Разработка «коммерческих» приложений в среде .NET

Façade

Тема №1. Общая архитектура приложений

Архитектура Microsoft .NET Framework

Общая архитектура .NET-приложений

- Современные приложения достигли такого уровня развития, что термин «архитектура» в применении к ним уже давно не удивляет
- Считается, что «грамотно построить информационную систему, эффективно и надежно функционирующую не проще, чем сконструировать и возвести современное многофункциональное здание»

Общая архитектура .NET-приложений

- Когда речь заходит о Application Architecture, обычно не возникает недостатка в определениях
- Есть даже Web-сайты, которые собирают такие определения, см., например, http://www.sei.cmu.edu/architecture/start/community.cfm
- Одно из них (техническое) гласит:
  - *Software architecture is an abstraction, or a high-level view of the system. It focuses on aspects of the system that are most helpful in accomplishing major goals, such as reliability, scalability, and changeability. The architecture explains how you go about accomplishing those goals.*

Общая архитектура .NET-приложений

- Мы могли бы привести сотни подобных определений, и они все были бы вполне подходящими
- Однако, к примеру, если бы мы проектировали приложение для e-mail рассылки, тогда архитектура должна была нам помочь в том, какой тип приложения нам выбрать:
  - Windows Service
  - Web Service
  - console utility
  - Пакетный файл, запускаемые по расписанию и т.д.
- **Но**: архитектура не включает такие детали как:
  - Как обрабатывать исключения?
  - Как измерять производительность?
  - Как и куда программа логирует ошибки?
  - Как создать программу переиспользуемой?

Общая архитектура .NET-приложений

- Со временем некоторые известные и широко используемые архитекторами техники проектирования приложений развились и объединились под так называемыми *архитектурными стилями*
- Архитектурный стиль – это набор принципов, высокоуровневая схема, обеспечивающая абстрактную инфраструктуру для семейства систем
- Он улучшает секционирование и способствует повторному использованию дизайна благодаря обеспечению решений часто встречающихся проблем
- Архитектурные стили (или шаблоны) можно рассматривать как набор принципов, формирующих приложение

Общая архитектура .NET-приложений

- Наиболее известными архитектурными стилями считаются:
  - N-tier модель, или многоуровневая архитектура
  - Многослойная (N-layer) архитектура
  - Microsoft DNA
  - Data-centric
  - Service Oriented Architecture
  - Plug-in system
- Существуют множество других архитектурных стилей

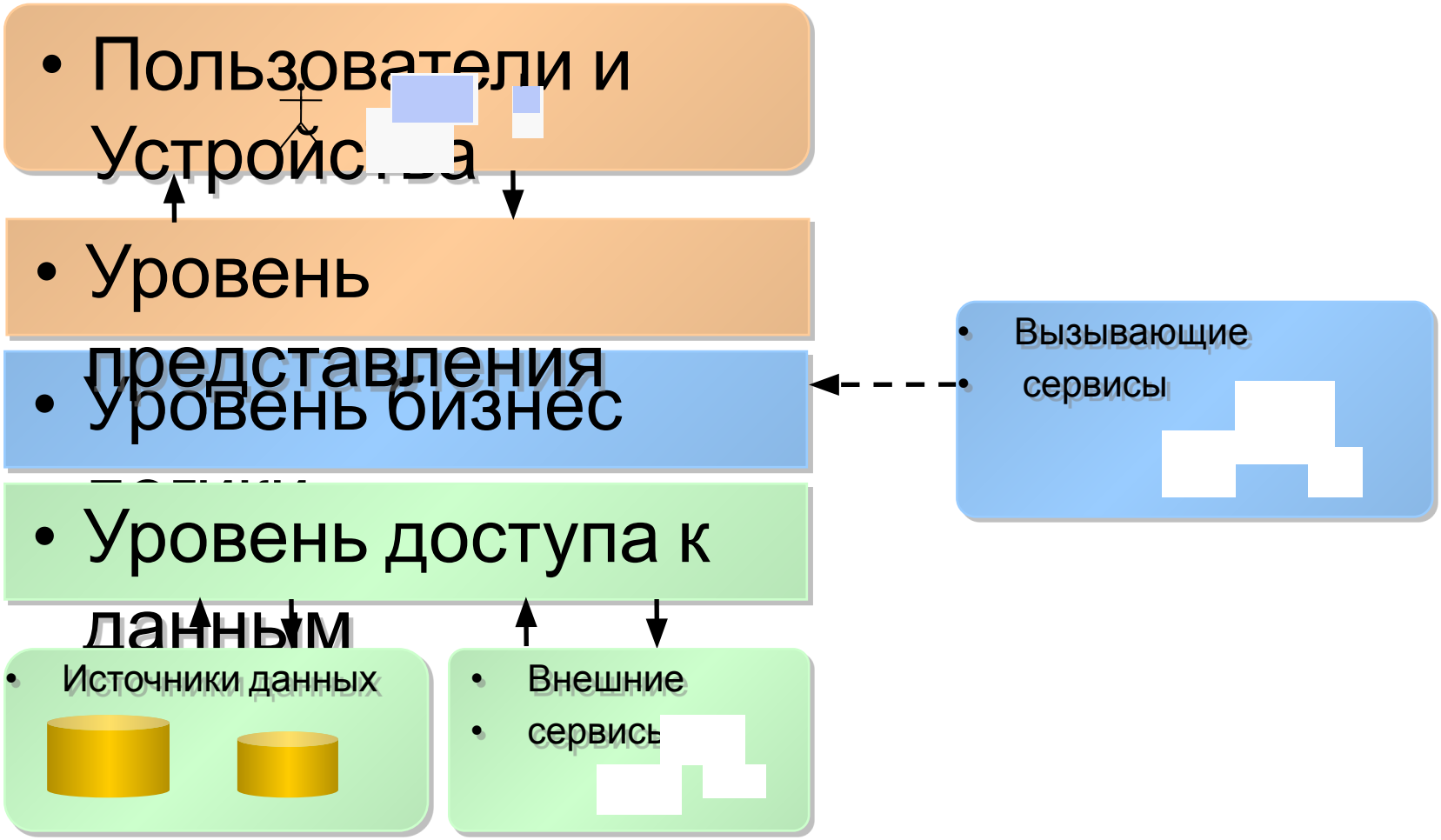
Общая архитектура .NET-приложений

- Среди начинающих разработчиков иногда имеется путаница между многослойной и многоуровневой архитектурой
- Иначе говоря, чем отличаются друг от друга слои и уровни?
- Приложения можно разбивать на части, получая физической и логическое представление
- В многоуровневой (N-tier) архитектуре мы делим код на разные сборки или наборы сборок физически
- У нас, например, может быть сборка для Web-проекта, в другая – для кода, обрабатывающего бизнес-объекты
- Если нам нужно разворачивать наше приложение на нескольких серверах, удаленных географически, то нам нужно использовать N-tier архитектуру

Общая архитектура .NET-приложений

- Разделение на слои означает, что мы логически разделяем код, а все приложение – это часть одной физической сборки
- Мы можем распределить код в разные папки с собственным пространством имен, но у нас не будет отдельной сборки для каждого пространства
- В отличие от физического представления мы не сможем распределенно разворачивать каждый слой по отдельности
- Иначе говоря, *tier* – это единица развертывания, а *layer* – это логическое распределение кода по ответственностям

Общая архитектура .NET-приложений



Общая архитектура .NET-приложений

- Когда мы говорим про многослойные приложения, у всех перед глазами всплывает стандартная схема, где имеются слои представлений, слой бизнес-логики, доступа к данным и собственно хранилище данных
- Это многим знакомая схема, но когда мы приступаем непосредственно к разработке и проектированию системы мы должны понимать, что за этими квадратиками скрывается, и как они взаимосвязаны

Общая архитектура .NET-приложений

- Так, уровень представления отвечает за форматирование, за представление данных пользователя на различных типах операционных систем различных типов пользовательских интерфейсов, отображение данных, получение ввода данных от пользователей, первоначальная валидация данных и т.д.
- Также к уровня представления можно отнести компоненты процесса пользовательского интерфейса

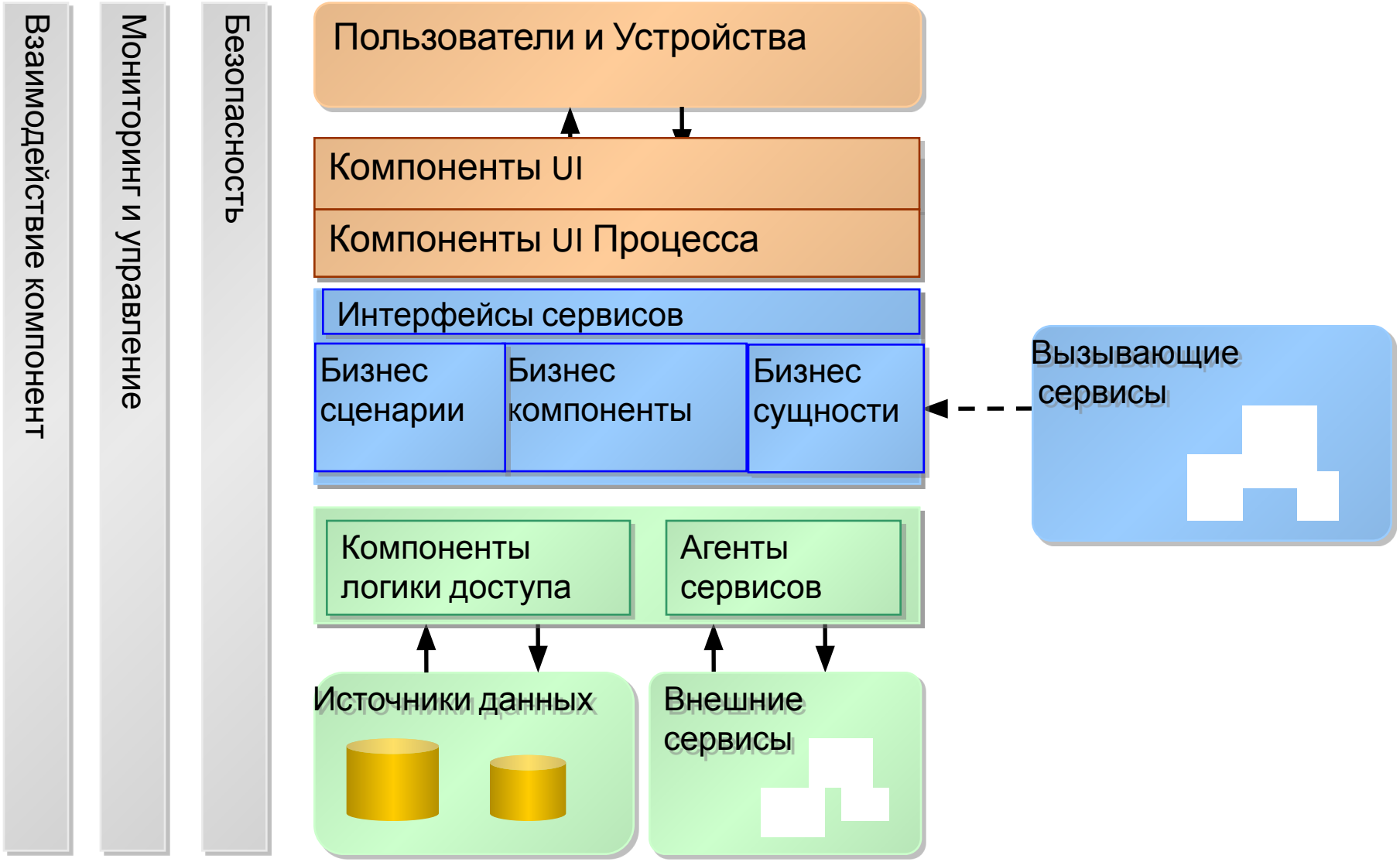
Общая архитектура .NET-приложений

- Если ввод данных в какой-то операции занимает у пользователя много времени, если он связан с логическими переходами между различными формами или страницами интерфейса, если необходимы операции возврата пользователя на предыдущие действия, хранения истории изменения его данных, то здесь необходимо выделять отдельно логику по форматированию и отображению данных от логики непосредственно процесса, связанного с вводом данных

Общая архитектура .NET-приложений

- Когда мы оформляем на веб-сайте заказ на покупку, мы проходим несколько шагов по вводу данных, и эти шаги завершаются еще до того, как начинает работать непосредственно бизнес-логика
- Связанное выполнение этих шагов должно лежать на компонентах процесса пользовательского интерфейса
- То есть, у нас есть несколько шагов заполнения адресных данных, данных, связанных с платежными реквизитами, данных, связанных с доставкой товара
- Эти последовательные шаги должны быть связаны через компоненты процесса пользовательского интерфейса

Общая архитектура .NET-приложений. Компоненты



Общая архитектура .NET-приложений

- Компонент пользовательского интерфейса отображает данные и фактически является концевой точкой связи с данными
- Важно при разработки компонент пользовательского интерфейса учитывать, что они никогда не должны участвовать в транзакции и могут быть связаны с компонентами процесса пользовательского интерфейса
- Компоненты процесса пользовательского интерфейса фактически осуществляют рендеринг данных для пользователя

Общая архитектура .NET-приложений

- Типичные примеры и технологии, используемые для отображения пользовательских компонент разбиваются на несколько категорий
- Первая категория это пользовательский интерфейс Windows, то есть полноценное доступное приложение, которое пользуется всеми возможностями локальных вычислительных ресурсов для работы с пользователем

Общая архитектура .NET-приложений

- Здесь есть несколько возможностей
- Мы строим приложение с Windows Forms, пользуемся базовыми классами .Net Framework
- Можем использовать в своем приложении внедренный HTML, когда помимо Windows Forms мы позволяем отображать в нашем приложении и HTML страницы, включая и страницы со скриптовыми элементами и можем реализовывать интерфейс в виде дополнительных модулей к другим приложениям

Общая архитектура .NET-приложений

- Есть несколько рекомендаций для разработки компонент ориентированных на создание поиска интерфейса Windows:
  - использовать связанные элементы управления данных пользователя (это позволяет легко синхронизировать данные между различными панелями отображения)
  - использовать один источник данных, связанный с различными контроллерами (это позволяет разработчику абстрагироваться от проблем синхронизации введенных пользователем данных с объектом, хранящим эти данные)

Общая архитектура .NET-приложений

- Другая категория пользовательского интерфейса – Web-Interface
- Здесь мы пользуемся технологией ASP .Net
- В сам ASP.Net входит набор элементов управления, который позволяет строить Web-Interface, но с другой стороны, позволяет пользоваться нам событийной парадигмой программирования

Общая архитектура .NET-приложений

- Предположим, нам предстоит разработать простую гостевую книгу для нашего сайта
- Один из сценариев так же достаточно прост: пользователь заходит на страничку, нажимает на кнопку с надписью «Показать все записи», а система запрашивает у СУБД и отображает все записи из книги
- При разработке системы мы должны создать Web-форму с кнопкой и кодом для получения всех записей из БД
- Код размещается внутри ASPX-формы
- Затем решение может быть откомпилировано в dll-сборку

Общая архитектура .NET-приложений

- Альтернатива – отделить код (например,C#) от разметки
- Далее можно отделить бизнес-логику от пользовательского интерфейса, например, в виде двух разных библиотек классов
- Это мы продемонстрируем в дальнейшем

Общая архитектура .NET-приложений

- В WinBased-проекте, который также известен как толстый клиент, многоуровневая архитектура может иметь вид:
  - WindowsForms или WPF – слой презентации (UI или Presentation layer)
  - C#-код для управления бизнес-логикой (BLL)
  - Код доступа к данным (возможно, на C#) - DAL
  - Реальная СУБД – хранилище данных – DataLayer

Общая архитектура .NET-приложений

- DAL – слой доступа к данным – набор классов, инкапсулирующих методы доступа к данным, типа создания, чтения, изменения удаления (CRUD), и другие операции над данными, находящимися в хранилище данных (*data store*, *data layer*)
- Иначе, главная задача DAL – это связь с хранилищем (DL)
- В качестве DL мы можем использовать СУБД, один или несколько XML- или текстовых файлов и т.д.

Общая архитектура .NET-приложений

- DAL должен действовать как «немой слой» для уровня BLL или других сервисов/служб
- DAL не должен содержать специфичную логику в своих классах
- Он должен использоваться как утилита или помощник (*helper class*) для загрузки и сохранения данных из/в хранилище данных

Общая архитектура .NET-приложений

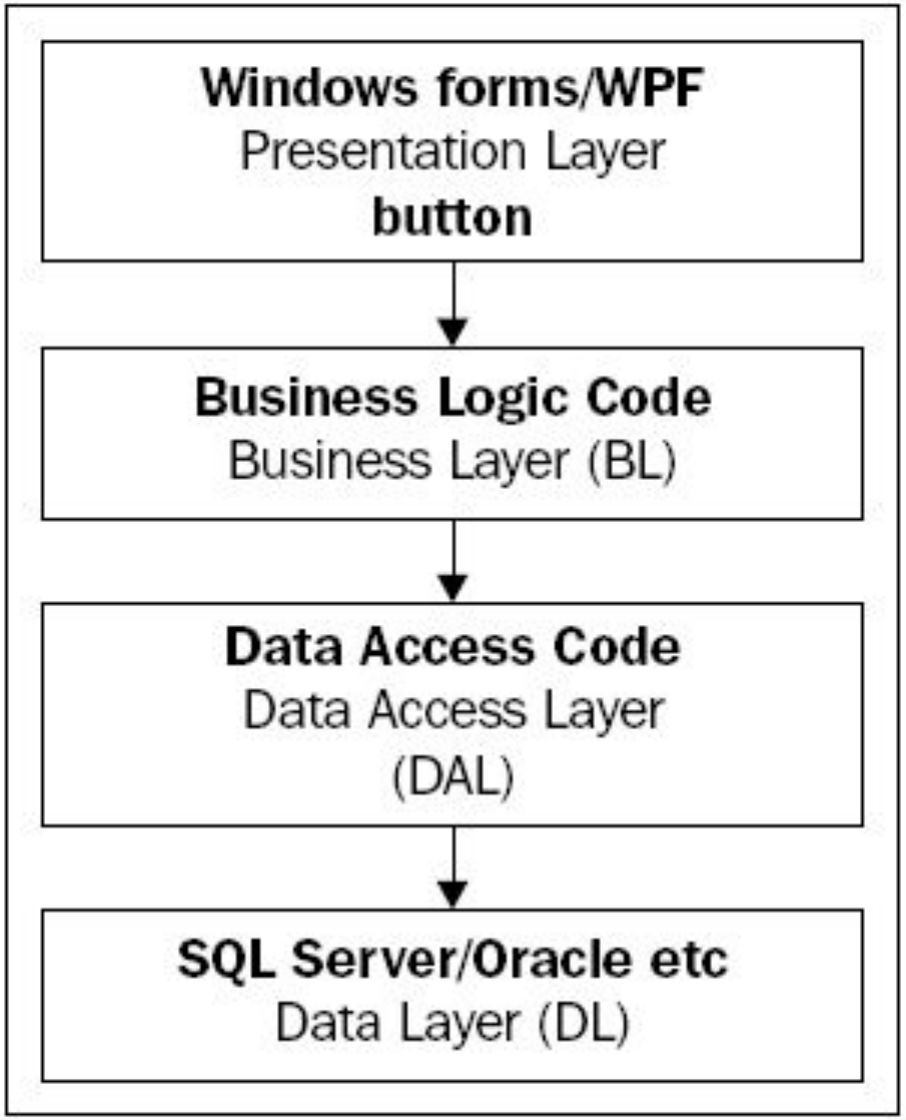
- Слой бизнес-логики (или BLL) содержит бизнес-логику и набор правил, специфичных для приложения, и «говорит» слою DAL:
  - Загрузить данные, к которым он применит правила;
  - Сохранить данные после того, как к ним были применены правила
  - Выполнить операции и проверить данные
- Слой BLL обычно предоставляет данные вышележащим уровням (например, слой GUI) после применения к ним бизнес-правил
- Помимо этого, BLL может содержать обработку ошибок, логирование, стратегии обработки исключительных ситуаций

Общая архитектура .NET-приложений

- Слой UI содержит графические компоненты и файлы типа ASPX, ASCX, MasterPages, таблицы стилей и т.д.
- Как правило, тип проекта – Website или Web Project в решении Visual Studio solution для технологии ASP.NET
- Иногда путают слои DataLayer и DAL
- Это не одно и то же, т.к. DAL – код для доступа к данным, а DataLayer – это сама база данных

Общая архитектура .NET-приложений

- На рисунке показано, как слои взаимодействуют друг с другом



Общая архитектура .NET-приложений

- Если мы отделим код каждого слоя и разместим каждый из них в собственном проекте и библиотеке классов, то мы получим 4-уровневый проект: Presentation tier, BL tier, DAL tier и DL tier (физическая БД)
- Однако, для web-based приложений мы имеем 3-уровневую архитектуру по умолчанию
  - Presentation tier – браузер клиента вместо Windows forms)
  - код web forms, BL и DAL в одной сборке – это Application tier
  - физическая БД как Data tier
- Это архитектура с «тонким клиентом»

Общая архитектура .NET-приложений

- Слой бизнес-логики (BLL) обычно включает следующие компоненты:
  - ***Фасад приложения*** - необязательный компонент , который обеспечивает упрощенный интерфейс для компонентов бизнес-логики, часто объединяя множество бизнес-операций, что упрощает использование бизнес-логики. Это сокращает количество зависимостей, потому что вызывающим сторонам извне нет необходимости знать детали компонентов бизнес-слоя и отношения между ними

Общая архитектура .NET-приложений

- Слой бизнес-логики (BLL) обычно включает следующие компоненты:
  - **Компоненты бизнес-логики** отвечают за извлечение, обработку, преобразование и управление данными приложения; применение бизнес-правил и политик и обеспечение непротиворечивости и актуальность данных. Чтобы создать наилучшие условия для переиспользования, компоненты BLL не должны содержать поведения или логики приложения конкретного варианта использования или пользовательской истории

Общая архитектура .NET-приложений

- Компоненты бизнес-логики можно подразделить на две категории:
  - **Компоненты бизнес-процесса**. После того, как компоненты UI получили необходимые данные от пользователя и передали их в BLL, приложение может использовать эти данные для осуществления бизнес-процесса. Большинство бизнес-процессов включают множество упорядоченных этапов, которые могут взаимодействовать друг с другом через различные механизмы координирования. Компоненты определяют и координируют долгосрочные бизнес-процессы и могут быть реализованы с помощью различных инструментов. Они работают с компонентами бизнес-процесса, которые создают экземпляры и осуществляют операции с компонентами рабочего процесса

Общая архитектура .NET-приложений

- Компоненты бизнес-логики можно подразделить на две категории:
  - **Компоненты бизнес-объектов** инкапсулируют бизнес-логику и данные, необходимые для представления в приложении объектов реального мира, таких как заказчики (Customers) или заказы (Orders). Они хранят значения данных и предоставляют их через свойства; содержат бизнес-данные приложения и управляют ими; предоставляют программный доступ с сохранением состояния к бизнес-данным и связанной функциональности. Бизнес-объекты также проверяют данные, содержащиеся в сущности; они инкапсулируют бизнес-логику для обеспечения непротиворечивости данных, а также реализации бизнес-правил и поведения.

Общая архитектура .NET-приложений

- При проектировании BLL стоит задача максимально упростить приложение путем разделения задач на разные функциональные области
- Например, логика обработки бизнес-правил, бизнес-процессов и бизнес-объектов – все они представляют разные функциональные области
- В рамках каждой области проектируемые компоненты должны быть сфокусированы на решении конкретной задачи и не должны включать код, связанный с другими функциональными областями.

Общая архитектура .NET-приложений

- Основные паттерны проектирования для BLL организованы по категориям и представлены далее
- Компоненты BLL обычно строят с применением паттернов:
  - *Application Façade* (*Фасад приложения*), централизующий и агрегирующий поведение для обеспечения унифицированного слоя сервисов
  - *Chain of Responsibility* (*Цепочка обязанностей*) устраняет возможность связывания отправителя запроса с его получателем
  - *Command* (*Команда*) инкапсулирует обработку запроса в отдельный командный объект с общим интерфейсом выполнения

Общая архитектура .NET-приложений

- Компоненты бизнес-объектов обычно строят с применением паттернов:
  - *Domain Model* (*Модель предметной области*) - набор бизнес-объектов, представляющих сущности предметной области и отношения между ними
  - *Entity Translator* (*Транслятор сущностей*) - объект, преобразующий типы данных сообщения в бизнес-типы для запросов и выполняющий обратные преобразования для ответов
  - *Table Module* (*Модуль таблицы*) - единый компонент, реализующий бизнес-логику для всех строк таблицы или представления базы данных

Общая архитектура .NET-приложений

- Компоненты рабочих процессов обычно строят с применением паттернов:
  - *Data-Driven Workflow* (*Управляемый данными рабочий процесс*) - включает задачи, последовательность выполнения которых определяется значениями данных
  - *Human Workflow* (*Рабочий процесс, управляемый оператором*) включает задачи, выполняемые вручную
  - *Sequential Workflow* (*Последовательный рабочий процесс*) включает задачи, выполняющиеся в определенной последовательности, когда выполнение одной задачи запускается после завершения предыдущей
  - *State-Driven Workflow* (*Управляемый состоянием*

Общая архитектура .NET-приложений

- Компоненты рабочих процессов обычно строят с применением паттернов:
  - *Data-Driven Workflow* (*Управляемый данными рабочий процесс*) - включает задачи, последовательность выполнения которых определяется значениями данных
  - *Human Workflow* (*Рабочий процесс, управляемый оператором*) включает задачи, выполняемые вручную
  - *Sequential Workflow* (*Последовательный рабочий процесс*) включает задачи, выполняющиеся в определенной последовательности, когда выполнение одной задачи запускается после завершения предыдущей
  - *State-Driven Workflow* (*Управляемый состоянием*

Общая архитектура .NET-приложений

- То, какие компоненты BLL будут использоваться для обработки запросов, определяет общий дизайн и тип создаваемого приложения
- Например, компоненты BLL Web-приложения обычно работают с основанными на сообщениях запросами
- Приложение Windows Forms обычно взаимодействует с компонентами BLL напрямую с помощью основанных на событиях запросов
- Кроме того, существуют и другие факторы, которые необходимо учесть при работе с разными типами приложений

Общая архитектура .NET-приложений

- Некоторые из этих факторов являются общими для различных типов, тогда как некоторые характерны лишь для конкретного типа приложений
- Рассмотрим ключевые решения, которые должны быть приняты при проектировании компонентов BLL
- Компоненты BLL будут размещаться на клиенте, на сервере приложений или и там, и там?

Общая архитектура .NET-приложений

- Часть или все компоненты BLL размещаются на клиенте, если создается изолированный насыщенный клиент или RIA-приложение, если хочется улучшить производительность, либо если используется модель предметной области
- Часть или все компоненты BLL размещаются на сервере приложений, если общая бизнес-логика должна поддерживать множество типов клиентов, если компоненты BLL требуют доступа к ресурсам, недоступным с клиента, или по соображениям безопасности

Общая архитектура .NET-приложений

- Если компоненты BLL и PresentationLayer размещаются на одном уровне, то нужно использовать компонентные взаимодействия через события и методы
- Это обеспечивает максимальную производительность

Общая архитектура .NET-приложений

- **Однако**: реализуется интерфейс сервиса и используется взаимодействие посредством обмена сообщениями между слоем представления и компонентами BLL, если его компоненты и Web-сервер располагаются на разных уровнях;
- Или если разрабатывается Web-приложение со слабым связыванием между слоем представления и BLL;
- Или при наличии насыщенного клиента или приложения RIA

Общая архитектура .NET-приложений

- Если насыщенное клиентское приложение или RIA подключаются к серверу приложений или Web-серверу лишь изредка, необходимо внимательно подойти к вопросу проектирования интерфейса сервиса, который должен обеспечивать повторную синхронизацию клиента при подключении

Общая архитектура .NET-приложений

- Бизнес-объекты хранят значения данных и предоставляют их через свойства
- Они содержат и управляют бизнес-данными, которые используются приложением, и обеспечивают программный доступ с сохранением состояния к бизнес-данным и связанной функциональности

Общая архитектура .NET-приложений

- Бизнес-объекты также должны проводить проверку содержащихся в них данных и инкапсулировать бизнес-логику для обеспечения непротиворечивости и реализации бизнес-правил и необходимого поведения
- Таким образом, правильное проектирование или выбор бизнес-объектов жизненно важны для обеспечения наилучшей производительности и эффективности BLL

Общая архитектура .NET-приложений

- Существуют различные способы представления бизнес-объектов
- У каждого из них есть свои преимущества и недостатки
- Перечислим наиболее распространенные форматы:
  - Собственные бизнес-объекты
  - DataSet или DataTable
  - XML

Общая архитектура .NET-приложений

- **Собственные бизнес-объекты** – это объекты общеязыковой среды выполнения (CLR), описывающие сущности системы
- Для создания этих объектов может использоваться технология объектно-реляционного проецирования (O/RM), такая как ADO.NET Entity Framework (EF) или NHibernate, или их можно создавать вручную

Общая архитектура .NET-приложений

- Собственные бизнес-объекты подходят в случаях, когда требуется инкапсулировать сложные бизнес-правила или поведение вместе со связанными с ними данными
- Если требуется выполнять доступ к бизнес-объектам через границы *AppDomain* (*Домен приложения*), процесса или физические границы, можно реализовать слой сервисов, который будет обеспечивать доступ посредством *объектов передачи данных* (DTO) и операций обновления или редактирования бизнес-объектов

Общая архитектура .NET-приложений

- **Объекты DataSet** – это разновидность БД в памяти, которая обычно очень близко соответствует фактической схеме БД
- Объекты DataSet, как правило, используются только, если не используется механизм O/RM и создается объектно-ориентированное приложение, в котором данные логики приложения очень близко соответствуют схеме БД

Общая архитектура .NET-приложений

- DataSet не может расширяться для инкапсуляции бизнес-логики или бизнес-правил
- Несмотря на то, что DataSet могут быть сериализованы в XML, к ним не должен предоставляться доступ через границы процесса или сервиса

Общая архитектура .NET-приложений

- **XML** - это основанный на стандартах формат для организации структурированных данных
- XML обычно используется для представления бизнес-объектов, только если этого требует слой представления, или если логика должна работать с содержимым, исходя из его схемы
- Например, система маршрутизации сообщений, логика которой направляет сообщения на основании некоторых известных элементов XML-документа
- Использование и обработка XML может требовать больших объемов памяти.

Общая архитектура .NET-приложений

- Если принято решение о том, что собственные объекты обеспечат наилучшее представление бизнес-объектов, следующим шагом является проектирование этих объектов
- Подход, применяемый к проектированию собственных объектов, зависит от сорта объекта, который планируется использовать
- Например, сущности модели предметной области требуют глубокого анализа предметной области, тогда как сущности модуля таблицы требуют понимания схемы БД

Общая архитектура .NET-приложений

- Рассмотрим общие подходы к проектированию при использовании бизнес-объектов
- *Модель предметной области* (*Domain Model*) – объектно-ориентированный паттерн проектирования
- Цель проектирования – определение бизнес-объектов, которые представляют реальные сущности предметной области
- Бизнес-объекты и сущности предметной области включают и поведение, и структуру
- Иначе говоря, бизнес-правила и отношения инкапсулированы в модели предметной области

Общая архитектура .NET-приложений

- Проектирование предметной области требует глубокого анализа предметной области и, как правило, не сопоставляется с реляционными моделями, используемыми большинством БД
- Эту модель принято применять, если предметная область содержит сложные бизнес-правила или создается насыщенный клиент, и модель может инициализироваться и удерживаться в памяти
- Модель предметной области не может применяться при работе с BLL, не сохраняющим состояние, который требует инициализации модели предметной области при каждом

Общая архитектура .NET-приложений

- *Модуль таблицы* (*Table Module*) – это объектно-ориентированный паттерн проектирования
- Цель проектирования модуля таблицы – определение сущностей на основании таблиц или представлений базы данных
- Операции, используемые для доступа к базе данных и заполнения сущностей модуля таблицы, обычно инкапсулированы в сущности

Общая архитектура .NET-приложений

- Однако для осуществления операций с базой данных и заполнения сущностей модуля таблицы могут также использоваться компоненты доступа к данным
- Модуль таблицы применяется, если таблицы или представления базы данных достаточно точно представляют бизнес-объекты, используемые приложением, или если бизнес-логика и операции касаются одной таблицы или представления

Общая архитектура .NET-приложений

- *Специальные XML-объекты* (*Custom XML objects*) представляют десериализованные XML-данные, которые могут обрабатываться кодом приложения
- Объекты являются экземплярами классов, определяемыми атрибутами, которые сопоставляют свойства классов с элементами и атрибутами XML-структуры
- .NET Framework предоставляет компоненты, которые могут использоваться для десериализации XML-данных в объекты и сериализации объектов в XML

Общая архитектура .NET-приложений

- Этот подход рекомендуется использовать:
- если данные уже поступают в XML-формате (например, XML-файлы или операции базы данных, возвращающие XML как результирующее множество);
- если требуется формировать XML-данные из источника не XML-данных;
- или при работе с документами, доступными только для чтения.

Общая архитектура .NET-приложений

- Слой доступа к данным может включать следующие компоненты:
  - Компоненты доступа к данным
  - Агенты сервисов
- **Компоненты доступа к данным** абстрагируют логику, необходимую для доступа к базовым хранилищам данных
- Они обеспечивают централизацию общей функциональности доступа к данным, что способствует упрощению настройки и обслуживания приложения

Общая архитектура .NET-приложений

- Некоторые инфраструктуры доступа к данным могут требовать, чтобы общая логика доступа к данным была определена и реализована в отдельных вспомогательных или служебных компонентах доступа к данным, пригодных для повторного использования
- Другие инфраструктуры доступа к данным, включая многие инфраструктуры O/RM, реализуют такие компоненты автоматически, сокращая объем кода доступа к данным, который должен написать разработчик

Общая архитектура .NET-приложений

- Если компонент BLL должен выполнять доступ к данным от внешнего сервиса, может понадобиться реализовать код управления семантикой взаимодействия с этим конкретным сервисом
- **Агенты сервисов** реализуют компоненты доступа к данным, которые изолируют меняющиеся требования вызова сервисов от приложения и могут обеспечивать дополнительные сервисы, такие как кэширование, поддержку работы в автономном режиме и базовое преобразование между форматом данных, предоставляемых

Общая архитектура .NET-приложений

- При выборе технологии доступа к данным необходимо учесть тип данных, с которыми предполагается работать, и то, как эти данные будут обрабатываться в приложении
- Для каждого конкретного сценария есть наиболее подходящие технологии
- Чтобы правильно выбрать технологию доступа к данным, соответствующую сценариям создаваемого приложения, можно руководствоваться следующими рекомендациями

Общая архитектура .NET-приложений

- **ADO.NET Entity Framework** (*EF*) можно использовать:
- когда нужно создать модель данных и соотнести ее с реляционной БД;
- соотнести один класс с множеством таблиц, используя наследование;
- или выполнять запросы к реляционным хранилищам, не входящим в семейство продуктов Microsoft SQL Server

Общая архитектура .NET-приложений

- **ADO.NET Data Services** построена на базе EF и позволяет предоставлять части модели сущностей (Entity Model) через REST-интерфейс
- Эту технлогию можно использовать, если разрабатывается RIA или n-уровневое насыщенное клиентское приложение и требуется обеспечить доступ к данным через ресурсно-ориентированный интерфейс сервиса

Общая архитектура .NET-приложений

- **ADO.NET Core** используется:
- если для обеспечения полного управления доступом к данным в приложении необходим низкоуровневый API;
- если нужно использовать уже сделанные инвестиции в ADO.NET-решения;
- если используется традиционную логику доступа к данным.
- если нет необходимости в дополнительной функциональности, как в других технологиях, или если приложение должно поддерживать сценарии доступа к данным без постоянного подключения

Общая архитектура .NET-приложений

- Рекомендуется использовать **ADO.NET Sync Services** при проектировании приложения, которое должно поддерживать сценарии без постоянного подключения или требует синхронизации БД
- Если приложение работает с XML-данными, запросы к которым необходимо выполнять, применяя синтаксис LINQ, рекомендуется использовать **LINQ to XML**

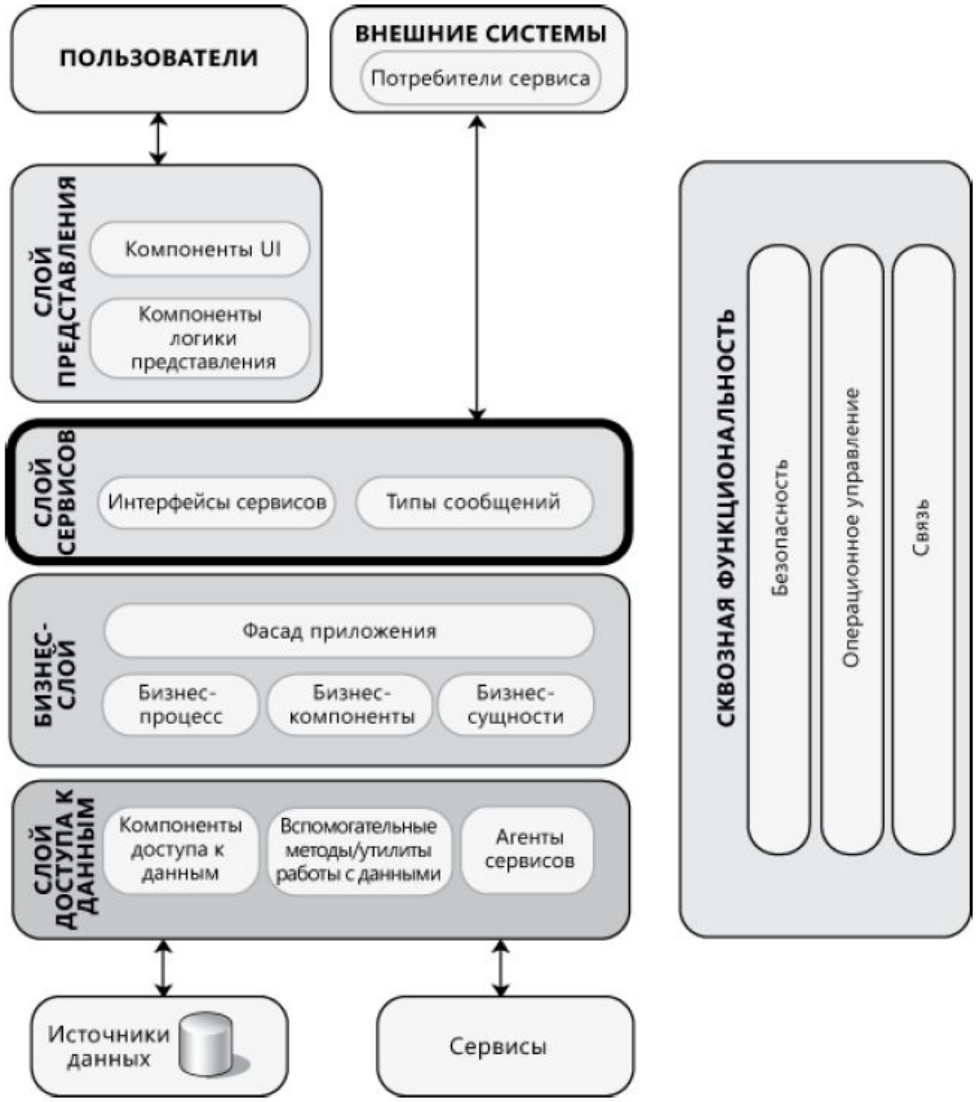
Общая архитектура .NET-приложений

- При предоставлении доступа к функциональности приложения через сервисы функции сервиса должны быть выделены в отдельный *слой сервисов*
- В слое сервисов определяется и реализуется интерфейс сервиса и контракты данных (или типы сообщений)
- Одним из самых важных моментов, о котором нельзя забывать, является то, что сервис никогда не должен раскрывать детали внутренних процессов или бизнес-объектов, используемых в приложении

Общая архитектура .NET-приложений

- В частности, необходимо гарантировать, что сущности BLL не будут оказывать большого влияния на контракты данных
- Слой сервисов должен предоставлять компоненты-трансляторы, которые будут обеспечивать преобразование форматов данных между сущностями BLL и контрактами данных
- На следующем слайде показано место слоя сервисов в общей архитектуре приложения

Общая архитектура .NET-приложений



Общая архитектура .NET-приложений

- Слой сервисов обычно включает следующие компоненты:
  - **Интерфейсы сервисов**. Сервисы предоставляют интерфейсы, в которые передаются все входящие сообщения. Интерфейс сервиса можно рассматривать как фасад, предоставляющий потенциальным потребителям доступ к BLL
  - **Типы сообщений**. При обмене данными через слой сервисов структуры данных заключаются в структуры сообщений, поддерживающие разные типы операций. Туда же обычно включают типы и контракты данных, которые определяют используемые в сообщениях типы данных

Общая архитектура .NET-приложений

- При проектировании слоя сервисов необходимо учесть множество факторов
- Многие из этих аспектов проектирования касаются проверенных практик, имеющих отношение к созданию многослойных архитектур
- Однако для сервисов необходимо рассматривать также и факторы, связанные с обменом сообщениями

Общая архитектура .NET-приложений

- Основное, на что требуется обратить внимание: сервисы взаимодействуют посредством обмена сообщениями, как правило, по сети, что по сути своей медленнее, чем прямое взаимодействие внутри процесса, и обычно сервисы взаимодействуют с потребителями асинхронно
- Сообщения могут быть маршрутизированы, изменены, доставлены не в том порядке, в котором они отправлялись, или потеряны, если не реализован механизм гарантированной доставки
- Все это требует архитектуры, где будет учтен такой недетерминированный обмен

Общая архитектура .NET-приложений

- При проектировании интерфейсов сервисов используются следующие паттерны проектирования:
- *Façade* (*Фасад*) реализует унифицированный интерфейс для набора операций, чтобы обеспечить упрощенный интерфейс и уменьшить связанность систем

Общая архитектура .NET-приложений

- При проектировании интерфейсов сервисов используются следующие паттерны проектирования:
- *Remote Façade* (*Удаленный фасад*) создает обобщенный унифицированный интерфейс для набора операций или процессов в удаленной подсистеме, обеспечивая слабо детализированный интерфейс для детализированных операций, чтобы упростить использование этой подсистемы и свести до минимума вызовы по сети

Общая архитектура .NET-приложений

- При проектировании интерфейсов сервисов используются следующие паттерны проектирования:
- *Service Interface* (*Интерфейс сервиса*) может использоваться другими системами для взаимодействия с сервисом

Общая архитектура .NET-приложений

- Microsoft предлагает две технологии обмена сообщениями: Windows Communication Foundation (WCF) и ASP.NET Web Services (ASMX)
- WCF обеспечивает полноценный механизм реализации сервисов для широкого диапазона ситуаций и обеспечивает возможность точного регулирования конфигурации и содержимого сервисов

Общая архитектура .NET-приложений

- WCF подходит в следующих случаях:
  - Для взаимодействия с Веб-сервисами, когда необходимо обеспечить возможность взаимодействия с другими платформами, которые также поддерживают SOAP
  - Для взаимодействия с Веб-сервисами, использующими не SOAP-сообщения, например, приложениями, работающими с такими форматами, как Really Simple Syndication (RSS).
  - Для взаимодействия посредством SOAP-сообщений и двоичного кодирования для структур данных, когда и сервер, и клиент используют WCF
  - Для создания сервисов REST Singleton и Collection Services, ATOM Feed и Publishing Protocol Services, а также HTTP Plain XML Services

Общая архитектура .NET-приложений

- ASMX обеспечивает более простое решение для создания Веб-сервисов на базе ASP.NET и их предоставления через Веб-сервер IIS. ASMX имеет следующие характеристики:
  - Обеспечивает возможность доступа через Интернет с использованием только протокола HTTP. По умолчанию использует порт 80, но изменить это не составляет никакого труда
  - Не поддерживает транзакций координатора распределенных транзакций (DTC). Для программирования длительных транзакций необходимо использовать собственные реализации

Общая архитектура .NET-приложений

- ASMX обеспечивает более простое решение для создания Веб-сервисов на базе ASP.NET и их предоставления через Веб-сервер IIS. ASMX имеет следующие характеристики:
  - Поддерживает аутентификацию IIS, роли, хранящиеся как группы Windows для авторизации, олицетворение IIS и ASP.NET и безопасность на транспортном уровне с использованием протокола SSL
  - Поддерживает технологию конечных точек, реализованную в IIS
  - Обеспечивает возможность межплатформенного взаимодействия

Использованные источники

- Руководство Microsoft по проектированию архитектуры приложений. 2е издание»
- Справочник «Паттерны проектирования» - http://design-pattern.ru
- Vivek Thakur. ASP.NET 3.5. Application Architecture and Design/ Packt Publishing, 2008
- http://martinfowler.com/eaaCatalog/
- http://www.intuit.ru/goto/course/mwebtech/