Платформа .NET и ее применение для объектно-ориентированного подхода к программированию

Обзор .NET. Основные понятия

ПЛАТФОРМА (в контексте ИТ) –

среда, обеспечивающая выполнение программного кода.

Платформа определяется характеристиками процессоров, особенностями операционных систем.

Framework – это инфраструктура среды выполнения программ, нечто, определяющее особенности разработки и выполнения программного кода на данной платформе.

Предполагает средства организации взаимодействия с ОС и прикладными программами, методы доступа к БД, средства поддержки распределенных приложений, ЯП, множества базовых классов, унифицированные интерфейсы пользователя, парадигмы программирования.

.NET Framework

Microsoft .NET* – *платформа.

.NET Framework – инфраструктура платформы Microsoft .NET.

<u>Основные компоненты:</u>

- Common Language Runtime (CLR)
- .NET Framework Class Library (.NET FCL).

Концепция и возможности подхода .NET

.NET – это подход к проектированию и реализации ПО, включает компоненты:

- идеология проектирования и реализации ПО;
- модель эффективной поддержки ЖЦ прикладных систем;
- унифицированная, интегрированная технологическая платформа для программирования;
- современный, удобный в использовании, безопасный инструментарий для создания, развертывания и поддержки ПО.

.NET Framework

Важнейшие аспекты идеологии .NET:

- легкость развертывания приложений в глобальной среде Internet;
- экономичная разработка ПО;
- "бесшовная", гибкая интеграция программных продуктов и аппаратных ресурсов;
- предоставление ПО как сервиса;
- новый уровень безопасности и удобства использования.

Идеология .NET

Корпорация MS предложила компонентно-ориентированный подход к проектированию:

интеграция объектов (возможно, гетерогенной природы), производится на основе интерфейсов, представляющих эти объекты (или фрагменты программ) как независимые компоненты.

Это облегчает написание и взаимодействие программных "молекул"- компонентов в гетерогенной среде проектирования и реализации.

Стандартизируется хранение и повторное использование компонентов программного проекта в условиях распределенной сетевой среды вычислений, где различные компьютеры и пользователи обмениваются информацией.

Идеология .NET

Преимущество - возможность практической реализации принципа ***"всякая сущность представляет собой объект гетерогенной программной среды".***

Это стало реализуемым благодаря усовершенствованной системе типизации Common Type System (CTS).

Строгая иерархичность организации пространств для типов, классов и имен сущностей программы позволяет стандартизировать и унифицировать реализацию.

Новый подход к интеграции компонентов приложений в среде Internet (web-сервисы) - возможность ускоренного создания приложений для глобальной аудитории пользователей.

Универсальный интерфейс .NET Framework обеспечивает интегрированное проектирование и реализацию компонентов приложений, разработанных в соответствии с различными подходами к программированию.

.NET обеспечивает одновременную поддержку проектирования и реализации ПО с использованием различных ЯП.

Поддерживаются десятки ЯП: от самых первых (COBOL, FORTRAN) до современных (C#, VB).

Технология web-сервисов

- масштабируемость и интероперабельность

- ***Масштабируемость*** - возможность плавного роста времени ответа программной системы на запрос с ростом числа одновременно работающих пользователей;

в случае web-сервисов реализуется посредством распределения вычислительных ресурсов между сервером и компьютером пользователя.

Интероперабельность - возможность интегрированной обработки гетерогенных данных, поступающих от разнородных прикладных программ.

Благодаря интероперабельности возможна унификация взаимодействия пользователей через приложение с ОС на основе специализированного интерфейса прикладных программ, API-интерфейса (Application Programming Interface).

Технология .NET

официально признана, это отражено в стандартах ECMA (European Computer Manufacturers Association)

Инструментальные возможности .NET

- Поддержка многоязыковой среды разработки приложений CLR (Common Language Runtime). Эта возможность появилась благодаря универсальному межъязыковому интерфейсу Common Language Infrastructure (CLI), он поддерживает разработку программных компонентов на различных ЯП.

Преимущество для программистов - они могут разрабатывать (дорабатывать) ПО на наиболее подходящем ЯП.

Следует учитывать характер задачи (рекурсия или символьная обработка – реализуется на языке функционального программирования, а формализация структуры предметной области – на ОО языке).

Необходимо учитывать и опыт работы программистов в команде разработчиков и ЯП, на котором изначально создавалась система.

2 важных обстоятельства:

1) основные сервисные возможности для разработчиков, которые предоставляет .NET (отладка, анализ кода) не зависят от конкретного ЯП → программистам не надо заново постигать особенности среды разработки при "переходе" на другой ЯП
2) не все ЯП поддерживаются .NET, но возможно самостоятельно разработать транслятор для любого ЯП.

Безопасность .NET

Безопасность - важнейший элемент любой идеологии, технологии и инструментального средства программирования.

.NET как инструментальное средство призвано обеспечивать необходимый уровень безопасности.

Для этого в .NET реализована мера безопасности - автоматизированное управление ЖЦ ПО.

Для программиста это проявляется:

в автоматической реализации процедуры "сборки мусора",

в запрете использования указателей на области памяти с неопределенным значением ("висячих" ссылок) и ссылающихся на себя указателей (циклических ссылок).

Безопасность .NET

- Автоматизация обеспечения синтаксической коррекции кода (безопасные вызовы функций, контроль выхода за границы размера статически распределяемых областей памяти, запрет использования переменных, если им не задано значение по умолчанию).

- Обязательная проверка промежуточного кода (IL – Intermediate Language) на корректность типизации в рамках стратегии расширенного контроля соответствия типов.

- Усовершенствованы права доступа пользователей к ресурсам (для включения компонента в проект необходимо проверить источник кода, заверенный автором цифровой подписью и подлинность отправителя).

- Широкий спектр динамически корректируемых в соответствии с профилями пользователя политик доступа.

- Криптографические методы для шифрования конфиденциальной, коммерческой информации, передаваемой по Internet-каналам

Концепция web-сервисов

- средство поддержки распределенных компонентных вычислений в глобальной сети.

Задачи:

- интерактивная обработка пользователями информации (документов, таблиц, графики), представленной в электронном виде;
- организация совместной работы пользователей с прикладным ПО (заседания рабочих групп, конференции и т.д.);
- поддержка взаимодействия прикладных программ.

В концепции .NET сформулирована и решена задача адаптации изначально не структурированной Internet-среды для достижения возможности интеграции приложений.

Основные направления решения задачи:

- унификация информационной инфраструктуры;
- достижение необходимого уровня интероперабельности ПО;
- достижение необходимого уровня масштабируемости прикладного ПО.

Задача достижения необходимого уровня масштабируемости прикладного ПО является технически сложной.

Задача поддержки интероперабельности прикладного ПО решается с помощью универсальной высокоуровневой языковой среды Common Language Infrastructure (CLI).

Возможности CLI: Поддержка различных ЯП и подходов к программированию: функционального, ОО, компонентного.

Интегрированное использование ЯП осуществляется с единой системой типов (CTS), включающей общую иерархию для примитивных типов, типов-значений и ссылочных типов.

В .NET в обобщенном варианте реализовано управление оперативной памятью: централизованно осуществляются "сборка мусора" и тестирование кода на безопасность. Унифицирован механизм обработки исключительных ситуаций.

Универсальная высокоуровневая языковая среда Common Language Infrastructure обеспечивает межъязыковую отладку.

Стандарты

Для тиражируемого коммерческого программного продукта необходимо подтверждение его соответствия мировым стандартам.

Теоретические достижения и технологические инновации технологии .NET подтверждаются мнением европейской ассоциации по стандартам ECMA (European Computer Manufacturers Association).

ЯП C# и среда Common Language Infrastructure (CLI) ратифицированы организацией ECMA в качестве международного стандарта.

Разработчики ПО осуществляют проекты в соответствии со спецификациями ECMA (реализация .NET под управлением ОС Linux).

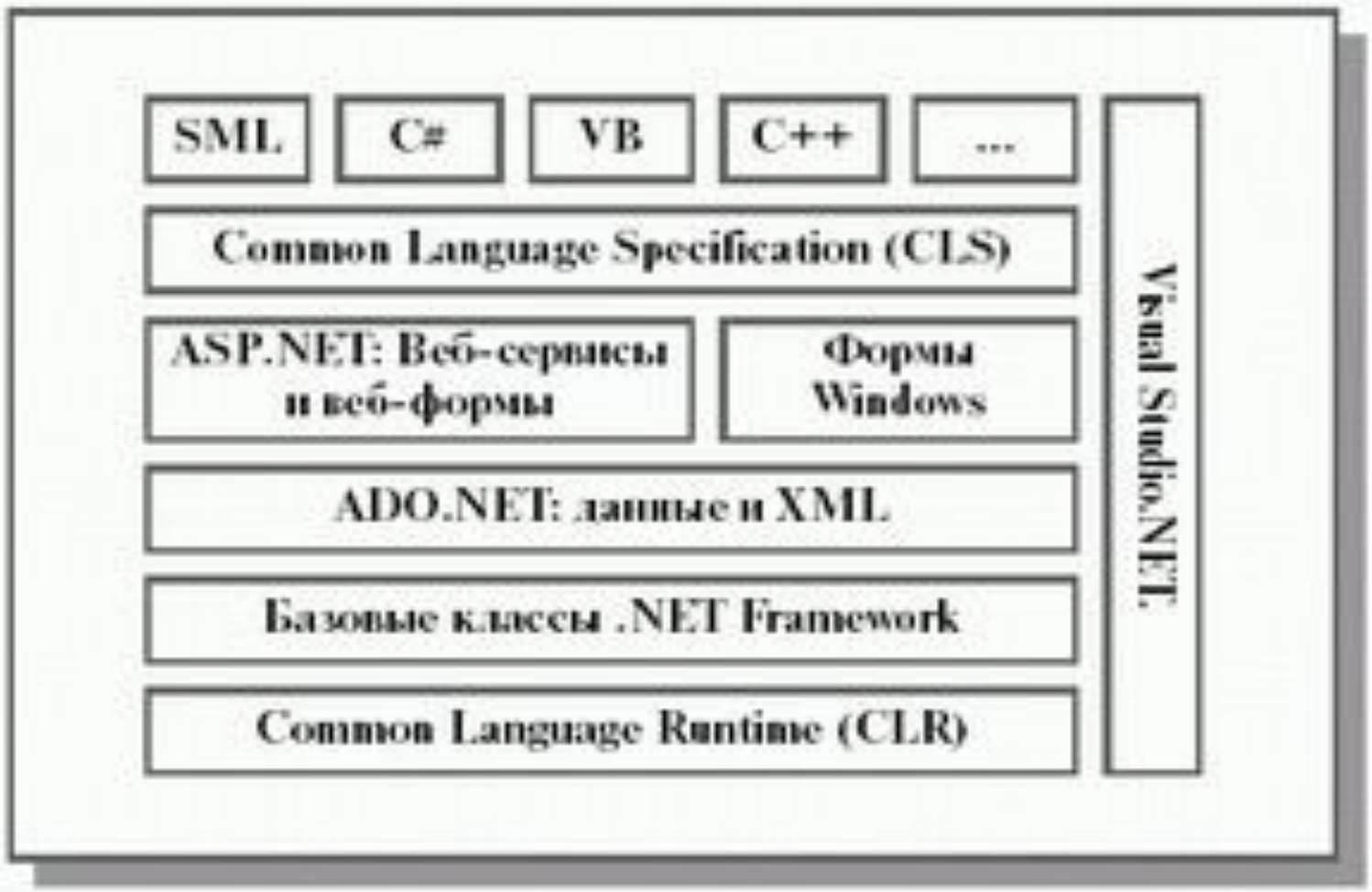
Основные аспекты архитектурного решения Microsoft .NET Framework

Важную роль играет среда разработки Microsoft Visual Studio.NET.

Первостепенное значение имеет среда выполнения программ – Common Language Runtime (CLR).

CLR реализует управление памятью, типами данных, межъязыковым взаимодействием, развертыванием (deployment) приложений.

Архитектурная схема .NET Framework и Visual Studio.NET.



Преимущество конструктивного решения .NET - ***компонентно-ориентированный подход*** к проектированию и реализации ПО.

Суть подхода - принципиальная возможность создания независимых составляющих ПО с унифицированной интерфейсной частью для многократного повторного и распределенного использования.

Продуктивность решения обусловлена многоязычностью интегрируемых программных проектов (концепция .NET потенциально поддерживает произвольный ЯП).

При компиляции программа на .NET-совместимом ЯП трансформируется в соответствии с заранее заданной обобщенной спецификацией языка ***Common Type System (CTS***).

Система типов CTS описывает все типы данных, поддерживаемые средой выполнения, определяет их взаимосвязи и хранит их отображения в системе типов .NET.

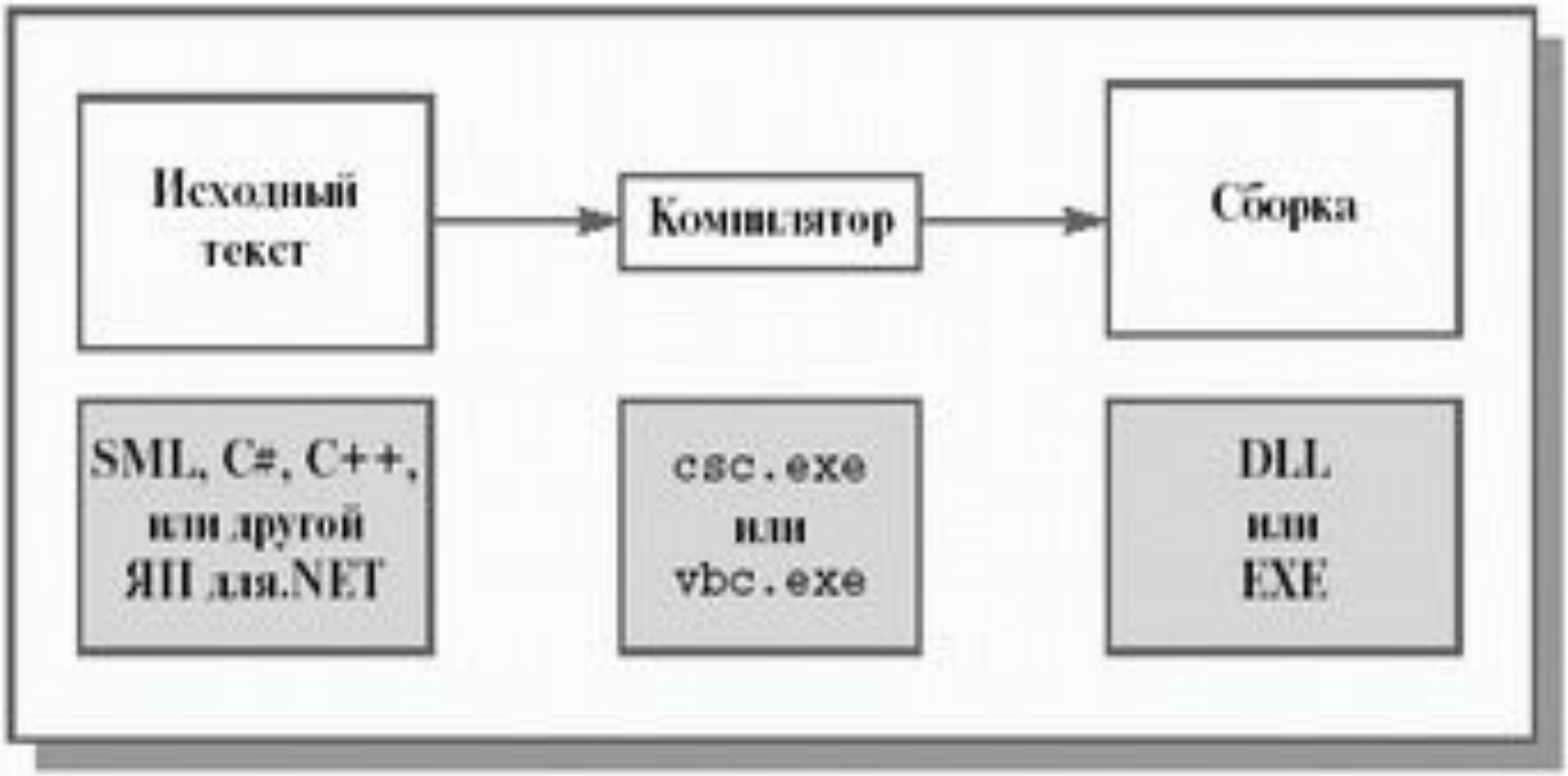
Common Language Specification (CLS)

- набор правил, определяющих подмножество обобщенных типов данных, в отношении которых гарантируется, что они безопасны при использовании во всех языках .NET.

Интерфейсы реализуются посредством форм Windows и ASP.NET для веб-приложений.

При трансляции исходный текст программы (SML, C#, VB, C++ или др.) преобразуется компилятором в сборку (assembly) и сохраняется в виде файла динамически присоединяемой библиотеки (Dynamically Linked Library, DLL) или исполняемого файла (Executable, EXE).

Схема компиляции Common Language Runtime



Для каждого компилятора (csc.exe, vbc.exe) средой времени выполнения производится необходимое отображение используемых типов в типы CTS, а программного кода – в код "абстрактной машины" .NET –

MSIL (Microsoft Intermediate Language).

В итоге программный проект формируется в виде ***сборки*** – самодостаточного компонента для развертывания, тиражирования и повторного использования.

Сборка идентифицируется цифровой подписью автора и уникальным номером версии

Поддержка ЖЦ ПО в рамках подхода .NET

Для установки на ПК пользователей ранее созданного прикладного ПО создаются инсталляционные комплекты в форме сборок.

Сборка - множество модулей, необходимых для инсталляции ПО

Сборка характеризуется уникальностью, обеспечиваемая идентификатором версии и цифровой подписью автора.

Сборка - самодостаточная единица для установки ПО. Возможно сетевое и индивидуальное использование сборки на основе компонентной технологии.

Сборка обеспечивает простой и удобный механизм инсталляции, экономит средства на развертывание ПО.

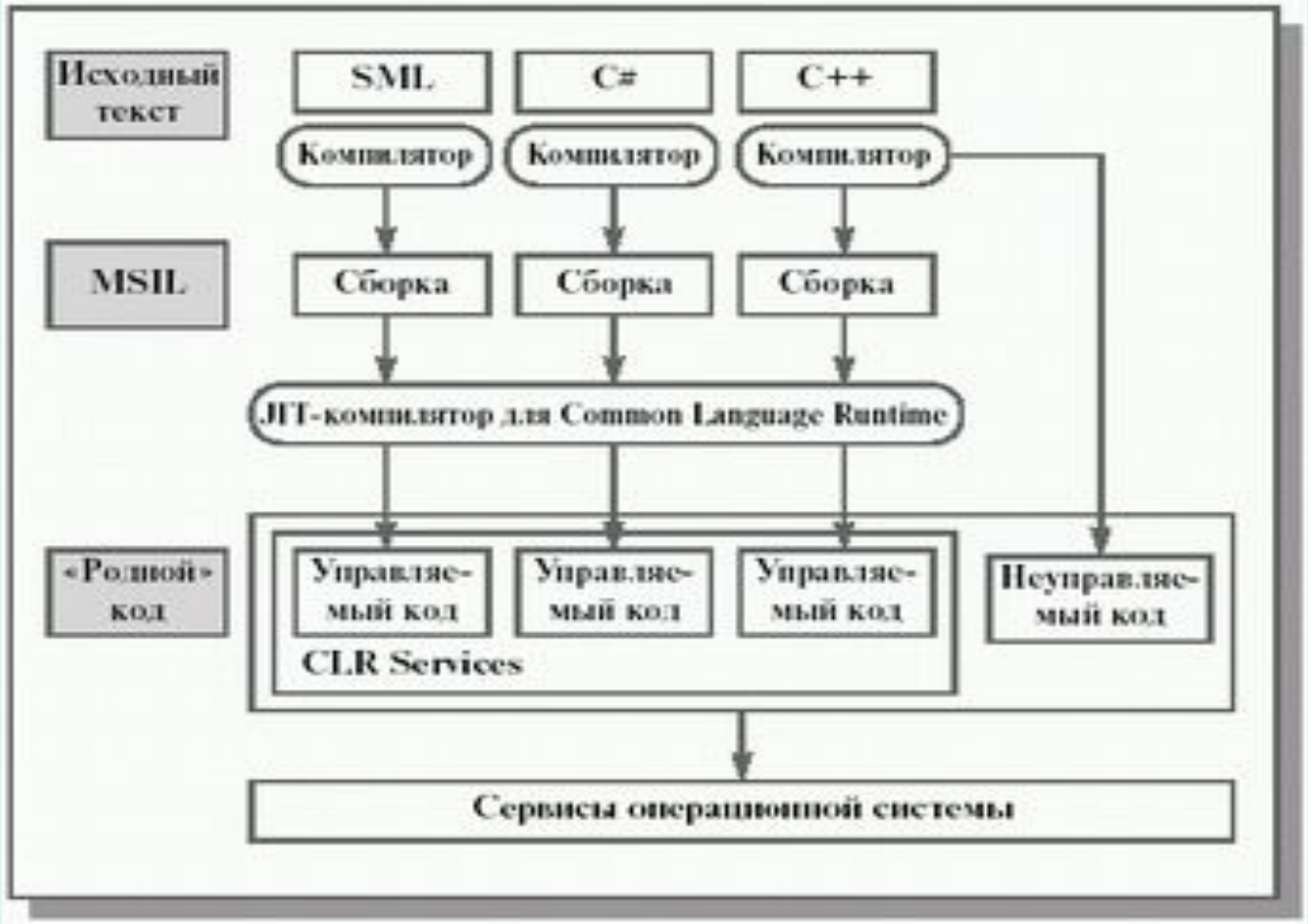
Описание сборки - в *манифесте* (идентификация автора, версия, режим и политика ее использования).

Пример трансляции многокомпонентного гетерогенного программного проекта под управлением Microsoft .NET:

Пусть компоненты проекта написаны на трех ЯП: SML, C# и C++(характеризуется возможностью создания потенциально небезопасного кода - в частности, динамического распределения памяти).

Исходные тексты компонент проекта транслируются компиляторами с языков SML, C# и C++ в унифицированный MSIL-код и сохраняются в файлах в виде сборок.

Схема выполнения CLR.



В ходе компоновки и выполнения программного проекта Just-In-Time (JIT) компилятор среды CLR производит выполнение проекта с "ленивым" (по мере необходимости) означиванием оттранслированного промежуточного кода сборок.

Потенциально небезопасный код на C++ невыполним собственно JIT-компилятором, но исполняется посредством сервисов ОС. Ответственность за работоспособность программы и безопасность кода лежит уже не на среде проектирования и разработки ПО .NET, а на программисте-разработчике.

Отличие MS .NET от аналогов - универсальная система типизации.

В ходе компиляции программа на .NET-совместимом ЯП трансформируется в соответствии с заранее заданной Common Type System (CTS) обобщенной спецификацией языка.

Система типов CTS полностью описывает все типы данных, поддерживаемые средой выполнения, определяет их взаимосвязи и хранит их отображения в систему типов .NET.

CTS

- представляет собой частично упорядоченное множество, понимаемое на качественном уровне как ISA-иерархия.

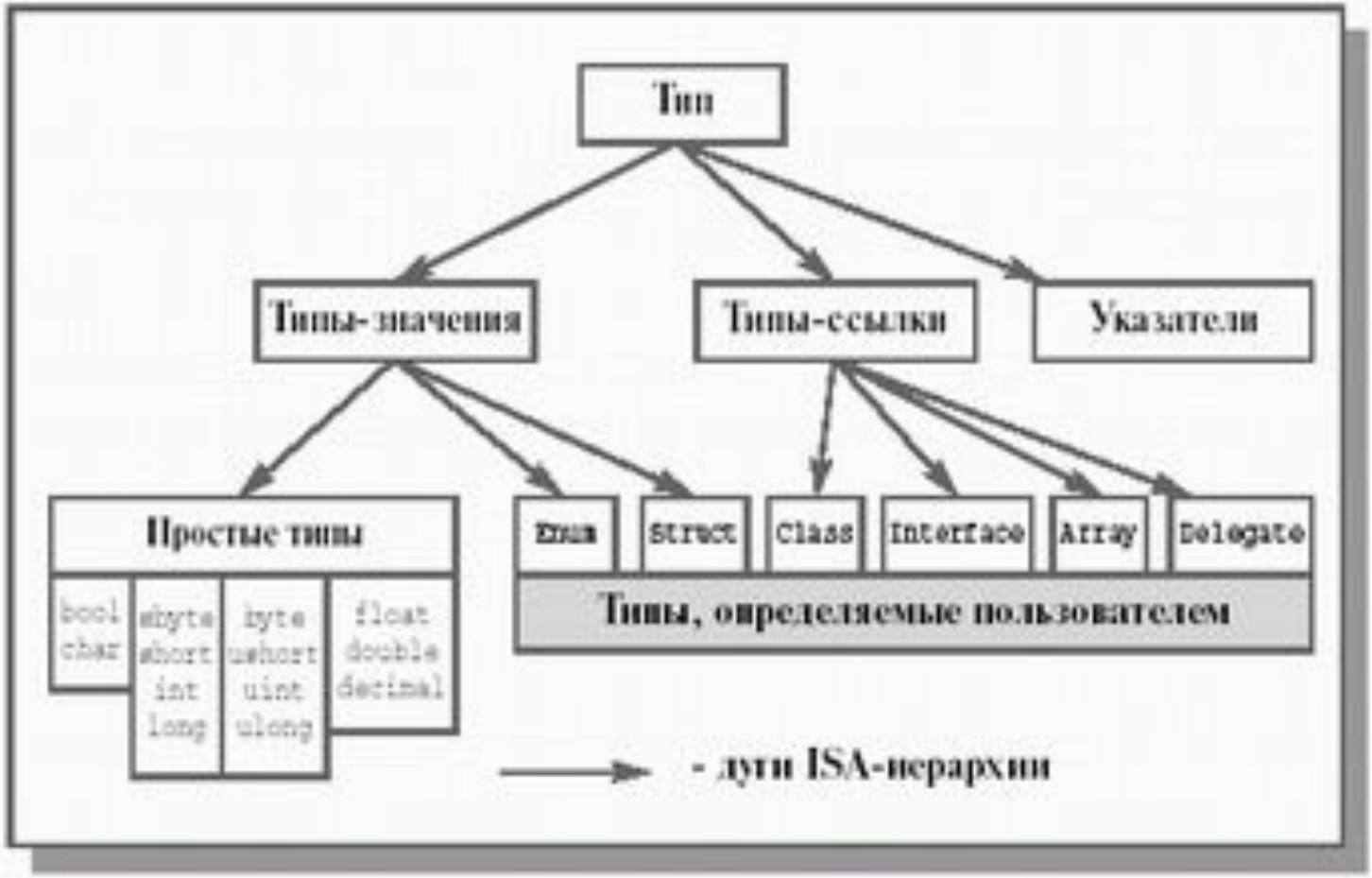
Например, высказывание ***STUDENT ISA PERSON*** означает, что тип STUDENT является подтипом типа PERSON.

Система типов MS .NET образует иерархию с возрастанием общности снизу вверх, в которой выделяются две большие группы типов: типы-ссылки и типы-значения.

Различие между ними определяется особенностями вызова в процедурах: по имени или по значению (call-by-name, CBN) и по ссылке (call-by-reference, CBR).

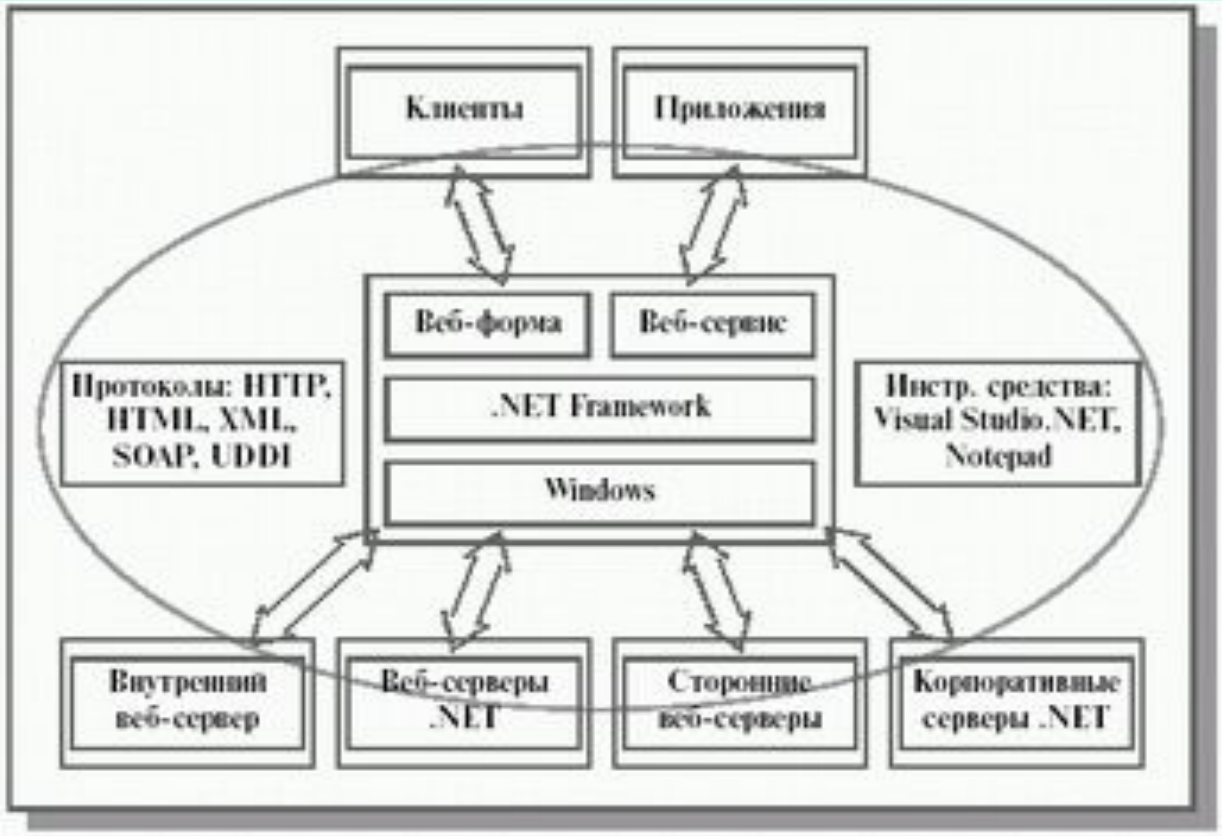
Система типизации MS .NET позволяет пользователю создавать собственные типы (типы-ссылки и типы-значения) на основе уже существующих.

Универсальная система типизации (UTS)



Веб-сервисы

Значение - распределение возможностей разработанных прикладных систем по каналам Internet.



Центральный блок -.NET Framework (библиотека базовых объектов и операций над ними)

Среда разработки прикладных систем - Microsoft Visual Studio .NET.

Интерфейсная часть прикладной программной системы в Internet-архитектуре представлена веб-формами для ввода и вывода данных в унифицированном формате.

Язык реализации - HTML. Взаимодействие между клиентом и приложением в простейшем случае - с использованием протокола передачи данных HTTP.

Структурированные данные хранятся в формате XML (вариант HTML с более строгим синтаксисом).

Технология веб-сервисов допускает интеграцию с компонентами независимых производителей.

Веб-сервисы - программируемые компоненты прикладных программных систем, доступные для клиента посредством стандартных протоколов, применяемых для работы в Internet-среде, предназначены для реализации основополагающего принципа «ПО как сервис».

Веб-сервисы организуются на традиционных стандартах взаимодействия приложений в Internet:

- HTTP – стандартный протокол обмена гипертекстовыми документами в Internet с возможностью передачи данных посредством веб-форм;
- XML – формат хранения структурированных данных с возможностью обмена ими по Internet-каналам;
- SOAP – стандартный протокол взаимодействия компонент (глобально) распределенного приложения (Simple Object Access Protocol);
- UDDI – стандарт интеграции приложений (Universal Description, Discovery and Integration);
- WSDL – универсальный язык описания веб-сервисов (Web Service Description Language) и др.

Компонентный подход к программированию

Центральная концепция - понятие компонента.

Компонент - независимый модуль ПО, который можно использовать повторно, тиражировать.

Свойства компонентов:

- компонент обладает более высоким уровнем абстракции;
- компоненты могут содержать в своем составе множественные классы;
- компоненты с т.зр. пользователя инвариантны к ЯП, на котором они реализованы.

Попытки построения компонентных программных систем - и другими разработчиками ПО (технология JavaBeans производства Sun Microsystems), а также международных ассоциаций исследователей и практиков в области ООП (стандарт брокеров объектных запросов CORBA организации Object Management Group).

В основе таких попыток - варианты объектных моделей.

Один из вариантов, детально проработанный с математической точки зрения,– модель двухуровневой концептуализации.

Особенности известных объектных моделей

Компонентная модель Microsoft COM - основной стандарт MS для компонентного проектирования и реализации ПО. Самая развитая и практически удачная модель, обеспечивает возможность инициализации и использования компонентов внутри одного процесса, между процессами или между ПК независимо от языка реализации.

COM-модель поддерживается в идеологии .NET для ряда ЯП (C#, SML, Visual Basic, C++ и др.), является основой для ActiveX, OLE и других технологий Microsoft.

Модель Java Beans, базовый стандарт Sun Microsystems для компонент, оказывается зависимой от языка реализации.

Итоги:

Microsoft считает.NET своей стратегической идеологией и технологической платформой на ближайшее время.

Превосходство над существующими средствами автоматизированного проектирования и быстрой реализации прикладного ПО (Inprise Delphi и JBuilder, Oracle Developer, Microsoft Visual Studio и др.) достигается за счет факторов:

- интероперабельность и межъязыковое взаимодействие;
- многоуровневая, гибкая и надежная политика безопасности;
- интеграция с технологией web-сервисов;
- упрощение процедуры развертывания и использования ПО.

Подход .NET оказывает влияние на коммерческую индустрию программирования и способствует радикальному совершенствованию отрасли в процессе рыночной конкуренции.

Терминология:

CLS (Common Language Specification) – общая спецификация ЯП. Это набор конструкций и ограничений, являющихся руководством для создателей библиотек и компиляторов в среде .NET Framework.

Библиотеки, построенные в соответствии с CLS, могут быть использованы из любого ЯП, поддерживающего CLS.

Языки, соответствующие CLS (Visual C#, Visual Basic, Visual C++), могут интегрироваться друг с другом. CLS – это основа межъязыкового взаимодействия в рамках платформы Microsoft .NET.

CLR (Common Language Runtime) – Среда Времени Выполнения или Виртуальная Машина. Обеспечивает выполнение сборки. Основной компонент .NET Framework.

Виртуальная Машина - абстракция инкапсулированная (обособленная) управляемой ОС высокого уровня, обеспечивает выполнение (управляемого) программного кода.

Управляемый код – программный код, при выполнении использует службы, предоставляемые CLR.

Задачи CLR:

- Управление кодом (загрузка и выполнение).
- Управление памятью при размещении объектов.
- Изоляция памяти приложений.
- Проверка безопасности кода.
- Преобразование промежуточного языка в машинный код.
- Доступ к метаданным (расширенная информация о типах).
- Обработка исключений, включая межъязыковые исключения.
- Взаимодействие между управляемым и неуправляемым кодами (в том числе и COM-объектами).
- Поддержка сервисов для разработки (профилирование, отладка и т.д.).

CLR – это набор служб, необходимых для выполнения управляемого кода.

Два главных компонента CLR:

- ядро (mscoree.dll)
- библиотеки базовых классов (mscorlib.dll).

Наличие этих файлов на диске –признак того, что на компьютере была предпринята попытка установки платформы .NET.

Ядро среды выполнения реализовано в виде библиотеки mscoree.dll.

При компоновке сборки в нее встраивается информация, которая при запуске приложения (EXE) или при загрузке библиотеки приводит к загрузке и инициализации CLR.

FCL (.NET Framework Class Library) – соответствующая CLS-спецификации ОО библиотека классов, интерфейсов и системы типов (типов-значений), которые включаются в состав платформы Microsoft .NET.

Обеспечивает доступ к функциональным возможностям системы и предназначена служить основой при разработке .NET-приложений, компонент, элементов управления.

.NET FCL могут использовать ВСЕ .NET-приложения, независимо от назначения архитектуры используемого при разработке ЯП, в частности:

- встроенные (элементарные) типы, представленные в виде классов (на платформе .NET все построено на структурах или классах);
- классы для разработки графического пользовательского интерфейса (Windows Forms);
- классы для разработки web-приложений и web-служб на основе технологии ASP.NET (Web Forms);
- классы для разработки XML и Internet-протоколов (FTP, HTTP, SMTP, SOAP);
- классы для разработки приложений, работающих с базами данных (ADO .NET) и многое другое.

MSIL (Microsoft Intermediate Language) – промежуточный язык платформы Microsoft .NET. Исходные тексты программ для .NET-приложений пишутся на ЯП, соответствующих спецификации CLS.

Для них может быть построен преобразователь в MSIL. Программы на этих языках могут транслироваться в промежуточный код на MSIL. Благодаря соответствию CLS, в результате трансляции программного кода на разных языках, получается совместимый IL-код.

Фактически MSIL является ассемблером виртуального процессора.

МЕТАДАННЫЕ – при преобразовании программного кода в MSIL формируется блок МЕТАДАННЫХ, содержит информацию о данных, используемых в программе. Фактически это наборы таблиц, включающих информацию о типах данных, определяемых в модуле. Ранее такая информация сохранялась отдельно.

Метаданные используются для:

- сохранения информации о типах. При компиляции не требуются заголовочные и библиотечные файлы. Всю необходимую информацию компилятор читает непосредственно из управляемых модулей;
- верификации кода в процессе выполнения модуля;
- управления динамической памятью (освобождение памяти) в процессе выполнения модуля;
- обеспечения динамической подсказки (IntelliSence) при разработке программы стандартными инструментальными средствами (Microsoft Visual Studio .NET) на основе метаданных.

Языки, для которых реализован перевод на MSIL: Visual Basic, Visual C++, Visual C# 2.0, и другие.

Исполняемый модуль – независимо от компилятора (и входного языка) результатом трансляции .NET-приложения является управляемый исполняемый (управляемый) модуль. Это стандартный переносимый исполняемый (PE – Portable Executable) файл Windows.

Управляемый модуль содержит управляемый код.

Управляемый код – это код, выполняемый в среде CLR.

Строится на основе объявляемых в исходном модуле структур и классов, содержащих объявления методов. Ему должен соответствовать определенный уровень информации (метаданных) для среды выполнения.

Код C#, Visual Basic, и JScript - управляемый по умолчанию.

Код Visual C++ не является управляемым по умолчанию, но компилятор может создавать управляемый код.

Особенность управляемого кода - наличие механизмов, позволяющих работать с управляемыми данными.

Управляемые данные – объекты, в ходе выполнения кода модуля размещаются в управляемой памяти (куче), уничтожаются сборщиком мусора CLR.

Данные C#, Visual Basic и JScript .NET - управляемые по умолчанию.

Данные C# также могут быть помечены как неуправляемые.

Сборка (Assembly) – базовый строительный блок приложения в .NET Framework.

Управляемые модули объединяются в сборки.

Сборка - логическая группировка одного или нескольких управляемых модулей или файлов ресурсов.

Управляемые модули в составе сборок исполняются в Среде Времени Выполнения (CLR).

Сборка может быть исполняемым приложением (файл .exe) или библиотечным модулем (файл .dll).

Ничего общего с обычными (старого образца!) исполняемыми приложениями и библиотечными модулями сборка не имеет.

Декларация сборки (Manifest) – составная часть сборки. Это набор таблиц метаданных, который:

- идентифицирует сборку в виде текстового имени, ее версию, культуру и цифровую сигнатуру;
- определяет входящие в состав файлы (по имени и хэшу);
- указывает типы и ресурсы, существующие в сборке;
- перечисляет зависимости от других сборок;
- указывает набор прав, необходимых сборке для корректной работы.

Эта информация используется в период выполнения для поддержки корректной работы приложения.

Процессор **НЕ МОЖЕТ** выполнять IL-код.

Перевод IL-кода осуществляется **JIT-компилятором** (Just In Time – в нужный момент), он активизируется CLR по мере необходимости и выполняется процессором.

Результаты деятельности JIT-компилятора сохраняются в оперативной памяти.

В среде CLR допускается совместная работа и взаимодействие компонентов ПО, реализованных на различных ЯП.

CLR решает многих проблем, которые традиционно находились в зоне внимания разработчиков приложений.

Функции, выполняемые CLR:

- Проверка и динамическая (JIT) компиляция MSIL-кода в команды процессора.
- Управление памятью, процессами и потоками.
- Организация взаимодействия процессов.
- Решение проблем безопасности (в рамках существующей в системе политики безопасности).

Структура среды выполнения CLR (основные функциональные элементы среды)

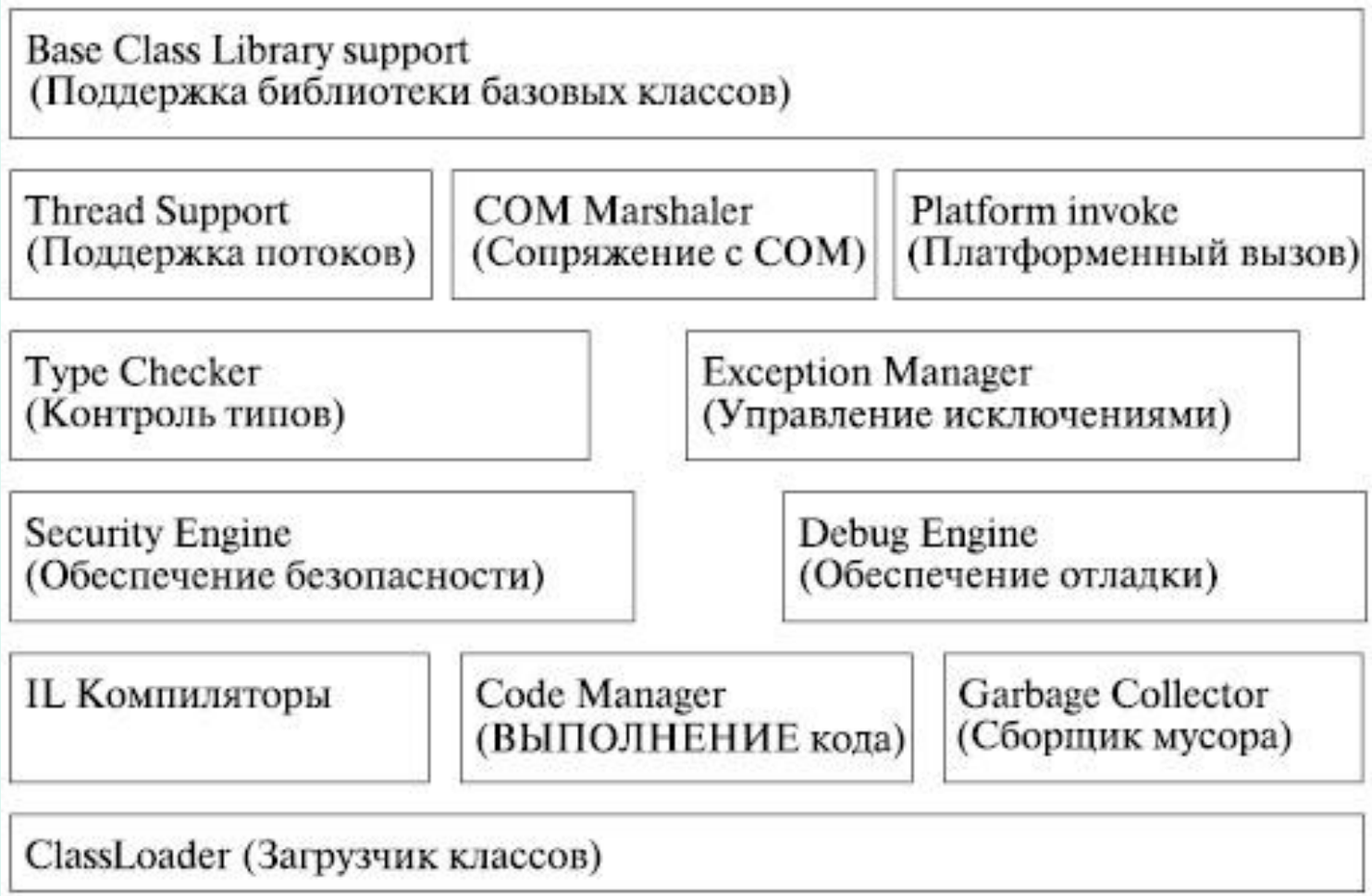
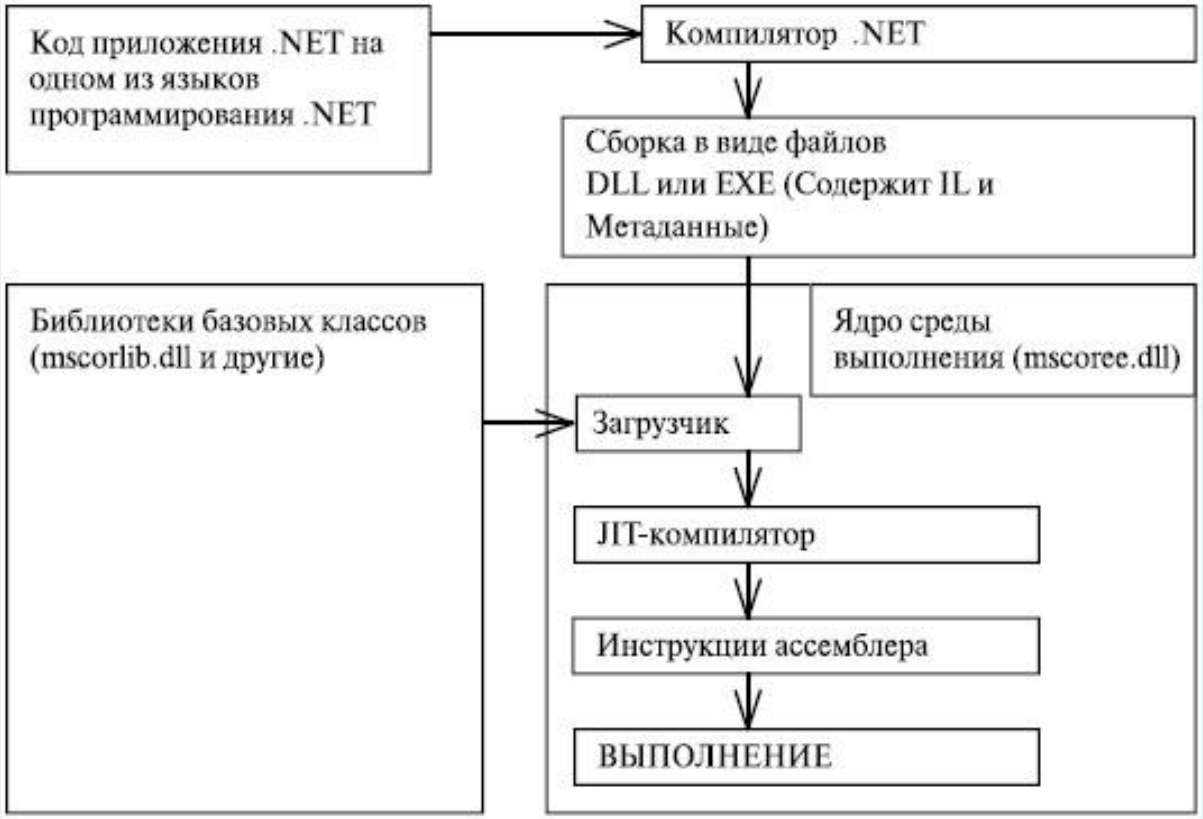


Схема выполнения .NET-приложения в среде CLR



Пространство имен – это способ организации системы типов в единую группу.

В рамках .NET существует единая (общеязыковая) библиотека базовых классов. Концепция пространства имен обеспечивает эффективную организацию и навигацию по этой библиотеке.

Вне зависимости от ЯП, доступ к классам обеспечивается за счет их группировки в рамках общих пространств имен.

Сборка мусора – механизм, позволяющий CLR определить, когда объект становится недоступен в управляемой памяти программы.

При сборке мусора управляемая память освобождается.

Для разработчика приложения наличие механизма сборки мусора означает, что он больше не должен заботиться об освобождении памяти.

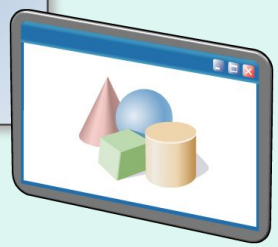
Это может потребовать изменения в стиле программирования, особое внимание следует уделять процедуре освобождения системных ресурсов.

Необходимо реализовать методы, освобождающие системные ресурсы, находящиеся под управлением приложения.

Среда разработки Visual Studio 2008

Веб-формы, ASP.NET AJAX

Windows Формы, WPF



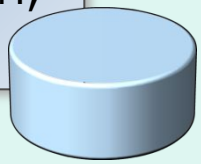
Microsoft® Visual Studio® 2008

Множество языков

Обработка ошибок

Веб-службы, WCF

Доступ к данным, LINQ

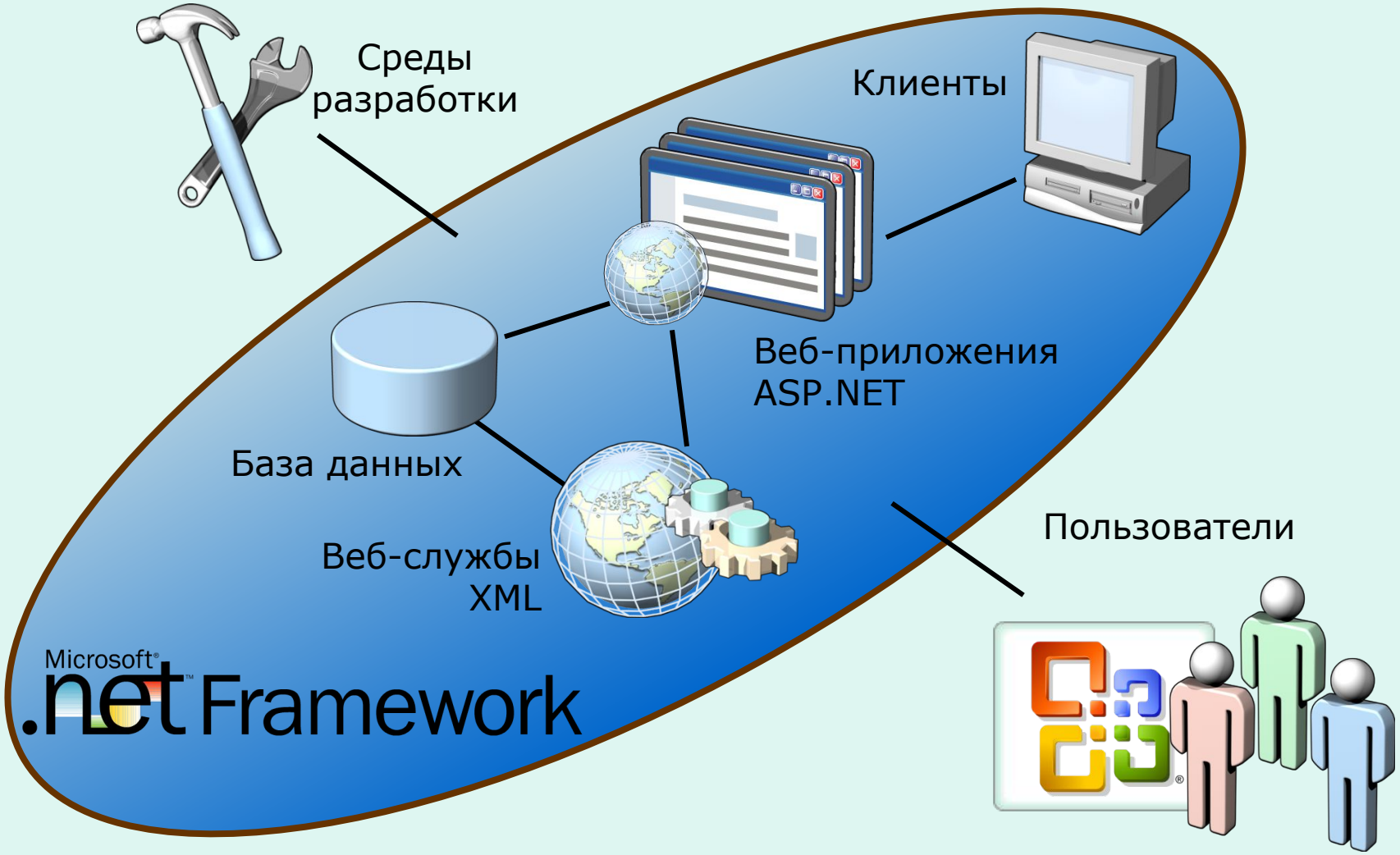


Проектиро-вание → Разработка → Отладка → Внедрение

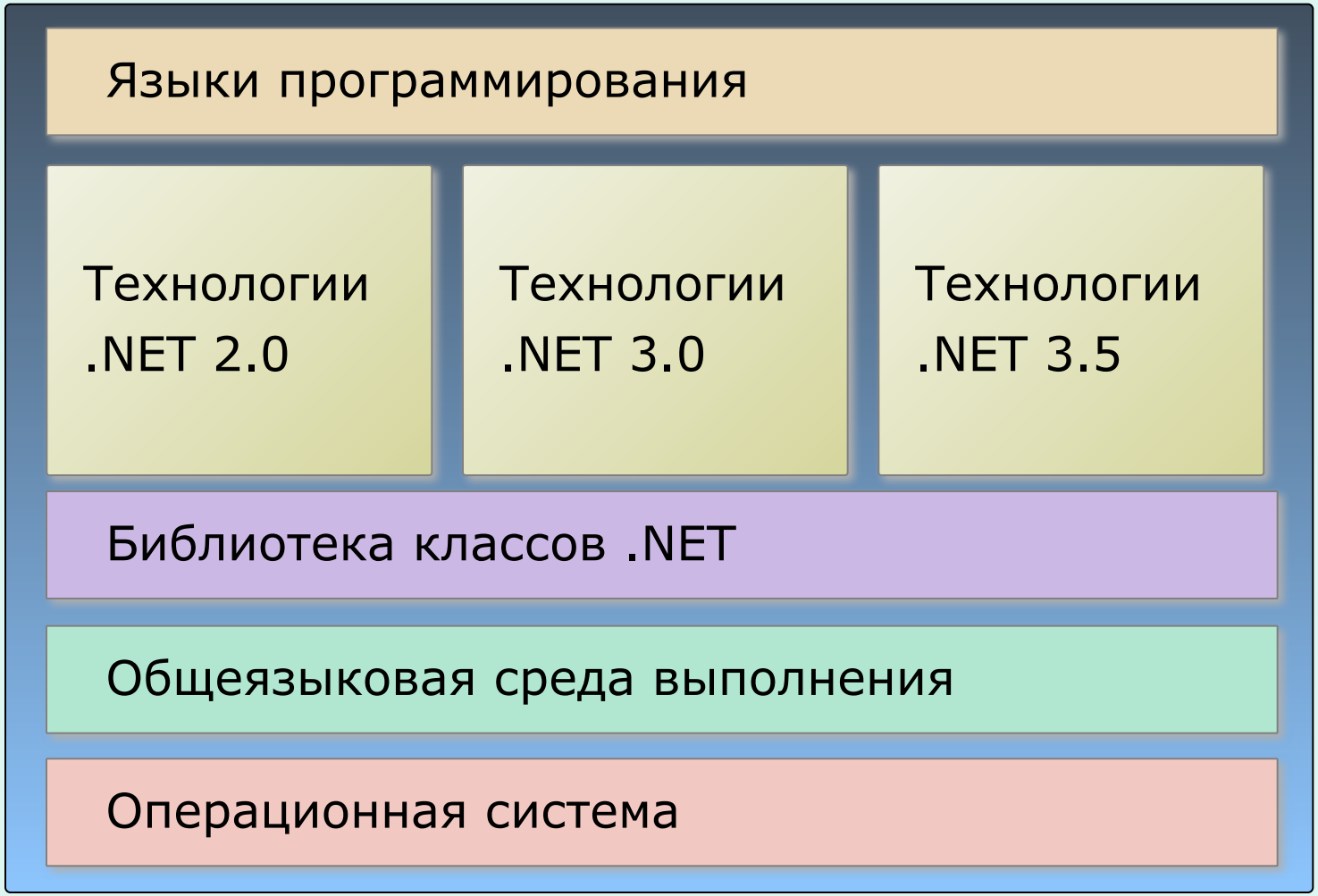
обладает богатым набором средств для эффективной разработки и позволяет управлять разработкой на всех этапах создания приложений

Что такое Microsoft .NET?

Microsoft.NET Framework – это независимая от платформы и от устройства система, разработанная для работы по Интернету

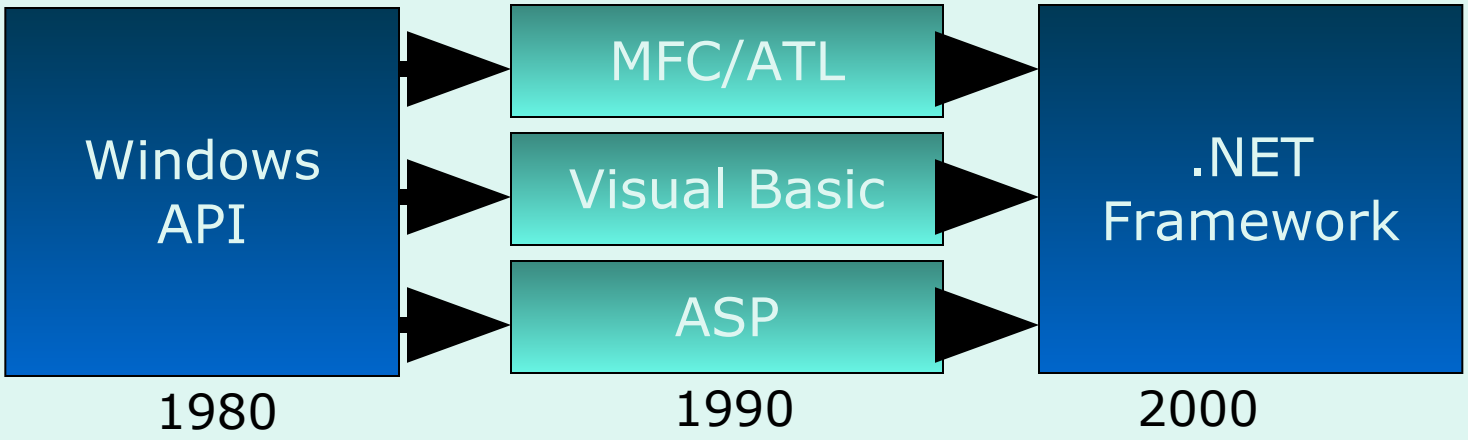


Компоненты .NET Framework



Преимущества .NET Framework

- Основан на Веб-стандартах и опыте разработок
- Классы .NET Framework общедоступны
- Код организован по иерархическим пространствам имен и классам
- Масштабируемость и независимость от языков



Разработка приложений в среде Microsoft Visual Studio

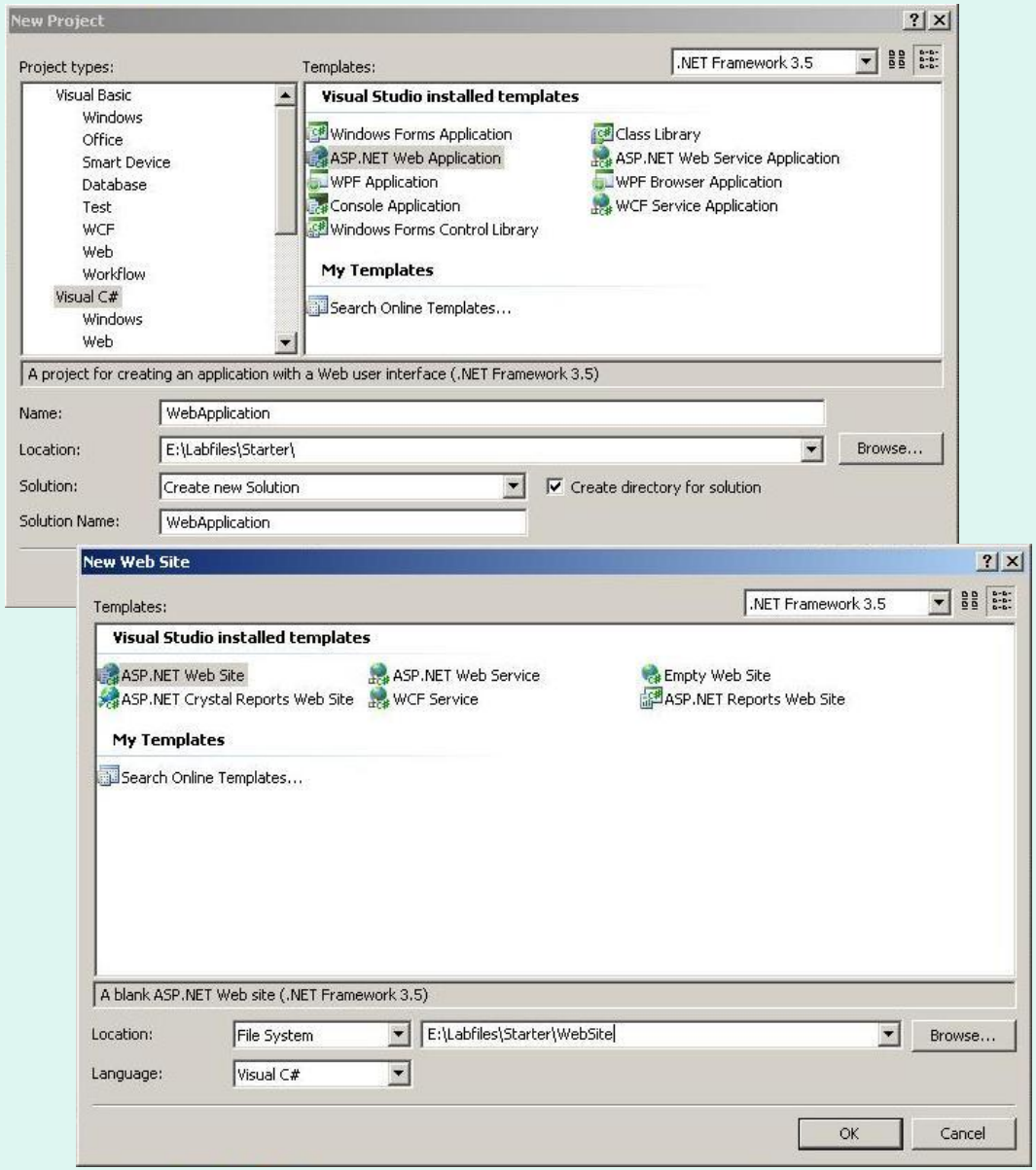
Почему Visual Studio?

- Единая интегрированная среда разработки для множество языков и множества типов шаблонов проектов
- Множество языков в проекте
- Множество типов проектов в пределах решений
- Поддержка приложений, функционирующих под множество версий .NET Framework
- Интегрированный обозреватель
- Поддержка отладки
- Настраиваемый интерфейс
- WPF, WCF, проектирование рабочего процесса и поддержка проекта
- ASP.NET AJAX и LINQ

Процесс разработки

Типы веб-приложений и файловая структура

- Шаблоны проектов веб-приложений ASP.NET
- Обеспечивают жесткий контроль над проектом
- Шаблоны проектов веб-сайтов ASP.NET
  - Проще в использовании
  - Предоставляет больше возможности и дополнительную гибкость



Файлы веб-приложений

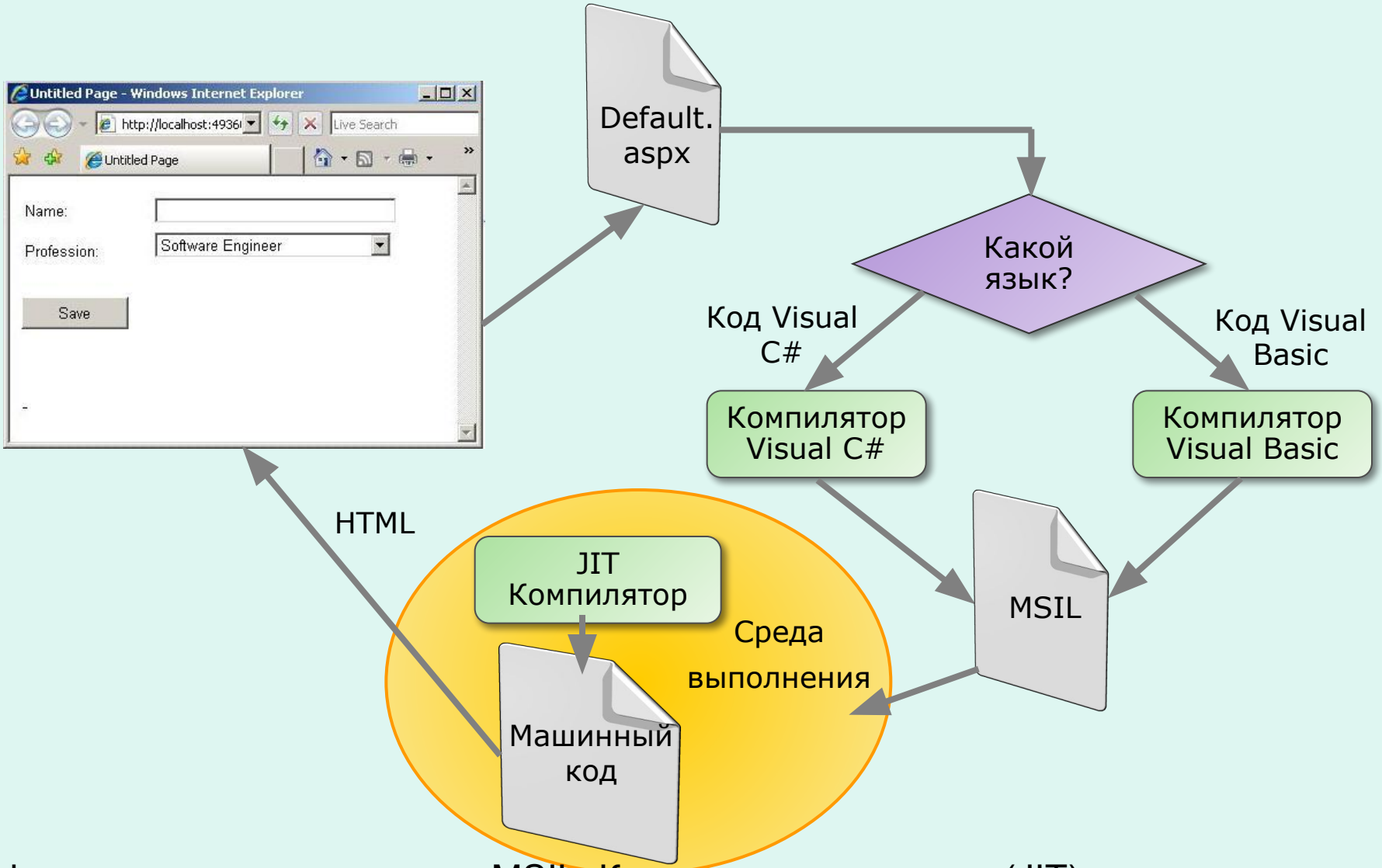
- Файлы веб-приложений
  - Веб-формы ASP.NET (.aspx)
  - Веб-службы ASP.NET (.asmx)
  - Классы и страницы с выделенным кодом (.vb или .cs)
  - Global application classes (.asax)
  - Файл Web.config
- Другие файлы
  - Файлы не основанные на языках программирования

Common Language Runtime

- *среда выполнения программ (CLR) реализует управление памятью, типами данных, межъязыковым взаимодействием, развертыванием приложений*

- Единое выполнение для всех.NET ориентированных языков
- Управляет потоками и памятью
  - Сборщик мусора
- Обеспечивает безопасность кода
- Устраняет проблемы управления версиями DLL
  - Может выполняться одновременно несколько версий DLL
  - Приложения могут указать версию используемой DLL

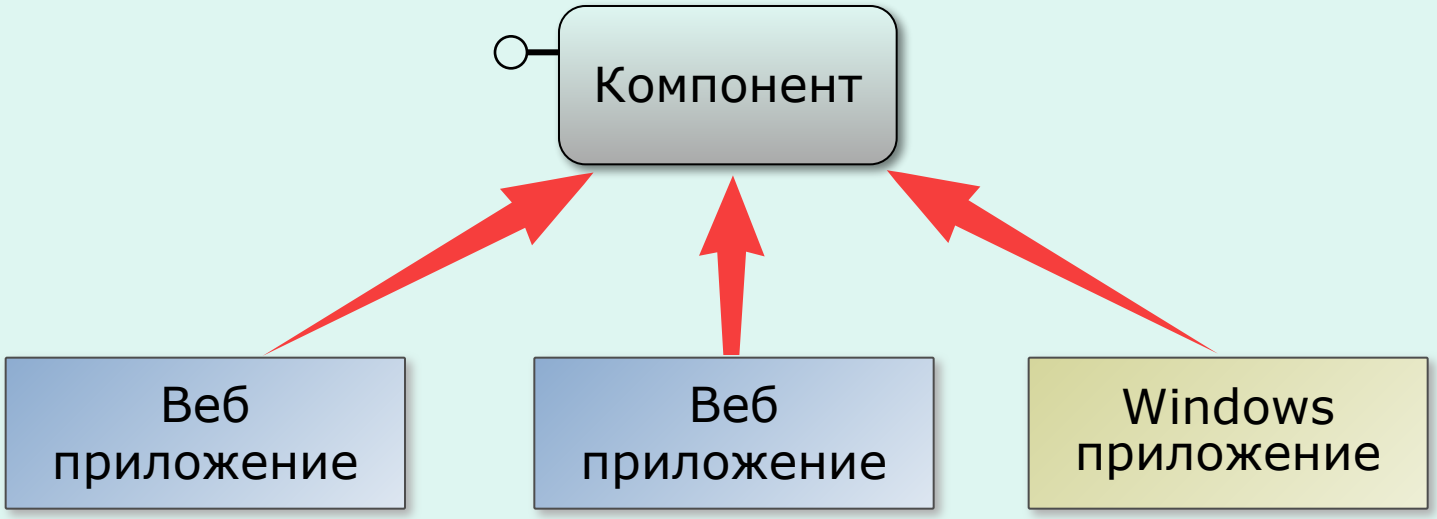
Компиляция и среда выполнение



Web странички исполняются в MSIL. Компиляция «на лету» (JIT).
Если код страницы изменился, то процесс компиляции начинается заново.

Создание компонента с использованием Visual Studio

- Классы группируют код без пользовательского интерфейса
- Компоненты – скомпилированные классы
  - Компоненты составлены в виде файлов DLL
- Компоненты используются для совместного использования кода между приложениями



Создание класса

- Создать проект библиотеки классов в Visual Studio 2008
  - Visual Studio 2008 создает пространство имен по умолчанию
- Создайть методы класса

```
[Visual C#]
public class Shipping
{
  public Single CalShipping (Single price)
  {
    ...
    return cost;
  }
}
```

```
[Visual Basic]
Public Class Shipping
  Function CalShipping (ByVal price As Single) As Single
    ...
    Return (cost)
  End Function
End Class
```

Доступ компонентов в веб-формах ASP.NET

- Добавить ссылку на DLL
- Экземпляр класса объекта:

[**Visual C#**]
```
CompanyA.Shipping shippingObject =
  new CompanyA.Shipping();
```

[**Visual Basic**]
```
Dim shippingObject As New _
  CompanyA.Shipping
```

```
namespace CompanyA
{
  class Shipping
  {
    public void CalShipping (…) { }
  }
}
```
component.dll

```
Namespace CompanyA
  Class Shipping
    Function CalShipping (…)
 End Class
End Namespace
```
component.dll

- Использование объекта:

```
cost =
  shippingObject.CalShipping(price);
```

```
cost = _
  shippingObject.CalShipping(price)
```