопросу фактически не существовало.

— добавляет в конфигурационный файл machine.config («C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\CONFIG\machine.config») следующие данные:

<system.data>

<DbProviderFactories>

<add name="Linter Data Provider" invariant="System.Data.LinterClient" description=".Net 2.0 Framework Data Provider for Linter" type="System.Data.LinterClient.LinterClientFactory, System.Data.LinterClient, Ver-

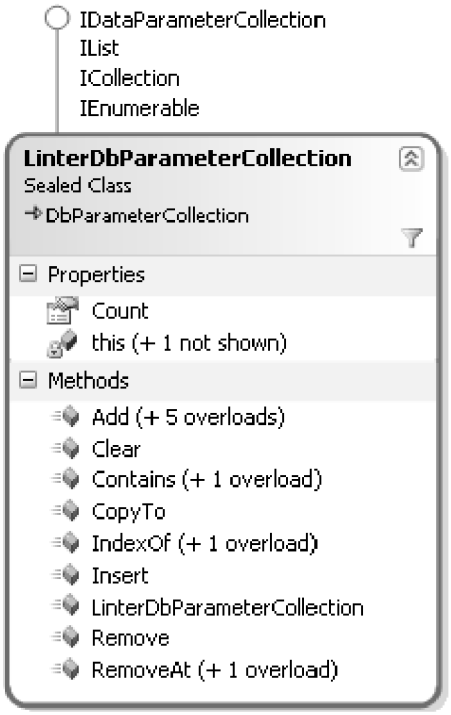

Рис. 8. UML-диаграмма класса LinterDbParameter-Collection

sion=1.0.0.0, Culture=neutral, PublicKeyToken =96657cbf967a2a75" />

</DbProviderFactories>

</system.data>

— инсталлирует в GAC (Global Assembly Cache) сборку System.Data.LinterClient.dll и в тот же каталог дописывает еще 2 библиотеки, которые нужны провайдеру (inter325.dll, dectic32.dll), а также LinterClient.Designer.dll — DDEX провайдер.

2.ТЕСТИРОВАНИЕ

Для того, чтобы выявить проблемы и доработать провайдер по результатам тестов, было разработано тестовое приложение — тестирование производилось в среде NUnit — LinterClientTests, которое состоит из 12 модулей, реализованных посредством прямого вызова основных, несущих доступную пользователю функциональность методов, а также их возможных прогнозируемых комбинаций.

ЗАКЛЮЧЕНИЕ

Разработана архитектура классов провайдера, отвечающая минимальным требованиям спецификации MSVS а также всем особенностям СУБД ЛИНТЕР и отличная от аналогичных архитектур для других СУБД, таких как, например, Oracle или SQLServer.

Предложенный в работе подход к созданию интеграционного пакета для расширения возможностей Microsoft Visual Studio реализован на практике и полностью подтвердил свою работоспособность и эффективность, что позволяет рекомендовать его для использования разработчиками СУБД.

ЛИТЕРАТУРА

1. Он-лайн документация на СУБД ЛИНТЕР — (http://www.linter.ru/ru/main).

2. Visual Studio 2005 Technical Articles " Introduction to Visual Studio Data Designer Extensibility (DDEX)" — (http://msdn2.microsoft.com/en-us/library/ms379576(vs.80).aspx).

3. *Троелсен Э.* Язык программирования C# 2005 и платформа .NET 2.0 / Э. Троелсен; перевод с англ.: — М. Вильямс. 2007, 1167 с.

4. *Сеппа Д.* Программирование на Microsoft ADO. NET 2.0 / Д. Сеппа. — СПб.: Питер. 2007, 784 с.

5. *Малик С.* ADO.NET 2.0 для профессионалов / С. Малик; перевод с англ.: - М. Вильямс. 2006, 560 с.

Статья принята к опубликованию 25 октября 2007 г.