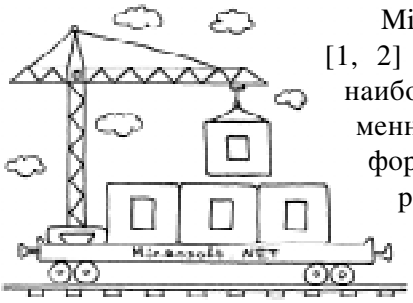
ТЕХНОЛОГИЧЕСКОЕ ОБРАЗОВАНИЕ

Сафонов Владимир Олегович

ПЛАТФОРМА MICROSOFT.NET: ПРИНЦИПЫ, ВОЗМОЖНОСТИ, ПЕРСПЕКТИВЫ



Microsoft.NET [1, 2] – одна из наиболее современных платформ для разработки программного обеспечения, наряду с Java-технологией [3, 4].

В настоящее время в университетах всего мира изучению платформы Microsoft.NET уделяется огромное внимание.

В частности, на математико-механическом факультете СПбГУ накоплен уникальный опыт преподавания как Microsoft.NET (более 3 лет), так и Java-технологии (более 7 лет).

Он нашел свое отражение в книге [4] по Java, популярной среди российских студентов, и в электронном учебном курсе по .NET [5] на английском языке, который в июне 2004 года выложен на Web-сайт учебных материалов Microsoft Curriculum Repository для использования его во всем мире.

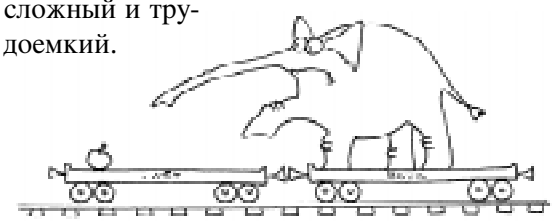
На платформе .NET развивается также наш исследовательский проект Aspect.NET [6] – инструментарий поддержки аспектно-ориентированного программирования [7] для .NET.

В работе [3] я уже писал о том, какую необыкновенную популярность, начиная с 1990-х годов, приобрела Java-технология.

Теперь практически то же самое можно сказать и о платформе Microsoft.NET.

Обе эти платформы – .NET и Java – являются самыми современными и самыми популярными технологиями и инструментариями для создания современного программного обеспечения.

Многие коммерческие компании как за рубежом, так и в России, уже начали активно использовать .NET для решения широкого класса задач, несмотря на то, что переход на новую платформу – процесс сложный и трудоемкий.



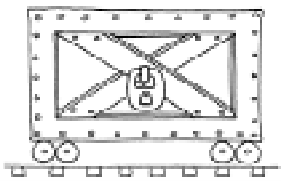
В данной статье мы рассмотрим основные особенности, понятия и возможности .NET, сравним ее с Java-технологией и с другими предшествующими технологиями, проанализируем ее достоинства, недостатки и перспективы.

Для более глубокого изучения Microsoft.NET отлично подходят работы [1, 2], а также значительное число других книг по отдельным аспектам .NET, выпущенных в 2002–2003 гг. издательством Microsoft Press. Многие из них переведены на русский язык.

1. ОСОБЕННОСТИ СОВРЕМЕННОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Чтобы лучше понять архитектуру .NET, рассмотрим современные задачи программирования и классы программ.

За почти 60 лет развития программирования подход к нему существенно изменился. Еще в 1960-х и 1970-х годах большинство программ были уникальными «монолитами», разработанными по специальным заказам огромными коллективами программистов. Подчас для решения каждой новой серьезной задачи разрабатывался и реализовывался новый язык программирования, что растягивало решение задачи на несколько лет и существенно усложняло ее. Не было всемирных сетей (но зато, к сожалению, существовал «железный занавес»), и обмен информацией как между программистами, так и между программами был затруднен. Использовать существовавшие тогда, хотя и очень полезные, библиотеки и пакеты программ было крайне сложно из-за неудобного интерфейса (необходимо было указывать чуть ли не физические адреса обрабатываемых данных, о мнемонике имен и говорить не приходилось). В такой ситуации наименьшим злом становилась разработка «с нуля» огромных и сложных программных систем, в которых (заново) реализовывались все необходимые компоненты, включая пользовательский интерфейс (который был тогда крайне неудобен – в «телетайпном» стиле), пакеты математических функций и др.

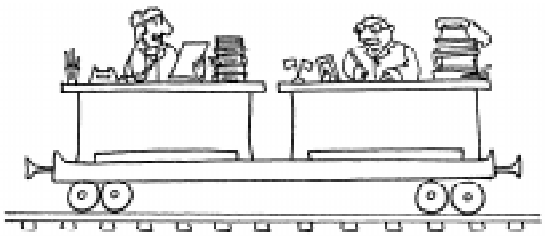


Вот некоторые важные классы задач, решаемых программными системами XXI века.

• *Разработка клиент-серверных систем*. Практически все современные программные инструменты основаны на клиент-серверной парадигме, обеспечивая поддержку разработки и использования (consuming) файл-серверов, серверов приложений (application servers), серверов баз данных (database servers), Web-серверов и др.

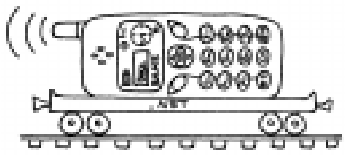
• *Интернет-приложения.* Ориентация на использование программ через Интернет и на использование программой ресурсов, взятых из Интернета, – важнейшая черта современного программирования. В связи с повсеместным использованием Интернета, практически любой программный продукт должен быть создан с учетом Интернета (Internet-aware). При этом современная платформа разработки программ должна обеспечивать создание простых Интернет-приложений встроенными средствами, без необходимости обращения к сложным клиентским программам, например, браузерам.

• *Решения (solutions).* Для поддержки производственных процессов на предприятиях, в фирмах и организациях, включая не только собственно производство, но и учет рабочего времени, передачу информации по корпоративной сети и хранение ее в корпоративных базах данных, обеспечение защиты информации и т. д., необходима разработка интегрированных *решений* – больших распределенных специализированных программных комплексов, сочетающих в себе поддержку как системной части, так и *бизнес-логики*, используемой в данной области и в данной компании. Естественные требования к подобным системам – повышенная надежность и удобный и простой пользовательский интерфейс.



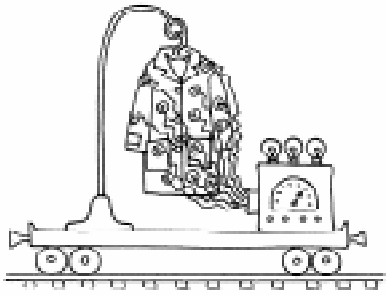
• *Встроенные системы.* Программные системы, предназначенные для управления специализированным оборудованием самого разного рода, всегда были актуальны, в особенности теперь, когда спектр применения подобных систем существенно расширился – от простых smart-карт до управления ядерными реакторами.

• *Программное обеспечение мобильных устройств (mobile intelligent devices)* в настоящее время особенно актуально. Набор таких устройств включает мо-



бильные телефоны, Web-телефоны, электронные органайзеры и др. Весьма важным требованием к подобным системам является возможность связывания мобильных устройств в сеть как с другими такими же устройствами, так и с «обычными» настольными и портативными компьютерами. Особенность подобного рода аппаратуры, с точки зрения программирования, – жесткие ограничения на ресурсы, прежде всего – на объем памяти.

• *Программное обеспечение носимых компьютеров (wearable computers).* Носимый компьютер – это специализированный процессор, соединенный с набором датчиков и встроенный в одежду человека, на-



значение которого состоит в анализе его состояния и ориентации, выдаче рекомендаций по их улучшению и по поведению в каждой ситуации. Естественно, подобные устройства, как и мобильные устройства, имеют жесткие ограничения на используемые ресурсы.

В связи с этим, современные программные системы имеют следующие особенности:

– ориентация на использование через Интернет (всемирную сеть), в том числе WWW, и Интранет (корпоративные сети);

– универсализация представления моделей (на языке UML – Unified Modeling Language) и данных (на языке XML – eXtended Markup Language);

– повышенные требования к *безопасности* (security) и *надежности* (reliability);

– *интеграция* различных *языков*, систем программирования, баз данных, знаний и сетевых средств в *единую инфраструктуру*;

– ориентация на создание многократно используемых (reusable) программных *компонент*.

2. СРАВНЕНИЕ С JAVA-ТЕХНОЛОГИЕЙ

В настоящее время наиболее популярными платформами для разработки программного обеспечения, в которых учтены все описанные выше особенности, являются:

– Java (разработки фирмы *Sun Microsystems, 1995*) – платформа для разработки программ на базе объектно-ориентированного языка Java, компилируемого в Java-байткод (постфиксную, или обратную польскую, запись программы);

– .NET (*Microsoft, 2000*) – объектно-ориентированная *многоязыковая* платформа с единой системой типов, единым промежуточным языком MS IL (также основанным на постфиксной записи) и единым представлением информации на базе XML.

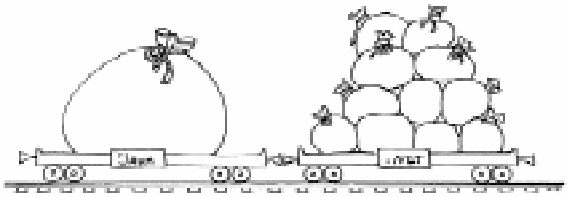
Отметим два основных отличия .NET от Java:

1) *Многоязыковость*. Платформа Microsoft.NET ориентирована на создание программ, сочетающих в себе компоненты, написанные на разных языках, наиболее

удобных пользователям. Платформа Java прежде всего ориентирована на создание программного обеспечения на языке Java. При необходимости использования других языков (например, C или C++) компоненты, написанные на этих языках, необходимо искусственно оформлять в виде так называемых *native-методов* (платформно-зависимых методов), каковыми они на самом деле не являются. Новый мощный язык программирования C#, разработанный одновре-

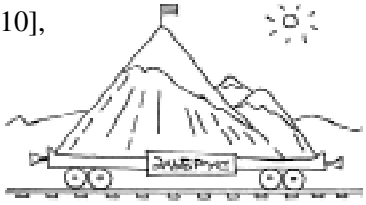
менно с платформой .NET, является наиболее удобным для программирования в среде .NET, но не является обязательным или, тем более, единственным.

2) *Наличие международных стандартов*. По существу, в отличие от Java, .NET – *это не конкретный программный продукт, а набор международных стандартов,* разработанных международной ассоциацией ECMA и специфицирующих архитектуру системы, общую систему типов, требования к многоязыковой совместимости, интерфейс базового набора библиотек и язык C#. То есть, *сначала были созданы международные стандарты, описывающие .NET, и только лишь потом, в 2000 г. – Microsoft.NET (реализация .NET фирмы Microsoft)*. Именно такой подход нам представляется наиболее правильным. Уже сейчас существуют несколько реализаций стандартов .NET: *коммерческая версия .NET,* реализованная Microsoft; *открытая «академическая» версия .NET,* доступная вместе с документацией и исходными текстами – Shared Source Common Language Infrastructure (SSCLI, или «Rotor») [8], реализованная Microsoft и развиваемая международным академическим сообществом; *система Mono* [9], также развиваемая международным сообществом программистов и ориентированная на Linux и Windows. Вполне возможно, что читатели данной статьи в ближайшем будущем примут участие в развитии этих или других реализаций .NET, в соответствии с ее международными стандартами, – например, для какой-либо новой архитектуры компьютеров и новой операционной системы. К сожалению, как я уже писал [3], Java-технология до сих пор не имеет международного стандарта (ISO, ANSI, ECMA и т.д.) спецификации языка, библиотек и виртуальной машины, – последние находятся «в руках» фирмы Sun, что сдерживает развитие Java-технологии.



3. НЕМНОГО ИСТОРИИ

Многоязыковые системы – не новость в истории программирования. Не новой является также идея использования постфиксной записи в качестве промежуточного языка. Интересным примером такой системы, возможно, известным не всем читателям, является система «Эльбрус» [10], созданная по мотивам идей Дж. Айлифа [11], высказанных еще в конце 60-х – начале 70-х гг., и под влиянием американской системы Burroughs 6700/7700. Именно эта историческая аналогия возникла в моей памяти, когда я впервые познакомился с архитектурой .NET.



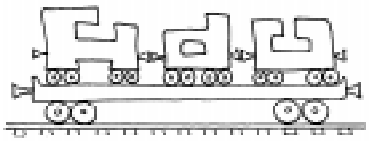
В системе «Эльбрус» была реализована аппаратная поддержка основных конструкций и механизмов реализации *процедурных языков программирования* (Паскаль, АЛГОЛ, ФОРТРАН и др.). Компилятор системы Эльбрус транслировал программу в объектный код, который, так же, как и MS IL в .NET, был *обратной польской записью* программы. Интересно, что, по существу, это был не «окончательный», а именно *промежуточный* код, так как аппаратура «Эльбруса» во время выполнения транслировала его в обычный *трехадресный регистровый код*, то есть, говоря современным языком, осуществляла аппатартую *just-in-time–компиляцию.* Кроме того, компилятор для «Эльбруса» генерировал *дополнение к файлу объектного кода – унифицированные таблицы, содержащие информацию о структуре программы и определенных и использованных в ней типах*. То есть, фактически, компиляторы в Эльбрусе генерировали *метаданные* (выражаясь в терминологии .NET), и эта информация использовалась во время выполнения для динамической диагностики, отладки и линковки в терминах исходных языков программирования. Заинтересованных читателей отсылаю к монографии [10].

Данный пример подтверждает общие принципы развития, в частности, в области программирования: идеи, реализованные в системах 30-летней давности, не получивших столь широкого распространения, развиваются и реализуются на новом уровне через 30 лет, на базе более современной – объектно-ориентированной – парадигмы программирования и становятся основой мощных современных коммерческих систем, используемых во всем мире.

4. ОСНОВНЫЕ ВОЗМОЖНОСТИ .NET

С точки зрения пользователя, платформа .NET предоставляет следующие основные возможности.

• *Многоязыковое программирование*. Для платформы .NET, в отличие от Java, все языки равноправны. Любая программа может быть разработана из *компонент*, написанных на разных языках. При этом обеспечивается простая стыковка модулей на разных языках на базе общей системы типов,



отладка и динамическая диагностика в терминах исходного языка, на котором написана компонента. Фирма Microsoft предоставляет реализацию пяти языков для .NET – C#, Visual Basic.NET, JScript, Managed C++, J# (последний является аналогом Java). Кроме того, для .NET доступны реализации многих популярных языков, созданные другими фирмами, – Perl, Python, Component Pascal, Eiffel, Active Oberon (Zonnon), ML, COBOL и многих других. По существу, появление платформы .NET дало новый, невиданный толчок развитию работ в области разработки компиляторов и инструментальных средств для их создания.

• *Управляемый код* – исполнение с динамическим контролем типов и безопасности. Общая среда выполнения – *Common Language Runtime (CLR)* – обеспечивает полный контроль соответствия использования операций (или методов) типам, для которых они определены, а также проверку соблюдения полномочий безопасности (security permissions), предоставленных выполняемому коду. Все это существенно повышает надежность, исключает наиболее распространенные ошибки (такие, как неверное использование указателей) и обеспечивает диагностику обнаруженных во время выполнения ошибок и нарушений безопасности с помощью *исключений (exceptions)* – механизма, который в .NET является языково-независимым.

• *Расширяемость*. В .NET возможна

динамическая генерация новых сборок, а также их загрузка из Интернета или Интранета, то есть любая программа для .NET является потенциально расширяемой, что просто необходимо для современного программирования. При этом обеспечивается полный контроль безопасности выполнения полученных таким образом фрагментов программы.

• *Развитые средства Web-программирования,* интегрированные со средствами доступа к данным. .NET позволяет создавать *Web-сервисы* и использовать их. При этом обеспечивается возможность обращения к базам данных и использования экранных *Web-форм*. Средства Web-программирования в .NET [12] – одна из самых привлекательных возможностей .NET.

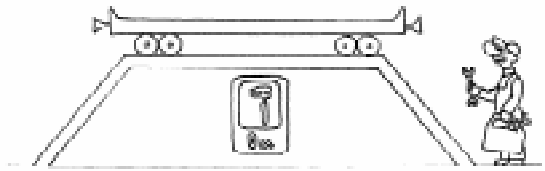
• *Возможность разбиения больших решений на прикладные области (Application Domains)* – «логические» (облегченные) процессы, обеспечивающие удобство управления ресурсами.

• *Удобная интегрированная инструментальная среда Visual Studio.NET*. Среда Visual Studio.NET развивает традиции

Microsoft Visual Studio, содержит визуальные дизайнеры и шаблоны проектирования и реализации для различных классов приложений. Visual Studio.NET – отдельный продукт, который должен быть отдельно приобретен, однако базовый инструментальный комплекс *Microsoft.NET Framework SDK* распространяется бесплатно через Интернет. Объем его дистрибутива – около 130 мегабайт. Он содержит все основные компиляторы, визуальный отладчик, библиотеки классов и целый ряд утилит, вызываемых из командной строки.

5. ОСНОВНЫЕ ПРИНЦИПЫ АРХИТЕКТУРЫ .NET

Для того, чтобы лучше понять специфику .NET, кратко рассмотрим ее «изнутри»,



то есть разберемся в том, каким образом достигаются и реализуются перечисленные возможности.

• *Единый промежуточный код и метаданные*. Компилятор с любого языка для платформы .NET транслирует исходный текст программы или компоненты в единый промежуточный код – *Common Intermediate Language (CIL)*; другое название – *Microsoft Intermediate Language (MS IL)*. Как уже отмечалось, он представляет собой постфиксную запись программы и напоминает Java байт-код – промежуточный язык Java-технологии. CIL – полноправный язык .NET, имеющий свой ассемблер (*ilasm*) и дизассемблер (*ildasm*). Он является уникальным примером «объектно-ориентированного языка ассемблера». Для сравнения, в Java-технологии не поощряется и не поддерживается использование байт-кода в качестве языка программирования.

Кроме CIL-кода, компилятор генерирует *метаданные (metadata)* – стандартизованное представление информации о типах данных, определяемых и используемых в данной программной компоненте. Метаданные фактически представляются в виде набора *таблиц*, и работа с ними организуется таким же образом, как работа с реляционной базой данных.

Отметим, что по нашему опыту именно концепция метаданных вызывает определенные трудности для восприятия студентами, хотя по существу она очень проста: *метаданные – это типы*, представленные в стандартной форме, понятной всем инструментам .NET. Именно эта идея – ключ к пониманию метаданных.

Метаданные доступны впоследствии во время выполнения, что и обеспечивает возможность полного динамического контроля типов, то есть тот режим исполнения, который в .NET называется *исполнением управляемого кода (managed code execution)*.

Метаданные содержат также *атрибуты (attributes)* – языково-независимые аннотации. Любая именованная величина (сборка, класс, поле, метод и др.) может содержать такие атрибуты, с помощью которых описываются способы работы с ней, например, возможность *сериализации* – преобразования в последовательный поток байтов, полномочия безопасности и др.

Кроме того, возможны, что особенно важно, *атрибуты, определяемые пользователем (custom attributes)*, то есть, по существу, с каждой именованной величиной может быть связана *произвольная* информация, необходимая пользователю. Это могут быть, например, *формальные спецификации*, на основе которых может быть формально доказана правильность программы, информация об *аспектах* [6], определенных или использованных в данной компоненте, информация об *авторе* программы и многое другое. Подобная возможность является уникальной возможностью расширения среди коммерческих языков и систем (хотя идея атрибутов заимствована из языка описания интерфейсов IDL стандарта CORBA) и отсутствует, например, в Java.

Подобный подход обеспечивает также переносимость программ, представлен-

ных описанным образом, и возможность их передачи по сети и исполнения на любой машине и в любой системе, в которой имеется реализация .NET.

• *Сборки и их версии.* С логической

точки зрения, совокупность CIL-кода и метаданных образует *сборку (assembly)* – основную единицу манипулирования программами и развертывания программ в .NET. Сборка физически может быть представлена либо в виде одного *файла переносимого исполняемого кода (portable executable – PE)*, либо в виде нескольких таких файлов. Сборка имеет также *манифест (manifest)* – «инвентарную опись» всех файлов, входящих в сборку, включая и файлы ресурсов (например, рисунки или аудиофайлы).

Со сборкой может быть связан *номер версии*, имеющий вид: (*Major, Minor, Build, Revision*), где *major* – основной номер версии, *minor* – дополнительный, *build* – номер компоновки, *revision* – номер исправления. Именно на основе номера версии строится так называемое *полное имя (strong name)* сборки, по которому сборка, занесенная в специальный каталог (GAC – global assembly cache), становится доступной другим программам. Преимущество такой нумерации в том, что кажда сборка четко знает, какие версии других сборок она использует, что полностью решает для .NET известную в более ранних системах проблему путаницы с версиями библиотек и позволяет даже использовать в одной и той же программе несколько версий одной и той же сборки.

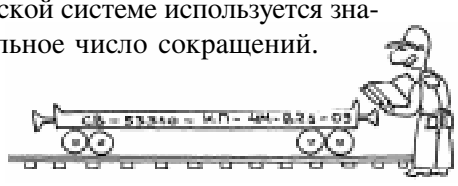
• *Just-in-time–компиляция.* Известно, что любое использование промежуточного кода может привести к значительному падению производительности программы, в особенности если этот промежуточный код интерпретируется. Эта проблема возникла в первой версии Java, что и привело к идее *динамической компиляции (just-in-time compilation* – буквально, «компиляции как раз вовремя»), реализованной в последующих версиях JDK. Авторы .NET учли эту проблему с самого начала, поэтому можно сказать, что just-in-time (сокращенно – JIT) компиляция является одной из основ архитектуры .NET. Принцип JIT-компиляции состоит в следующем. Во время исполнения при первом вызове какого-либо метода, представленного в CIL-коде, этот промежуточный код (динамически) компилируется в код конкретной целевой платформы (native code), на которой происходит выполнение. При последующих вызовах данного метода управление сразу передается уже скомпилированному native-коду. Если метод не вызывается, его JIT-компиляции не происходит. Этот важнейший принцип существенно повышает производительность выполнения программы, если метод вызывается многократно.

• *Использование XML.* Разработчики .NET учли и тот факт, что с 1999 года язык XML стал de facto-стандартом для представления информации в Интернете и передачи информации через Интернет. В .NET в унифицированной XML-форме представляются как информация, касающаяся Web-сервисов (спецификация интерфейса Web-сервиса, данные передаваемые ему и его результаты), так и все конфигурационные файлы, управляющие поиском сборок, задающие полномочия безопасности и т. д.

Преимуществами XML является структурированная форма представления информации и наглядность текстов на этом языке. Фактически, при необходимости любой пользователь может легко прочесть, понять и модифицировать содержимое любого конфигурационного файла.

6. ОСНОВНЫЕ ТЕРМИНЫ И СОКРАЩЕНИЯ .NET

Как известно, в любой сложной технической системе используется значительное число сокращений.



К сожалению, многие их них похожи друг на друга, что может затруднить их понимание и использование. Не является исключением и .NET.

Приведем и поясним основные сокращения и соответствующие им концепции .NET.

• *CLI (Common Language Infrastructure)* – стандарт ECMA для единой инфраструктуры языков, по которому реализована .NET. В этом стандарте описаны основные принципы архитектуры .NET, включая универсальное представление типов, особенности исполнения программ и т. д.

• *CTS (Common Type System)* – единая система типов данных, используемая в .NET. Эта система типов основана на объектно-ориентированной парадигме, то есть содержит такие категории типов, как класс и интерфейс, специфицирует элементы типа (класса) – поля, методы, свойства, события, *делегаты* (типизированные указатели на методы – обработчики событий). Подобные типы относятся к *ссылочным типам* (*reference types*), то есть семантика объектов этих типов, подобно Java, основана на понятии *объектной ссылки* и размещении объектов в *куче* (*heap*) – глобальной области памяти. Однако, для повышения эффективности и совместимости с языками C и C++, в CTS введены также *типы-значения (value types),* к которым, кроме простых типов, относятся также *структуры*. Все эти значения хранятся в стеке.

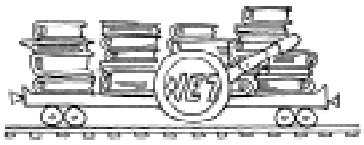
Фактически, одной из основных задач, стоящих перед любым разработчиком компилятора для .NET, является отображение системы типов реализуемого языка в CTS. Оно представляет определенную трудность, как и при использовании любой унификации, то есть вынуждает разработчиков компилятора мыслить о типах «своего» языка в терминах типов .NET, что само по себе может быть воспринято как ограничение. Но без унификации нет и многоязыковой совместимости. Кроме того, тот факт, что для .NET уже реализованы десятки языков, основанных на самых разных парадигмах, – не только объектно-ориентированные, но и процедурные, функциональные, продукционные, языки со статической и динамической типизацией, – доказывает, что эта задача вполне решаема.

• *CLS (Common Language Specification)* – единая спецификация требований к реализациям языков программирования для .NET, соблюдение которой обеспечивает совместимость программных компонент, написанных на разных языках. Поясним на примере, о чем идет речь. Разные языки программирования различным образом трактуют даже самые простые понятия, например, идентификаторы. В одних языках (например, в Паскале) идентификатор может быть записан символами в любом регистре, в других (например, в C#) имеет место чувствительность к регистру. Кроме того, возможно, что идентификатор одного языка совпадает по написанию с ключевым словом другого. Согласно правилам CTS, в реализации каждого языка для .NET должна быть обеспечена чувствительность к регистру идентификаторов, а никакой идентификатор не должен совпадать с ключевым словом никакого другого языка, чтобы не нарушить многоязыковой совместимости и не вызвать конфликтов имен. Это только два примера правил CTS, а всего их – 41.

• *CLR (Common Language Runtime)* – единая среда выполнения для .NET, которую можно рассматривать как *виртуальную машину*, по аналогии с Java. Именно CLR обеспечивает динамический контроль типов и полномочий безопасности, JIT-компиляцию, управление памятью, сборку мусора, обработку исключительных ситуаций и многие другие функции. Вся эта функциональность реализована единым образом для всех языков, благодаря CTS и CLS. То есть, в частности, исключение, сгенерированное в компоненте на C#, может быть обработано в вызвавшей ее компоненте, написанной на Visual Basic.NET.

CLR содержит огромное число библиотек классов, которые могут быть использованы в любой программе.

7. ПОЧЕМУ СЛЕДУЕТ ИСПОЛЬЗОВАТЬ И ИЗУЧАТЬ .NET



Мы рассмотрели только самые основные понятия и возможности .NET. Полностью они описаны в сотнях книг, написанных ее разработчиками и другими ведущими экспертами по .NET, такими как специалисты фирмы Wintellect Джеффри Рихтер и Дино Эспозито.

Но даже этот краткий перечень дает представление о реальных возможностях .NET – платформы, которая обеспечивает решение всех рассмотренных классов задач на любом удобном программисту языке, реализованном для .NET, для широкого диапазона устройств, включая мобильные телефоны и электронные органайзеры.

Только лишь средства Web-программирования .NET, дающие возможность разработки Web-сервисов, заслуживают того, чтобы ради них изучить и начать использовать .NET.

Безусловно, использование .NET можно рекомендовать программистам, разрабатывающим интегрированные программные решения для предприятий. .NET фактически создана для них, как и для разработчиков Web-сервисов.

Весьма привлекательным является язык программирования C#. При его создании учтен опыт создания Java, и авторы C# привнесли в язык все то полезное, что они нашли в Java. Однако следует учитывать, что C# гораздо мощнее, чем Java, так как содержит много дополнительных возможностей – *индексаторы* (позволяющие рассматривать объекты некоторого класса как абстрактные массивы), *свойства* (аналог полей с программируемыми операциями доступа и присваивания), *делегаты, циклы по коллекциям*, *дополнительные модификаторы доступа* для элементов типов и многое другое.

Опыт использования и сравнительной оценки платформ Java и .NET говорит о том, что аналогичные задачи решаются в .NET более эффективно – например, вызов виртуального метода в .NET на той же аппаратной платформе и в той же ОС работает в 1,5–2 раза быстрее.

Кроме того, как было показано, подход .NET к разработке программ гораздо более широкий, чем в Java, он более открытый, стимулирует к развитию, реализации и использованию новых языков при сохранении переносимости программ. Кроме того, наличие международных стандартов позволяет перенести архитектуру .NET на новые платформы.

По мнению экспертов Microsoft, .NET – платформа следующего десятилетия. Последующий опыт покажет правомерность этого тезиса. На данный момент Java лидирует в области создания программного обеспечения для мобильных телефонов, однако и .NET имеет соответствующие возможности – .NET Compact Framework, Microsoft Smartphone.

Так или иначе, для полноценного университетского образования в области системного программирования изучение и .NET, и Java в равной степени необходимо. Знание обеих этих платформ становится, например, одним из важных критериев при приеме на работу.

Наш опыт показывает, что для студентов большой интерес представляет именно сравнительное изучение .NET и Java. Кроме того, весьма полезны рассмотрение и анализ концепций .NET в исторической ретроспективе. Все это облегчает изучение платформы .NET, которая, как приходится признать, более сложна для понимания студентами, по сравнению с Java.

В заключение хотелось бы выразить надежду, что данная статья и учебный курс [5] еще более повысят интерес к .NET и позволят изучить эту перспективную платформу многим программистам.

Литература

1. *Рихтер Д.* Программирование на основе Microsoft.NET Framework. М.: MS Press, 2002.
2. *Просиз Д.* Программирование для Microsoft.NET. М.: MS Press, 2002.
3. *Сафонов В.О.* Java-технология: история, состояние и перспективы // Компьютерные инструменты в образовании, 2003, № 2.
4. *Сафонов В.О.* Введение в Java-технологию. СПб.: Наука, 2002.
5. *Safonov V.* Microsoft.NET architecture and the C# language // Microsoft Curriculum Repository, www.msdnaa.net, June 2004.
6. *Safonov V.* Aspect.NET – a new approach to aspect-oriented programming // .NET Developer's Journal, 2003, # 4.
7. Web-сайт по аспектно-ориентированному программированию: www.aosd.net
8. Web-сайт SSCLI (Rotor): www.sscli.net
9. Web-сайт Mono: www.go-mono.com
10. *Сафонов В.О.* Языки и методы программирования в системе Эльбрус. М.: Наука, 1989.
11. *Айлиф Дж.* Принципы построения базовой машины. М.: Мир, 1972.
12. *Платт Д.* Знакомство с Microsoft.NET. М.: MS Press, 2001.

Сафонов Владимир Олегович,
доктор техн. наук,
профессор кафедры информатики
Санкт-Петербургского
университета.