Проектирование архитектур программного обеспечения

лекция 2

Зозуля А.В.

КСПТ

2016 г.

Ранее..

- Введение
- Информационные системы
- Архитектура ПО
- Архитектурные слои
- Моделирование

Содержание

- Элементы модели: объекты, отношения, службы, модули
- Сервисный слой
- Типовые решения организации бизнес-логики

Моделирование - итеративный процесс

- Уточнение словаря языка модели
- Моделирование «вслух»
- Не модель для диаграмм, а диаграммы для модели
- Эффект «черновика»
- Исходный код
- Комментарии к исходному коду
- Дополняющая код документация
- Перекрестные ссылки между артефактами
- Поддержка артефактов в актуальном состоянии

Элементы модели

- Объекты-сущности (Entity Objects)
- Объекты-значения (Value Objects)
- Отношения (Ассоциации)
- Службы (Services)
- Модули (Modules)

Объект-сущность

- Не только набор атрибутов
- Непрерывность существования
- Имеет выраженный жизненный цикл
- Несколько представлений
- Индивидуальная логическая единица
- Возможность однозначно идентифицировать

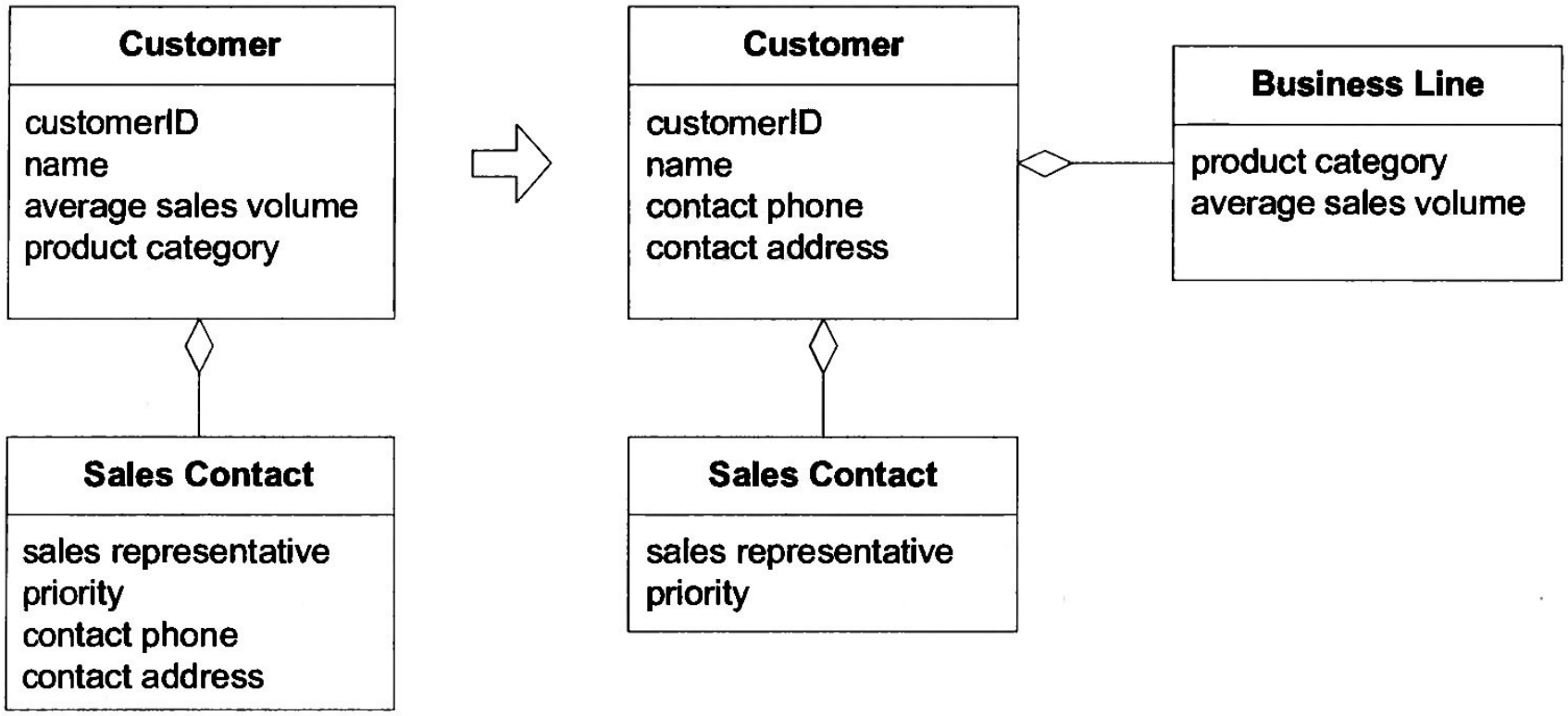
Примеры:

- Что является Объектом-сущностью: «Чек», «Сумма»?
- Чем идентифицировать сущность «Человек»?

Жизненный цикл объекта

Пример моделирования сущностей

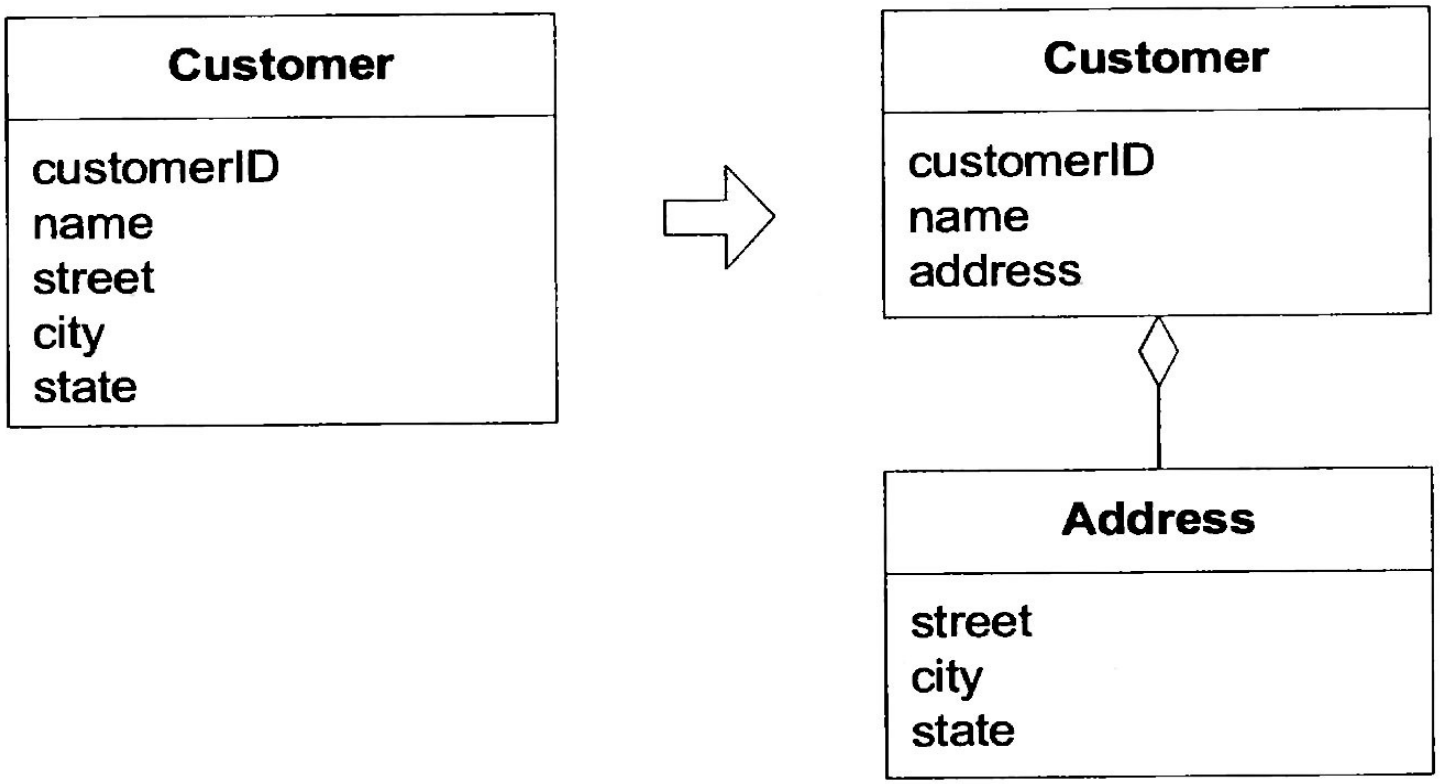
Иллюстрации: Предметно-ориентирование проектирование. Эрик Эдванс



Объект-значение

- Нет индивидуального существования
- Описательный аспект предметной области
- Характеризует один или несколько объектов-сущностей (атрибут)
- Может быть совокупностью других объектов-значений
- Может быть параметром сообщений между сущностями
- Чаще не изменяемый, а заменяемый (read-only, copy-on-paste)

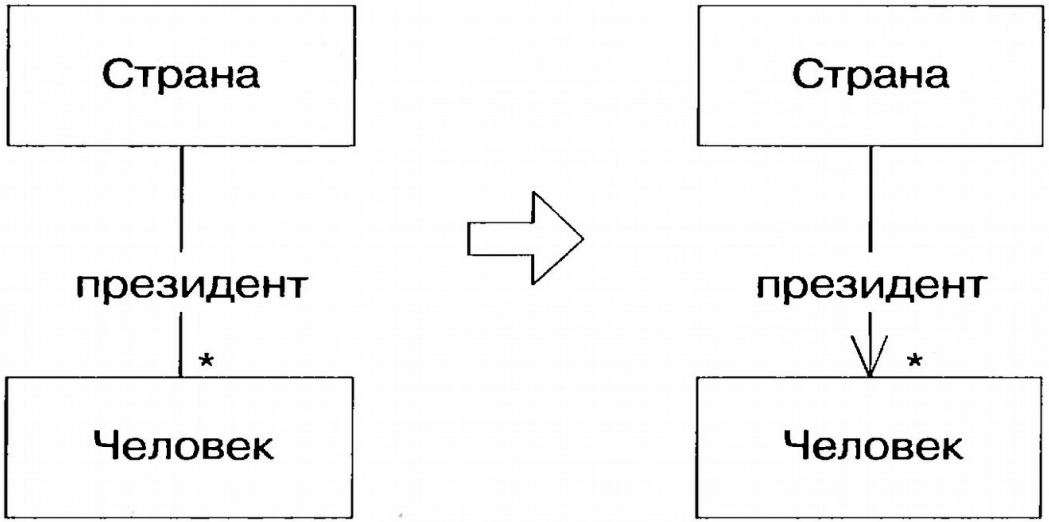
Пример проектирования объектов-значений

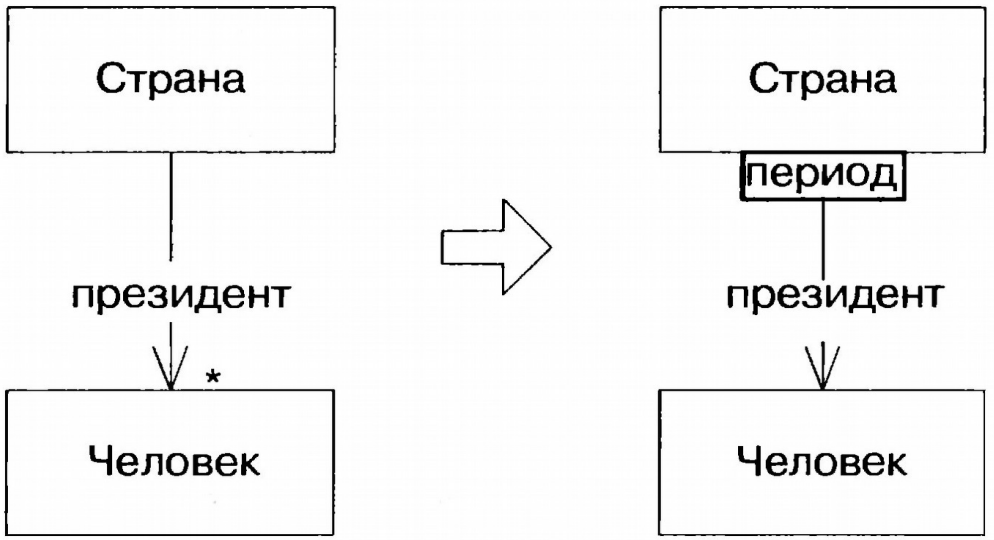


Отношения (ассоциации)

- Каждой ассоциации модели должна соответствовать ассоциация в программе
- В реальном мире чаще ассоциации двунаправленные и многим-ко-многим
- При моделировании следует минимизировать число связей:
    - Вводить направление
    - Снижать кратность
    - Упразднять лишние

Пример упрощения ассоциаций





Службы

- Операции, которые нельзя отнести к объектам
- Являются видами деятельности
- Сложные и сильносвязанные методы у объекта — признак необходимости выделения Службы
- Очищают объекты
- Не имеют состояний
- Интерфейс (параметры и результат) — объекты модели
- Могут встречаться на всех уровнях:
    - Операционном / Служб
    - Предметной области / Домен
    - Инфраструктурном / Хранения

Пример распределение служб по уровням

Операционный — *Операционная служба перевода средств:*

- принимает входные данные (например, XML-запрос);
- посылает сообщение в службу модели для выполнения;
- ожидает подтверждения;
- принимает решение об отправке извещения через службу инфраструктурного уровня

Предметной области — *Служба перевода средств в предметной модели:*

- взаимодействует с нужными объектами **`Счет (Account)`** и **`Книга (Ledger)`**, выполняя операции дебита и кредита;
- посылает подтверждение результата (разрешен перевод или нет и т.п.)

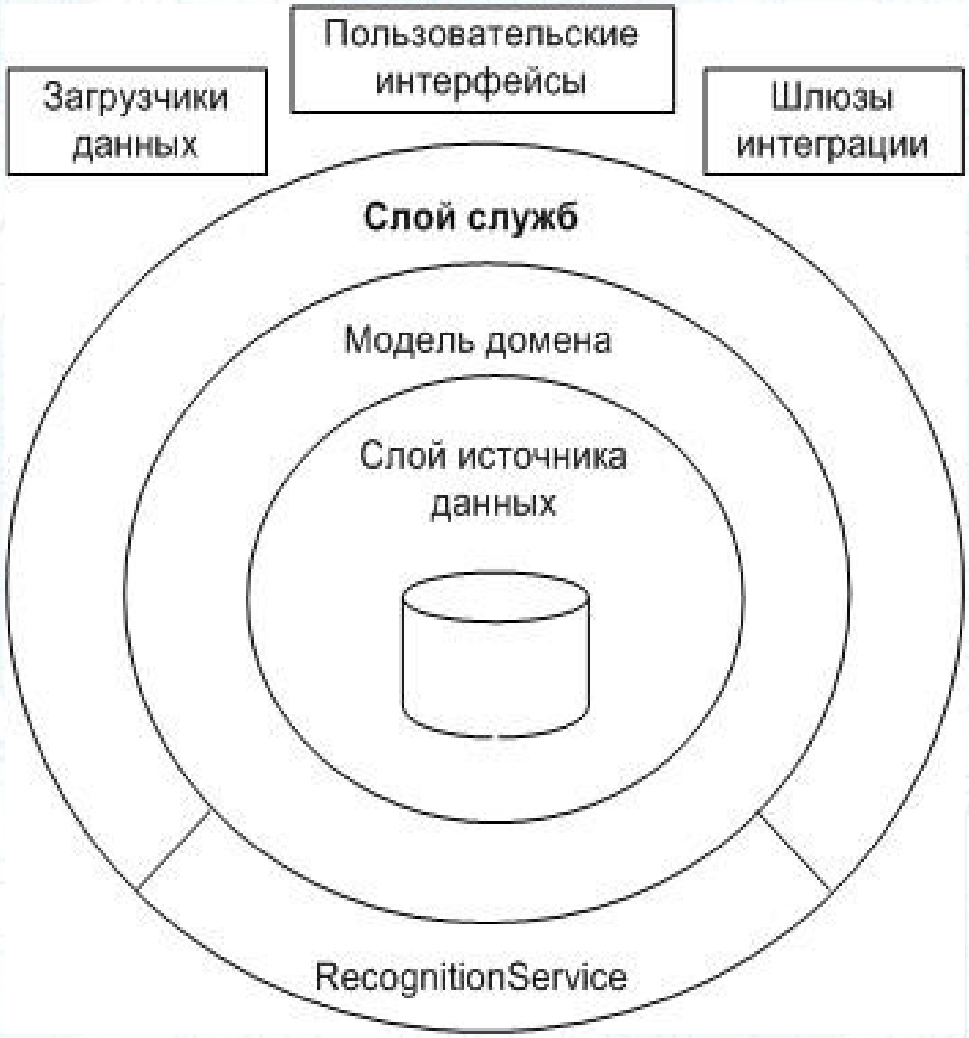
Инфраструктурный — *Служба рассылки извещений:*

- рассылает бумажные и электронные письма, осуществляет связь другого рода по указаниям операционного уровня.

Модули

- Устоявшиеся элементы архитектуры
- Уменьшение смысловой сложности
- Декомпозиция функциональных частей
- Сильная связность внутри и слабая снаружи
- Имена из единого языка
- Эволюционируют вместе с другими компонентами модели
- Затруднен рефакторинг
- Минимум в уровне предметной области

Сервисный слой (Служб)



Слой служб

- Устанавливает множество доступных действий
- Координирует отклик приложения на каждое действие
- Облегчает предоставление нескольких интерфейсов
- Инкапсулирует бизнес-логику, управляет транзакциями
- Варианты реализации:
    - Интерфейс доступа к домену (Domain Facade): «тонкий» интерфейсы «поверх» модели предметной области
    - Сценарий операции (Operation Script): «толстые» классы, содержащие операционную логику (не бизнес-логику)

Типовое решение

- Описание повторяющейся проблемы и обобщенного пути ее решения
- Выросли из практики
- «Полуфабрикат»
- Независимо от других типовых решений
- Может применяться совместно с другими решениями
- Формирует язык проектировщика

Типовые решения организации бизнес-логики

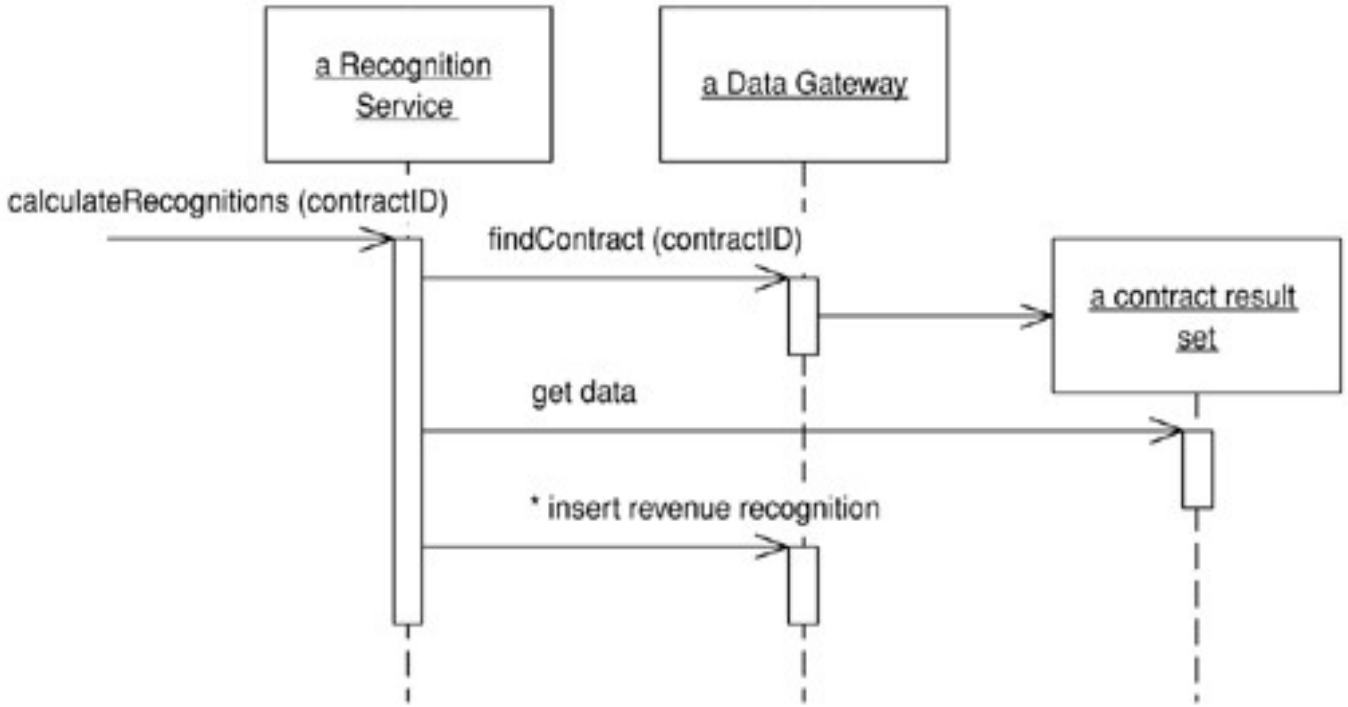
- Сценарий транзакции (Transaction Script)
- Модель предметной области (Domain Model)
- Модуль таблицы (Table Module)

Сценарий транзакции

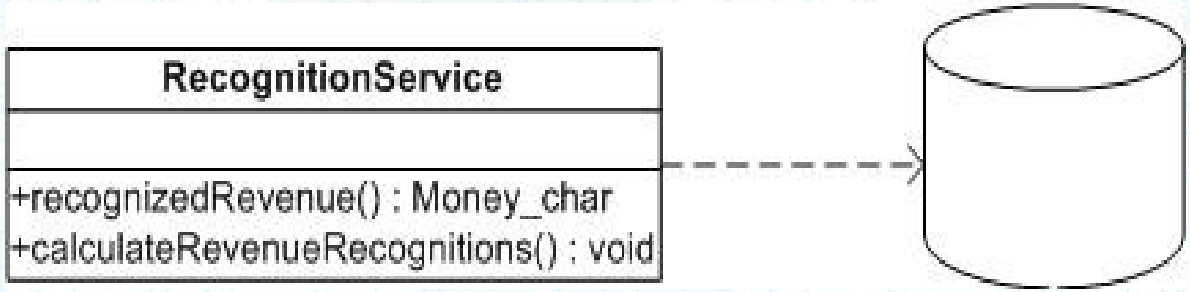
- Простейший подход к реализации бизнес-логики
- На входе данные от слоя представления
- Осуществляет выборку и необходимые расчеты
- Сохраняет в БД результаты
- Активизирует операции др. систем
- Возвращает слою представления результат
- Т.о. бизнес-логика описывается набором процедур по одной на каждую операцию

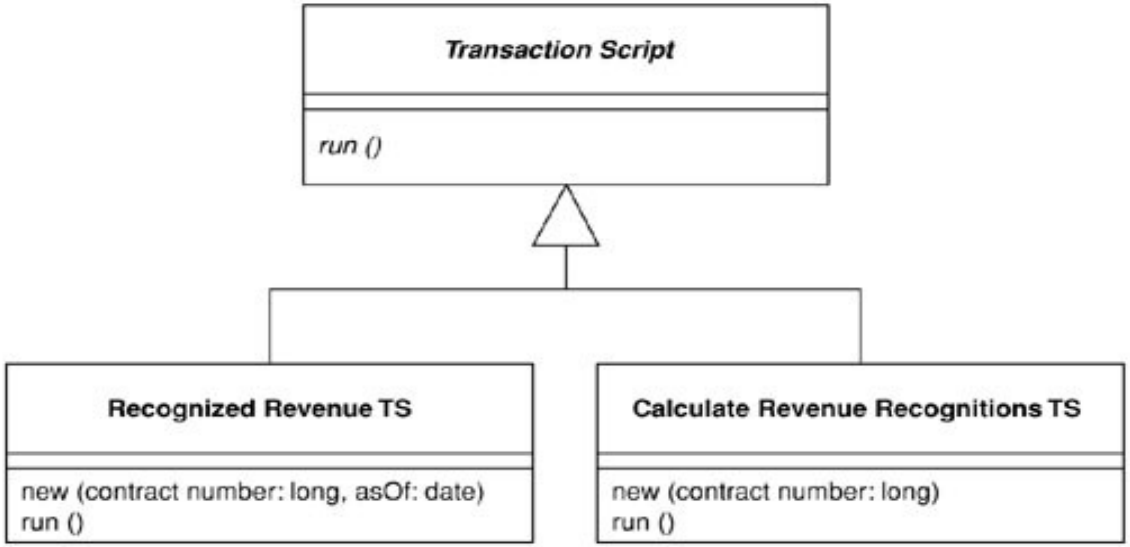
Пример сценария транзакции

Иллюстрации: Архитектура корпоративных программных приложений. М. Фаулер



Пример сценария транзакции





Пример сценария транзакции

```
class RecognitionService...
        public Money recognizedRevenue (Long contractItemNumber, MfDate  asOf) {
                Contract ci = Registry.getContractItem(contractItemNumber);
                Money result = Money.dollars(0);
                RevenueRecognition[] recs = ci.getRecognitions();
                for (int i =0; i < recs.length; i++) {
                        if (asOf.after(recs[i].getDate()) || asOf.equals(recs[i].getDate()))
                                result = result.add(recs[i].getAmount());
                }
                return result;
        }
        public void calculateRecognizedRevenue (Long contractItemNumber) {
                Contract ci = Registry.getContractItem(contractItemNumber);
                if (ci.getProduct().getRecognitionMethod() == Product.WORD_PROCESSOR) {
                        ci.addRevenueRecognition(new RevenueRecognition(ci.getRevenue(),
ci.getWhenSigned()));
                }
                else if (ci.getProduct().getRecognitionMethod() == Product.DATABASE) {
                        double percentageNow = 0.25;
                        ci.addRevenueRecognition(
new RevenueRecognition(ci.getRevenue().multiply(percentageNow), ci.getWhenSigned()));
                        ci.addRevenueRecognition(
new RevenueRecognition(ci.getRevenue().multiply(1-percentageNow),
ci.getWhenSigned().addDays(90)));
                }
                else Assert.unreachable();
        }
```

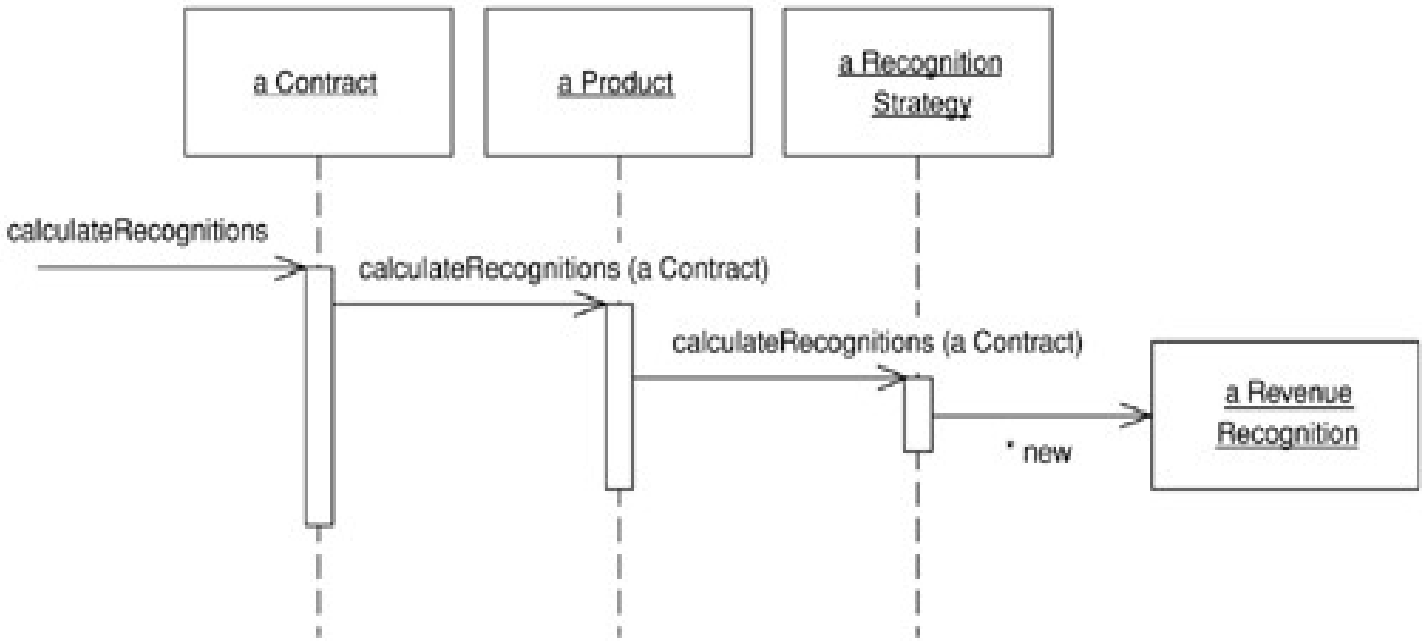
Сценарий транзакции

- Преимущества:
  - Удобная процедурная модель, легко воспринимаемая разработчиками
  - Удачно сочетается с простыми схемами организации слоя источника данных
  - Определяет четкие границы транзакций
- Недостатки:
  - Высокая сложность сценариев
  - Дублирование фрагментов кода

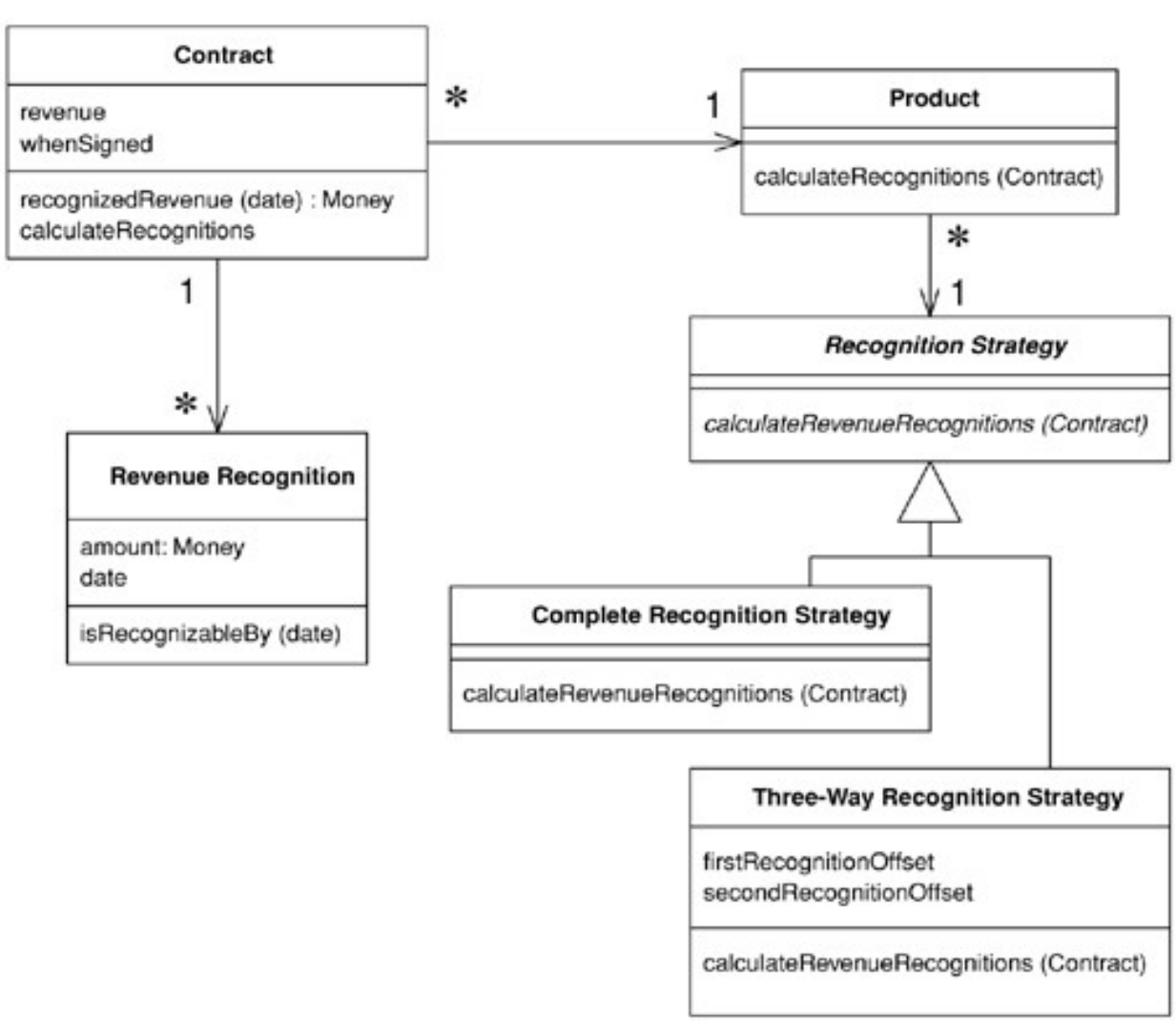
Модель предметной области

- Каждый объект наделяется функциями, соответствующими его природе
- Сеть взаимосвязанных объектов
- Функции тесно сочетаются с данными
- Для сложных часто меняющихся бизнес-правил
- Для информационной системы с «большим будущем»

Пример реализации модели предметной области



Пример реализации модели предметной области



Пример реализации модели предметной области

```
class Contract...
        private Product product; private Money revenue; private MfDate whenSigned;

        public Contract(Product product, Money revenue, MfDate whenSigned) {
                this.product = product; this.revenue = revenue; this.whenSigned = whenSigned;
        }

class Product...
        private String name;   private RecognitionStrategy recognitionStrategy;

        public Product(String name, RecognitionStrategy recognitionStrategy) {
                this.name = name;    this.recognitionStrategy = recognitionStrategy;
        }
        public static Product newWordProcessor(String name) {
                return new Product (name, new CompleteRecognitionStrategy());
        }

class RecognitionStrategy...
        abstract void calculateRevenueRecognitions(Contract ci);

class CompleteRecognitionStrategy...
        void calculateRevenueRecognitions(Contract ci) {
                ci.addRevenueRecognition(new RevenueRecognition(ci.getRevenue(),
ci.getWhenSigned()));
        }

class NinetyDayRecognitionStrategy...
```

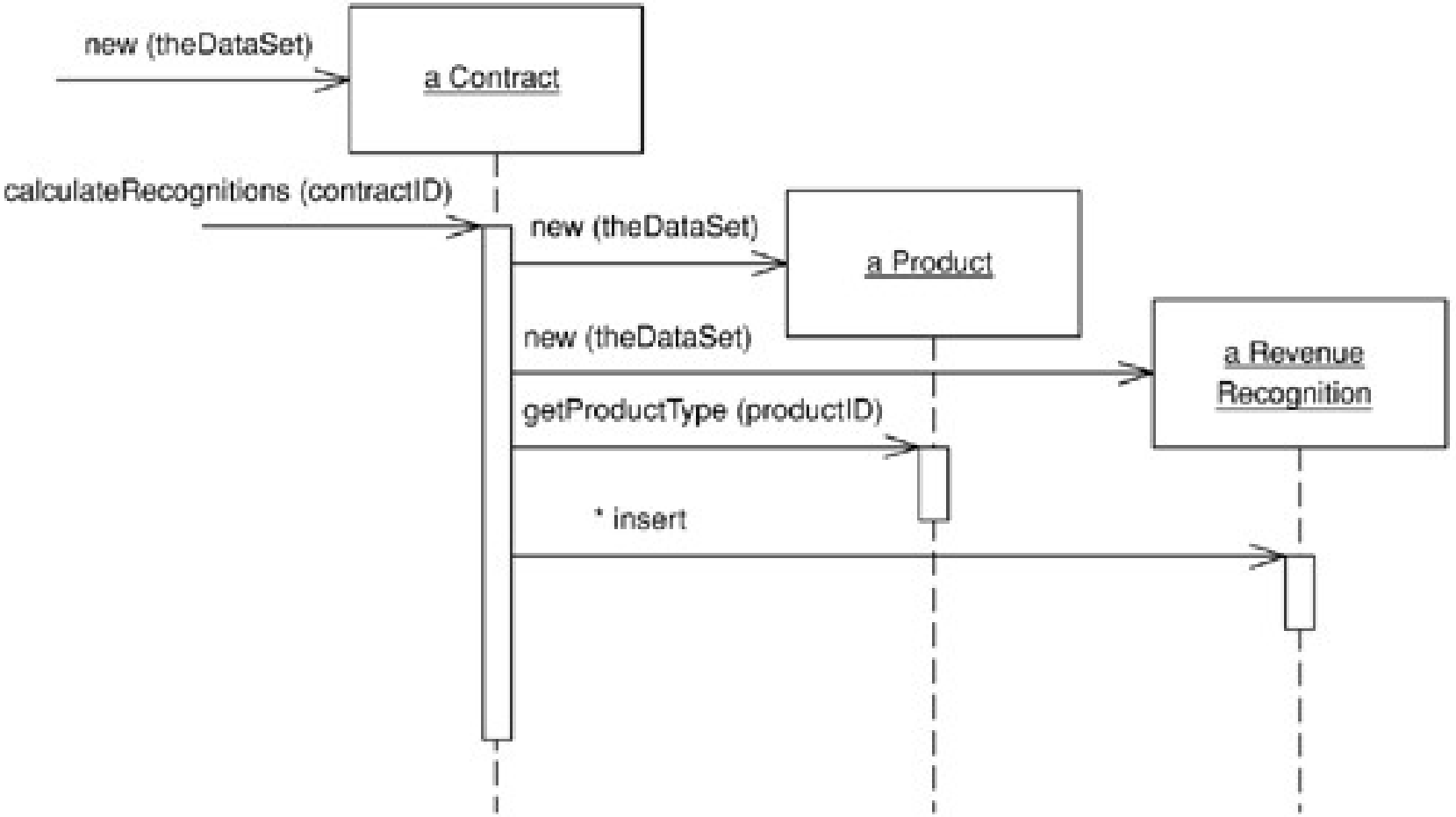
Модель предметной области

- Преимущества:
  - Перераспределение логики между объектами
  - Сложность алгоритмов перетекает в связи между объектами
  - Масштабируемость
- Недостатки:
  - Требует опыта «объектного мышления»
  - Более трудоемка
  - Множество связей затрудняет понимание

Модуль таблицы

- Основывается на табличной структуре данных
- Объект, охватывающий логику обработки всех записей хранимой или виртуальной таблицы
- Создание по одному классу на каждую таблицу
- Экземпляр содержит всю логику обработки данных таблицы
- Объект содержит множество записей (Record Set)
- Объединяет данные и методы

Пример модуля таблицы



Пример модуля таблицы

```
class Contract...
        public void calculateRecognitions (long contractID) {
                DataRow contractRow = this[contractID];
                Decimal amount = (Decimal)contractRow["amount"];
                RevenueRecognition rr = new RevenueRecognition (table.DataSet);
                Product prod = new Product(table.DataSet);
                long prodID = GetProductId(contractID);
                if (prod.GetProductType(prodID) == ProductType.WP) {
                        rr.Insert(contractID, amount, (DateTime) GetWhenSigned(contractID));
                } else if (prod.GetProductType(prodID) == ProductType.SS) {
                        Decimal[] alloc = allocate(amount,3);
                        rr.Insert(contractID, alloc[0], GetWhenSigned(contractID));
                        rr.Insert(contractID, alloc[1], GetWhenSigned(contractID).AddDays(60));
                        rr.Insert(contractID, alloc[2],  GetWhenSigned(contractID).AddDays(90));
                } else if (prod.GetProductType(prodID) == ProductType.DB) {
                        ...
                } else throw new Exception("invalid product id");
        }

class RevenueRecognition...
        public long insert (long contractID, Decimal amount, DateTime date) {
                DataRow newRow = table.newRow();
                long id = GetNextID();
                newRow["ID"] = id;
                newRow["contractID"] = contractID;
                newRow["amount"] = amount;
                newRow["date"]= String.Format("{0:s}", date);
                table.Rows.Add(newRow);
                return id;
        }
```

Модуль таблицы

- Преимущества:
  - Учитывает структуру хранения данных
  - Объединение данных и логики
  - Множественная обработка данных
- Недостатки:
  - Затруднены связи между объектами
  - Сложность применения полиморфизма
  - Зависимость от способа хранения данных

Хранилище (Repository)

- Позволяет написать тесты для бизнес-логики до проектирования слоя хранения
- Паттерн уровня слоя хранения
- Выступает в роли посредника между слоем бизнес-логики и слоем хранения, предоставляя интерфейс в виде коллекции для доступа к объектам домена
- Достигается изоляция и односторонняя зависимость между слоем отображения и слоем домена

Хранилище (Repository)

```
public class Person {
    …
}

public class PersonRepository extends Repository {
    public void save(Person person);
    public List getAll();
    public Person findById(int personId);

}

abstract class Repository {
    …
}
```

далее..

Шаблоны проектирования