ПРОЕКТИРОВАНИЕ АРХИТЕКТУРНОЙ МОДЕЛИ УНИВЕРСАЛЬНОГО СКРИПТА ЗАПИСИ КЛИЕНТОВ БЛОКА «СЕРВИС» СЕТИ АВТОМОБИЛЬНЫХ ДИЛЕРСКИХ ЦЕНТРОВ

УДК 004.9

Александра Юрьевна Скорнякова,
аспирант кафедры Математического обеспечения информационных систем и инноватики, Московский государственный университет экономики, статистики и информатики (МЭСИ)
Тел.: 8 (985) 129-12-13
Эл. почта: a.y.skornyakova@gmail.com

В статье приводится описание метода проектирования модели универсального скрипта записи клиентов, обсуждается применение паттерна «Фабрика» для решения проблемы добавления в систему новых дилерских центров и предлагается возможная оптимизированная модель программного средства.

Ключевые слова: *информационные системы, поддержка принятия решений, трехуровневая архитектура, дилерский центр, блок «Сервис», автоматизация, «Фабричный метод».*

Alexandra Y. Skornyakova,
Post-graduate student, the Department of Software of Information Systems and Innovations, Moscow State University of Economics, Statistics and Informatics (MESI)
Tel.: +7(985)129-12-13
E-mail: a.y.skornyakova@gmail.com

DEVELOPMENT OF ARCHITECTURAL MODEL OF UNIVERSAL RECORDING SERVICE DEALER SYSTEM SCRIPT

In the present article there is a review of the development of architectural model of universal recording service dealer system script, the use of pattern «Factory» to solve the problem in adding new dealerships is discussed and a possible optimized system model is suggested.

Keywords: *Information Systems, decision-making support, Multitier architecture, dealer center, block «Service», automation, pattern "Factory".*

1. Введение

Информация в современном мире превратилась в один из наиболее важных ресурсов, а информационные системы (ИС) стали необходимым инструментом практически во всех сферах деятельности. Разнообразие задач, решаемых с помощью ИС, привело к появлению множества разнотипных систем, отличающихся принципами построения и заложенными в них правилами обработки информации.

Информационные системы можно классифицировать по целому ряду различных признаков. В основу рассматриваемой классификации положены наиболее существенные признаки, определяющие функциональные возможности и особенности построения современных систем. В зависимости от объема решаемых задач, используемых технических средств, организации функционирования, информационные системы делятся на ряд групп (классов).

Предположим, была поставлена конкретная задача: «Разработать единую систему управления блоком «Сервис» (Service Shop) для использования в мульти брендовой сети дилерских центров». Проведя анализ предлагаемой к решению задачи, можно классифицировать ее по нескольким признакам:

По масштабу применения данная система является *корпоративной информационной системой*, т.к. призвана поддерживать работу территориально-распределенной сети.

По сфере применения данную систему можно классифицировать, *как информационную систему организационного управления*. Основным назначением системы будет автоматизация функций управленческого персонала, перспективное и оперативное планирование, оперативный контроль и регулирование.

С точки зрения программно-аппаратной реализации данная ИС является *клиент-серверной.* База данных и СУБД находятся на сервере, а на рабочих станциях находятся клиентские приложения.

При разработке такой системы потребуется строгая регламентация процесса проектирования ИС и обеспечение управления этим процессом с тем, чтобы гарантировать выполнение требований как к самой ИС, так и к характеристикам процесса разработки. Основными задачами, которые стоят перед проектировщиком подобной ИС, являются следующие:

- обеспечивать создание корпоративных ИС, отвечающих целям и задачам организации, а также предъявляемым требованиям по автоматизации деловых процессов заказчика;
- гарантировать создание системы с заданным качеством в заданные сроки и в рамках установленного *бюджета проекта*;
- поддерживать удобную дисциплину сопровождения, модификации и наращивания системы;
- обеспечивать преемственность разработки, т.е. использование в разрабатываемой ИС существующей информационной инфраструктуры.

2. Структура сервиса в дилерском центре

Дилерская сеть – это совокупность посредников компании-производителя, которые помогают продвигать ее продукцию конечному потребителю [1]. Сервис в общей системе дилерского автоцентра обычно фигурирует в качестве одного или нескольких самостоятельных подразделений, среди которых можно выделить следующие крупные блоки:

• Слесарный и кузовной цеха с сопутствующей им информационной инфраструктурой (информация о типах и специфике постов, информация об обрабатываемых брендах и т.п.);

• Служба клиентской поддержки, включающая в себя первичную запись на сервис и наглядное отслеживание всех этапов работ над автомобилем;

• Управление персоналом, предназначенное для работы с графиками сотрудников и разделению оных по правам и должностям.

Подобная схема построения дилерского сервиса типична и оптимальна, если дилерский центр представляет продукцию одного бренда. Не составляет особенной проблемы разработать системы автоматизации работы с данными блоками. Если же брендов несколько, то неминуемо начинают возникать различные проблемы, вызванные специфическими требованиями импортеров.

К примеру, импортер «TOYOTA» требует от своих дистрибьюторов следования системе стандартов TSM (Toyota Customer Service Management). Основой программы TSM служит производственная система Тойота (Toyota Production System), которая является наиболее известным и экономичным процессом производства в мире. Подробнее узнать про нее можно в книге Джеффри Лайкера [2].

Вкратце же следует отметить, что основная задача производственной системы Тойоты — связать воедино и обеспечить выравнивание, синхронизацию и производство в потоке единичных изделий на более ранних процессах. В частности, следование этому принципу при разработке программного средства приведет нас к тому, что требования к работе подразделений блока сервис будут существенно отличаться от аналогичных требований иных брендов. Но у других брендов может также быть своя специфика, отличающаяся от требований тойоты.

Из всего вышесказанного вытекает проблема проектирования. Требуется спроектировать систему так, чтобы она была легко расширяема в случае добавления новых брендов, т.е. следовала пяти принципам дизайна классов SOLID (Single responsibility, Open-closed, Liskov substitution, Interface segregation и Dependency inversion), и позволяла реализовывать одни и те же действия для различных брендов без множественных ветвлений в структуре системы. При этом со стороны пользователей работа на любом дилерском центре сети с любым брендом должна выглядеть единообразно.

Решить эту проблему могут помочь паттерны проектирования, а именно паттерн «Фабрика» (Factory Method).

3. Модель скрипта записи

Паттерн «Фабрика»

Фабричный метод – паттерн, порождающий классы, также известен под именем виртуальный конструктор (Virtual Constructor).

Данный паттерн определяет интерфейс для создания объекта, но оставляет подклассам решение о том, какой класс инстанцировать. Фабричный метод позволяет классу делегировать инстанцирование подклассам (Рис. 1).

Паттерн фабричный метод стоит использовать, когда:

• Классу заранее неизвестно, объекты каких классов ему нужно создавать;

• Класс спроектирован так, чтобы объекты, которые он создает, специфицировались подклассами;

• Класс делегирует свои обязанности одному из нескольких вспомогательных подклассов, и вы планируете локализовать знание о том, какой класс принимает эти обязанности на себя.

• Product – определяет интерфейс объектов, создаваемых фабричным методом;

• ConcreteProduct – реализует интерфейс Product;

• Creator – объявляет фабричный метод, возвращающий объект типа Product. Может вызывать фабричный метод для создания объекта Product;

• ConcreteCreator – замещает фабричный метод, возвращающий объект ConcreteProduct;

Создатель «полагается» на свои подклассы в определении фабричного метода, который будет возвращать экземпляр подходящего конкретного продукта.

Фабричные методы избавляют проектировщика от необходимости встраивать в код зависящие от приложения классы. Код имеет дело только с интерфейсом класса Product, поэтому он может работать с любыми определенными пользователями классами конкретных продуктов.

Подробнее об этом паттерне и других рассказано в книге «Приемы объектно-ориентированного проектирования. Паттерны проектирования» [3].

Применение паттерна «Фабрика» для решения задачи проектирования класса работы с визитами в составе блока «Сервис» в мульти брендовой сети дилерских центров.

Программирование на уровне интерфейса позволяет оградить себя от многих изменений в системе. Благодаря полиморфизму код, написанный для интерфейса, будет работать с любыми новыми классами, реализующими этот интерфейс.

Независимо от бренда известно, что каждый визит должен уметь выполнять следующие действия:

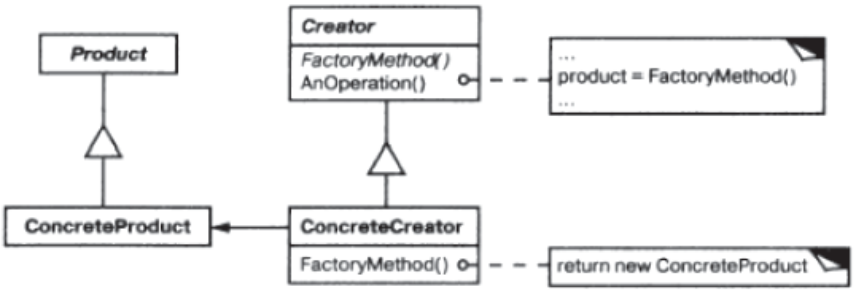

Рис. 1. *Структура фабричного метода*

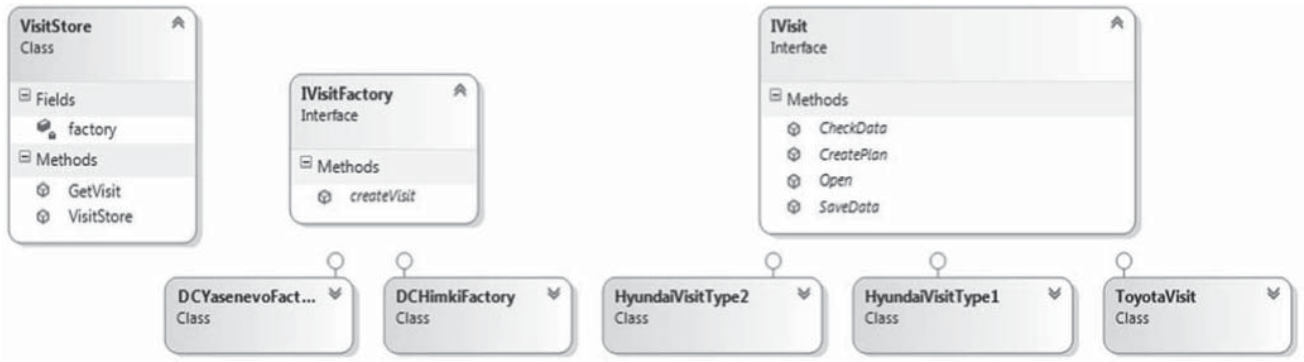

Рис. 2. *Диаграмма классов проектного решения*

• Открытие для редактирования;

• Проверка корректности введенных данных;

• Планирование в цех;

• Сохранение данных.

Следовательно, интерфейс для работы с визитами может быть определен следующим образом:

```
interface IVisit
{
    void Open();
    void CheckData();
    void CreatePlan();
    void SaveData();
}
```

Начнем с самой фабрики. Текущая задача – определить класс, инкапсулирующий создание объектов для визитов всех брендов. В фабрике определяется метод createVisit(), который будет использоваться всеми клиентами для создания новых визитов. Код параметризуется по типу визита. Выглядеть это может следующим образом:

```
public class SimpleVisitFactory
{
    public IVisit createVisit(string brand)
    {
        IVisit visit = null;
        if (brand.Equals("HYUNDAI"))
            visit = new HyundaiVisit();
        else if (brand.Equals("TOYOTA"))
            visit = new ToyotaVisit();

        return visit;
    }
}
```

Таким образом работу со списком визитов можно построить так, чтобы все операции создания объектов выполнялись фабрикой. Классу *VisitStore* передается ссылка на *SimpleVisitFactory*. Метод *GetVisit*() обращается к фабрике с запросом на создание объекта, передавая выбранный бренд. Вызов оператора *new* заменяется вызовом метода create объекта фабрики, что позволяет избежать созданий экземпляров конкретного типа.

```
public class VisitStore
{
    SimpleVisitFactory factory;
    public VisitStore(SimpleVisitFactory _factory)
    {
        this.factory = _factory;
    }

    public IVisit GetVisit(string select_visit_brand)
    {
        IVisit visit;
        visit = factory.createVisit(select_visit_brand);
        visit.Open();
        return visit;
    }
}
```

Однако различия в работе с визитами могут быть не только у брендов, но также и у различных дилерских центров сети. Следовательно, в программе нужно предусмотреть возможность предлагать различные стили работы с брендированными визитами в зависимости от их местонахождения.

Все дилерские центры сети должны использовать код *VisitStore*(), чтобы все визиты обрабатывались по единым правилам. *SimpleVisitFactory* заменяется несколькими различными фабриками: *DCHimkiFactory*, *DCDiamantFactory*, *DCYasenevoFactory*. В этом случае *VisitStore* связывается с подходящей фабрикой посредством композиции. Это позволит нам, к примеру, завести отдельные классы для визитов HYUNDAI на различных ДЦ, а для визитов TOYOTA использовать один и тот же класс.

```
DCHimkiFactory himki = new DCHimkiFactory();
VisitStore visit1 = new VisitStore(himki);
visit1.GetVisit("HYUNDAI");

DCYasenevoFactory yasenevo = new DCYasenevoFactory();
VisitStore visit2 = new VisitStore(yasenevo);
visit2.GetVisit("HYUNDAI");
```

В итоге работа с визитами в сети мульти брендовых дилерских центров может быть спроектирована способом, представленным на диаграмме классов (рис. 2):

4. Архитектура программного решения автоматизации предложенной модели

При проектировании решения данной задачи применялась трехуровневая архитектура.

В компьютерных технологиях трёхуровневая архитектура, синоним трёхзвенная (англ. three-tier или Multitier architecture) предполагает наличие следующих компонентов приложения: клиентское приложение (обычно говорят «тонкий клиент» или терминал), подключенное к серверу приложений, который в свою очередь подключен к серверу базы данных (рис. 3).

Все функции по формированию пользовательского интерфейса реализуются на клиенте, все функции по управлению данными – на сервере. Реализация бизнес-правил делится между клиентом и сервером, используя механизмы программирования сервера (хранимые процедуры, триггеры, представления и т.п.). Таким образом, в трехуровневом приложении можно выделить третий, промежуточный уровень, реализующий бизнес-правила, которые являются наиболее часто изменяемыми компонентами приложения. В текущей работе представлена классическая трехуровневая модель.

Слой представления данных

Слой представления данных представлен классом VizitScriptEdit.cs и выполнен в Windows Forms. В данном классе содержатся следующие группы методов:

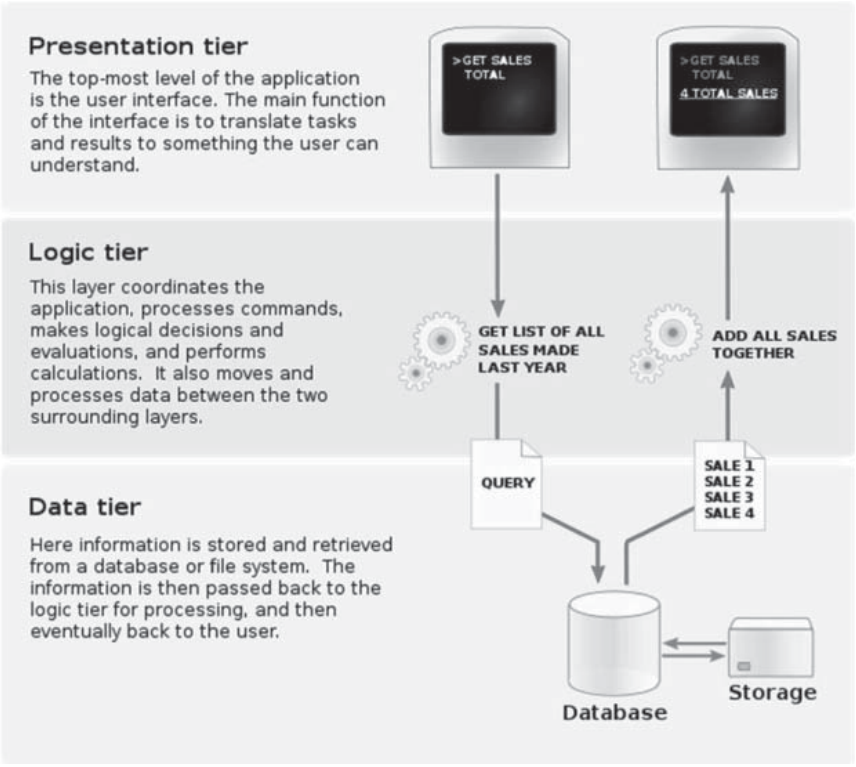

Рис. 3. *Трёхуровневая архитектура информационной системы*

• Собственная логика формы, а именно методы, отвечающие за содержание текстовых полей и внешний вид, управляемый из фабрики:

• Логика переходов между страницами, включающая в себя методы для проверки доступности шагов 1..*n* при нахождении на шаге *i* (*i* <= n) и методы обработки события Click на различных управляющих контролах;

• Логика обработки полей на форме, представленная методами инициирования изменений в различных логически связанных группах полей;

• Методы вызывающиеся в результате возведения событий со слоя логики.

Слой представления отвечает только за передачу данных между пользователем и слоем логики. Никакие действия, связанные с бизнес-процессами, на нем не выполняются. К реализации слоя представления данных в соответствии с описанной ранее фабричной архитектурой применяется ряд следующих обязательных критериев.

Во-первых, на нем должна быть предусмотрена возможность настройки содержимого всех статичных текстовых полей скрипта записи, названий шагов и т.д. Логика настройки, включающая в себя конкретизацию по ДЦ и/или бренду должна быть описана на уровне интерфейса и применима к любым экземплярам скрипта записи.

Во-вторых, необходимо предусмотреть возможность настройки проверки корректности заполнения полей. В частности, набор обязательных полей на различных шагах может сильно разниться от бренда к бренду. Другим примером является формат телефона, которых будет различным для ДЦ Hyundai в России, Белоруссии, Казахстане и других странах.

Слой логики и слой данных

Слой логики представлен классом LG_FastVizitScriptEdit.cs. На него вынесена вся логика обработки данных. На этом слое происходит заполнение различных блоков переменных в зависимости от переданных со слоя представления сообщений. К примеру, в момент выбора на слое представления существующего клиента его идентификатор передается на слой логики и инициирует загрузку списка автомобилей по клиенту, который в свою очередь тот час же отображается на слое представления. В соответствии с фабричным методом, в зависимости от типа созданного экземпляра скрипта записи на этом слое могут быть изменены алгоритмы расчета расписания и поиска данных.

При работе со слоем требуется предусмотреть возможность возвращаться на любой предыдущий шаг для изменения данных. При возврате на предыдущие шаги, при попытке изменения данных, инициировать запрос об удалении всех несовместимых ранее заполненных данных на последующих шагах. При согласии эти данные должны удаляться по всем последующим шагам, при отказе изменения, произведенные пользователем, откатываются.

В момент загрузки слой логики обращается к слою данных, который непосредственно общается с базой данных. Слой данных представлен классом VisitBD.cs. Особенностью данного слоя является то, что все его методы описываются единым образом, а именно как:

public IDictionary<String, Object>
ИМЯ_МЕТОДА
(<Список_параметров>);

Таким образом достигается максимальная независимость слоя логики от слоя данных, т.к. не происходит завязки на конкретную базу данных ни на одном слое, кроме слоя данных. А использование описанного ранее фабричного метода позволяет уточнять детали реализации непосредственно для каждого слоя.

5. Заключение

Описанные в статье методы проектирования позволяют добиться высокого уровня гибкости проектируемого программного средства. Встраивание обособленной логики для различных дилерских центров, относительно стандартного поведения системы, становится возможным за минимальное время. Так же приведенный шаблон проектирования позволяет добавлять в систему и производить запуск новых дилерских центров, поддерживающих

ПРОБЛЕМА	РЕШЕНИЕ	РЕАЛИЗАЦИЯ
Дублирование больших объёмов кода при добавлении в систему логики работы с новыми брендами	Применение паттерна «Фабрика» для получения возможности частичного разделения логики	Разработка единой формы скрипта записи, класса, отвечающего за общую логику процесса и закладывание возможности безболезненного конкретизирования логики для отдельных брендов.
Сильная взаимосвязь брендовой логики и внешнего представления данных на форме	Применение трехуровневой архитектуры	Разработка единой формы скрипта записи с применением трехуровневой архитектуры

блок «Сервис», практически полностью избегая редактирования уже существующих блоков кода. Как следствие, можно сделать вывод о хорошем соответствии программы пяти основным принципам дизайна классов в объектно-ориентированном проектировании SOLID.

Использование в бизнес-процессах блока «Сервис» автомобильного дилерского центра автоматизации предложенной архитектуры программного средства позволяет решить следующие проблемы (табл.).

Как видно из приведенной таблицы, одним из главных достоинств внедрения предложенной архитектуры является переход от большого количества несвязанных друг с другом брендовых форм к единой схеме, гарантирующей высокие целостность, связность, постоянную актуальность и простоту конкретизации поведения в зависимости от пожеланий различных дистрибьюторов.

Литература

1. Сальников О. В. Процессный подход к развитию дилерских сетей. 3, Ульяновск : б.н., 2007 г., Проблемы современной экономики.

2. Лайкер, Джеффри К. ДАО TOYOTA 14 принципов менеджмента ведущей компании мира. б.м. : Альпина Бизнес Букс, 2005. 5-9614-0124-3.

3. Э.Гамма, Р. Хелм, Р. Джонсон, Д. Влиссидес. Приемы объектно-ориентированного проектирования. Паттерны проектирования. Санкт-Петербург : Питер, 2013. 978-5-459-01720-5.

4. СкорняковаА.Ю. Исследование состава и структуры информационных потоков сервисных подразделений автомобильного дилерского центра. Возможная модель их оптимизации. Москва : МЭСИ, 2013.

5. Скорнякова А.Ю., Комлева Н.В. Применение паттерна «Фабрика» при проектировании распределенной клиент-серверной системы на примере блока «Сервис» сети дилерских центров // Современные информационные технологии в управлении и образовании: материалы двенадцатой научно-практической конференции, 18 апреля 2013 г., сб. Научных трудов, часть 2, ФГУП «Научно-исследовательский институт «Восход», М., 2013. С. 112–121.

References

1. Salnikov O. V.. Process approach to the development of dealerships. Yliyanovsk 2007.

2. Jeffrey K. Liker The Toyota Way: 14 Management Principles from the World's Greatest Manufacturer. Alpina Business Books 2005.

3. E. Gamma, R. Helm, R. Johnson, J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. St.Petersburg 2013.

4. Skornyakova A.Y. Research of the composition and structure of information flows car dealership service departments center. Possible model for optimization. MOSCOW: MESI, 2013.

5. Skornyakova A. Y., Komleva N.V. Use of pattern "Factory" in designing of distributed client-server system on an example the block "Service" dealerships. Moskow 2013.