

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ  
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«Рязанский государственный радиотехнический университет  
имени В. Ф. Уткина»**

## **ПРОЕКТИРОВАНИЕ ПРОГРАММНЫХ СИСТЕМ**

Методические указания к курсовому проектированию

**Рязань 2022**

**Проектирование информационных систем:** Методические указания к курсовому проектированию / Рязан. гос. радиотехн. ун-т. им. В.Ф. Уткина; Сост.: В.В. Белов, В.И. Чистякова. Рязань, 2022. 40 с.

Излагаются методические указания по изучению материалов, необходимых для выполнения курсового проекта и оформления пояснительной записки к выполненному проекту по дисциплине «Проектирование программных систем». Теоретическую платформу указаний образуют объектно-ориентированная методология проектирования и систематический подход к проектированию, основанный на использовании каскадной модели жизненного цикла программно-информационной системы, принципа ФИСАП и компонентно-ориентированного программирования.

Предназначены бакалаврам всех форм обучения направления 09.03.04 «Программная инженерия».

Библиогр.: 5 назв.

*Жизненный цикл, модель, анализ, проектирование, реализация, UML, архитектура, поведение, класс, сценарий, автомат, компонент.*

Рецензент: кафедра вычислительной и прикладной математики Рязанского государственного радиотехнического университета (зав. кафедрой д-р техн. наук, проф. Г.В. Овечкин)

#### Проектирование программно-информационных систем

Составители   Б е л о в Владимир Викторович  
                      Ч и с т я к о в а Валентина Ивановна

Подписано в печать XX.XX.22. Усл. печ. л. 2.5.

Тираж XX экз.

Рязанский государственный радиотехнический университет  
390005, Рязань, ул. Гагарина, 59/1.

Редакционно-издательский центр РГРТУ.

## СОДЕРЖАНИЕ

<b>I. ПРЕДВАРИТЕЛЬНЫЕ ЗАМЕЧАНИЯ .....</b>	<b>5</b>
<b>II. СИСТЕМАТИЧЕСКИЙ ПОДХОД</b>	
<b>К ПРОЕКТИРОВАНИЮ ПриС .....</b>	<b>5</b>
<b>III. РЕКОМЕНДУЕМЫЕ ЭЛЕМЕНТЫ</b>	
<b>ПОЯСНИТЕЛЬНОЙ ЗАПИСКИ.....</b>	<b>9</b>
<b>ЗАДАНИЕ .....</b>	<b>9</b>
<b>СОДЕРЖАНИЕ .....</b>	<b>9</b>
<b>ВВЕДЕНИЕ .....</b>	<b>10</b>
<b>1. ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПриС.....</b>	<b>11</b>
1.1 Модель предметной области AS IS .....	11
1.1.1 Функциональный аспект .....	11
1.1.2 Информационный аспект.....	11
1.1.3 Структурно-архитектурный аспект.....	12
1.1.4 Аспект поведения .....	14
1.2 Обоснование необходимости создания ПриС .....	15
1.3 Модель требований к проектируемой системе .....	15
1.3.1 Модель TO BE .....	15
1.3.2 Функциональные требования .....	15
1.3.3 Требования к информативности .....	16
1.3.4 Архитектурные требования .....	16
1.3.5 Требования к поведению .....	17
1.3.6 Функциональная спецификация .....	18
<b>2. ПРОЕКТИРОВАНИЕ СИСТЕМЫ .....</b>	<b>18</b>
2.1 Архитектура данных и процессов .....	18
2.2 Модель поведения.....	19
2.3 Модель реализации .....	21
2.3.1 Предварительные замечания .....	21
2.3.2 Модель исходного кода.....	26
2.3.3 Модель исполняемого кода .....	29
2.3.4 Модель поставляемых артефактов.....	32
2.3.5 Модель размещения артефактов .....	34
2.3.6 Модели управления.....	36
2.3.7 Интерфейс пользователя.....	37
<b>3. РЕАЛИЗАЦИЯ ПриС.....</b>	<b>37</b>
3.1 Реализация бизнес-логики.....	37
3.2 Реализация пользовательского интерфейса.....	37
3.3 Реализация базы данных .....	37
3.4 Реализация средств доступа к базе данных .....	38
<b>4. ТЕСТИРОВАНИЕ ПриС.....</b>	<b>38</b>

<b>5. СОСТАВ ПРОЕКТА .....</b>	<b>38</b>
<b>6. КОМПИЛЯЦИЯ И СБОРКА СИСТЕМЫ .....</b>	<b>39</b>
6.1 Правила создания системы (приложения) .....	39
6.2 Состав приложения .....	39
<b>7. ДОКУМЕНТАЦИЯ .....</b>	<b>39</b>
7.1 Назначение системы и выполняемые функции .....	39
7.2 Условия применения системы .....	39
7.3 Правила инсталляции системы .....	39
7.4 Сценарий взаимодействия с пользователем .....	39
<b>ЗАКЛЮЧЕНИЕ .....</b>	<b>40</b>
<b>СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ .....</b>	<b>40</b>
<b>ПРИЛОЖЕНИЯ .....</b>	<b>40</b>
<b>IV. ПРАВИЛА СДАЧИ ПРОЕКТА</b>	
<b>РУКОВОДИТЕЛЮ ПРОЕКТА.....</b>	<b>40</b>
<b>V. ИНСТРУМЕНТАЛЬНЫЕ ТРЕБОВАНИЯ .....</b>	<b>40</b>
<b>БИБЛИОГРАФИЧЕСКИЙ СПИСОК .....</b>	<b>41</b>

## I. ПРЕДВАРИТЕЛЬНЫЕ ЗАМЕЧАНИЯ

Предполагается, что читатели знакомы с основными положениями изучаемой дисциплины и достаточно тщательно изучили учебники [1] и [2].

Если нет специальных указаний, то для оформления пояснительной записки рекомендуется использовать шаблон (файл **3. ПЗ Шаблон.dot**) из раздаточных методических материалов к курсовому проектированию.

Подписуточные подписи, заголовки таблиц, ссылки на рисунки и таблицы, нумерацию формул следует выполнять в соответствии с ГОСТ 2.105-95 и документом «Правила оформления записки к курсовому проекту» в файле **5. Oformlenie\_KP.pdf** из раздаточных методических материалов к курсовому проектированию.

## II. СИСТЕМАТИЧЕСКИЙ ПОДХОД К ПРОЕКТИРОВАНИЮ

Предлагаемый систематический подход основан на следующих положениях:

1) каскадное представление фаз жизненного цикла программно-информационной системы (ПРИС);

2) принцип ФИС/АП (акроним от слов Функциональность – Информационность – Структура/Архитектура – Поведение);

3) принцип иерархичности/декомпозиции;

4) представление принимаемых проектных решений в виде системы моделей:

а) модель предметной области;

б) модель требований;

в) модель проектирования:

\*----- Предварительное проектирование -----\*

в1) модель логической архитектуры

(архитектура данных и процессов);

\*----- Детальное проектирование -----\*

в2) модель поведения:

в3) модель физической архитектуры (модель реализации):

в31) модель исходного кода;

в32) модель исполняемого кода;

в33) модель артефактов, поставляемых Заказчику;

в34) модель базы данных;

в35) модель размещения артефактов по устройствам ПРИС.

Модель «в1» создаётся на этапе предварительного проектирования. Модели групп «в2» и «в3» создаются на этапе детального проектирования.

### **Фазы и этапы жизненного цикла программно-информационной системы**

---

Белов В.В., Чистякова В.И. Проектирование программных систем. Методические указания к курсовому проектированию

В процессе создания (конструирования, разработки) и использования программно-информационной системы можно выделить следующие состояния (фазы) и этапы.

1. Анализ.

- 1.1. Обследование предметной области, завершаемое созданием модели предметной области.
- 1.2. Техничко-экономическое обоснование (ТЭО) проекта.
- 1.3. Изложение требований в форме технического задания (ТЗ) и модели требований.

2. Проектирование.

- 2.1. Предварительное (архитектурное) проектирование – создание модели архитектуры ПриС.
- 2.2. Окончательное (алгоритмическое) проектирование – создание модели поведения ПриС, модели данных ПриС и модели реализации ПриС.
- 2.3. Разработка пользовательского интерфейса (GUI).

3. Реализация (программирование, кодирование).

4. Тестирование.

5. Внедрение.

6. Сопровождение.

7. Извлечение из эксплуатации.

Указанные фазы и этапы в графическом виде представлены на рисунке 1.1. Они образуют жизненный цикл программно-информационной системы (ЖЦ ПриС) в версии методологии, называемой классической, или каскадной, или водопадной (Waterfall model). Эти же фазы и этапы составляют кардинальное содержание итеративных методологий, включая Agile. Заметим, что перечень этапов приведен только для двух первых фаз, представляющих наибольший интерес в рамках изучаемой дисциплины.

Важно понимать, что представленная каскадная модель в «чистом» виде представляет собой некоторую теоретическую парадигму, которую вряд ли можно реализовать в реальном проектировании. Практически с неизбежностью возникают необходимости возвратов на предыдущие этапы и даже фазы. Поэтому существует большое количество модификаций Waterfall, которые не только предполагают повторы этапов и фаз, но и отражают дополнительные аспекты жизненного цикла, вполне естественные и правильные, но не концептуальные, а уточняющие, дополняющие, детализирующие – такие как параллельное выполнение работ, предпроектирование, тщательное тестирование результатов каждой работы и тому подобное. В некоторых случаях акцентуация таких дополнений оказываются основанием для объявления самостоятельности некоторых видов субмоделей, подобных V-модели, главной особенностью которой является «усиленный разговор» о тестировании, ве-

рификации и валидации, реализация которых целесообразна в рамках любой модели без исключения.

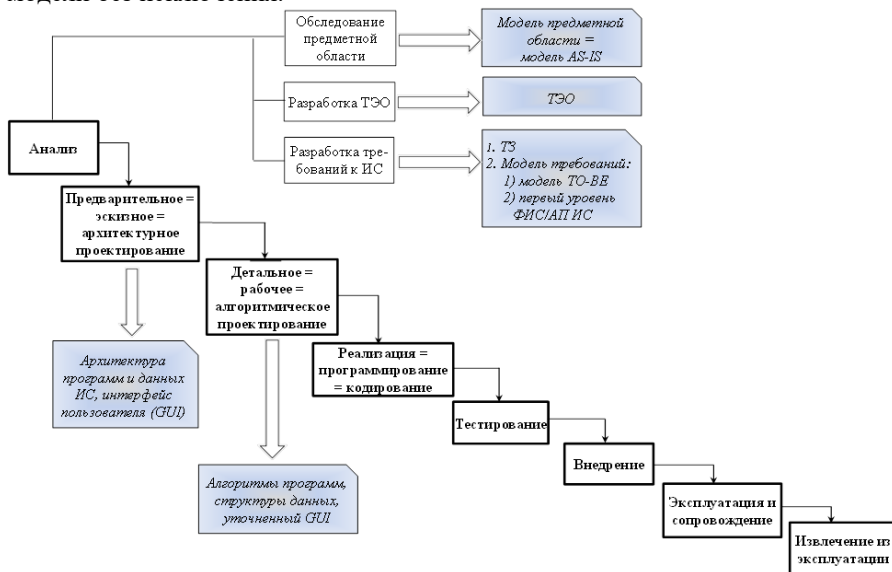


Рисунок 1.1 – Каскадная модель жизненного цикла ПриС

## Принцип ФИС/АП

Имеются следующие основные (фундаментальные) аспекты рассмотрения ПриС как в процессе её изучения, так и в процессе её создания.

1. Функциональность.
2. Информационность.
3. Структура/Архитектура.
4. Поведение.

Функциональность – это совокупность всех работ (задач, функций) выполняемых без использования либо с использованием ПриС.

Информационность – это совокупность всех видов информации, потребляемых и создаваемых в предметной области и в ПриС в процессе её функционирования, включая исходные данные, результаты вычислений, запоминаемые промежуточные значения, знания в форме регламентирующих документов, алгоритмов решения задач, бизнес-правил и прочее.

Структура/Архитектура – это совокупность всех исполнителей функциональности и носителей информационности, связей между ними и мест их размещения.

Поведение – это совокупность алгоритмов и сценариев, реализуемых в предметной области без использования либо с использованием ПриИС. Алгоритмы описывают способы решения содержательно сложных задач.

Принцип ФИС/АП заключается в утверждении, что указанные фундаментальные аспекты рассмотрения ПриИС имеют место в каждой из фаз, на каждом этапе ЖЦ ПриИС при создании всех моделей, но с различной степенью значимости (важности, интенсивности). На рисунке 1.2 показаны уровни значимости фундаментальных аспектов рассмотрения ПриИС при создании основных моделей ЖЦ.

Как видно из рисунка 1.2, при создании модели предметной области и модели требований наивысший (абсолютный) уровень значимости имеет аспект функциональности, т. е. описание работ (задач), выполняемых в модели предметной области без использования, а в модели требований – с использованием ПриИС.

Существенное различие значимостей аспекта поведения в моделях предметной области и требований обусловлено следующим:

1) поведение в предметной области исчерпывается правилами (сценариями) «ручного» выполнения актуальных работ, описание которых может быть весьма поверхностным,

2) в модели требований описывается выполнение актуальных работ с помощью ПриИС, и поэтому это описание должно быть абсолютно полным и точным.

Абсолютная значимость аспекта структуры/архитектуры в модели предварительного проектирования обусловлена тем, что главная работа при этом состоит в том, что окончательно определяются все элементы логической структуры ПриИС – классы, а при необходимости модули, структуры и иные типы.

Высокая значимость аспекта структуры/архитектуры в модели окончательного проектирования обусловлена тем, что на этапе окончательного проектирования определяются элементы физической структуры ПриИС – важнейшие артефакты: файлы исходных, объектных и исполняемых кодов, артефакты, поставляемые Заказчику и устройства, в которых размещаются поставляемые артефакты.



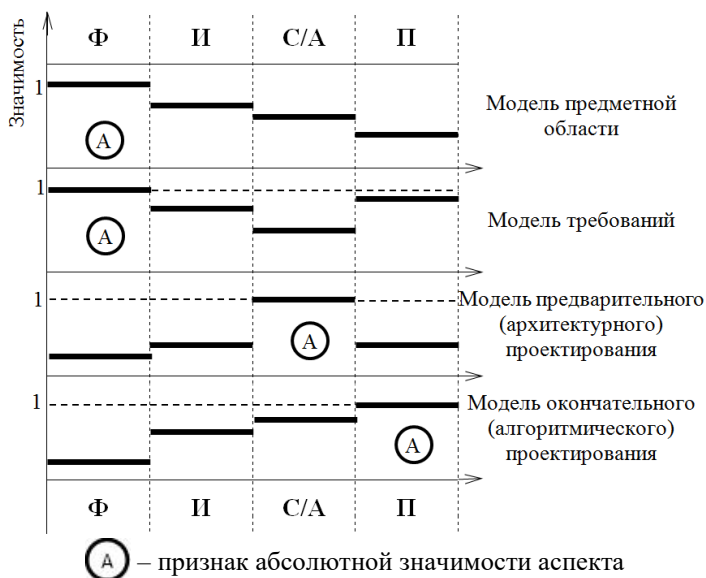


Рисунок 1.2 – Уровни значимости фундаментальных аспектов рассмотрения ПриС (ФИС/АП) при создании основных моделей фаз анализа и проектирования

### III. РЕКОМЕНДУЕМЫЕ ЭЛЕМЕНТЫ ПОЯСНИТЕЛЬНОЙ ЗАПИСКИ

Обратим внимание на тот факт, что структура элементов данного раздела совпадает со структурой пояснительной записки к курсовому проекту: заголовки элементов данного раздела в своей совокупности образуют оглавление пояснительной записки.

#### ЗАДАНИЕ

Бланк **Задания** содержится в раздаточных методических материалах к курсовому проекту.

#### СОДЕРЖАНИЕ

**Содержание** должно быть создано с помощью средства «Добавление оглавления в документ». Ручное формирование содержания без возможностей автоматического обновления и переходов по гиперссылкам категорически запрещено.

## ВВЕДЕНИЕ

Главной целью **Введения** должно быть описание предметной области, определение объекта рассмотрения и предмета автоматизации. Во введении необходимо также сформулировать цель проекта и обосновать необходимость или целесообразность разработки.

Предметной областью проектируемой ПриС может являться некоторое подразделение предприятия или некоторый аспект деятельности этого предприятия. Для упрощения изложения вопросов моделирования и функциональной спецификации в качестве заместителя термина «предметная область» часто используется более краткий и образный термин «предприятие».

Определение объекта и предмета автоматизации можно толковать как сужение зоны рассмотрения предметной области. Объект рассмотрения – это часть предметной области, в которой сосредоточены наиболее значимые бизнес-процессы, – наиболее существенно влияющие на результативность социально-экономической деятельности предприятия. Предмет автоматизации – это часть объекта рассмотрения, для которой предполагается создание ПриС в качестве механизма автоматизации (компьютерной поддержки) бизнес-процессов.

Введение должно содержать следующие компоненты:

1) описание предметной области:

- объекты и субъекты, участвующие в формировании и потреблении проблемной информации;
- структура и содержание проблемной информации;
- информационные потоки между объектами и субъектами предметной области;

2) предметная цель проекта, – например, повышение производительности труда сотрудников (указываются единицы измерения производительности), сокращение времени выполнения заказа, уменьшение количества ошибок в технологической и учётной информации, повышение надёжности хранения информации, увеличение числа решаемых задач (указывается конкретно, – например, расширение состава предоставляемых услуг, обеспечение доступа к удалённым источникам информации и т. п.); целью не может быть «разработка программно-информационной системы»;

3) пути и средства достижения цели; базовым средством достижения предметной цели указывается «разработка программно-информационной системы»;

4) оценка современного состояния предметной области в контексте решаемой задачи (анализ прототипов); изложение оценки должно подводить к выводу о целесообразности разработки новой программно-информационной системы;

5) исходные данные для разработки системы;

б) инструментальные средства реализации проекта – перечисляются и кратко характеризуются инструментальные средства, используемые для реализации проекта; обосновывается их использование.

## **1. ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРИС**

### **1.1 Модель предметной области AS IS**

Модель AS IS отражает текущее состояние дел в предметной области, в частности, *текущую* организацию бизнес-процессов. Как любая другая ИТ-модель, модель AS IS должна отражать функциональный, информационный, структурно-архитектурный и поведенческий аспекты ПриС. В общем случае модель содержит тексты, рисунки, таблицы и бизнес-диаграммы UML с текстовыми описаниями потоков действий, реализующих задачи (работы, функции) предметной области и тексты, поясняющие элементы диаграмм.

#### **1.1.1 Функциональный аспект**

Функциональный аспект деятельности предприятия отражается перечислением всех задач (бизнес-процессов), выполняемых предприятием. Пункты *перечисления* должны быть информативными, – содержащими пояснения, позволяющими читателю получить четкое представление о содержании элементов перечисления.

Графически выполняемые задачи (бизнес-процессы) целесообразно представить в виде *диаграммы* бизнес-прецедентов. Тексты пояснений к названиям бизнес-прецедентов помещаются в окно документационного редактора «Documentation» и в окно редактора вложений «Attachment». В «Documentation» помещаются пояснения элементов диаграммы. В «Attachment» прикрепляется Word-документ, содержащий описание потоков действий реализующих бизнес-прецеденты – задачи (работы, функции) – предметной области.

Окна документационного редактора «Documentation», редактора вложений «Attachment» и редактора свойств «Properties» отображаются в правом нижнем углу экрана StarUML, в области, называемой областью инспектора. Выбор нужного окна осуществляется с помощью закладок.

И пояснения «Документирования», и текст прикрепленного файла должны включаться в текст пояснительной записки.

#### **1.1.2 Информационный аспект**

Информационный аспект модели деятельности предприятия отражается *перечислением* и характеристикой всех видов информации и носителей, используемых в процессе реализации бизнес-процессов, включая следующие виды информации:

управляющую – приказы, распоряжения, правила как внешнего, так и внутреннего происхождения;

---

Белов В.В., Чистякова В.И. Проектирование программных систем. Методические указания к курсовому проектированию

- входную, – используемую как исходные данные для решения задач, подлежащую регистрации и хранению;
- выходную, – выдаваемую пользователям системы;
- внутреннюю, – создаваемую внутри системы, имеющую характер важных промежуточных результатов.

Перечисления видов информации целесообразно оформлять в виде таблиц со столбцами: «Название»; «Класс» (Упр., Вход, Выход, Внутр.); «Носитель» (Бумаж., дос-файл, xl-файл); «Место хранения» (Бух.Папка, Бух.Комп и т. п.)

### 1.1.3 Структурно-архитектурный аспект

Структурно-архитектурный аспект предметной области содержит описание схемы управления предприятием, функциональных подразделений предприятия и конкретных *исполнителей* бизнес-процессов. Кроме этого, структурно-архитектурный аспект может дополнять и описание «исполнителей» информационности системы – содержать *подробное* описание форм и мест хранения данных и носителей информации ПриС.

Схема управления предприятием представляется в виде иерархии руководителей и административных подразделений с описанием выполняемых функций.

В описании функциональных подразделений предприятия приводится перечень функциональных подразделений и схема их производственных взаимосвязей. Для каждого подразделения приводится перечень выполняемых функций (задач) с указанием должностей исполнителей бизнес-процессов.

Графически исполнителей бизнес-процессов предметной области следует представить в виде диаграммы объектов, которая строится как диаграмма классов, на которой классы представляются в виде *иконок* и снабжаются стереотипами «CaseWorker» и «InternalWorker» (см. рисунки 1.3 и 1.4 – «Диалоговые окна для назначения стереотипов «CaseWorker» и «InternalWorker» классу»).

CaseWorker – это исполнитель бизнес-процесса, который может взаимодействовать с пользователями (клиентами) предприятия.

InternalWorker – это внутренний исполнитель бизнес-процесса, – взаимодействующий только с другими исполнителями.

Диаграмма объектов должна содержать и «исполнителей» информационности ПриС – классов со стереотипом «entity» (см. рисунок 1.5 – «Диалоговое окно для назначения стереотипа «entity» классу»).

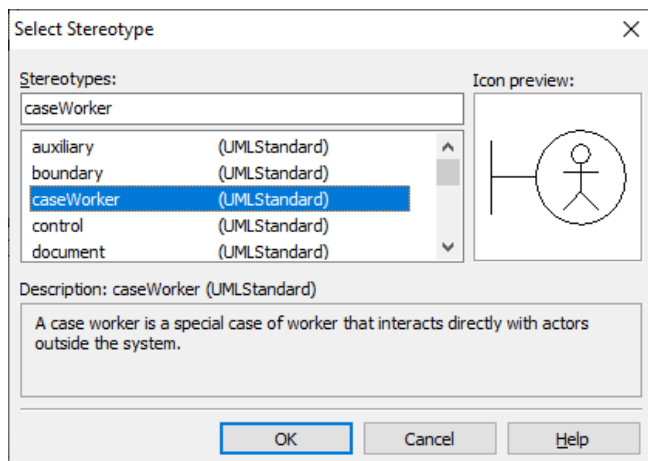


Рисунок 1.3 – Диалоговое окно для назначения стереотипа «CaseWorker» классу

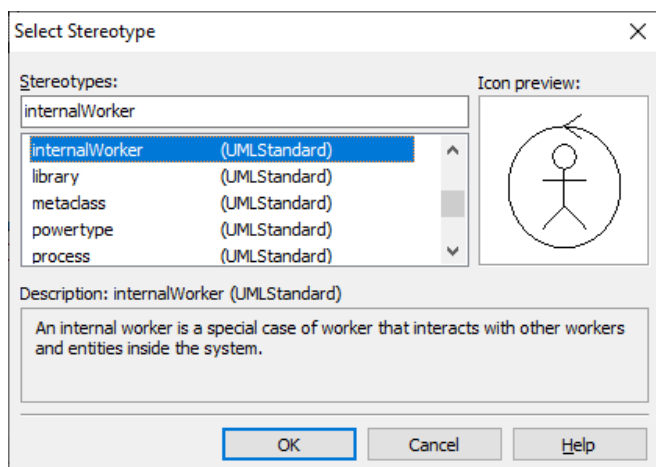


Рисунок 1.4 – Диалоговое окно для назначения стереотипа «InternalWorker» классу

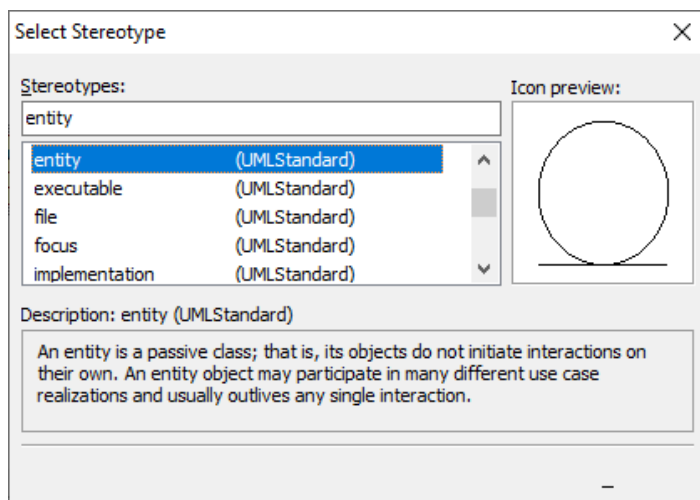


Рисунок 1.5 – Диалоговое окно для назначения стереотипа «entity» классу

#### 1.1.4 Аспект поведения

Аспект поведения – совокупность алгоритмов реализации бизнес-прецедентов и бизнес-сценариев. Алгоритмы изображаются с помощью диаграмм бизнес-деятельности или бизнес-автомата, а сценарии – с помощью диаграмм бизнес-коммуникаций (бизнес-сотрудничества). На диаграммах бизнес-деятельности в названиях дорожек (Swimlane) и на диаграммах бизнес-коммуникаций в названиях объектов следует использовать названия исполнителей бизнес-процессов, определенные в описании структурно-архитектурного аспекта предметной области.

Диаграммы деятельности или автомата, – создаются как «реализации» бизнес-прецедентов. Вид диаграммы выбирается в зависимости от наличия или отсутствия среди деятельностей потенциально бесконечно длящихся, таких как ожидание сообщения или сигнала. Если ожидания имеются, то используется автомат, а иначе – деятельность.

Из диаграмм взаимодействия – последовательности либо коммуникации выбирается та, которая больше импонирует аналитику, поскольку эти диаграммы семантически эквивалентны друг другу. Количество сценариев в прецеденте может быть достаточно велико – вследствие альтернативных путей и повторений, поэтому при построении диаграмм последовательности/коммуникации следует использовать составные шаги взаимодействия (combined fragment) для представления на диаграммах ветвлений и циклов.

В процессе построения диаграмм последовательности/коммуникации продолжается процесс определения бизнес-объектов, участвующих в реализации прецедентов – бизнес-работников (исполнителей) и бизнес-сущностей – предметов и сведений, задействуемых в реализации бизнес-прецедента.

## **1.2 Обоснование необходимости создания ПриС**

Суть обоснования необходимости создания ПриС сводится к формуле: «Показать Заказчику эффекты от внедрения разрабатываемой ПриС и доказать, что затраты на разработку ПриС «окупятся» ожидаемыми эффектами». Заметим, что окупаемость может быть не только экономической, но и технологической (повышение эффективности управления, увеличение количества решаемых задач и/или сокращение времени их решения), и социально-эргономической (улучшение условий труда работников, исходя из физических и психических особенностей человеческого организма, повышение социального статуса работников).

## **1.3 Модель требований к проектируемой системе**

### **1.3.1 Модель TO BE**

Модель TO BE представляет собой описание новой – требуемой организации бизнес-процессов. Здесь излагаются *изменения* бизнес процессов, которые необходимо выполнить при создании ПриС. Принципиально модель TO BE отличается от модели AS IS тем, что в предметной области появляется проектируемая система, и актуальные процессы предметной области реализуются с помощью этой системы.

Естественным продолжением модели TO BE, а, по сути дела, её вторым воплощением являются требования к проектируемой системе, см. следующий параграф.

### **1.3.2 Функциональные требования**

Приводятся системные диаграммы вариантов использования (прецедентов) в нотации UML, отличающиеся использованием создаваемой системы в бизнес-процессах предприятия.

Напомним, что диаграммы вариантов использования в обязательном порядке должны иметь спецификации – описания в окне Documentation – элементов диаграмм и прецедентов. Описания прецедентов представляют собой текстовые представления алгоритмов (сценариев) реализации этих прецедентов и оформляются в виде потоков поведения. Тексты потоков прикрепляются к модели редактором вложений в окне Attachment инспектора модели.

Содержимое спецификаций, естественно, включается в текст пояснительной записки. Описание потоков поведения каждого из прецедентов целесообразно помещать в отдельный структурный элемент пояснительной записки

четвёртого уровня. Пример названия такого структурного элемента: «1.3.2.1 Алгоритмическая реализации прецедента «Регистрация клиента»

### **1.3.3 Требования к информационности**

Указываются требования к составу данных, подлежащих хранению и использованию в проектируемой системе. По сути дела, эти требования во многом основываются на сведениях п. 1.1.2. Отличие обусловлено неизбежными изменениями, вызванными переходом к использованию проектируемой системы. Меняется состав сведений, осуществляется переход к машинному представлению информации (к данным), изменяется форма и место хранения информации.

Требования можно представить так же в виде таблицы, как и в п. 1.1.2. Целесообразно также построить диаграмму классов-сущностей с указанием идентификаторов атрибутов. Типы атрибутов можно не указывать вовсе, но можно и указывать, причём в свободной нотации, например: Целый, Вещественный, Integer, Float и т. п.

Особое внимание следует уделить способам представления управляющей информации – различного рода регламентирующих и нормативных документов – приказов, распоряжений, инструкций, внутрикорпоративных стандартов, положений законодательства и т. п. Требования могут содержать не только перечень управляющей информации, но и рекомендации по способам её представления и организации механизмов доступа к ней в проектируемой системе.

### **1.3.4 Архитектурные требования**

Указываются требования к базовой архитектуре системы – архитектуре верхнего уровня. Наиболее часто указывается одна из следующих альтернатив:

- настольная бессерверная – с доступом к базе данных через систему интерфейсов и драйверов в отсоединённом режиме;
- настольная клиент-серверная (двухзвенная, трёхзвенная, многозвенная);
- сетевая клиент-серверная (двухзвенная, трёхзвенная, многозвенная).

Требования излагаются в текстовой форме и в виде диаграммы развёртывания, построенной на уровне типов. В StarUML для построения этой диаграммы используются элементы (предметы) Node. Для построения диаграмм развёртывания уровня экземпляров используются предметы NodeInstance.

На диаграмме развёртывания уровня типов в узлах указываются их имена (без подчёркивания), выражающие семантику звеньев проектируемой системы. Как правило, в узлах диаграммы указывают типы артефактов, размещаемых в этих узлах. Примеры диаграммы показаны на рисунках 1.6 и 2.11.





LAN – Local Area Network (Локальная вычислительная сеть)  
 VPN – Virtual Private Network (Виртуальная частная сеть)

Рисунок 1.6 – Пример диаграммы развёртывания уровня типов с типами коммуникационной связи

Типичным для диаграммы развёртывания уровня типов является использование пометок множественности на ассоциациях. [В диаграммах уровня экземпляров вместо множественности изображается реальное количество устройств.] Также традиционно в диаграммах развёртывания уровня типов указывают тип или конкретный вид коммуникационной связи (CommunicationPath link) в виде имени ассоциации. Тип связи указывается в случае, если её конкретика не является предметом требований. Эта ситуация показана на рисунке 1.6. Если же вид связи регламентируется, то указывается конкретный протокол, например http, как это показано на втором примере, – на рисунке 2.11.

### 1.3.5 Требования к поведению

Требования к поведению проектируемой системы начинаются (а в некоторых редких случаях и заканчиваются) ссылкой на текстовые описания потоков реализации прецедентов, показанных на диаграмме вариантов использования. Пример ссылки: «Поведение проектируемой системы должно удовлетворять алгоритмам реализации, представленным в текстовой форме в пункте 1.3.2 Функциональные требования».

Далее следует представить алгоритмы реализации наиболее важных прецедентов в виде диаграмм поведения – Activity Diagram, State Machine (State Chart) Diagram, Sequence Diagram, Communication (Collaboration) Diagram. Выбор конкретного вида диаграммы осуществляется модельером на основе личных предпочтений.

В случае Sequence Diagram, Communication (Collaboration) Diagram требуется определение граничных классов. Это осуществляется, если осуществляется изложение требований к GUI.

\*\*\*\*\*

- 1) диаграмма граничных классов (классов-форм);
- 2) описание внешнего вида классов-форм (управлявшие элементы форм и их расположение на формах);

3) укрупнённая диаграмма управляющих классов с именами вариантов использования (прецедентов);

4) диаграммы последовательности, представляющие, главным образом, последовательности отображения окон (объектов классов форм) в процессе реализации вариантов использования проектируемой системы.

### 1.3.6 Функциональная спецификация

*Факультативный элемент.* Излагаются требования к программно-информационной системе в формате стандарта IEEE830. Структура описания требований к программному обеспечению согласно IEEE830 приведена в п.п. 4.7.2 учебника [1] на стр. 194 – 195. В пункте требований «2.2. Функциональность продукта» указываются ссылки на диаграммы модели TO BE, а в приложении Б ссылки на диаграммы модели AS IS.

## 2. ПРОЕКТИРОВАНИЕ СИСТЕМЫ

Следует помнить, что символическим термином, связанным с архитектурой ПриС, является термин «Исполнители». Процессы, исполнители процессов и носители данных должны быть определены в первую очередь при разработке архитектуры системы. Далее следует определить связи между исполнителями и носителями.

### 2.1 Архитектура данных и процессов

Основным средством графического представления архитектуры данных и процессов при использовании объектно-ориентированного подхода к моделированию систем являются диаграммы классов.

Диаграмма классов создается на основе диаграммы прецедентов. Классы разделяются по пакетам: 1) классы-сущности (носители информации); 2) граничные классы (формы пользовательского интерфейса); 3) управляющие классы (решатели прикладных задач, т. е. исполнители процессов).

Диаграмму классов следует представлять в трёх видах:

1) диаграмма пакетов с вложенными в пакеты классами; классы в этих диаграммах можно (хотя и необязательно) представлять кратко – одним именем;

2) диаграмма детального представления классов; классы в этих диаграммах следует (обязательно) представлять максимально подробно – помимо имён должны указываться все атрибуты и все операции с указанием спецификаторов доступа (+ | - | #), типов атрибутов, типов параметров операций и типов результатов, возвращаемых операциями; при построении этой версии диаграммы следует тщательно задокументировать все члены класса, – указать их назначение и обосновать их атрибуты и типы;

3) диаграмма отношений между классами: на этих диаграммах степень детальности представления классов может быть произвольной и *разной* для

*разных* классов, – для одних классов можно приводить только атрибуты, отменив представление методов, для других наоборот – можно указать только методы; а для третьих – можно ограничиться одними именами классов; самое главное для этой версии диаграммы – *детальное документирование* стрелок, выражающих отношения между классами.

Диаграмма классов в первом ее приближении создается на основе диаграммы прецедентов. Классы диаграммы участвуют в представлении всех основных аспектов рассмотрения ПриС: 1) операции (методы) классов представляют функциональность системы; 2) атрибуты (поля) классов представляют информационность системы; 3) объекты/роли классов, участвующие в диаграммах последовательности (коммуникации), отражают поведение ПриС; 4) классы, в своей совокупности, участвуя в модели исходного кода на диаграммах компонентов, описывают структуру системы.

Традиционной формой представления архитектуры данных является система логических моделей [4]: 1) Entity-Relationship Model, выражаемая, обычно, в виде ERD (Entity-Relationship Diagram) – модель сущность связь; 2) Key-Based Model – модель, основанная на ключах; 3) Full-Attributed Model – полная атрибутивная модель.

## **2.2 Модель поведения**

Модель поведения ПриС на этапе детального проектирования представляет собой описание алгоритмов реализации прецедентов системы.

Для разных прецедентов могут использоваться различные поведенческие (алгоритмические) диаграммы UML: 1) последовательности / коммуникации; 2) деятельности; 3) автомата. Использование всех этих диаграмм вместе не обязательно. Все алгоритмы можно представить одним видом диаграмм, но всё же следует учесть следующее:

1) наглядные и приятные для рассмотрения диаграммы последовательности / коммуникации вполне пригодны для представления «почти линейных» алгоритмов, т. е. алгоритмов с несложными ветвлениями и малым числом несложных циклов; для представления алгоритмов с нетривиальными ветвлениями и циклами следует применять диаграммы деятельности или автомата;

2) более простые для восприятия диаграммы деятельности следует применять, если среди последовательности работ алгоритма нет деятельности с теоретически бесконечной длительностью выполнения, к которым, в частности, относятся простое ожидание и зацикленное отображение рекламы или инструкций;

3) изящные и возбуждающие желание разобраться в сути алгоритма диаграммы автомата следует использовать при наличии в алгоритме состояний с теоретически бесконечной длительностью выполнения каких-либо работ, – тех же ожиданий или отображений рекламы / инструкций.

---

Белов В.В., Чистякова В.И. Проектирование программных систем. Методические указания к курсовому проектированию

При построении диаграмм деятельности следует избегать ошибки, содержащейся в некоторых методических указаниях, например, в [5] и в интернет-источниках – использования слов «Да» и «Нет» над стрелками, исходящими из узла решения. Эта практика является вольным переносом нотации блок-схем алгоритмов в UML, что совершенно недопустимо. Дело в том, что в UML узел решения – это будущий оператор switch в С-подобных или case в Паскале-подобных языках. Над исходящими стрелками следует размещать условия, при этом над одной из стрелок можно разместить английское else. Следует знать, что условия и else набираются в свойствах (Properties) стрелок, а именно в свойствах «GuardCondition», называемых в некоторых реализациях UML просто «Condition» или «Guard». Набираемые в свойствах условия появляются над стрелками автоматически и в квадратных скобках. Изложенное замечание наглядно иллюстрируется рисунком 2.1.

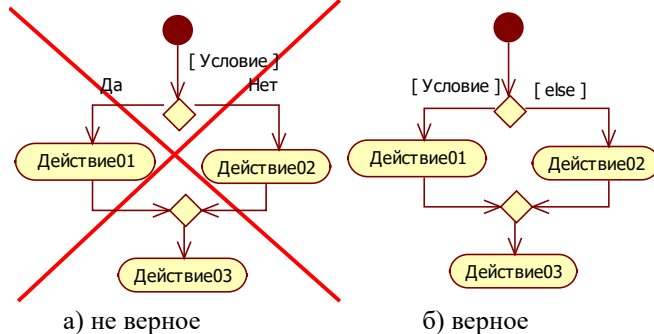


Рисунок 2.1 – Изображение ветвления в UML

Рисунок 2.1 иллюстрирует также и тот факт, что узел решения в UML является не только средством ветвления, но и средством слияния стрелок. В блок-схемах такого средства нет – стрелки соединяются простым соприкосновением.

При работе с диаграммами автомата следует чётко помнить и использовать следующий минимальный набор знаний: 1) общие элементы и различия понятий действия и деятельности; 2) структура простого состояния; 3) структура надписей над стрелками переходов.

Действие (Action) – это процесс выполнения некоторой несложной кратковременной и, главное, – непрерываемой операции.

Деятельность (Activity) – это процесс выполнения не кратковременной и, главное, – прерываемой операции.

Во время выполнения действия объект не может покинуть текущее состояние. Во время выполнения деятельности возможно прерывание текущего состояния под действием триггерного события и последующий переход объекта в новое состояние.

Структура простого состояния, показана на рисунке 2.2.

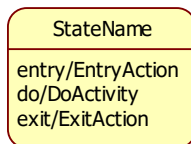


Рисунок 2.2 – Структура простого состояния

## 2.3 Модель реализации

### 2.3.1 Предварительные замечания

#### 2.3.1.1. Структура модели реализации

Модель реализации состоит из четырёх подмоделей:

- 1) модель исходного кода;
- 2) модель исполняемого кода;
- 3) модель компонентов времени исполнения;
- 4) модель размещения артефактов по физическим устройствам.

Первые три модели образуют так называемое компонентное представление системы – Component View в терминологии подхода Rational.

Все модели реализации создаются с помощью двух видов диаграмм – компонентов и развёртывания. При этом вы должны помнить следующий минимум сведений.

В моделях исходного и исполняемого кода с помощью диаграммы компонентов отображаются два шага перехода от логических элементов приложения к артефактам:

- 1) от классов (программных модулей) к файлам исходных кодов;
- 2) от файлов исходных кодов к загружаемым файлам.

Заметим, что файлы, содержащие программный код и данные, – это наиболее распространённый вид артефактов – физических субстанций, создаваемых в процессе разработки системы.

Для начинающего программиста понятия «класс» и «файл, содержащий операторы этого класса», не разделимы и практически не различимы. [Начинающий программист попросту не задумывается над такими различиями]. Но программист-профессионал должен чётко осознавать два факта: 1) существует класс как замысел – проектное решение, отображаемое прямоугольником на диаграмме классов, и 2) существует программный (исходный) код этого класса, записанный на конкретном языке программирования и размещённый в конкретном файле программы.

### 2.3.1.2. Понятия компонента и артефакта

В модели реализации интенсивно используются понятия компонент и артефакт. В практике использования UML, исходя из идей основателей объектно-ориентированного подхода к моделированию систем и первых версий языка UML, сложилась традиция рассматривать компоненты как элементы физической модели проектируемой системы, или, как ещё говорят, элементы системы времени выполнения (run-time). Поскольку слово «компонент», по-русски означает просто «часть» чего-либо, то часто вели речь о логических и физических компонентах. Под логическими компонентами понимались части системы времени её проектирования (design time) – программы и данные как замысел, а под физическими – файлы и устройства хранения программ и данных системы, т. е. элементы run-time. Для физических (run-time) частей системы определён специальный термин – артефакты.

Иногда в отечественной литературе, например, в [2] излагается мнение о том, что UML 2 якобы локализовал понятие компонента логическим представлением системы, а физическое начало компонента закреплено за термином «артефакт». Однако этому мнению противоречит документация по UML [3]. Рисунок 2.4, заимствованный из [3] ясно свидетельствует: компонент – это понятие, символизирующее часть системы, включая все её представления – и логическое (включающее предоставляемые и потребляемые интерфейсы, классы, реализующие и потребляющие эти интерфейсы), и физическое (включающее артефакты – файлы, содержащие программные коды классов).

Артефактом принято называть любую физически существующая сущность, создаваемую в процессе разработки системы.

Итак, запоминаем следующее.

1. Логическое представление компонента – это смысл, семантика программ и данных проектируемой системы, а также логическая архитектура системы – организация программ и данных – конструкции (объединения программ и данных). Вся совокупность описаний программ, данных и их конструкций называется логической, или design time моделью проектируемой системы.

2. Физическое представление компонента – это совокупность артефактов, т. е. файлов – физически существующих сущностей, хранящих (воплощающих, реализующих) элементы логического представления компонента.

Совокупность описаний артефактов системы образует физическую, или run-time модель проектируемой системы.

Своеобразной эмблемой логической модели является термин «класс»: классы – наиболее часто используемые элементы, являющиеся фундаментальным понятием объектно-ориентированного подхода в программировании и моделировании систем.

Своеобразной эмблемой физической модели является термин «файл»: файлы – первичные хранилища программ и данных – образуют наиболее многочисленный отряд представителей артефактов.

### 2.3.1.3. Компонент как изобразительное средство UML

В модели проектируемой системы компонент представляет *относительно самостоятельную заменяемую* часть проектируемой системы, имеющую следующие характеристики:

1) предоставляемые интерфейсы (provided interfaces) – перечень элементов собственной функциональности, предоставляемой другим компонентам;

2) необходимые интерфейсы (required interfaces) – список элементов сторонней функциональности, необходимой для реализации собственной функциональности, предоставляемой другим компонентам;

3) реализации (realizations) – перечень классов, реализующих собственную функциональность компонента (предоставляемые интерфейсы) и потребляющих при этом стороннюю функциональность (необходимые интерфейсы); естественно, реализующие классы связаны с компонентом отношением реализации, поскольку компонент символизирует и экспонирует интерфейсы, реализуемые классами

4) артефакты (artifacts) – перечень файлов, «проявляющих» (манифестирующих) классы, а точнее – содержащих коды классов, реализующих предоставляемые интерфейсы, на физическом уровне, т. е. файлы с исходным программным кодом; сюда же следует отнести и прочие файлы, содержащие программный код классов, – объектные и исполняемые.

На диаграммах обычно используется краткая (традиционная) форма представления интерфейса, показанная на рисунке 2.3. Для большей информативности по требованию руководителя проекта на дополнительных диаграммах могут изображаться компоненты в полной форме представления по принципу «белого ящика», показанной на рисунке 2.4, заимствованном из документации по UML. В общем случае компонент может быть составным (включающим) – может содержать в себе другие компоненты, информационные и графические артефакты. При этом появляется дополнительная характеристика компонента, отображаемая секцией *package elements* (см. рисунки 2.6, 2.7, 2.10), в которой отображаются эти включаемые (пакетируемые) элементы.

Иногда применяются явные представления отношений между компонентом и его частями, например, между компонентом и реализующими его классами. Соответствующий пример показан на рисунке 2.5. Строго говоря, понятие «классы, реализующие компонент» является условным, упрощённым, образным, поскольку реально реализуются *интерфейсы* компонента (provided interfaces). В приводимом примере классы OrderHeader и LineItem реализуют интерфейсы ItemAllocation и Tracking компонента Order. Если сохранять строгость суждений, то классы и компоненты (именно компоненты) свя-

заны отношением не реализации, а композиции, – поскольку компонент – это композиция: 1) интерфейсов, 2) классов, реализующих интерфейсы, 3) артефактов, содержащих программные коды классов и ассоциированные данные.

Возникает вопрос: если компонент – это композиция, то что символизирует фигура компонента на диаграмме? Наиболее правилен ответ: всё зависит от контекста, но чаще всего контексты таковы, что фигура компонента символизирует артефакт этого компонента.

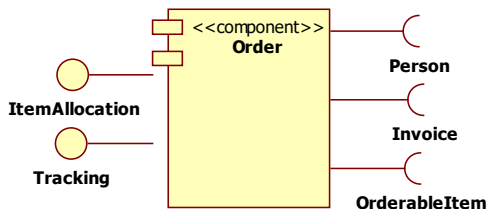


Рисунок 2.3 – Традиционное представление компонента с интерфейсами

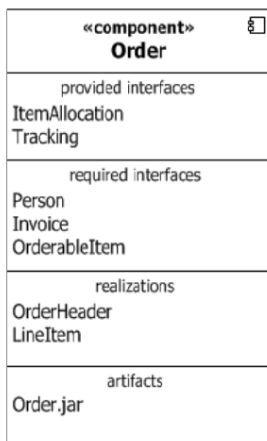


Рисунок 2.4 – Опциональное полное представление компонента по принципу «белого ящика». Пример из документации по UML

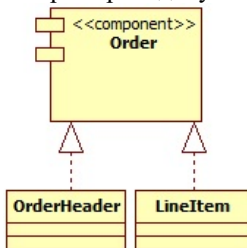




Рисунок 2.5 – Отношение реализации между компонентом и его классами

#### **2.3.1.4. Некоторые особенности представления компонентов и артефактов в StarUML**

К сожалению, изобразительные средства систем моделирования в отношении компонентов и артефактов далеки от спецификации [3]. Это обусловлено нечёткостью синтаксиса и семантики изобразительных средств, предлагаемых стандартом для компонентов и их частей. В частности имеют место следующие особенности StarUML.

1. StarUML не позволяет назначать надлежащий стереотип артефактам. Артефакты автоматически снабжаются стереотипом «artifact», который не изменяем. Назначить собственный стереотип можно, но он отображается только в навигаторе модели (в правом верхнем углу в области браузера на закладке Model Explorer), в фигуре же артефакта ничего не изменяется – отображается только «artifact». Эта особенность имеет принципиальное значение – она не позволяет строить диаграммы модели реализации с наглядным представлением физических сущностей системы. По этой причине рекомендуем для представления всех артефактов в StarUML использовать изобразительное средство Component, а не Artifact. При этом собственно компоненты следует изображать с потребляемыми и поставляемыми интерфейсами, как показано рисунке 2.3, а простые артефакты (файлы), естественно, без интерфейсов, – поскольку они оных не имеют, – и со стереотипами, соответствующими типам этих файлов, например, «source», «object», «executable», «library», «table», «document».

2. StarUML не обеспечивает представление секций для отображения частей компонента – realization, artifacts, package elements, internal structure. Вместо них StarUML предлагает использовать не упоминаемую в стандарте секцию residents, позволяющую компактно (без украшений) поместить перечисление частей, входящих в компонент. Кроме этого, в StarUML можно симитировать любую секцию компонента, хотя и не полноценно, с помощью инструмента Part (часть). Разместив Part на площади компонента, можно присвоить ей любое имя, например, package elements и внутри разместить любые части компонента.

3. StarUML не позволяет сгруппировать наложенные друг на друга геометрические фигуры, что существенно затрудняет применение имитаций стандартных конструкций UML с помощью доступных инструментальных средств StarUML.

Заметим, что указывать секцию с перечнем классов, реализующих и использующих интерфейсы компонента, необязательно, поскольку систематическое проектирование предполагает построение модели исходного кода, в

которой отображаются классы и включающие их программные файлы (артефакты).

### 2.3.1.5. Компонентное представление (Component View) проектируемой системы

По признаку «логика или физика», т. е. «идея или материя», компонентное представление системы является и логическим, и физическим, поскольку манипулирует и логическими сущностями – классами, компонентами – и физическими сущностями – артефактами. По этой причине в подходе к моделированию Rational оно выделено в самостоятельное представление Component View, а не рассмотрено как часть логического представления Logical View.

Компонентное представление системы в UML является отражением концепции разработки и структурирования системы на основе компонентов, в которой компоненты моделируются на протяжении всего жизненного цикла разработки и последовательно уточняются при переходе от проектирования к развертыванию и эксплуатации.

В компонентном представлении система рассматривается как *совокупность взаимосвязанных компонентов* – самостоятельных и заменяемых частей. Самостоятельность, заменяемость и наличие интерфейсов – важнейшие, концептуальные особенности компонента. Самостоятельность (autonomous) компонента выражается в высокой значимости и инкапсулированности его функциональности, в чёткости определения этой функциональности, в чёткости определения взаимосвязи компонента с другими компонентами. Заменяемость (replaceable) означает возможность «бесшовной» замены одного компонента другим компонентом с тем же набором предоставляемых интерфейсов (совпадения остальных элементов спецификации компонента не требуется). Наличие интерфейсов означает наличие чётких описаний некоторых функций компонента и неизменяемых правил использования этих функций.

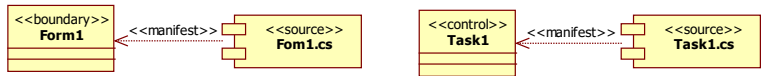
### 2.3.2 Модель исходного кода

Основная идея модели исходного кода – показать файлы, в которых размещается текст программы (классы, сценарии, плагины), т. е. указать модули приложения, воплощающего проектируемую систему. Образно можно говорить, что основная идея модели исходного кода – показать переход от логического представления программы к физическому представлению: от смысла операторов, классов, скриптов, плагинов к местам физического хранения текстовых записей этих операторов, классов, скриптов, плагинов.

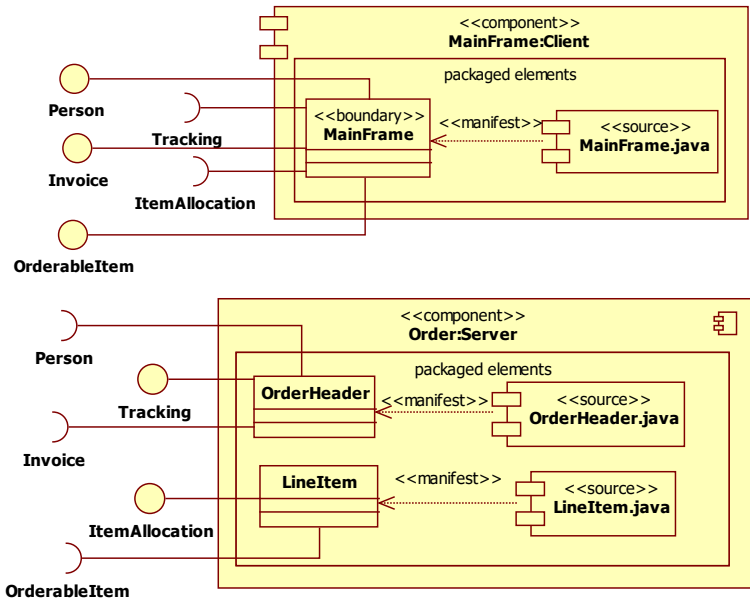
Заметим, что JavaScript под классами принято понимать функции-конструкторы – функции, вызываемой при создании экземпляра, т. е. при выполнении оператора new, со ссылкой на прототип – объект, содержащий свойства (данные) и методы (функции) класса.

В модели исходного кода на диаграмме компонентов наиболее значимые логические элементы – классы и/или сценарии (скрипты) – соединяются стрелкой с артефактами. Стрелки направляются от артефакта к классу/сценарию и снабжается стереотипом (надписью) «manifest» (от позднелатинского manifestum – «призыв» и итальянского manifesto – «объявление»). Таким образом, модель исходного кода показывает – какие классы/сценарии в каких файлах содержатся.

Пример простейшей модели исходного кода для приложения с одним интерфейсным окном и одним модулем реализации бизнес-логики (прикладной задачи) показан рисунке 2.6 а).



а) модель исходного кода для несложного приложения на языке C#



б) модель двухкомпонентного исходного кода для приложения на языке Java

Рисунок 2.6 – Модели исходного кода для несложных приложений на языках C# и Java

Модель, представленная на рисунке 2.6 а), несёт следующую информацию, направленную от проектировщика программисту:

Белов В.В., Чистякова В.И. Проектирование программных систем. Методические указания к курсовому проектированию

– проектируемая система должна быть реализована на языке C# в виде двух классов:

1) класс формы, реализующей пользовательский интерфейс, имя класса в модели Form1;

2) класс реализации варианта использования системы (решения некоторой прикладной задачи), имя класса в модели Task1;

– каждый из указанных классов должен быть запрограммирован в собственном файле, т. е. должны быть созданы файлы с именами Form1.cs и Task1.cs для классов соответственно Form1 и Task1.

Модель, представленная на рисунке 2.6 б), основана на концепции компонентного проектирования, которая в традиционной практике разработки систем активно используется только при моделировании компонентов времени исполнения и развёртывания артефактов по устройствам. Мы же настоятельно рекомендуем использовать концепцию компонентного проектирования во всех моделях реализации (физической архитектуры), включая и модель исходного кода.

Модель, представленная на рисунке 2.6 б), более информативна по сравнению с моделью рисунка 2.6 а). Она сообщает следующее:

– проектируемая система должна быть реализована на языке Java в виде в виде двух компонентов:

1) компонент клиентской части, имя компонент в модели MainFrame:Client;

2) компонент серверной части – реализации варианта использования системы (решения некоторой прикладной задачи), имя класса в модели Order:Server;

– функциональность компонента MainFrame:Client должна быть реализована в виде класса с именем MainFrame;

– исходный программный код класса MainFrame должен быть размещён в файле с именем MainFrame.java;

– функциональность компонента Order:Server должна быть реализована в виде двух классов с именами OrderHeader и LineItem;

– исходный программный код класса OrderHeader должен быть размещён в файле с именем OrderHeader.java;

– исходный программный код класса LineItem должен быть размещён в файле с именем LineItem.java;

– класс MainFrame должен реализовывать интерфейсы: Person, Invoice и OrderableItem;


– класс MainFrame, для реализации интерфейсов Person, Invoice и OrderableItem должен использовать интерфейсы Tracking и ItemAllocation, реализуемые классами OrderHeader и LineItem компонента Order:Server;

- класс `OrderHeader`, для реализации интерфейса `Tracking` должен использовать интерфейсы `Person` и `Invoice`, реализуемые классом `MainFrame` компонента `MainFrame:Client`;
- класс `LineItem`, для реализации интерфейса `ItemAllocation` должен использовать интерфейс `OrderableItem`, реализуемый классом `MainFrame` компонента `MainFrame:Client`.

Обратите внимание на взаимосоответствие по реализации и использованию интерфейсов классов `MainFrame:Client` и `Order:Server`.

Заметим, что на рисунке 2.6 б) использованы два равносильных варианта обозначения компонента:

1) прямоугольник компонента с двумя меньшими прямоугольниками, выступающими из его левой стороны,

2) прямоугольник компонента с пиктограммой .

Следует не забыть об изложении всего, представленного на диаграмме, в текстовых дополнениях, помещаемых в секцию `Documentation` и в текстовый файл, на который размещается ссылка в `Attachment`.

### 2.3.3 Модель исполняемого кода

В модели исполняемого кода на диаграмме компонентов отображаются только артефакты – файлы с исходными, объектными и загружаемыми кодами.

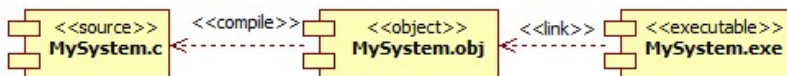
На этой диаграмме артефакты соединяются стрелкой зависимости, направленной от исходного артефакта к порождаемому артефакту, со стереотипом «`compile`», «`link`», «`build`», «`make`», «Собрать» или иным – в зависимости от целевой системы программирования.

Стереотип «`compile`» символизирует тот факт, что порождаемый файл объектного кода, например, с именем `BusinessLogic.obj`, формируется в результате компиляции исходного файла, например, с именем `BusinessLogic.c`.

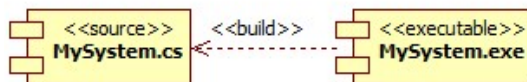
Стереотипом «`link`» помечаются стрелки зависимости, направленные от файла готового приложения (или динамически подключаемой библиотеки) к файлам с объектным кодом. Они символизируют тот факт, что порождаемый файл, например, с именем `BusinessLogic.dll`, формируется в результате редактирования связей в коде объектного файла, например, с именем `BusinessLogic.obj`.

Представить модель исполняемого кода в виде диаграммы компонентов, которая может иметь один из следующих видов.

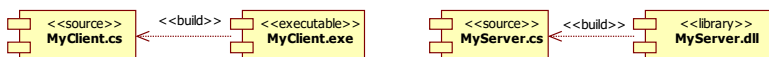
А. Простейший вариант – система реализована в виде одного или двух файлов исходного кода. Примеры моделей исполняемого кода для трёх языков программирования показаны на рисунке 2.7.



а) язык программирования С; однофайловый вариант приложения



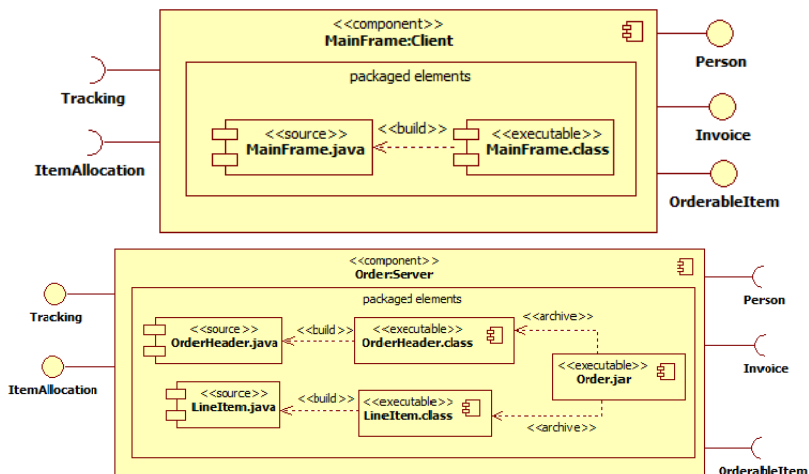
б) язык программирования С#; однофайловый вариант приложения



в) язык программирования С#; двухфайловый вариант:  
разделение приложения на клиентскую и серверную часть



г) язык программирования Java; однофайловый вариант приложения



д) язык программирования Java;  
двухкомпонентный вариант приложения

Рисунок 2.7 – Модель исполняемого кода в простейшем случае

Если использование Java-программы предполагается в операционной системе Windows, то очень удобно для class-файла применить exe-оболочку, например, оболочку «JanelWindows.exe». Эта оболочка переименовывается в

имя вашей программы, и далее именно она запускается как обычный exe-файл, обеспечивая скрытый запуск JVM и последующее выполнение вашего class-файла под личиной Windows-приложения. Модель исполняемого кода при этом можно представить, показанном на рисунке 2.8.

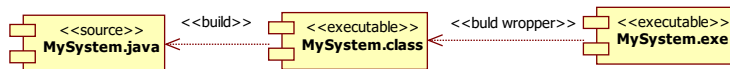


Рисунок 2.8 – Модель исполняемого кода  
при использовании exe-оболочки для class-файла

**Б. Вариант с пакетами проектов.** При использовании систем программирования, создающих проекты, диаграмма модели исполняемого кода содержит пакеты проектов, соединенные стрелками зависимости с исполняемыми файлами. Исполняемые файлы очень часто имеют расширение имени: exe, dll, class, jar, war.

При этом следует выполнять следующие правила:

- 1) стрелки должны быть направлены от исполняемых файлов к пакетам проектов;
- 2) стрелки должны быть снабжены стереотипом создания исполняемого кода – «build» или «make», или «Собрать», или аналогичным, – с тем же смыслом;
- 3) внутри пакетов проектов размещаются файлы исходного кода;
- 4) внутри пакетов размещать следует файлы только тех исходных кодов, к которым «прикладывались руки» программиста; файлы дизайна, конструирования, конфигурации и прочие, генерируемые системой программирования автоматически, показывать не следует.

Пример модели исполняемого кода для случая, когда система предназначена для решения трёх задач, представлен на рисунке 2.9. В примере предполагается: 1) файлы исходных кодов Task01.cs, Task02.cs и Task03.cs реализуют основную функциональность (основные задачи) проектируемой системы; это могут быть три самостоятельных варианта использования системы или три подзадачи одного сложного варианта использования; 2) файлы исходных кодов Form01.cs, Form02.cs и Form03.cs реализуют пользовательский интерфейс системы во время решения задач.

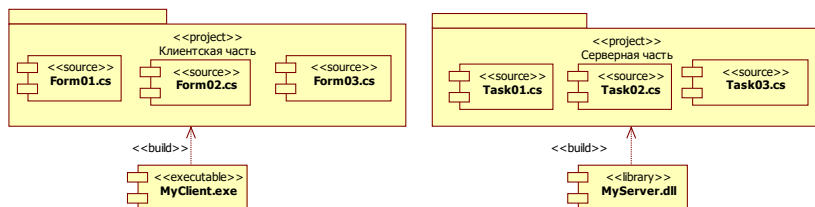


Рисунок 2.9 – Модель исполняемого кода при использовании проектов систем программирования

**В. Вариант с интерпретируемыми языками.** Если в проектируемой системе предполагается использование Web-сервера, то проект не обходится без использования интерпретируемых языков программирования, в частности, скриптовых или интерпретируемых вариантов мультипарадигменных языков (Basic, Python, Perl, Mathematica, Matlab, Maple, Mathcad, Lua, Javascript, AngelScript, PHP, Squirrel, Tcl (Tool command language), REBOL, Ruby, AutoIt, Pawn). Диаграмма компонентов модели исполняемого кода при этом состоит из несвязанных прямоугольников, символизирующих исходные и одновременно исполняемые (интерпретируемые) файлы системы.

При составлении диаграммы компонентов не забывайте выполнять принцип: в диаграмме всё должно быть ясно, понятно и однозначно. Поэтому не забывайте снабжать фигуры диаграммы соответствующими стереотипами, выражающими тип (смысл) этих фигур. Естественно, если для каких-то типов компонентов имеются специальные пиктограммы (например, для динамически подключаемых библиотек и таблиц базы данных), то их можно и даже желательно использовать. Но выбор у вас сохраняется: вы либо рисуете прямоугольник со стереотипом, либо изображаете соответствующую пиктограмму (см. рисунки 5.23, 5.24 в учебнике [1]). Всё или наиболее важное, представленное на диаграмме, следует изложить в виде текстовых дополнений в секциях Documentation и Attachment.

### 2.3.4 Модель поставляемых артефактов

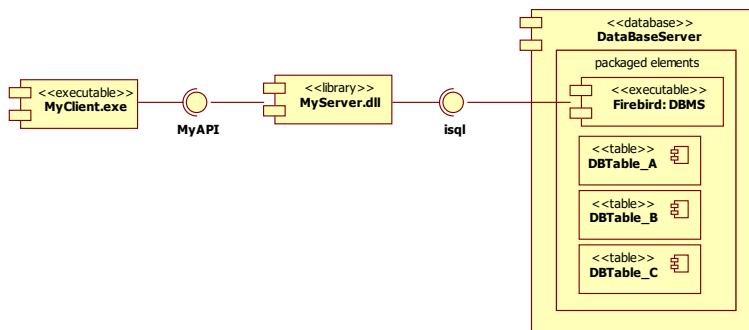
Модель поставляемых артефактов – это альтернативное представление архитектуры проектируемой системы и достаточно простая конструкция. Она представляется диаграммой компонентов, на которой отображаются артефакты исполняемого кода и данных, отторгаемые от разработчика, и передаваемые потребителю – заказчику или покупателю. Артефакты соединяются линиями, символизирующими их соединения, по которым передаются вызовы функций / процедур и данные. Эти соединения реализуются посредством интерфейсов, изображаемым в виде леденца на палочке (lollipop).



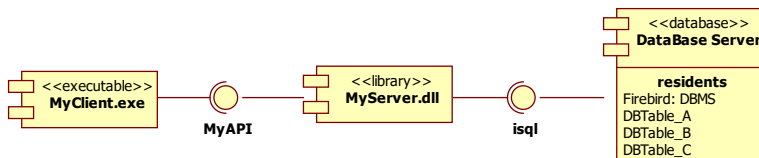
В модели поставляемых артефактов, естественно, должны присутствовать все исполняемые файлы, представленные в модели исполняемого кода. Помимо них могут присутствовать:

- 1) информационные файлы, содержащие таблицы баз данных, таблицы Excel, CSV-файлы (Comma-Separated Values, текстовые файлы значений, разделённых запятыми) и прочее;
- 2) стороннее программное обеспечение (например, web-серверы, интерпретаторы, СУБД и прочее) и/или сторонние данные, необходимые для построения законченной полнофункциональной системы.

Пример модели поставляемых артефактов показан на рисунке 2.10.



а) с использованием наглядной формы представления составного артефакта



а) с использованием компактной формы представления составного артефакта

Рисунок 2.10 – Модель поставляемых артефактов: альтернативные формы представления

Модель на рисунке 2.10, информирует читателя о том, что потребителю проектируемой системы передаются следующие артефакты:

- 1) файл **MyClient.exe**, – содержащий программу клиентской части системы;
- 2) файл **MyServer.dll**, – содержащий программу серверной части системы;

3) файлы, функционирующие в рамках артефакта, называемого базой данных программно-информационной системы, символически обозначенной идентификатором DataBaseIS, включающей:

а) три файла таблиц базы данных с именами DBTable\_A, DBTable\_B и DBTable\_C;

б) некоторое количество (в рамках данной модели не определённое как не имеющее принципиального значения) файлов, образующих артефакт Firebird – систему управления базами данных (СУБД, англ. Database Management System, или DBMS).

Все поставляемые артефакты, за исключением Firebird, являются результатом работы исполнителя разработки системы. Указанная модель соответствует случаю, когда стороннее программное обеспечение (СУБД) согласно договору поставляется потребителю исполнителем.

Помимо поставляемых артефактов модель, представленная на рисунке 2.10, показывает, что клиентская часть системы MyClient.exe получает «услуги» от серверной части MyServer.dll посредством интерфейса MyAPI, реализуемого серверной частью. В свою очередь, серверная часть запрашивает и получает данные от базы данных программно-информационной системы посредством интерфейса isql, реализуемого системой управления базами данных Firebird.

При использовании StarUML для построения диаграммы, аналогичной показанной на рисунке 2.10, прямоугольник клиента – потребителя интерфейса – соединяется с кружочком, изображающим интерфейс, стрелкой зависимости (Dependency), а сервер – поставщик интерфейса – соединяется с кружочком интерфейса стрелкой реализации (Realization).

### **2.3.5 Модель размещения артефактов**

Модель размещения артефактов отображает распределение артефактов, поставляемых потребителю по физическим устройствам. Эта модель представляется диаграммой развёртывания (Deployment Diagram), на которой в виде узлов отображаются устройства и среды исполнения.

В определённой степени модель размещения является альтернативным средством представления поставляемых потребителю артефактов. Различия между моделями поставляемых артефактов (МПА) и размещения этих артефактов (МРА) таковы:

1) МПА носит характер перечисления, главное назначение МПА – перечислить физические части проектируемой системы, которые будут поставлены потребителю;

2) главное назначение МРА – указать места (узлы) размещения физических частей проектируемой системы, поставляемых потребителю.

Узлы изображаются в виде прямоугольных параллелепипедов. Узлы устройств снабжаются стереотипом «Device». Узлы сред исполнения – стереотипом «Execution Environment».

Среда исполнения представляет собой совокупность обрабатывающей программы или системы и обрабатываемых данных.

Примеры среды исполнения:

- 1) совокупность СУБД и файлов базы данных;
- 2) совокупность сервера приложения и ассоциированных файлов.

Обратим внимание на перегруженность часто применяемой терминологии. Термин «Сервер базы данных» может означать следующее:

- 1) тип программного обеспечения, предназначенного для работы с данными, называемый СУБД;
- 2) конкретную СУБД, например, Oracle, SQL SERVER (Microsoft), SQL BASE SERVER, Oracle SERVER (Oracle Corporation), IBM DB2, Informix, Ingres, Interbase, MySQL, PostgreSQL, FirebirdSQL;
- 3) аппаратное средство в совокупности с конкретной СУБД и дополнительным программным обеспечением, предназначенное для работы с базами данных.

На рисунке 2.11 представлен пример модели размещения артефактов системы, соответствующий модели, показанной на рисунке 2.10.

Указанная модель отражает следующее проектное архитектурное решение:

- 1) проектируемая программно-информационная система должна иметь трёхмашинное воплощение в виде устройств (компьютеров) трёх типов – Client, Application Server и Database Server;
- 2) в каждом компьютере размещается программное обеспечение соответствующего типа; тип (классификатор) программного обеспечения DBMS явно указан только для устройства типа Database Server; названия типов программного обеспечения, размещаемого в других устройствах, предполагаются совпадающими с названиями типов устройств;
- 3) машины располагаются с сети и взаимодействуют друг с другом по протоколу http – позволяющему получать различные ресурсы, например HTML-документы, лежащему в основе обмена данными в Интернете, являющемуся протоколом клиент-серверного взаимодействия.

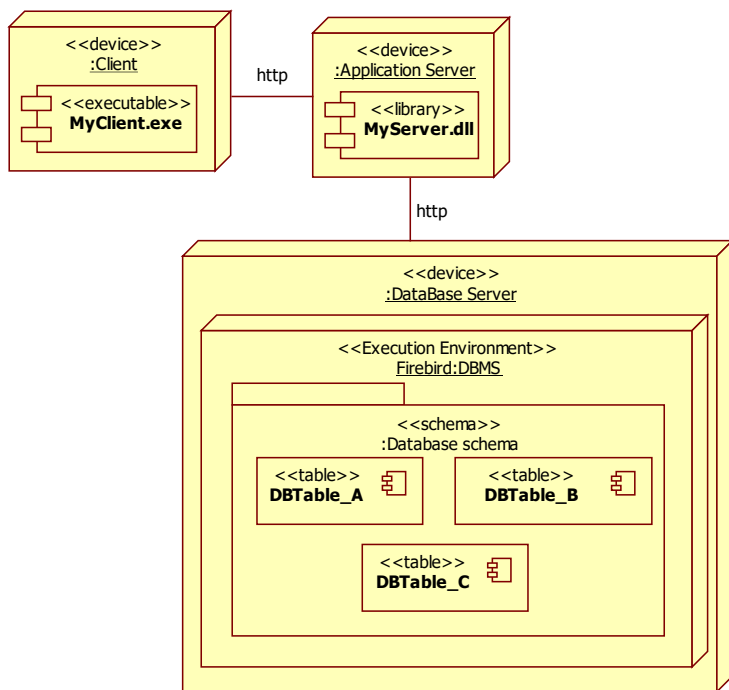


Рисунок 2.11 – Модель размещения артефактов проектируемой системы

С диаграммами развёртывания, обычно, проблем не бывает – на них отображаются устройства, чаще всего компьютеры и артефакты, размещённые в них. Но всё же почитать учебник и посмотреть следующие сайты следует.

[http://www.info-system.ru/designing/methodology/uml/theory/deployment\\_diagram\\_theory.html](http://www.info-system.ru/designing/methodology/uml/theory/deployment_diagram_theory.html)  
<http://www.uchi-it.ru/7/3/13.html>  
<http://www.intuit.ru/departement/pl/umlbasics/13/>

### 2.3.6 Модели управления

Излагаются используемые модели связей управления между частями системы – указываются методы классов, вызываемые процедурно (как подпрограммы или как сопрограммы), как обработчики событий, как самостоятельные процессы; приводятся схемы процедурных вызовов, схемы вызовов обработчиков и самостоятельных процессов.

### 2.3.7 Интерфейс пользователя

Описываются основные идеи структуры интерфейса пользователя. Изображается дерево форм и излагается алгоритм переключения между формами. Приводятся виды и описываются элементы наиболее важных интерфейсных окон.

## 3. РЕАЛИЗАЦИЯ ПРИС

### 3.1 Реализация бизнес-логики

Окончательно формируется совокупность классов ПриС, реализующих бизнес-логику и пользовательский интерфейс. Составляются коды методов. Возможно, определяются дополнительные поля классов, выявляются и реализуются дополнительные классы, потребность в которых возникла в процессе реализации ранее определённых классов. Формируется окончательная диаграмма классов ПриС.

Создаётся простейший эмулятор базы данных (БД), – хранилище в виде файлов, позволяющее хранить предметные данные, необходимые для формирования объектов и вызова методов бизнес-логики, а также данные, получаемые в результате работы этих методов.

### 3.2 Реализация пользовательского интерфейса

Приводятся виды и описания форм приложения. Для каждой формы указываются значения свойств её компонентов в виде таблицы, аналогичной следующей:

Компонент	Назначение	Свойство	Значение
Form1	Главная форма	Name	fmMain
		Text	Телефонная книга
		Caption	Шрифт – MS Sans Serif; начертание – обычный; размер – 14; цвет – черный
		Hint	Окно телефонной книги
		ShowHint	True

(приводятся только те свойства, значения которых изменены, а не установлены по умолчанию). Описываются ассоциированные с формами обработчики событий (с приведением кодов обработчиков).

### 3.3 Реализация базы данных

Диаграмма классов трансформируется в модель базы данных с помощью утилиты ERwin Translation Wizard, вызываемой из среды Rational Rose. Можно осуществлять и ручное отображение объектной модели в реляционную модель (Mapping from the Class Model to the Relational Model).

Приводятся таблицы с указанием типов полей. Описывается состав файлов физической базы данных

### **3.4 Реализация средств доступа к базе данных**

Создаются классы, реализующие взаимодействие с БД, обеспечивающих получение из БД данных, необходимых для формирования требуемых состояний объектов бизнес-логики и реализующих типовые (CRUD) операции с базой данных – ввод, чтение (поиск, фильтрацию), изменение, удаление.

Приводятся и описываются фрагменты программного кода, содержащие:

- описание необходимых объектов, классов, интерфейсов;
- подключение к источнику данных;
- выполнение команд;
- получение данных;
- при необходимости – прочие действия с данными.

## **4. ТЕСТИРОВАНИЕ ПРИС**

Создаваемая система должна содержать исходные данные для решения тестовых задач. Желательно предусмотреть в программе специальный пункт меню «Проверка системы» с подпунктами – контрольными задачами и подпунктами «Описание контрольных задач», «Решение контрольных задач».

В разделе, посвященном описанию процесса тестирования разработанной системы, следует предусмотреть доказательство того, что тестовые задачи обеспечивают выполнение известных из теории правил тестирования программ. Наиболее распространенная ошибка: описываются теоретические сведения по тестированию, а затем приводятся результаты решения проверочных задач без доказательства полноты тестов.

Следует обратить внимание на наглядность представления результатов тестирования и простоту их интерпретации.

## **5. СОСТАВ ПРОЕКТА**

Сведения раздела «Состав проекта» должны быть достаточными для полноценной сборки проекта для дальнейшего развития или устранения ошибок. В частности, в этом подразделе необходимо описать состав основных, системных и вспомогательных файлов проекта. Рекомендуется указанные описания оформлять в виде пунктов подраздела с заголовками:

- «Основные файлы проекта»;
- «Системные файлы проекта»;
- «Вспомогательные файлы проекта».

В этих пунктах для каждого файла указывается:

- имя;
- функциональное назначение;
- местоположение (необходимое или рекомендуемое расположение файла на внешнем носителе информации).

## **6. КОМПИЛЯЦИЯ И СБОРКА СИСТЕМЫ**

### **6.1 Правила создания системы (приложения)**

В разделе «Компиляция и сборка системы» следует описать порядок компиляции и сборки системы для отладки и для инсталляции на вычислительной системе пользователя.

Обязательно следует указывать условия компиляции – требования к ЭВМ и системному программному обеспечению.

### **6.2 Состав приложения**

Содержание этого пункта полностью аналогично содержанию раздела «СОСТАВ ПРОЕКТА», но файловый состав излагается для созданного приложения, – того, что подлежит дистрибуции (распространению) среди потребителей.

## **7. ДОКУМЕНТАЦИЯ**

Раздел должен включать «Руководство пользователя», содержащее следующие подразделы.

### **7.1 Назначение системы и выполняемые функции**

### **7.2 Условия применения системы**

### **7.3 Правила инсталляции системы**

### **7.4 Сценарий взаимодействия с пользователем**

Подраздел «Назначение системы и выполняемые функции» должен содержать общую характеристику ИИС и перечень обслуживаемых запросов.

Подраздел «Условия применения системы» должен содержать требования к техническим характеристикам ЭВМ (типу процессора, объему оперативной и внешней памяти, периферийному оборудованию), операционной системе и вспомогательному программному обеспечению.

Правила инсталляции должны завершаться описанием файлового состава установленной системы, аналогично описаниям файловых составов проекта (глава 5) и приложения (параграф 6.1). Необходимо указать перспективы изменения файлового состава системы. Все эти сведения должны способствовать облегчению процесса восстановления системы в непредвиденных ситуациях.

Сценарий взаимодействия с пользователем – это описание процесса решения наиболее типичных задач с помощью ПриС. Оно должно включать следующие описания:

- структура меню программы;
- общая схема использования программы;

- описание последовательности действий пользователя во время решения типовых задач и при выполнении базовых операций (таких как подготовка данных, просмотр результатов, запуск процедур обработки запросов).

## **ЗАКЛЮЧЕНИЕ**

В **Заключении** следует привести краткий обзор выполненных работ в соответствии с элементами **Содержания** (оглавления) пояснительной записки с обязательным учетом специфики предметной области и конкретики принятых проектных решений.

## **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

Список использованных источников следует оформлять в соответствии со стандартом ГОСТ Р 7.05–2008 «Библиографическая ссылка». Рекомендуем ознакомиться с образцами в файлах «4 Пример оформления списка литературы.doc» и «4. Пример оформления списка литературы Конф Сборники.doc».

## **ПРИЛОЖЕНИЯ**

В одном из приложений обязательно должны приводиться распечатки наиболее важных фрагментов программы. Объем распечаток не должен превышать 25 страниц. Приводимые тексты программ должны быть тщательно прокомментированы.

## **IV. ПРАВИЛА СДАЧИ ПРОЕКТА РУКОВОДИТЕЛЮ ПРОЕКТА**

1. Предъявляются следующие материалы:
  - 1) пояснительная записка на твердом носителе;
  - 2) пояснительная записка в электронном виде;
  - 3) исходные коды проекта, включая файлы базы данных и справочной системы, в электронном виде.
2. Проверяется соответствие пояснительной записки предъявляемым требованиям.
3. Как правило, осуществляются:
  - 1) компиляция и сборка проекта;
  - 2) решение тестовых задач.

## **V. ИНСТРУМЕНТАЛЬНЫЕ ТРЕБОВАНИЯ**

1. ЭВМ – ПЭВМ, совместимая с IBM PC.
2. Инструментальная среда – не регламентируется. Рекомендуется MS Windows, MS Visual Studio, ADO.Net, Java, любой сервер базы данных.
3. Допустимо использовать для хранения данных коллекции, сохраняемые в потоках, эмулирующие базы данных.
4. Язык программирования – не регламентируется.



## БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Белов В.В., Чистякова В.И. Проектирование информационных систем: учебник – М.: КУРС, 2018. – 400 с. ISBN 978-5-906923-53-0 (КУРС) (45 экз. в БФ РГРТУ).
2. Иванов, Денис Юрьевич. Унифицированный язык моделирования UML [Электронный ресурс]: учебное пособие для вузов по направлению подготовки "Системный анализ и управление" / Д.Ю. Иванов, Ф.А. Новиков; Санкт-Петербургский государственный политехн. ун-т. – Электрон. текстовые дан. (1 файл: 1,83 Мб). – Санкт-Петербург, 2011. – Загл. с титул. экрана. – Электронная версия печатной публикации. – Свободный доступ из сети Интернет (чтение, печать, копирование). – Текстовый документ. – Adobe Acrobat Reader 7.0. URL: <http://elibrary.spbstu.ru/dl/2962.pdf/download/2962.pdf>
3. OMG® Unified Modeling Language® (OMG UML®) Version 2.5.1 December 2017. URL: <https://www.omg.org/spec/UML/>
4. Маклаков, С. В. Erwin и Erwin: CASE средства разработки информационных систем / С. В. Маклаков. – М.: Диалог-МИФИ, 2000. 256 с.
5. Каюмова А.В. Визуальное моделирование систем в StarUML: Учебное пособие/ А.В. Каюмова. Казань. – Казанский федеральный университет, 2013. – 104 с.