Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Институт информационных технологий

Кафедра микропроцессорных систем и сетей

Н. Б. Киреев, И. В. Кашникова

ТЕХНОЛОГИИ ПРОЕКТИРОВАНИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ИНФОРМАЦИОННЫХ СИСТЕМ. ЛАБОРАТОРНЫЙ ПРАКТИКУМ

Рекомендовано УМО в сфере дополнительного образования взрослых в качестве учебно-методического пособия для специальности переподготовки 9-09-0612-02 «Программное обеспечение информационных систем»

Минск БГУИР 2025

УДК 004.45-027.31(076.5)
ББК 32.973.202-018.2я73
К43

Р е ц е н з е н т ы:

кафедра информационных технологий
УО ФПБ «Международный университет «МИТСО»
(протокол № 7 от 28.02.2024);

профессор кафедры математических методов в экономике
УО «Белорусский государственный экономический университет»
доктор экономических наук, кандидат физико-математических наук,
профессор Э. М. Аксень

Киреев, Н. Б.

К43 Технологии проектирования программного обеспечения информационных систем. Лабораторный практикум : учеб.-метод. пособие / Н. Б. Киреев, И. В. Кашникова. – Минск : БГУИР, 2025. – 64 с. : ил.

ISBN 978-985-543-816-9.

Излагаются основные теоретические аспекты методологии анализа и проектирования программных систем с фокусом на использовании унифицированного языка визуального моделирования UML (Unified Modeling Language). Участники обучения будут ознакомлены с ключевыми этапами бизнес- и системного анализа, осуществляемыми в рамках унифицированного процесса разработки программного обеспечения (Unified Software Development Process, USDP). Применяя полученные знания на практике в ходе работы над небольшим проектом, слушатели смогут усвоить принципы и методы UML для создания эффективных моделей систем.

Предназначен для слушателей переподготовки по специальности 9-09-0612-02 «Программное обеспечение информационных систем».

УДК 004. 45-027.31(076.5)
ББК 32.973.202-018.2я73

ISBN 978-985-543-816-9

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ....4
РАЗДЕЛ 1. ТЕОРЕТИЧЕСКИЕ ОСНОВЫ
ДЛЯ МОДЕЛИРОВАНИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ....5
1.1. Основные понятия и определения в области анализа и проектирования....5
1.2. Объектно-ориентированный подход к проектированию
ПО и его стандартизация....6
1.3. Унифицированный процесс разработки ПО....7
1.4. Принцип MDA – от модели к коду. Аналитическая модель....7
1.5. Общие рекомендации по моделированию....9
1.6. Выбор инструментов для моделирования....12
РАЗДЕЛ 2. РАЗРАБОТКА УЧЕБНОГО ПРОЕКТА....13
2.1. Требования к учебному проекту....13
2.2. Структура учебного проекта....14
2.3. Подготовительный этап....14
2.4. Этап 1. Бизнес-моделирование (Business Model)....18
Лабораторная работа № 1 «Уточнение целей, моделирование
сущностей предметной области»....18
Лабораторная работа № 2 «Моделирование бизнес-субъектов,
бизнес-процессов и пространственного размещения программной
системы»....26
2.5. Этап 2. Выявление и анализ требований....33
Лабораторная работа № 3 «Выявление и документирование
функциональных требований к ПО»....33
Лабораторная работа № 4 «Моделирование вариантов использования»....37
2.6. Этап 3. Системный анализ. Моделирование объектов и классов....43
Лабораторная работа № 5 «Создание сценария для поиска
объектов программной системы и моделирование элементов GUI»....43
Лабораторная работа № 6 «Моделирование объектов
и классов приложения»....47
ПРИЛОЖЕНИЕ 1. Шихтовка металлолома....55
ПРИЛОЖЕНИЕ 2. Задания для самостоятельной работы....60
ЗАКЛЮЧЕНИЕ....62
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ....63

ВВЕДЕНИЕ

Лабораторный практикум «Технологии проектирования программного обеспечения информационных систем» представляет собой комплексный курс, разработанный для слушателей переподготовки по специальности 9-09-0612-02 «Программное обеспечение информационных систем» и курсов повышения квалификации для бизнес- и системных аналитиков. Основная цель пособия заключается в обучении студентов теоретическим аспектам и методологии анализа и проектирования программных систем с использованием унифицированного языка визуального моделирования UML (*Unified Modeling Language*).

Пособие охватывает все этапы бизнес- и системного анализа, которые проводятся в рамках унифицированного процесса разработки ПО (*Unified Software Development Process*, USDP). Этот процесс предлагает структурированный подход к разработке программного обеспечения, который включает в себя такие этапы, как сбор требований, анализ предметной области, моделирование системы, разработка архитектуры и тестирование.

Курс разбит на шесть практических работ, каждая из которых посвящена определенному этапу анализа и моделирования ПО. Слушатели начинают с изучения основных концепций и терминологии UML, а затем применяют полученные знания на практике, создавая визуальные модели предметной области. После этого они переходят к моделированию системы, создавая диаграммы классов, диаграммы последовательностей, диаграммы активностей и другие.

Особое внимание уделяется процессу создания архитектуры разрабатываемой программной системы. Слушатели учатся последовательно трансформировать визуальные модели предметной области в модели системного анализа, используя принципы и практики, описанные в пособии. Такой подход помогает лучше понять, как проектирование ПО связано с конкретными бизнес-потребностями и как создать эффективную архитектуру системы.

В итоге курс «Технологии проектирования программного обеспечения информационных систем» позволяет студентам освоить ключевые принципы и методы анализа и проектирования ПО с использованием UML. После окончания курса студенты будут готовы применять свои знания в практическом проекте и успешно работать в области программной разработки.

РАЗДЕЛ 1. ТЕОРЕТИЧЕСКИЕ ОСНОВЫ ДЛЯ МОДЕЛИРОВАНИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

1.1. Основные понятия и определения в области анализа и проектирования

Приведем основные понятия и определения, лежащие в основе методологий разработки ПО.

Программирование – процесс отображения определенного множества целей (требований) на множество машинных команд и данных, интерпретация которых на компьютере или вычислительном комплексе обеспечивает достижение поставленных целей [1].

Процесс разработки *ПО* – совокупность процессов, обеспечивающих создание и развитие ПО [1].

Модель разработки *ПО* – формализованное представление процесса разработки [1].

Жизненный цикл *ПО (ЖЦ ПО)* – период времени, который начинается с момента принятия решения о необходимости создания программного продукта и заканчивается в момент его полного изъятия из эксплуатации [1].

Цель разработки *ПО* – разработать качественное ПО, максимально полно удовлетворяющее реальным потребностям (целям) пользователей, уложившись в определенный срок и бюджет [1].

Архитектура *(architecture)* – логическая и физическая структура системы, сформированная всеми стратегическими и тактическими проектными решениями [1].

Артефакт *(artifact)* – элемент информации, используемый или порождаемый в процессе разработки программного обеспечения (например, внешний документ или результат работы). Артефактом может быть модель, диаграмма, описание или программный продукт [4].

Визуальное моделирование – процесс графического представления объектной модели с помощью некоторого стандартного набора графических элементов [1].

Бизнес-логика (*domain logic*) – совокупность правил, принципов, зависимостей поведения объектов предметной области (области человеческой деятельности, которую система поддерживает). Является синонимом термина «*логика предметной области*». Проще говоря, бизнес-логика – это реализация поведения объектов предметной области в программной системе. К ней относятся, например, формулы расчета ежемесячных выплат по ссудам (в финансовой индустрии), автоматизированная отправка сообщений электронной почты руководителю проекта по окончании выполнения частей задания всеми подчиненными (в системах управления проектами), отказ от отеля при отмене рейса авиакомпанией (в туристическом бизнесе) и т. д. [6].

1.2. Объектно-ориентированный подход к проектированию ПО и его стандартизация

Как следует из приведенных выше определений, в основе любой разработки ПО лежат *цели пользователей* (их пожелания, потребности, требования, функциональные обязанности и т. д.), которые они планируют достичь при помощи программной системы, обладающей определенным поведением или *функциональностью*. С выявления, описания и моделирования требуемой функциональности программной системы начинается разработка конкретной архитектуры ПО, независимо от выбранной модели процесса разработки: водопадная, итерационная инкрементальная или гибкая.

Выбор типа элементов, из которых будет создана архитектура системы и которые, в свою очередь, будут реализованы в программном коде, определяет как саму архитектуру, так и подходы к ее моделированию и разработке, а также выбор языка программирования. Современное ПО отражает реалии окружающего нас *объектного мира* и для него наиболее перспективным и популярным является объектно-ориентированный подход (ООП), поэтому архитектура, создаваемая в процессе работы над проектом, тоже является объектно-ориентированной.

Теория ООП широко описана в литературе [1, 3]. Для дальнейшего понимания хода процесса разработки и логики построения моделей полезно учитывать следующие положения [1].

– *Объектно-ориентированный анализ* (OOA) – методология, при которой требования к системе воспринимаются с точки зрения классов и объектов, выявленных в предметной области.

– *Объектно-ориентированное проектирование* (OOD) – методология проектирования, объединяющая процесс объектной декомпозиции и приемы представления логической и физической, а также статической и динамической моделей проектируемой системы.

– *Объектно-ориентированное программирование* (OOP) – методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования.

– *Поведение* (функциональность) объектно-ориентированной программной системы реализуется взаимодействующими объектами, которые относятся к связанным между собой классам программной системы.

– Взаимодействие объектов описывается *клиент-серверной моделью*: объект «*клиент*» посылает *сообщение* объекту «*сервер*» с целью вызова соответствующей операции.

1.3. Унифицированный процесс разработки ПО

Унифицированный процесс разработки программного обеспечения (*Unified Software Development Process, USDP*) представляет собой структурированный и итеративный методологический подход к разработке программных систем. USDP предоставляет набор фреймворков, охватывающих все этапы жизненного цикла разработки программного обеспечения – от начальной концепции до окончательной поставки и поддержки.

Основные характеристики унифицированного процесса включают в себя следующее.

– Итеративность и инкрементальность: разработка ПО осуществляется через последовательные итерации, позволяя постепенно уточнять и дополнять требования.

– Архитектурная ориентированность: USDP уделяет внимание разработке архитектурных решений, подчеркивая их важность для успешного выполнения проекта.

– Фокус на компонентной архитектуре: поддерживается идея построения ПО из множества малых, независимых компонентов.

– Использование языка моделирования UML (*Unified Modeling Language*): UML применяется для визуализации, спецификации и документации проекта.

– Ролевой подход: команды разработки делятся на роли (например, аналитики, проектировщики, тестировщики), каждая из которых выполняет свои специфические функции.

– Гибкость: USDP адаптируется к различным видам проектов и может быть настроен под конкретные потребности организации.

Унифицированный процесс разработки программного обеспечения был создан на основе лучших практик и опыта в области разработки программных продуктов, и его цель – обеспечить систематизированный и эффективный подход к созданию высококачественных программных систем.

1.4. Принцип MDA – от модели к коду. Аналитическая модель

Основной целью разработчиков является создание качественного программного обеспечения, которое максимально удовлетворяет реальным требованиям пользователей, при соблюдении оговоренных сроков и бюджета [3]. Главное в данном контексте – получить рабочий код, реализующий требуемую функциональность, вместо формального следования определенному процессу или создания набора моделей.

В идеале быстрым решением может быть непосредственное написание кода с минимальными затратами времени на моделирование и разработку архитектуры системы. Такая практика находит применение в гибких методологиях разработки, таких как XP (*eXtreme Programming*), Scrum и др. Однако у нее есть

существенные ограничения при использовании в крупных проектах с большим числом участников.

Альтернативой является так называемое MDA (*Model Driven Architecture*) – разработка ПО в процессе последовательной трансформации моделей. На рис. 1 приведена модель MDA из источника [3] с небольшими текстовыми дополнениями, поясняющими смысл последующих действий в реальном проекте.

Рис. 1. Принцип MDA [3]

Принцип MDA «От модели к коду» представляет собой методологический подход в области разработки программного обеспечения, который выделяет важность использования моделей для создания высокоуровневых аналитических представлений перед переходом к кодированию.

Принцип «От модели к коду» в MDA предполагает, что на начальных этапах разработки программного продукта акцент делается на создании высокоуровневых аналитических моделей, а затем эти модели автоматически преобразуются в код на целевой платформе, что повышает эффективность и повторное использование кода.

В соответствии с принципом MDA строятся три глобальные модели:

- ***бизнес-модель*** (модель предметной области или ***domain model***);
- ***модель программной системы*** (модель системного анализа или ***application model***);
- ***модель проектирования*** (***design model***).

Первые две модели объединяются в одну ***аналитическую модель***.

Согласно [3] «***Аналитическая модель*** *– это точное, четкое представление задачи, позволяющее отвечать на вопросы и строить решения. На этапе проектирования вы будете ссылаться именно на аналитическую модель, а не на исходную формулировку задачи»*.

1.5. Общие рекомендации по моделированию

Каждая из глобальных моделей, входящих в состав аналитической модели, может содержать несколько более мелких моделей, которые, в свою очередь, включают одну или несколько ***UML-диаграмм***. Каждая диаграмма отображает предметную область или программную систему (подсистему, модуль) с определенной точки зрения и помещается в предназначенное для нее ***представление*** (*View*).

Термин «*представление модели*» или «*вид модели*» означает различные перспективы, аспекты или уровни детализации, которые могут быть использованы для анализа и проектирования системы. В рамках аналитической модели каждая глобальная модель может включать несколько подмоделей, представляющих собой конкретные аспекты системы. Эти подмодели, в свою очередь, могут содержать одну или несколько диаграмм UML.

Все проектные артефакты, в том числе подмодели, диаграммы, файлы, внешние ресурсы, которые создаются и используются в рамках USDP, хранятся в пакетах (папках, *англ. package*) или ссылками привязаны к ним. Имя папки отражает название соответствующей модели. Каждая модель относится к деятельности определенного специалиста (группы специалистов). Например, папки *Business Use Case Model* и *Business Object Model* относятся к деятельности бизнес-аналитиков, папки *Use Case Model* и *Analysis Model* – к деятельности системных аналитиков, а пакет *Design Model* (модель проектирования) – к деятельности проектировщиков программной системы.

Основными представлениями в рамках аналитической модели (рис. 2) являются:

- *Logical View* – представление логических структур данных;
- *Use Case View* – представление функциональности;
- *Process View* – представление процессов;
- *Deployment View* – представление размещения.

Каждое подразделение модели представляет собой специфичный аспект системы, и диаграммы UML в этих подмоделях служат инструментами для визуализации, анализа и проектирования соответствующих аспектов. Данный подход позволяет команде разработчиков более детально рассмотреть области анализа и специализироваться в каждой из них.

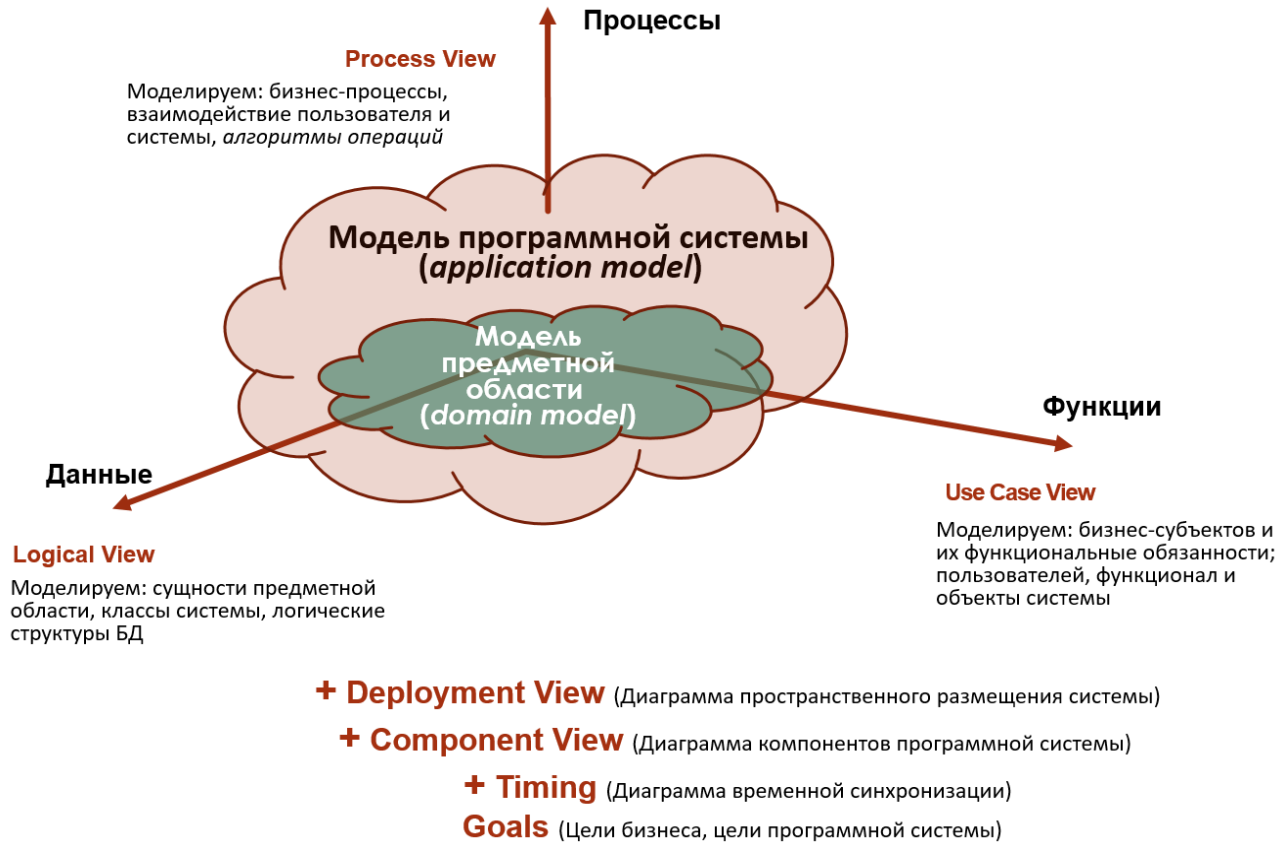


Рис. 2. Представления моделей в UML [8]

Кроме принципов ООП следует принимать во внимание общие рекомендации по построению визуальных моделей [8].

– Создаваемые модели должны быть полезны заинтересованным сторонам: пользователям (заказчикам) и разработчикам (проектировщикам, архитекторам и т. д.). Перед построением каждой диаграммы необходимо уточнить ее конкретные цели, а также для кого из участников проекта она предназначена. Не следует строить диаграммы «по аналогии» с другими проектами, так как каждый из них, по сути, уникален и требует своих решений. Применение шаблонов и паттернов оправдано при проектировании архитектуры, реализующей стандартные функции системы (прием-передача данных, сохранение данных и т. д.), но не при моделировании предметной области или функций бизнес-логики.

– Модели должны быть понятными, читабельными, исключающими неоднозначную интерпретацию представленной в них информации. Если на диаграмме много элементов (более 20–30) и сложные связи, то ее следует упрощать разбивая на несколько более простых диаграмм либо жертвовать излишней детализацией.

– В моделях, в соответствующих представлениях, фиксируется все, что непосредственно связано с самой программой и ее функционированием, а также с бизнесом или бизнес-процессом, который она обеспечивает.

– Модели, как и некоторые другие артефакты, например глоссарий, создаются итерационно. Даже опытному аналитику с первого раза редко удается

построить удачную модель, адекватно передающую пользователю всю необходимую информацию.

– Количество создаваемых артефактов должно быть достаточным, чтобы отразить все важные аспекты разрабатываемой программной системы, и минимальным, чтобы максимально сократить время, затрачиваемое на этапы анализа и проектирования.

– Качество работы аналитика должно определяться не по количеству создаваемых им артефактов, а по тому, насколько быстро и адекватно воспринимается смысл требований бригадой разработчиков и как быстро можно перейти к процессу написания программного кода.

При моделировании на UML кроме принципов ООП полезно принимать во внимание следующие рекомендации.

1. Учитывать потребности заинтересованных сторон: перед созданием каждой диаграммы необходимо определить ее цели и аудиторию, чтобы модель была нужной и информативной для пользователей и разработчиков. Следует избегать простого копирования диаграмм из других проектов, так как каждый проект уникален.

2. Обеспечить понятность и читаемость моделей: диаграммы должны быть понятными и исключать неоднозначную интерпретацию информации. Если диаграмма слишком сложная, ее следует упростить или разбить на более простые части.

3. Фиксация связанной информации: в моделях должна быть представлена вся необходимая для разработчиков информация, связанная с доменной областью, с бизнес-процессами, функционированием и структурами программной системы.

4. Итеративный процесс создания моделей: модели создаются итерационно, поэтапно улучшаясь и дополняясь. Даже опытному аналитику может потребоваться несколько итераций для создания идеальной модели.

5. Найти баланс между количеством и качеством артефактов: количество создаваемых артефактов должно быть достаточным для отражения всех важных аспектов системы, но при этом минимальным для сокращения времени, затрачиваемого на анализ и проектирование.

6. Оценка качества работы аналитика: важно учитывать не количество созданных ими артефактов, а то, как выполненная аналитиками работа позволила разработчикам улучшить восприятие требований и ускорить процесс создания кода.

Поэтому при построении визуальных моделей на UML необходимо удовлетворять потребности пользователей и разработчиков, обеспечивать понятность и читаемость моделей, фиксировать необходимую информацию, работать итеративно, находить баланс между количеством и качеством артефактов, а также оценивать качество работы аналитика по пониманию требований разработчиками и скорости перехода к разработке кода.

1.6. Выбор инструментов для моделирования

Инструментарий, доступный аналитикам для моделирования, является важным аспектом в процессе разработки.

Существует множество инструментов, в которых, используя нотацию языка UML, можно строить визуальные модели как предметной области, так и программной системы. Их спектр – от простых графических редакторов (типа *MS Visio*) до сложных систем автоматизированного проектирования с генерацией программного кода на выбранном языке программирования. Какой инструментарий выбрать?

Определение подходящего инструментария может зависеть от конкретных потребностей проекта, предпочтений команды и доступных ресурсов. Некоторые аналитики могут предпочитать использовать инструменты с более широкими функциональными возможностями, тогда как другие предпочитают простоту и удобство в использовании.

В принципе аналитик может строить и обсуждать модели с заказчиком даже на салфетках, сидя за чашкой кофе в уютном кафе, поэтому один из четырех принципов манифеста гибкой (*Agile*) разработки [7] гласит: «*Люди и взаимодействие* ***важнее*** *процессов и инструментов»*. Не так важны процессы и инструменты, которые используются в разработке, как взаимопонимание между заказчиками и исполнителями и одинаковое представление о программном продукте, который создается совместными усилиями.

Однако будет лучше, если рабочий инструмент аналитика будет обладать следующим минимальным набором возможностей:

– иметь нотацию (стереотипы) для раздельного моделирования предметной области (машинно-независимых моделей CIM), архитектуры системного анализа (платформо-независимых моделей PIM) и для проектирования архитектуры программной системы (платформо-зависимой модели PSM);

– позволять вводить и сохранять спецификации элементов модели;

– обеспечивать создание моделей, предназначенных для обсуждения с заказчиками и пользователями, которые будут интуитивно понятны всем, даже тем, кто не имеет опыта работы с языком UML;

– позволять строить модели с позиции как минимум трех основных представлений: логической структуры (*Logical View*), функциональности (*Use Case View*), а также процессов или состояний (*Process View*);

– обеспечивать принципы модульности и поддерживать иерархичность, как в самих моделях, так и в организационной структуре хранения и поиска информации. Каждый создаваемый в процессе разработки артефакт должен храниться в определенном месте, а участники проекта – иметь возможность его легко найти;

– поддерживать выбранный процесс разработки;

– желательно иметь портативную и/или онлайн-версию для удобства использования в различных сценариях работы.

Многими перечисленными выше достоинствами, включая интуитивно понятный интерфейс и портативную версию, обладает популярное CASE-средство StarUML™ (*WhiteStarUML*) [10].

StarUML предоставляет обширные возможности для UML-моделирования, включая разнообразные типы диаграмм и элементы, такие как классы, объекты, компоненты, варианты использования, действия, состояния и многие другие. Кроме того, инструмент обладает функциональностью генерации кода на различных языках программирования, основываясь на созданных UML-диаграммах. StarUML соответствует стандартам UML, обеспечивая возможность создания моделей, соответствующих индустриальным требованиям. Среда поддерживает совместную работу, что облегчает коллективное ведение проектов, и она доступна для использования на различных операционных системах, таких как Windows, macOS и Linux, обеспечивая универсальность в использовании (для лицензионных версий).

Поэтому представленный в настоящем пособии унифицированный процесс разработки ПО будет описан на примере *WhiteStarUML* (аналог StarUML с открытым кодом и публичной лицензией), но при желании может быть использован любой другой инструмент визуального моделирования. Подробное описание интерфейса StarUML представлено в [9].

РАЗДЕЛ 2. РАЗРАБОТКА УЧЕБНОГО ПРОЕКТА

2.1. Требования к учебному проекту

Целью настоящего пособия является практическое освоение методики визуального моделирования в рамках унифицированного процесса разработки ПО. Основной задачей является демонстрация возможностей методики на основе небольшого конкретного проекта, обладающего следующими характеристиками:

– содержание относительно небольшого объема разработки, что позволит эффективно пройти все этапы анализа, начиная от бизнес-моделирования и заканчивая моделями архитектуры системного анализа, в ограниченное количество занятий;

– отношение к незнакомой для обучающихся предметной области, но без избытка специфических терминов и процессов. Это стимулирует обучающихся вдумываться в смысл как исходных текстов, так и разрабатываемых им моделей;

– подробное описание, выполненное в стиле, близком к изложению, характерному для конечных пользователей системы (сумбурное, запутанное, с повторениями), что позволит слушателям столкнуться с реальными условиями работы аналитиков;

– отношение к многопользовательской системе, что добавит сложности и реалистичности в моделировании;

– кроме клиентских приложений содержание базы данных, что позволит обратить внимание слушателей и на вопросы моделирования данных;

– реализация в виде функционирующего прототипа программной системы, чтобы слушатели могли увидеть, как моделирование отражается в действующем программном приложении.

В качестве учебного предлагается небольшой проект, исполнявшийся по заказу немецкой фирмы, бизнес которой был связан с переплавкой металлолома. Концепция системы представлена в *приложении 1* настоящего пособия.

2.2. Структура учебного проекта

На основе представленного в приложении 1 описания работа разделена на три последовательных этапа, каждый из которых включает в себя проведение двух лабораторных работ. В ходе выполнения этих этапов осуществляется создание визуальных моделей с использованием CASE-средства StarUML [10]. Весь процесс моделирования осуществляется в соответствии с методологией USDP, описанной в [3].

Этап I. Бизнес-моделирование (*Business Model*)

Лабораторная работа № 1 «Уточнение целей, моделирование сущностей предметной области».

Лабораторная работа № 2 «Моделирование бизнес-субъектов, бизнес-процессов и пространственного размещения программной системы».

Этап II. Выявление и анализ требований

Лабораторная работа № 3 «Выявление и документирование функциональных требований к ПО».

Лабораторная работа № 4 «Моделирование вариантов использования».

Этап III. Системный анализ. Моделирование объектов и классов

Лабораторная работа № 5 «Создание сценария для поиска объектов программной системы и моделирование элементов GUI».

Лабораторная работа № 6 «Моделирование объектов и классов приложения».

2.3. Подготовительный этап

Цель подготовительного этапа заключается в создании в приложении StarUML организационной структуры проекта в соответствии с шаблоном унифицированного процесса разработки ПО (USDP).

Перед выполнением работ с приложением WhiteStarUML необходимо запустить его портативную версию (файл *WhiteStarUML_portabl.exe*) или выпол-

нить полную инсталляцию продукта (файл *WhiteStarUMLSetup-5.9.1.exe*), дополнив ее набором элементов бизнес-модели (файл *setupBM0.5.exe*), скачав указанные файлы с ресурса *SourceForge.net* [10].

Проект – основная структурная единица в StarUML™. Он может содержать одну или более визуальных моделей. Проект – корневой пакет верхнего уровня, которому присваивается определенное имя. В общем случае один проект сохраняется в одном файле в формате: *имя проекта.uml* (*.uml* – расширение файлов *WhiteStarUML*).

Организационная структура проекта представляет собой систематизированное иерархическое распределение ресурсов, ответственностей и задач внутри проекта. Она включает в себя разделение проекта на логические модули, подразделения или группы, каждая из которых отвечает за определенные функции или аспекты работы. Организационная структура проекта – это своего рода «шкаф с полками», предназначенный для упорядоченного ввода и хранения артефактов. Без ее создания и поддержки любой проект превратится в хаос. Как правило, полная структура проекта представлена в браузере (*Model Explorer*) и может быть определена с помощью пакетов, подсистем и модельных элементов. Для диаграмм определяется формат по умолчанию. Каждый артефакт, который используется в рамках проекта, должен храниться в соответствующем проектном модуле (папке) и быть доступным из браузера.

При открытии нового проекта в StarUML необходимо выбрать ***подход*** и произвести конфигурацию ***профиля***, добавить в структуру проекта соответствующие модели.

Подход *(Approach)* – шаблон структуры проекта (включает представления, модели, диаграммы, стереотипы и фреймворки). Он выбирается разработчиками ПО в зависимости от применяемой методологии разработки и требований групп разработчиков.

Профиль (*Profile Manager*) – позволяет включать/отключать модели и ***фреймворки*** (библиотеки спецификаций классов для прикладных инструментов типа MFC, VCL или JFC). На рис. 3 представлено окно нового проекта с выбором подхода.

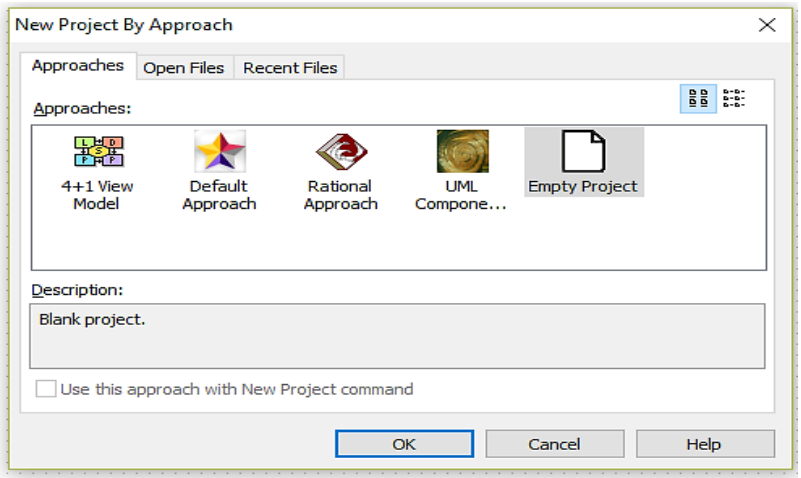


Рис. 3. Окно выбора подхода

Для обеспечения дальнейшей практической работы над учебным проектом создадим организационную структуру проекта по шаблону унифицированного процесса разработки ПО (*Unified Software Development Process*).

Для этого при запуске приложения StarUML или WhiteStarUML в окне *New Project By Approach* выбираем *Rational Approach*. Далее в окне управления профилем (*Profile Manager*) к стандартному UML-профилю (*UML Standard Profile*) подключаем профили бизнес-модели (*Business Model*) и модели данных (*Data Modeling*).

Далее в браузере проекта добавляем папки моделей с установкой соответствующих стереотипов (строка *Stereotype*):

– в представлении *Logical View – Business Object Model*, *Analysis Model* и *Design Model*;

– в представлении *Use Case View – Business Use Case Model* и *Use Case Model*.

Размещенные в браузере модели выставляем на организационных диаграммах соответствующих представлений (*Main Use Case View* и *Main Logical View*).

На диаграмме *Main Logical View* добавляем папку для модели классов приложения (классы системного анализа, *Application Classes*) и папку для модели структуры данных (*Database Model*). При необходимости на диаграммах можно добавить поясняющие примечания.

Таким образом, структура проекта (браузер проекта) будет иметь вид, представленный на рис. 4.

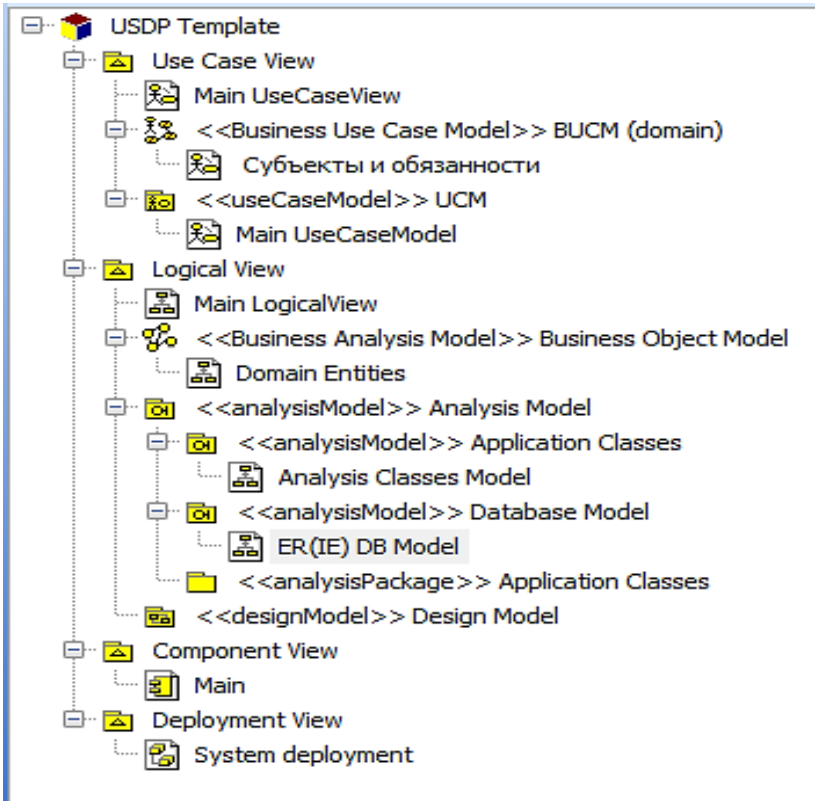


Рис. 4. Шаблон структуры проекта в рамках USDP

Организационные диаграммы представлений *Use Case View* и *Logical View* будут иметь вид, как на рис. 5 и 6.

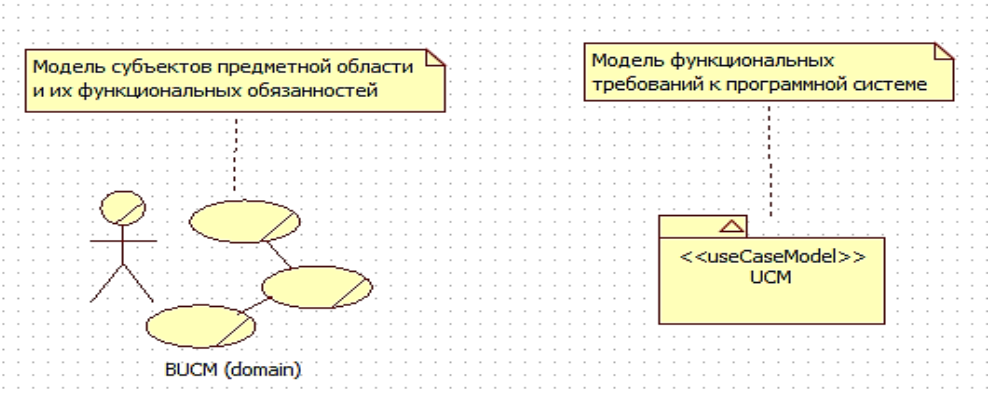


Рис. 5. Организационная диаграмма представления Use Case View

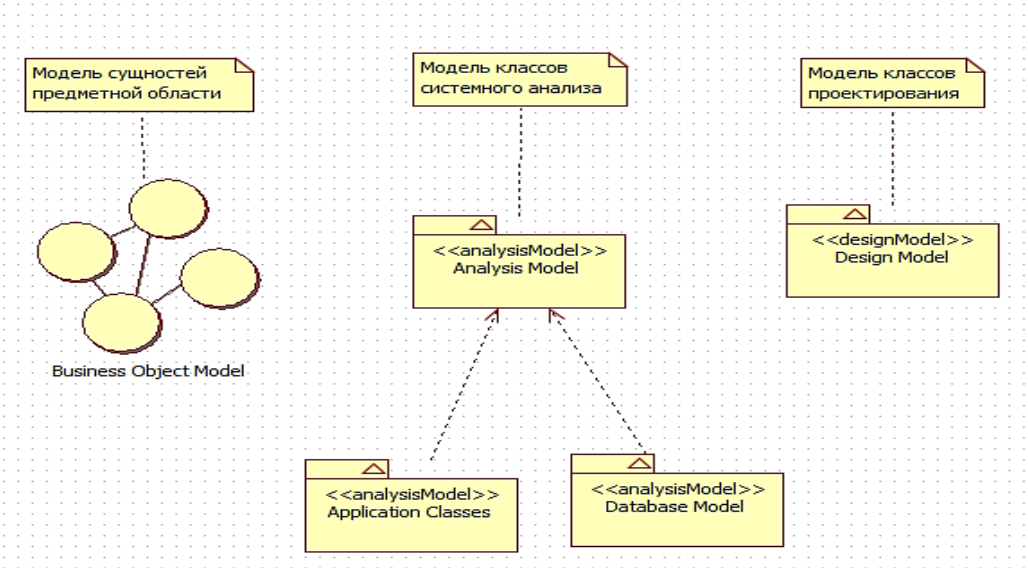


Рис. 6. Организационная диаграмма представления *Logical View*

Следует отметить, что данное пособие в первую очередь предназначено для освоения методики визуального моделирования, а не для изучения конкретного инструмента моделирования, такого как StarUML. В случае возникновения трудностей с подключением моделей, папок и других элементов в структуру проекта рекомендуется обращаться к инструкции по использованию данного CASE-средства [9].

После того как структура проекта будет создана, его следует сохранить, присвоив уникальное имя. Поскольку это общая унифицированная структура и ее можно использовать во всех случаях, когда требуется бизнес-моделирование, то проект можно сохранить в качестве шаблона для последующих разработок, задав соответствующее имя, например *TemplateUSDP.uml*.

2.4. Этап 1. Бизнес-моделирование (Business Model)

В результате выполнения лабораторных работ по бизнес-моделированию будут созданы следующие артефакты:

1. Выявлены и задокументированы цели, включая бизнес-цели, цели программной системы и нормативные ограничения в соответствии с законодательством.
2. Составлен текстовый глоссарий специфических терминов предметной области.
3. Создана модель сущностей предметной области с описанием их существенных характеристик (атрибутов).
4. Разработана модель субъектов предметной области с подробным описанием их деятельности и функциональных обязанностей.
5. Построены модели бизнес-процессов для видов деятельности, в рамках которых субъекты – физические лица взаимодействуют с системой через графический пользовательский интерфейс.
6. Создана модель пространственного размещения рабочих мест и отдельных модулей, представляющих разрабатываемую программную систему.

Лабораторная работа № 1 «Уточнение целей, моделирование сущностей предметной области»

Цель лабораторной работы состоит в уточнении целей проектируемой программной системы, а также в выявлении, моделировании и описании бизнес-сущностей и соответствующего набора атрибутов. В рамках работы также требуется составление текстового глоссария терминов предметной области.

Теоретическая часть

При проектировании программных систем важной частью является формирование представления о предметной области. Этот процесс позволяет абстрагироваться и организовать реальные объекты или идеи, с которыми в будущем взаимодействует создаваемая система.

Бизнес-сущность (сущность предметной области, *business entity*) – это абстракция реального объекта, группы однотипных объектов или концептуального понятия предметной области, характеризуемая набором существенных характеристик (данных), прямо или косвенно связанная с проектируемой программной системой [4].

Некоторые из этих сущностей будут прямо представлены в виде классов программной системы, в то время как другие, хотя не будут отражены в виде классов, тем не менее оказывают влияние на ход выполнения программы через свое участие в бизнес-процессах. При моделировании их следует отображать на диаграмме сущностей предметной области, дополняя соответствующими спецификациями.

Бизнес-сущности – это кандидаты в классы (со стереотипом *entity*), эти классы создают объекты программной системы, отвечающие за целостность и хранение данных. Выявление бизнес-сущностей, а вместе с ними набора характеризующих их атрибутов, которыми будет оперировать разрабатываемая программа, – главная цель данной лабораторной работы.

Для моделирования бизнес-сущностей в языке UML применяется стереотип *Business Entity*.

Важно отметить, что не все выявленные сущности будут отображены в виде классов программной системы. Только те из них, которые инкапсулируют данные, являющиеся существенными с точки зрения функционирования программы, будут представлены в ее структуре. Сущности, чьи атрибуты не используются в процессах функционирования программы, но имеют значение для общего понимания бизнес-системы или бизнес-процессов, будут отражены в моделях предметной области. Это позволит команде разработчиков лучше понять предметную область и процессы, происходящие в ней.

Пример. В приложении 1 имеются сущности «Чан для плавления» и «Дозировочный Контейнер». Они участвуют в общем технологическом процессе и их следует отобразить на диаграмме сущностей, однако атрибутов (данных), которые будет контролировать разрабатываемая система, у них может и не быть. В этом случае они не будут отображаться в классах программной системы.

Цели (*goals*) – согласно [2] относятся к функциональным требованиям самого высокого уровня абстрактности, к так называемым бизнес-требованиям (*Business Requirements*).

При этом цели следует подразделить на три группы: *цели отрасли* (контекст), *цели предприятия* (бизнес-цели) и *цели программной системы* [8].

Выявленные в ходе разработки ПО цели, относящиеся к первым двум группам, могут оказаться слишком абстрактными с точки зрения последующей разработки ПО или недоступными для непосредственного использования разработчиками. Поэтому именно *цели программной системы* следует выявить, конкретизировать, детализировать, проверить на полноту и достижимость (разработав объективные критерии) и использовать в разработке, начиная с моделирования сущностей предметной области. Таким образом, они должны быть представлены как *SMART-цели (Specific, Measurable, Attainable, Relevant, Time-bound)*.

Пример. В качестве примера рассмотрим программную систему для учета платных медицинских услуг. В таблице 1 показаны цели отрасли, бизнеса и разрабатываемой программной системы. Следует обратить внимание на отличие SMART-целей от более абстрактных. В левом столбце таблицы изображены пиктограммы UML, представляющие стереотип *Goal* (цель).

Таблица 1. Цели в проекте «Учет платных медицинских услуг»

Цель	Пример	Уровень абстрактности	Необходимость	Артефакт
 Общие цели (цели отрасли)	Улучшение медицинского обслуживания населения	самый высокий	необязательны	текст Vision
 Цели Бизнеса	Получение прибыли от оказания платных медицинских услуг населению	высокий	желательны	текст Vision
 Цели ПС	· Учет платных мед.услуг, оказываемых поликлиникой населению; · Учет врачей поликлиники, оказывающих платные мед.услуги; · Учет пациентов, пользующихся платными мед.услугами; · Расчет стоимости платных мед.услуг, оказанных пациентам	**SMART**	**обязательны**	**текст SRS**

Глоссарий – это текстовый документ, направленный на разъяснение специфических терминов (включая жаргон, сленги, аббревиатуры), используемых в определенной предметной области, с целью создания общего языка в команде разработчиков, уточнения понятий и исключения недопониманий.

В глоссарии фиксируются синонимы, омонимы и изменения терминологии, что помогает обеспечить единообразие терминов и избежать путаницы в понимании ключевых понятий. Обычно заказчик обладает экспертным знанием предметной области, но он тоже может пользоваться глоссарием для обеспечения единообразия в понимании терминологии между заказчиком и разработчиками.

Глоссарий при этом не должен дублировать информацию, введенную в моделях и в спецификациях к ним. Важно избегать дублирования, информация вводится в проект всегда однократно.

Отличительной чертой глоссария является его специфичность и ориентация на потребности команды разработчиков. Он не служит для формального определения терминов, а скорее предназначен для их разъяснения с точки зрения функционирования конкретной программной системы.

Пример. Для изучаемого нами проекта рассмотрим термин **шихтовка** и связанные с ним **шихтовать**, **шихта**, взятые из энциклопедических словарей [*enc-dic.com*]:

Шихтовка *-и, ж. спец. Действие по знач. глаг. Шихтовать;* ***Шихтовать*** *-ту́ю, -ту́ешь; прич. страд. прош. шихто́ванный, -ван, -а, -о; несов., перех. спец. Составлять шихту;* ***Шихта*** *(от нем. Schicht) – смесь сырых материалов, а в некоторых случаях (напр., при выплавке чугуна в доменной печи) и топлива, подлежащая переработке в металлургич., хим. и др. агрегатах. Ш. загружают либо в виде равномерной смеси, подготовл. вне агрегата, либо порциями или слоями, состоящими из отд. ее составных частей.*

В данном примере стоит отметить, что в принципе предоставленные определения способны расширить знания команды разработчиков, но они не отражают смысл данного термина в контексте разрабатываемой программной системы. Аналогично и формальное определение типа: «*Шихтовка – это процесс составления плавильной смеси*» – также не связано с предметом конкретной разработки.

Поэтому в глоссарии будет предпочтительнее написать следующее определение: «*Шихтовка (шихтовать) – это процесс составления плавильной смеси, в котором обеспечивается заданная пропорция набираемых металлических компонентов (МК) и контролируется подача добавок (кокса, извести и спецкокса) в соответствии с набранным суммарным весом МК».*

Понятия «плавильная смесь», «кокс», «известь», «спецкокс», «заданная пропорция МК», «суммарный вес МК» в глоссарий вноситься не будут, так как они должны быть введены и описаны в спецификациях на диаграмме сущностей предметной области.

Выявление и моделирование бизнес-сущностей. Методики выявления бизнес-сущностей могут быть разные (интервью с пользователями, наблюдение на рабочих местах, сбор бланков и листов и т. д.). В данной лабораторной работе предлагается выявлять сущности предметной области и их атрибуты путем чтения текста концепции и анализа встречающихся в нем имен существительных. Если некая сущность участвует в бизнес-процессе/бизнес-системе и инкапсулирует данные, которыми должна оперировать разрабатываемая программа в соответствии с ее целями, то она показывается на диаграмме бизнес-сущностей, а набор их существенных характеристик (данных) должен быть показан в виде атрибутов.

При этом каждый атрибут должен отражать только существенные (с точки зрения требований к программе) характеристики, а не полный набор данных, которыми описывается реальный объект предметной области. Для определения, является ли конкретная характеристика реального объекта предметной области существенной, соотнесем ее с абстрактным представлением функциональных требований – с *целями программной системы*.

На рис. 7 представлен фрагмент текста концепции с выделенными именами существительными, которые могут быть представлены в виде сущностей предметной области или в виде их атрибутов.

Инструментом для этого служит ***кран***, который забирает ***металлические компоненты*** (МК) своим ***магнитом*** и потом опускает в дозировочный ***контейнер***. В соответствии с ***весом набранных МК*** подаются ***кокс***, ***известь***, ***спецкокс***. По заполнении контейнера его содержание со всеми занесенными в него металлическими компонентами направляется в ***чан*** для плавления, в который добавляются ***легирующие добавки***. В конце ***рабочего дня*** чан направляется в **плавильную печь**. Так как в процессе плавления должны использоваться определенные ***соотношения веса различных МК***, программа должна обеспечить требуемый состав **плавильной смеси**. Этот контроль и составление смеси компонентов для плавки называется ***шихтовкой***.

Программная система устанавливается на ***рабочем месте крановщика*** и используется им для контроля над процессом шихтовки. Кроме того, она позволяет ***технологу*** задать определенную ***пропорцию МК*** в конечном ***сплаве*** и осуществлять сохранение **информации** о ***составе*** и свойствах полученных ***сплавов*** в ***базе данных***.

Рис. 7. Выявление сущностей в тексте концепции ПО

Не все имена существительные в тексте будут относиться к бизнес-сущностям: одни из них обозначают общие понятия, не имеющие значения в программной системе, другие относятся к внешним пользователям (у них важны не их атрибуты, а взаимодействие с программной системой), третьи обозначают процессы, четвертые – это данные, которые следует показать в виде атрибута сущности.

Отношения на диаграмме бизнес-сущностей. Поскольку сущности (*business entity*) моделируются на диаграмме классов и сами, по своей сути, аналогичны конкретными классам, экземплярами которых являются реальные объекты в предметной области, то в рамках языка UML между ними возможны три типа отношений: ассоциация (*association*), агрегация (*aggregation*), композиция (*composition*). Последние два типа описывают отношения «часть – целое» – это самые очевидные типы отношений в предметной области и их сразу можно показать в моделях.

Агрегация (Aggregation) означает, что объекты одного класса могут быть включены в объекты другого класса и при этом этот вложенный объект может находиться в нескольких объектах-контейнерах. Если объект-контейнер удаляется, то вложенный объект не удаляется. Может быть двунаправленной (отсутствие стрелки на отношении) либо однонаправленной (стрелка всегда направлена от агрегата к его части). Это означает, что агрегат всегда «знает» об атрибутах и операциях своих частей (рис. 8).

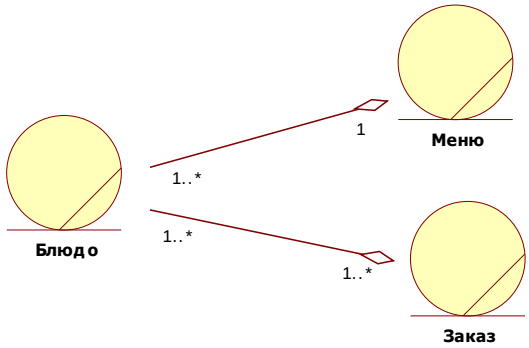


Рис. 8. Пример отношения агрегации

Композиция (Composition) подразумевает, что элементы одного класса могут включаться в элементы другого класса, причем такой вложенный элемент может присутствовать только в одном контейнере рис. 9. Если удаляется контейнер, то и вложенный элемент удаляется. Это очень тесная связь между «частью» и «целым».

Рис. 9. Пример отношения композиции

Ассоциация (Association) – семантическое (произвольное смысловое) и самое распространенное отношение между классами. Она может использоваться в разных случаях: если объекты одного или нескольких классов включены в объекты класса-контейнера, но не являются частями последнего, если один класс является атрибутом другого класса, если объекты одного класса используют один или несколько общих атрибутов и методов, взятых у объектов другого класса и т. д. Ассоциации могут быть как однонаправленными (направление указывается стрелкой на конце отношения), так и двунаправленными. Благодаря ассоциациям объекты исходного класса могут оперировать данными, которые хранятся в атрибутах объектов целевого класса.

Агрегации, композиции и ассоциации являются однотипными отношениями классов, их экземплярами являются связи (*object link*) между соответствующими объектами программной системы.

Для того чтобы окончательно выяснить ассоциации (их наличие, направление, связанные с ними операции и ограничения), используются объектные модели разрабатываемой программной системы (*sequence diagram, collaboration/ communication diagram*), которые создаются на этапе системного анализа. Надо учитывать, что при разработке бизнес-модели идентифицируются только будущие классы-сущности и возникают большие сомнения относительно ассоциаций между этими сущностями. При этом сами ассоциации и их направления часто определяются лишь субъективным мнением конкретного разработчика. Поэтому моделирование ассоциаций для классов программного приложения рекомендуется производить на этапе системного анализа после того, как была выделена структура взаимосвязей между объектами и определен общий набор классов приложения, включая классы-интерфейсы и классы-менеджеры.

Процесс выполнения лабораторной работы

1. Через меню *StarUML File – Open* загружаем созданный ранее шаблон проекта (файл *TemplateUSDP.uml).*
2. Читая концепцию программной системы (приложение 1), определяем и фиксируем в текстовом файле SMART-цели программной системы. К папке *Use Case View* с помощью закладки *Attachments* подкрепляем файл с целями, а также перенесенную в отдельный текстовой файл концепцию программной системы.
3. Анализируя текст концепции, фиксируем сущности предметной области в представлении *Logical View* в папке *Business Object Model* на диаграмме *Domain Entities*. Для этого создаем новый класс, переходя на закладку *Properties,* вводим имя класса и, кликая по строке *Stereotypes,* выбираем стереотип *business entity*. Далее, кликая по строке *Attributes*, с помощью окна *Collection Editor* (рис. 10) вводим атрибуты сущности.

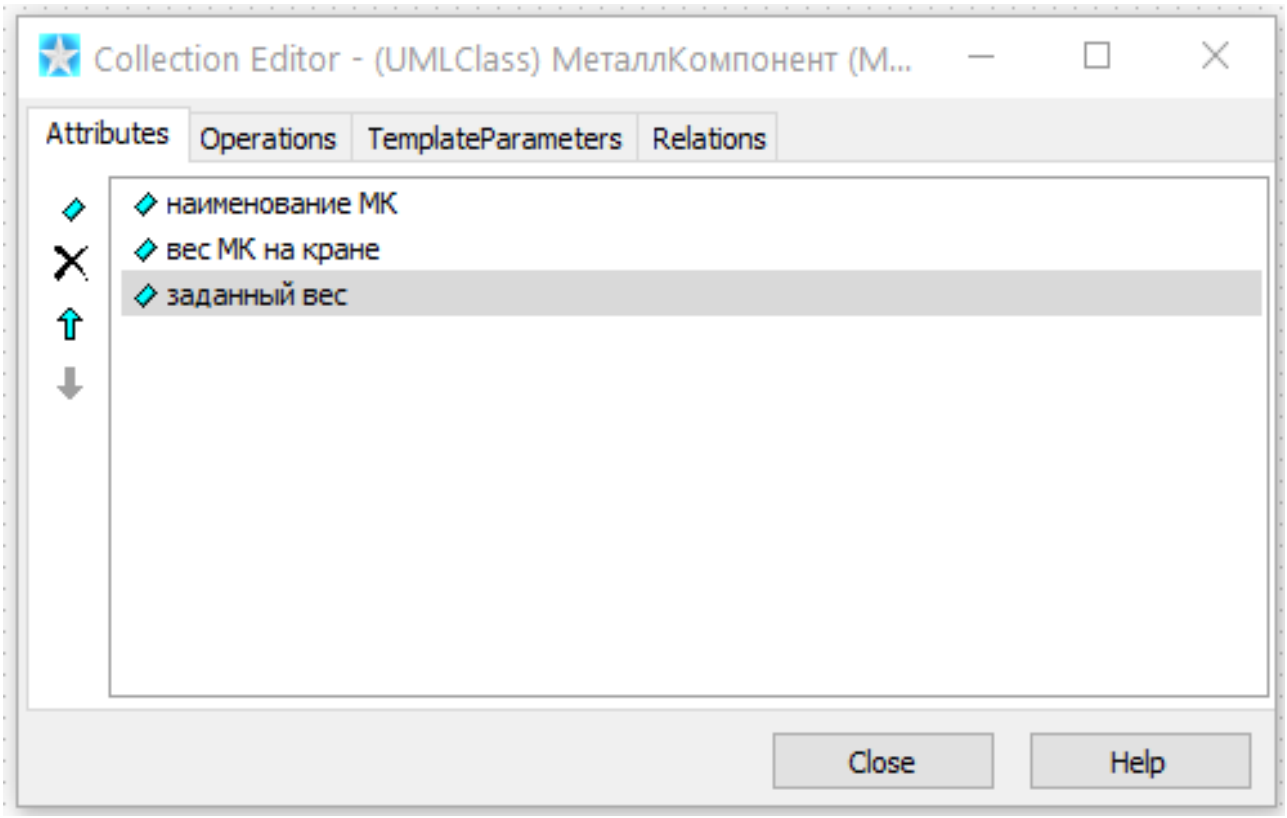


Рис. 10. Окно ввода атрибутов

4. Проводим описание бизнес-сущностей и их атрибутов. Информативность UML-диаграмм значительно повышается, если к элементам диаграмм добавить поясняющие текстовые описания. Для этого сущность или ее атрибут выделяется на диаграмме или в браузере проекта (*Model Explorer*), после чего описывается в окне *Documentation*. Примеры описаний бизнес-сущностей и их атрибутов приведены на рис. 11 и 12.

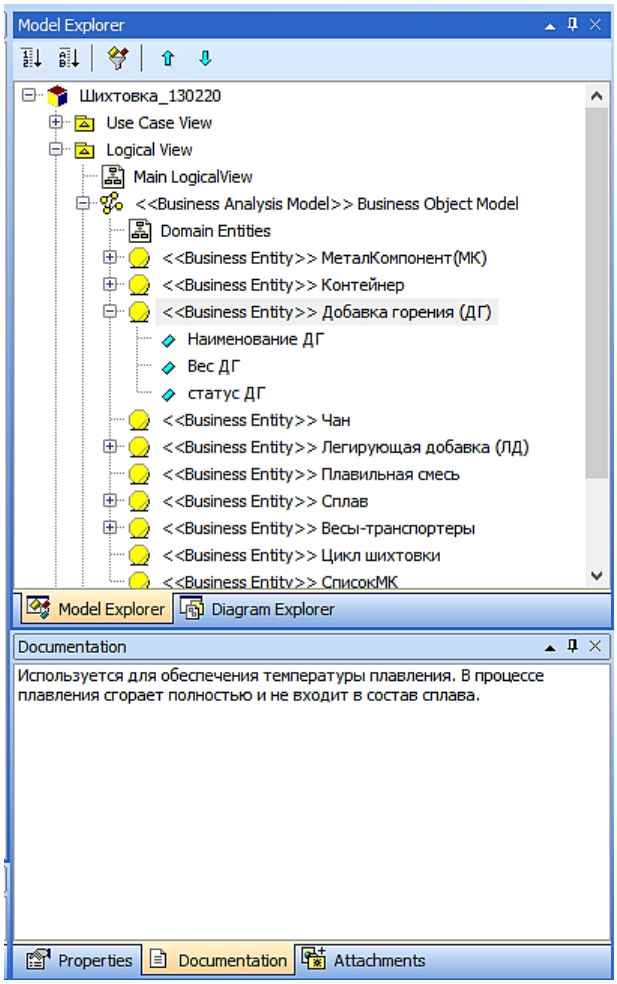


Рис. 11. Пример описания сущности

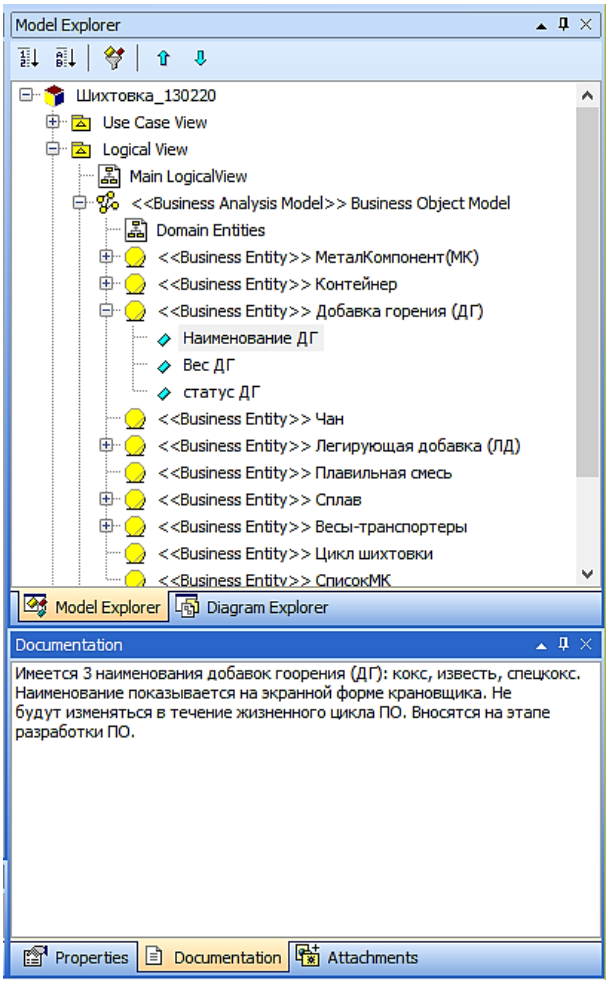


Рис. 12. Пример описания атрибута

Примечание: если рассматриваемое понятие не является бизнес-сущностью или атрибутом, а относится к бизнес-субъектам, выполняющим определенную деятельность в рамках рассматриваемого бизнес-процесса или бизнес-системы, то моделируем его в представлении *Use Case View* в папке *Business Use Case Model* в виде бизнес-актера (стереотип *Business Actor*).

5. Описываем отношения на диаграмме бизнес-сущностей. Показываем только агрегации и композиции, которые являются очевидными отношениями бизнес-сущностей.

6. Моделируем логическую структуру базы данных. В отличие от структуры классов приложения, в которую входят классы-сущности (entity), классы-интерфейсы (boundary) и классы-менеджеры (control), логическая структура базы данных моделируется с помощью диаграммы «*сущность – связь*» (*ERD, Entity Relationship Diagram*). В StarUML эту диаграмму можно строить в одном из двух вариантов: в нотации UML (упрощенная диаграмма классов) или в нотации Дж. Мартина («*вороньи лапки*»).

В нотации UML представляются сущности, их атрибуты, отвечающие за данные, предназначенные для хранения в БД, и ассоциации между ними. На концах ассоциаций указывается мощность (множественность), направление дей-

ствия ассоциации, могут быть показаны роли, сама ассоциация может быть представлена как идентифицирующая (в виде агрегации) и неидентифицирующая (в виде ассоциации).

Задание для лабораторной работы

Используя описание концепции системы (приложение 1) и CASE-средство *StarUML/WhiteStarUML*:

1) определите и зафиксируйте цели программной системы в соответствии с критериями SMART;

2) составьте текстовый глоссарий специфических терминов предметной области;

3) создайте диаграмму сущностей предметной области, покажите на ней бизнес-сущности и их атрибуты, опишите их в окне спецификаций (*Documentation).*

Лабораторная работа № 2 «Моделирование бизнес-субъектов, бизнес-процессов и пространственного размещения программной системы»

Цель лабораторной работы заключается в выявлении и моделировании бизнес-субъектов и их функциональных обязанностей, а также моделировании бизнес-процессов и пространственного размещения программной системы.

Теоретическая часть

Рассмотрим основы каждого из этих методов, понимание которых играет важную роль в создании эффективных и функциональных программных систем.

Модель бизнес-субъектов и их обязанностей представляется в виде диаграммы вариантов использования, основные элементы которой *Business Actor* (бизнес-субъект) и *Business Use Case* (функциональная обязанность субъекта) являются стереотипами бизнес-модели. Эта диаграмма позволяет визуализировать функциональную деятельность (поведение) бизнес-субъектов (бизнес-актеров) в контексте рассматриваемого бизнес-процесса или бизнес-системы.

Диаграмма бизнес-субъектов (*Business Use Case Model*) позволяет в рамках общего бизнес-процесса выделять и впоследствии детально моделировать отдельные, самые важные с точки зрения разработчиков процессы, – те, в которых происходит взаимодействие пользователей физических лиц и программной системы.

Приступая к разработке программного обеспечения, можно условно выделить следующие три круга лиц, прямо или косвенно заинтересованных в результатах его функционирования.

Заинтересованное лицо (*stakeholder*) – это индивидуум или группа внешних фигурантов, прямо или косвенно заинтересованных в результате создания системы. Это самый широкий круг лиц, связанных с разрабатываемой программой. В него входят как ее непосредственные пользователи, так и спонсоры, финансирующие проект, а также лица и организации, утверждающие нормативы, стандарты и требования, регламентирующие ее функционирование. Обычно не

возникает необходимости в построении визуальных моделей для этого круга лиц, однако с их интересами следует согласовывать высокоуровневые бизнес-требования (*business requirements*), т. е. цели бизнеса и цели программной системы.

Бизнес-субъект *или участник бизнеса/бизнес-процесса* (*business actor*) – это штатная единица или группа, исполняющая свои функциональные обязанности в данной бизнес-системе или в бизнес-процессе. Ими могут быть физлица – работники предприятий, занимающие определенные должности и исполняющие возложенные на них должностные обязанности (директор, главный инженер, контролер ОТК, мастер и т. д.), либо структурные подразделения (производственный отдел, сектор контроля качества и т. д.). В отдельных случаях в качестве бизнес-субъектов целесообразно показывать и внешние системы, исполняющие определенные функции (кран, весы-транспортеры и т. д.).

Следует заметить, что *бизнес-субъектами* могут быть и участники, не являющиеся *пользователями* программной системы, но оказывающие влияние на бизнес-процесс и косвенно на функционирование системы. Так, например, курьер в интернет-магазине не является пользователем, но он участвует в бизнес-процессе и исполнение или неисполнение им своих обязанностей косвенно влияет на работу программной системы.

Бизнес-субъекты моделируются в представлении *Use Case View* в папке *Business Use Case Model* при помощи стереотипа *Business Actor*, а их функциональные обязанности с помощью стереотипа *Business Use Case*. Построение этого типа диаграмм является одной из целей настоящей лабораторной работы.

Действующее лицо, пользователь (*actor*) (синонимы – актер, актант) – абстрактное понятие, которое характеризует внешнего пользователя (или группу пользователей), непосредственно взаимодействующего с программной системой.

Главное отличие действующих лиц от других бизнес-субъектов в том, что первые всегда имеют линию связи с системой, взаимодействуют с ней через интерфейс, инициируя исполнение функциональных сервисов, предоставляемых им системой. Действующие лица моделируются в представлении *Use Case View* в папке *Use Case Model*. Назначение и построение use-case-моделей рассматривается в лабораторной работе № 4 настоящего пособия.

Бизнес-процесс *(business process)* – последовательность взаимосвязанных действий (в языке UML действие называется *action*) или деятельностей (деятельность – *activity*), направленная на достижение значимого бизнес-результата.

Моделирование бизнес-процессов с использованием диаграммы деятельности (*Activity Diagram*) в языке UML позволяет получить визуальное (графическое) представление бизнес-процесса и последовательности действий в рамках самого бизнеса, что полезно при их анализе и оптимизации.

Бизнес-архитектура – как правило, под этим понятием понимается диаграмма пространственного размещения программной системы (*deployment diagram*), отражающая клиентские рабочие места, серверы бизнес-логики и данных, интерфейсы, включая программные (API), компоненты и связи между ними.

Процесс выполнения лабораторной работы

1. **Построение модели бизнес-субъектов и их обязанностей (*business use case diagram*)**

Как правило, модель бизнес-субъектов и их функциональных обязанностей располагается на одной диаграмме в папке *Business Use Case Model* в представлении *Use Case View*.

В *StarUML* в папке *Business Use Case Model* выбирается меню *Add Diagram – Business Use Case Diagram*, далее с помощью панели инструментов (*Toolbox*) на диаграмму заносятся пиктограммы *Actor* и *Use Case* устанавливаются соответствующие стереотипы (*Business Actor* и *Business Use Case*), в окнах *Documentation* вносятся соответствующие описания.

Синтаксис модели бизнес-субъектов и их обязанностей (*business use case diagram*) включает в себя элементы, представленные на рис. 13.

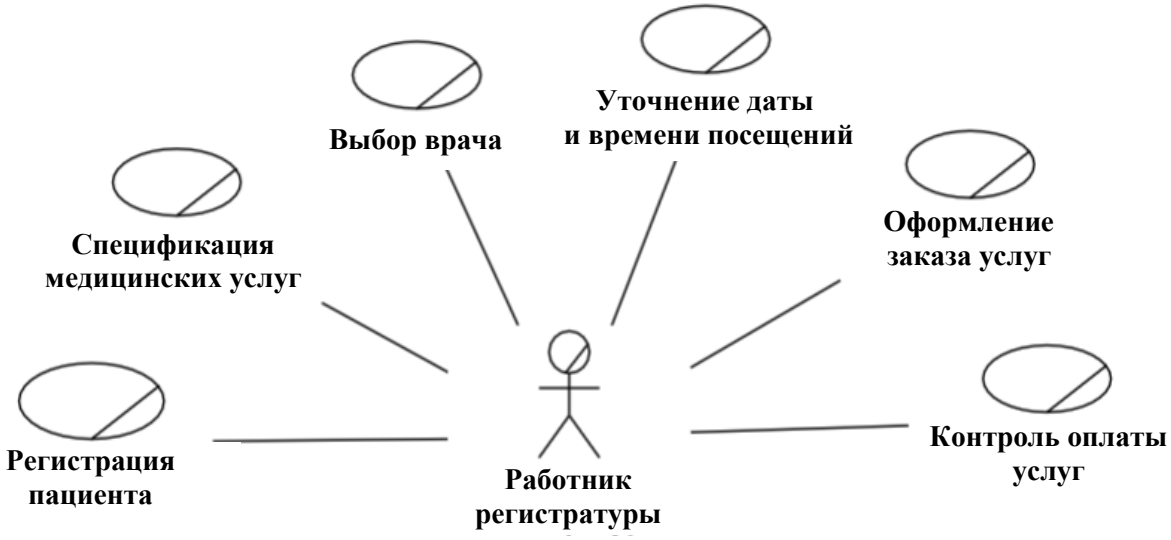


Рис. 13. Пример диаграммы бизнес-субъектов
и их функциональных обязанностей

2. **Моделирование бизнес-процессов (*activity diagram*)**

Бизнес-процессы моделируются с помощью диаграмм деятельности (*Activity Diagram*). Для более детального представления сложные процессы следует разбивать на более простые, иначе диаграммы будут нечитабельными или слишком абстрактными.

Использование диаграммы деятельности для моделирования бизнес-процессов помогает наглядно представить и понять последовательность действий в организации, что способствует анализу и оптимизации бизнес-процессов.

Для присоединения диаграммы деятельности к выбранному функционалу выделяем его на диаграмме и, кликая правой клавишей мыши, двигаемся по всплывающему меню: *Add Diagram – Activity Diagram*, после этого в окне диаграмм откроется чистое поле диаграммы деятельности с соответствующим набором инструментов.

При создании диаграмм деятельности применяются разные подходы. Важно избегать построения как слишком абстрактных и общих диаграмм, так и чересчур сложных, перегруженных элементами. Для упрощения перегруженных диаграмм можно моделировать отдельно каждый поток управления (основной поток, альтернативный поток) либо разбивать бизнес-процесс на более мелкие подпроцессы.

Ниже приведены отдельные примеры построения диаграмм деятельности (*activity diagram*), взятые из различных источников.

Пример 1. Самый простой вариант (рис. 14) построения диаграмм без детализации деятельности (*activity*) и «плавательных дорожек» (*swimlane*). Выполнен в Visual Paradigm [11].

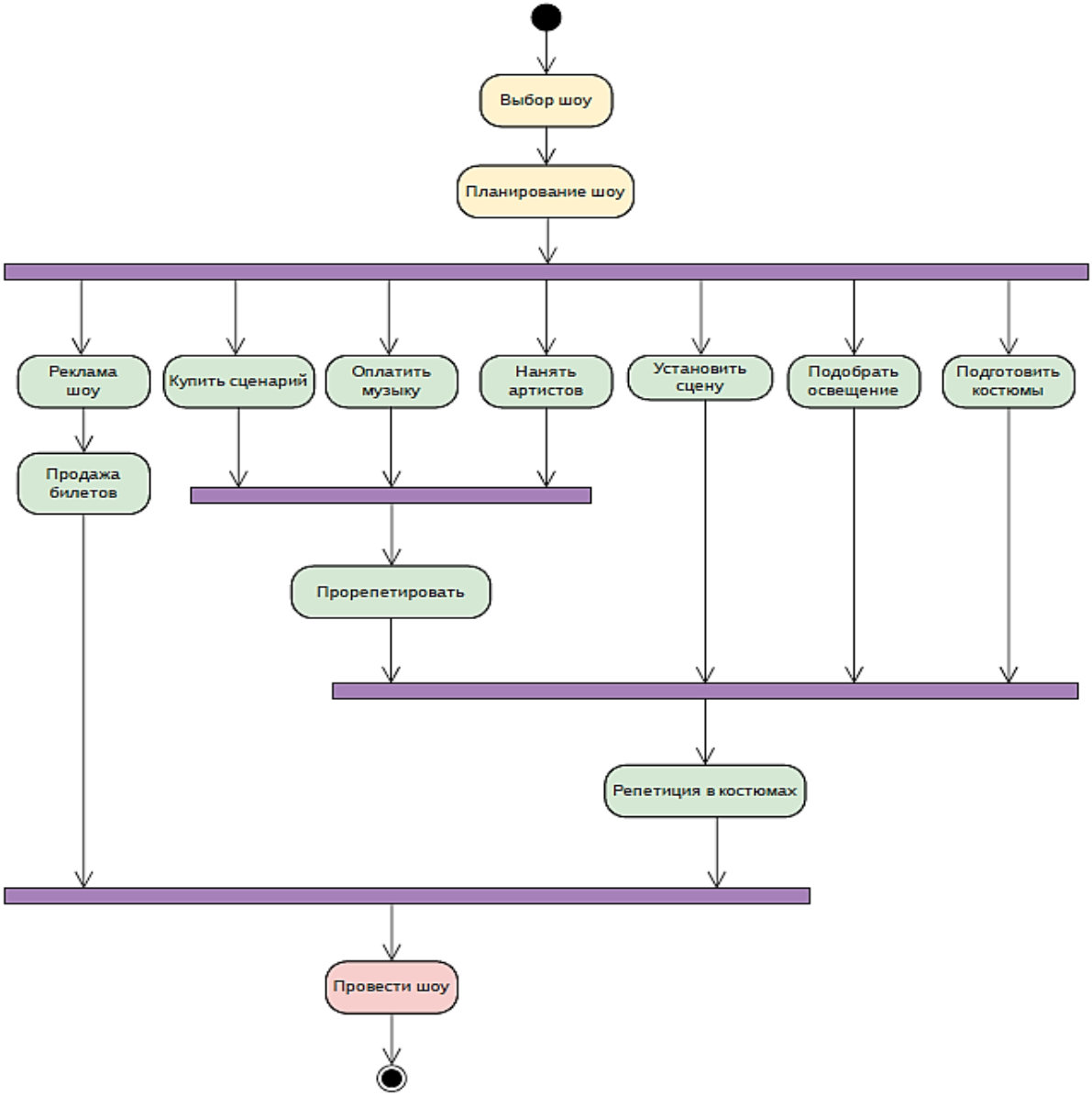


Рис. 14. Пример планирования шоу [11]

Пример 2. Простой вариант без детализации элементов *activity*, но с применением «плавательных дорожек» (рис. 15). Выполнен в Visual Paradigm [11].

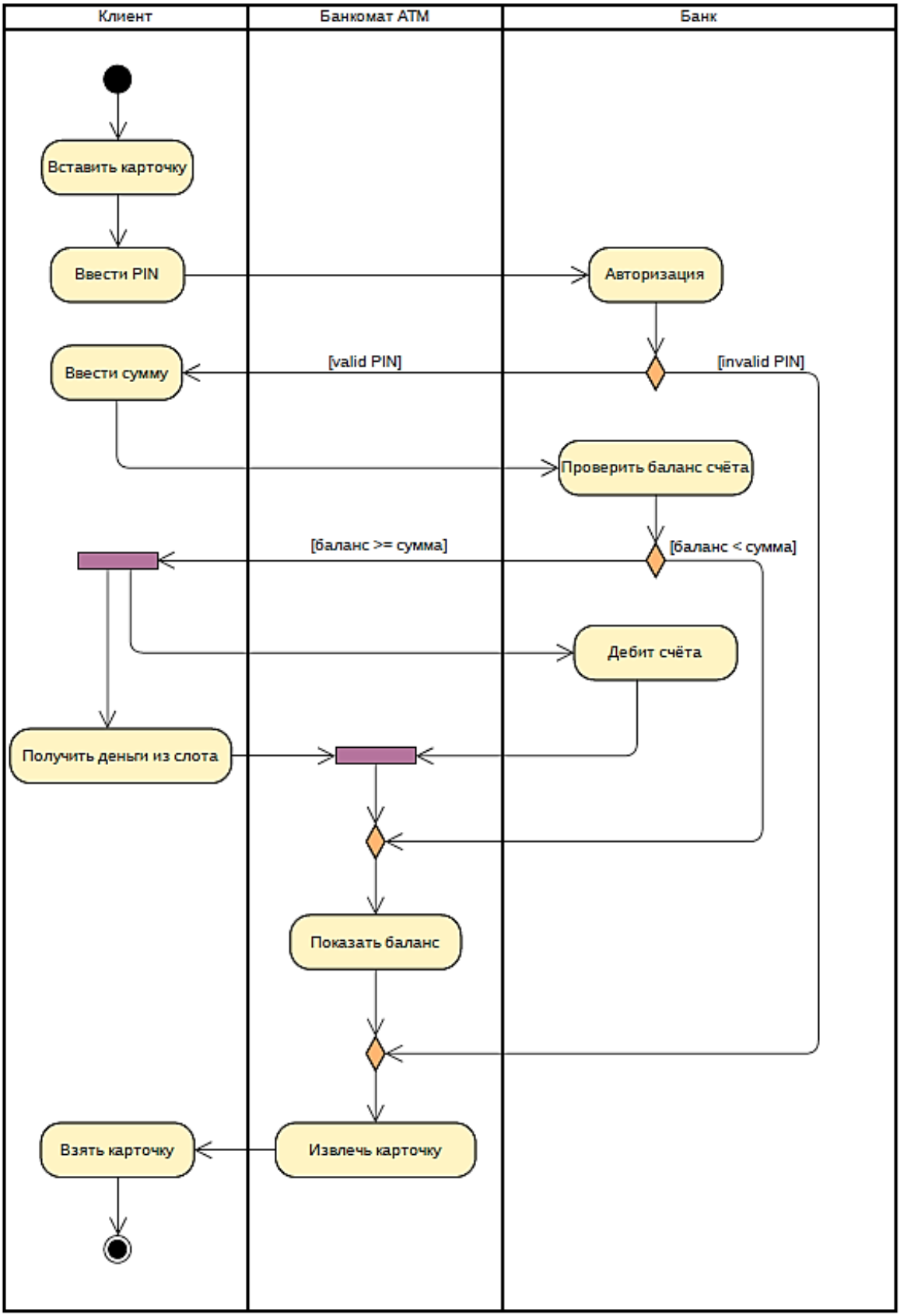


Рис. 15. Пример работы банкомата [11]

Пример 3. Диаграмма деятельности, особенностью которой является распределение действий бизнес-логики по различным состояниям системы (в данном случае по окнам программного приложения). Выполнена в CASE-средстве *WhiteStarUML* (рис. 16).

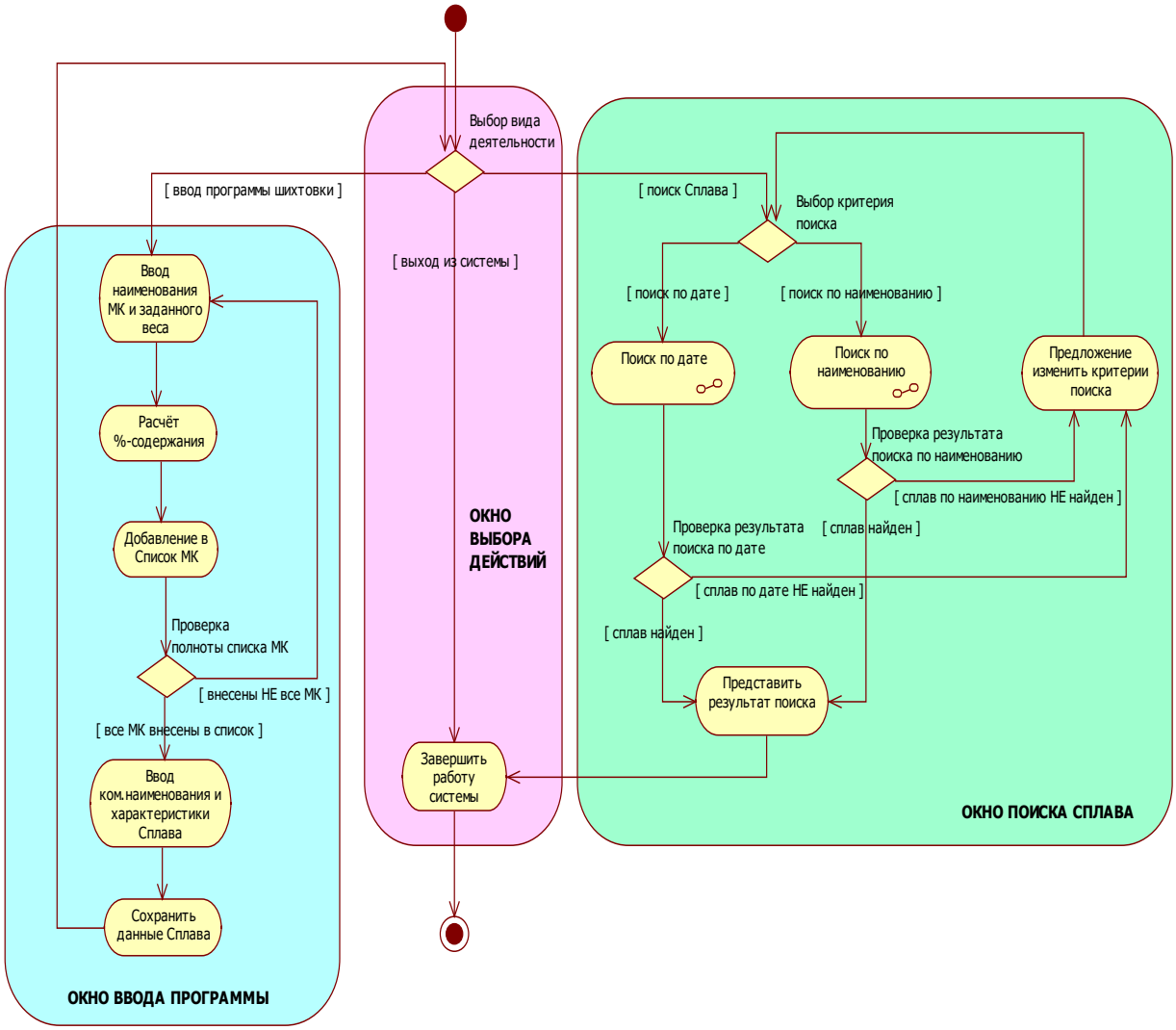


Рис. 16. Деятельность технолога в проекте «Шихтовка МК»

3. Моделирование пространственного размещения системы (*deployment diagram*)

Современные программные системы часто являются многопользовательскими с распределенными в пространстве модулями. Поэтому в представлении *Deployment View* строятся одноименные диаграммы, показывающие пространственное распределение разрабатываемой программной системы. Часто эти модели называют *бизнес-архитектурой системы*.

Диаграмма развертывания (*Deployment Diagram*) в языке UML используется для визуализации физической архитектуры системы, описывая, как компоненты программной системы размещаются на узлах (например, серверах или компьютерах) и как эти узлы соединены между собой.

Пример моделирования бизнес-архитектуры на UML в виде *Deployment Diagram Rational Rose* представлен на рис. 17.

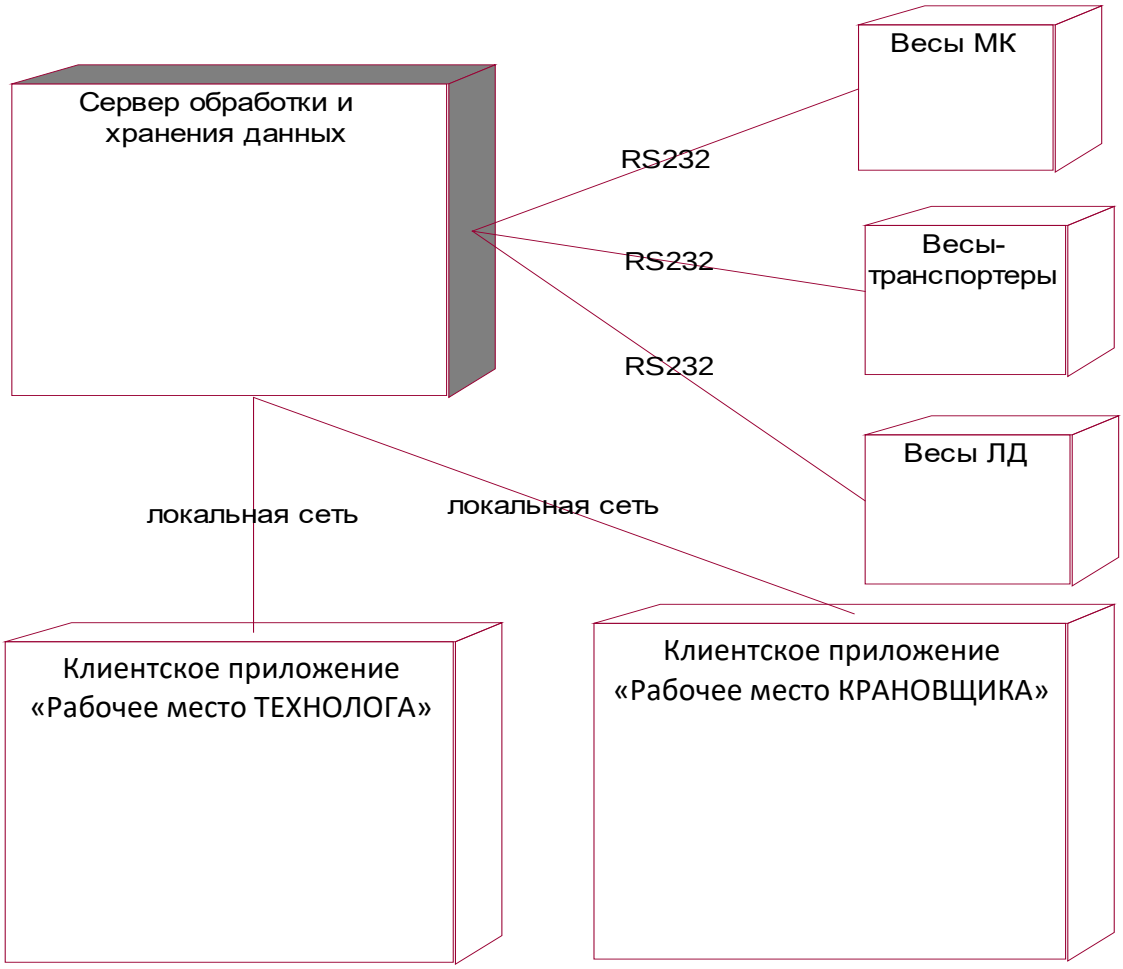


Рис. 17. Пространственное размещение модулей проекта «Шихтовка»

Задание для лабораторной работы

Используя описание концепции системы (приложение 1), CASE-средство и ранее выполненные модели:

– в представлении *Use Case View* постройте диаграмму бизнес-субъектов и их функциональных обязанностей (*Business Use Case Diagram*). В окнах *Documentation* приведите текстовые описания для бизнес-субъектов и их обязанностей;

– постройте диаграмму деятельности (*Activity Diagram*) для бизнес-прецедента, связанного с исполнением процесса шихтовки;

– постройте в StarUML диаграмму пространственного размещения модулей системы, аналогичную представленной на рис. 17.

2.5. Этап 2. Выявление и анализ требований

Данный этап направлен на выявление и документирование функциональных требований. На этом этапе происходит взаимодействие с заказчиком или представителями бизнеса для полного и точного выявления их потребностей.

В результате выполнения лабораторных работ по выявлению и моделированию функциональных требований будут созданы следующие артефакты.

1. Создана трассировочная таблица, в которой пользовательские требования (*UR*) детализированы до уровня функциональных требований (*FR*), относящихся к бизнес-логике работы системы.
2. Для пользовательских требований, описывающих взаимодействие актеров-физлиц с системой, разработаны вайрфремы.
3. Построены диаграммы вариантов использования (*Use Case Diagram*), отражающие как общую концепцию системы, так и детализированные сервисы, предоставляемые системой пользователям-физлицам.
4. По диаграммам вариантов использования произведена оценка полноты и достаточности выявленных функциональных требований.

Лабораторная работа № 3 «Выявление и документирование функциональных требований к ПО»

Целью лабораторной работы является выявление функциональных требований с использованием элементов графического интерфейса пользователя (*wireframes*), составление таблицы трассировки и подготовка документа спецификации требований (*Software Requirements Specification*, SRS).

Теоретическая часть

Анализ требований *– это процесс сбора требований к программному обеспечению, их систематизации, документирования, анализа, выявления противоречий, неполноты, разрешения конфликтов в процессе разработки ПО. В англоязычной среде также говорят о дисциплине «инженерия требований»* (*Requirements Engineering*) [13].

Определения и отличия функциональных и нефункциональных требований, пользовательских (*User Requirements*) и функциональных требований (*Functional Requiremens*) приведены К. Вигерсом в [2]. Пользовательские требования связаны с обязанностями самих пользователей. Они могут извлекаться из разных источников при помощи различных аналитических методик: опросы пользователей и заинтересованных лиц (интервью, анкеты); обсуждения и семинары; наблюдение на рабочих местах; анализ сопутствующих текстовых документов (концепции, правила, стандарты, описания, прайс-листы и т. д.).

Вариант использования *(Use Case, прецедент),* согласно Г. Бучу [1], *– внешняя спецификация последовательности действий, которые система может выполнять в процессе взаимодействия с актерами (пользователями) для получения определенного значимого для них результата*.

Понятия *функция системы* и *прецедент* (*Use Case*) разные, однако если результат исполнения функциональности или функционального сервиса и результат выполнения варианта использования совпадают, то каждая выявленная функция или сервис системы будут соответствовать конкретному прецеденту в модели вариантов использования.

Пользовательские требования, функциональные требования и варианты использования фиксируются в ***трассировочной таблице*** или *трассировочной матрице*.

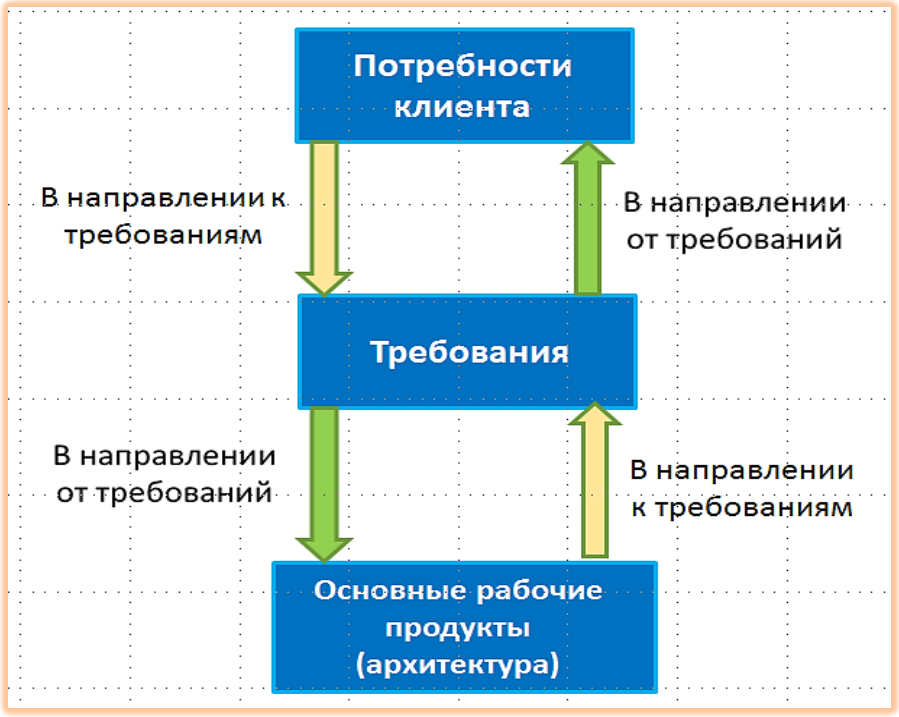


Рис. 18. Прямая и обратная трассировки

Цель трассировки – отображение проектных связей, которые затем используются для проверки проекта и управления изменениями. Обычно трассировка позволяет проследить непрерывную связь от пользовательских требований до структурных элементов разрабатываемой архитектуры.

В соответствии с рекомендациями К. Вигерса [2] важны как прямая, так и обратная трассировки (рис. 18). Трассировочную таблицу можно создать в текстовом редакторе, использовать специально предназначенную для этого программную оболочку (например, *Rational Requisite Pro*) или специализированное средство разработчика (например, *3SL Cradle* [12]), позволяющее сохранять и отслеживать трассируемость проектных элементов.

Пример трассировочной таблицы приведен на рис. 19.

Traceability Matrix

Nr.	Customer Requirement	User Story	Priority	Attribute	Realization			Model
					Functional Requirement	Use Case	Scenario	

Рис. 19. Пример трассировочной таблицы

Процесс выполнения лабораторной работы

Для выявления функциональных требований (*FR*) существует множество методик, некоторые из них подробно описаны в известной книге Карла Вигерса [2]. В большинстве этих методик, как правило, сначала выявляются более общие пользовательские требования (*UR*), представляющие собой функциональные сервисы, которые система выполняет во взаимодействии с пользователем, далее выявленные пользовательские требования детализируются до отдельных функций системы, отвечающих за ее бизнес-логику.

Каждое пользовательское требование (*UR*) можно представить в виде процесса взаимодействия пользователя (*Actor*) и системы через соответствующий интерфейс. В случае если пользователь – физическое лицо, то взаимодействие будет осуществляться через графический интерфейс (*GUI*), т. е. посредством элементов экранных форм. Пользователь будет вносить данные в определенные поля, нажимать кнопки, выбирать элементы списков, инициируя таким образом отдельные функции системы, связанные с бизнес-логикой. Таким образом, представляя взаимодействие пользователя с элементами GUI, выявляют конкретные функции бизнес-логики. Элементы графического интерфейса моделируются с помощью *вайрфрейма* (wireframe) – низкодетализированного изображения элементов GUI, участвующих во взаимодействии с пользователем при выполнении конкретного функционального сервиса (*UR*).

Пользовательские требования (*UR*) могут быть представлены в разных формах (идея, описание, сценарий, документ и т. д.), степенях абстрактности и детализации. В разрабатываемом учебном проекте их следует выявить, скопировать из документа концепции (*Vision*) и зафиксировать в упрощенном варианте трассировочной таблицы в столбце «*Пожелания заказчика*» (рис. 20).

Далее, представляя соответствующие вайрфреймы, необходимо детализировать *UR* до уровня отдельных функций системы, обеспечивающих ее бизнес-логику. Функции бизнес-логики (*FR*) размещаются в столбце «*Функциональное требование*». К одному пользовательскому требованию может относиться не менее одной конкретной функции системы (включая скрытые). Если к некоторому

пожеланию заказчика невозможно отнести одно или несколько конкретных функций системы, то такое пожелание называется *абстрактным*, и оно исключается из дальнейшего анализа.

ТАБЛИЦА ТРАССИРОВКИ: NEEDS – FUNCTINAL REQUIREMENTS – USE CASES

Шаблон

№	Пожелания заказчика (Stakeholder Needs, User Requirements)	№ ф-ции	Функциональное требование (Features, Functional Requirements)	Use Cases

Рис. 20. Упрощенная форма таблицы трассировки

Представленная выше трассировочная таблица (рис. 20) является упрощенным промежуточным документом, так как в нем содержатся только те функции, которые упоминаются в концепции (*vision*). После построения и анализа модели вариантов использования добавятся новые функции бизнес-логики, о которых авторы концепции не упоминали.

Следует обратить внимание, что столбец таблицы «*Функциональное требование*» является исходным артефактом для подготовки стандартной «*Спецификации требований к программной системе*» (*Software Requirements Specification, SRS*), в окончательный вариант которой войдут не только функции, связанные с бизнес-логикой работы системы, но и функции *UX/UI* (*User Experience / User Interface*), а также нефункциональные требования.

Задание для лабораторной работы

В текстовом редакторе MS WORD создайте упрощенную трассировочную таблицу (см. рис. 20). Файл с таблицей присоедините к папке *Use Case Model* представления *Use Case View*.

Далее, читая описание концепции (приложение 1) от начала и до конца текста, выполните следующие действия:

– выделите в тексте концепции пожелания заказчика, относящиеся к функциям бизнес-логики, скопируйте и зафиксируйте их в соответствующем столбце таблицы в качестве пользовательских требований;

– представляя вайрфреймы, поставьте в соответствие каждому пользовательскому требованию одно или несколько функциональных требований (функций системы) и зафиксируйте их в столбце «Функциональное требование»;

– для каждого выявленного пользовательского требования в столбце *Use Case* зафиксируйте название для данного сервиса, сформулированное в глагольной форме.

Лабораторная работа № 4 «Моделирование вариантов использования»

Целью лабораторной работы является выявление ролей бизнес-субъектов, структурирование и построение модели вариантов использования.

Теоретическая часть

Модель вариантов использования (*use case model*) – это модель, которая описывает функциональные требования к системе в терминах вариантов использования.

В отличие от модели бизнес-субъектов и их обязанностей (*Business Use Case Diagram*), построенной в лабораторной работе № 2, модель вариантов использования детально представляет *функциональные сервисы (user requirements)*, которые исполняются программной системой при взаимодействии с внешними пользователями (актерами).

Диаграмма вариантов использования (*use case diagram*) – это диаграмма, на которой изображены отношения, существующие между актерами (*actor*) и вариантами использования (*use case*).

Вариант использования (прецедент, use case) – это последовательность действий, направленных на достижение определенного результата. Эти действия выполняются системой в процессе взаимодействия с пользователями. Совокупность всех вариантов использования полностью определяет поведение системы.

Вариант использования, инициируемый пользователем, называется ***базовым*** и моделирует функциональный сервис, который система предоставляет пользователю. Функциональный сервис объединяет одну или несколько функций программной системы.

Актер (*actor*) (синонимы – актант, действующее лицо) – это абстрактное ролевое понятие, которое характеризует внешнего пользователя (или нескольких пользователей), взаимодействующего с системой. Каждый актер соответствует одной-единственной роли, в которой выступает бизнес-субъект (пользователь) при взаимодействии с системой.

Следует помнить: не каждый бизнес-субъект есть пользователь программной системы, но каждый пользователь есть определенная роль бизнес-субъекта.

Различают **три типа пользователей:** физлицо, внешняя система или модуль и время.

Обобщение (*generalization*) – это отношение, которое связывает специализированный и более общий варианты использования.

Включение (*include use case*) – указывает на включение последовательности поведения варианта использования – поставщика в последовательность взаимодействия варианта использования – клиента.

Расширение (*extend use case*) – позволяет базовому варианту использования при определенном условии использовать функциональность варианта использования – клиента.

Рекомендации для построения модели вариантов использования:

− каждый вариант использования должен быть инициирован актером или вступать в отношения «*include*», «*extend*» с другими вариантами использования;

− фокусировка вариантов использования должна быть на «ЧТО», а не на «КАК» система должна выполнять действия;

− следует избегать функциональной декомпозиции, так как она не подходит для моделей вариантов использования [3] (см. рис. 21);

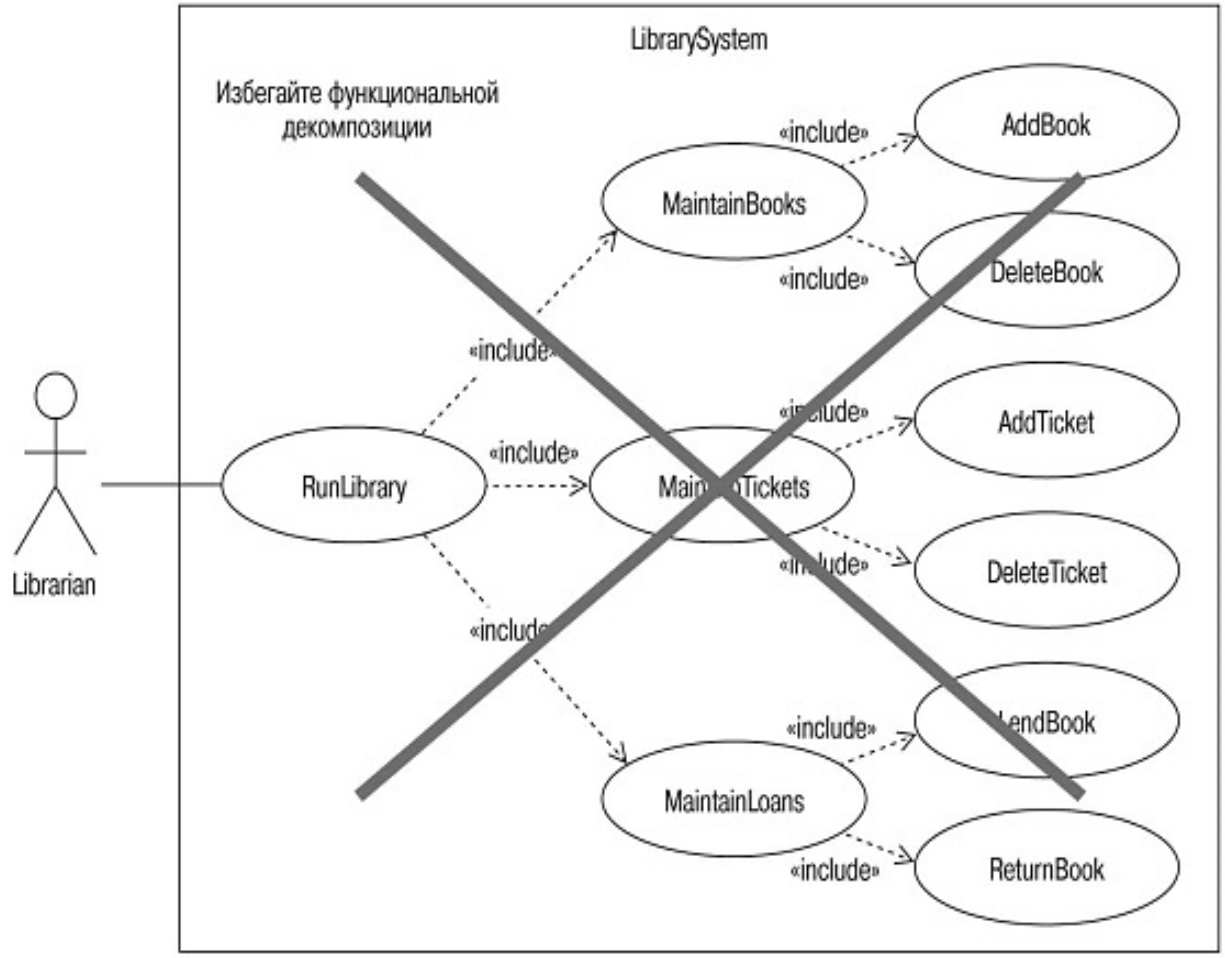


Рис. 21. Функциональная декомпозиция, которую следует избегать [3]

− предпочтительнее создание простой модели вариантов использования без отношений «*include*» и «*extend*»;

− рекомендуется организовывать модели вариантов использования в папках (*Package*), а не стараться показать все элементы на одной диаграмме;

− для каждой роли физического лица рекомендуется создавать отдельную папку (диаграмму) и в ней отображать функциональные сервисы, инициируемые этим актером;

− общие функциональные сервисы, предоставляемые нескольким актерам, лучше отображать отдельно.

Структура модели вариантов использования. Диаграммы вариантов использования должны служить не только средством визуализации базовых сервисов программной системы, но и эффективным инструментом выявления и управления функциональными требованиями, обеспечивающими бизнес-логику приложения. В идеале уровень детализации вариантов использования должен быть достаточным для отображения отдельных функций бизнес-логики программной системы. Однако достичь такого уровня детализации на одной диаграмме практически невозможно. Это может привести к перенасыщенности и нечитабельности диаграммы или, наоборот, созданию слишком абстрактной и малоинформативной модели при моделировании только базовых сервисов.

Поэтому рекомендуется создавать организационную структуру, в которой (в различных пакетах папках) представлены как более общие, так и детальные диаграммы для актеров – физических лиц. Пример подобной структуры для учебного проекта «*Шихтовка МК*» (см. приложение 1) представлен на рис. 22.

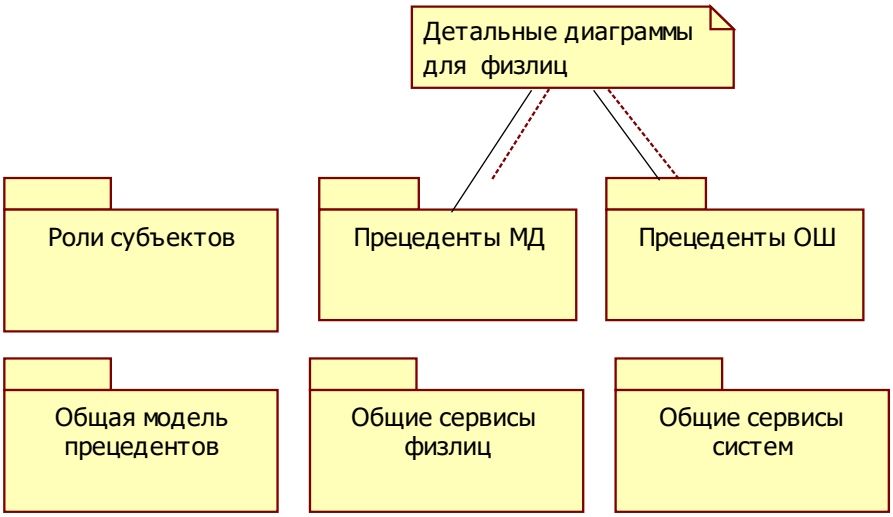


Рис. 22. Пример рекомендуемой организационной структуры модели вариантов использования

В папке *Роли субъектов* представляют пользователей системы (*actors*), являющихся ролями соответствующих бизнес-субъектов.

В папке *Общая модель прецедентов* представляют основных пользователей системы, базовые функциональные сервисы и границы системы, характеризующие объем разработки (*scope*). Пример общей (контекстной) диаграммы вариантов использования представлен на рис. 23.

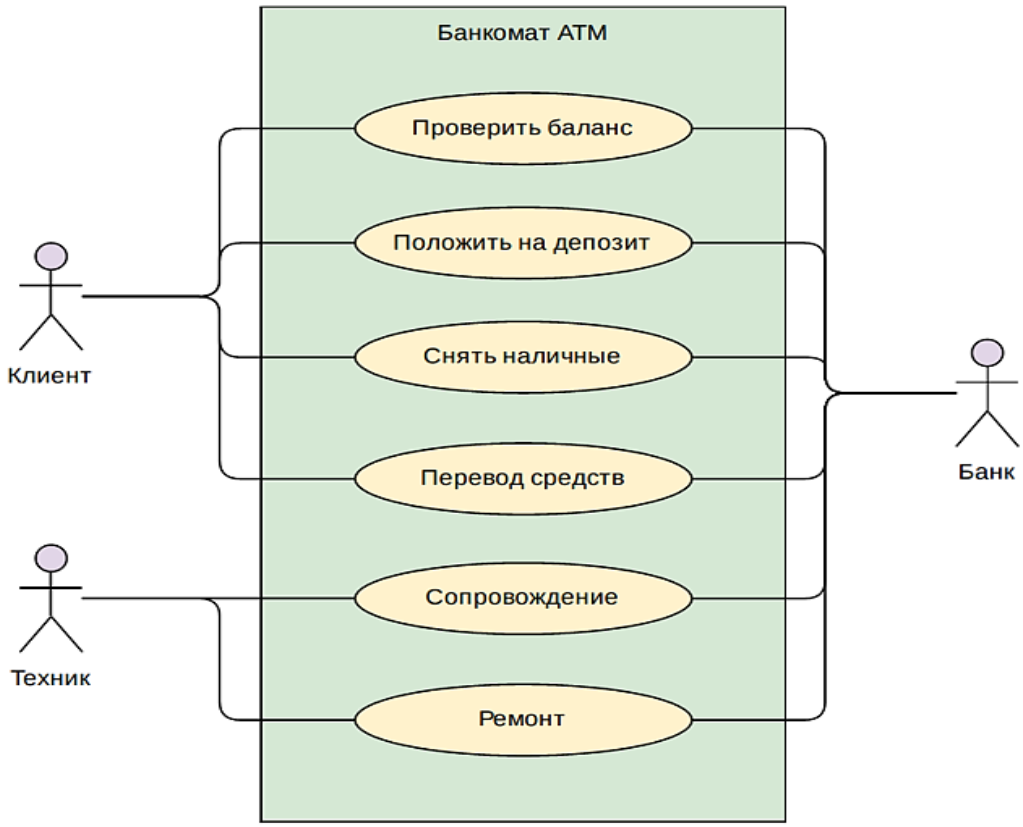


Рис. 23. Пример контекстной диаграммы вариантов использования [11]

В папках *Общие сервисы физлиц* и *Общие сервисы систем* показывают функциональные сервисы, которые программная система предоставляет сразу нескольким актерам – физическим лицам или внешним системам (рис. 24 и 25).

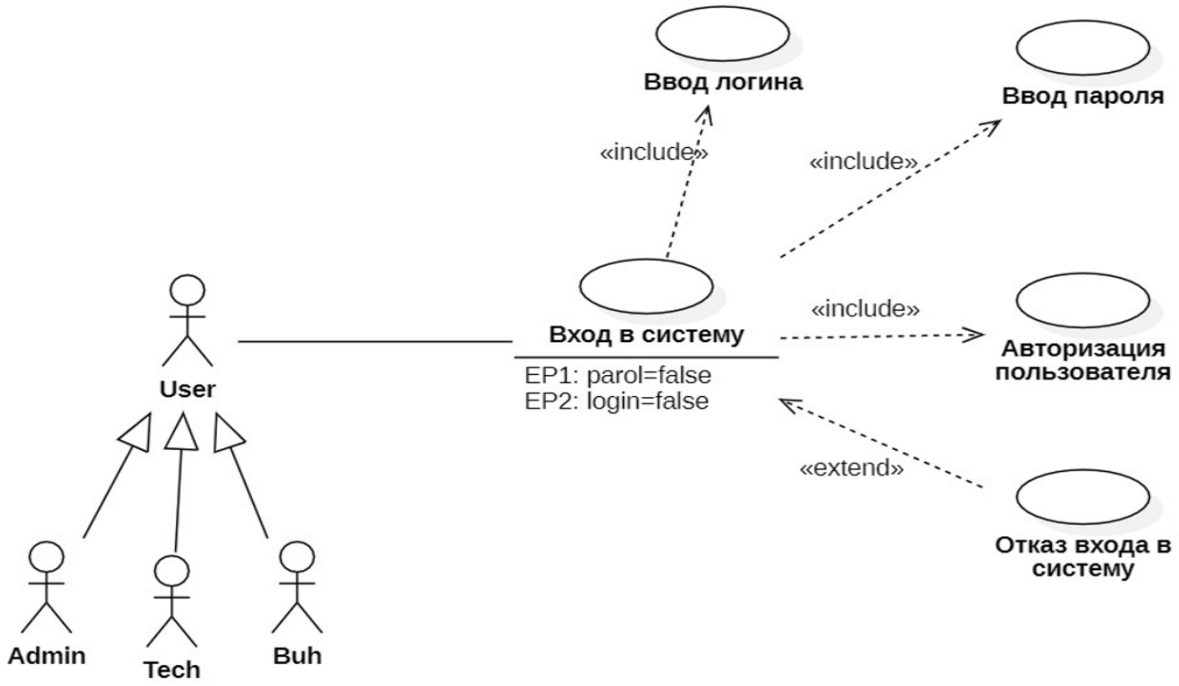


Рис. 24. Пример актеров, использующих общий функциональный сервис системы

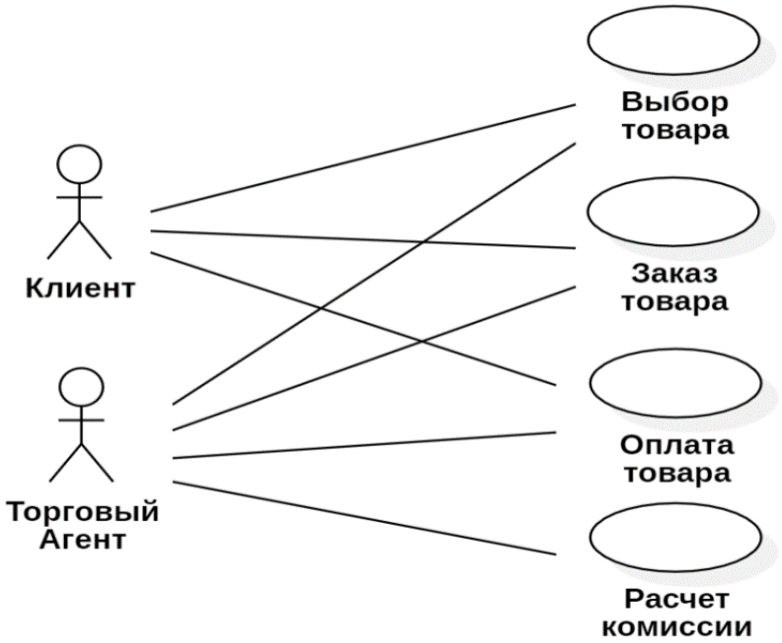


Рис. 25. Пример актеров, пользующихся общими вариантами использования

В папках детального представления функциональных сервисов, которые система предоставляет актерам – физическим лицам (в данном проекте это папки *Прецеденты МД* и *Прецеденты ОШ*) содержатся базовые варианты использования, также включенные («*include*») и расширяющие («*extend*») – более мелкие прецеденты, моделирующие отдельные функции программной системы. Пример диаграммы представлен на рис. 26.

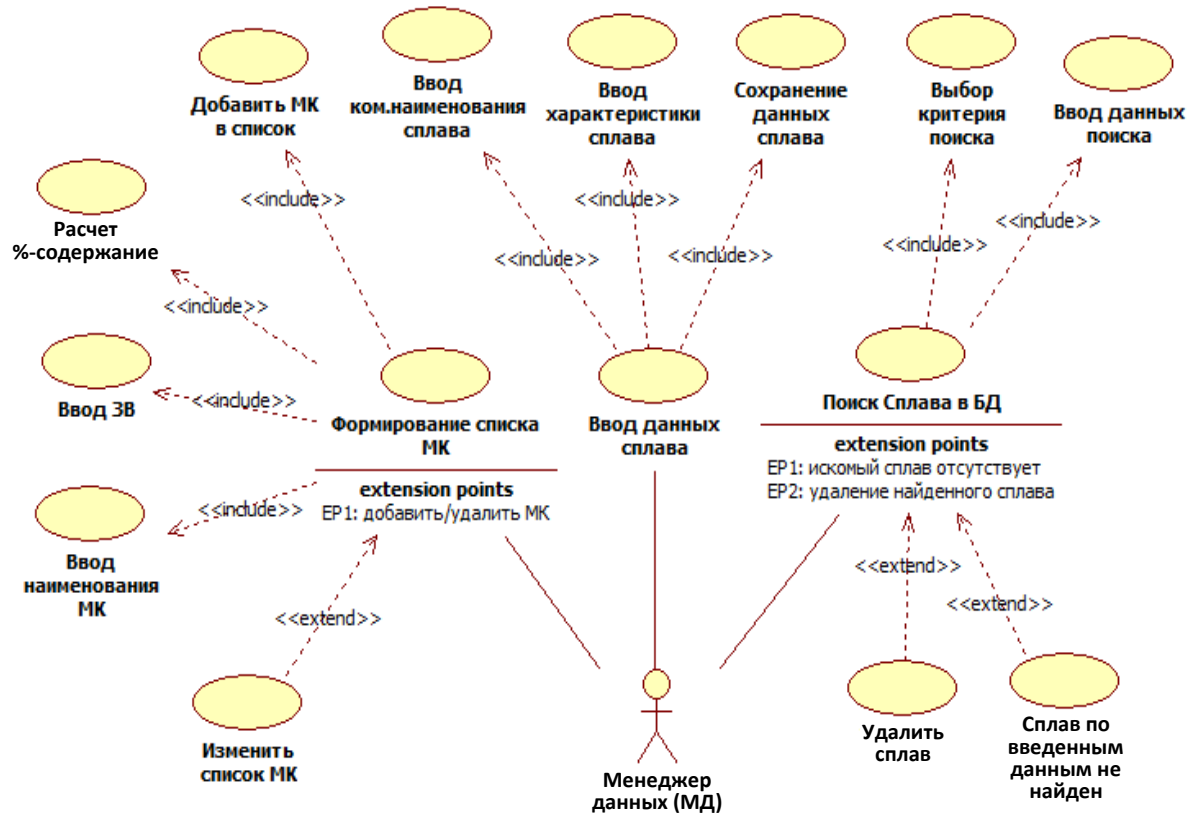


Рис. 26. Пример детализированной диаграммы прецедентов для роли технолога в проекте «Шихтовка МК»

По полученным детальным диаграммам можно судить о полноте, достаточности и непротиворечивости функционала, который система предоставляет пользователям. Функции системы, представленные на этих диаграммах, могут стать основой для *Спецификации требований к программной системе* (*Software Requirements Specification, SRS*).

Процесс выполнения лабораторной работы

1. В браузере проекта (*Model Explorer*) выделяется папка *Use Case Model*, используя пункты меню *Model – Add Diagram – Use Case Diagram*, создается организационная диаграмма представления *Use Case Model*, в окне *Properties* вносится ее наименование.
2. С помощью инструмента *Toolbox – Package* создается структура, представленная на рис. 22.
3. В каждой папке организационной структуры модели прецедентов создаются диаграммы вариантов использования и вносятся их названия в позиции *Name* окон *Properties*, с помощью инструмента *Toolbox – Text* названия помещаются на диаграммах.
4. Определяются роли пользователей и размещаются с помощью инструмента *Toolbox – Actor* на диаграмме в папке *Роли Субъектов* (*Actors*), дополняются соответствующими описаниями в окнах *Documentation*.

5. В папке *Прецеденты МД* с помощью инструментов *Toolbox* строится диаграмма прецедентов, аналогичная представленной на рис. 26 для актера, являющегося ролевым представлением бизнес-субъекта *Технолог*.

6. В папке *Модель прецедентов ОШ* с помощью инструментов *Toolbox* строится диаграмма прецедентов для актера, являющегося ролевым представлением бизнес-субъекта *Крановщик*.

7. В папке *Общие сервисы физлиц* с помощью инструментов *Toolbox* строится диаграмма, аналогичная представленной на рис. 24.

8. В папке *Общие сервисы систем* с помощью инструментов *Toolbox* строится диаграмма (пример на рис. 25), моделирующая общие сервисы, предоставляемые системой актерам, являющимся ролевыми представлениями внешних (не входящих в границы разработки) систем или модулей.

Задание для лабораторной работы

Используя трассировочную таблицу, модель бизнес-процессов и диаграмму бизнес-субъектов, содержащиеся в предыдущих лабораторных работах, выполните следующее:

– определите и проанализируйте набор сервисов, предоставляемый программной системой, а также определите пользователей (актеров), которые будут их инициировать;

– распределите сервисы и актеров по папкам (модулям). Выявите общие варианты использования и постройте для них диаграммы в отдельных папках;

– конкретизируйте содержание базовых вариантов использования с помощью связанных с ними отношений *include* и *extend*.

2.6. Этап 3. Системный анализ. Моделирование объектов и классов

На этапе системного анализа выполняется моделирование объектов программной системы, включая объекты-интерфейсы и объекты-менеджеры, показываются последовательности их взаимодействий и структуры связей, создается начальная структура классов (классы системного анализа) с соответствующими атрибутами, операциями и отношениями.

Лабораторная работа № 5 «Создание сценария для поиска объектов программной системы и моделирование элементов GUI»

Цель лабораторной работы заключается в разработке сценария, направленного на выявление объектов программной системы.

В результате выполненного системного анализа разработчиками ПО создаются следующие артефакты:

1) подробные сценарии взаимодействия пользователей-физлиц с программной системой через GUI, соответствующие разработанным ранее вайрфреймам;

2) объектные диаграммы, отражающие последовательность исполнения методов, участвующих в потоке вариантов использования объектов (алгоритмы функционирования приложения) и структур связей между объектами с учётом направления обмена сообщений между ними;

3) диаграмма классов системного анализа с отношениями между ними (ассоциации, агрегации, композиции) с разделением ответственности между классами благодаря назначению соответствующих стереотипов (*entity*, *boundary*, *control*).

Теоретическая часть

Сценарий – это текстовое описание потоков действий (событий) при выполнении конкретного варианта использования, выражающее некий аспект поведения системы.

Сценарии служат для перехода от вариантов использования к объектам программной системы. Анализируются имена-существительные в тексте сценария. Некоторые из них будут действующими лицами, другие – объектами, а третьи – атрибутами объекта.

Существует двухэтапный подход к написанию сценариев. На первом этапе сценарий пишут в произвольной форме с высоким уровнем абстракции. Целью такого сценария будет не выявление объектов, а поверхностное описание для общего понятия выполняемого функционала в данном варианте использования. Такие сценарии часто называются *User Story*.

На втором этапе сценарии уточняют и подробно описывают в стандартной форме с указанием начальных и конечных условий, точек ветвления и привязывают к разработанному для данного сценария (или для группы сценариев) вайрфрейму или прототипу графического интерфейса пользователя (GUI).

Прототипы GUI строятся на основе use-case-моделей, они могут динамически отображать функции бизнес-логики и по возможности содержат основные кнопки, поля и элементы меню. Для построения действующих прототипов GUI используются специализированные программные средства, например, *Figma* или *Axure RP Pro* [14].

Для подробного описания сценария существует специальный шаблон. Каждый сценарий начинается с главного раздела (табл. 2), далее описываются типовой (табл. 3) и альтернативные потоки, для актеров физических лиц должны быть предусмотрены ошибочные потоки событий.

Каждый поток варианта использования заканчивается своим ожидаемым результатом. Типовой (основной) поток заканчивается желаемым для пользователя результатом.

В один сценарий могут быть включены описания несколько вариантов использования, выполняющихся последовательно друг за другом. Это следует отобразить на диаграмме ВИ в виде примечания (*Note*) или стереотипом (*Use Case Realization*). Сценарии с прототипами экранных форм (особенно с динамическими) являются первой действующей моделью разрабатываемой программы, которую можно представить для согласования Заказчику.

Пример сценария для варианта использования «*Ввод программы шихтовки*», соответствующий вайрфрейму, представленному на рис. 27.

Таблица 2. Главный раздел варианта использования «Ввод программы шихтовки»

Наименование варианта использования	**Ввод программы шихтовки**
Актеры	Администратор данных (технолог или АД)
Цель исполнения	Ввод программы шихтовки для оператора шихтовки (крановщика)
Краткое описание	Технолог в начале рабочего дня вводит список МК, по каждому МК вводится заданный вес, по которому рассчитывается %-содержание, задает атрибуты сплава и сохраняет программу для оператора шихтовки
Тип	Базовый
Начальные условия (precondition)	На мониторе отображается окно «*Ввода программы шихтовки*» (см. рис. 28)

Таблица 3. Основной поток управления варианта использования «Ввод программы шихтовки»

Действия актеров	**Отклик системы**
1. АД вводит наименование металлокомпонента (МК) в поле «*Наименование МК*» (см. поле 2, рис. 27). 2. Учитывая усредненный объем контейнера (поле 1), АД вводит значение заданного веса в поле «*Заданный вес*» (поле 3). 3. АД инициирует добавление МК в список МК (кнопка 4).	4. Система рассчитывает % – содержание МК в сплаве. 5. Наименование МК, заданный вес, %-содержание МК помещаются в строку «*Списка МК*» (поле 5). 6. Поля 2 и 3 очищаются.
Примечание: действия 1–4 повторяются для каждого МК.	
7. АД вводит наименование сплава в поле «*Коммерческое наименование сплава*» (поле 6). 8. АД вводит текстовую характеристику сплава в поле «*Характеристика сплава*» (поле 7). 9. АД инициирует сохранение введенных характеристик сплава (программу шихтовки) (кнопка 8).	10. Система переходит в окно «Программа шихтовки», предлагает внести изменения в программу шихтовки или подтвердить сохранение параметров сплава **Extend point 1: Изменения параметров сплава**
Действия актеров	**Отклик системы**
11. АД подтверждает сохранение параметров сплава.	12. Система сохраняет данные: коммерческое наименование сплава, характеристику сплава, МК, входящие в список с наименованием, заданным весом и %-содержанием. 13. Система переходит в окно «*Меню технолога*»
Конечные условия (post conditions)	В долговременной памяти сохранен сплав с введенными значениями атрибутов и списком включенных в него МК. На мониторе АД показывается «*Меню технолога*»

Процесс выполнения лабораторной работы

1. С использованием шаблона для описания сценария выполнения варианта использования (см. табл. 2 и 3) и построенной ранее диаграммы прецедентов ОШ разрабатывается несколько сценариев для основных базовых вариантов использования.

2. На диаграмме прецедентов к каждому описанному в сценарии варианту использования с помощью окна *Attachments* вводится соответствующая ссылка на подготовленный документ.

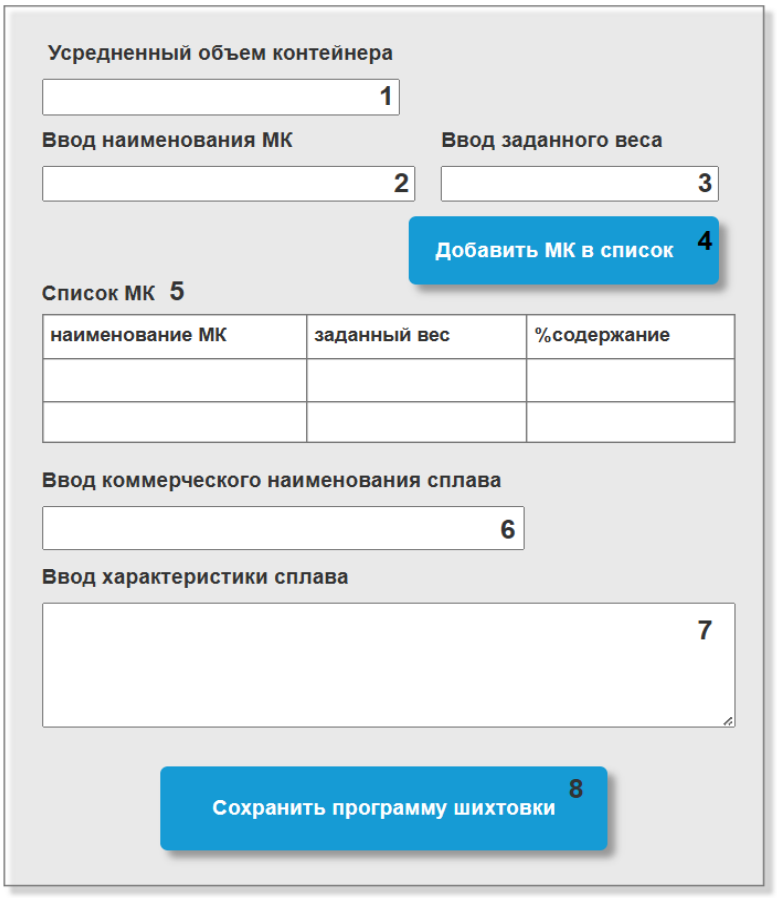


Рис. 27. Вайрфрейм для варианта использования
«Ввод программы шихтовки»

Задание для лабораторной работы

Применяя созданную ранее модель вариантов использования:

– в графическом редакторе постройте прототип графического интерфейса пользователя для двух выбранных вариантов использования, аналогично примеру на рис. 27;

– с использованием стандартных форм, изображенных в табл. 2 и 3, а также макета GUI, проведите подробное описание всех потоков вариантов использования. В описании укажите ссылки на поля и клавиши на макете GUI.

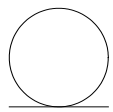
Лабораторная работа № 6 «Моделирование объектов и классов приложения»

Целью лабораторной работы является освоение методики построения объектной модели приложения в соответствии со сценарием варианта использования, а также создание диаграммы классов системного анализа на основе полученной объектной модели.

Теоретическая часть

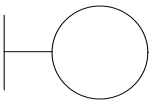
Классы программной системы представляют собой абстракции или шаблоны, описывающие общие характеристики и поведение объектов в системе. Классы определяют атрибуты (свойства) и методы (действия), которые объекты данного класса могут иметь.

Классы программной системы в системном анализе принято относить к трем видам, каждый из которых имеет определенное предназначение и обозначается своим стереотипом (пиктограммой):

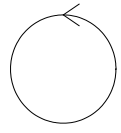


– **классы сущности** (*entity*) – описывают объекты программной системы, отвечающие за целостность и хранение данных (значений атрибутов). Создаются на основе сущностей предметной области (*Business Entity*).

Объекты классов сущностей не могут самостоятельно инициировать взаимодействия. Все присущие им операции связаны с их атрибутами и относятся к следующим видам: ***модификатор*** (операция, изменяющая состояние объекта), ***селектор*** (операция, имеющая доступ к состоянию объекта, но не изменяющая его), ***итератор*** (операция, обеспечивающая доступ ко всем частям объекта в строго определенном порядке), ***конструктор*** (операция, создающая объект и/или инициализирующая его состояние) и ***деструктор*** (операция, стирающая состояние объекта и/или уничтожающая сам объект);



– **граничные классы** (*boundary*) – описывают объекты программной системы, которые являются интерфейсами связи с внешними пользователями (актерами). В случае если пользователь физическое лицо, то с помощью объектов типа *boundary* моделируются окна GUI;



– **классы управления** (*control*) – описывают объекты программной системы, которые отвечают за выполнение методов (управляют взаимодействиями, реализуют математические расчеты, задают последовательности и т. д.). К этому типу классов относится главный класс приложения *Main*.

Диаграмма последовательности (S*equence Diagram*) описывает временную последовательность обмена сообщениями между объектами программной системы в одном из потоков событий варианта использования.

Это объектная диаграмма, отражающая динамику взаимодействия объектов программной системы. Следует заметить, что, для того чтобы назначить стереотип объекту, его необходимо ассоциировать с классом. Сообщения (*Object*

Message) на этапе системного анализа могут быть написаны обычным текстом, а на этапе проектирования нужно их ассоциировать с соответствующими операциями класса. Таким образом, диаграмма последовательности отражает временную последовательность вызова методов объектов программной системы.

Как правило, диаграмма последовательности строится в два этапа: на первом этапе в упрощенном виде без объектов класса *Main* и без ассоциирования сообщений с операциями соответствующих классов; на втором этапе все сообщения должны быть ассоциированы с операциями, а после объектов типа *boundary* вводятся объекты типа *control*, ассоциированные с классом *Main*, при этом должно выполняться правило *Boundary-Control-Entity* (*BCE*, рис. 28).

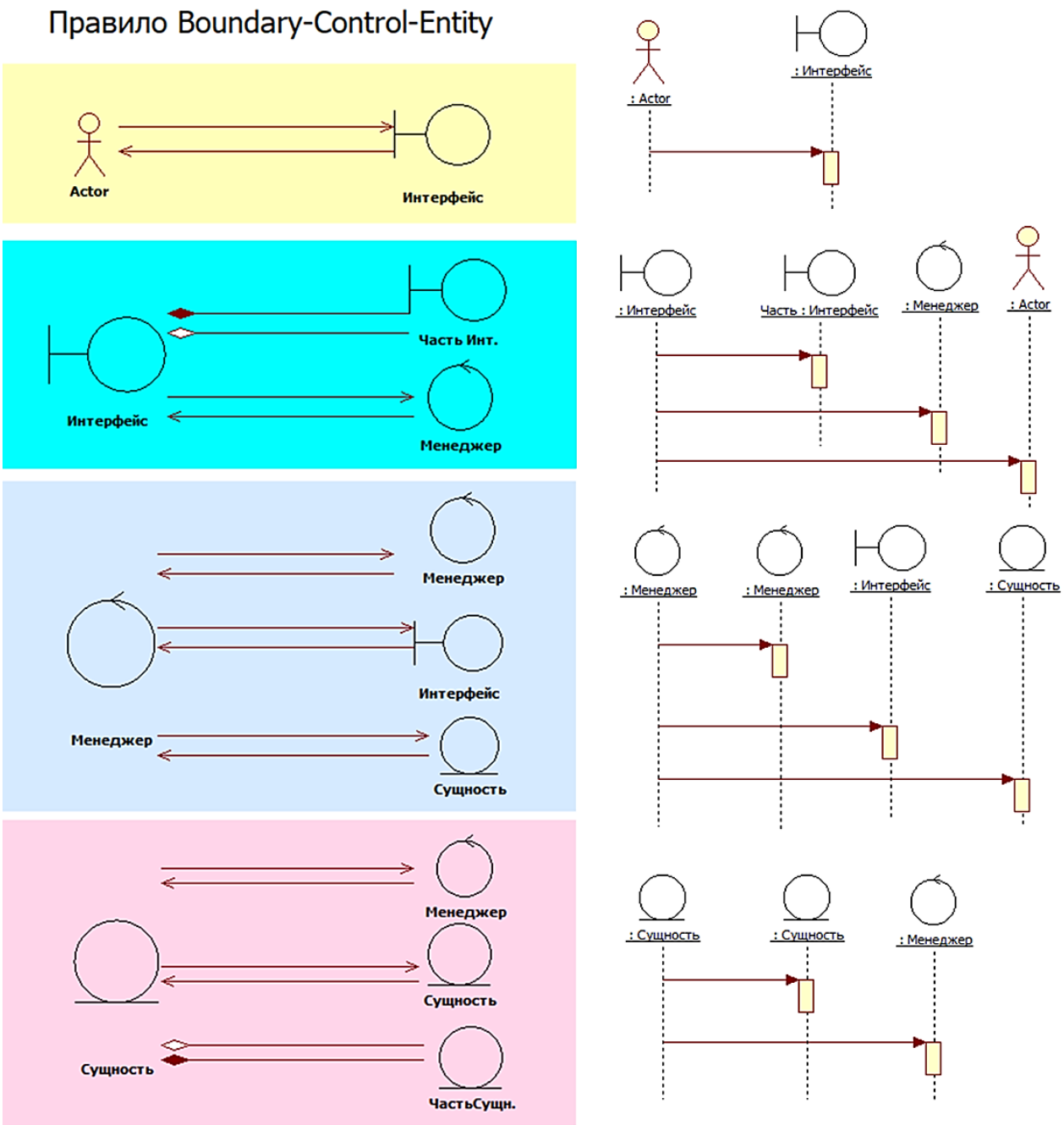


Рис. 28. Правило BCE [3]

Для каждого потока варианта использования строится отдельная диаграмма последовательности.

Пример диаграммы последовательности (упрощенный вариант с объектом-менеджером класса *Main*) для основного потока варианта использования «*Ввод программы шихтовки*» (табл. 2 и 3), выполненной в CASE-средстве *Rational Rose*, представлен на рис. 29.

Следует заметить, что каждое сообщение (*Message*) на данной диаграмме – это вызов метода объекта, который является экземпляром *«тривиальной»* операции класса. Далее проектировщики группируют тривиальные операции и окончательно описывают операции классов.

Таким образом, диаграмма последовательности отправки сообщений между объектами программной системы (*Sequence Diagram*), по сути, показывает последовательность исполнения операций классов (алгоритм работы системы) в рамках определенного потока вариантов использования.

Кооперативная диаграмма (*Collaboration Diagram / Communication Diagram*) показывает структуру взаимосвязей между объектами программной системы в одном из потоков вариантов использования. Она позволяет определить по связям между объектами (*Object Link*) и по направлению сообщений (*Message*) как сами ассоциации (*Association*) между классами системы, так и их направление.

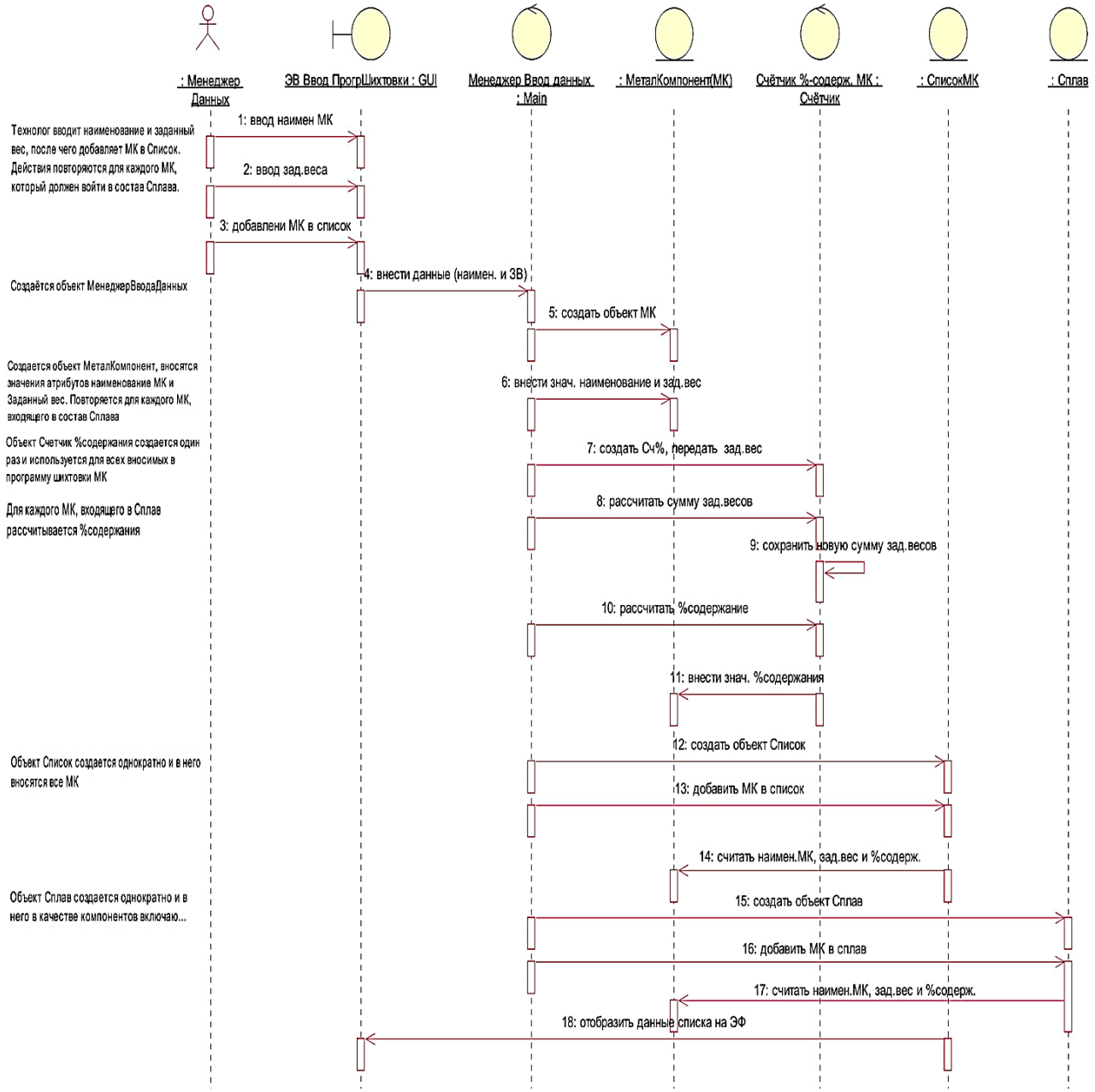


Рис. 29. Диаграмма последовательности фрагмента основного потока управления ВИ «Ввод программы шихтовки»

Для каждого потока варианта использования строится отдельная диаграмма кооперации. Пример диаграммы кооперации, выполненной в *Rational Rose*, представлен на рис. 30.

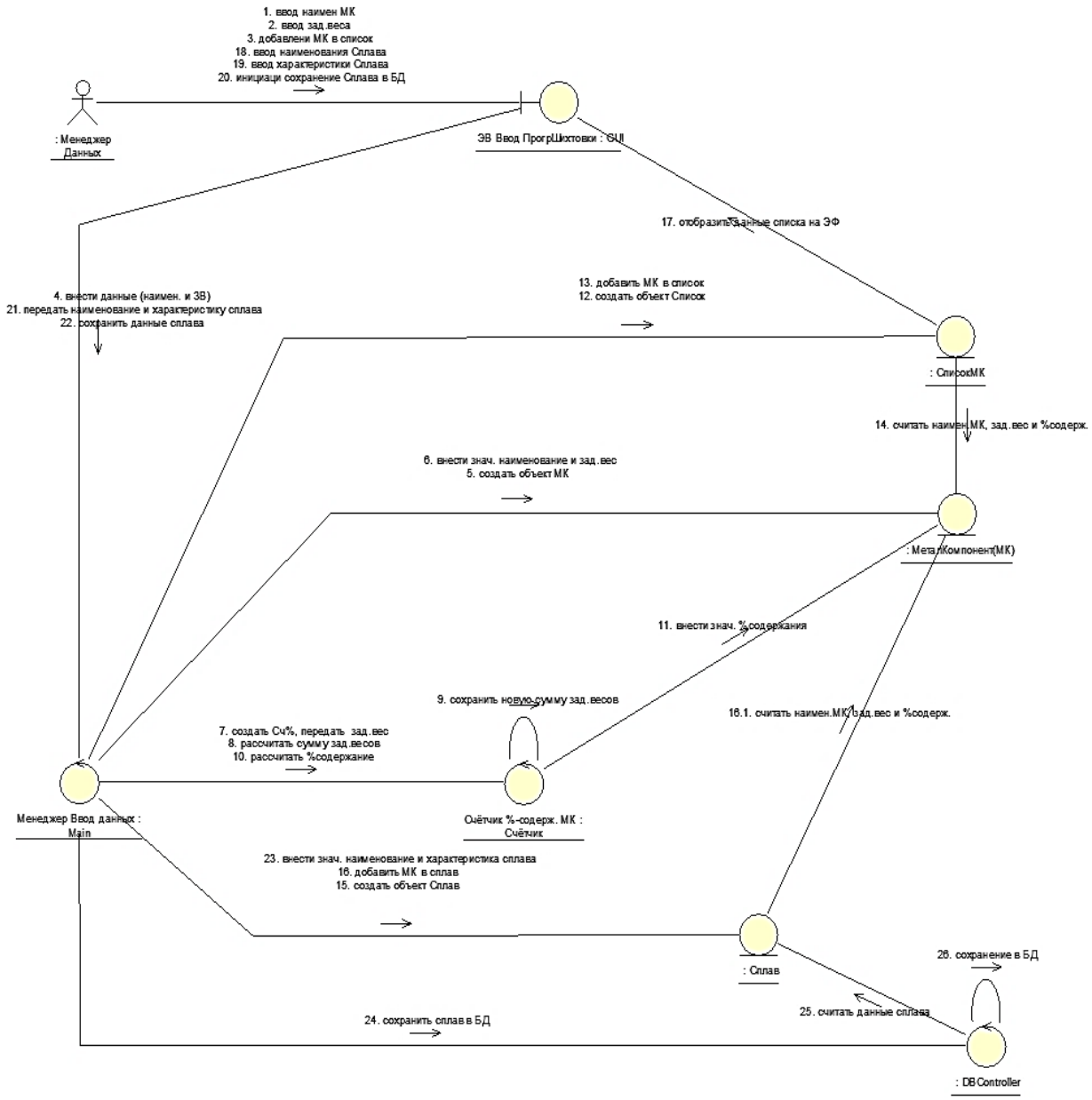


Рис. 30. Диаграмма кооперации основного потока управления ВИ «Ввод программы шихтовки»

Диаграмма классов (C*lass Diagram*) – это графическое представление статической модели, в которой отображены классы, их типы, содержимое и отношения.

На этапе системного анализа показываются не все типы отношений между классами, а только ассоциации и их более сильные типы: агрегации и композиции. Отношения обобщения (*Generalization* или *«наследование»*), зависимости (*Dependency*), а также интерфейсы (*Interface*) и отношения реализации (*Realization*) вносятся на диаграмму классов проектировщиками.

Диаграмма классов строится на основе анализа всех кооперативных диаграмм проекта, поэтому пример на рис. 31 показывает отдельный фрагмент диаграммы классов, созданный на основании одной диаграммы кооперации (см. рис. 30). Данная диаграмма построена с использованием CASE-средства *StarUML*.

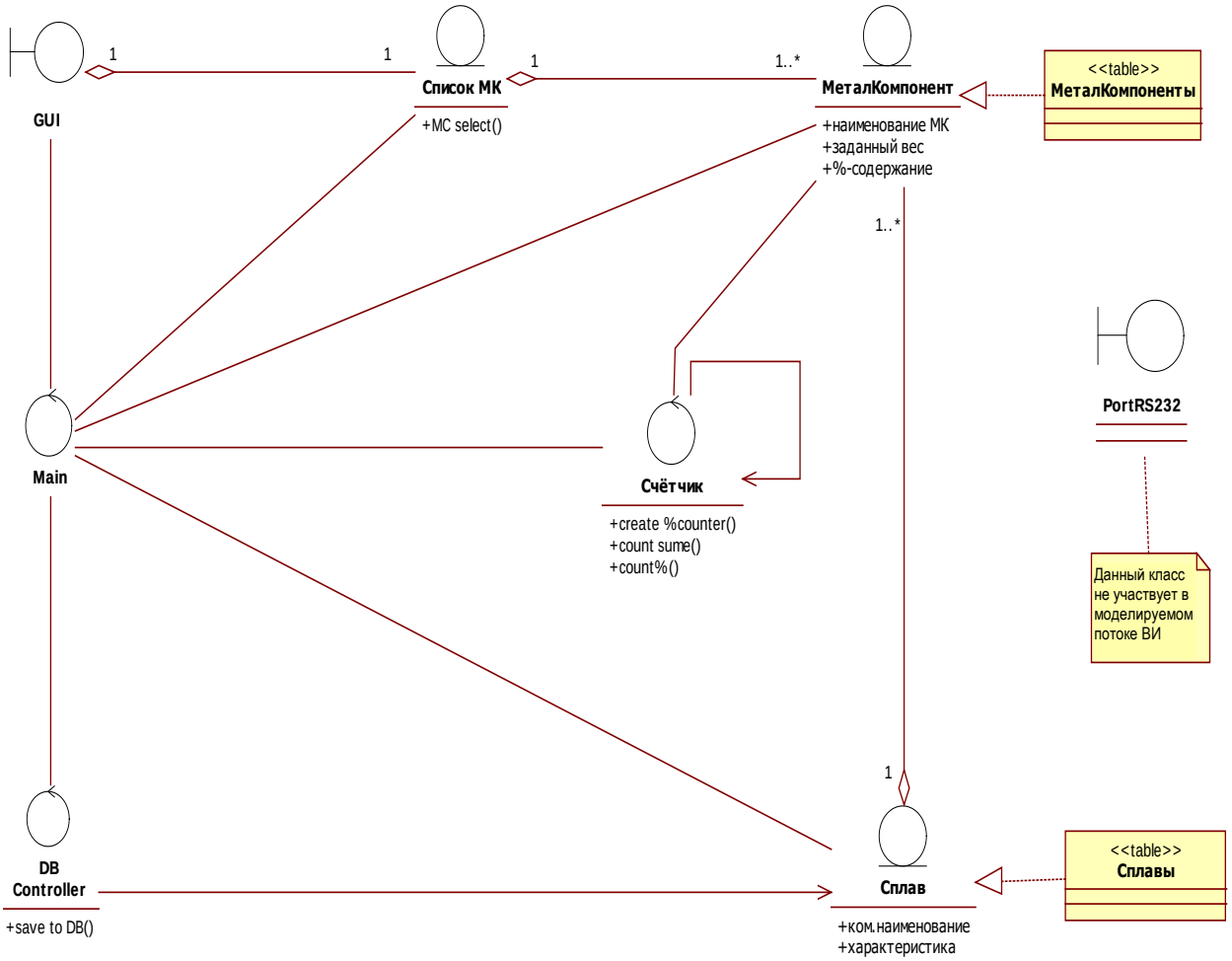


Рис. 31. Структура классов участников основного потока управления ВИ «Ввод программы шихтовки»

Процесс выполнения лабораторной работы

1. На диаграмме вариантов использования выбирается вариант использования, потоки которого должны быть отображены в диаграмме последовательности. Далее через пункты меню *Model – Add Diagram – Sequence Diagram* к выбранному варианту использования подсоединяется диаграмма последовательности. В пункте меню *Name* отражается название потока варианта использования (рис. 32).

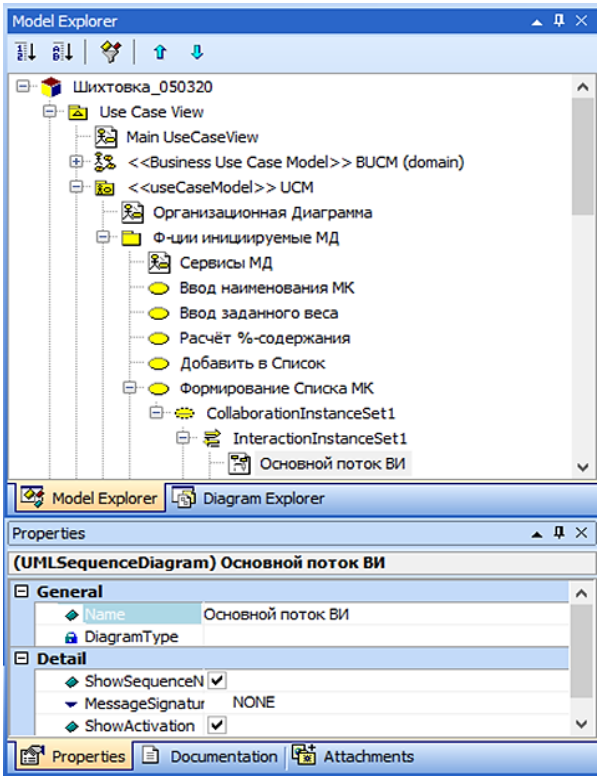


Рис. 32. Присоединение диаграммы последовательности

2. Методом перетягивания (*drag-and-drop*) из браузера на диаграмму помещается экземпляр инициирующего актера.

3. При чтении сценария выявляются взаимодействующие объекты, в *Model Explorer* они представлены в *Logical View* на диаграмме бизнес-сущностей. Далее у бизнес-сущностей изменяется стереотип (с *Business Entity* на *Entity*), и они перетягиваются в папку *Application Classes* и выставляются на диаграмме классов анализа. Далее перетягиванием созданного класса на диаграмму последовательности на ней создаются объекты и показываются сообщения между ними (см. рис. 29).

4. После окончательной доработки диаграммы последовательностей в *StarUML* выбираются пункты меню: *Model – Convert Diagram – Convert Sequence (Role) to Collaboration (Role)* и автоматически генерируется диаграмма кооперации (*Collaboration Diagram*). Растягивая объекты по площади диаграммы, следует ее привести к удобному для восприятия виду (см. рис. 30).

5. После построения диаграммы кооперации на диаграмме классов анализа в папке *Application Classes* показываются отношения ассоциаций между классами, объекты которых взаимодействуют в рамках рассматриваемого потока вариантов использования (см. рис. 31).

6. К классам приложения, объекты которых отвечают за данные, предназначенные для долговременного хранения, например в БД, с помощью отношения *Realization* добавляются элементы со стереотипом «*tabl*» (таблица), чтобы далее их использовать при моделировании логической структуры баз данных с помощью диаграмм «*Сущность – Связь*» (*Entity Relationship Diagram*).

ПРИЛОЖЕНИЕ 1

Шихтовка металлолома

Задача относится к области автоматизации производственных процессов. На небольших предприятиях, занимающихся утилизацией металлического лома, все необходимые компоненты для плавки собираются вручную. Инструментом для этого служит кран, который забирает металлические компоненты (МК) своим магнитом и потом опускает в контейнер. В соответствии с весом набранных МК подаются кокс, известь, спецкокс. Содержимое контейнера со всеми занесенными в него металлическими компонентами направляется в чан, в который добавляются легирующие добавки. В конце рабочего дня чан отправляется в плавильную печь. Так как в процессе плавления должны использоваться определенные соотношения веса различных МК, программа должна обеспечить требуемый состав плавильной смеси. Этот контроль и составление смеси компонентов для плавки называется шихтовкой.

Программная система устанавливается на рабочем месте крановщика и используется им для контроля над процессом шихтовки. Кроме того, она позволяет технологу задать определенную пропорцию МК в конечном сплаве и осуществлять сохранение информации о составе и свойствах полученных сплавов в базе данных.

Технологический процесс шихтовки

Ход процесса	Описание
Начало рабочего дня	Перед началом процесса шихтовки технолог, исходя из усредненного объема дозировочного контейнера, вводит в программу заданный вес по каждому МК, предназначенному для плавки, определяя таким образом их пропорцию в сплаве. Кроме того, он вводит коммерческое название и характеристику сплава. Такая программа плавки задается на каждый рабочий день
Взвешивание МК и передача данных	При шихтовке используется магнитный подъемный кран, который под управлением крановщика будет забирать желаемые МК (стружку металлическую, железо кровельное и др.). Для того чтобы сократить путь загрузки, отсортированные по наименованию МК помещаются вблизи контейнера. Вес поднятого краном МК определяется автоматически с помощью тензодатчика и вводится через последовательный порт (RS232) в компьютер. При разработке компьютерной программы «Шихтовка» необходимо применять стандартные протоколы и установки последовательных портов (см. документацию)

Ход процесса	Описание
Добавка кокса и извести	Кокс и известь будут при шихтовке подмешиваться автоматически и поступать с отдельных независимых весов-транспортеров. Вес кокса и извести обуславливается суммарным весом набранных в контейнер МК, который посылается на весы-транспортеры (протокол RS232). Кокс и известь подаются в контейнер смешанными, однако в протоколе их веса представлены по отдельности. Оператору представляет интерес суммарный вес кокса и извести. Как правило, по окончании набора контейнера крановщик посылает соответствующий запрос на весы-транспортеры (протокол RS232), сигнализирующий об окончании загрузки контейнера (посылка команды «дозировать кокс и известь»), после чего ему должна быть подана смесь кокса и извести, соответствующая набранному в нем суммарному весу МК, а система должна индицировать статус весов-транспортеров «загружены». Каждый раз по получении смеси кокса и извести крановщик должен послать команду разгрузки на весы-транспортеры, после чего смесь помещается в контейнер, а статус весов изменится на «разгружены»
Добавка спецкокса	Спецкокс взвешивается и подается отдельно от кокса и извести, но теми же весами-транспортерами. Его вес, аналогично весам кокса и извести, обуславливается суммарным весом МК в контейнере. Ход процесса подачи спецкокса полностью аналогичен описанному выше для кокса и извести
Подача легирующих добавок FeSi, FeMn, FeCr	Эти компоненты будут вноситься в чан от установки легирования и показываться крановщику только в качестве информации без контроля с его стороны. Вес каждой легирующей добавки отображается тремя разрядами
Текущий цикл шихтовки плавильной смеси	Металлический лом, утиль и другие МК с помощью крана загружаются в дозировочный контейнер. После его заполнения металлом и добавками горения содержимое контейнера пересыпается в чан для плавления, в котором должно быть обеспечено их соотношение. Это производится следующим образом. В программной системе на рабочем месте крановщика по каждому МК должен показываться заданный и остаточный веса, а также вес МК, поднятого краном.

Ход процесса	Описание
	Крановщик, выделив из списка наименование поднятого краном МК, должен нажать клавишу «Разгрузка». Вес, который был в этот момент на кране, будет вычтен из величины заданного веса. Таким образом определится остаточный вес для данного МК. Только после этого кран может быть освобожден с помощью стандартных элементов управления, а МК помещен в контейнер. По полученным остаточным весам крановщик определяет, сколько и каких МК ему необходимо добавить в контейнер для обеспечения заданной технологом пропорции. Если контейнер оказался заполнен, то оставшиеся остаточные веса должны быть учтены в следующем цикле шихтовки. При этом может так случиться, что возникнет отрицательная величина остаточного веса, если вес набранных МК превысит заданный. Этот «отрицательный» остаток веса МК должен быть вычтен из заданного веса в следующем цикле шихтовки. Если останется положительный остаток, то он должен добавиться к следующему плановому весу. Процесс должен быть обеспечен таким образом, чтобы остаточные веса учитывались в каждом последующем цикле шихтовки
Завершение цикла шихтовки	Если МК и добавки горения (известь, кокс, спецкокс) были взвешены и помещены в шихтовой контейнер, который оказался заполненным, то можно завершать текущий цикл шихтовки, содержимое контейнера пересыпать в чан и, если рабочий день не закончился, переходить к следующему циклу. Перед тем как завершить текущий цикл, должны быть выполнены следующие проверки: – если кокс и известь еще не поданы, то процесс не может быть завершен; – весы по взвешиванию кокса и извести должны иметь статус «опорожнены»
Функция Storno – отмена последнего зафиксированного веса МК	В программе на рабочем месте крановщика должна быть предусмотрена возможность удаления последнего зафиксированного веса МК (шаг назад, Storno). Выполняется, если МК при разгрузке крана не попал в контейнер
Корректура пропорции	Для технолога в программе должна быть предусмотрена возможность изменения введенного заданного веса

Ход процесса	Описание
Сохранение данных	Программа должна сохранять в базе данных (БД) наименование сплава, дату выплавки, вес сплава (без учета легирующих добавок), список составляющих МК, их фактическое процентное содержание и характеристику сплава. БД доступна на рабочем месте технолога. Программа обеспечивает поиск по дате выплавки и по коммерческому наименованию сплава. Сохранение данных в базе происходит при нажатии клавиши «Сохранить» на рабочем месте химика-технолога и при нажатии клавиши «Конец рабочего дня» на рабочем месте крановщика
Визуализация данных в программе	В программе должны выводиться на экран следующие данные. 1. Рабочее место крановщика: – наименование сплава; – текущая дата (дата выплавки); – список МК с планируемыми и остаточными весами; – МК, набираемый в настоящий момент, должен высвечиваться другим цветом; – вес МК на кране; – суммарный вес МК в контейнере; – вес сплава (общий суммарный вес набранных МК); – статус весов-транспортеров; – количество кокса, извести, спецкокса, суммарный вес кокса и извести; – количество FeCr, FeMn, FeSi 2. Рабочее место технолога при вводе характеристик сплава: – наименование сплава; – список МК, вошедших в состав сплава; – заданный вес по каждому МК; – процентное содержание каждого МК в сплаве; – характеристика сплава. 3. Рабочее место технолога при работе с БД: – поле даты выплавки; – поле ввода наименования сплава; – характеристика сплава; – вес сплава; – список МК, входящих в сплав; – фактическое процентное содержание каждого МК

Ход процесса	Описание
Организация пользовательских рабочих мест	Программная система распределена по двум рабочим местам: рабочее место технолога и рабочее место крановщика, объединенным в локальную сеть (*LAN*). Рабочее место технолога представляет собой windows-приложение, установленное на desktop-компьютере, включающее БД, размещенную на его жестком диске. Рабочее место крановщика организуется на базе промышленного портативного компьютера с сенсорным экраном (*TouchPC*) размером *800×600 dpi*, функционирующим под управлением Windows 3.10. Внешние устройства подключаются через последовательные порты (*RS232*). При разработке графического интерфейса не использовать выпадающие меню

ПРИЛОЖЕНИЕ 2

Задания для самостоятельной работы

Варианты предметной области	Исходные данные
1. ПОЛИКЛИНИКА	Поликлиника оказывает платные услуги. Необходимо вести список врачей, их оказывающих, список услуг, списки клиентов. При обращении клиенту формируется листок посещения с датой, врачами и перечнем услуг. После посещения врача перечень услуг может быть изменен. После посещения выполняется окончательный расчет стоимости заказа
2. ШКОЛА	В школе требуется вести учет успеваемости учащихся (выполнение планов) и хранить их анкетные данные. Ученики состоят в классах. Первичными документами являются учебные планы. Возможны зачисления учеников в класс, окончание школы и переводы между классами. Требуется формировать отчеты об успеваемости и приложения к аттестату
3. ДЕКАНАТ	Деканату вуза требуется вести учет успеваемости студентов (выполнение планов) и хранить их анкетные данные. Студенты обучаются на нескольких специальностях и состоят в группах. Первичными документами являются учебные планы специальности. Возможны зачисления студентов в группу, отчисления, восстановления и переводы. Требуется оформлять отчеты об успеваемости, академические справки и приложения к диплому
4. УЧЕБНЫЙ ОТДЕЛ	Учебному отделу деканата вуза требуется вести планирование и учет выполнения почасовой нагрузки преподавателями. Первичными документами являются учебные планы специальности и распределение нагрузки. Выполнение нагрузки и расчеты могут производиться за любой период, который выбирается преподавателем при оформлении акта. Требуется формировать договоры, акты выполнения нагрузки за период, отчеты о выполненной нагрузке, остатках и т. д.
5. ОТДЕЛ КАДРОВ	Отдел кадров ведет списки принятых и уволенных работников, а также списки пенсионеров и работников, состоящих на воинском учете. Надо иметь возможность просматривать списки работников всего предприятия и по подразделениям, осуществлять различные выборки с итоговыми расчетами по количественному и качественному составу работников. Требуется также распечатывать эти списки и выборки в виде отчетов

Варианты предметной области	Исходные данные
6. ОСНОВНЫЕ СРЕДСТВА	Необходимо вести учет движения основных средств (отражение поступления, выбытия, перемещения) внутри счета, осуществлять контроль за наличием и сохранностью в местах эксплуатации. Предусмотреть начисление амортизационных отчислений. Необходимо предусмотреть, что учет основных средств ведется по однородным группам, в которых выделяют натурально-вещественные и которые подразделяются по принадлежности, а также в разрезе отдельных инвентарных номеров объектов
7. РЕКЛАМНОЕ АГЕНТСТВО	Рекламное агентство собирает заявки от рекламодателей и публикует их в печатных изданиях (газетах, журналах). При этом требуется вести списки печатных изданий с их расценками на рекламу, списки рекламодателей, заявок. Заявка от рекламодателя может содержать публикацию в несколько печатных изданий и на различные даты выхода. Обеспечить оперативный просмотр списка заявок (печатные издания, рекламодатель, стоимость) на любую вводимую дату
8. АГЕНТСТВО ПО ТРУДОУСТРОЙСТВУ	Агентство по трудоустройству ведет списки лиц, ищущих работу, и списки вакансий, поступающих от организаций, с указанием должности, зарплаты и количества единиц. В заявках претендентов, кроме анкетных данных, указываются желаемые должности и зарплата. Каждая вакансия заполняется несколькими претендентами, согласно их анкетным данным, распечатывается на бумагу и передается работодателю. Работодатель независимо от агентства отбирает одного из претендентов (или исключает всех), который и должен занять вакансию в базе данных агентства, после чего вакансия и претендент «аннулируются»
9. АВТОМАГАЗИН	Фирма по продаже автомобилей производит их доукомплектование по желанию покупателя. При этом требуется вести учет заказов с перечнем дополнительно устанавливаемых деталей, расчет общей суммы, печать заказа и суммы продаж за определенный период времени
10. АВТОСЕРВИС	Для СТО нужно формировать заказ на выполнение работ, используя прейскурант цен. Нужно предусмотреть хранение заказов, их группировку по месяцам, по слесарям и т. п. для получения данных об объемах выполненных работ за определенный период. Выводить информацию для расчета заработной платы в зависимости от выполненных заказов

ЗАКЛЮЧЕНИЕ

В заключение следует отметить достоинства и недостатки, а также границы применения метода визуального моделирования на языке UML.

Прежде всего описанная в настоящем пособии методика, включая визуальное моделирование на UML, предназначена для разработки сложных многопользовательских, разнесенных в пространстве объектно-ориентированных программных систем, основными пользователями которых являются физические лица, взаимодействующие с системой через сложные насыщенные графические интерфейсы. Особенно эффективно подобная методика применяется в комплексе с динамическим прототипированием GUI для разработки «пионерских» систем (не имеющих аналогов) в специфических областях применения и с невыясненными функциональными требованиями.

К основным недостаткам описанной методики следует отнести ее сложность, высокую требовательность к уровню компетенций исполнителей, продолжительное время разработки, необходимость использования специализированных инструментов разработки (CASE-средств) и коммуникаций между участниками проекта. Поэтому ее не следует использовать при разработке простейших интернет-ресурсов – лендингов (от англ. *Landing Page* – «посадочная страница»), блогов и т. д.

Метод визуального моделирования на UML применим при разработке программных систем, но не может заменить специализированные нотации для проектирования архитектуры самого предприятия (бизнес-системы). Для проектирования бизнес-систем и оптимизации бизнес-процессов следует использовать такие нотации, как *ARIS, Business Studio, IRIS Business Architect, BPMN* и др.

Визуальное моделирование на UML и связанные с ним CASE-средства можно ограниченно использовать при моделировании алгоритмических программных систем (*Activity Diagram*) и логических структур баз данных (*Entity Relationship Diagram*).

Для ускорения процесса разработки программной системы в рамках объектно-ориентированного подхода допустимо использовать не всю методику в целом, а отдельные ее части, ограничившись, например, анализом требований и примерной логической структурой классов и баз данных без объектного моделирования. В этом случае необходимый объем моделирования в анализе и проектировании устанавливается самими разработчиками.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Объектно-ориентированный анализ и проектирование с примерами приложений / Г. Буч [и др.]. – 3-е изд. – М. : Вильямс, 2017.
2. Вигерс, К. Разработка требований к программному обеспечению / К. Вигерс, Дж. Битти. – М. : БХВ-Петербург, 2019.
3. Арлоу, Дж. UML-2 и Унифицированный процесс. Практический объектно-ориентированный анализ и проектирование / Дж. Арлоу, А. Нейштадт. – 3-е изд. – СПб. : Символ-Плюс, 2017.
4. Рамбо, Дж. UML: специальный справочник / Дж. Рамбо, А. Якобсон, Г. Буч. – СПб. : Питер, 2002. – 656 с. : ил.
5. Unified Modeling Language. Specification [Электронный ресурс]. – Режим доступа: https://www.omg.org/spec/UML. – Дата доступа: 11.03.2025.
6. Википедия [Электронный ресурс]. – Режим доступа: https://ru/wikipedia.org/wiki/Бизнес-логика. – Дата доступа: 11.03.2025.
7. Agile-манифест [Электронный ресурс]. – Режим доступа: https://agile-manifesto.org/iso/ru/manifesto.html. – Дата доступа: 11.03.2025.
8. Киреев, Н. Моделирование функциональности бизнеса и ПО на основе модифицированной структуры уровней требований // VII международная конференция по системному и бизнес-анализу Analyst Days 2017. – М., 2017.
9. StarUML. Руководство пользователя User Guide [Электронный ресурс]. – Режим доступа: https://usermanual.wiki/Document/userguide.1128996068/html. – Дата доступа: 11.03.2025.
10. Source Forge.net [Электронный ресурс]. – Режим доступа: https://source-forge.net/projects/staruml/. – Дата доступа: 11.03.2025.
11. Visual Paradigm [Электронный ресурс]. – Режим доступа: https://online.visual-paradigm.com/ru/diagrams/. – Дата доступа: 11.03.2025.
12. Обзор возможностей 3SL Cradle [Электронный ресурс]. – Режим доступа: http://cradle.saturs.ru/cradle-overview/. – Дата доступа: 11.03.2025.
13. Основы программной инженерии (SWEBOK) [Электронный ресурс]. – Режим доступа: https://github.com/ligurio/swebok-2004-in-russian. – Дата доступа: 11.03.2025.
14. Axure RP [Электронный ресурс]. – Режим доступа: https://www.axure.com. – Дата доступа: 11.03.2025.

Св. план 2024, поз. 40

Учебное издание

Киреев Николай Борисович
Кашникова Инна Васильевна

ТЕХНОЛОГИИ ПРОЕКТИРОВАНИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ИНФОРМАЦИОННЫХ СИСТЕМ. ЛАБОРАТОРНЫЙ ПРАКТИКУМ

УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ

Редактор *Ю. В. Граховская*
Корректор *Е. Н. Батурчик*
Компьютерная правка, оригинал-макет *В. А. Долгая*

Подписано в печать 04.06.2025. Формат 60×84 1/16. Бумага офсетная. Гарнитура «Таймс».
Отпечатано на ризографе. Усл. печ. л. 3,84. Уч.-изд. л. 4,0. Тираж 50 экз. Заказ 3.

Издатель и полиграфическое исполнение: учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники».
Свидетельство о государственной регистрации издателя, изготовителя,
распространителя печатных изданий №1/238 от 24.03.2014,
№2/113 от 07.04.2014, №3/615 от 07.04.2014.
Ул. П. Бровки, 6, 220013, г. Минск