Архитектура ИС

Структурирование слоя бизнес-логики.

Основные определения

Предметная область — часть реального мира, рассматриваемая в пределах данного контекста. Под контекстом здесь может пониматься, например, область исследования или область, которая является объектом некоторой деятельности.

Основные определения

Бизнес-процесс (Business process) – это совокупность взаимосвязанных мероприятий или задач, направленных на создание определенного продукта или услуги для потребителей. Для наглядности бизнес-процессы визуализируют при помощи блок-схемы бизнес-процессов.

Учёт успеваемости, Согласование договора, Формирование бюджета, и т.п.,
Учёт нематериальных активов, Подготовка специалиста в вузе.

Основные определения

Бизнес-правило (БП) (Business rule) – правило, принятое в компании (бизнесе). Правило, которое определяет или ограничивает некоторый аспект бизнеса и при применении принимает значения Ложь или Истина. БП могут быть применены к людям, процессам, поведению компьютерных систем.

БП: Операция списания средств по кредитной карте не может быть проведена, если будет превышен допустимый овердрафт.

Дополнительные сведения здесь

Основные определения

Бизнес-логика (Business logic, Domain logic) – Совокупность бизнес-правил, принципов, зависимостей поведения объектов предметной области (области человеческой деятельности, которую система поддерживает).

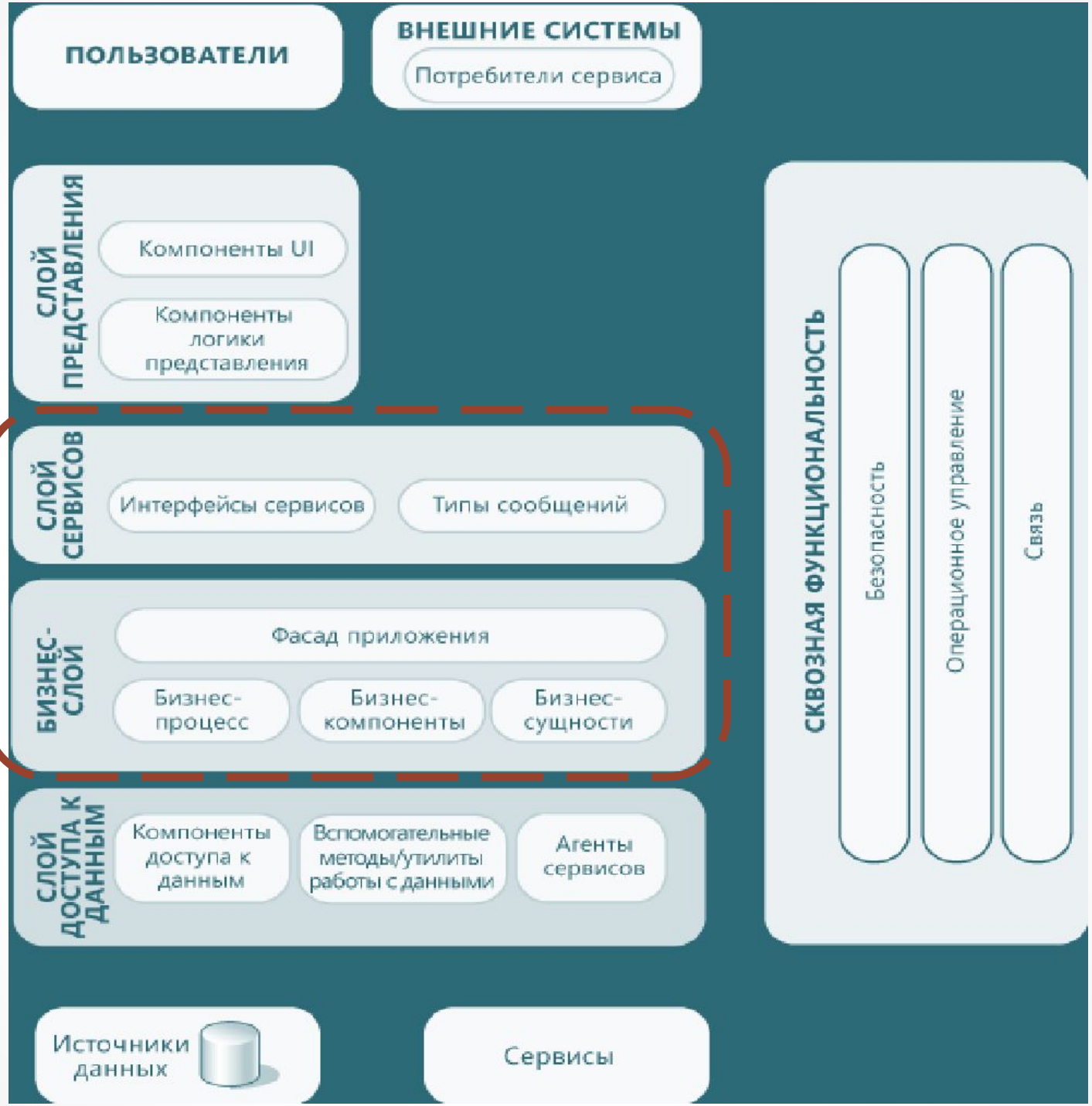
Основные определения

Бизнес-транзакция (Business transaction) –

Определение 1. Это взаимодействие между бизнесом (компанией) и его клиентами, вендорами и другими партнёрами.

Определение 2. Событие в экономике компании, в результате которого инициируется процесс учёта и производится запись данных о событии в информационной системе компании.

Регистрация Выбор счетов Указание суммы перевода Ввод подтверждающего кода (пришёл по SMS)



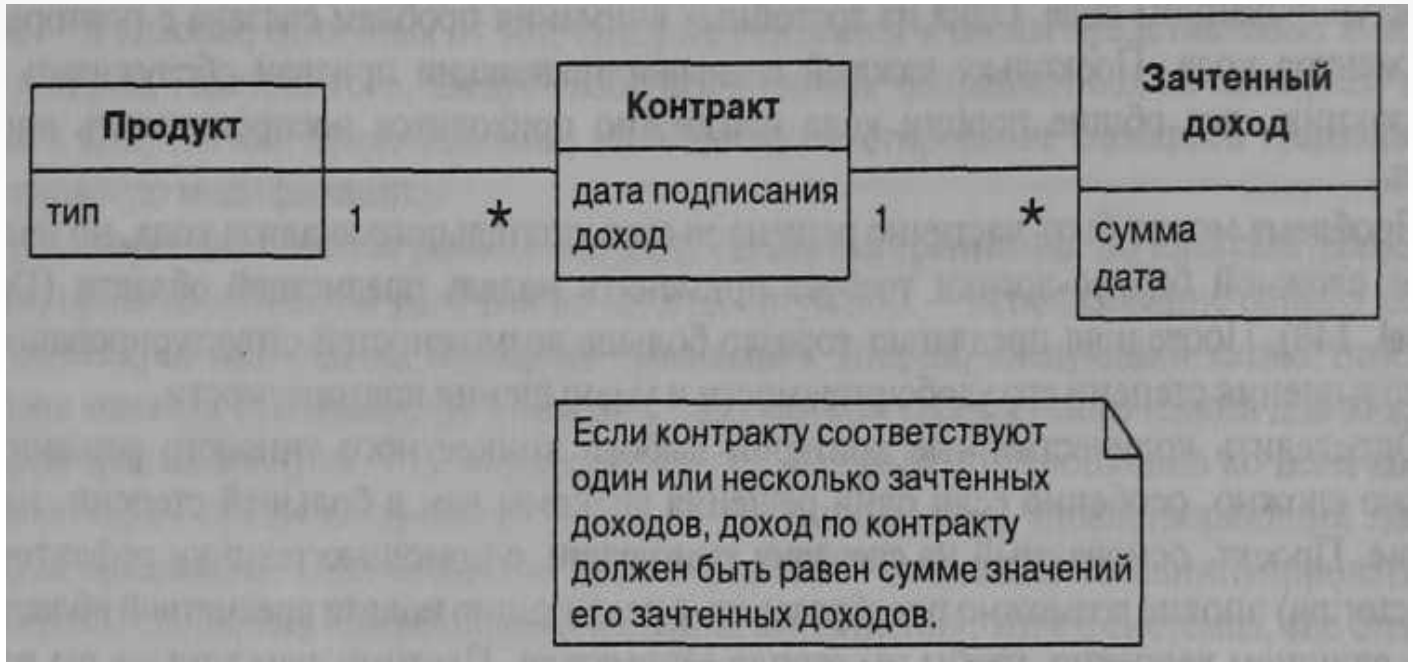
Слои информационной системы

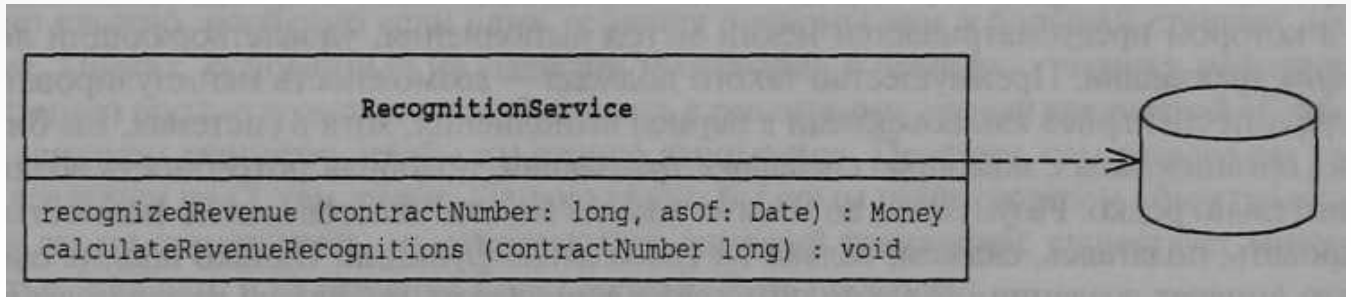
Источник картинки

Структурирование слоя БЛ по М. Фаулеру

- Сценарий транзакции
- Модуль таблицы
- Модель предметной области

Предметная область для примера

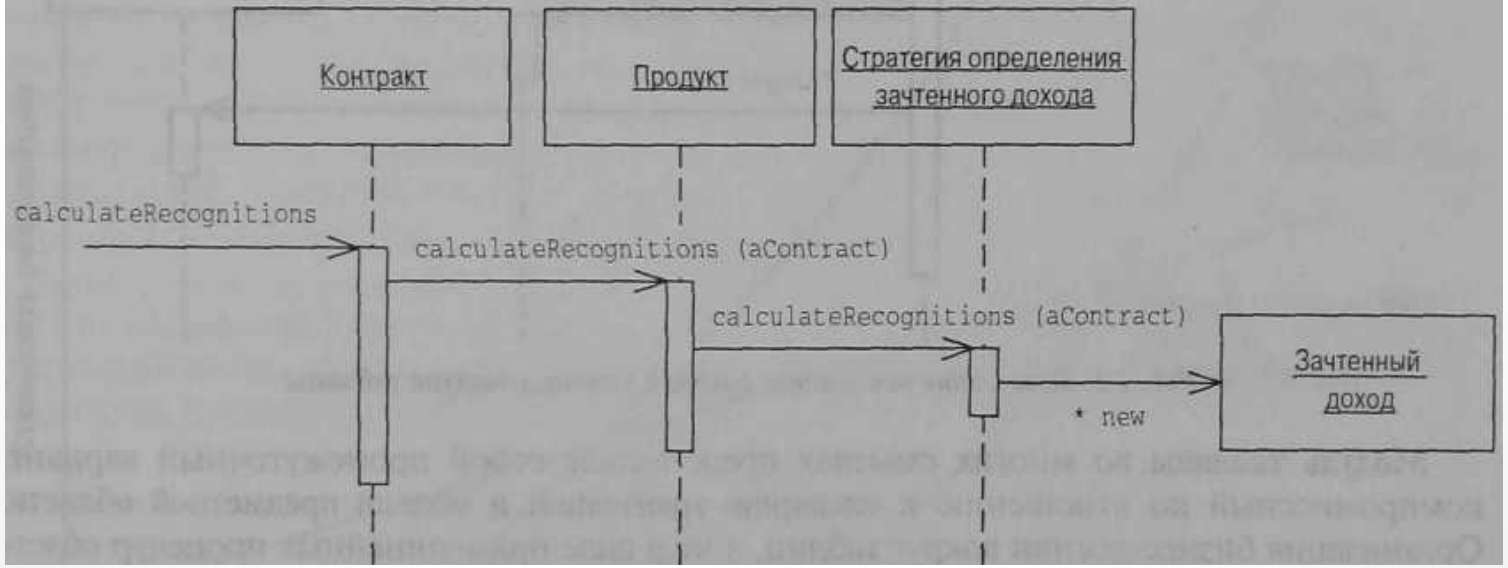




Сценарий транзакций

Способ организации бизнес-логики по процедурам, каждая из которых обслуживает один запрос, инициируемый споем представления

Рис. 2.1. Вычисление зачтенного дохода с помощью **сценария транзакции**

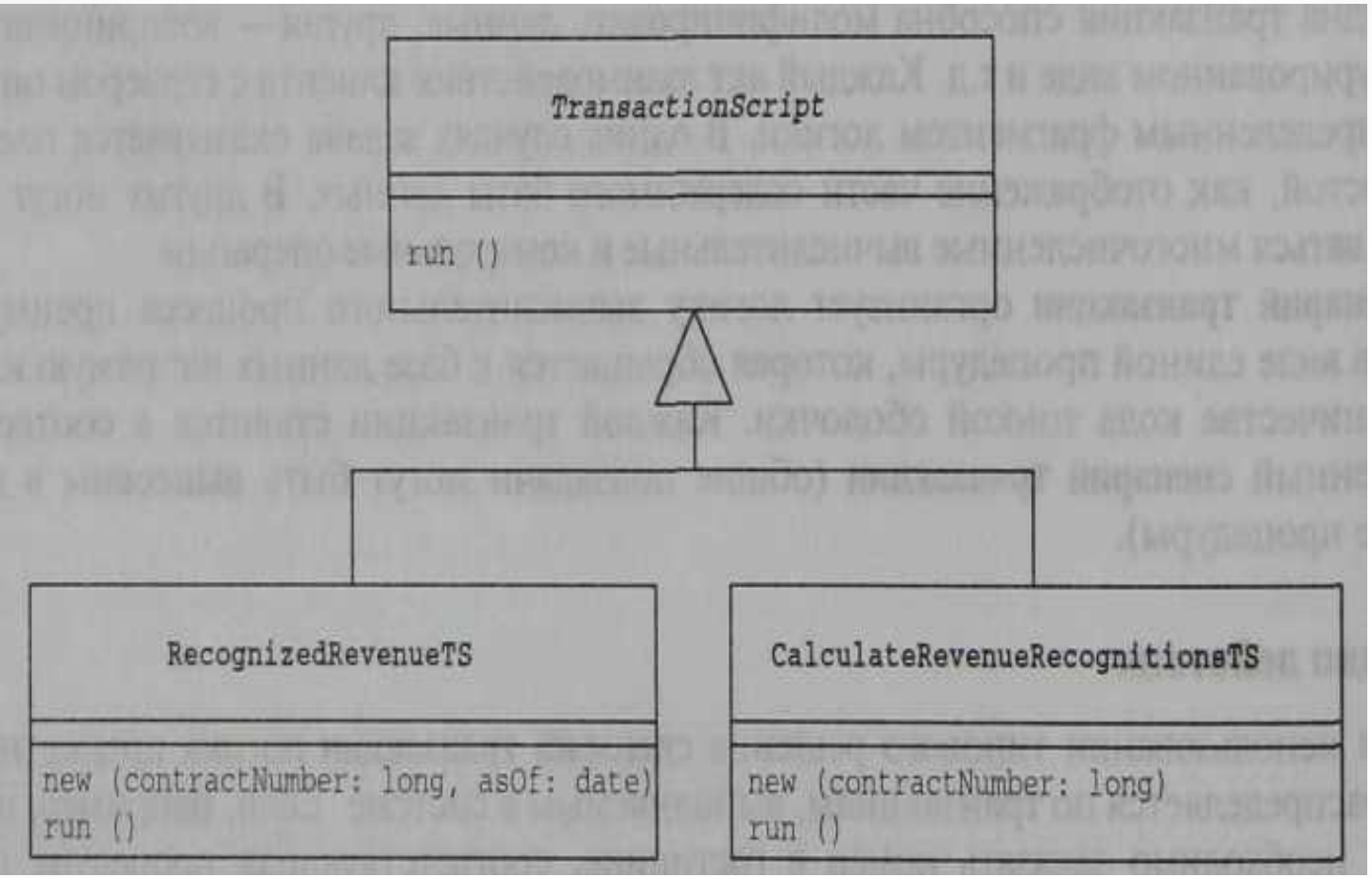


Пример сценария транзакций

Вычисление зачётного дохода

Пример

```
class RecognitionService
{
    private IDataGateway _dataGateway;
    public SystemController(IDataGateway dataGateway)
    {_dataGateway = dataGateway;}
      public void CalculateRecognitions(int contractID)
    {
            var contract = _dataGateway.findContract(contractID);
            var newRecognition = Calculate(contract);
      _dataGateway.InsertRecognition(contractID, newRecognition);
    }
}
```



Варианты реализации сценария транзакции

Источник картинки здесь

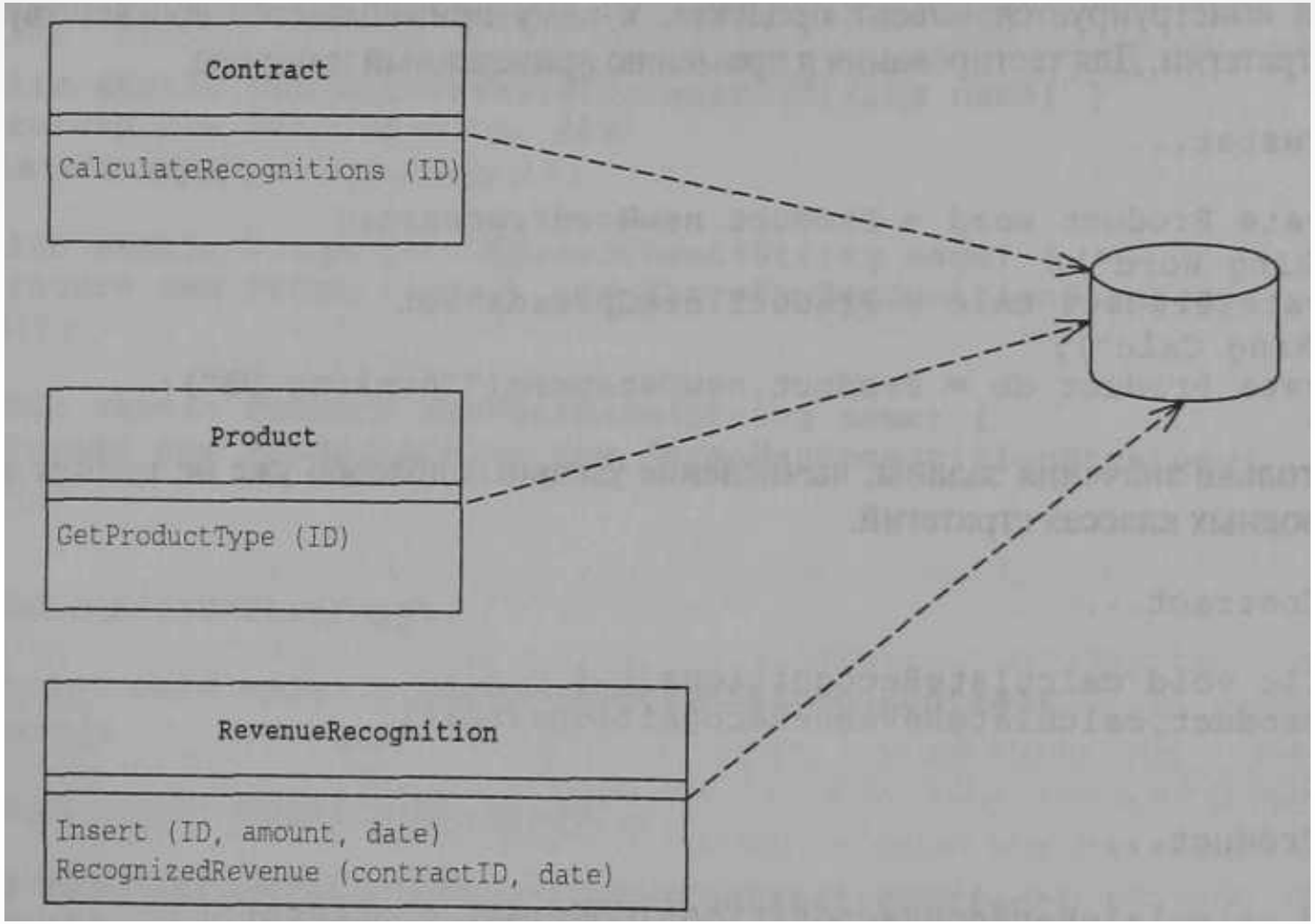
Сценарий транзакции. «За и против»

Преимущества:

1. представляет собой удобную процедурную модель, легко воспринимаемую всеми разработчиками;
2. удачно сочетается с простыми схемами организации слоя источника данных на основе типовых решений
3. определяет четкие границы транзакции.

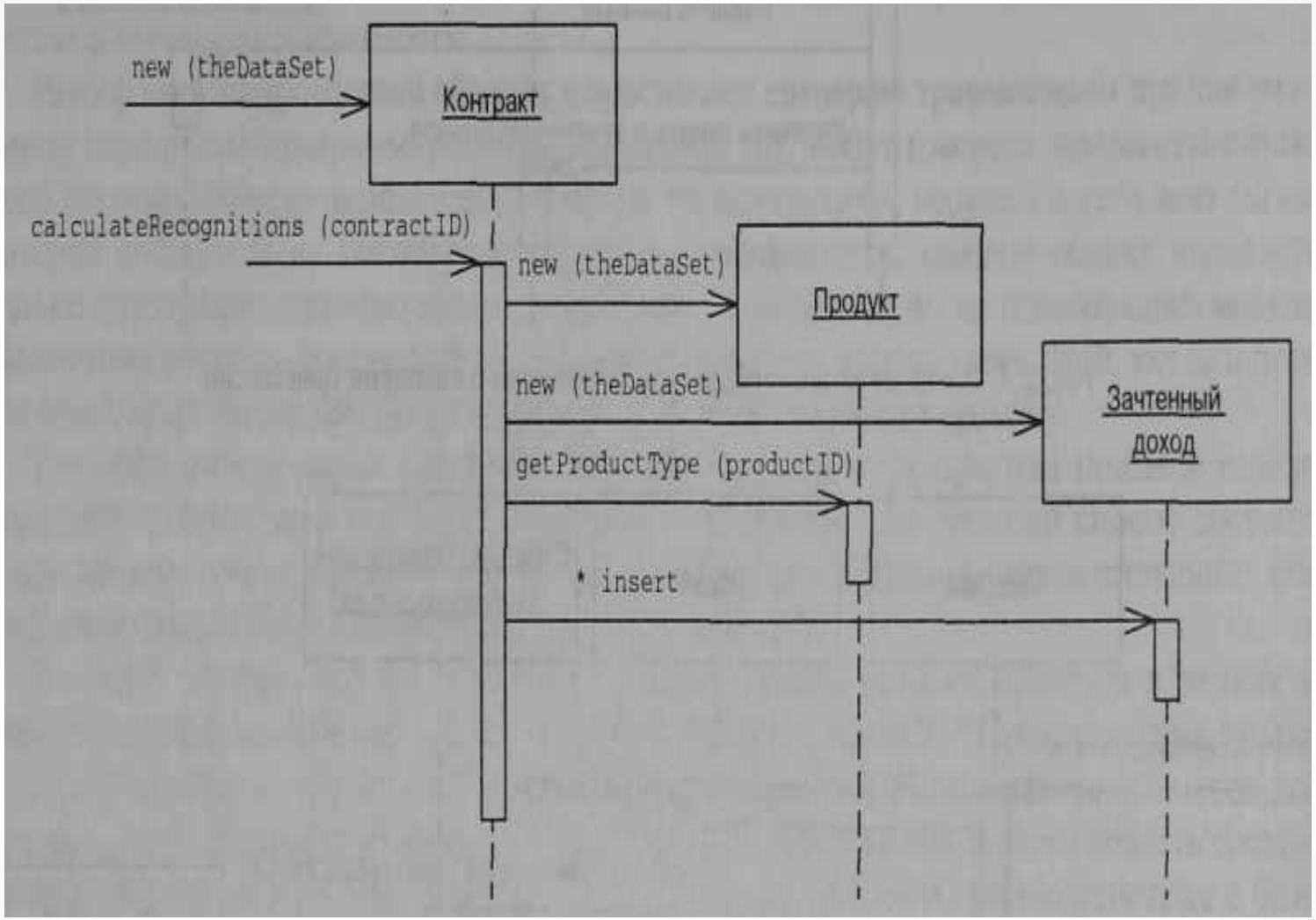
Недостатки:

1. При возрастании сложности системы существенно повышается дублирование кода в различных сценариях выполнения
2. Поддержка состояний на уровне слоя БЛ затруднена.
3. Низкая тестопригодность



Модуль таблицы

Объект, охватывающий логику обработки всех записей хранимой или виртуальной таблицы базы данных



Модуль таблицы

Источник картинки здесь

Пример

```
class Contracts : DataTable
{
    public void Insert( int ContractId, string contractor, …){…}
    public void Update( int ContractId, string newContractor, …) {…}
    public void Delete( int ContractId) {…}
    public bool Find( int ContractId) {…}
    public object GetValue( string columnName){…}
    public void SetValue( string columnName, object value){…}
    void CalculateRecognitions( int ContractId)
}
```

Пример

```
class   Contract...

public void CalculateRecognitions( int contractID)

{

      DataRow contractRow = this[contractID]; Decimal amount = (Decimal)contractRow["amount"]; RevenueRecognition rr = new RevenueRecognition( table.DataSet);

      Product prod = new Product(table.DataSet);

      int prodID = GetProductId(contractID);

      if (prod.GetProductType(prodID) == ProductType.WP) {

            rr.Insert(contractID, amount, (DateTime)GetWhenSigned( contractID) ) ;

      } else if (prod.GetProductType(prodID) == ProductType.SS)

     {

          Decimal![] allocation = allocate (amount, 3); rr.Insert(contractID, allocation[0],(DateTime)GetWhenSigned(contractID));

          rr.Insert(contractID, allocation[1], (DateTime)

         GetWhenSigned(contractID).AddDays(60)); rr.Insert(contractID, allocation [2], (DateTime)

         GetWhenSigned(contractID) .AddDays(90) ) ; } else if (prod.GetProductType(prodID) == ProductType.DB) {
```

Пример

```
        Decimal!] allocation = allocate(amount, 3); rr.Insert (contractID, allocation[0],
        (DateTime)GetWhenSigned(contractID) ) ;
        rr.Insert (contractID, allocation[1], (DateTime)
        GetWhenSigned(contractID).AddDays(30)); rr.Insert (contractID, allocation[2],
            (DateTime)
       GetWhenSigned(contractID).AddDays(60));
    }
   else throw new Exception("invalid product id");
}
```

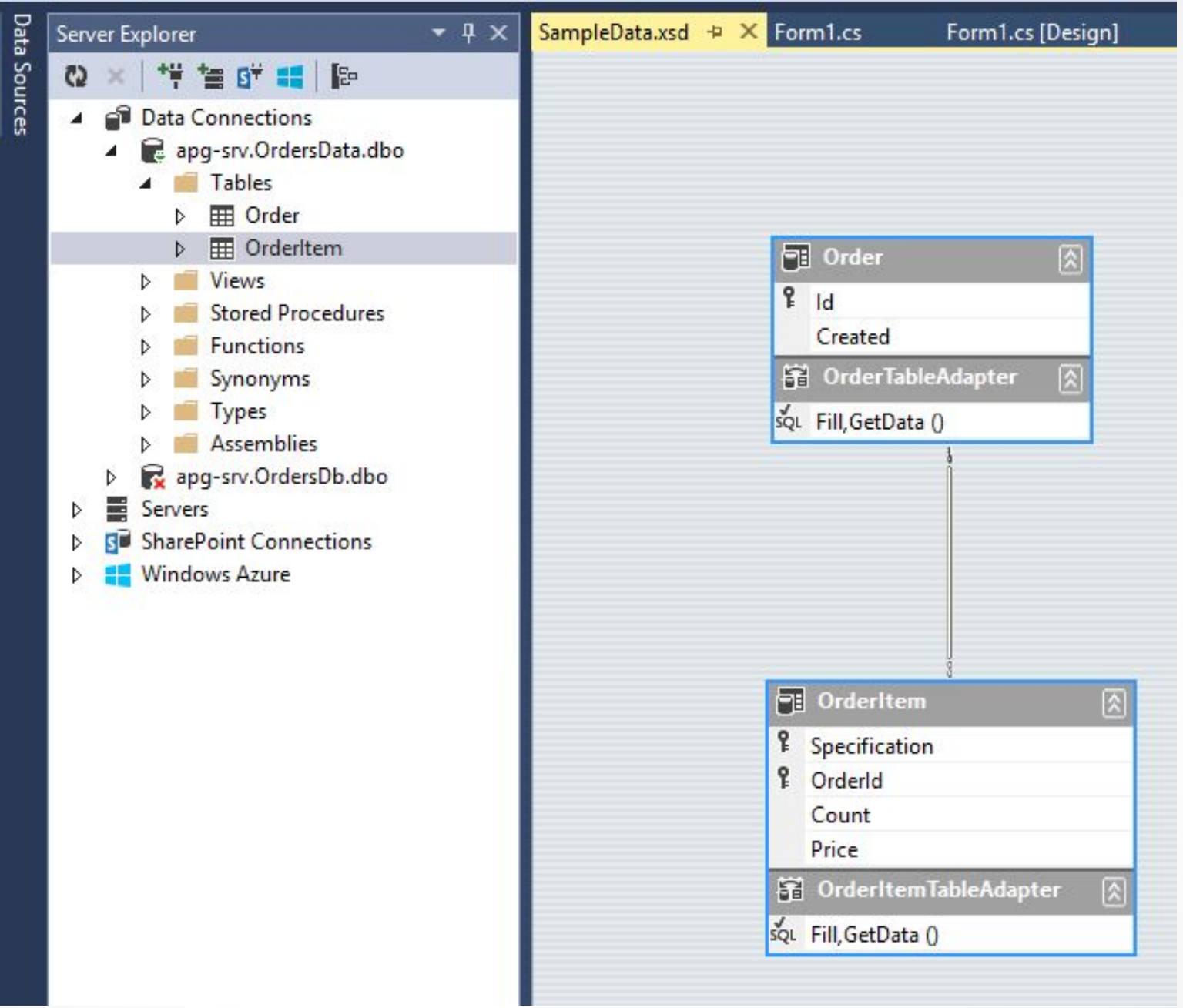
Модуль таблицы. «За и против»

Преимущества:

1. Представляет понятное решение – компромиссный вариант между сценарием транзакции и моделью предметной области
2. Представляет удобную абстракцию реляционного (или другой модели) хранилища на уровне слоя
3. Позволяет использовать некоторые объектно-ориентированные решения для декомпозиции системы в соответствие с моделью базы данных
4. Поддерживается многими средами разработки для создания приложений, ориентированных на взаимодействие с базами данных.
5. Тестопригодность – средняя.

Недостатки:

1. Невозможно задействовать многие из техник объектного подхода (типовых шаблонов проектирования), таких как наследование, стратегия, состояние, прокси и другие при разработке
2. Сильная привязка к модели хранения данных
3. Оперирование наборами данных не позволяет проводить объектную декомпозицию до уровня отдельных сущностей, ограничиваясь назначением обязанностей на уровне таблицы в целом.

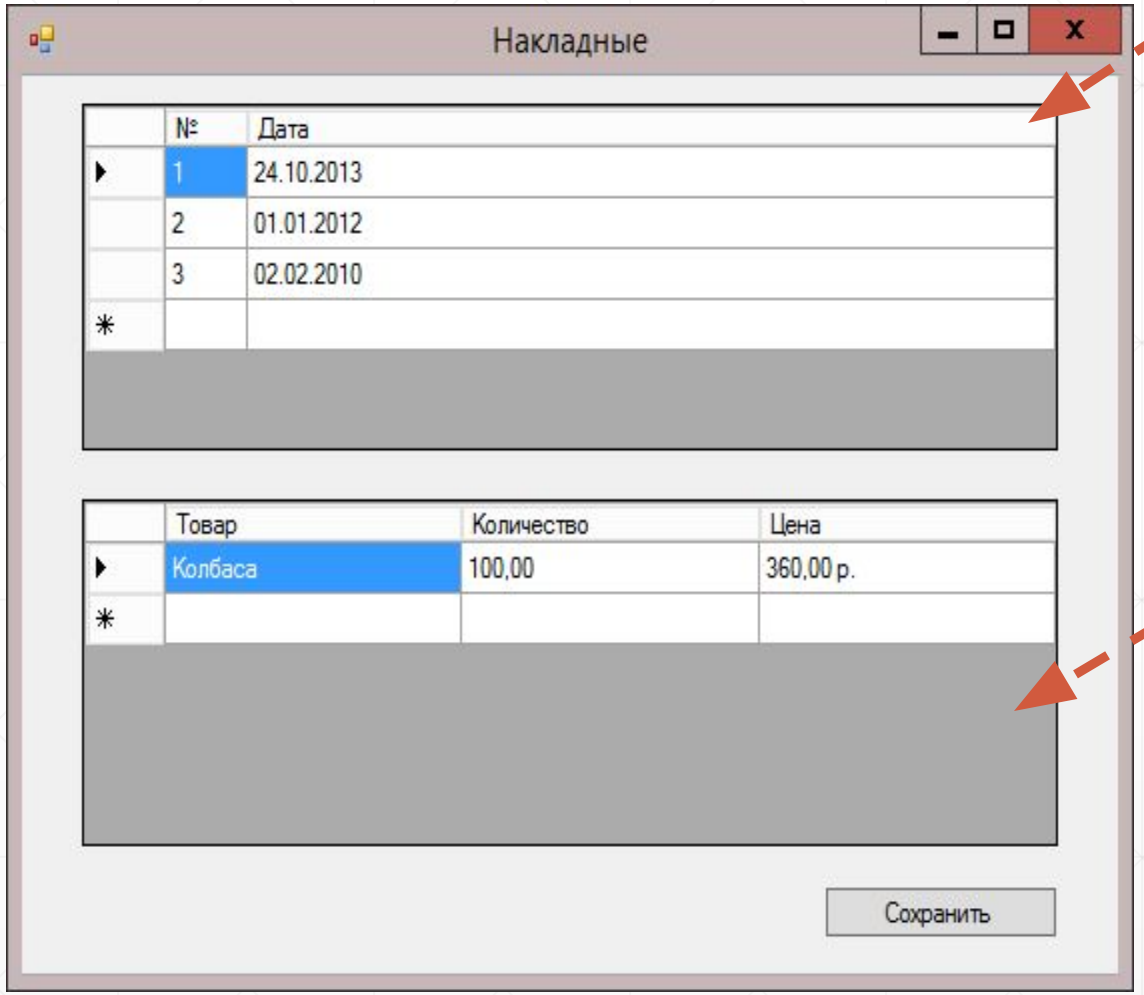


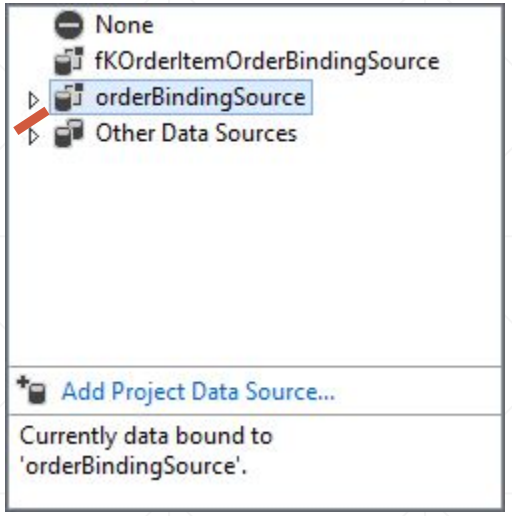
DataSet (.NET Framework)

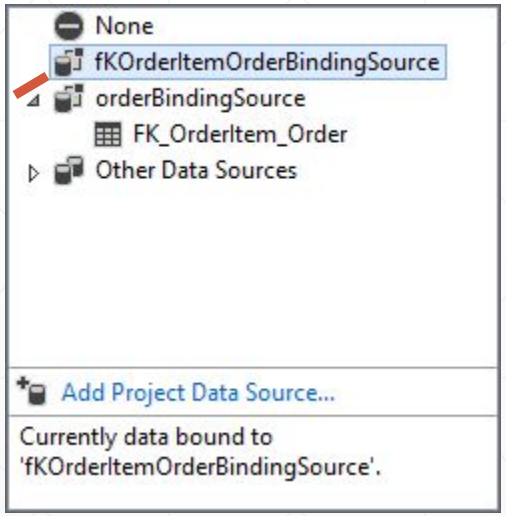
1) Представляет т.н. отсоединённый набор данных в табличном виде
2) Не зависит от используемой СУБД.
3) Для управления данными (выборка, обновление, вставка, удаление) в наборе используются адаптеры таблиц

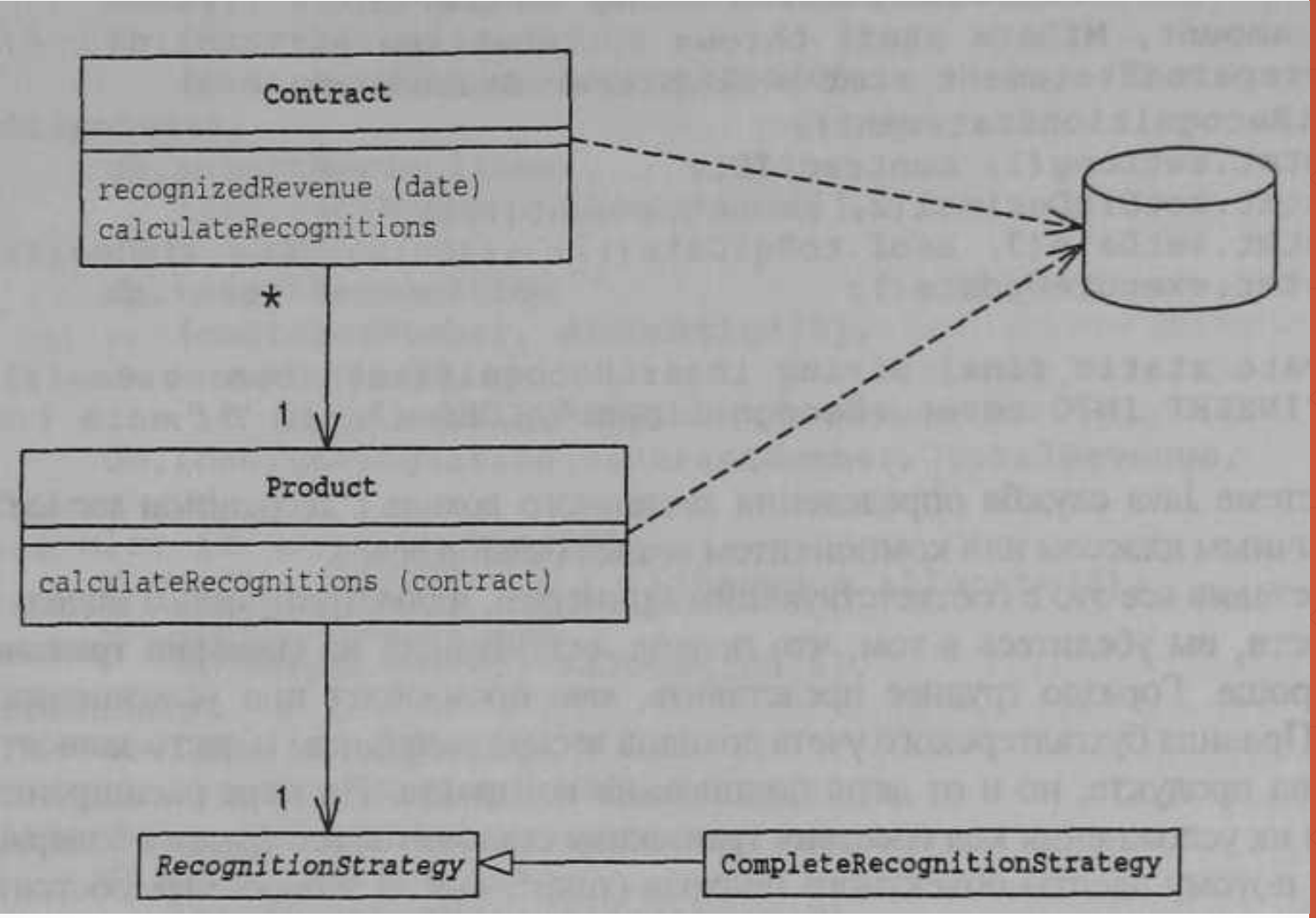
```
public partial class OrderWindow : Form
    {
        public OrderWindow()
        {
            InitializeComponent();
        }

        private void OrderWindow_Load(object sender, EventArgs e)
        {
            orderItemTableAdapter.Fill(sampleData.OrderItem);
            orderTableAdapter.Fill(sampleData.Order);

        }

        private void SaveBtn_Click(object sender, EventArgs e)
        {
            orderTableAdapter.Update(sampleData.Order);
            orderItemTableAdapter.Update(sampleData.OrderItem);
        }
    }
```

Данные привязки



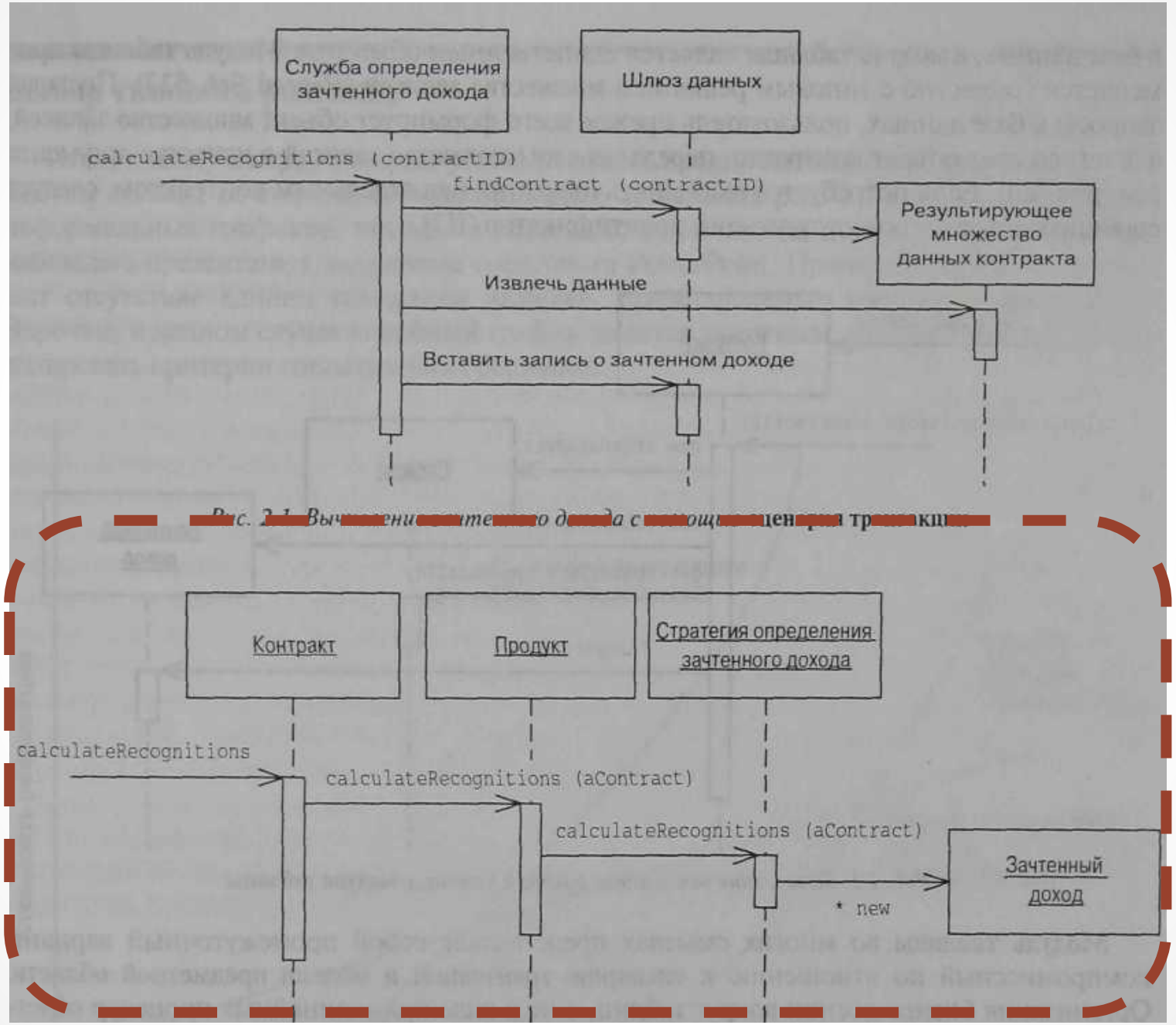






Модель предметной области

*Объектная модель домена, охватывающая **поведение (функции)** и свойства (данные)*



Пример модели предметной области

Вычисление зачётного дохода

Пример

```
class Contract : Entity
{
    Product _product;
    public Recognition CalculateRecognitions()
    {
        return _product.CalculateRecognitions( this );
    }
}
```

Пример

```
class Product : Entity
{
    public Product(IRecognitionStrategy recStrategy)
    { _recStrategy = recStrategy; }
    private IRecognitionStrategy _recStrategy;
    public Recognition CalculateRecognitions(Contract contract)
    {
        return _recStrategy.CalculateRecognition(contract, this);
    }
}
```

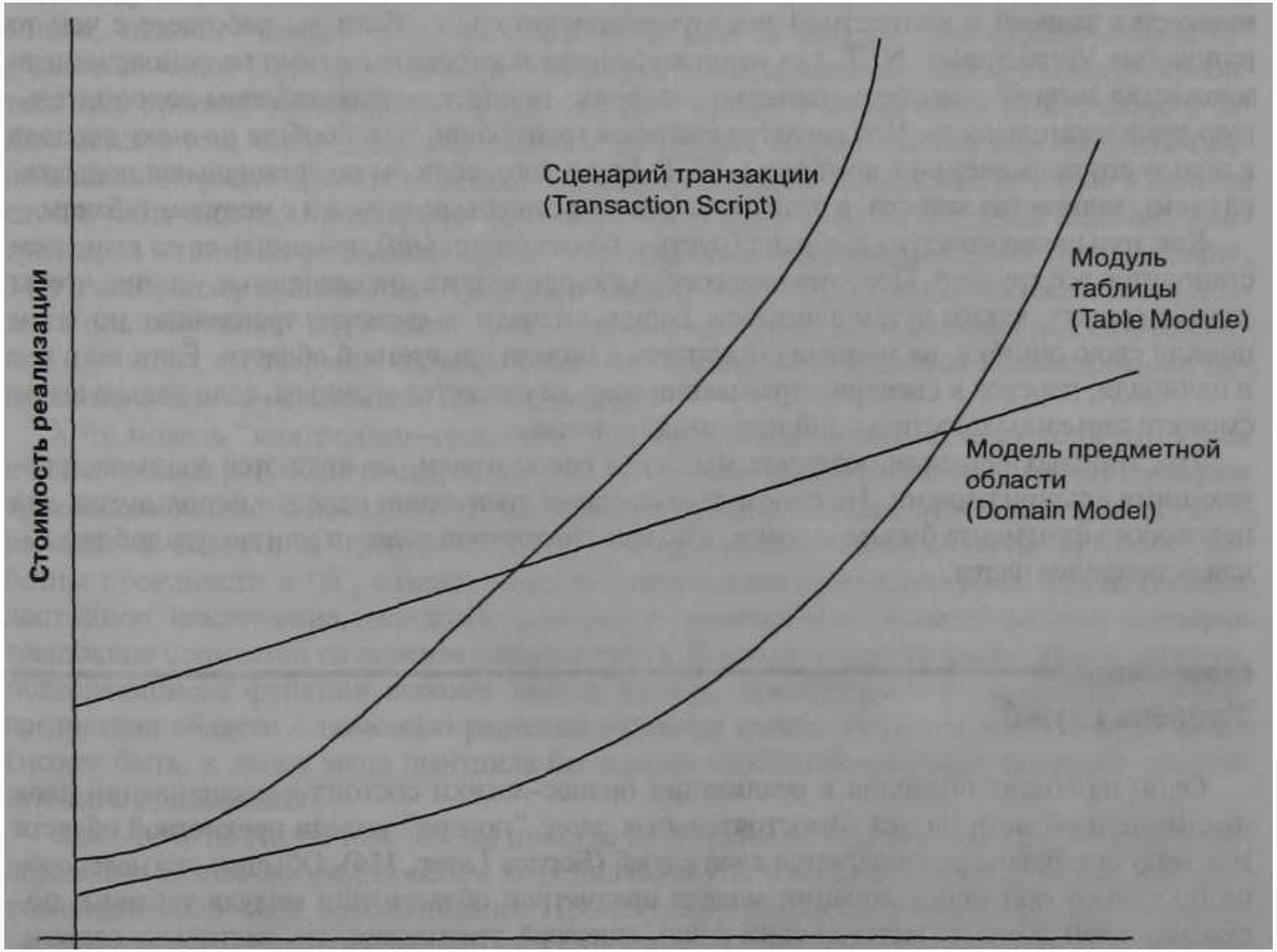
Модель предметной области. «За и против»

Преимущества:

1. Высокая эффективность борьбы со сложностью предметной области за счёт возможности использования всего арсенала средств моделирования (ОО, DSL,…)
2. Высокое приближение модели к моделируемой предметной области.
3. Поддержка со стороны современных средств разработки.
4. Тестопригодность – высокая.

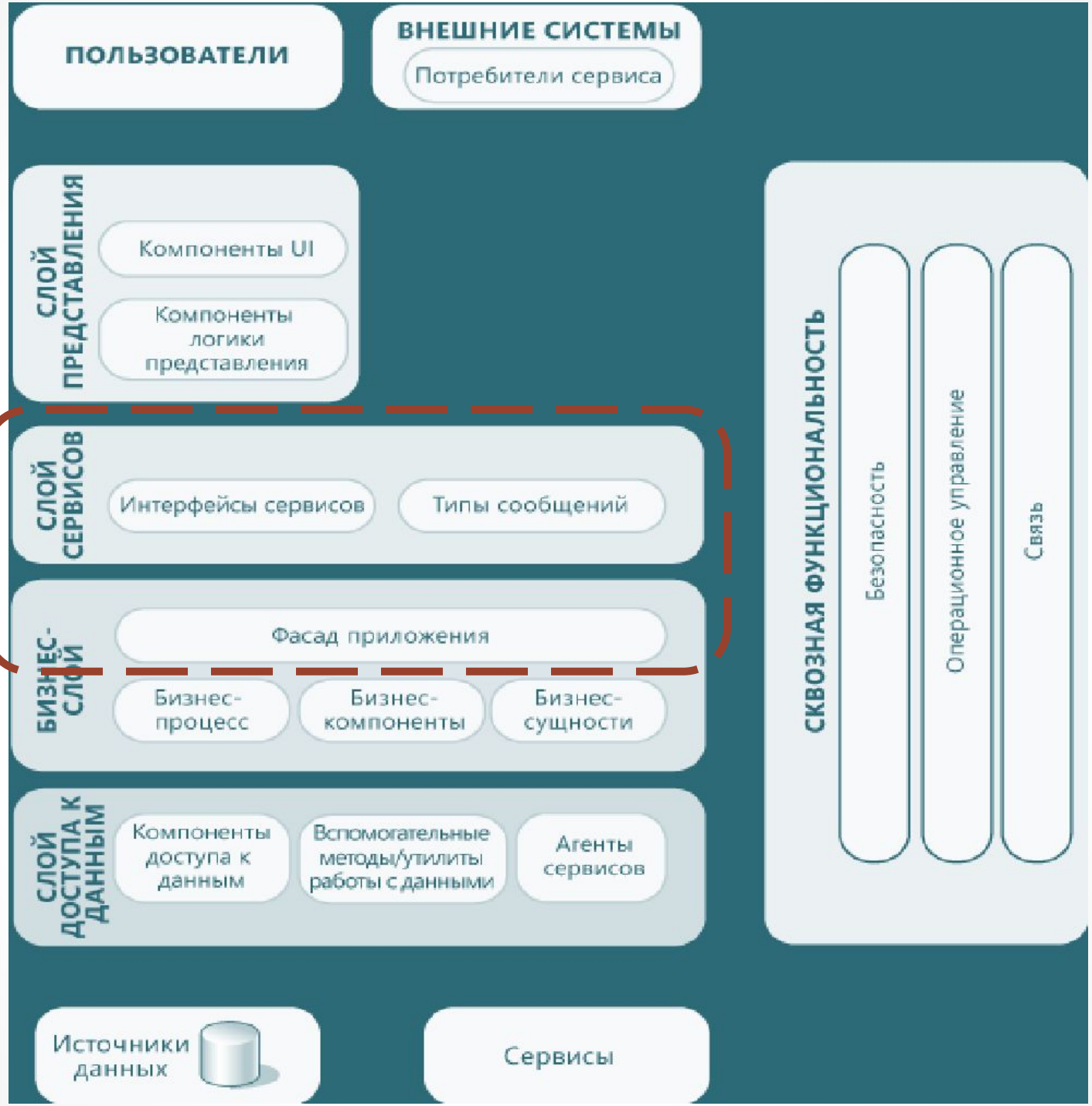
Недостатки:

1. Более высокие требования к квалификации разработчиков
2. При отсутствии поддержки со стороны средств разработки требуются значительные ресурсы на выстраивание архитектуры решения
3. Усложнённая схема отображения между объектной моделью и реляционной.



Зависимость стоимости реализации различных схем организации БЛ от её сложности

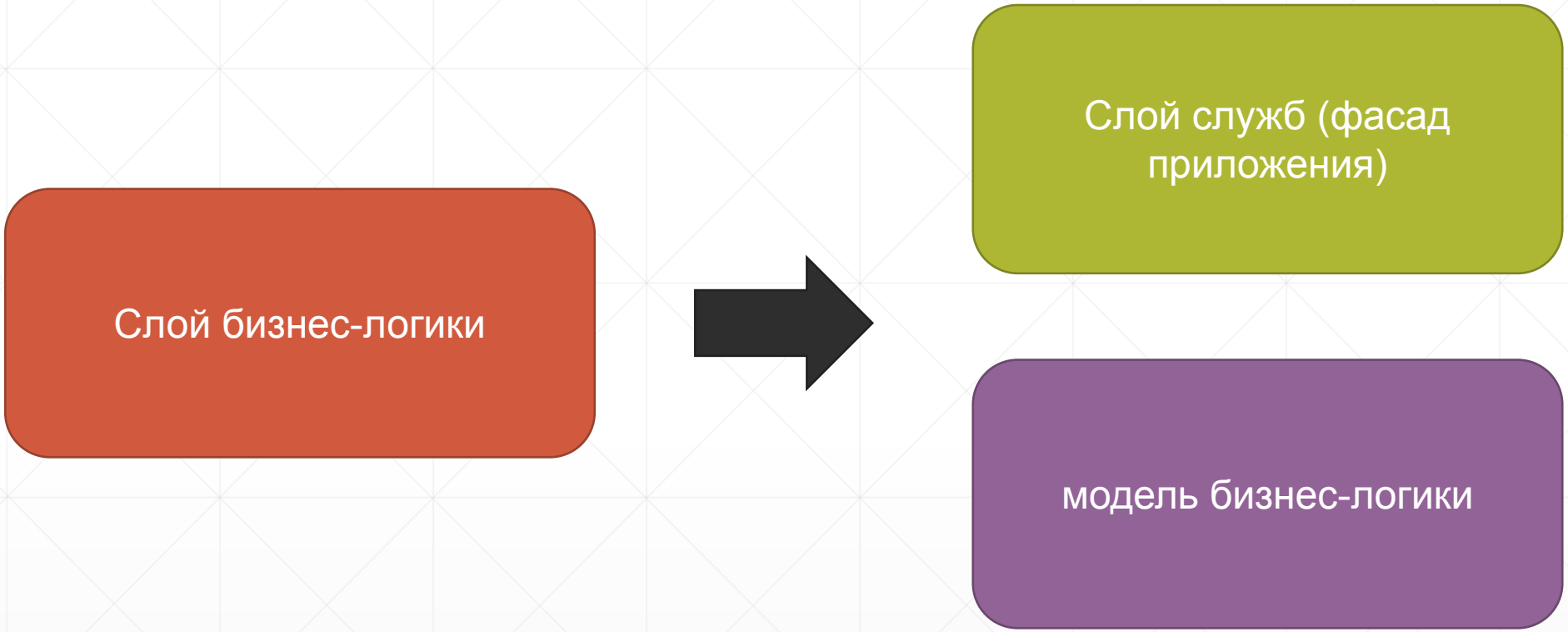
Источник картинки здесь



Уровень служб (сервисов)

Источник картинки

Расщепление слоя БЛ



Назначение слоя служб

- Определяет границы приложения и множество операций, предоставляемых им для интерфейсных клиентских слоев кода.
- Инкапсулирует бизнес-логику приложения, управляет транзакциями и координирует реакции на действия.
- Может служить для реализации *аспектов (безопасность, кэширование, логирование, и т.д.)*

Варианты реализации слоя служб

- Интерфейс доступа к домену (domain facade)
- Сценарий операции (operation script).

Интерфейс доступа к домену

- Реализуется как набор "тонких" интерфейсов, размещенных "поверх" **модели предметной области.**
- В классах, реализующих интерфейсы, никакая бизнес-логика отражения не находит — она сосредоточена исключительно в контексте **модели предметной** области.
- Тонкие интерфейсы устанавливают границы и определяют множество операций, посредством которых клиентские слои взаимодействуют с приложением.

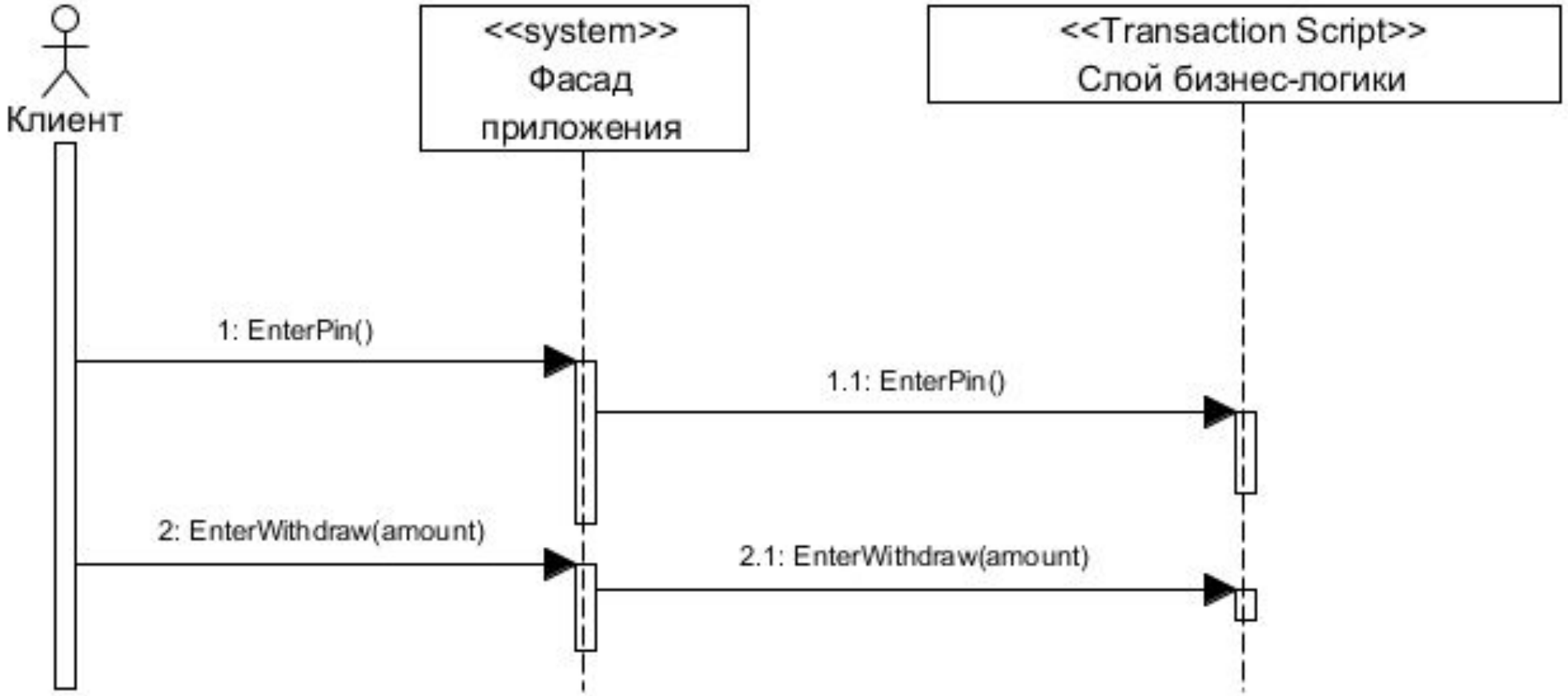
Сценарий операции

- Реализуется слой служб как множество более "толстых" классов, которые непосредственно воплощают в себе логику приложения, но за бизнес-логикой обращаются к классам домена.
- Операции, предоставляемые клиентам слоя служб, реализуются в виде сценариев, создаваемых группами в контексте классов, каждый из которых определяет некоторый фрагмент соответствующей логики.

Варианты взаимодействия слоя служб и слоя БЛ

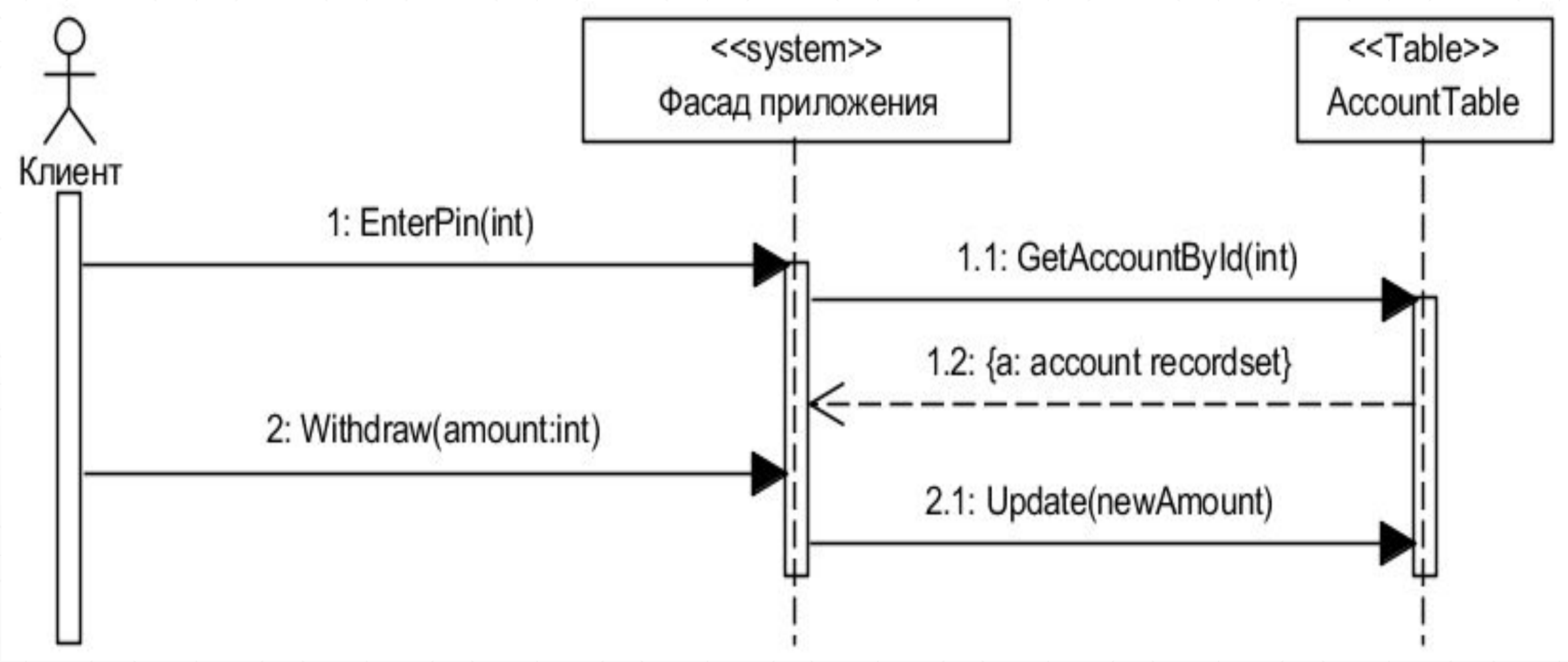
	1. Сценарий транзакции	2. Модуль таблицы	3. Модель предметной области
1. Сценарий операции	Не имеет смысла	Применимо	В целом применимо
2. Интерфейс доступа к домену	В целом применимо	В целом применимо	Применимо

1-1 (Сценарий операции – сценарий транзакции)

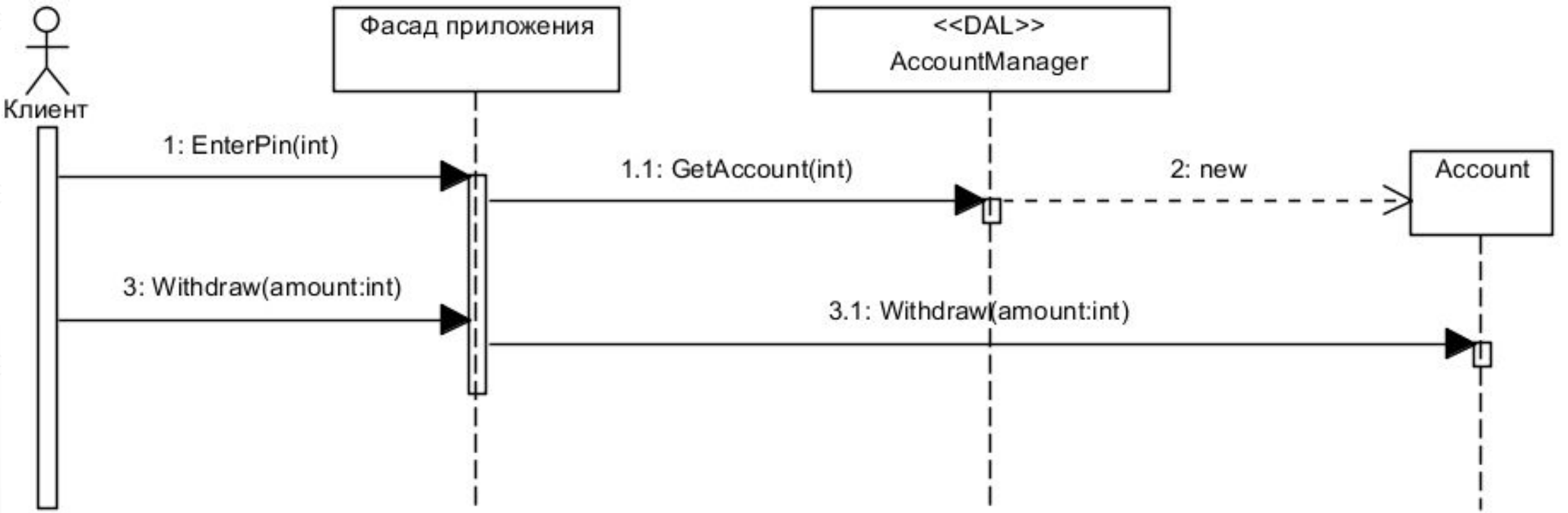


Фасад приложения занимается тем, что делегирует вызовы методов слою бизнес-логики. При этом методы слоя БЛ и фасада будут в большей части совпадать и по сигнатурам и по названию. Налицо увеличение сложности решения без получения дополнительных преимуществ

1-1 (Модуль таблицы – сценарий операции)



3-1 (Модель предметной области– сценарий операции)

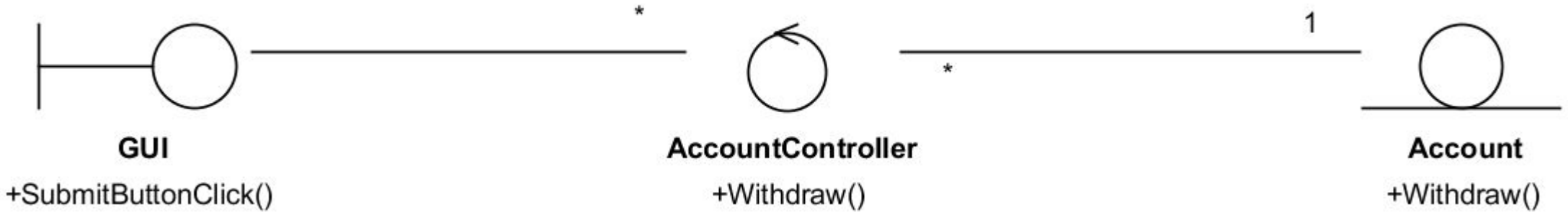


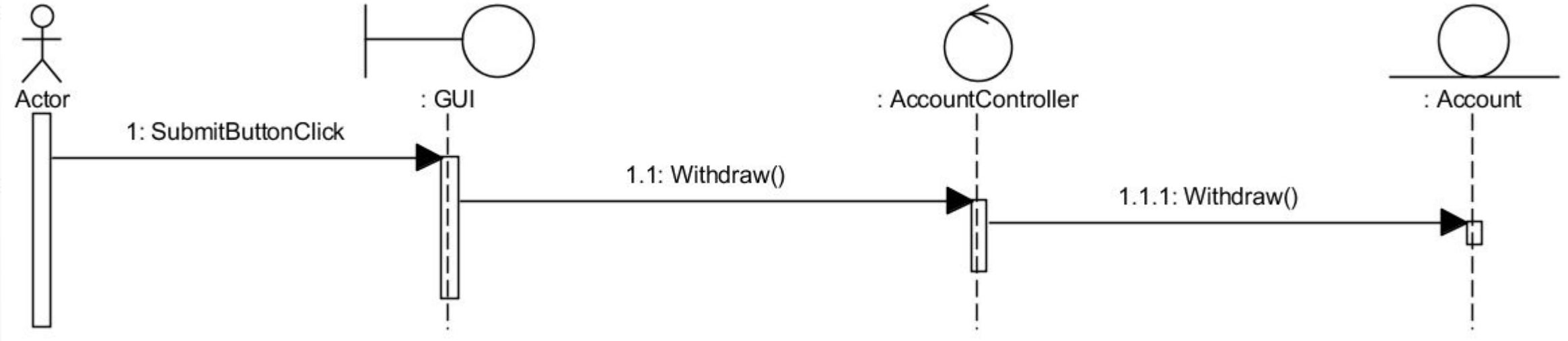
Типовые конфигурации слоя служб и слоя БЛ

(На базе интерфейса доступа к домену)

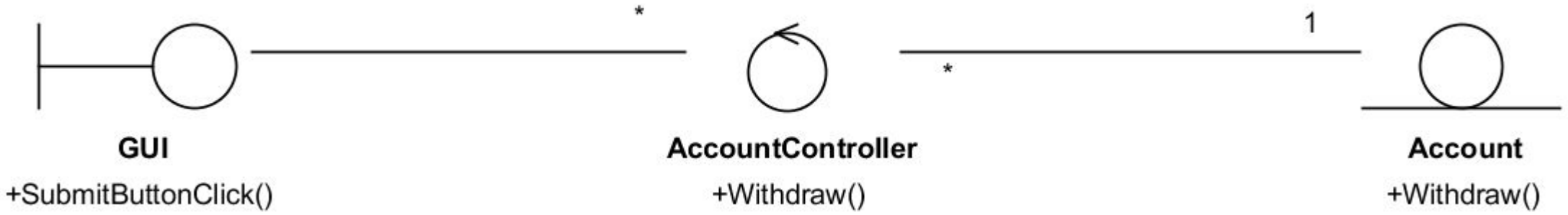
БЛ СС	1. Сценарий транзакции	2. Модуль таблицы	3. Модель предметной области
1. Граница-Контроллер – Сущность (Boundary-Controller-Entity)	Не имеет смысла	В целом применимо	В целом применимо
2. Контроллер приложения (Application Controller)	Применимо	Применимо	Применимо
3. Модель-Представление-Контроллер (Model-View-Controller, MVC)	В целом применим	Применимо	Применимо
4. Модель-представление-модель представления (Model-View-ViewModel, MVVM)	Не имеет смысла	Применимо	Применимо

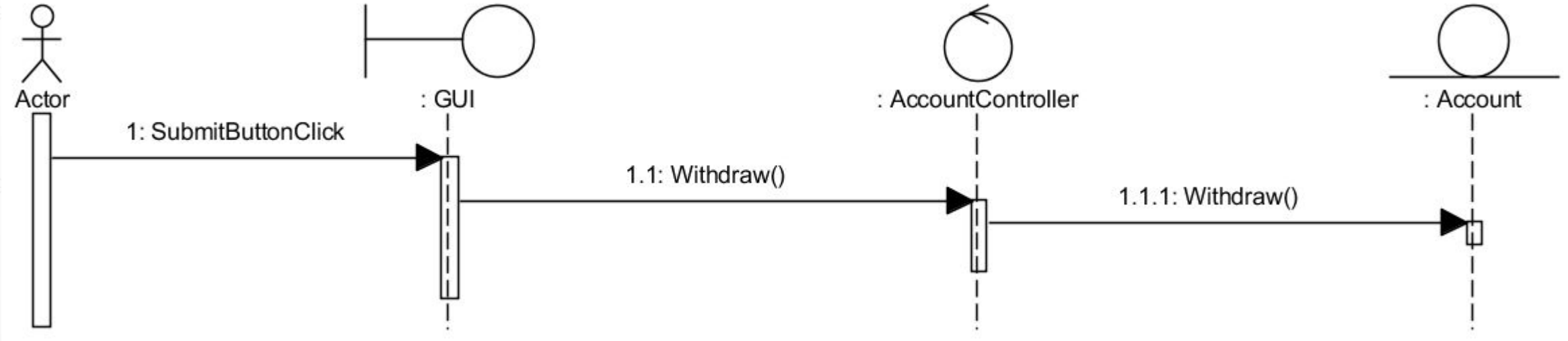
Граница-Контроллер – Сущность





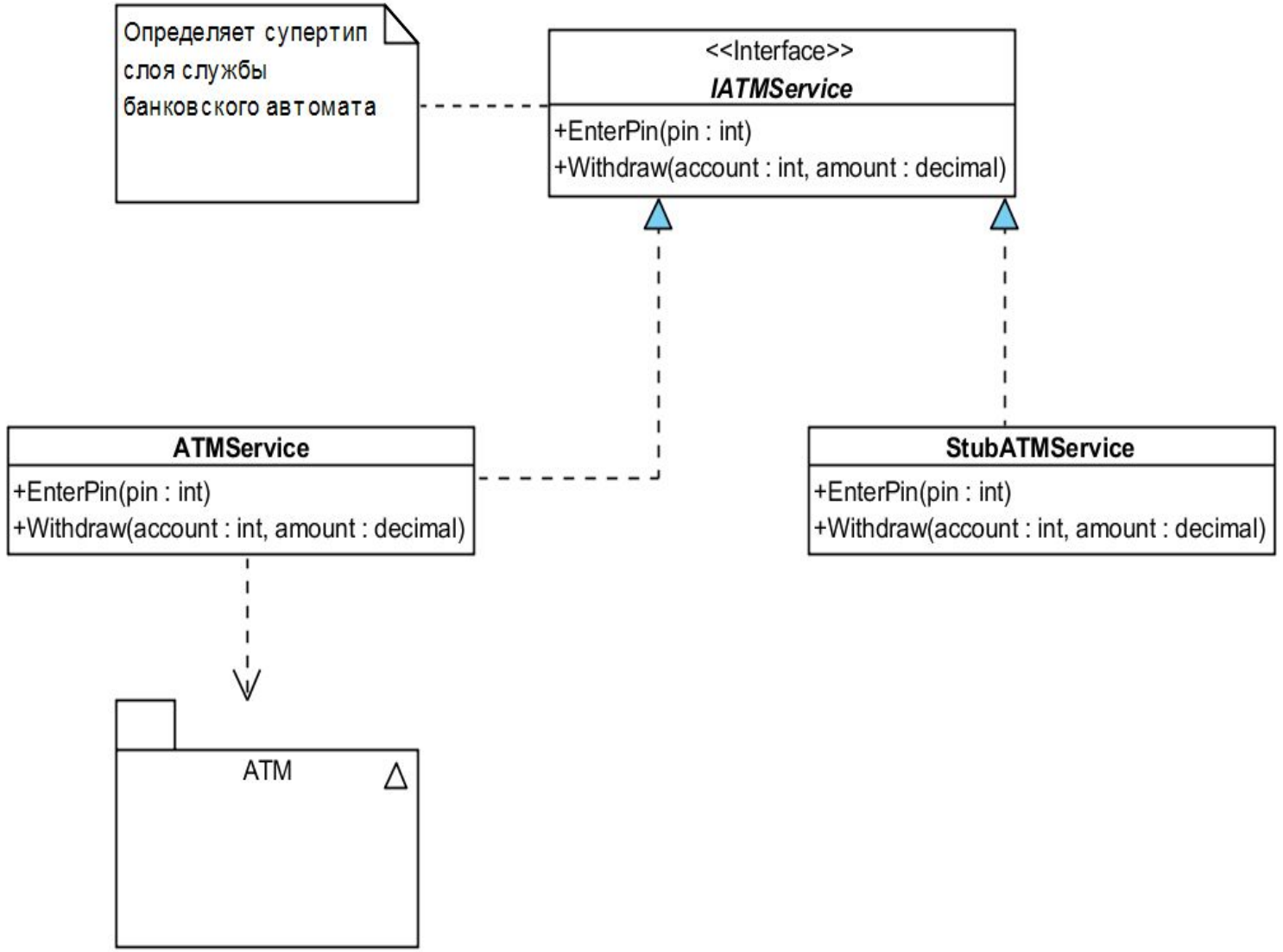
Граница-Контроллер – Сущность





Супертип слоя

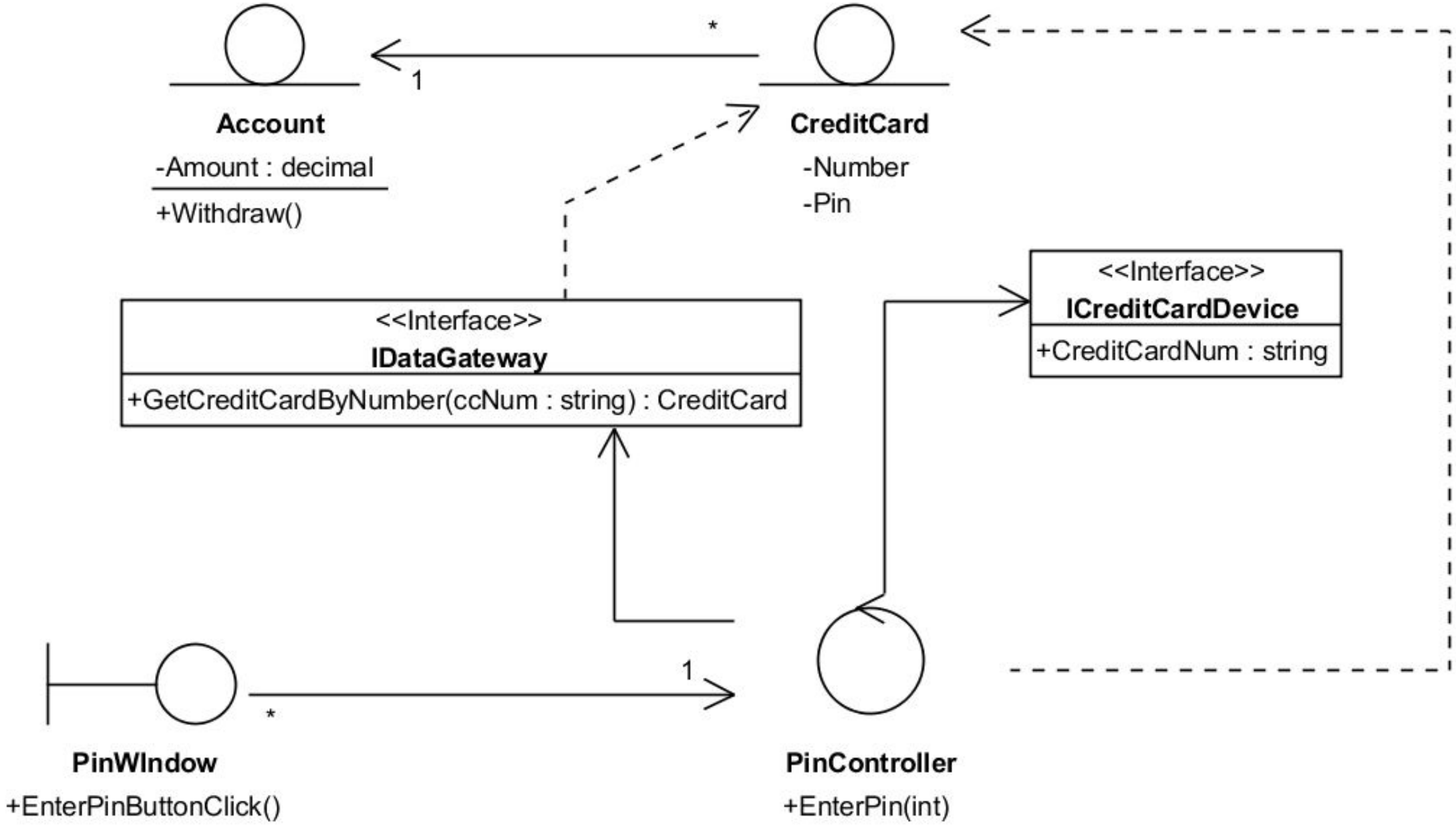
- Тип, выполняющий роль суперкласса для всех классов своего слоя
- Может быть определён для любого слоя
- Упрощает задачи интеграции компонентов системы, тестируемость, поощряет изменчивость.
- Используется для задания (спецификации) общего поведения типов некоторого слоя.
  - Например, можно определить для слоя БЛ тип IDomainObject, задающий спецификацию всех типов МПО.



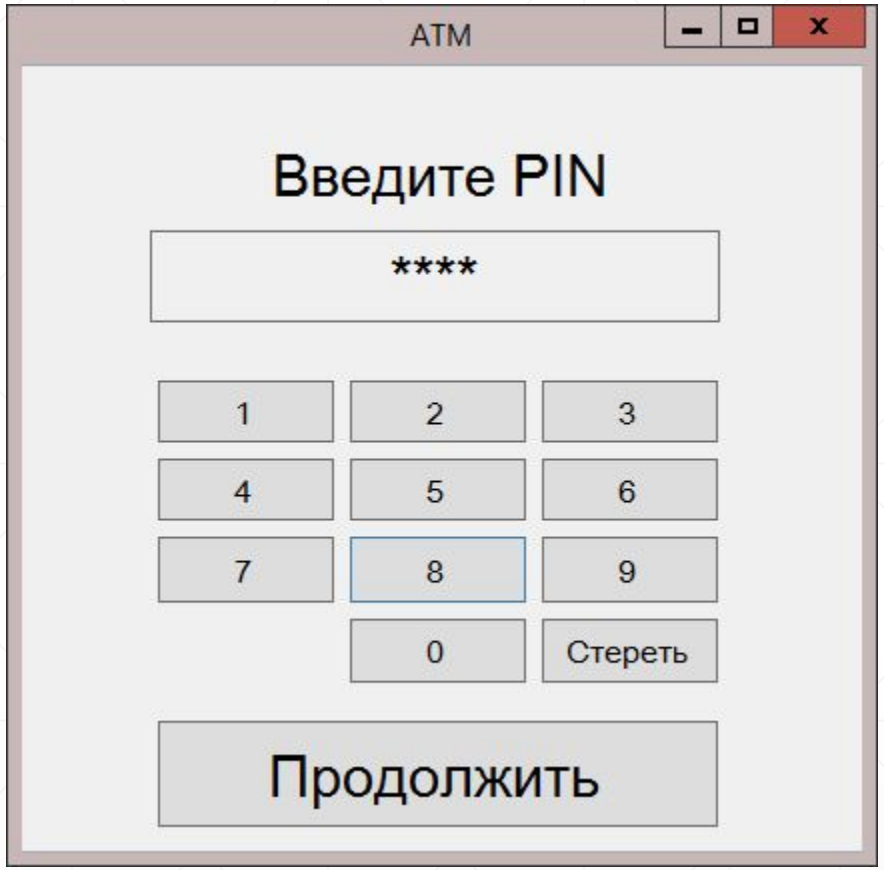
Пример супертипа слоя служб

1. Интерфейс IATMService задаёт спецификацию операция, выполняемых системой (банкоматом)
2. Позволяет развивать систему независимо от наличия всех аппаратных компонентов
3. Обеспечивает «безболезненную» интеграцию системы в рабочую среду.

Пример. ATM


*
1
Account
-Amount : decimal
+Withdraw()
CreditCard
-Number
-Pin
<<Interface>>
IDataGateway
+GetCreditCardByNumber(ccNum : string) : CreditCard
<<Interface>>
ICreditCardDevice
+CreditCardNum : string
1
*
PinWIndow
+EnterPinButtonClick()
PinController
+EnterPin(int)

```
static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            const int PIN = 1234;
            const string CardNum = "6578-3456-2345-6678";
            const int DefMaxPinLen = 4;

            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new PINWIndow(
                new PinController(DefMaxPinLen,
                    new StubCreditCardDevice(CardNum),
                    new StubDataGateway(CardNum, PIN))));
        }
    }
```




```
public partial class PINWIndow : Form
   {

        private readonly PinController _pinController;

        public PINWIndow(PinController pinController)
        {
            _pinController = pinController;
            InitializeComponent();
        }

        private void SubmitButtonClick(object sender, EventArgs e)
        {
            var pin = Convert.ToInt32(pinTextBox.Text);
            var result = _pinController.EnterPin(pin);
            MessageBox.Show("Enter PIN result is:" + result);

        }

        private void EnterDigitClick(object sender, EventArgs e)
        {
            var maxPinLen = _pinController.MaxPinLength;
            if (pinTextBox.Text.Length >= maxPinLen)
                return;
            pinTextBox.Text += (sender as Button).Text;
        }

        private void ClearPressed(object sender, EventArgs e)
        {
            pinTextBox.Clear();
        }
    }
```

```
public class PinController
    {
        private readonly ICreditCardDevice _creditCardDevice;
        private readonly IDataGateway _dataGateway;

        public PinController(int maxPinLen, ICreditCardDevice creditCardDevice, IDataGateway dataGateway)
        {
            _creditCardDevice = creditCardDevice;
            _dataGateway = dataGateway;
            MaxPinLength = maxPinLen;
        }

        public int MaxPinLength { get; private set; }

        public bool EnterPin(int pin)
        {
            var creditCard = _dataGateway.GetCreditCardByNumber(_creditCardDevice.CreditCardNum);
            return (creditCard != null) && (creditCard.Pin == pin);
        }
    }
```

```
public interface ICreditCardDevice
    {
        string CreditCardNum { get; }
    }
```

```
class StubCreditCardDevice : ICreditCardDevice
    {
        public StubCreditCardDevice(string ccNum)
        {
            CreditCardNum = ccNum;
        }

        public string CreditCardNum { get; private set; }
    }
```

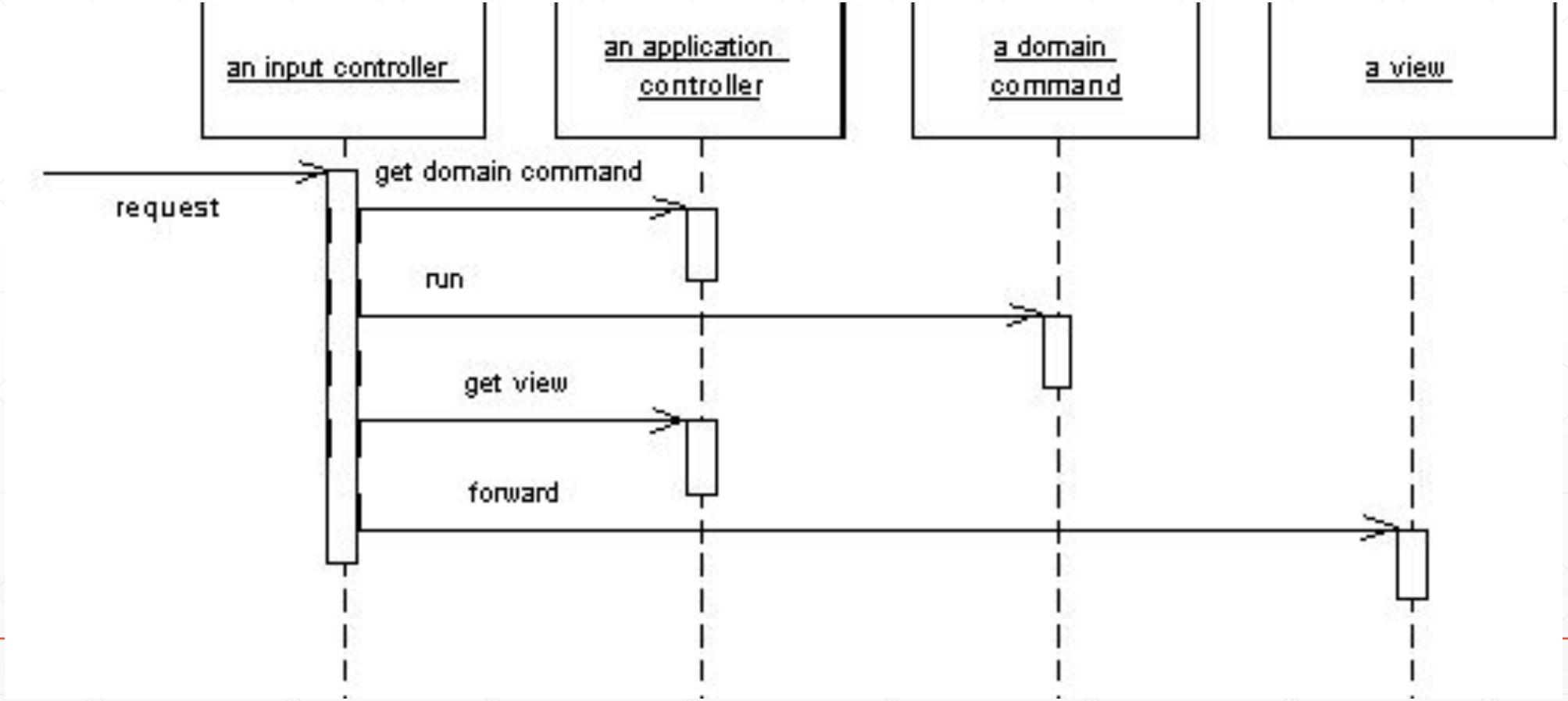
```
public interface IDataGateway
    {
        CreditCard GetCreditCardByNumber(string ccNum);
    }

class StubDataGateway : IDataGateway
    {
        private readonly string _ccNum;
        private readonly int _pin;


        public StubDataGateway(string ccNum, int pin)
        {
            _ccNum = ccNum;
            _pin = pin;
        }

        public CreditCard GetCreditCardByNumber(string ccNum)
        {
            if (ccNum==_ccNum)
                return new CreditCard(){Number = _ccNum, Account = new Account(){Amount = 1000}, Pin = _pin};
            return null;
        }
    }
```

```
public class CreditCard
   {
       public string Number { get; set; }
       public Account Account { get; set; }
       public int Pin { get; set; }
   }
```

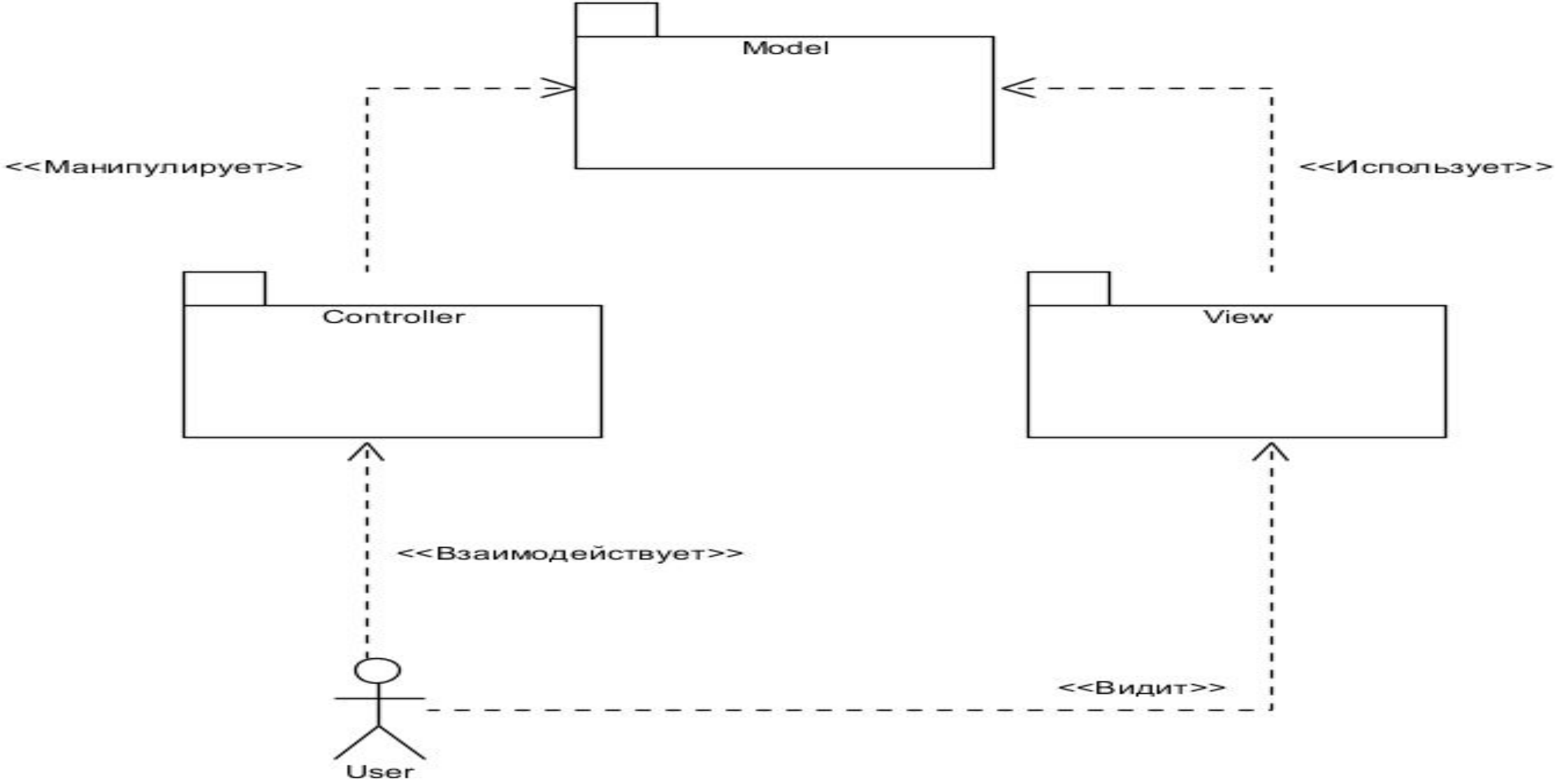
```
public class Account
   {
       public decimal Amount { get; set; }
   }
```

Application Controller

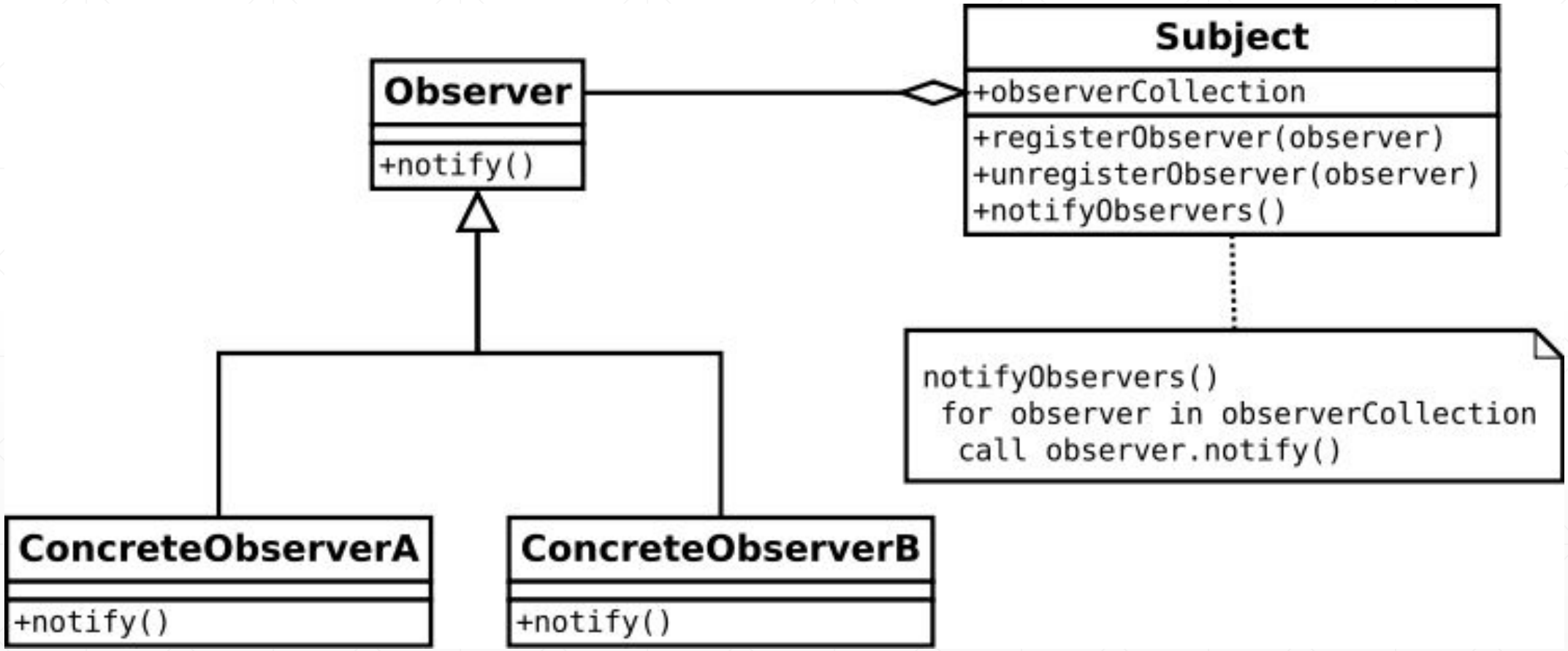
Позволяет инкапсулировать логику навигации по приложению



Model-View-Controller



Observer (наблюдатель)



Model-View-View Model (MVVM)