Алексей Дубовцев

bhv

Microsoft® .NET

- Архитектура и внутреннее устройство платформы .NET
- Недокументированные особенности
- Более 1000 примеров на языках C#, VB.NET, MC++, IL

Наиболее полное руководство

+CD

В ПОДЛИННИКЕ®

Алексей Дубовцев

Microsoft® .NET в подлиннике

Санкт-Петербург
«БХВ-Петербург»
2004

УДК 681.3.06
ББК 32.973.018.2
Д79

Дубовцев А. В.

Д79 Microsoft .NET в подлиннике / Под ред. В. Е. Пышкина. — СПб.: БХВ-Петербург, 2004. — 704 с.: ил.

ISBN 5-94157-478-9

Рассмотрены теоретические основы и практические приемы программирования на платформе .NET с использованием популярных языков C#, VB.NET, MC++, IL. Описаны метаданные, общая система типов, сборки, архитектура доменов, атрибуты и др. На большом количестве простых и понятных примеров рассмотрены обработка исключений, делегаты и события, потоки и др. Дано подробное представление низкоуровневого взаимодействия с операционной системой из среды .NET. Прилагается компакт-диск, содержащий большое количество примеров на языках C#, VB.NET, MC++, IL.

Для программистов

УДК 681.3.06
ББК 32.973.018.2

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. гл. редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Василий Плевалов*
Компьютерная верстка	*Натальи Караваевой*
Корректор	*Виктория Пиотровская*
Дизайн обложки	*Игоря Цырульникова*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 30.08.04.
Формат 70×100$^{1}/_{16}$. Печать офсетная. Усл. печ. л. 56,76.
Тираж 3000 экз. Заказ №
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Гигиеническое заключение на продукцию, товар № 77.99.02.953.Д.001537.03.02
от 13.03.2002 г. выдано Департаментом ГСЭН Минздрава России.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 5-94157-478-9

Содержание

Моим родителям

Речь верная красивой не бывает,
Красивые слова не убеждают.

Достоинство себя не в споре выражает,
Кто спорит — истиной не обладает.

Познанье истины — не есть образование,
А сумма сведений — еще не знание.

Кто тупо копит знания, ума не проявляет.
Мудрец, чем больше отдаёт, тем больше обретает.

Как Дао Неба — миру жизнь давать, не принося вреда,
Так Дао Мудрого — творить и не вступать в дебаты никогда.

Лао-цзы, "Дао де Цзин"

Мои искренние Благодарности

Воробьеву Дмитрию Анатольевичу — спасибо за то, что научили думать и указали путь.

Темнову Дмитрию Эдуардовичу — моему первому учителю, плюс как человеку, руками которого была написана моя первая программа — (print "Hello, World").

Силанову Виктору Алексеевичу — учителю, который приложил много личных усилий, для того чтобы сделать из меня профессионала.

Плещенкову Николаю Мироновичу — моему любимому школьному учителю.

Всей команде rsdn.ru — сайта, для которого я написал свою первую статью и с которого началась моя авторская работа.

Рыбакову Евгению — за то, что сподвигнул меня написать эту книгу.

Шишигину Игорю — за внимательность и понимание при подготовке книги.

Пышкину Евгению Валерьевичу — за отзывчивость и неоценимую профессиональную помощь.

Ложечкину Александру — за доверие и прекрасную аннотацию.

Шостенко Ирочке — за неоценимые советы по стилистике русского языка.

Романову Андрею и **Голубеву Дмитрию** — за те советы и замечания, которые они давали при подготовке книги.

Нанобашвили Вячеславу — за мудрые и жизненные советы.

Всему издательскому дому **BHV**, за понимание и веру.

Всему **преподавательскому составу** кафедры физической электроники факультета радиофизики Санкт-Петербургского государственного политехнического университета — за понимание и высококлассное профессиональное преподавание. В особенности Фотиади Александру Эпоминондовичу и Кораблеву Вадиму Васильевичу.

Особые благодарности: **Богачеву Филиппу**, **Титову Алексею**, **Вадиму Георгиевичу**, **Бучину Сереже**, **Ширину Мише, Лыкосову Сергею** за те инструменты и знания, которые они передали в мои руки.

Плюс теплые приветы **Диме Найдичу**, **Славе Ардентову**, **Ступину Вите**, **Антону Зеленскому**, **Саше Волкову**, **Коле Воробьеву** и всем остальным.

И наконец, привет моим **друзьям** и спасибо за то, что вы у меня есть:

Захаренкову Мише, Фазлееву Диме, Василевичу Диме, Шиповникову Андрею, Клищенко Саше, Сучкову Жоре, Сане Иванову, Афониной Ольге, Мартюшину Алексею, Кузнецову Андрею, Андрееву Юре.

Моим любимым **родственникам**:

Дубовцевым Валерию Федоровичу и **Вере Васильевне** — моим любимым дедушке и бабушке.

Топорской Оксане — моей любимой тетке.

Топорскому Сереже — моему лучшему дяде, который многому меня научил, всегда был внимателен и добр.

Топорской Сашке — самой моей любимой сестре на свете.

Селиверстову Игорю и всей его дружной семье — за любовь и понимание.

Селиверстову Сергею — за то, что научил работать с литературой, был добр и терпелив.

Персональный привет всей моей **Нижегородской семье**, **особенно тете Лизе**.

❤ **Дианке**...

Предисловие научного редактора

Перед читателем — написанная молодым автором книга, обобщающая опыт изучения им архитектуры и концептуальных элементов платформы Microsoft .NET Frameworks, обсуждение и использование которой стало едва ли не самым заметным явлением последних лет в области маркетинга средств разработки программного обеспечения. Следуя общеизвестному тезису "чтобы хорошо изучить какой-либо предмет или явление, напишите об этом книгу", автор провел впечатляющую работу по исследованию внутренних механизмов .NET, обеспечивающих такие решения, как поддержка компонентного программирования средствами языка C#, совместимость компонентов, разработанных на разных языках программирования, построение устойчивого, переносимого и безопасного кода, механизмы распространения кода с использованием сборок, организация взаимодействия управляемого и неуправляемого кода и т. д. С точки зрения содержания книги, весьма удачным представляется выбранное для книги название.

Несмотря на то, что книга посвящена в первую очередь рассмотрению аспектов *реализации* механизмов и концепций, лежащих в основе платформы .NET, и поэтому ориентирована на подготовленного читателя, отдельные разделы книги будут интересны и начинающим программистам. В частности, заслуживает быть отмеченным обсуждение фундаментального понятия "тип". Его подробное рассмотрение, вероятно, несколько выходит за рамки основной тематики книги, но при этом представляет собой весьма удачный пример понятного и наглядного объяснения трудных для понимания начинающих программистов концепций программирования. Примерами разделов, написанных на стыке общего знания и изучения механизмов и концепций .NET, могут также служить главы "Потоки" и "Архитектура доменов", от чтения которых удовольствие должны получить и опытные разработчики, и начинающие программисты.

В первой части книги обсуждаются фундаментальные концепции, лежащие в основе программирования на платформе .NET. В частности, подробно анализируются особенности исполнения управляемого кода, назначение

и реализация метаданных, используемых в ходе работы с управляемым кодом, организация и принципы работы исполняющей системы .NET. Подробно исследуются принципы построения общей системы типов и механизмы обеспечения межъязыковой интеграции в приложениях .NET.

Во второй части книги особое внимание автор уделяет новаторским элементам программирования, реализованным в рамках платформы .NET, таким как реализация механизма косвенных вызовов посредством делегатов; поддержка модели программирования, ориентированного на события встроенными средствами языка C#; реализация механизма использования информации времени разработки в ходе исполнения программы посредством специализированных атрибутов.

В третьей части детально обсуждаются аспекты работы с .NET в связи с управлением приложениями и механизмы реализации взаимодействия компонентов .NET. Особого внимания заслуживает изложение решений, лежащих в основе обеспечения взаимодействия с неуправляемым кодом.

Несмотря на то, что обозреваемая книга является первым столь объемным трудом автора, он предстает перед нами сложившимся специалистом со своим взглядом на вещи, открытым для конструктивного и благожелательного обсуждения отдельных разделов проделанной им большой работы. Неформальный и слегка ироничный стиль автора удачно оттеняет сложные для чтения разделы. В тех случаях, когда, по моему мнению, тот или иной момент заслуживал немного более строгого и "сухого" замечания, по согласию с автором и издательством, в текст добавлялось примечание научного редактора. Впрочем, во всей книге их всего около десятка.

Итак, вниманию читателя представляется достойное издание, которое, надеюсь, принесет заслуженную популярность его автору и станет удачным приобретением для читателей.

Пышкин Е. В.,
кандидат технических наук,
доцент кафедры автоматики
и вычислительной техники
Санкт-Петербургского государственного
политехнического университета

Глава 1

Общеязыковая среда исполнения

Общеязыковая среда исполнения .NET — новая мощная среда разработки от Microsoft, обеспечивающая уникальные возможности по межпрограммной и межплатформенной интеграции программных средств. Для ее реализации была разработана единая концепция использования управляемого кода. В главе приводятся сведения об основных принципах построения платформы .NET, рассматриваются внутреннее устройство среды исполнения, общие принципы работы виртуальных машин, а также природа и устройство управляемого кода. Прочитав эту главу, вы узнаете, для чего создавалась среда исполнения и какие задачи она способна решить.

1.1. Архитектура среды исполнения .NET

Общеязыковая среда исполнения (Common Language Runtime, CLR) — это общая независимая виртуальная программная среда для приложений, обеспечивающая эффективное и контролируемое исполнение пользовательского кода, а также предоставляющая разработчикам программных систем широкий спектр разнообразных сервисов. Она существенно облегчает разработку современных приложений, сочетая в себе многие преимущества технологии RAD (Rapid Application Development, Быстрая Разработка Приложений), с высокой производительностью и скоростью работы конечных приложений. Структурно общеязыковая среда исполнения .NET состоит из двух компонентов: виртуальной машины и общей библиотеки классов (рис. 1.1).

Виртуальная машина отвечает за исполнение пользовательского кода, а общая библиотека классов предоставляет разработчикам четко стандартизированный набор типов, предоставляющих самые разнообразные сервисы. Исходный язык программирования, на котором написано приложение для среды .NET, не имеет значения; от транслятора языка лишь требуется умение генерировать "управляемый" код, который понимает виртуальная машина .NET. Такой код работает только в среде CLR, что обеспечивает полный контроль над всеми процессами при исполнении приложения.

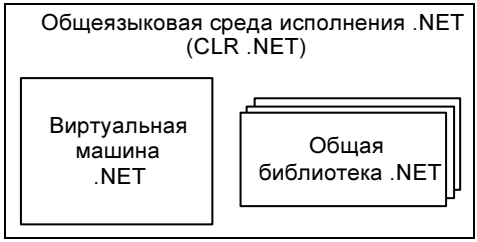

Рис. 1.1. Состав общеязыковой среды исполнения

Управляемый код — новое понятие, введенное разработчиками Microsoft, и которое поначалу вызывает затруднения для понимания у большинства программистов. Его особенностью является то, что он выполняется под постоянным надзором среды исполнения, которая контролирует все его шаги и опекает, как чуткий родитель своего ребенка. Именно из-за столь сильного контроля и надзора было решено назвать код управляемым. Он представляется в специальном формате, называемом .NET байт-код. Для визуального представления внутреннего .NET байт-кода разработан язык IL (Intermediate Language, Промежуточный Язык Microsoft). По сути дела, он находится на одном уровне с .NET байт-кодом, но это не сам байт-код, а лишь его текстовое представление. Для сравнения можно сказать, что IL и .NET байт-код соотносятся, как ассемблер и машинный код. Таким образом, все приложения поставляются для исполнения на виртуальной машине .NET в виде унифицированного .NET байт-кода.

Примечание научного редактора

По прочтении предшествующего текста у читателя может сложиться впечатление, что концепция управляемого кода как таковая является изобретением Microsoft. Это, разумеется, не так (хотя, пожалуй, стоит согласиться, что более или менее новым является сам термин "управляемый код").

Поэтому следует заметить, что сама идея создания кода, ориентированного на исполнение в рамках специальной среды, обусловлена необходимостью создания систем программирования, ориентированных на создание мобильного безопасного устойчивого кода. При этом под мобильностью кода понимается возможность выполнения одной и той же программы на разных вычислительных платформах. Под безопасностью кода понимается встроенная в язык защита от несанкционированного доступа к ресурсам компьютера, на котором исполняется приложение (в особенности, когда речь идет о сетевых приложениях). Устойчивость кода предполагает возможность обнаружения ошибок на возможно более ранних стадиях проектирования и исключение определенных классов ошибок (например, ошибок управления памятью и необрабатываемых исключительных ситуаций). Устойчивость кода предполагает также по возможности наиболее полный контроль корректности операций, совершаемых в процессе исполнения программы (преобразования типов, инициализация объектов, исключительные ситуации в связи с арифметическими вычислениями и т. д.).

Основной механизм, позволяющий обеспечить получение кода, обладающего такими свойствами, основывается на реализации специальной исполняющей системы, работающей не с кодом процессора, а с некоторым промежуточным кодом, который и является выходом компиляции исходного текста программы. Впечатляющий успех применения подобной организации обработки программы выпал на долю технологии Java. В основе решения проблем построения мобильного, безопасного и устойчивого программного обеспечения с помощью Java лежит трансляция исходного кода в промежуточное представление (байт-код) и его выполнение специальной исполнительной системой, называемой виртуальной машиной Java (Java Virtual Machine – JVM).

Код, разрабатываемый на языке, ориентированном на исполнение в контролирующей среде, называют управляемым, или контролируемым, кодом (managed code). Итак, по существу байт-код Java является примером управляемого кода. В этой терминологии, C/C++ генерирует неуправляемый код (unmanaged code), то есть код, ориентированный непосредственно на процессор.

Для создания низкоуровневого байт-кода из высокоуровневого языка, используются соответствующие компиляторы. Создавать приложения .NET возможно более чем на десяти различных языках программирования высокого уровня. При этом общие стандарты среды исполнения .NET, в лице общеязыковой спецификации CLS (Common Language Specification) и общей системы типов CTS (Common Type System), будут гарантировать интеграцию кода вне зависимости от исходного языка, использованного для создания приложения.

Интеграция кода подразумевает полную совместимость кода на всех уровнях: начиная от стандартов определения типов и заканчивая моделью поддержки асинхронных событий. К примеру, вы можете определить класс при помощи языка C#, а унаследовать его, используя Visual Basic .NET (VB .NET).

Особенностью системы .NET является использование оригинальной технологии интеграции кода, которая обеспечивает совместимость кода не на уровне исполняющего ядра (процессора, виртуальной машины .NET), а на уровне самой программной модели. Ранее, когда создавался код для процессоров с использованием ортодоксальных компиляторов, основной задачей программистов было обеспечение лишь односторонней совместимости с самим процессором, разработчики которого вводили собственные стандарты на машинный код, практически не думая о пользователях языков высокого уровня. В результате, сформированный машинный код представлял всего лишь последовательный набор команд для процессора, который предписывал ему совершить определенные действия.

.NET байт-код представляет собой не просто набор команд для виртуальной машины .NET. Он качественно отличается от машинного кода наличием полного описания внутренней структуры программы, начиная от локальной переменной и заканчивая классами и пространствами имен. Фактически, при классической компиляции в машинный код полностью теряется информация о внутреннем устройстве программы, которая по большому счету

и не нужна для ее исполнения. Зато такая информация жизненно необходима при решении задач межъязыковой и межпрограммной интеграции. При компиляции в управляемый .NET байт-код потерь этой информации не происходит и полное представление о внутренней организации программы сохраняется в специальных структурах, называемых метаданными.

Таким образом, технология управляемого байт-кода дает огромные преимущества в виде межъязыковой интеграции и кросс-платформенной переносимости кода. Но все преимущества подхода могут быть сведены к минимуму низкими показателями производительности из-за полной виртуализации исполнения байт-кода. Практическим подтверждением тому могут служить Java (классическая машина от Sun) и Microsoft Visual Basic .NET (ранних версий). Производительность при выполнении кода обеих систем оставляет желать много лучшего.

Существует два вида виртуальных машин — интерпретирующие и компилирующие. Интерпретирующие машины исполняют код, полностью эмулируя его работу. Фактически, они сами исполняют код вверенных им приложений. То есть на каждую инструкцию интерпретируемого байт-кода приходится N инструкций интерпретатора виртуальной машины. В итоге, происходит значительное снижение скорости выполнения программы. За что и не любят виртуальные машины такого типа.

Платформа .NET является виртуальной машиной принципиально другого типа. Это компилирующая виртуальная машина нового поколения. Подобные виртуальные машины не занимаются интерпретацией байт-кода, они компилируют его в машинный код платформы, на которой в данный момент происходит исполнение приложения (и всей виртуальной машины). При подобном подходе мы получаем на одну инструкцию байт-кода соответствующее количество машинных инструкций. Причем эти машинные инструкции будут соответствовать непосредственно исполняющейся программе, а не заранее скомпилированному коду виртуальной машины. При таком подходе конечный машинный код, исполняющийся на процессоре, получается более компактным и быстродействующим, чем обобщенный код интерпретирующей машины.

Примечание

Технологии компилируемых виртуальных машин разрабатывались и в России, в рамках проекта вычислительной системы "Эльбрус". В 2001 г. в статье Бориса Бабаяна "Основные Принципы Архитектуры E2K" **(http://www.mcst.ru/mcst/e2k.pdf)** подробно были описаны принципы функционирования такой технологии и рассмотрены преимущества ее применения. Команда Бабаяна разработала двоичный компилятор, который позволил запускать Windows и игровую программу Microsoft Flight Simulator на платформе Sun с весьма приличными показателями по скорости. Изучая виртуальную машину Microsoft .NET, никак не избавиться от ощущения того, что она построена по принципам, разработанным группой Бабаяна.

Для реализации подобного подхода в виртуальной машине требуется наличие очень важного компонента — компилятора времени исполнения JIT (Just-In-Time Compiler), который обеспечивает трансляцию .NET байт-кода в машинный прямо по ходу исполнения программы. Такие компиляторы обычно называют двоичными. По всем тестам на производительность, проведенным для платформы .NET, она практически ни в чем не уступает классическим компилирующим системам, а по некоторым — даже превосходит их. С подробными отчетами этих тестов можно ознакомиться на сайте **http://www. rsdn.ru**.

Среда исполнения .NET может базироваться на различных аппаратных (Intel, Pocket PC) и программных платформах (Windows, FreeBSD, MacOS, Linux). Для каждой поддерживаемой аппаратной и программной платформы должна существовать собственная реализация подобного компилятора и для каждой новой аппаратной платформы разработчикам придется создавать собственный JIT-компилятор, который будет переводить универсальный независимый .NET байт-код в конкретный машинный код. В этом заключается, пожалуй, основной недостаток этой технологии.

В общем виде внутреннее устройство виртуальной машины .NET можно представить следующим образом (рис. 1.2).

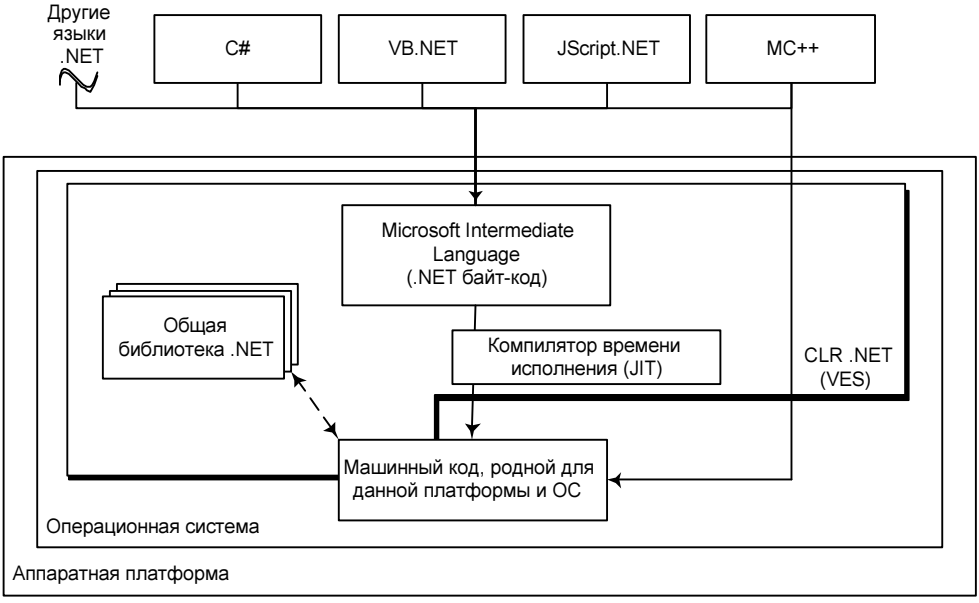

Рис. 1.2. Общая схема виртуальной исполняющей машины .NET

Все это вкупе носит название виртуальной исполняющей машины (VES, Virtual Execution System). Впоследствии, перед выходом платформы .NET "в свет" она была переименована в CLR. На самом верху системы находятся

языки высокого уровня. Программы, написанные на них, транслируются в байт-код платформы .NET, который может быть представлен в виде IL. Он же, во время исполнения программы, компилируется в машинный код аппаратной платформы. Но даже, несмотря на то, что исполнение программ происходит в чистом машинном коде, оно полностью контролируется со стороны виртуальной машины .NET. Достигается это за счет использования служебного контролирующего кода.

В итоге, мы получаем полное абстрагирование от уровня аппаратных и программных платформ. И, как следствие, полную унификацию предоставляемых сервисов, независимо от целевой платформы. К примеру, если предполагается в приложении .NET использовать потоки, то нас абсолютно не будут волновать детали реализации механизмов поддержки многопоточности в операционных системах, в среде которых будет исполняться наше приложение. С точки зрения разработчика, оно будет одинаково исполняться как на Windows системах, так и на FreeBSD.

Примечание

В действительности полной унификации предоставляемых сервисов для всех программных платформ, к сожалению, не наблюдается. Многие необходимые сервисы доступны только на платформах семейства Windows. Для примера, можно привести типы из пространства Windows.Forms, предназначенные для создания пользовательского интерфейса. Но это можно отнести скорее к нюансам маркетинговой политики компании, чем к принципиальным конструктивным недоработкам. Хотя необходимо отметить, что существуют реализации недостающих сервисов от сторонних разработчиков.

Таким образом, для программ .NET создается свой закрытый виртуальный мир, заглянуть за пределы которого им не разрешается. Однако нередко появляется необходимость доступа к специфичным программным и аппаратным ресурсам, которые могут быть не отражены в общей библиотеке классов .NET. Для разрешения этой проблемы разработчики .NET создали технологию исполнения совместного кода в рамках компилятора Managed Extensions for C++ (MC++, Управляемые Расширения для C++). Компилятор MC++ позволяет в одном файле комбинировать как управляемый, так и неуправляемый код. Задача выбора типа генерируемого кода возлагается полностью на программиста. Технология работы компилятора показана на рис. 1.3.

В листинге 1.1 приведен пример интеграции управляемого и неуправляемого кодов в рамках одной программы.

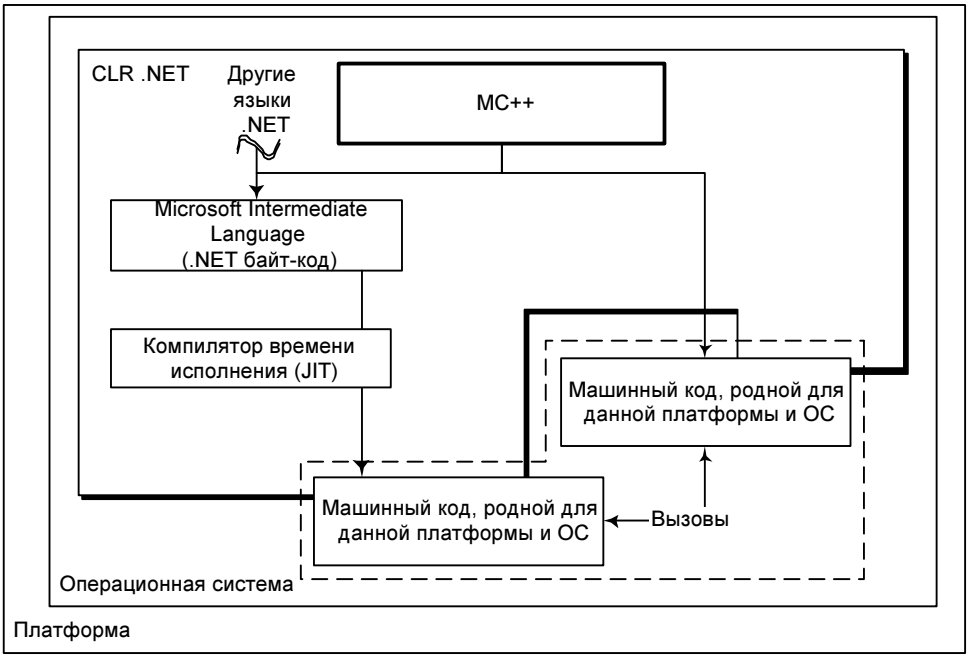

Рис. 1.3. Технология совместного кода, воплощенная в MC++

Листинг 1.1. Симбиоз управляемого и неуправляемого кода

```
/*
  Листинг 1.1
  File:   Some.cpp
  Author: Дубовцев Алексей
*/
// Подключим общую библиотеку классов .NET.
#using <mscorlib.dll>
// Подключим стандартную библиотеку.
#include <stdio.h>
// Директива компилятору на генерацию неуправляемого кода.
#pragma unmanaged
// Эта функция будет неуправляемой, она будет транслирована
// в чистый машинный код.
void UnmanagedFunction()
{
  printf("Hello, World, from UnmanadegFunction!");
```

```
}
// Прикажем компилятору вновь генерировать управляемый код.
#pragma managed
// Это управляемая функция, она будет транслирована в набор инструкций
// .NET байт-кода.
void ManagedFunction()
{
  // Прикажем компилятору использовать управляемую строку.
  System::Console::WriteLine(S"Hello, World, from ManagedFunction!");
  UnmanagedFunction();
}
// Эта функция также будет управляемой.
void main()
{
  ManagedFunction();
}
```

Такой тесный симбиоз управляемого и неуправляемого кода обеспечивает сочетание в одном приложении мощных и удобных средств .NET и наработанных ранее проектных решений. Помимо прямого взаимодействия с операционной системой из неуправляемого кода, среда исполнения предоставляет систему шлюзов, которые позволяют прямо из управляемого кода обращаться к сервисам операционной системы. При этом все обращения на нижний уровень будут контролироваться со стороны системы политики безопасности среды .NET, чего не скажешь о прямом использовании низкоуровневого платформенного кода, который потенциально может творить "что его душе угодно".

Для решения столь серьезных задач среда исполнения предоставляет разработчикам множество инструментов, важнейшими из которых являются:

- система автоматического управления памятью;
- прозрачная межъязыковая интеграция;
- автоматическое управление кодом на основе сборок;
- прозрачная межплатформенная совместимость;
- защита исполняемого кода;
- богатая, мощная и оптимизированная библиотека классов;
- сервисы взаимодействия с операционной системой;
- система автоматического контроля целостности кода;
- автоматическая система защиты переполнения буфера.

1.2. Общая библиотека классов

Общая библиотека классов .NET от Microsoft (Framework Class Library, FCL) мощна и многообразна и предоставляет пользователю огромный спектр сервисов. Перечислить их полностью не представляется возможным, приведем лишь некоторые из них (табл. 1.1).

Таблица 1.1. *Набор основных сервисов, предоставляемых через общую библиотеку классов*

Пространство Имен	Сервисы класса
`System`	Базовые типы, используемые всеми приложениями
`System.Collections`	Коллекции для хранения наборов объектов
`System.Diagnostics`	Сервисы для разработки и отладки приложений
`System.Drawing`	Сервисы для работы с двумерной графикой
`System.EnterpriseServices`	Поддержка транзакций, очередей компонентов, пулов объектов, JIT-активации и других серверных функций
`System.Globalization`	Поддержка региональных стандартов и системы локализации приложений
`System.IO`	Поддержка потокового ввода вывода, в том числе и файлового
`System.Managment`	Сервисы управления компьютерами при помощи WMI (Windows Management Instrumentation, ...)
`System.Net`	Поддержка сетевого взаимодействия
`System.Reflection`	Сервисы отражения, предоставляющие возможность динамического взаимодействия с программным кодом
`System.Resources`	Сервисы для работы с ресурсами
`System.Runtime.InteropServices`	Сервисы, предоставляющие шлюзы для взаимодействия с операционной системой
`System.Runtime.Remoting`	Сервисы, обеспечивающие удаленный доступ к типам
`System.Runtime.Serialization`	Сервисы, обеспечивающие сериализацию (упаковку) объектов в универсальные потоки данных, с возможностью их последующего восстановления

Таблица 1.1 *(окончание)*

Пространство Имен	Сервисы класса
`System.Security`	Сервисы поддержки защиты и безопасности программного кода
`System.Text`	Сервисы для работы с текстом
`System.Threading`	Сервисы поддержки многопоточного кода
`System.Xml`	Сервисы для работы с данными в формате XML
`System.Web.Services`	Средства, предназначенные для создания Web-сервисов XML
`System.Web.UI`	Сервисы для удобного создания Web-пользовательских интерфейсов (Web Forms)
`System.Windows.Forms`	Сервисы поддержки классических пользовательских интерфейсов
`System.ServiceProcess`	Средства поддержки работы с сервисами операционной системы, ответственные за взаимодействие со SCM (Service Control Manager, Менеджер управления сервисами)

Даже при первом рассмотрении набор предоставляемых возможностей впечатляет. А когда начинаешь знакомиться ближе, поражает стройность архитектуры библиотеки. Кажется, что в ней нет ничего лишнего и бесполезного. Общая библиотека классов жестко привязана к среде исполнения — она является ее неотъемлемой частью. В этом есть как свои плюсы, так и минусы. К плюсам можно отнести уменьшение размеров конечных приложений и оптимизацию библиотеки с повышением версии среды исполнения. К минусам относится возможная потеря совместимости сервисов общей библиотеки для различных версий среды исполнения.

С учетом требований к быстродействию кода общей библиотеки классов значительная ее часть написана на языке C++, который транслируется в чистый машинный код. Для этого была использована специальная технология шлюзов, обеспечивающая переход на уровень самой виртуальной машины. При вызове некоторых методов классов общей библиотеки, управление напрямую передается в виртуальную машину, где этот метод реализован в чистом машинном коде.

1.3. .NET байт-код и язык представления кода IL

Рассмотрим вопрос о соотношении между .NET байт-кодом и промежуточным языком IL. Здесь уместна аналогия из мира ортодоксального программирования, когда процессор обрабатывает двоичные машинные коды, а программист пишет и читает их на языке ассемблера. Виртуальная машина также занимается исполнением чистого .NET байт-кода и не имеет ни малейшего

понятия о IL, который служит для удобного представления этого кода в текстовом виде. Пример приложения на языке IL приведен в листинге 1.2.

Листинг 1.2. Пример приложения на языке IL

```
/*
       Листинг 1.2
       File:   Some.il
       Author: Дубовцев Алексей
*/
// Подключим общую библиотеку классов .NET.
.assembly extern mscorlib {}
// Зададим общее имя нашей сборки.
.assembly Some {}
// Это основная функция нашего приложения.
.method static public void GeneralMethod()
{
  // Указываем на то, что данная функция является точкой
  // входа в приложение.
  .entrypoint
  // Устанавливаем размер стека, для того чтобы поместить
  // туда ссылку на строку.
  .maxstack 1
  // Помещаем ссылку на строку в стек.
  ldstr "Hello, World!"
  // Выводим строку на консоль
  call void [mscorlib]System.Console::WriteLine(string)
  // Выходим из функции
  ret
}
```

Это простой текстовый файл, который, безусловно, не может исполняться виртуальной машиной .NET. Для этого необходимо представленный исходный код транслировать в .NET байт-код при помощи компилятора `ilasm`. Эта операция осуществляется при помощи следующей команды:

```
ilasm Some.il
```

После завершения компиляции формируется исполняемый .NET-файл, содержащий реальный байт-код. Именно он будет исполняться виртуальной машиной .NET, а отнюдь не программой на языке IL. IL — это лишь средство представления байт-кода в наглядном виде.