

Документ подписан простой электронной подписью
Информация о владельце:
ФИО: Клочков Юрий Сергеевич
Должность: Ректор
Дата подписания: 02.10.2026 10:31:18
Уникальный программный идентификатор:
3beb265d5d589e7ff4c954946f3ad99a1e70ac12

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«ТЮМЕНСКИЙ ИНДУСТРИАЛЬНЫЙ УНИВЕРСИТЕТ»

С. М. Каратун, И. О. Лозикова

**ПРОЕКТИРОВАНИЕ АВТОМАТИЗИРОВАННЫХ
ИНФОРМАЦИОННЫХ СИСТЕМ**

Учебное пособие

ДОКУМЕНТ ПОДПИСАН
ЭЛЕКТРОННОЙ ПОДПИСЬЮ

Сертификат: 6FA44C50384686A8E7BD5E277351798E
Владелец: Ефремова Вероника Васильевна
Действителен: с 14.06.2022 до 07.09.2023

Тюмень
ТИУ
2021

УДК 681.5(075.8) + 004.415.2(075.8)
ББК 32.965я73
К 215

К 215 Каратун, С. М. Проектирование автоматизированных информационных систем: учебное пособие / С. М. Каратун, И. О. Лозикова. – Тюмень: ТИУ, 2021. – 121 с. – Текст: непосредственный.
ISBN 978-5-9961-2638-5

Учебное пособие по выполнению курсового проекта и лабораторных работ дисциплины «Проектирование автоматизированных информационных систем» для обучающихся направлений подготовки 09.03.01- «Информатика и вычислительная техника», 09.04.01- «Информатика и вычислительная техника» всех форм обучения.

Содержит систематическое изложение основных понятий, на которых строится проектирование автоматизированных информационных систем. Рассмотрены общие вопросы проектирования систем, методологии проектирования, примеры использования методологий проектирования.

УДК 681.5(075.8) + 004.415.2(075.8)
ББК 32.965я73

ISBN 978-5-9961-2638-5

© Федеральное государственное
бюджетное образовательное
учреждение высшего образования
«Тюменский индустриальный
университет», 2021

СОДЕРЖАНИЕ

Введение	4
1 Структура и задачи Автоматизированных информационных систем	5
1.1 Управление и информация	6
1.2 Структура АИС	10
1.3 Пользовательские сценарии АИС.....	11
1.4 Жизненный цикл АИС	16
1.5 Модели жизненного цикла АИС.....	18
1.6 Методология и технология разработки АИС.....	20
2 Методологии проектирования Автоматизированных информационных систем	22
2.1 Понятие методологии проектирования.....	22
2.2 Объектно-ориентированная методология	23
2.2.1 Унифицированный процесс разработки системы.....	24
2.2.2 Модель представлений архитектуры системы	30
2.2.3 Модель требований системы	35
2.2.4 Описание языка моделирования UML	43
2.2.5 Методы объектно-ориентированного анализа	46
2.2.6 Методика анализа и проектирования.....	50
2.3 Структурные методологии	72
2.3.1 Методологии семейства IDEF.....	73
2.3.2 Методологии моделирования бизнес-процессов	75
2.4 Архитектурные шаблоны.....	80
3 Примеры проектирования систем.....	85
3.1 Разработка АРМ регистратуры больницы.....	85
3.2 Проектирование системы «Мобильный сервис вызова на дом врача для поликлиники города»	101
Библиографический список литературы.....	115
Приложение 1 Структура SRS шаблона на основе RUP.....	116
Приложение 2 Структура SRS на основе стандарта ISO/IEC/IEEE 29148:2011	117
Приложение 3. Процесс разработки АИС на основе спецификации [10]...	118
Приложение 4. Основные диаграммы UML.....	119

ВВЕДЕНИЕ

Настоящее учебное пособие содержит систематическое изложение основных понятий, на которых строится проектирование автоматизированных информационных систем (АИС). Рассмотрены общие вопросы проектирования систем, методологии проектирования, примеры использования методологий проектирования.

Весь материал пособия разделен на три главы, где представлены соответственно структура и задачи АИС, методологии проектирования и примеры проектирования систем.

В первой главе изложены основные понятия, которые определяют структуру, задачи и жизненный цикл АИС, а также введены понятия методологии и технологии разработки систем.

Во второй главе представлены методологии проектирования с описанием унифицированного процесса разработки, описаны модель представления архитектуры АИС, модель требований, методы анализа предметной области. Описана методика объектно-ориентированной разработки АИС в нотации UML. Методика представлена так, чтобы применяя ее для проектирования, обучающиеся смогли выполнить лабораторные, курсовые и самостоятельные работы.

В третьей главе представлены два примера проектирования автоматизированных информационных систем в разных методологиях.

Для лучшего понимания и усвоения материала рекомендуется пользоваться дополнительной литературой по общим вопросам и практике автоматизированных информационных систем [1-6], по различным нотациям моделирования и различным аспектам разработки программ [7-13].

1 СТРУКТУРА И ЗАДАЧИ

Автоматизированных информационных систем

Проектирование и разработка Автоматизированных информационных систем (АИС) связано с реализацией различных аспектов информатики, которая изучает вопросы получения информации, хранение, анализ и обработку, а также передачу и использование информации [1]. Как любая наука Информатика состоит из теоретической информатики и прикладной. Теоретическая информатика является математической дисциплиной, включающей теорию множеств, математическую логику, теорию алгоритмов, системный анализ, имитационное моделирование и др. Одним из направлений прикладной информатики является создание Информационных систем, связанных с областью документооборота и анализа экономической и технической информации. В рамках этого направления решаются задачи анализа и прогнозирования потоков информации различной природы с целью их минимизации, стандартизации и эффективной компьютерной обработки. Выбирается способ представления и хранения информации, создание специальных языков для формального описания информации, кодирования, аннотирования и реферирования. Определяются методы извлечения информации из информационных хранилищ.

Основной частью информационной системы (ИС) является хранилище, снабженной процедурами ввода, поиска, хранения и выдачи информации. Работа ИС заключается в обслуживании входного информационного потока и выдаче хранимых, обработанных или проанализированных данных.

В большинстве АИС основным направлением работы является решение задач управления организациями. В этом случае необходимо обеспечивать взаимодействие АИС с организационными структурами управления. АИС позволяет осуществлять принятие необходимых решений управленческому персоналу за счет своевременного получения полной и достоверной информации. Внедрение АИС часто влечет за собой изменение организационных структур и методов управления - реинжиниринг бизнес процессов. С этой целью необходимо осуществить разработку новых моделей документооборота и методов оптимизации и статистической обработки информации.

Обобщенная функциональная структура автоматизированной системы управления организацией приведена на Рис. 1.1. Источник информации выдает сведения о состоянии объекта управления, то есть сведения о работе организации: планах, запасах кадрах и пр. В системе сбора поступающая информация может проходить предварительную обработку. Далее информация поступает в Систему хранения, которая может состоять из нескольких распределенных баз данных (БД).

Система обработки и анализа информации является одной из главных. В ней решаются диспетчерские задачи управления, обслуживание баз

данных, обновление и защита от несанкционированного доступа, выработка управленческих решений, планирование и финансовые задачи и др.

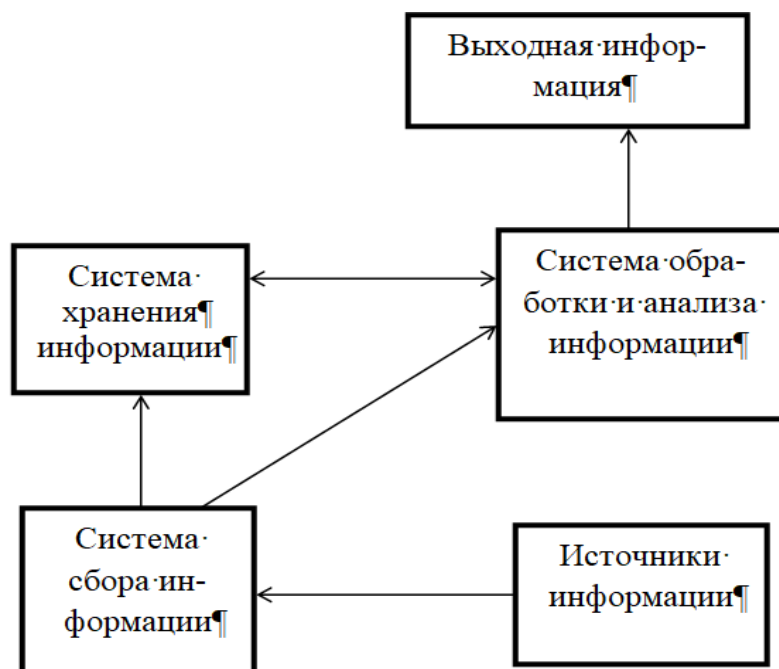


Рис. 1.1 Функциональная структура автоматизированной системы управления

Таким образом, главными задачами АИС по управлению деятельностью организации является получение, хранение, обработка, анализ и передача информации.

1.1 Управление и информация

К числу основных методов формализованного описания информации относятся классификация и кодирование различных видов информации, по которым осуществляется упорядочение, поиск и логическая обработка социально-экономических и технико-экономической показателей.

Информация движется от источника к потребителю, который использует ее для решения задач управления. Если информация не используется в конкретный момент, она должна быть сохранена в БД и при необходимости проанализирована и обработана.

Потоки информации имеют схему потока и элементы потока. Схема потока задается указанием отношений вхождения относительно каждого элемента потока. Элемент потока – это документы, элементы документов (показатели и реквизиты) и участники процесса – сотрудники или клиенты, подразделения или устройства.

Виды управленческой информации представлены на Рис. 1.2.



Рис. 1.2 Классификация управленческой информации

Существуют исходные документы (показатели) – это документы, которые служат источниками информации. Они используются для формирования других документов и отчетов. Производные документы - формируются на основе других документов. Документы могут быть исходными по отношению к одной группе документов и производные к другой. Поэтому документооборот – это процесс перехода от исходных документов к производным путем преобразования исходных показателей.

Первичные документы непосредственно отражают входную информацию, тогда как конечные документы оказывают влияние на объект управления (организацию) и выходные документы.

Потоки информации имеют направление и плотность. Направление потока это граф, который задается местом его входа и выхода в системе управления. Плотность потока определяется объемом потока (в байтах или количестве документов) к длительности передачи, приема или обработки. В потоке информации может быть постоянная составляющая, содержащая информацию, передаваемую непрерывно и составляющие потока, имеющие период суток, декады и др. Переменные составляющие имеют разную интенсивность к концу периода.

На Рис. 1.3 представлена схема выработки управляющих воздействий.



Рис. 1.3 Процесс выработки управляющих воздействий

Процесс управления состоит из сбора текущей информации о состоянии объекта управления и внешней информации, которая затем сравнивается с плановыми показателями, хранится, обрабатывается, анализируется и используется для принятия управленческих решений.

В качестве примера на Рис.1.4 приведены источники и носители информации транспортной организации. Имеются подразделения: автоколонна, зона технического обслуживания и ремонта и служба аппарата управления. В подразделениях работают конкретные сотрудники водители, ремонтные сотрудники и служащие. Они выполняют определенные виды работ по перевозке грузов и пассажиров, проводят техническое обслуживание и ремонт техники и осуществляют обработку документов и работу с внешними организациями. При этом генерируются документы в виде путевого и ремонтного листа.

В общем случае процесс управления при использовании Автоматизированных информационных систем может быть представлен на Рис. 1.5. Производственный процесс генерирует первичную информацию на основе показателей работы организации и внешних воздействий. Первичные данные о ходе производства обрабатываются и сравниваются с плановыми по-

казателями. Результат сравнения используется для выработки управляющих воздействий.

Схема Рис. 1.5 реализует основные этапы процесса управления: планирование, учет, анализ и регулирование процесса деятельности организаций.

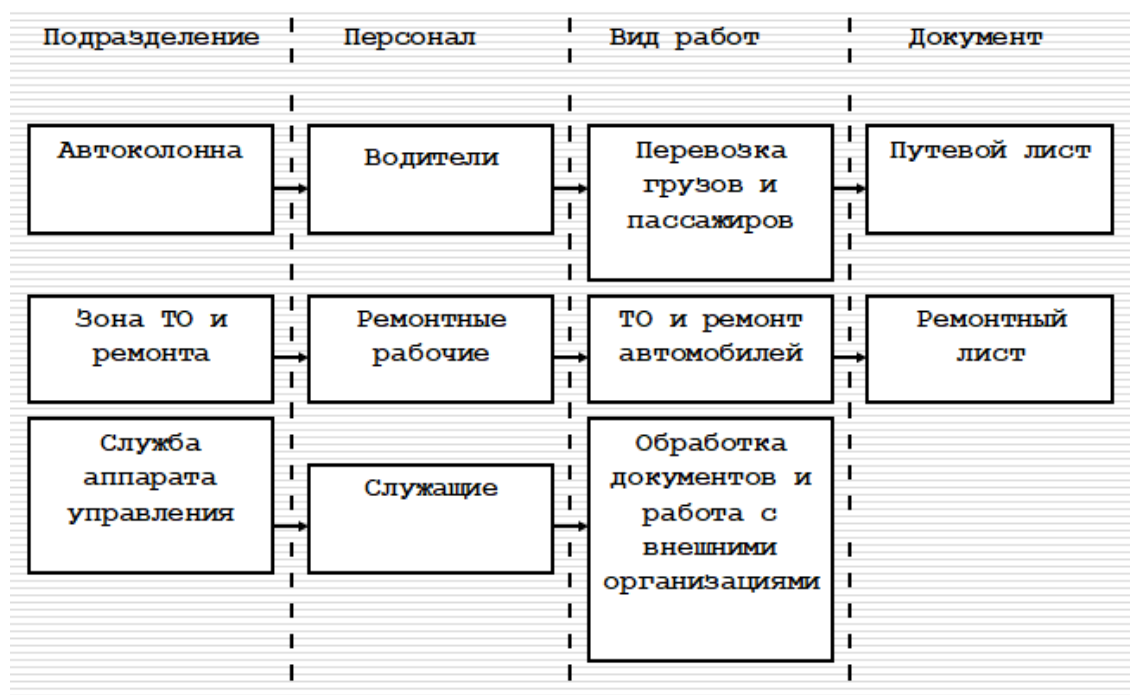


Рис. 1.4 Источники и носители информации о деятельности автотранспортного предприятия



Рис. 1.5 Схема компьютерного управления производственным процессом

1.2 Структура АИС

Работа сотрудников подразделения организации в АИС обеспечивается организацией Автоматизированных рабочих мест (АРМ). Это рабочее место, оснащенное комплексом компьютерных средств, позволяющих автоматизировать часть выполняемых операций. Основа АРМ - персональный компьютер, подключенный к сети, в сочетании с монитором, принтером и сканером. Кроме этого в зависимости от вида деятельности специалиста в состав АРМ могут входить комплекс измерительных приборов, анализаторов и др. Эффект от применения АРМ будет наибольшим если они будут оптимальным образом связаны между собой на протяжении всего цикла работы и смогут пользоваться единой базой данных. Это обеспечит наивысшую скорость обработки и полноту информации.

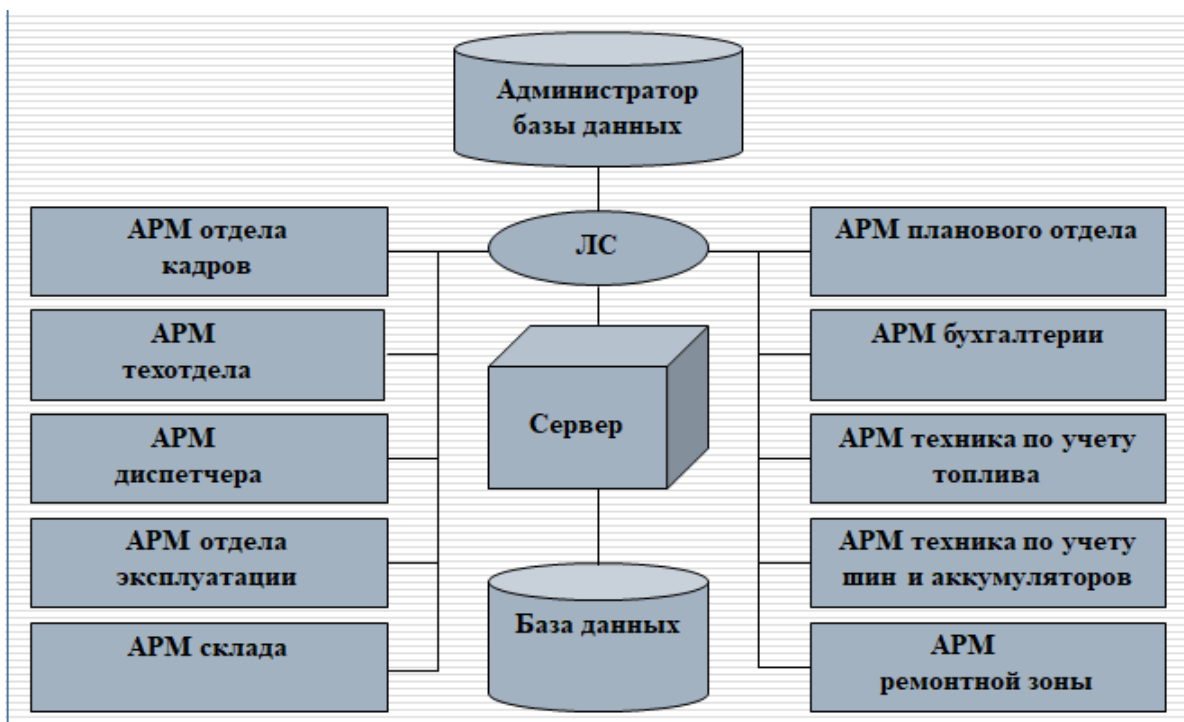


Рис. 1.6 Общая структурная схема автоматизированного управления автотранспортного предприятия

На Рис. 1.6 представлен вариант организации АРМ информационной системы автотранспортного предприятия.

Основным элементом всей системы является база данных, в которой хранится вся поступающая, обработанная и проанализированная информация. Администратор базы данных разграничивает доступ всех участников процесса управления. Все АРМ через линию связи (ЛС) подключаются к серверу, для проведения транзакций, доступа к БД, обработке и анализу информации. Все АРМ делятся на универсальные, кото-

рые имеются в любой организации и АРМ, управляющие непосредственно деятельностью организации.

Универсальными являются АРМ планового отдела, бухгалтерии, отдела кадров и склада. Для автотранспортного предприятия в организации деятельности участвуют АРМ ремонтной зоны, техотдела, диспетчера, отдела эксплуатации, техника по учету топлива, техника по учету шин и аккумуляторов.

1.3 Пользовательские сценарии АИС

В плановом отделе с помощью автоматизированной системы сотрудники должны планировать технико-экономические показатели, убытки и доходы. Кроме того, здесь выполняется расчет премий и анализ деятельности предприятий.

На предприятии функционирует база данных, представленная на Рис. 1.7, содержащая всю информацию о финансах. База данных заполняется следующим образом. Сотрудник планового отдела вносит информацию о планируемых доходах, убытках, а также технико-экономических показателях. Сотрудник планового отдела также заносит информацию в справочник.

После поступления информации в базу данных происходит расчет доходов, расходов и отработанных часов, после чего эти данные выводятся. Произведенный анализ предприятия и формирование премий отправляются в систему. Исходя из потребностей действующих лиц/акторов: сотрудника планового отдела и сотрудника организации, выделяются следующие сценарии работы:

- Формирование заявки;
- Планирование доходов;
- Планирование убытков;
- Планирование ТЭП;
- Занести информацию о предприятии;
- Формирование премий;
- Анализ деятельности предприятия;
- Показать доходы;
- Показать расходы;
- Показать отработанные часы.

Для АРМ бухгалтера по учету материальных ценностей система должна позволять вводить накладные, фиксировать их состояние, передачу и убытие. В фирме должна функционировать база данных, представленная на Рис. 1.8, содержащая всю информацию о материальных ценностях. Информация о каждой материальной ценности должна включать наименова-

ние, стоимость, номер и нахождение. Бухгалтер производит учёт, может вносить новые данные и корректировки. Цель автоматизации учёта материальных ресурсов – обеспечить своевременную обработку первичных документов и оперативно сформировать сведения о текущем состоянии и движении материальных ценностей, мест хранения и материально-ответственных лиц.

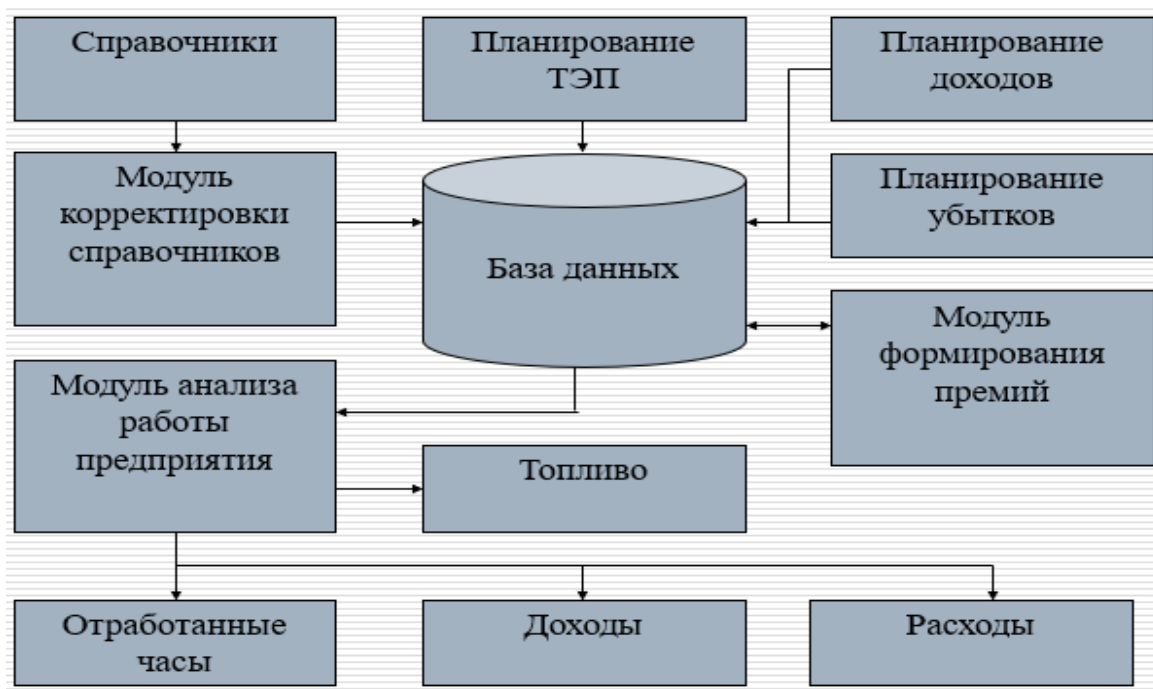


Рис. 1.7 Структура АРМ планового отдела

Исходя из потребностей действующих лиц/акторов бухгалтера и сотрудника организации, выделяются следующие сценарии работы:

- Внести данные о ценности.
- Изменение информации о ценностях.
- Фиксация состояния ценности.
- Передача в эксплуатацию ценности.
- Списание ценностей.
- Корректировать справочники.

Для АРМ бухгалтера по работе с клиентами на предприятии должна функционировать база данных, представленная на Рис. 1.9, содержащая всю необходимую информацию:

- Платежи
- Договоры
- Клиенты
- Справочники
- Акты сверки

Бухгалтерский учет отражает процессы, происходящие на предприятиях, в организациях и учреждениях при выполнении хозяйственных планов и те средства, при помощи которых эти процессы осуществляются.

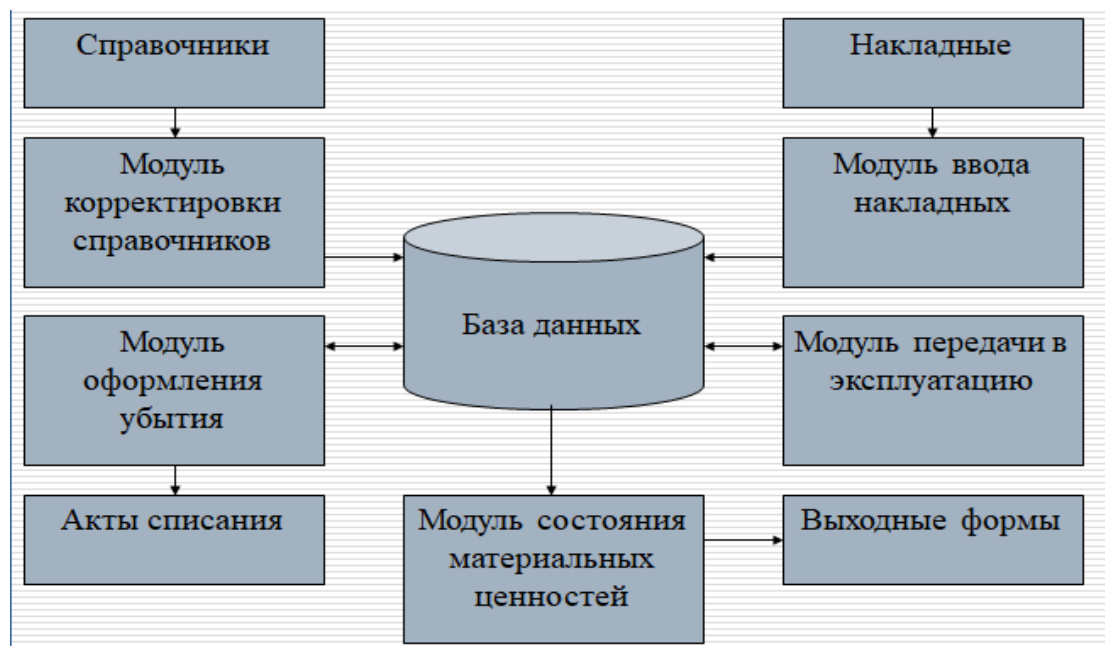


Рис. 1.8 Структура АРМ бухгалтера по учету материальных ценностей

Исходя из потребностей действующих лиц/акторов бухгалтера и клиента, выделяются следующие сценарии работы:

- Заполнить / корректировать данные справочников
- Подписать договор
- Заполнить данные по договорам
- Провести платежные операции
- Заполнить данные о движении денежных средств
- Запросить состояние расчетов
- Предоставить формы состояния расчетов
- Выдать состояние взаиморасчетов.

Для АРМ сотрудника отдела эксплуатации на многих предприятиях, чей автопарк имеет десятки или сотни машин, существует сценарий подбора автомобиля под каждый поступающий заказ. Важным моментом данного сценария является оптимальное использование ресурсов. Чаще всего на предприятии, в отделе эксплуатации, существует сотрудник, который подбирает автомобили.

Цель разработки – автоматизация процессов регистрации заказов и учета перевозок. Система должна позволять работнику отдела эксплуатации отслеживать процесс выполнения перевозок, наличие свободных автомобилей, а так же регистрации заказа. Работники отдела эксплуатации

должны иметь доступ к системе, чтобы указать какие есть на данный момент не выполняемые заказы, просмотр всех заказов и автомобилей.

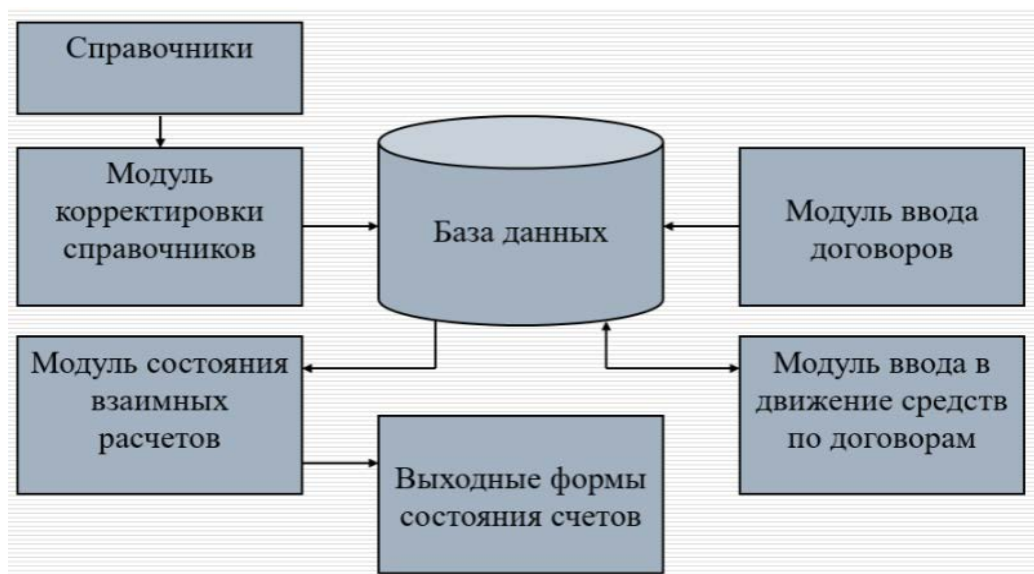


Рис. 1.9 Структура АРМ бухгалтера по работе с внешними клиентами

На предприятии должна функционировать база данных, представленная на Рис. 1.10, содержащая всю информацию о заказах и автомобилях. Регистрация нового заказа происходит следующим образом. Каждый день отдел эксплуатации получает список доступных заказов от отдела продаж. Информация о каждом заказе должна включать стоимость, суммарную массу груза, габариты, дальность перевозки и дополнительную информацию. Далее система, согласно нормам ПДД, подбирает подходящую машину для выполнения заказа из доступных автомобилей. Сотрудник отдела эксплуатации подбирает для каждого заказа соответствующий автомобиль. Отдел продаж присылает информацию о новых заказах. Каталог заказов представляет собой базу данных, содержащую информацию о всех заказах. Список автомобилей представляет собой базу данных, содержащую список свободных автомобилей.

Исходя из пользовательских сценариев, выделяются следующие варианты использования/прецеденты системы:

- Прислать информацию о новом заказе
- Просмотреть информацию о заказах
- Просмотреть информацию о автомобилях
- Зарегистрировать заказ

Для АРМ диспетчера на предприятии функционирует база данных, представленная на Рис. 1.11, содержащая всю информацию со справочниками (маршруты, режимная таблица, расписания). Оформление новых заявок происходит следующим образом. Диспетчер, получив заявку, должен

сформировать путевые документы, которые содержат следующую информацию: ФИО водителя, гос. номер, модель и марку ТС, стоимость перевозки, маршрут. Затем заказ отправляется в расчётную систему, где выполняются построение маршрута движения, и проверяется оплата заказа, после отправляется на выполнение водителю.

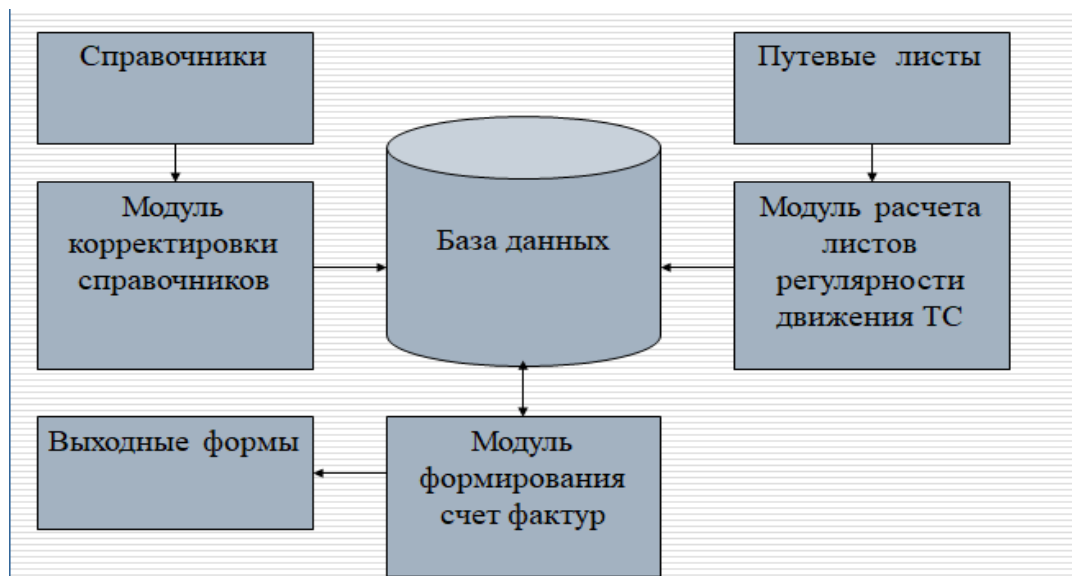


Рис. 1.10 Структура АРМ сотрудника отдела эксплуатации

Исходя из сценариев действующих лиц, выделяются следующие варианты использования АРМ диспетчера:

- сформировать путевые документы;
- провести инструктаж водителей об особенностях перевозок на маршруте;
- контролировать график движения и расписания;
- осуществлять контроль водителей автотранспорта на линии;
- иметь возможность вносить корректировки в имеющиеся сформированные путевые листы.

Следовательно, система должна позволять сотруднику оперативно-диспетчерского отдела регистрировать заказ в системе, отслеживать процесс выполнения заказа, загруженность водителей и ТС. Сотрудник отдела должен иметь доступ к системе, для внесения путевых документов, просмотр всех заявок, водителей и ТС.

Можно автоматизировать трудоемкие процессы, анализируя сценарии работы отделов в организации, которые необходимо выполнять. Разрабатывается структура базы данных и приложения, которые будут осуществлять обработку и анализ данных. На основе анализа структуры организации и выполняемых работ определяются стереотипы классов программного обеспечения, которые необходимо разработать.

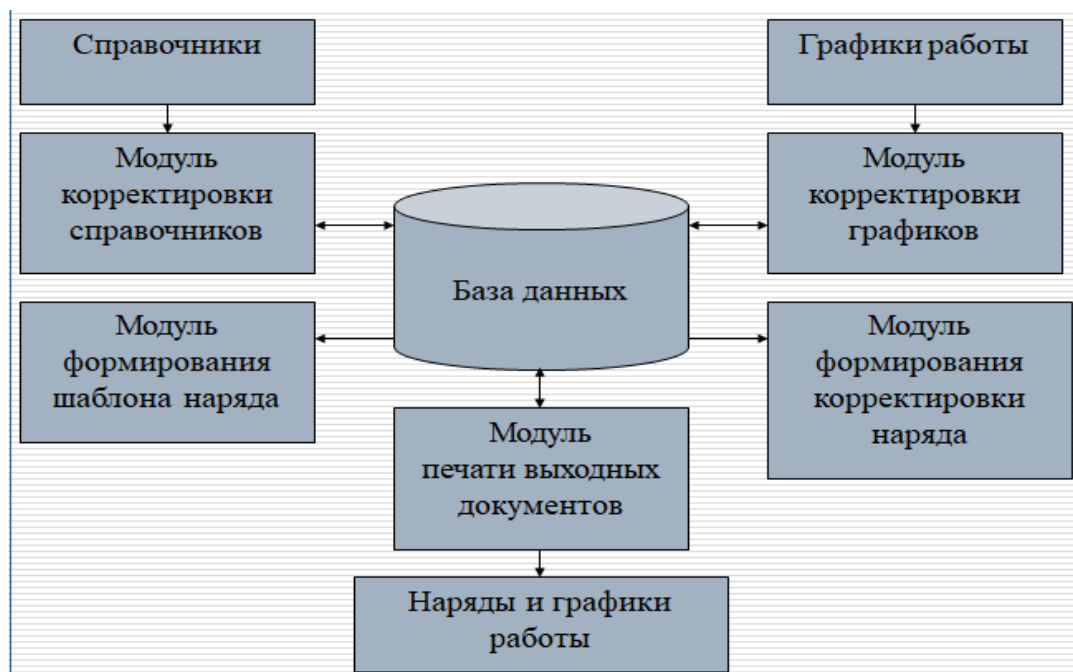


Рис. 1.11 Структура АРМ диспетчера

К основным стереотипам в потоках событий варианта использования UML относятся классы трех типов: граничные классы (Boundary), классы-сущности (Entity) и управляющие классы (Control), которые будут рассмотрены во втором разделе Методология проектирования Автоматизированных информационных систем.

1.4 Жизненный цикл АИС

Особенности предметной области конкретного предприятия влияют на структуру АИС. Но в то же время структуры различных организаций в целом схожи между собой. Любая организация состоит из нескольких подразделений, осуществляющих производственную деятельность. Организация рассматривается как совокупность взаимодействующих элементов (отделов), каждый из которых имеет свою структуру. Взаимосвязи между отделами сложны и делятся на три вида:

- функциональные связи, когда каждый отдел выполняет определенный вид работ в рамках единого бизнес процесса;
- информационные связи, в рамках которых отделы обмениваются информацией (приказами, распоряжениями, документами);
- внешние связи, которые могут быть как функциональными, так и внешними.

Автоматизированная информационная система разрабатывается как некоторый проект системы управления, представляющий собой множество

взаимосвязанных элементов. Многие особенности управления проектами и фазы разработки проекта (фазы жизненного цикла) являются общими, не зависящими от предметной области.

Существуют международный стандарт, регламентирующий жизненный цикл информационных систем ISO/IEC 12207, который определяет структуру жизненного цикла, содержащую процессы, действия и задачи, которые должны быть выполнены во время создания АИС.

Согласно стандарту структура жизненного цикла основана на трех группах процессов:

- основные процессы жизненного цикла (приобретение, поставка, разработка, эксплуатация и сопровождение);
- вспомогательные процессы, обеспечивающие выполнение основных процессов (документирование, управление конфигурацией, обеспечение качества, верификация, аттестация, оценка, аудит);
- организационные процессы (управление проектами, создание инфраструктуры проекта, определение, оценка и улучшение самого жизненного цикла проекта, обучение).

Среди основных процессов жизненного цикла наибольшую важность имеют разработка, эксплуатация и сопровождение. Каждый процесс характеризуется определенными задачами и методами их решения, данными, полученными на предыдущих этапах, и результатами.

Полный жизненный цикл АИС включает в себя стратегическое планирование, анализ, проектирование, реализацию, внедрение и эксплуатацию. В принципе это деление достаточно произвольно и зависит от используемой технологии разработки программного обеспечения. Например, популярное, универсальное CASE - средство Rational Rose как инструмент RUP (Rational Unified Process) цикл подразделяет на четыре фазы:

- начало;
- уточнение;
- конструирование;
- переход (передача в эксплуатацию).

Границы каждой фазы, определяются моментами достижения определенной ключевой цели.

На начальной фазе определяются все внешние объекты, с которыми взаимодействует разрабатываемая система, классифицируются функциональные возможности системы и производится их описание.

На фазе уточнения проводится анализ предметной области, разрабатывается архитектура автоматизированной системы. При этом необходимо описать большинство функциональных возможностей системы и учесть взаимосвязи между отдельными ее элементами.

На фазе конструирования разрабатываются логическая и физическая структуры базы данных, приложения, реализующие функциональные воз-

можности системы. Также оценивается работоспособность разработанного программного обеспечения.

На фазе внедрения, при эксплуатации разработанной системы в реальных условиях, возникают различного рода проблемы, которые требуют дополнительных работ по внесению корректив во внедряемый продукт. Это связано с обнаружением ошибок и недоработок.

1.5 Модели жизненного цикла АИС

Под моделью жизненного цикла любого программного продукта понимают порядок процессов, действий и задач, выполняемых на протяжении жизненного цикла автоматизированной системы и взаимосвязи между ними.

В стандарте не конкретизируются в деталях методы реализации действий и задач, входящих в процессы жизненного цикла АИС, лишь описываются контуры этого процесса. Регламенты стандарта являются наиболее общими для любых моделей, методологий и технологий разработки систем и реализации жизненного цикла.

Модель жизненного цикла зависит от специфики автоматизированной системы и условий, в которых она создается и работает. Наиболее распространены каскадная и спиральная модели жизненного цикла.

Каскадная модель предусматривает последовательную организацию работ. Переход от одного этапа к другому происходит после того, как завершаются работы предыдущего этапа. Каждый этап завершается выпуском полного комплекта документации, чтобы любой разработчик мог продолжить работу над созданием системы. Можно выделить ряд этапов, не зависящих от предметной области:

- анализ требований и постановка задачи на разработку системы;
- проектирование системы;
- разработка системы;
- тестирование и опытная эксплуатация;
- сдача готового продукта в эксплуатацию.

На первом этапе проводится исследование предметной области, определяются задачи, которые должны быть решены. Постановка задачи (в виде технического задания) согласовывается со всеми заинтересованными сторонами. Основными предметами исследований являются выбор участников процесса функционирования системы, общие и функциональные требования к разрабатываемой системе и структура хранимой информации, входных и выходных данных.

На втором этапе разрабатываются проектные решения, удовлетворяющие поставленным требованиям, сформулированным в постановке задачи. На этом этапе могут быть изменено содержание и направление потоков информации, оптимизирована структура информации, предназначен-

ная для хранения, входные и выходные данные, уточнены общие и функциональные требования.

На третьем этапе при реализации проекта осуществляется разработка программного обеспечения в соответствии с проектными решениями. Результатом выполнения данного этапа являются готовые приложения, принимающие и выдающие данные, сохраняющие ее в БД, выполняющие обработку и анализ информации.

На четвертом этапе проводится тестирование всего программного продукта на предмет соответствия установленным требованиям постановки задачи, выявляются скрытые недостатки, проявляющиеся в реальных условиях работы.

На этапе сдачи продукта проверяется соответствие системы поставленным требованиям.

Подобная модель жизненного цикла позволяет проводить итерационные процедуры уточнения требований к системе и вариантов проектных решений. Такая модель жизненного цикла чаще всего используется для разработки АИС малых и средних организаций. Последовательные работы по созданию системы позволяют планировать сроки завершения и затраты. Недостатками такой модели является задержка получения результата. В результате последовательного характера работ несоответствие поставленным требованиям может выясниться на более поздних этапах. Это потребует пересмотра всего проекта функциональной и информационной модели, модели интерфейса пользователя и пользовательской документации. Возврат на предыдущие этапы разработки может быть причиной срыва графика работ.

Недостатками каскадной модели является также сложность параллельного ведения работ. Это связано с необходимостью постоянного согласования отдельных частей проекта. Сложность управления проектом при каскадной модели обусловлена строгой последовательностью стадий разработки и наличием сложных взаимосвязей между различными частями проекта.

Спиральная модель жизненного цикла предполагает итерационный процесс разработки АИС. Возрастает важность начальных этапов жизненного цикла, таких как анализ и проектирование. Каждая итерация представляет собой законченный цикл разработки, приводящий к выпуску версии изделия (или подмножества конечного продукта), которое совершенствуется от итерации к итерации, чтобы стать законченной системой. Каждый виток спирали соответствует созданию фрагмента или версии программного изделия. На нем уточняются цели и характеристики проекта, определяется его качество, и планируются работы следующего витка спирали. На каждой итерации углубляются и последовательно конкретизируются детали проекта. В результате чего выбирается обоснованный вариант, который доводится до окончательной реализации.

Использование спиральной модели позволяет осуществлять переход на следующий этап выполнения проекта, не дожидаясь полного завершения работы на текущем этапе. Недоделанную работу можно выполнить на следующей итерации.

Основной проблемой спиральной модели является определение момента перехода на следующий этап. Для ее решения необходимо ввести временные ограничения на каждый из этапов жизненного цикла.

1.6 Методология и технология разработки АИС

Методология создания АИС заключается в организации процесса построения системы и обеспечения управления этим процессом для того, чтобы гарантировать выполнение требований к самой системе и к характеристикам процесса разработки. Основными задачами, решение которых должна обеспечивать методология создания АИС являются следующие:

- обеспечение создания систем, отвечающим целям и задачам деятельности организации и соответствующих предъявляемых к ним требованиям по автоматизации всех процессов;
- гарантия создания системы с заданными параметрами в течении заданного времени;
- простота сопровождения, модификации и расширение системы с целью обеспечения ее соответствия изменяющимся условиям функционирования организации;
- обеспечение создания АИС, отвечающих требованиям открытости, переносимости и масштабируемости;
- возможность использования в создаваемой системе разработанных ранее средств информационных технологий.

Основой технологии проектирования являются технологические инструкции, состоящие из описания последовательности технологических операций и условий, в зависимости от которых выполняется та или иная операция, и описаний самих операций.

Технология проектирования может быть представлена совокупностью трех составляющих:

- заданной последовательности выполнения технологических операций проектирования;
- критериев и правил, используемых для оценки результатов выполнения технологических операций ;
- графических и текстовых средств (нотаций), используемых для описания проектируемой системы.

Каждая технологическая операция должна обеспечиваться следующими материальными и информационными ресурсами:

- исходными данными или данными, полученными на предыдущей операции, представленными в стандартном виде;
- методическими материалами, инструкциями, нормативами и стандартами;
- программными и техническими средствами;
- исполнителями.

Результаты проектирования должны представляться в стандартном виде, обеспечивающем их адекватное восприятие при выполнении следующих технологических операций, на которых они используются как исходные данные.

Перечислим ряд требований, которым должна удовлетворять технология проектирования, разработки и сопровождения автоматизированных систем:

- поддерживать полный жизненный цикл системы;
- обеспечивать гарантированное достижение целей разработки системы с заданным качеством и заданные сроки;
- обеспечивать возможность разделения крупных проектов на ряд подсистем, осуществлять декомпозицию проекта на составные части, разрабатываемые группами исполнителей, с последующей интеграцией составных частей.

2. МЕТОДОЛОГИИ ПРОЕКТИРОВАНИЯ Автоматизированных информационных систем

2.1 Понятие методологии проектирования

Проектирование – это структурирование и моделирование сложных систем с целью их разработки. Методология проектирования – это идея (концепция) и метод разработки, которая определяет процессы, модели, методику, этапность и правила разработки проекта.

Состав методологии проектирования системы:

- 1 *Набор моделей* для описания требований, проектных и программных решений системы. Модель характеризуется конструкцией (*нотацией*) и правилом применения (синтаксисом нотации).
- 2 *Методика применения* набора моделей для проектирования системы. Методика описывает определенный набор моделей и последовательность их построения для проектирования системы.
- 3 *Процесс* проектирования - это последовательность шагов, которая включает виды, этапы и исполнителей работ по разработке автоматизированной системы. Процессы регламентируются стандартами и зарекомендовавшими себя технологиями.

В моделях отражается предметная область системы. *Предметная область системы (domain)* - часть реального мира, которая исследуется с целью организации управления и автоматизации, рассматривается как совокупность реальных процессов и объектов (сущностей), имеющих непосредственное отношение к функционированию системы.

Модели представлены диаграммами. *Диаграмма (diagram)*- графическое представление композиции элементов модели в форме связного графа, вершинам и ребрам которого приписывается определенная семантика.

Для проектирования автоматизированных информационных систем используют комбинацию двух методологических подходов: структурного и объектно-ориентированного. Концептуально подходы отличаются принципом декомпозиции системы. Структурный подход использует функциональную декомпозицию системы, а объектно-ориентированный - объектную декомпозицию.

Функциональная декомпозиция рассматривает структуру системы в терминах иерархии функций (задач, работ) и передачи информации (данных) при выполнении функции. В основе реализации проекта лежит алгоритмическая декомпозиция программ на подпрограммы (процедуры и функции), где каждая подпрограмма или набор подпрограмм выполняет проектную функцию (задачу, работу). Методология возникла исторически первой и отражает технологию разработки программ того времени - процедурное программирование. Сейчас данная методология используется в

проектировании систем для исследования предметной области системы, выявления основных бизнес-процессов системы и требований к системе.

Объектная декомпозиция рассматривает структуру системы как набор объектов предметной области и связей между ними, а поведение системы в терминах обмена сообщений между объектами. В основе реализации проекта системы – объектно-ориентированное программирование, где описание объекта представлено в виде класса (типа), обладающего методами взаимодействия с другими классами. Данная методология является основной для моделирования, проектирования и представления программных решений автоматизированных систем.

2.2 Объектно-ориентированная методология

Основная идея объектного подхода к разработке информационных систем была выражена одним из ее основателей Г. Бучем «... полезнее всего создавать такие модели, которые фокусируют внимание на объектах, найденных в самой предметной области, и образуют то, что мы назвали объектно-ориентированной декомпозицией».

Концептуальная основа методологии – объектная модель. Основные принципы построения: абстракция, инкапсуляция, наследование, полиморфизм. Основные понятия: объект, класс, атрибут, операция, интерфейс, отношения классов и др.

Основные процессы объектно-ориентированной методологии:

1. объектно-ориентированный анализ (object-oriented analysis, OOA);
2. объектно-ориентированное проектирование (object-oriented design, OOD, ООП);
3. объектно-ориентированное программирование (object-oriented programming, OOP, ООП).

Взаимосвязь процессов для разработки моделей показана на Рис. 2.1 и Рис. 2.2.



Рис. 2.1 Взаимосвязь анализа, проектирования и программирования

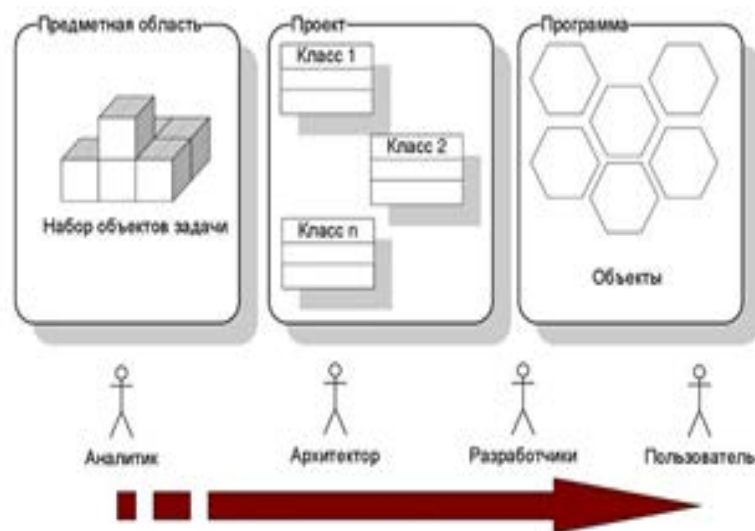


Рис. 2.2 Представления системы в процессе разработки

2.2.1 Унифицированный процесс разработки системы

Процесс производства программного обеспечения (Software Engineering Process, SEP или Software Development Process) систем отвечает на вопросы: кто и какую работу выполняет, в какой последовательности создается система.

SEP – это процесс разработки, который требования пользователя преобразует в программное обеспечение (ПО) автоматизированной системы.

Унифицированный процесс разработки программного обеспечения (Unified Software Development Process, USDP) – это SEP от авторов унифицированного языка моделирования UML (Unified Modeling Language) объектно-ориентированной методологии. Процесс называют Унифицированным процессом - UP, а UML – основной нотацией методологии.

Проектирование на UML определяется языком моделирования и процессом производства программного обеспечения. Язык диаграмм UML – это визуализация моделей проекта, а UP¹ – это процесс.

RUP (Rational Unified Process) - унифицированный процесс компании Rational – это коммерческая версия UP². RUP имеет стандарт, инструменты средства и веб-окружение, включающие всю документацию процесса и полные руководства для каждого инструментального средства.

Для моделирования понятия «кто выполняет» UP вводит концепцию **исполнителя** (worker). Это роль, которую выполняет в проекте отдельный индивидум или команда.

¹ UML стандартизован OMG, а UP не стандартизован. OMG (Object Management Group) — некоммерческий консорциум (рабочая группа), занимающийся разработкой и продвижением объектно-ориентированных технологий и стандартов.

² Поддерживает и разрабатывает компания IBM

Понятие «какую работу» представлено в виде деятельности и артефактов. **Деятельности** – это задачи, которые будут выполняться в проекте исполнителями. При определенной деятельности исполнители будут исполнять конкретные роли, для любой деятельности в UP (RUP) определен участвующий в ней исполнитель (роль). Деятельности могут быть разбиты на более мелкие уровни детализации. **Артефакты** – это сущности, являющиеся входными данными и результатами проекта: исходный код, исполняемая программа, стандарты, документация и т. д. Для представления артефактов используются различные пиктограммы в зависимости от типа артефакта (Рис. 2.3).

Понятие «последовательность» в UP представлено **рабочими потоками** как последовательности взаимосвязанных деятельности, осуществляемых исполнителями. RUP рабочие потоки высокого уровня (Требования, Тестирование) определяет как дисциплины (disciplines). Рабочий поток разбивают на одну или более составляющих, которые описывают деятельности, роли и артефакты, участвующие в нем. Эти составляющие рабочего потока в UP идентифицируют только по имени, а в RUP - пиктограммами.

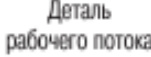
UP	RUP	Семантика
 Исполнитель	 Роль	Кто – роль, выполняемая в проекте отдельным индивидуумом или командой
 Деятельность  Артефакт	 Деятельность  Артефакт	Что – единица работы, осуществляемая исполнителем (ролью), или артефакт, производимый в проекте
 Рабочий поток  Деталь рабочего потока	 Дисциплина  Деталь рабочего потока	Когда – последовательность взаимосвязанных действий, составляющих основную ценность проекта

Рис. 2.3 Пиктограммы UP и RUP для представления основных понятий

Для каждого конкретного проекта процесс UP разработки должен настраиваться как уникальный. Процесс настройки включает:

- определение и объединение внутренних стандартов;
- определение и объединение шаблонов документов;
- определение и объединение инструментальных средств: компиляторов, инструментов управления конфигурацией и т. д.;
- определение и объединение баз данных: отслеживание ошибок, слежение за проектом и т. д.;
- определение и объединение изменений жизненного цикла: например, более сложные методы контроля качества для систем к обеспечению безопасности.

UP основан на трех аксиомах:

- процесс управляет требованиями и риском;
- процесс архитектуро-центричный;
- процесс итеративный и инкрементный.

Цель UP – разработать пошагово (итеративно) качественную и надежную систему.

«Каждая итерация создает базовую версию, включающую в себя частично завершенную версию целевой системы и документацию проекта. В ходе последовательных итераций базовые версии наращиваются до тех пор, пока не будет создан окончательный полный вариант системы. Разница между двумя последующими базовыми версиями – это инкремент. Вот почему UP называют итеративным и инкрементным жизненным циклом» [7]. Ключевой момент – каждая итерация включает все стандартные этапы проекта по разработке системы: планирование, анализ и проектирование, построение, интеграция и тестирование, версия для внутреннего или внешнего использования.

Базовая версия очередной итерации состоит из набора рассмотренных и утвержденных артефактов, созданных в данной итерации и:

- предоставляет базу для дальнейшего рассмотрения и разработки;
- изменяется только формальными процедурами управления конфигурацией и изменениями.

Инкременты – это разница между двумя последовательными базовыми версиями, шаги по направлению к окончательной версии системы.

Итерации группируются в фазы, которые образуют структуру процесса UP. Каждую итерацию определяют пять основных рабочих потоков, где для каждого указано, что должно быть сделано и какие навыки для этого необходимы. Вспомогательные потоки – оценка, планирование и другие характерные для проекта потоки. Пять основных рабочих потоков (Рис. 2.4, 2.5):

- определение требований (Т);
- анализ (А) – уточнение и структурирование требований;
- проектирование (П) – реализация требований в архитектуре системы;
- реализация (Р) – разработка программного обеспечения;
- тестирование (Т) – проверка, отвечает ли реализация требованиям.

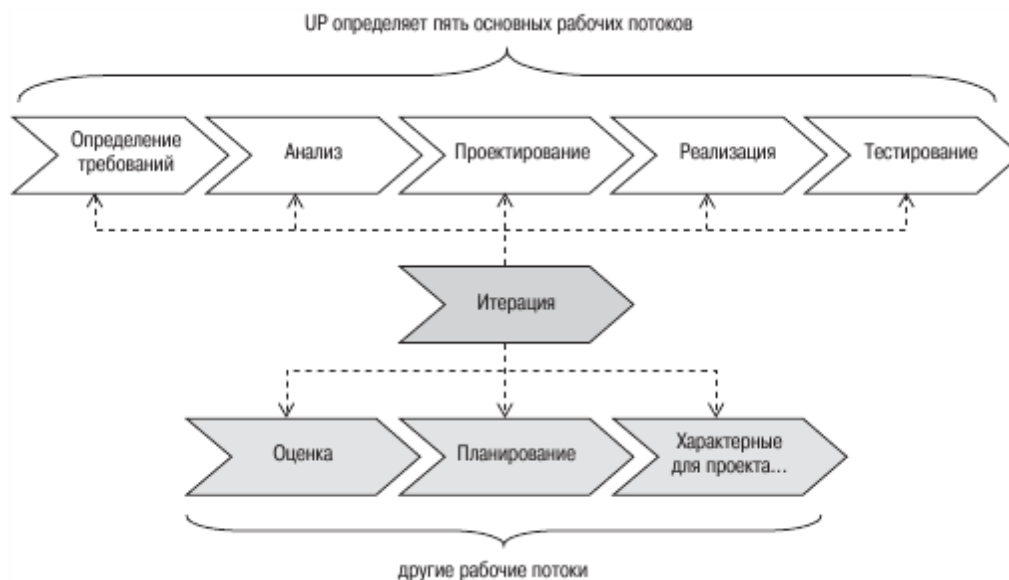


Рис. 2.4 Рабочие потоки и итерации

Разбиение проекта на серию итераций обеспечивает планирование упорядоченной во времени последовательности итераций, где каждая последующая итерация выполняется на базе предыдущей. Но итерации можно расположить параллельно. Это более сложное планирование, так как предполагает понимание зависимостей между артефактами каждой итерации и требует основанного на архитектуре и моделировании подхода к разработке ПО. Преимущество параллельных итераций – короткое время вывода нового изделия на рынок и более рациональное использование команды.



Рис. 2.5 Структура UP

Структура UP – это жизненный цикл проекта (Рис. 2.5), содержит четыре фазы, каждая из которых имеет свои контрольные точки. Фаза состоит из одной или более итерации, в каждой итерации выполняются пять основных и любое количество вспомогательных рабочих потоков. Число итераций в фазе зависит от размера проекта, длительность итерации - не

более двух-трех месяцев. Организационно каждая фаза определяется целью, основной деятельностью с акцентом на одном или более рабочих потоках и контрольной точкой.

Четыре фазы:

- Начало (Inception) – цели жизненного цикла;
- Уточнение (Elaboration) – архитектура жизненного цикла;
- Построение (Construction) – базовая функциональность;
- Внедрение (Transition) – выпуск продукта.

Принцип работы UP представлен на Рис. 2.6, где кривые на графике показывают относительный объем работы, выполняемый в каждом из пяти основных рабочих потоков по мере прохождения проекта по фазам.

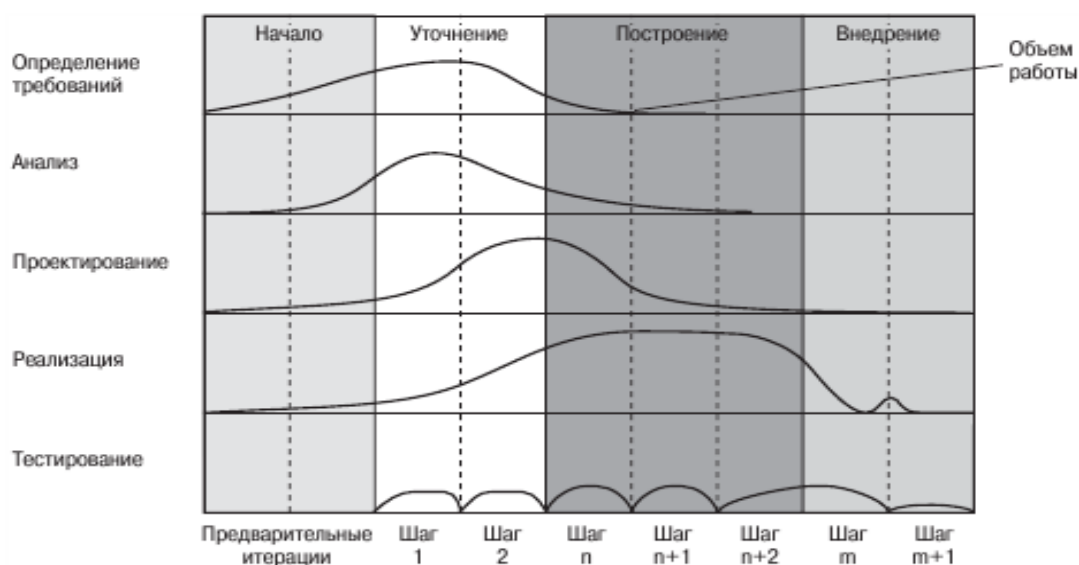


Рис. 2.6 Относительный объем работы, выполняемых в рабочих потоках по мере прохождения проекта по фазам [7]

Назначение каждой фазы:

- Начало (Inception): первичная оценка проекта и принятие решения о том, стоит ли вкладывать средства в фазу уточнения.
- Уточнение (Elaboration): идентифицируются и разрабатываются основные прецеденты проекта, в итеративном процессе создается программное обеспечение, для того чтобы развернуть архитектуру системы. Результат фазы - понимание требований и прототип работающей системы, которая станет основой дальнейшей разработки. Обнаружить и разрешить основные риски проекта.
- Построение (Construction): разрабатывается функциональность системы, достаточная для выпуска продукта.
- Внедрение (Transition): различные работы, выполняемые в конце и в не итеративном режиме - развертывание демоверсии системы, обучение пользователей и др.

Рассмотрим рабочие потоки УР в диаграммах [7]. Диаграммы называются детализацией рабочего потока, так как детализируют составляющие задачи определенного потока работ. Детализация рабочего потока показывает исполнителей и деятельности, вовлеченные в конкретный рабочий поток. Представлены диаграммы:

- 1) Детализация рабочего потока определения требований, Рис. 2.7.
- 2) Детализация рабочего потока анализа, Рис. 2.8.
- 3) Детализация рабочего потока проектирования, Рис. 2.9.
- 4) Детализация рабочего потока реализации и тестирования, Рис. 2.10.

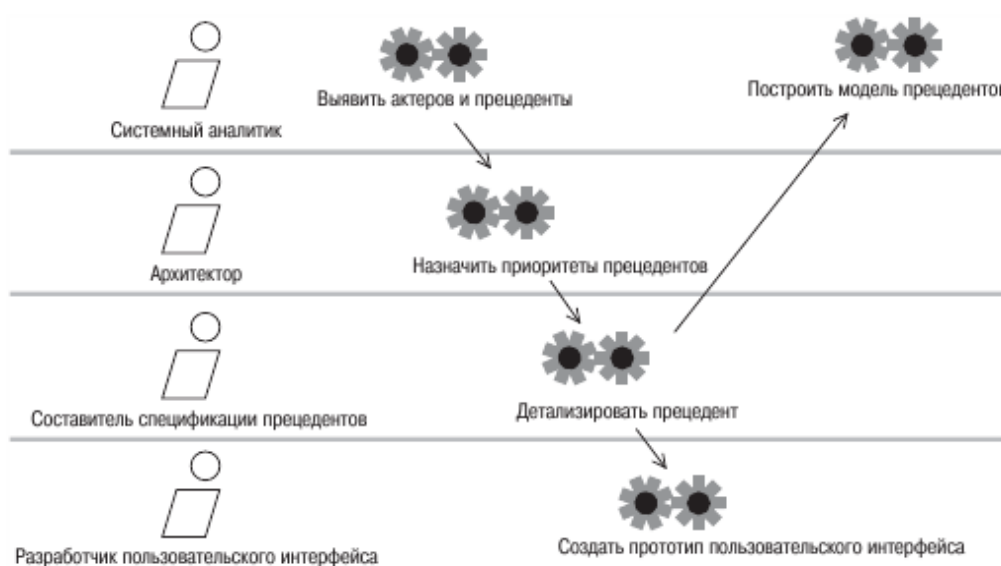


Рис. 2.7 Детализация рабочего потока определения требований

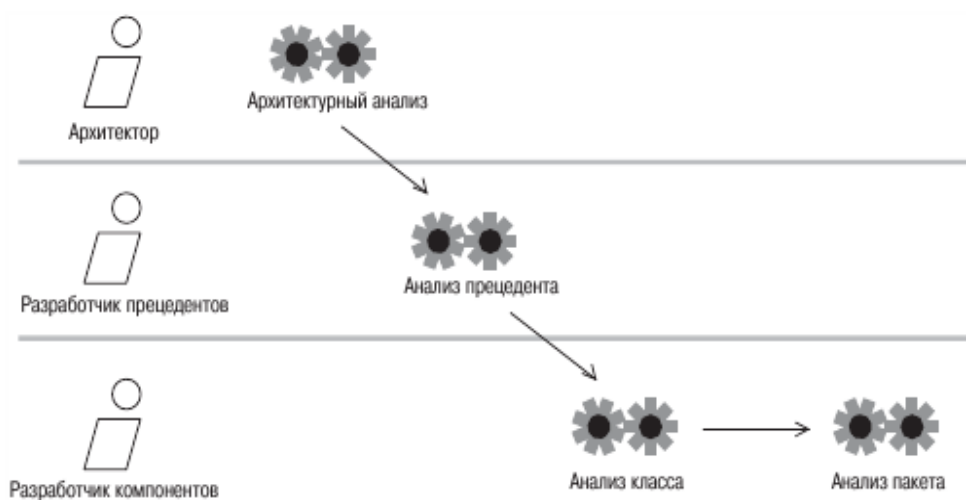


Рис. 2.8 Детализация рабочего потока анализа

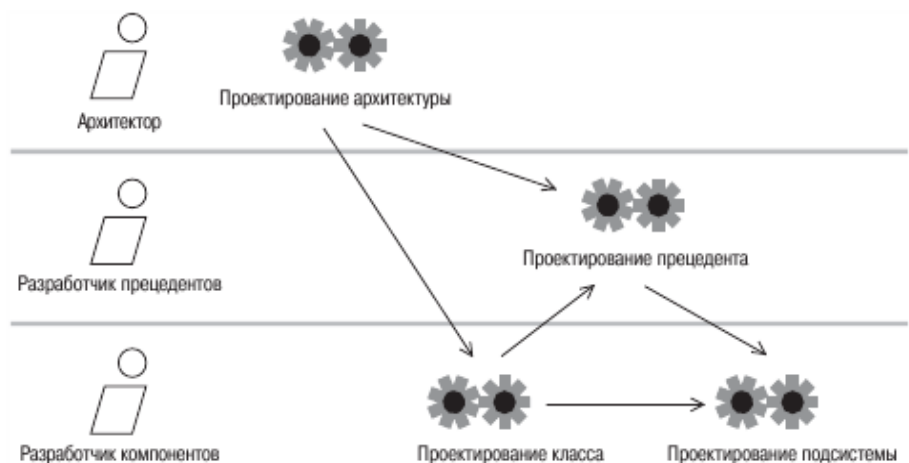


Рис. 2.9 Детализация рабочего потока проектирования

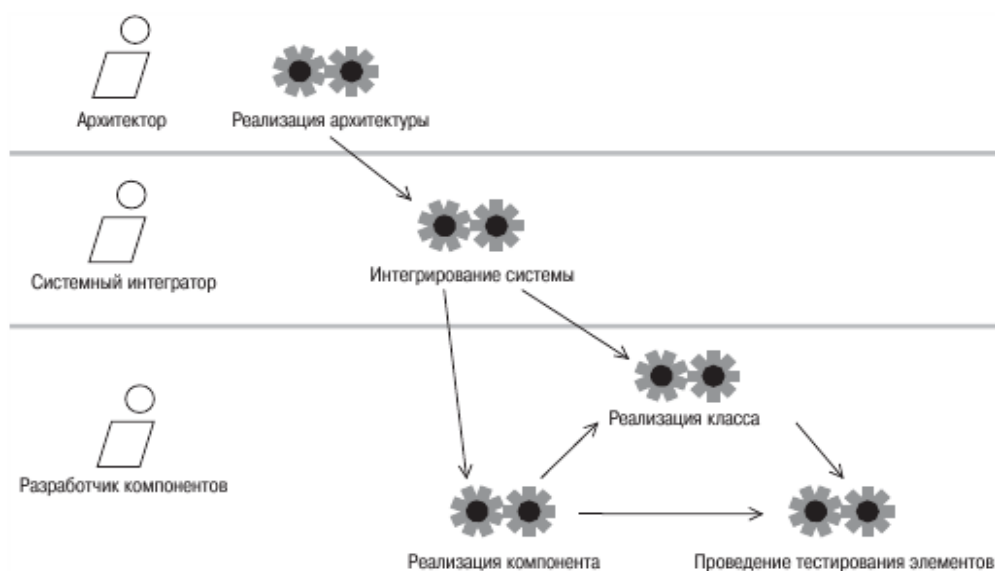


Рис. 2.10 Детализация рабочего потока реализации и тестирования

2.2.2 Модель представлений архитектуры системы

В основе процессов моделирования, проектирования и в целом разработки системы лежит понятие *архитектуры системы*. ГОСТ Р 57100-2016/ISO/IEC/IEEE 42010:2011 – «Системная и программная инженерия. ОПИСАНИЕ АРХИТЕКТУРЫ» регламентирует это понятие.

"Архитектура – фундаментальные концепции или свойства системы в ее окружении, выраженные в ее элементах, отношениях и принципах их проектирования и развития" (Стандарт ISO/IEC/IEEE 42010:2011).

«Архитектура системы может быть представлена как ряд структурированных архитектурных сущностей и их отношений, таких, как

функции, потоки, взаимодействия, объекты потока ресурсов, элементы информации/данных, физические компоненты, контейнеры, узлы, связи, коммуникационные ресурсы и т. д.» (ГОСТ Р 57123-2016).

Архитектура программного обеспечения определяет высокоуровневую структуру абстракции системы, используя методы декомпозиции и композиции и должна соответствовать модели требований системы. Формализованное описание архитектуры:

Архитектура ПО = {Элементы, Формы, Обоснование / Ограничения}

Архитектура - это фундаментальные, ключевые решения в целом о системе. В процессе проектирования одно решение зависит от других. На основании архитектурных решений происходит управление детальным проектированием и ограничениями проекта [10]. Архитектура отвечает за *управление проектом разработки*, она определяет не только принципы организации системы, но и принципы её реализации, испытаний, внедрения, эксплуатации и сопровождения.

Архитектуру будущей системы исследуют в разрезе "точки зрения" (view point) заинтересованного лица на представление архитектуры (view).

В 1995г. Ф. Крухтен предложил модель представлений программной архитектуры «4 + 1 View Model» [13]. Эта модель основана на одновременном использовании нескольких представлений архитектуры. Использование нескольких представлений позволяет разделить точки зрения заинтересованных лиц - конечные пользователи, разработчики или руководители проекта, - и работать с функциональными и нефункциональными требованиями. Модель содержит пять взаимодополняющих представлений.

Четыре представления – это логическая структура, разработка, процессы и физическая структура системы (Рис. 2.11- 2.12). Сценарии использования иллюстрируют описываемую архитектуру. Поэтому модель состоит из 4+1 представления.

Logical view: логическое представление моделирует функциональность, которая предоставляется системой для конечных пользователей. В этом представлении используются UML-диаграммы классов, связей и последовательностей.

Development view: представление разработки показывает систему с точки зрения разработчика и проектирует управления программной системой. Другое название этого представления - представление реализации. Используются UML-диаграммы компонентов и пакетов для описания компонентов системы и их объединения в логические пакеты (например, слои).

Process view: представление процесса - это динамика, поведение системы, отражает происходящие в системе процессы и связи между ними. Процессы представляют поведение системы во время выполнения программы, включая параллелизм выполнения, распределение, интеграцию, производительность, масштабирование и т.п. Для представления процесса используют UML-диаграммы активности/деятельности.

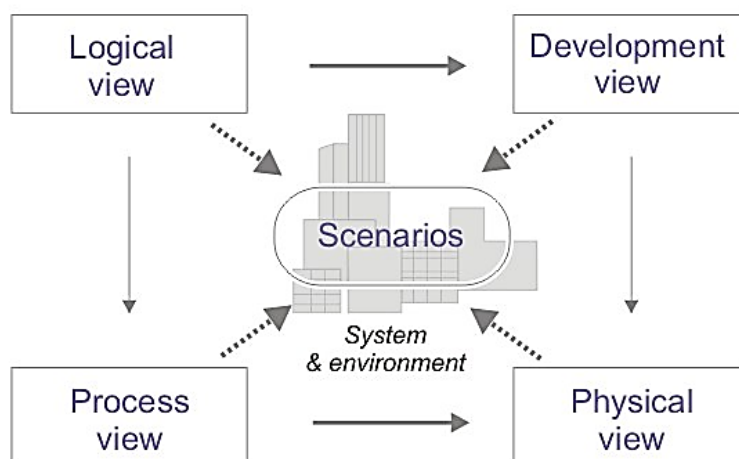


Рис. 2.11 Модель 4+1 преставлений архитектуры системы

Physical view: представление физической структуры показывает систему с точки зрения системного инженера, где разрабатывается распределение программных компонентов по физическим уровням и физические каналы связи между уровнями. Это представление развёртывания системы. Используют UML-диаграмму развёртывания.

Scenarios: описание архитектуры системы должно быть продемонстрировано набором вариантов использования/прецедентов, или сценариев. Набор сценариев составляет пятое представление. Сценарии моделируют взаимодействие между объектами и между процессами системы. Сценарии используются для идентификации архитектурных элементов и для тестирования архитектурного прототипа системы, что проверяет качество архитектуры системы. Сценарии описываются UML-диаграммами вариантов использования/прецедентов.

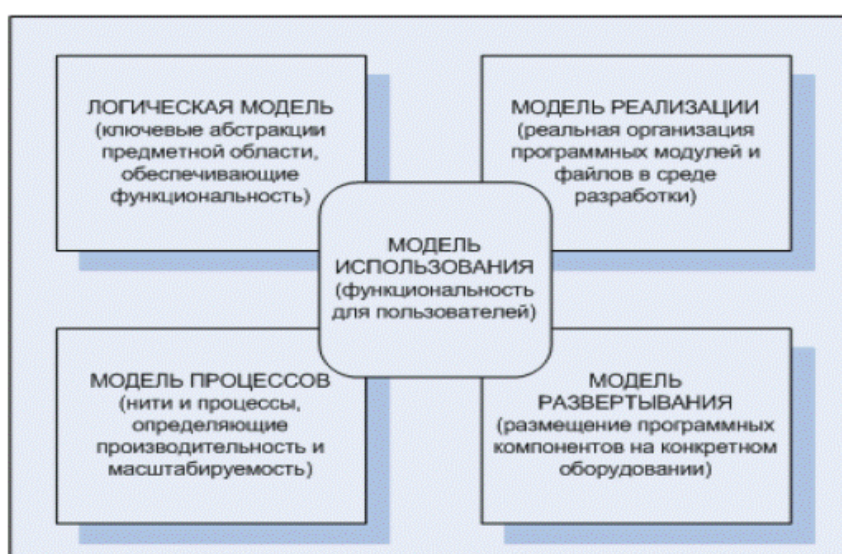


Рис. 2.12 Описание представлений Модель 4+1

Точка зрения на архитектуру заинтересованного лица (Рис. 2.13) – это интересы и подходы каждого заинтересованного лица (или группы лиц) системы. Представления, которые есть у заинтересованного лица должны совпадать с представлением, которое архитектор ему демонстрирует. Это означает, что достигнуто взаимопонимание между заинтересованным лицом и архитектором, что позволяет архитектору корректно сочетать интересы всех заинтересованных лиц и разработать в результате качественный проект [10].

Рассмотрим представления заинтересованных лиц.

Представление пользователя позволяет увидеть, как система отражает предметную область и вписывается в повседневную работу непосредственных пользователей системы, включая прототипы пользовательских интерфейсов.

Представление бизнес-процессов позволяет увидеть, как система вписывается в процессы деятельности организации и автоматизирует их.

Представление разработки (или сопровождения) системы показывает, как система должна быть реализована (или уже была реализована).

Представление развёртывания даёт информацию о конфигурации системы в её окружении в процессе эксплуатации.

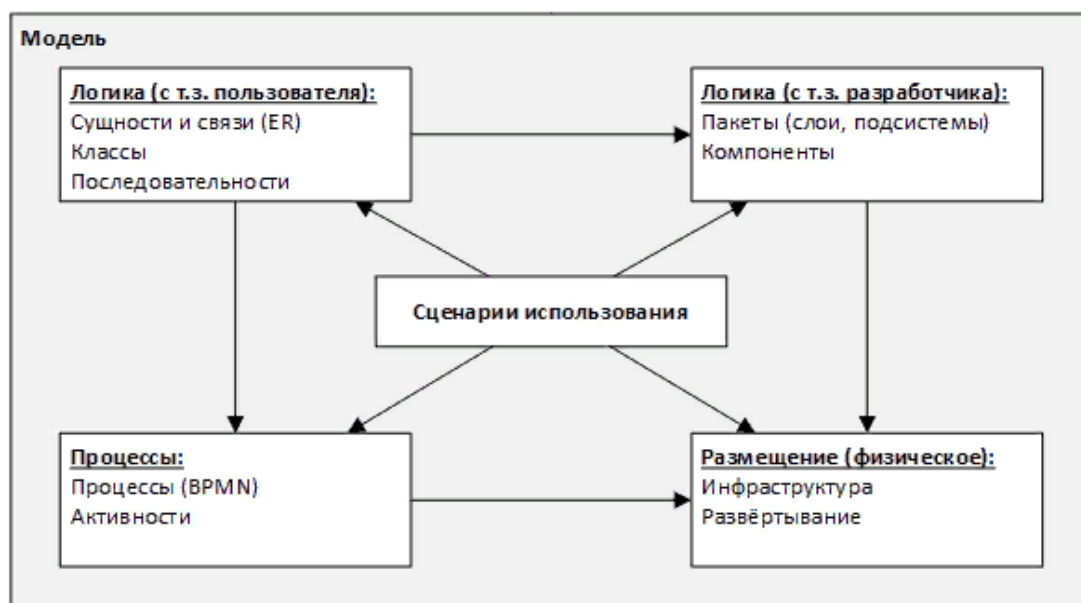


Рис. 2.13 Точка зрения заинтересованных лиц Модели 4+1[10]

В центре модели 4+1 помещены сценарии/варианты использования/прецеденты системы, это функциональные сценарии интересные для конечных пользователей. Сценарии несут определенную информацию для представлений бизнес-процессов, разработки и развёртывания системы, но этой информации недостаточно для разработки системы в целом. Набор сценариев на основе пользовательских историй не всегда достаточен даже для проектирования пользовательского интерфейса.

Архитектору системы для заинтересованных лиц, например, разработчиков и системных администраторов необходимо более детальное проектирование представлений архитектуры.

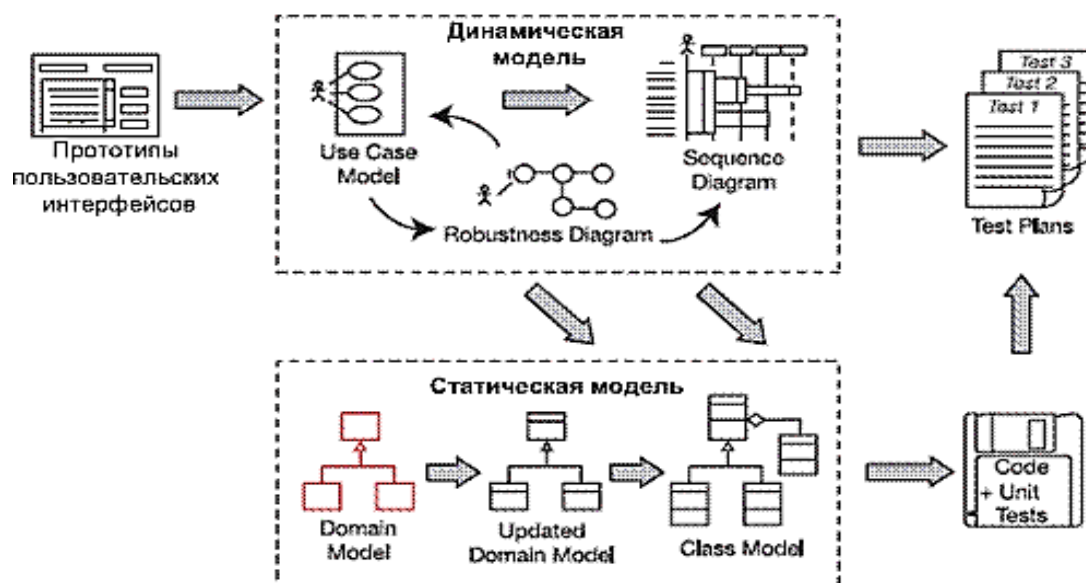


Рис. 2.14 Детальное представление системы

Для этого детальное проектирование модели прецедентов (Use Case model) углубляется, охватывая прецеденты, выполняемые на более глубоких уровнях системы (уровне реализации), показав разработчикам, как должны использоваться отдельные программные модули и компоненты. Для разработчиков дополнительно создаются сценарии испытаний (включая сценарии тестирования), развёртывания (с конфигурированием), взаимодействия с внешними системами, мониторинга работоспособности и т.п.

Из набора сценариев Use Case model, определённых на основе пользовательских историй необходимых интерфейсов, можно создать модель предметной области (Domain model), выделив сущности (объекты и субъекты, ключевые абстракции) и их связи (Рис. 2.14). Сценарии детализируются в виде диаграмм взаимодействия и/или последовательностей. Всё вместе составит логическое представление системы.

Используя согласованную с заказчиком модель предметной области и модель прецедентов, моделируются бизнес-процессы. Из предметной области берутся субъекты (акторы, которые управляют процессом) и объекты (например, передаваемые данные). Прецеденты, в первом приближении, являются шагами бизнес-процесса, со своими пред- и постусловиями, инвариантами и т.п.

Модели бизнес-процессов и модель предметной области после согласования с заказчиком используются для формирования представлений разработчиков: программистов, дизайнеров интерфейса пользователя, проекти-

ровщиков API, разработчиков баз данных и др. Представление разработчика дает каждому достаточную информацию для выполнения ими своей части работы над системой. Модель данных, используемых в системе, значительно отличается от модели предметной области. Модель данных, хранимых в хранилище, отличается от модели обрабатываемых данных.

Правильное представление разработчика фактически является постановкой задачи на разработку конкретного модуля системы, с определением всех его внешних связей и внутренней структуры, с определением его функций и алгоритмов их выполнения. Представление разработчика содержит информацию о компонентах системы, которое формирует представление развёртывания этих компонентов на физическом уровне.

Качественное документирование системы требует разработки сценариев работы с документацией различных заинтересованных лиц. Данные сценарии определяют состав и содержание каждого документа из состава технической и эксплуатационной документации на систему.

Основное назначение описания архитектуры - дать целостную картину системы всем участникам проекта, всем заинтересованным лицам.

2.2.3 Модель требований системы

Требования системы – это возможности, ограничения, функции, которыми должна обладать и соответствовать система, описывают, *что должна делать система*, но не способ реализации.

«Требования это спецификация того, что должно быть реализовано. В них описано поведение системы, свойства системы или ее атрибуты. Они могут служить ограничениями в процессе разработки системы» [12].

Различают функциональные и нефункциональные требования.

Функциональные требования – это действия, функции системы, выполнение которых она должна обеспечивать. Разрабатываются на основе описания входных и выходных сообщений системы, и соответственно определяют входное и выходное поведение системы. Моделируют диаграммами функционального моделирования, диаграммой вариантов использования UML (use-case model) и описанием пользовательских сценариев (use cases, потоков событий). Набор пользовательских сценариев (use cases) описывает варианты взаимодействия между пользователями и программным обеспечением, это требует предварительной идентификации будущих пользователей системы.

Нефункциональные требования описывают окружение системы, накладывают ограничения на дизайн и реализацию, такие как требования производительности, стандарты качества или проектные ограничения.

Организация работ согласно процессу UP представлена на Рис. 2.15. Диаграмма на Рис. 2.16 расширяет поток деятельности по определению требований.

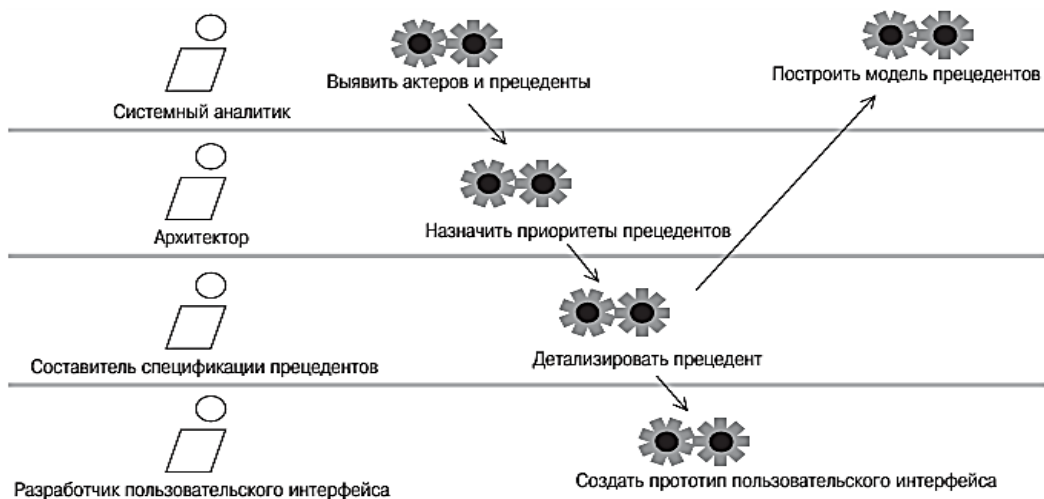


Рис. 2.15 Рабочий потока определения требований [7]

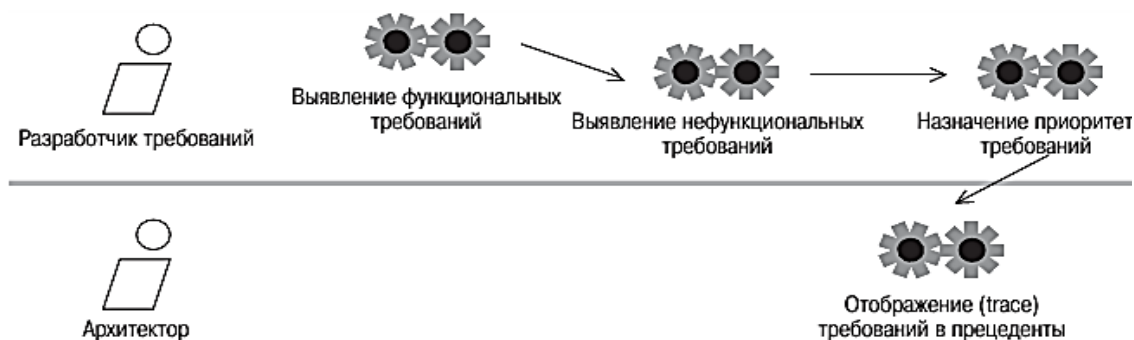


Рис. 2.16 Расширенный рабочий поток определения требований [7]

В итоге полное представление требований реализуют двумя моделями – моделью требований и моделью прецедентов, которые образуют метамодель, Рис. 2.17.

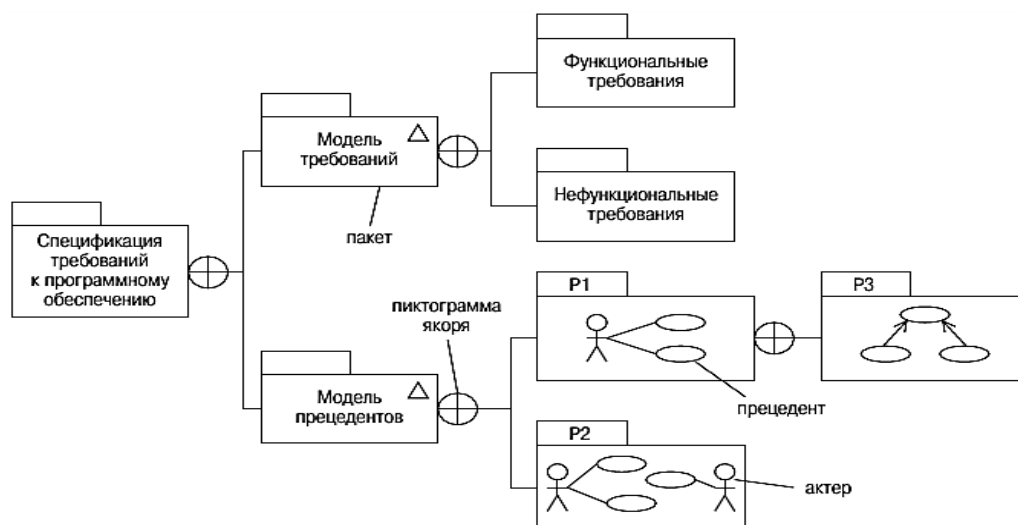


Рис. 2.17 Метамодель требований [7]

Пиктограммы в виде папок – это пакеты UML, они являются механизмом группировки и содержат элементы модели UML. Папки (пакеты UML) используются для организации и группировки взаимосвязанных сущностей. Пиктограмма якоря демонстрирует, что сущность слева от кружка, содержит сущность, находящуюся справа, на другом конце линии.

Модель требований - структурированное описание требований на основе шаблонов. Шаблон SRS или спецификации требований программного обеспечения (Software Requirements Specification, SRS) был разработан для процесса RUP. Шаблон описания такой модели представлен в Приложении 1.

SRS — это структурированный набор требований к программному обеспечению и его внешним интерфейсам. Для корректного описания предлагается формат записи формулировки каждого требования, Рис.2.18.

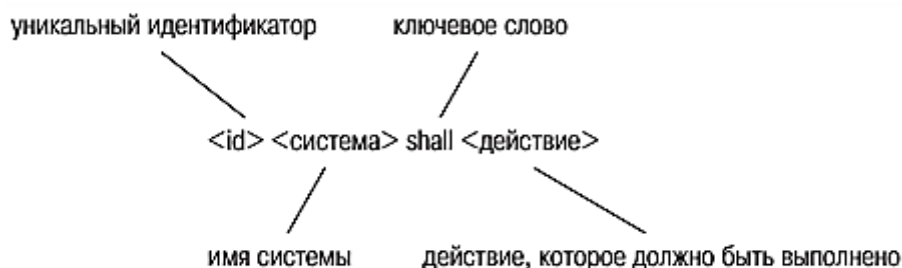


Рис. 2.18 Формат формулировки требований

Каждое требование описано уникальным идентификатором (например, число), ключевым словом «должен» (shall) и описанием действий системы. Применение унифицированного формата упрощает синтаксический разбор требований для инструментов управления требованиями³.

Категории и подкатегории требований в шаблоне представлены моделью ФУНПП+ (FURPS+). Это первые буквы следующих категорий:

- функциональность (Functionality),
- удобство использования (Usability),
- Надежность (Reliability),
- Производительность (Performance),
- Поддерживаемость (Supportability);

"+" подключает следующие категории:

- конструктивные ограничения,
- требования к интерфейсам,
- физические требования.

Подкатегории категорий представлены на Рис. 2.19.

³ Широкое распространение получили такие системы управления требованиями как IBM Rational RequisitePro, Telelogic DOORS, Sybase PowerDesigner и Borland Caliber RM и др.

Конструктивные требования (DES) определяют ограничения проектирования и конструкции системы. Ограничения проектирования (дизайна) представляют собой конструктивные решения, которые введены и должны соблюдаться. Ограничения определяют языки программирования, требования к процессу разработки программного обеспечения, предписанного использования инструментов разработки, архитектурные и конструктивные ограничения, покупные комплектующие, библиотеки классов и т.д.

Функциональность (FUN)	Удобство использования (USA)	Надежность (REL)	Производительность (PER)	Поддерживаемость (SUP)
•наборы функций, •возможности, •безопасность.	•человеческий фактор, •эстетика, •последовательность пользовательского интерфейса, •он-лайн и контекстно-зависимая справка, •мастера и агенты, •пользовательская документация, •учебные материалы.	•доступность (в процентах от имеющегося времени, часов эксплуатации, техническому обслуживанию доступа, ...) •частота / степень интенсивности отказов, •восстанавливаемость, •предсказуемость, •точность, •среднее время наработки на отказ (MTBF).	•пропускной способности (например, транзакций в секунду), •время отклика, •время восстановления, •использование ресурсов (память, диск, процессор, ...).	•тестируемость, •расширяемость, •адаптивность, •ремонтпригодность, •совместимость, •конфигурируемость, •удобство обслуживания, •устанавливаемость, •локализацию (интернационализацию).

Рис. 2.19 Модель категорий требований ФУНПП+

Требования к интерфейсам (INT) определяют интерфейсы, которые должна поддерживать будущая система. Это спецификации, протоколы, порты и логические адреса и т.п., такие, что программа может быть разработана и проверена с учетом требований к интерфейсу. Требования к интерфейсу можно разделить на требования к пользовательскому, аппаратному и программному интерфейсу, Рис. 2.20.

Физические требования (HAR) используют для представления аппаратных требований, таких как необходимая физическая конфигурация сети, минимальные технические характеристики аппаратного обеспечения и др.

Пользовательский интерфейс	Аппаратный интерфейс	Программный интерфейс
• пользовательские интерфейсы, которые должны быть реализованы с помощью программного обеспечения	• аппаратные интерфейсы, которые должны поддерживаться программным обеспечением, в том числе аппаратная конфигурация, логическая структура, физические адреса и ожидаемое поведение, и т.д.	• программные интерфейсы для разработанных компонентов системы и приобретенных компонент; • интерфейсы для взаимодействия с внешними системами.

Рис. 2.20 Виды интерфейсов

Применимые стандарты (STD) описывают любые стандарты и конкретные разделы стандартов, которые применяются к будущей системе: правовые, нормативные, отраслевые стандарты, стандарты качества, совместимости, локализации, соответствия операционным системам и т.д. В стандарте ISO/IEC/IEEE 29148:2011 содержится рекомендации к структуре и методам описания программных требований — «Recommended Practice for Software Requirements Specifications», см. Приложение 2.

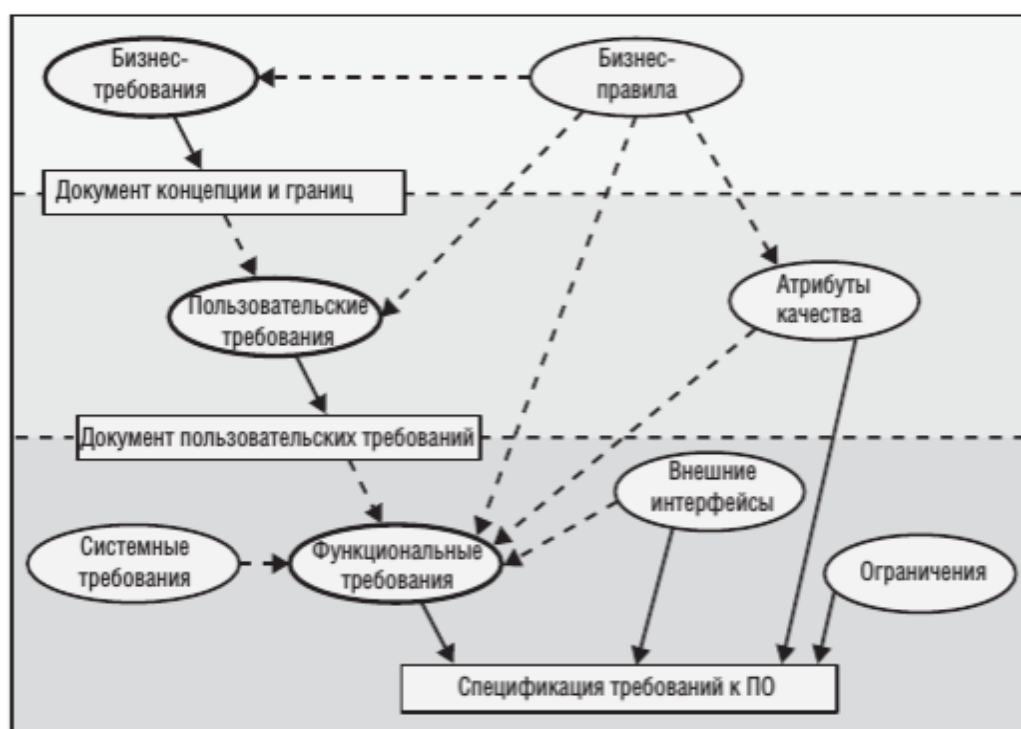


Рис. 2.21 Модель требований Вигерса К.

Современный подход и методологию разработки модели требований предложили Вигерс Карл и Битти Джой в книге «Разработка требований к программному обеспечению» [9]. Предложенная модель требований представлена на Рис. 2.21. Согласно модели, требования к системе состоят из трех уровней: бизнес-требования, пользовательские и функциональные требования. На каждом уровне есть свои нефункциональные требования, Рис. 2.22. Данная модель требований соответствует шаблону SRS.

Для понимания модели авторы вводят словарь терминов, определяющих требования. Словарь представлен в Таблице 2.1.

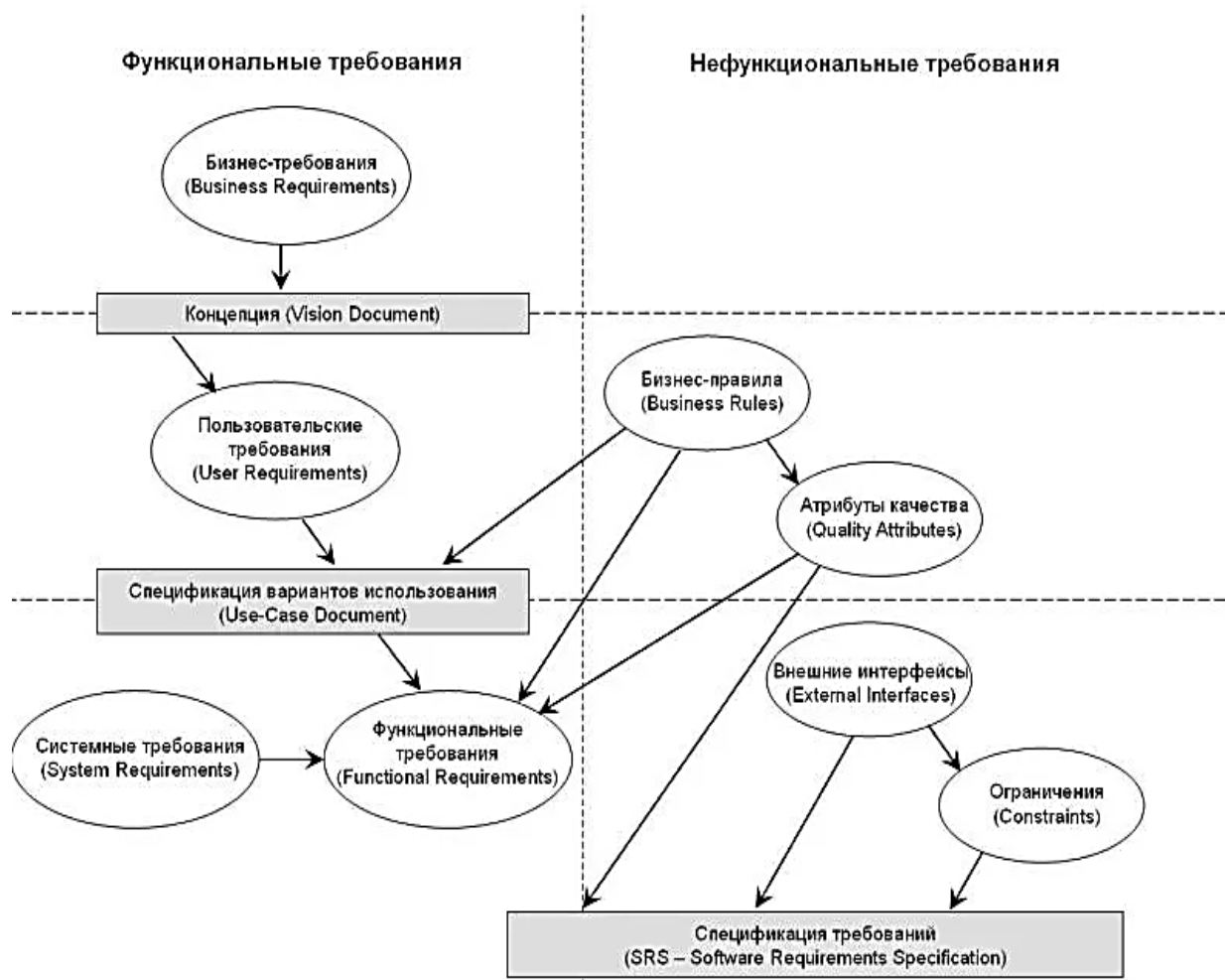


Рис. 2.22 Нефункциональные требования модели Вигерса К.

Словарь терминов, применяемых в сфере разработки требований [9] Таблица 2.1

Понятие	Определение
Бизнес-требование	Высокоуровневая бизнес-цель организации или заказчиков системы.
Бизнес-правило	Политика, предписание, стандарт или правило, определяющее или ограничивающее некоторые стороны бизнес-процессов. По своей сути это не требование к ПО, но оно служит источником нескольких типов требований к ПО.
Ограничение	Ограничение на выбор вариантов, доступных разработчику при проектировании и разработке продукта.
Внешнее требование	Описание взаимодействия между ПО и пользователем, к интерфейсу другой программной системы или устройства.

Характеристика	Одна или несколько логически связанных возможностей системы, которые представляют ценность для пользователя и описаны рядом функциональных требований.
Функциональное требование	Описание требуемого поведения системы в определенных условиях.
Нефункциональное требование	Описание свойства или особенности, которым должна обладать система, или ограничение, которое должна соблюдать система
Атрибут качества	Вид нефункционального требования, описывающего характеристику сервиса или производительности продукта.
Системное требование	Требование верхнего уровня к продукту, состоящему из многих подсистем, которые могут представлять собой ПО или совокупность ПО и оборудования.
Пользовательское требование	Задача, которую определенные классы пользователей должны иметь возможность выполнять в системе, или требуемый атрибут продукта

Обобщенная схема процесса разработки требований на основе спецификаций, согласно стандартам РФ (ГОСТ 34.601-90), представлена в Приложении 3.

Обобщенная схема процесса разработки метамодели требований на основе модели сценариев и модели 4+1 представления архитектуры системы представлена на Рис. 2.23.

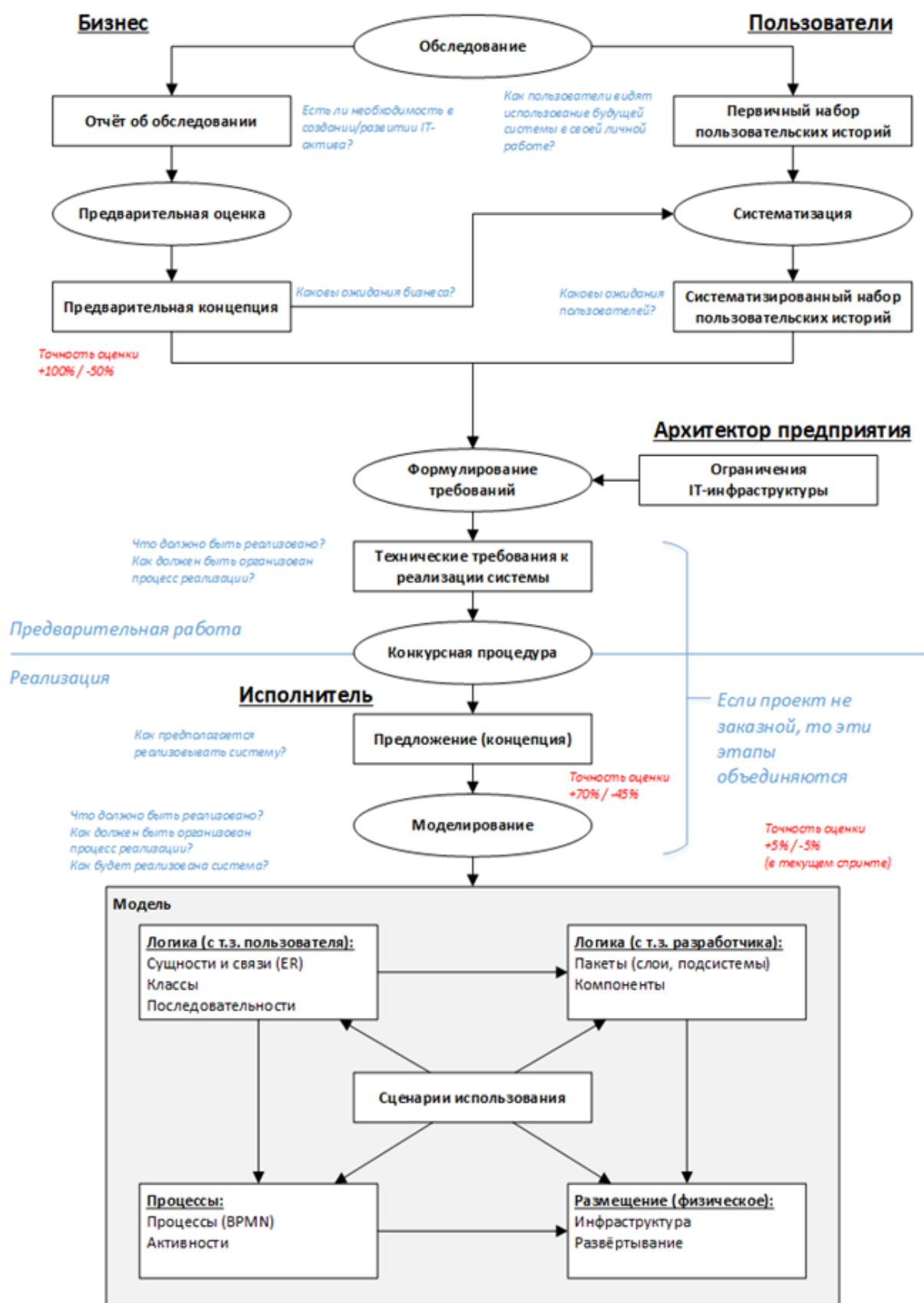


Рис. 2.23 Процесс разработки модели требований на основе модели сценариев [10]

2.2.4 Описание языка моделирования UML

Основное назначение UML – графическое представление различных аспектов разработки программной системы объектно-ориентированной методологии. Основа языка – это структурные и поведенческие диаграммы (Рис. 2.24).

Язык UML – это графический язык моделирования общего назначения, предназначенный для спецификации, визуализации, проектирования документирования всех артефактов, создаваемых при разработке программных систем. Синтаксис и семантика выражаются графическими конструкциями, облегчающими обмен информацией заинтересованными лицами.

Основным способом передачи информации является модель. Диаграмма – это представление модели. Спецификация модели – это описание, документ, подтверждающий то, что модель отвечает требованиям системы. Проектирование в UML позволяет разработать архитектурные решения и шаблоны, которые автоматически генерируются в программный код заданного языка программирования на основе концепции MDA⁴.



Рис. 2.24 Классификация диаграмм UML по назначению

⁴ MDA (Model Driven Architecture) - архитектура, управляемая моделью, продвигаемый консорциумом OMG модельно-ориентированный подход к разработке программного обеспечения.

UML состоит из трех элементов [8] – сущности, отношения, диаграммы (Рис. 2.25). Сущности – это ключевые абстракции. Отношения – это связи между сущностями. Диаграммы – способ визуализации набора сущностей, они детализируют состав, структуру и поведение моделируемой системы. Виды и назначение диаграмм представлены в Таблице 2.2 [11].

Таблица 2.2

Назначение диаграмм UML

Диаграмма	Назначение
Прецедентов/ вариантов использования	Сценарии взаимодействия пользователя с системой
Классов	Классы, свойства и отношения
Составных структур	Декомпозиция класса во время выполнения
Объектов	Вариант конфигурации экземпляров классов в момент времени выполнения
Конечных автоматов/ состояний классов	Как события изменяют объект в течение его жизни
Пакетов	Иерархическая структура времени компиляции
Компонентов	Структура и взаимосвязи между компонентами
Взаимодействия	Взаимодействие между объектами; акцент на связях
Временная	Взаимодействие между объектами; акцент на синхронизации
Последовательности	Взаимодействие между объектами/классами; акцент на последовательности действий
Обзора взаимодействий	Комбинация диаграммы последовательности и диаграммы деятельности
Деятельности	Процедурное и параллельное поведение системы
Развертывания	Развертывание артефактов в узлы

Нотация UML определяет общие правила и механизмы, которые применяются в целом к моделированию элементами данного языка. К ним относятся спецификации, декорации и механизмы расширения и классификации.

Модели UML – это диаграммы и текстовое описание их смысла (семантики). Спецификации – это текстовое описание семантики элемента на заднем плане (semantic backplane).

Декорации – это дополнительная визуализация элемента, которая подчеркивает важные характеристики модели и делает диаграмму более понятной и удобочитаемой.

Механизм классификации основан на классификаторах – классификатор/экземпляр и интерфейс/реализация. Пусть имеем абстрактное описание типа сущности - средство передвижения, и конкретный экземпляр этой абстракции - мой автомобиль. Тип сущности – это классификатор, а конкретные сущности – экземпляры этого типа, что основано на концепции ООП «класс – объект класса».

Существует три основных вида классификаторов: классы, типы данных и интерфейсы. Однако к классификаторам также относятся ассоциации, акторы, сигналы, компоненты, узлы и подсистемы. Существует около 30 классификаторов.

Механизмы расширения – это ограничения, заданные логическими условиями в фигурных скобках, которые моделируемый объект должен поддерживать в состоянии истины и механизмом стереотипов атрибутов.



Рис. 2.25 Состав UML

Преимущество UML заключается в возможности разностороннего описания системы: структура – статическое описание системы; поведение – описание системы в динамике; архитектура – общий взгляд на систему.

Основные диаграммы UML представлены в Приложении 4. Для их изучения и корректного применения необходимо изучить дополнительно руководства среды разработки и учебники по языку.

2.2.5 Методы объектно-ориентированного анализа

Основная задача на этапе анализа предметной области – это выявление ключевых абстракций, составляющих **словарь предметной области**.

Ключевая абстракция (Key Abstractions) – это сущность, класс или объект, который входит в словарь предметной области. Главная ценность ключевых абстракций заключена в том, что они определяют границы предметной области системы: выделяются те объекты и операции с ними, реализация которых существенна и отбрасываются лишние объекты.

На этапе анализа ключевые абстракции выявляются, а затем последовательно исследуется их внутренняя структура, поведение, взаимодействие и возможная физика представления абстракций (Рис.2.26).

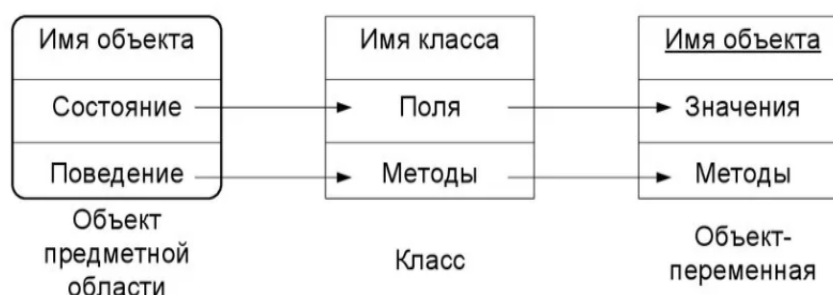


Рис. 2.26 Преобразование ключевой абстракции в программный объект

Методы и способы выявления ключевых абстракций предметной области:

- анализ предметной области;
- классический подход;
- анализ поведения объектов;
- анализ вариантов использования;
- CRC-карточки;
- метод неформального описания.

Анализ предметной области – это выявление ключевых абстракций и отношений путем построения модели предметной области. Способы построения модели:

- а. построение модели предметной области системы на основе совокупности сценариев работы системы, представленных будущими пользователями системы – пользовательские истории;
- б. построение модели предметной области на основе сравнительного анализа функционала готовых систем в данной области с участием заинтересованных лиц.

Классический подход к построению модели предполагает классификацию, упорядочивание объектов предметной области по классификационным признакам (одно или несколько свойств), выбранных для опре-

деления сходства или различия между этими объектами. Объекты разбивают на непересекающиеся множества в зависимости от наличия или отсутствия некоторого признака. Например, на Рис. 2.27 представлены классификации по признаку: 1- идентификация фигуры множеством точек, 2- принадлежность к транспортным средствам.

В классы и объекты могут быть выбраны:

- предметы (транспортные средства, учебная дисциплина);
- роли (работник, преподаватель, клиент);
- события (выбор учебной дисциплины, запрос);
- взаимодействие (оплата, встреча).

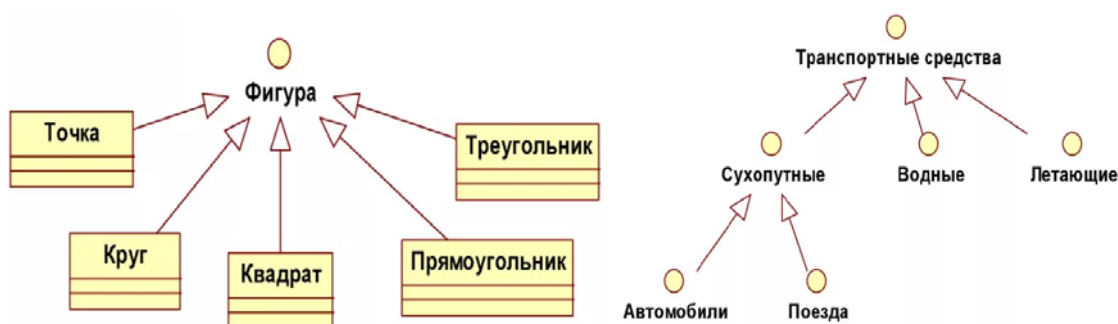


Рис. 2.27 Классификация объектов

Анализ поведения, частный случай классического подхода, где основной декомпозиции будет признак одинакового поведения объектов, характеризующий существенные для системы требования. Классы формируются на основе групп объектов, демонстрирующих одинаковое поведение. В результате анализа объединяют объекты, имеющие сходное поведение. Иерархию классов строят на основе того, что каждый класс дополняет общий список поведенческих операций, отражающих функциональное поведение системы.

CRC-карточки (Class-responsibility-collaboration card, карта «Класс-Ответственность-Кооперация») — метод, предназначенный для определения классов и их взаимодействий с другими классами (Рис. 2.28).

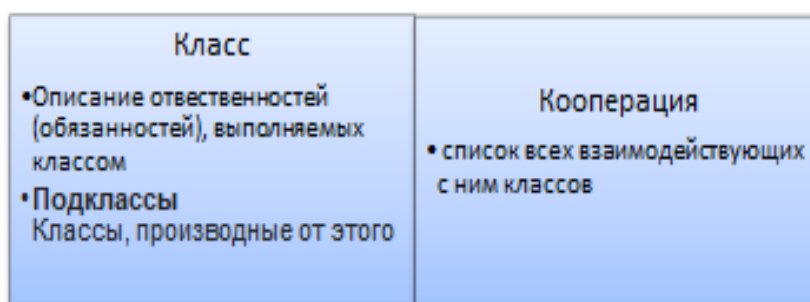


Рис. 2.28 Шаблон CRC-карточки

Использование карточек уменьшает сложность на этапе анализа. Используя документ (техническое задание), описывающий предметную область задачи, на каждую нужную сущность заводится класс, имя которого записывается в шапке карточки. В левой колонке карточки записывается ответственность класса, то есть что класс должен делать. После анализа ответственности класса, можно часть ответственностей с одного класса передать другому классу, или выделить в новые, более детальные классы. Ответственности - это операции будущего класса. В правой колонке карточки вписывают классы, с которыми взаимодействует класс (Рис. 2.29).

Набор карточек – это словарь ключевых абстракций предметной области.

Клиент	
Вставляет карточку в банкомат Вводит пароль Указывает тип операции Вводит сумму Получает деньги Вынимает карточку	Банкомат
Банкомат	
Отображает информацию для клиента Передаёт информацию в банк Выдает купюры Распечатывает счет	Клиент Банк Служба безопасности банка
Служба безопасности банка	
Проверяет пароль Проверяет подлинность карточки Идентифицирует клиента Следит за правильностью транзакций	Банк Банкомат
Банк	
Проверяет возможность выдачи средств Сообщает о наличии денег Выдает информацию об остатке Хранит информацию о счете клиента	Банкомат Служба безопасности банка

Рис. 2.29 Пример анализа на основе CRC-карточки

Анализ вариантов использования – это подход, который можно успешно сочетать с любыми методами анализа для понимания функциональности ключевых абстракций. **Вариант использования, прецедент** – это сценарий или пример использования будущей системы пользователем.

Для выявления вариантов использования пользователи, эксперты и разработчики перечисляют примеры использования – пользовательские истории – существенные для работы системы, не углубляясь в детали. Затем детализируют пример, последовательно формируя поток событий данного сценария. При этом необходимо описать как реагирует система на каждое событие и какие ключевые абстракции участвуют в этом сценарии. По мере разработки системы набор вариантов использования системы увеличивается с учетом исключительных ситуаций и расширения проектных классов. Варианты использования моделируют функциональные требования к системе.

Сочетая данный метод с CRC-карточками по ходу анализа сценария, можно создавать или дополнять CRC-карточки. Раскладывая CRC-карточки согласно варианту использования можно построить модели взаимодействия объектов. Такая модель демонстрирует возможную иерархию классов, а также поток событий и сообщений между объектами.

Метод неформального описания предполагает, что источником ключевых абстракций будет описание предметной области и модель требований системы, где выделяют основные существительные, именные группы и глаголы. По мере уточнения перечня должно происходить следующее:

- ✓ имена существительные и именные группы становятся объектами и атрибутами; родительный падеж показывает, что имя существительное должно быть атрибутом, а не объектом;
- ✓ глаголы и глагольные группы становятся операциями и ассоциативными связями между классами.

Далее следует отсеять из списка кандидатов на звание класса ненужные (избыточные, несущественные) и непригодные (слишком расплывчатые или представляющие концепции, лежащие за рамками модели) элементы.

Механизмы поиска, выбора и уточнения ключевых абстракций.

Выявления ключевых абстракций может идти путем открытия и путем изобретения. Открываются абстракции путем изучения технического задания, документов описания предметной области системы, консультациями с заинтересованными лицами. Создавая новые классы и объекты, разработчик изобретает абстракции и они не обязательно являются частью предметной области, это классы, отвечающие за функционирование системы. Например, пользователь банкомата говорит «счет», «снять», «положить»; эти термины – часть словаря предметной области. В процессе проектирования системы, изобретаются классы «форма», «список», «монитор» и т.д.

Эффективный способ выделения ключевых абстракций – сведение задачи к готовым решениям. Готовая система классов называется шаблоном проектирования, или паттерном (design pattern) - повторяемая проектная конструкция, представляющая собой решение часто возникающей задачи.

Целью объектно-ориентированного анализа является трансформация функциональных требований к ПО в предварительный системный проект и создание стабильной основы архитектуры системы. В процессе проектирования системный проект "погружается" в среду реализации с учетом всех нефункциональных требований.

2.2.6 Методика анализа и проектирования

1. Модель прецедентов.

Цель - создать модель прецедентов Use Case model и отобразить с помощью одноименных диаграмм UML архитектурное представление Scenarios.

Основные задачи:

А. Используя словарь предметной области, выделить акторов и информационные сущности: акторы – это субъекты, которые управляют и иницируют процессы, сущности – это объекты данных. Акторами могут быть пользователи, время и внешние системы.

В. Построить диаграмму прецедентов, где представлены только акторы. Преобразовать эту диаграмму, классифицируя акторов иерархически, обобщая их по возможности на основе связи Generalization.

С. Для каждого функционального требования модели требований создать пакет: сколько функций – столько диаграмм пакетов. Для каждого функционального пакета формируем свою диаграмму прецедентов. Пакеты принадлежат уже логическому представлению архитектуры Logical view.

Д. Разработать последовательно диаграммы прецедентов для всех выявленных сценариев функционального пакета на основе пользовательских историй. На этапе разработки модели требований выполняется систематизация пользовательских историй, возникают сценарии использования, см. схему Рис.2.23. Истории сортируются по соответствующим функциональным пакетам. Выбираем пользовательские истории, которые нацелены на получение одного и того же результата разными путями. История с самым оптимальным путём достижения целевого результата становится базовым сценарием прецедента. Остальные истории – альтернативные сценарии того же прецедента.

Е. Описать в виде потока событий базовые и альтернативные сценарии, указав предусловия и постусловия каждого потока. Поток событий должны представлять действия системы по обработке сценария.

На Рис. 2.30 представлен UP процесс разработки модели прецедентов [7]. Результат - модель прецедентов, распределенная по функциональным пакетам и потоки событий для всех прецедентов.

Рассмотрим пример. Пусть есть требования к системе, автоматизирующей регистрацию студентов на курсы в предстоящем семестре. После систематизации пользовательских историй мы получили следующую модель:

- пользователи системы авторизуются регистратором на предстоящий семестр и получают сертификат электронной подписи для подачи заявок на курсы дисциплин (этот прецедент рассматривать не будем);
- неавторизованный пользователь может выполнить вход в систему для просмотра списка курсов;
- авторизованный пользователь может записаться на курс и редактировать записи на курсы в ограниченный срок регистрации, просмотреть свои записи на курсы, экспортировать их в pdf-файл, по окончании срока регистрации должен подписать с помощью сертификата электронной подписи, так как система в дальнейшем, например, выставит счета за обучения через систему оплаты.

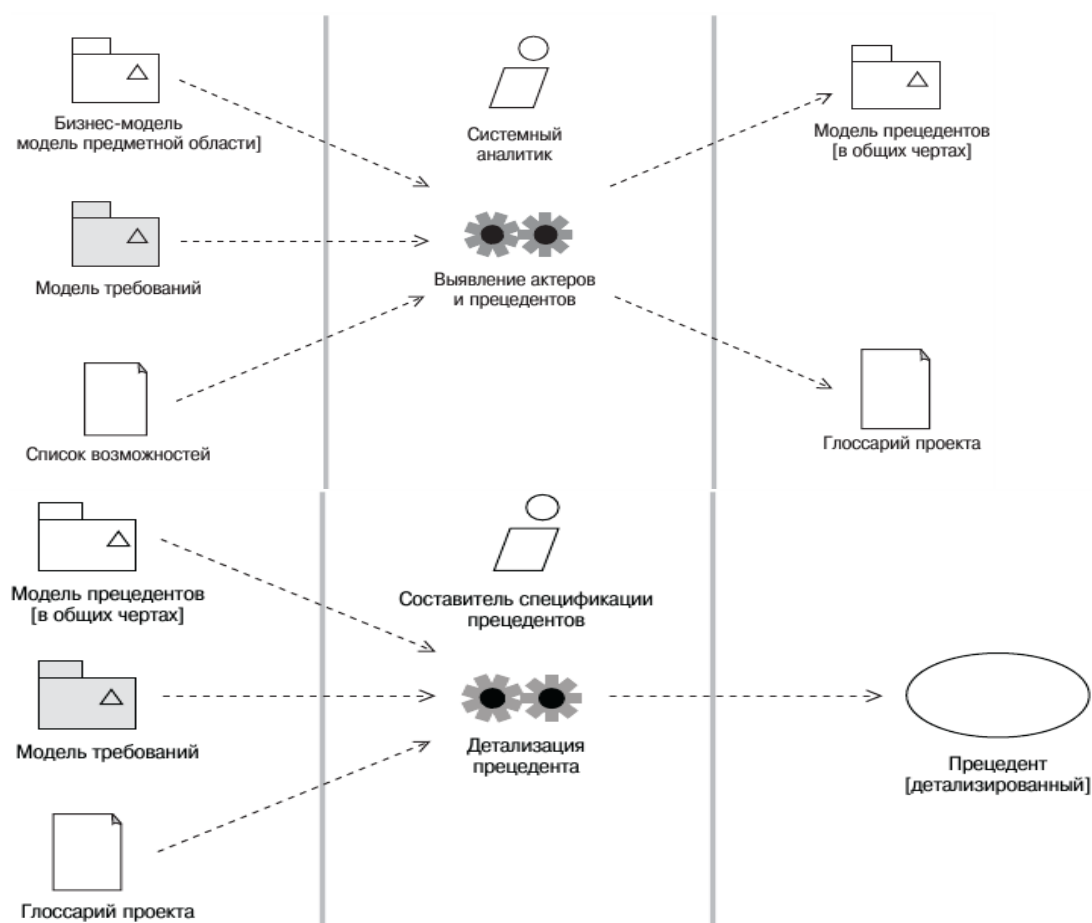


Рис. 2.30 Процесс UP разработки модели прецедентов

Основные моменты:

- модель акторов: все реальные пользователи наследуются от некоторого абстрактного базового актора AnyUser (Рис. 2.31). Если в будущем появится некая функция, которая должна быть доступна всем пользователям, то соответствующий прецедент будет связан именно с этим базовым актором. Если бы авторизованный пользователь наследовался напрямую от

неавторизованного, ему была бы навязана операция типового входа в систему, что нарушает принцип разделения интерфейсов – один из базовых принципов проектирования, который говорит, что пользователи не должны зависеть от методов, которыми они не пользуются. Такой подход возможен и для внешних систем.

- пакеты: прецеденты разделены на два пакета в рамках одного требования: первый пакет связан с действиями над учётными записями, второй пакет работает с агрегатом, состоящим из заявки на курсы и её записей.

- связи между прецедентами. "Включение" (include) - это обязательно выполняемый безусловный вызов одного прецедента другим из любого сценария. Без такого вызова результат вызывающего прецедента не может быть достигнут. На диаграмме стрелка всегда направлена в сторону включаемого прецедента. Связь показывает, что включающий прецедент включает в себя включаемый прецедент. В примере изменение заявки на курсы всегда требует сохранения заявки. Если изменения не будут сохранены, то весь смысл создания или редактирования заявки будет потерян. "Расширение" (extend) - это вызов, выполняемый по определённому условию. Связь показывает, что некоторый прецедент расширяет другой прецедент при выполнении заданного условия. Стрелка на диаграмме всегда направлена от расширяющего прецедента к расширяемому. В рассматриваемом примере подпись заявки не является обязательной. Она требуется только в случаях, когда срок регистрации окончен или когда студент определился со списком.

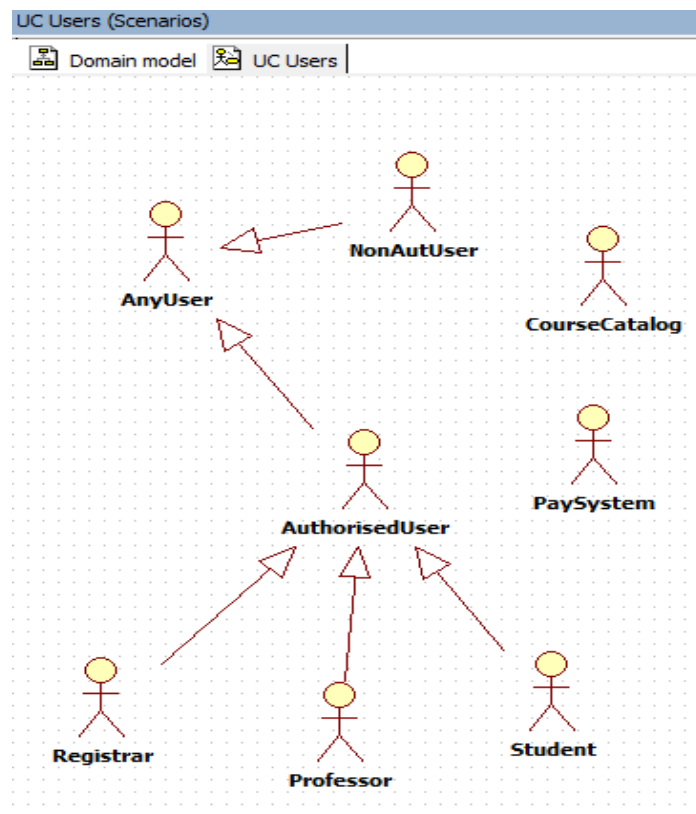


Рис. 2.31 Диаграмма акторов системы

– количество диаграмм: для примера диаграмм должно быть минимум три – диаграмма акторов, диаграмма пакета Account и диаграмма пакета CoursesLog. Для примера представление объединило все прецеденты на одной диаграмме (Рис. 2.32).

В реальной системе всё сложнее, но если правильно систематизировать пользовательские истории, то задача будет легко раскладываться в набор прецедентов с простыми зависимостями.

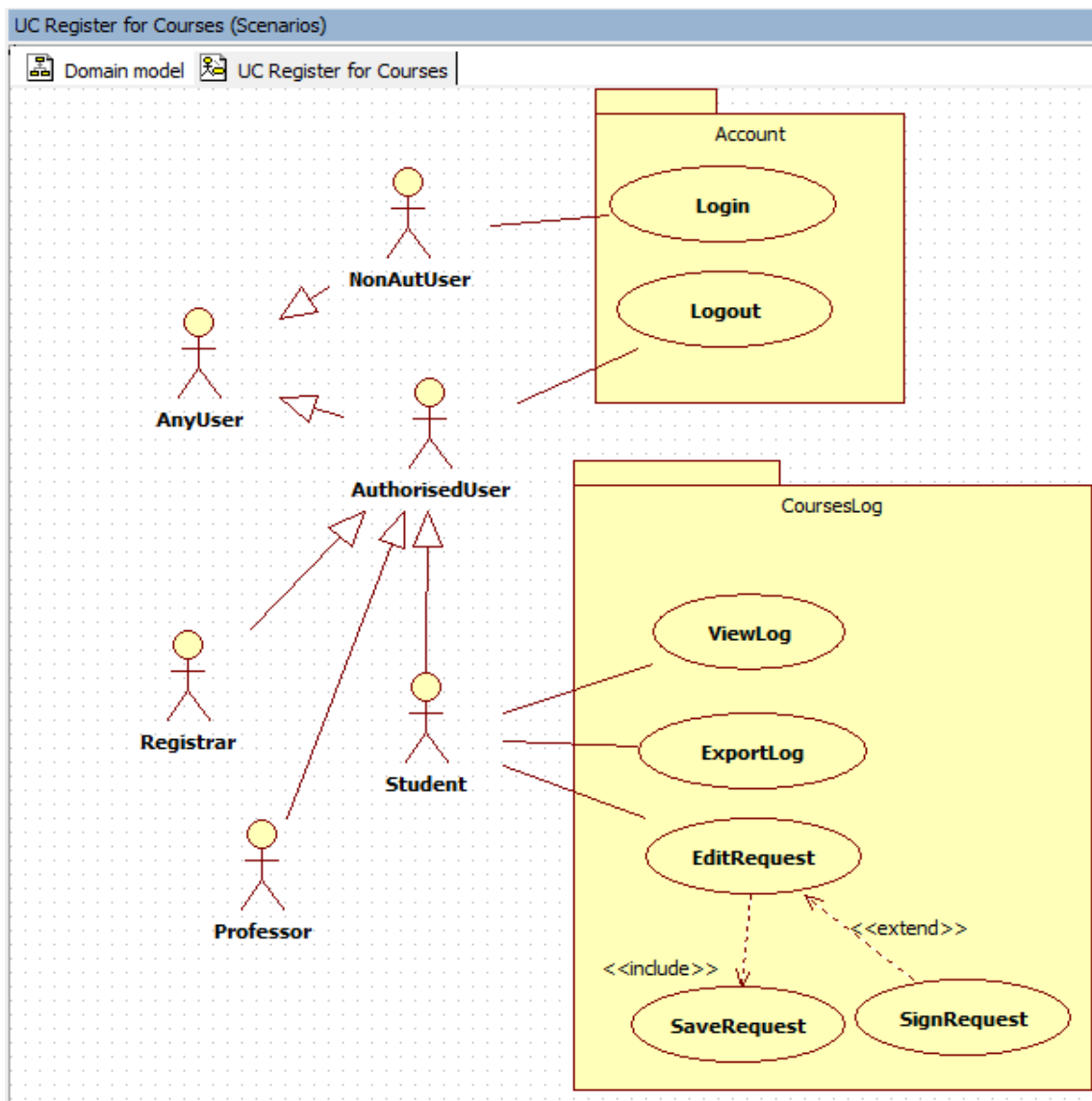


Рис. 2.32 Модель прецедентов, распределенная по функциональным пакетам

2. Модель предметной области (domain model).

Цель – разработать доменную модель, представив посредством диаграммы классов UML основную модель логического представления системы **Logical view**. Представляет точку зрения пользователей системы и является основной аналитической моделью предметной области системы.

Для разработки требуется решение следующих задач:

- 1) собрать и систематизировать пользовательские истории в ходе разработки модели требований, согласно схеме на Рис. 2.23.
- 2) разработать модель прецедентов, п.1 - Модель прецедентов.
- 3) все сущности, идентифицированные в модели требований и сценариях модели прецедентов, представить в модели предметной области и определить для них исходный состав атрибутов, операций и взаимосвязи (Рис. 2.33).

Если в среде разработки есть модель предметной области (domain model), то лучше использовать данную модель, но можно отображать её посредством диаграммы классов UML. Альтернативой может быть ER-модель (модель "сущность-связь"). В этом случае отличается только нотация, а принципы проектирования модели предметной области остаются теми же.

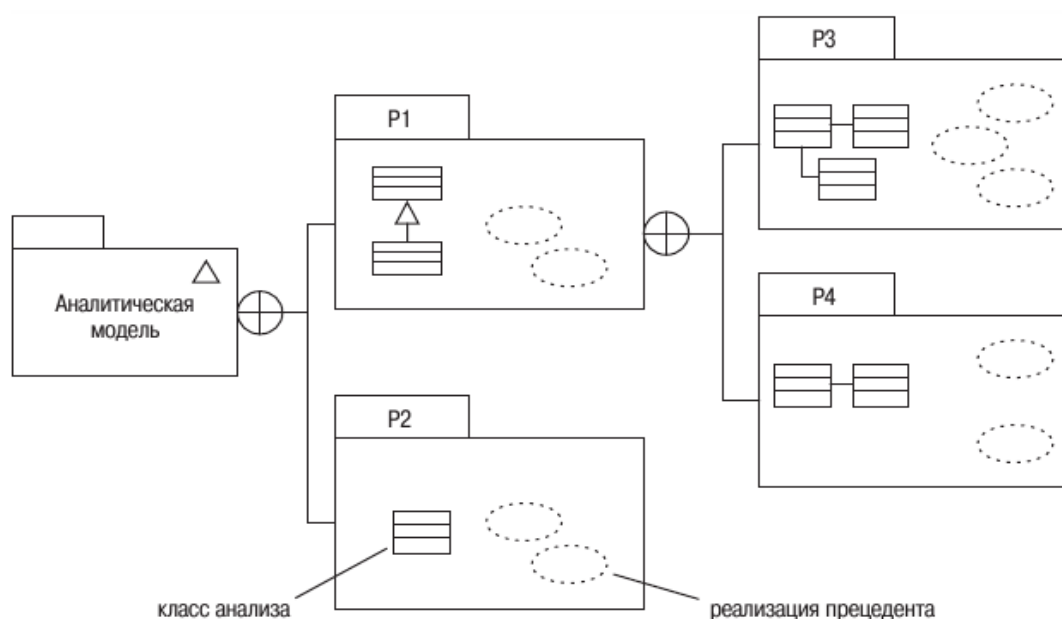


Рис. 2.33 Разработка модели предметной области

Рассмотрим способ, которым можно создать модель предметной области на примере.

Выявленные сущности – ключевые абстракции словаря могут быть двух видов: субъекты и объекты. Субъекты - это сущности, которые управляют другими сущностями. Объекты - это сущности, которые являются управляемыми. Субъектами являются акторы/действующие лица.

А. Создать классы всех акторов модели прецедентов в новом пакете Actors с учётом их иерархии. Актор AnyUser становится абстрактным классом AnyUser, так как реальных экземпляров этого класса нет: все пользователи относятся к дочерним классам. Остальные акторы также становятся классами со стереотипом сущность (entity), Рис. 2.34.

В. Сущности, участвующие в сценариях прецедентов из пакета Account (Рис. 2.35), в виде классов помещаются в этот же пакет. Всего в пакете оказывается три сущности:

учётная запись Account, связанная с авторизованным пользователем;
роль пользователя RoleUser, которая может быть назначена учётной записи, причём ролей у одного Account может быть несколько;

сертификат Certificate, выдаваемый пользователю для подписи заявок; сертификатов у одной учётной записи может быть несколько: пока один ещё действует, пользователь получает новый.

В упрощённой модели у пользователя только одна учётная запись. В реальности пользователь может иметь несколько учётных записей, например, одну – как частное лицо, а другую – как представитель некой организации. Тогда структура и связи внутри пакета Account будут несколько иными.

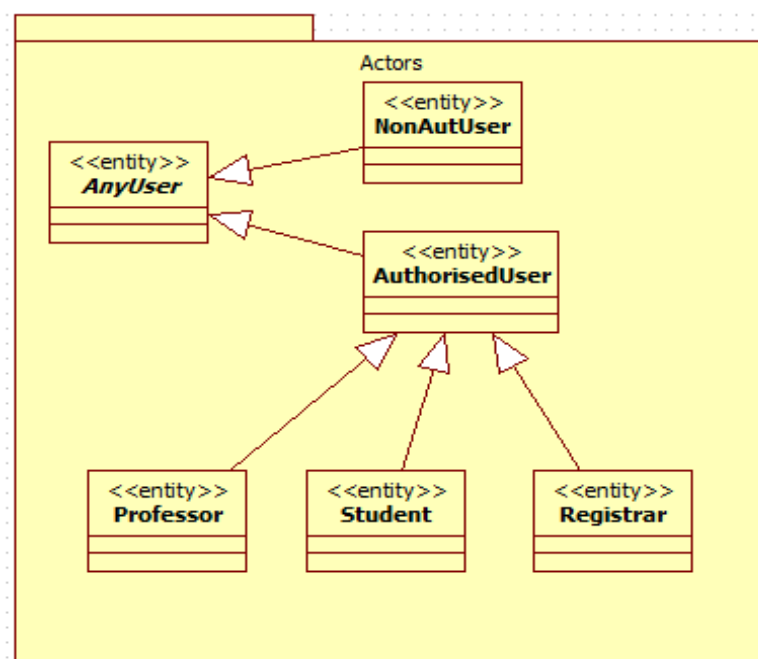


Рис. 2.34 Пакет Actors

С. Создать и поместить классы сущностей прецедентов пакета CoursesLog в это же пакет. Было создано несколько классов сущностей, представляющих три агрегата.

Агрегатом называется структура (дерево) взаимосвязанных сущностей, изменяемых как единое целое. Агрегат состоит из одной или более сущностей, причём в случае нескольких сущностей, они входят в состав друг друга (связь агрегация или композиция).

Разработаны три агрегата: курсы и записи о них - более сложный агрегат, чем заявка на регистрацию курсов и расписание студента.

Изменение записи курса CourseOffering влечёт изменение всей заявки в виде перерасчёта состава групп, нагрузки профессора, счета, обновления агрегированного состояния, времени последнего обновления, определения срока и состава будущего расписания и т.п. На этом этапе в состав классов можно внести исходные операции и атрибуты. Эскизно это спрятано за агрегатом заявки.

Изменение расписания Schedule не влечёт за собой каких-либо явных изменений в нем, просто изменится значения атрибута списка курсов. Расписание будет ссылаться на ту же сущность CourseOffering, операции которой выполняют все пересчеты и изменения.

Подача или изменение заявки на курсы не влияет на расписание до тех пор, пока не закончится срок регистрации на курсы. Изменения в ней появятся только тогда, когда срок регистрации закончится и необходимо будет зафиксировать невозможность изменения ее изменения.

Правильно раскладывать сущности по пакетам согласно числу агрегатов. Так пакет CoursesLog содержит три подпакета Courses, Request и Schedule, так как информацию о курсах пользователь просматривает на одной форме, заявку вводит в другой, а информацию о расписании просматривает в третьей. Агрегаты создают функциональные границы, которые закладываются в модель в виде структуры пакетов, которая потом транслируется в структуру программных модулей. Это касается обработки агрегатов на уровне бизнес-логики, структуры хранения данных и др.

Диаграмма модели предметной области представлена на Рис. 2.35.

D. Определить отношения между классами предметной области, если они существуют. Отношения между классами – это ассоциация, обобщение, агрегация и зависимость. Отношения определяют семантические (значимые) связи между элементами модели и являются способом объединения сущностей в UML. Отношения – это способы взаимодействия объектов классов путем обмена сообщениями, то есть, вы отвечаете на вопрос: будет класс взаимодействовать с другим классом и какое назначение этой связи для реализации сценария.

Основным отношением между классами на этапе анализа являются ассоциации. Отношениям ассоциации по возможности нужно давать имена и указывать кратность. Связь говорит, что отношение будет, а имя связи указывает назначение связи.

При проектировании и реализации классов все связи ассоциации преобразуются в другие типы отношений – агрегацию, зависимость, наследование. Ассоциация не программируется, а другие отношения образуют известные программные конструкции.

Синтаксис описания отношений в UML четко разработан, подробное описание смотрите в руководствах соответствующих сред разработки.

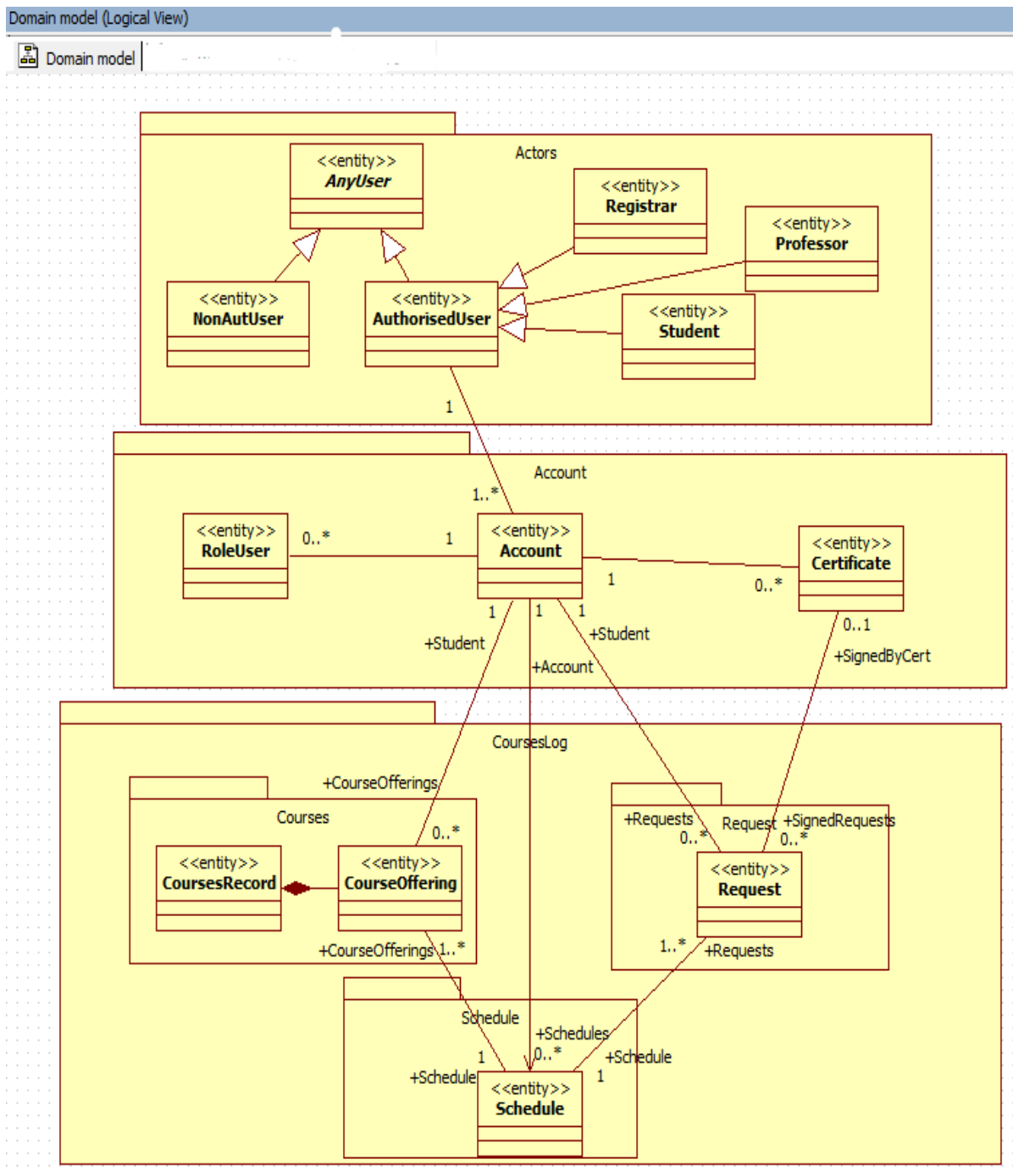


Рис. 2.35 Модель предметной области системы

Е. Определить атрибутивный состав классов и исходный набор очевидных операций. В результате в модели предметной области могут появиться новые сущности, не всегда упоминаемые в сценариях:

- ✓ сущности-значения - типы данных, такие, как перечисления, единицы измерения и т.п.;
- ✓ сервисы предметной области - некоторые внешние источники бизнес-данных, независимо от их реализации и реализации взаимодействия с ними;

- ✓ события предметной области - состояния предметной области, при достижении которых происходит определённая реакция: от оповещения акторов до автоматического запуска или останова бизнес-процессов;
- ✓ сущности, которые являются составляющими частями сущностей, упоминаемых в сценариях.

Примером события может стать поступление новой заявки на курс. Это сразу потребует реакции системы проверки доступности курса для данного студента, изменения расписания, оповещению профессора о новом студенте, и т.д. Примером сервиса можно рассматривать работу с автоматическим формированием счета для системы оплаты в момент закрытия регистрации.

Появление сервисов, событий и составных сущностей, может привести к пересмотру сценариев. Этот обратный процесс всегда присутствует в проектировании. В средах разработки для этого применяется проверка модели⁵ и соответствующий инструмент, отслеживающий корректность элементов в единой архитектурной модели.

Г. Для полного представления точки зрения пользователя необходимо разработать прототипы пользовательских интерфейсов, то есть пользователю нужно увидеть макеты экранных форм и отчётов, формируемых системой.

При описании архитектуры в целом, архитекторы исходят из соображений необходимости и достаточности информации для конкретного заинтересованного лица и его способности разобраться в предлагаемых схемах и диаграммах. Не обязательно формировать все известные диаграммы UML. Нужно сосредоточиться только на том минимуме, который хорошо проиллюстрирует архитектуру будущей системы, сделает её понятной для пользователя.

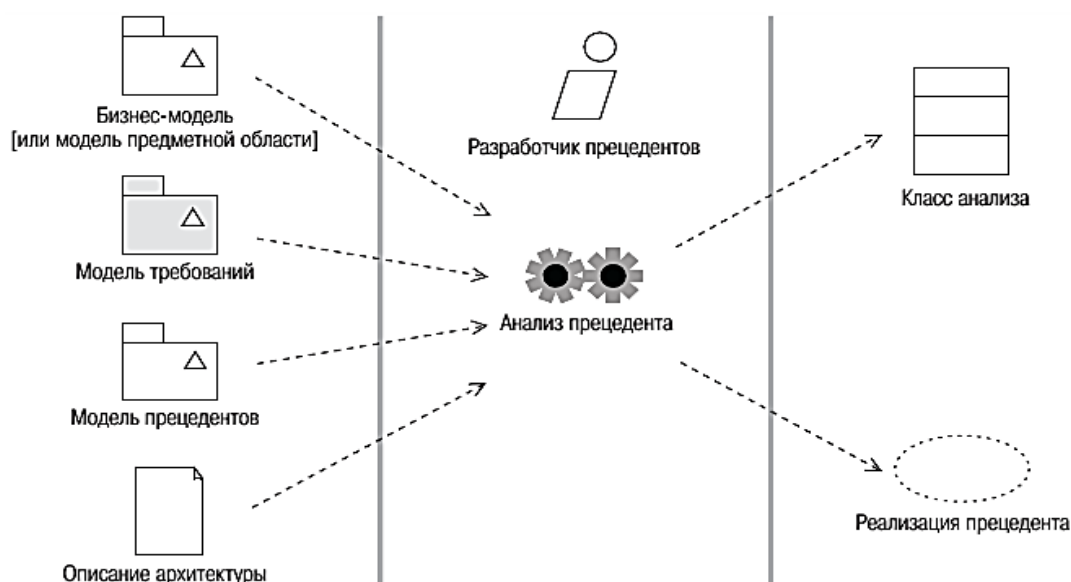


Рис. 2.36 Процесс реализации прецедента

⁵ StarUML – опция [Model] -> [Verify Model...].

Построив модель предметной области и классы анализа, следующим шагом в анализе будет поиск **реализаций прецедентов** - это наборы классов, реализующих поведение, определенное в прецеденте, Рис. 2.36. Статическая структура системы хорошо описывается моделью предметной области, а поведение системы - моделью поведения и набором диаграмм состояний, деятельности, взаимодействия и последовательности.

3. Диаграммы последовательности для сценариев

Цель - разработать модели поведения системы, представив посредством диаграмм последовательности (Sequence Diagram) и других диаграмм взаимодействия UML. Модели поведения системы – это логическое представление архитектуры системы **Logical view**. Представляет точку зрения пользователей системы.

Модели поведения системы моделируют поведение сущностей предметной области, их взаимодействие в динамике, изменение состояния и т.п. Для этого служат модели состояний и модели взаимодействия, выражаемые в UML-диаграммах последовательностей, временных диаграммах, диаграммах взаимодействия и диаграммах состояний классов.

Диаграмма прецедентов статична. Сформированная на её основе модель предметной области тоже статична: она содержит только сущности и связи. Необходимо показать, как прецеденты работают в контексте классов и их связей. Для этого используют диаграммы последовательностей.

Разработка диаграммы последовательности решает сразу несколько сложных для разработчика задач:

1) **Идентификация классов**, участвующих в реализации потоков событий сценария. В потоках событий прецедента используют как минимум классы трех типов (Stereotype):

- *граничные классы (Boundary)* - служат посредниками при взаимодействии акторов с системой. Как правило, для каждой пары «актор - прецедент» определяется один граничный класс. Типы граничных классов: пользовательский интерфейс (обмен информацией с пользователем, без деталей интерфейса - кнопок, списков, окон), системный интерфейс и аппаратный интерфейс (обмен с внешними системами, терминальным оборудованием, используемые протоколы, без деталей их реализации);

- *классы-сущности (Entity)* - представляют собой классы предметной области системы. Источники классов-сущностей: модель предметной области, ключевые абстракции словаря системы, потоки событий;

- *управляющие классы (Control)* - обеспечивают координацию поведения объектов в данном прецеденте или системе в целом. Могут отсутствовать в некоторых вариантах использования, ограничивающихся простыми манипуляциями с хранимыми данными. Как правило, для каждого прецедента определяется один управляющий класс. Примеры управляющих классов: менеджер транзакций, координатор ресурсов, обработчик ошибок.

2) Идентификация операций классов, участвующих в одном или нескольких прецедентах за счет моделирования и распределения поведения между классами прецедента. Операции в классе появляются путем преобразования сообщений между классами диаграммы последовательностей в операции.

Типов классов в средах разработки (Stereotype) много, не обязательно классы прецедента будут относиться к этим стереотипам, но для начала разработки диаграммы последовательности можно опираться на этот минимальный набор. В процессе итеративной разработки один управляющий или граничный класс может быть использован для нескольких прецедентов или наоборот большой класс будет разбит на несколько. Какие-то агрегаты могут быть представлены показавшими эффективность шаблонами проектирования (Pattern) т.д. Это будет определяться логикой проектирования и реализации в ходе итераций разработки.

В первую очередь строится диаграмма (одна или более), описывающая основной поток событий и его подчиненные потоки. Для каждого альтернативного потока событий строится отдельная диаграмма.

Рассмотрим пример формирования только одной диаграммы, поскольку для иллюстрации принципа формирования этого достаточно. В реальности может потребоваться несколько разных диаграмм, и они могут быть весьма нетривиальными.

Самый сложный прецедент в модели примера связан с редактированием заявки на курсы. Базовый сценарий использования этого прецедента предполагает, что авторизованный пользователь может вызвать форму заявки, заполнить её, затем подписать данные заявки и сохранить заявку. Основные логические аспекты сценария:

1) Пользователь может не создавать новую заявку, а отредактировать имеющуюся, указав её идентификатор. Тогда форма заявки уже будет заполненной данными из сохранённой ранее заявки. Это - альтернативный сценарий данного прецедента.

2) Пользователь может подписать заявку, а может этого не делать. Это опциональный прецедент, который связан с редактированием заявки связью "extend".

3) Пользователь должен сохранить изменения, и для этого был создан отдельный прецедент для сохранения заявки, который связан с рассматриваемым прецедентом связью "include".

В рамках прецедента исследуем агрегат, включающий в себя одну сущность - заявку Request. Вся описанная выше логика выражается в виде диаграммы последовательности (Рис. 2.37).

Введем два граничных класса для страниц пользовательского интерфейса - главную страницу и страницу редактирования заявки - и управляющий класс как обработчик прецедента EditRequest. Страницы экранных форм обычно присутствуют в пользовательской истории, но не являются

ключевыми абстракциями предметной области, поэтому ранее на диаграмме классов предметной области эти классы отсутствовали. Может показаться, что страницы и обработчик должны относиться к представлению разработки, а не пользователя. Но в пользовательской истории они обычно присутствуют, а представление пользователя в описании архитектуры должно ещё иметь описания пользовательского интерфейса - форм, что представлены на диаграммах.

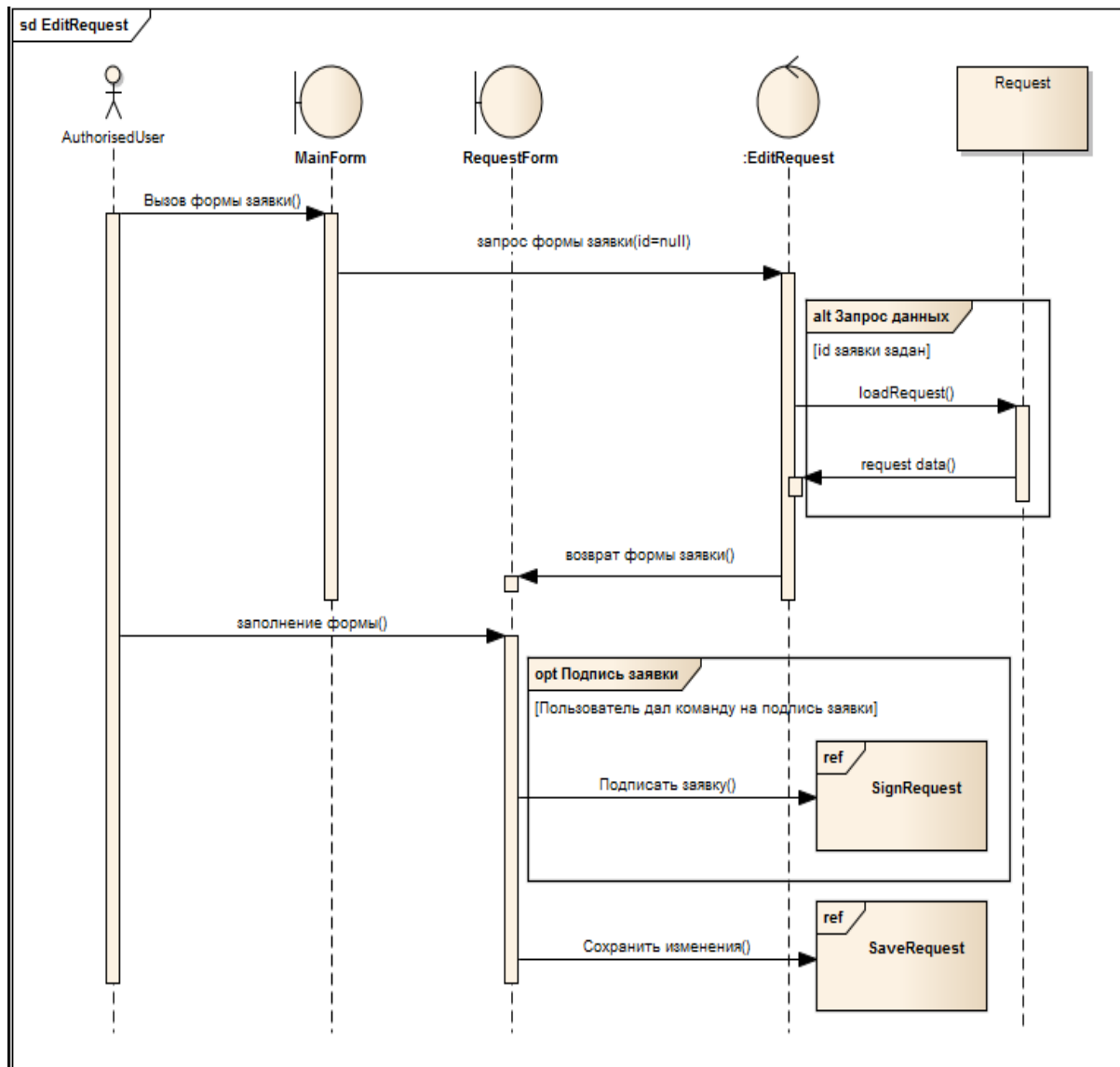


Рис. 2.37 Диаграмма последовательности прецедента EditRequest

Основное назначение диаграммы последовательности - это детализация поведения акторов и сущностей в рамках сценариев использования. Правило: на каждый сценарий (основные, альтернативные, опциональные и исключительные сценарии в рамках каждого прецедента) создаётся отдельная диаграмма. Исключения из правила:

а. если сценарий тривиален, его можно не выносить на отдельную диаграмму - это затраты на создание и сопровождение диаграммы без видимой пользы.

б. маленькие опциональные и альтернативные сценарии можно показать на одной диаграмме с основным сценарием, если это не приводит к чрезмерному усложнению диаграммы.

Дополняют диаграммы последовательности другими диаграммами взаимодействия, если требуется, чтобы заинтересованное лицо получило понятное представление о системе.

4. Моделирование бизнес-процессов

Цель - разработать модели бизнес-процессов, представив посредством диаграммы деятельности/активностей UML, либо в виде диаграммы нотации BPMN, либо в иной нотации, понятной заинтересованным лицам. Модели бизнес-процессов – это процессное представление архитектуры системы **Process view**. Заинтересованные лица – это руководители организации-заказчика, её аналитики, технологи, деятельность которых полностью или частично автоматизирует разрабатываемая система.

Бизнес-процессы моделируются на основе согласованной с заказчиком модели предметной области и модели прецедентов. Прецеденты и их потоки событий эскизно будут событиями бизнес-процесса со своими пред- и постусловиями, инвариантами и т.п.

Диаграмма деятельности UML позволяет показать многопоточность процесса, события, циклы и обмен сообщениями. Прецеденты станут узлами активностей на диаграмме деятельности. Построим диаграмму активности для бизнес-процесса редактирования заявки.

А. Для акторов модели предметной области, которые участвуют в бизнес-процессе, необходимо создать свою «дорожку». Активности (прецеденты), выполняемые конкретным актором, размещаются на его «дорожке».

В. Из модели предметной области берутся объекты, явно передаваемые от одной активности в другую. В примере это заявка Request, Рис.2.38. Перегружать диаграмму не рекомендуется: указать только объекты, отсутствие которых при передаче управления между активностями нарушает бизнес-процесс.

С. Если в процессе возможно возникновения ситуации сбоя (исключения), нужно не забыть внести обработку исключительной ситуации в модель процесса.

Д. Если модель предметной области предполагает наличие бизнес-событий, сигналов или сообщений, в том числе из внешних источников, они должны быть отражены в модели бизнес-процессов.

Е. Если в процессе присутствует субъект или объект, отсутствующий в модели предметной области, нужно скорректировать модели предметной области и прецедентов.

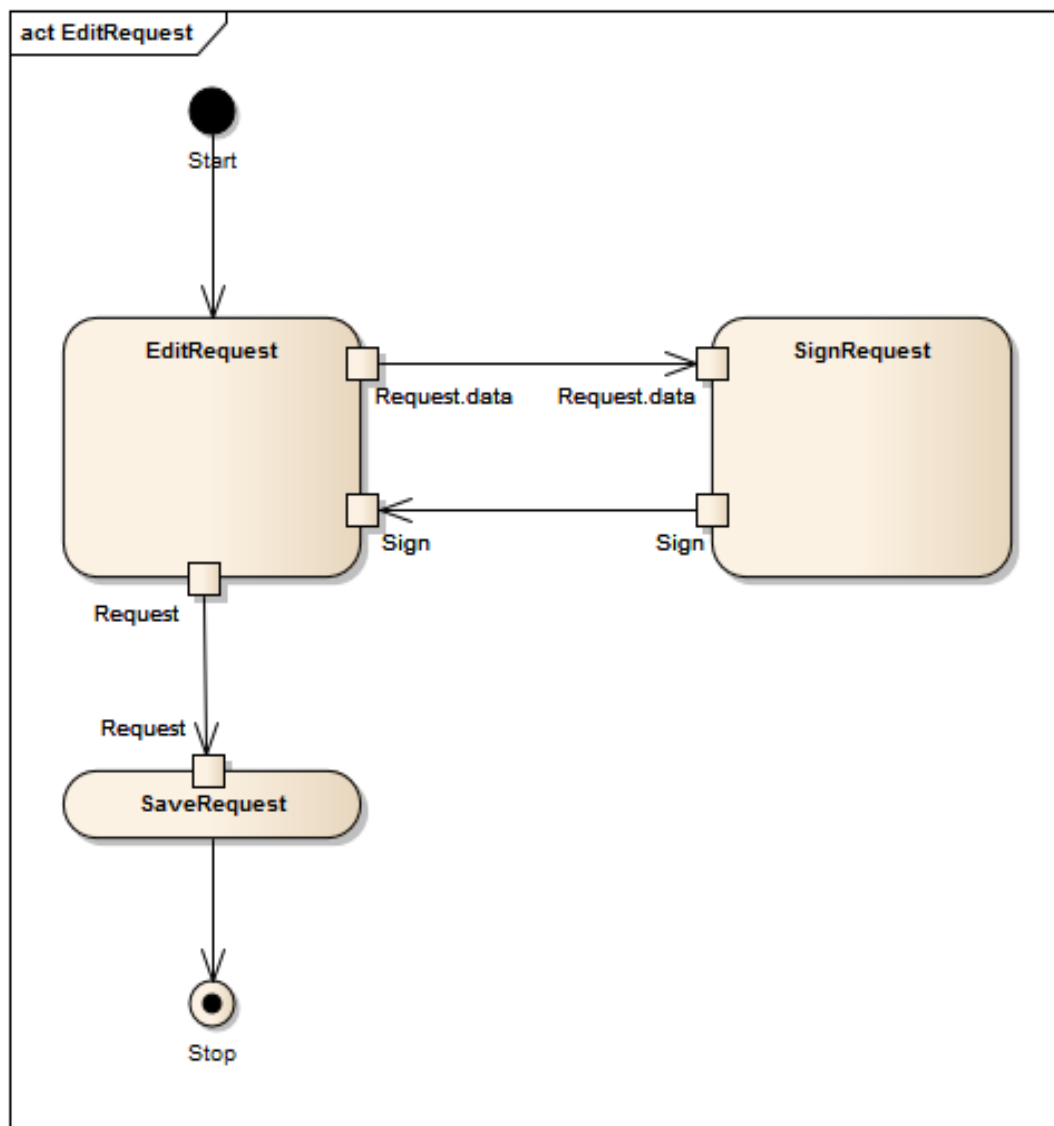


Рис. 2.38 Диаграмма деятельности прецедента EditRequest

Ошибки описания алгоритма процесса:

- проверяйте достижимость окончания процесса из любой активности с учётом всех условий, часто в сложных процессах появляются ветви, ведущие в никуда;
- бесконечные циклы: если явный цикл чаще всего аккуратно представлен, то цикл неявный, связанный с параллельным выполнением задач, одна из которых снова приводит в начало подпроцесса, - это место постоянных ошибок;
- акторы: если одну и ту же операцию могут сделать два или более пользователя (роли), можно создать в иерархии акторов абстрактного актора, наследниками которого будут реальные акторы, выполняющие операцию. Для абстрактного актора создают дорожку, и размещают на ней активность. Возможна ситуация множественного

наследования, когда один и тот же актор оказывается наследником нескольких других акторов – в этом случае нужно проверить ролевую модель и исключить такое наследование.

5. Разработка проектной модели

Цель - разработать проектную модель системы как детализацию аналитической модели с добавлением деталей на языке реализации и конкретных технических решений. Проектная модель – это представление разработки архитектуры системы **Development view**. Используются UML-диаграммы классов, компонентов, пакетов для описания компонентов системы и их объединения в логические пакеты (например, слои многослойной архитектуры). Заинтересованные лица – разработчики системы.

Целью анализа было построение логической модели системы отражающей функциональность, а цель проектирования – определить в полном объеме, как будет реализовываться эта функциональность.

Проектная модель – это аналог аналитической, но все артефакты в ней представлены в деталях реализации. Аналитический класс является эскизом с парой атрибутов и только ключевыми операциями. В проектном классе прописаны языком программирования все атрибуты и операции, включая возвращаемые типы и списки параметров (Рис. 2.39).

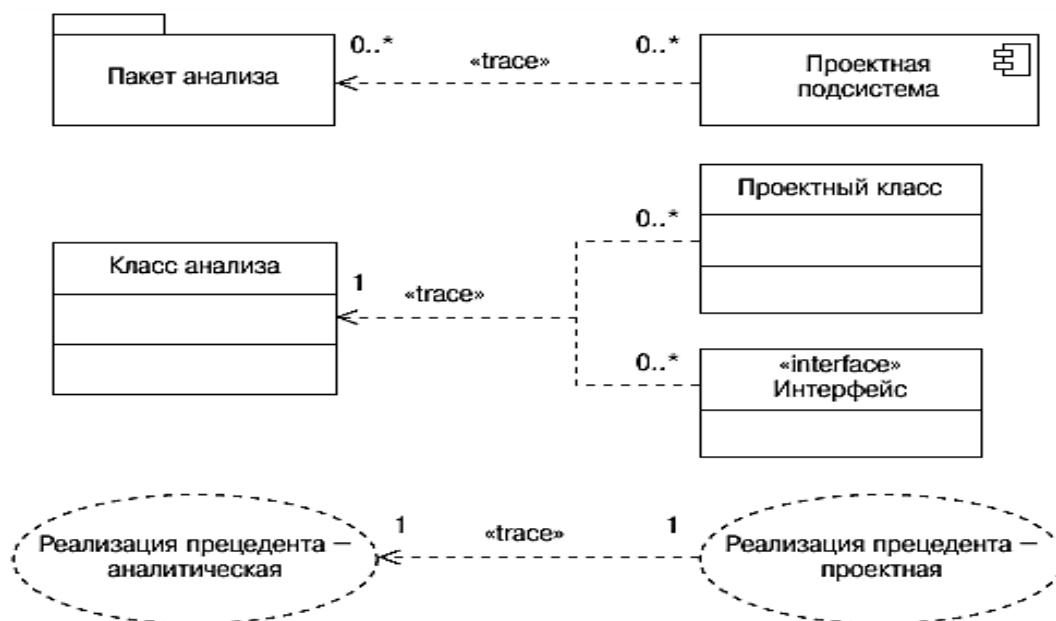


Рис. 2.39 Отношения между ключевыми артефактами анализа и проектирования

Одними из ключевых артефактов проектирования являются **интерфейсы**, они позволяют разложить систему на подсистемы, которые могут разрабатываться параллельно.

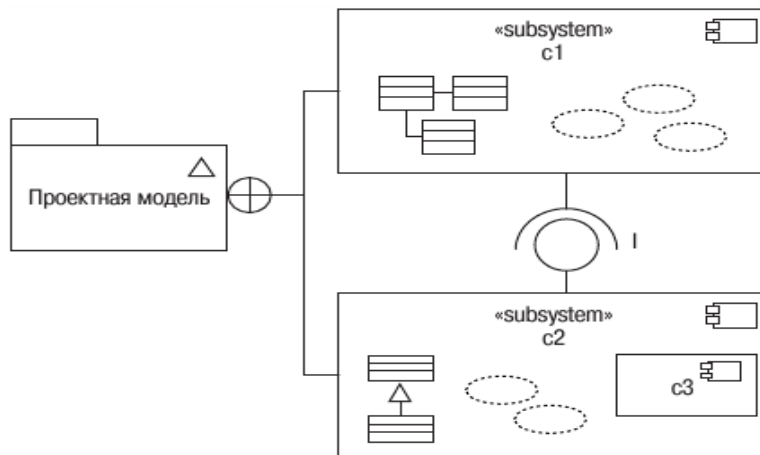


Рис. 2.40 Мета модель проектной модели

Проектная модель содержит ряд проектных подсистем, на Рис. 2.40 показаны только две подсистемы. Подсистемы – это компоненты, которые могут включать различные типы элементов модели.

Планируемые виды деятельности.

1. *Проектирование архитектуры* (architectural design). Результатом проектирования архитектуры должны стать подсистемы, интерфейсы, проектные классы, эскиз модели развертывания и общее архитектурное описание системы⁶(Рис.2.41). Диаграмма развертывания создается как эскиз и показывает распределение программной системы на физических вычислительных узлах, она имеет значение для проектирование компонентов и подсистем.

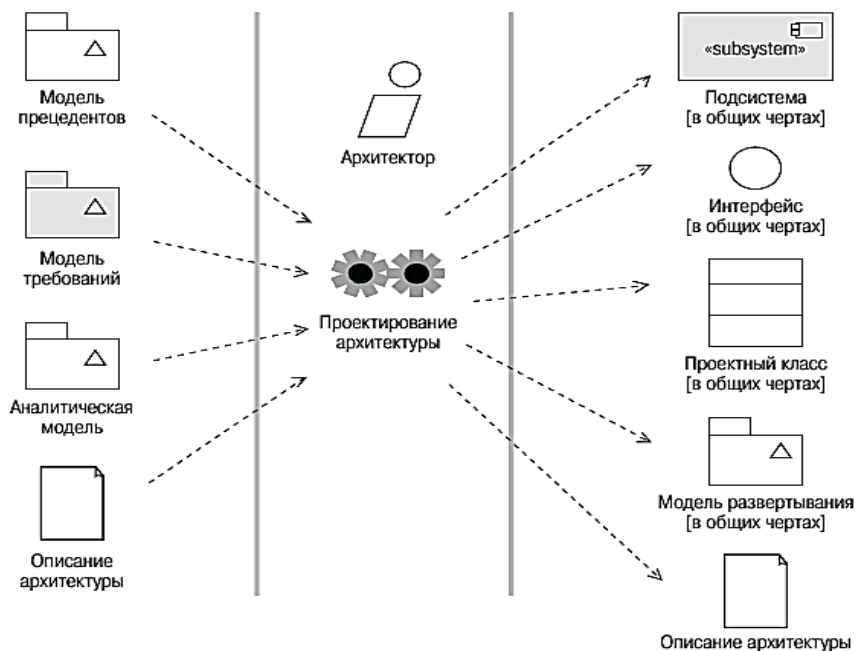


Рис. 2.41 Деятельность Проектирование архитектуры

⁶ См. п. Архитектурные шаблоны

2. *Разработка проектных классов* - классов, описания которых настолько полные, что они могут быть реализованы, запрограммированы на выбранном языке. Проектные классы имеют два источника происхождения:

- **предметная область** посредством уточнения классов анализа, добавление деталей реализации; в процессе класс анализа иногда необходимо разбить на два или более детализированных проектных класса; реализацию класса анализа описывает отношение «trace», устанавливаемое между ним и одним или более проектными классами.

- **область готовых решений** – это библиотеки поставляемых классов и многократно используемых компонентов (например, из библиотеки шаблонов STL - List, Vector и др.), базы данных (реляционные, объектные), компонентные инфраструктуры, такие как .NET, CORBA или EJB⁷, компоненты интерфейса пользователя GUI, фреймворки.

Классы анализа не содержат полного набора атрибутов и операций – это фактически только эскиз, отражающий ключевой состав класса. При переходе к проектированию все операции и атрибуты класса должны быть полностью описаны (Рис.2.42), и поэтому часто класс становится слишком большим. В этом случае необходимо разбить его на несколько небольших классов. Надо стремиться проектировать небольшие классы, которые справляются с одной-двумя функциями. В проектных классах необходимо четко определить детали реализации класса на языке программирования:

- а) набор атрибутов, для каждого описать идентификатор (согласно синтаксису языка программирования), тип, видимость и возможное значение по умолчанию;
- б) набор операций, для каждой операции описать заголовок, то есть имя, список параметров с типами и возвращаемый тип.

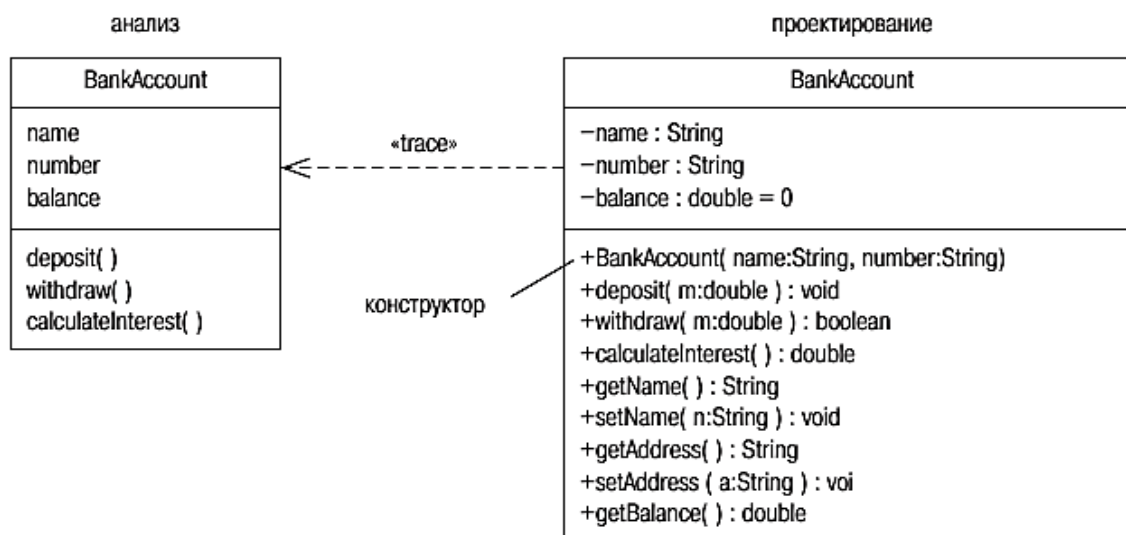


Рис. 2.42 Пример реализации проектного класса на основе класса анализа

⁷ Enterprise JavaBeans - технология Java EE для разработки серверных компонентов.

При проектировании классов для выявления операций реализации прецедента, в котором он участвует, строится соответствующая *диаграмма состояний класса*. Данная диаграмма моделирует поведение на основе модели вычислений конечного автомата. Диаграммы состояний создаются для моделирования объектов со сложным поведением.

3. *Уточнение отношений, выявленных при анализе.* При проектировании отношений между классами анализа необходимо уточнить и преобразовать их в программируемые отношения между проектными классами.

Отношения ассоциации (двунаправленные ассоциации, классы ассоциации или ассоциации многие-ко-многим) аналитической модели не могут быть запрограммированы, их надо преобразовать. Ассоциация может быть преобразована:

- а. в отношения агрегации, композиции или зависимости;
- б. в класс ассоциаций.

Для преобразования ассоциации должны быть точно описаны:

- а. признаком навигации;
- б. кратностью на обоих концах;
- с. должно быть указано имя ассоциации или имя роли на целевом конце.

4. *Проектирование подсистем.* Проектирование подсистем (Design subsystem) – это деятельность по проектированию подсистем как компонентов и интерфейсов их взаимодействия. Подсистемы – это компонент со стереотипом «subsystem». С точки зрения процесса UP подсистемы являются деятельностью по структурированию системы, см. Рис.2.43. Подсистема выступает единицей декомпозиции большой системы на управляемые части.

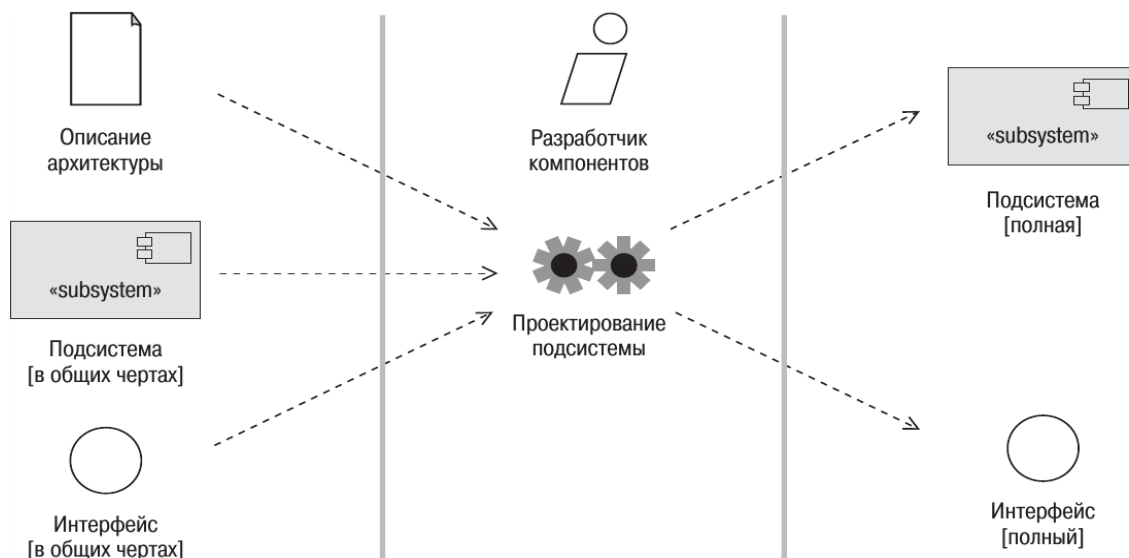


Рис. 2.43 Деятельность UP Проектирование подсистемы

Проектирование подсистемы заключается в разделении системы на максимально независимые друг от друга части. Взаимодействуют подси-

стемы через интерфейсы. Правильное проектирование подсистем – минимизировать их зависимость друг от друга (связанность системы), это достигается за счет интерфейсов, которые обеспечат подсистемам оптимальное взаимодействие.

Компонент - это модуль системы, внешнее поведение которого полностью определяется предоставляемыми и требуемыми интерфейсами (Рис. 2.44), поэтому легко можно заменить один компонент другим, поддерживающий тот же интерфейс. Компоненты инкапсулируют в себе атрибуты, операции и могут участвовать в отношениях ассоциации и обобщения.

С точки зрения UML, компоненты – это **структурированные классификаторы**⁸. У них может быть внутренняя структура, включающая части и соединители, в зависимости от выбранного стереотипа для компонента. UML 2 обладает большим набором стандартных стереотипов компонента.

Создание систем как набора взаимосвязанных интерфейсами компонент, известно как компонентно-ориентированная разработка (component based development, CBD).

Интерфейсы объединяют подсистемы, создавая гибкую архитектуру системы, так как построение программного обеспечения осуществляется из подключаемых через интерфейс частей.

Интерфейс определяет именованный набор открытых свойств, тем самым скрывая детали реализации подсистем и отделяя описание функциональности (интерфейса) от ее реализации.

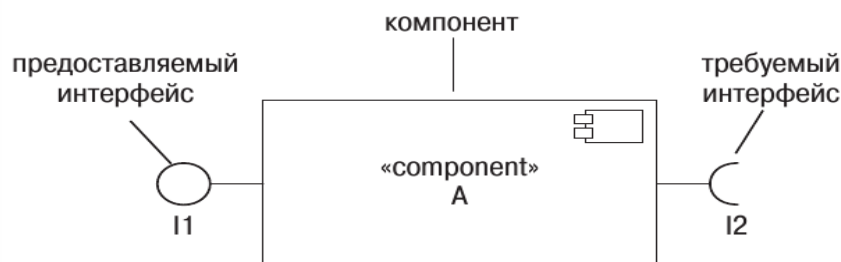


Рис. 2.44 Представление компонента в UML

Если проектировать только классы, то это реализация («проектирование реализации») жестко связанных классов, но если классы с интерфейсом, то этот интерфейс может быть гибко использован любым количеством классов и других классификаторов. Этот процесс можно назвать «проектирование контракта» взаимодействия. Создать экземпляр интер-

⁸ Классификатор UML — обобщённая концепция описания структурных и поведенческих свойств некоего элемента системы.

фейса невозможно, это объявление контракта взаимодействия, заданного классификатором.

Интерфейс определяет сервис, предлагаемый классом, подсистемой или компонентом. В некоторых языках программирования (Java), интерфейсы реализованы как языковая конструкция. Если язык реализации не поддерживает интерфейсы явно, то их заменяют описанием абстрактного класса. Например, в C++ описывается полностью абстрактный класс, где все операции абстрактны (чисто виртуальные).

Диаграмма компонент представляет подсистемы и их интерфейсы. Пример. На диаграмме компонент (Рис. 2.45) представлены две подсистемы GUI и BusinessLogic, связанные интерфейсами CustomerManager, AccountManager и OrderManager. На диаграмме видно, что предоставляет интерфейсы система BusinessLogic. Подсистемы не знают о внутренней реализации друг друга. Если подсистему BusinessLogic полностью заменить одной или несколькими подсистемами, то они должны просто предоставить этот же набор интерфейсов. Подсистема GUI даже не заметит замены.

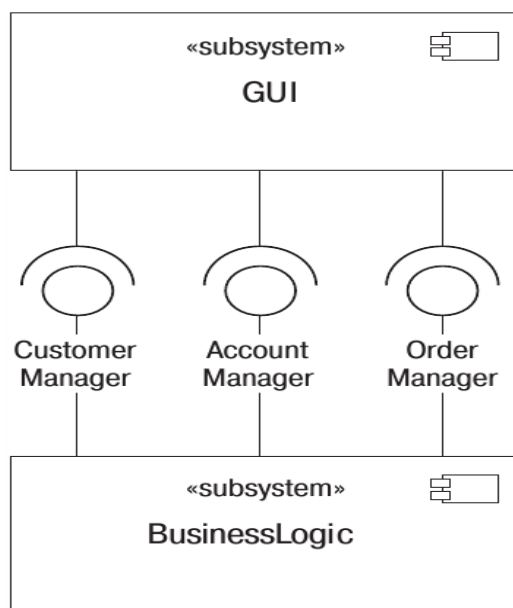


Рис. 2.45 Диаграмма компонент GUI и BusinessLogic

Таким образом, использование интерфейсов разделяет подсистемы и обеспечивает гибкость архитектуры.

5. *Проектирование прецедентов.* Проектная реализация прецедента — это взаимодействие проектных объектов и проектных классов, реализующих прецедент. Проектирование реализации прецедента определяет решения уровня реализации и реализует нефункциональные требования.

Проектная реализация прецедента состоит из: проектных диаграмм взаимодействий; диаграмм классов, включающих участвующие в ней проектные классы с интерфейсами; подсистем с интерфейсами, состав которых, реализует функциональность прецедента (Рис. 2.46).

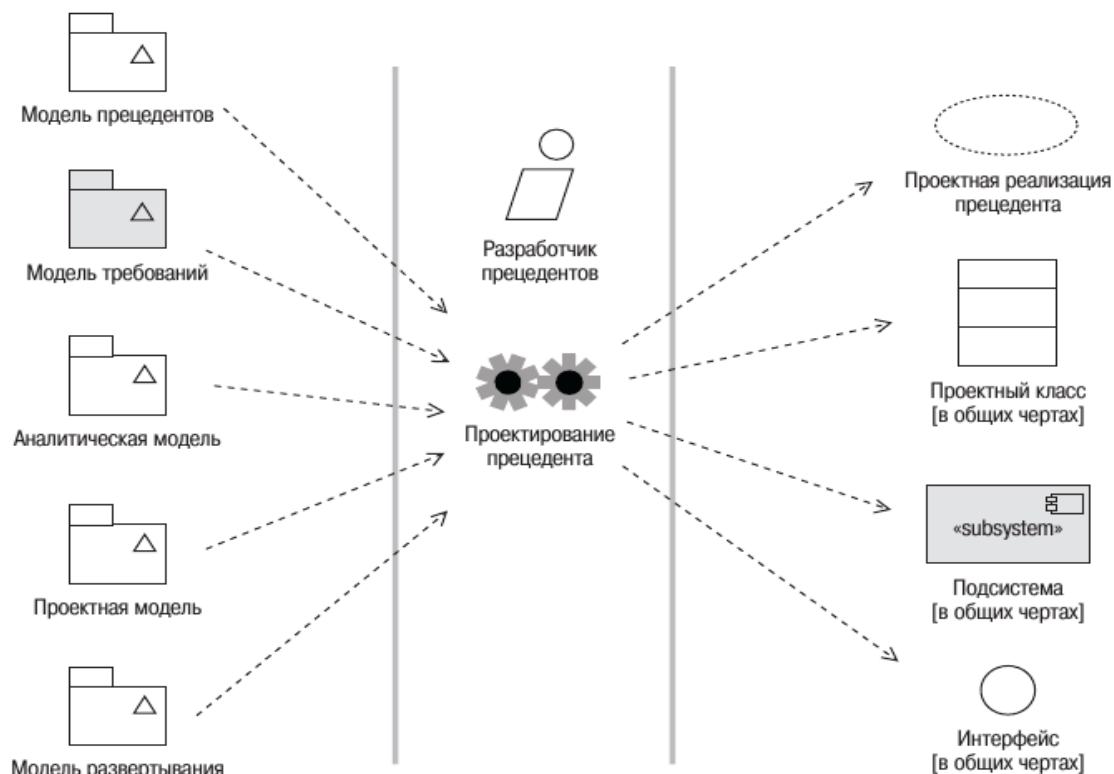


Рис. 2.46 Деятельность UP Проектирование прецедента

Удобное представление того, как архитектура реализует прецеденты демонстрирует диаграмма подсистем в общих чертах, где представлено взаимодействия между подсистемами. Подсистема рассматривается как черный ящик, который предоставляет и требует сервисы, описанные как предоставляемые и требуемые интерфейсы. Взаимодействие объектов внутри подсистемы данная диаграмма не представляет.

6. Модель реализации

Цель - представить проектную модель с точки зрения реализации. Модель реализации – это проектная модель, развернутая на физическом оборудовании (Рис. 2.47). Модель демонстрирует аппаратную инфраструктуру автоматизированной системы.

Модель реализации – это представление архитектуры системы **Physical view**. Используется UML-диаграмма развёртывания. Заинтересованные лица – системный инженер, программист.

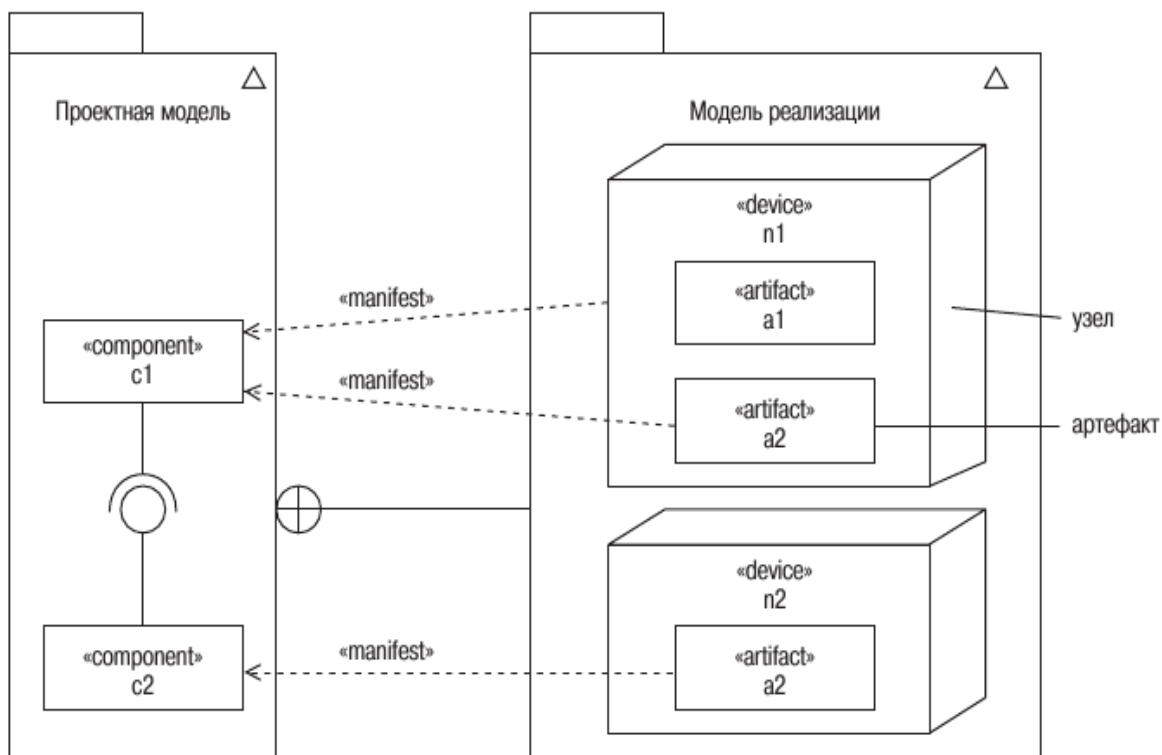


Рис. 2.47 Отношения между проектной моделью и моделью реализации

Процесс UP - это деятельность Реализация архитектуры (Architectural implementation) для создания диаграммы развертывания. Поток реализации – это основной поток фазы Построение (Рис. 2.48).

Диаграмма развертывания UML представляет системную архитектуру, включающую аппаратное и программное обеспечение системы. Она требует распределения артефактов программной части системы по аппаратным узлам или экземпляров артефактов по экземплярам аппаратных узлов, то есть программная архитектура отражается на аппаратной архитектуре системы. Здесь требуется определить технические средства выполнения программной системы.

В распределенных системах она моделирует распределение программного обеспечения по физическим распределенным узлам.

Реализация зависит на языке программирования, инструментов создания кода, и поэтому здесь происходит распределение проектных классов по компонентам и преобразование проектной модели в исполняемый код. Главная задача процесса реализации - это производство исполняемого кода.

Детальное и явное моделирование реализации проводят в следующих случаях:

а. автоматическая генерация кода из модели CASE-средством разработки, в этом случае необходимо определить компоненты и исходные файлы;

в. с целью повторного использования компонентов ведется компонентно-ориентированная разработка и распределение проектных классов и интерфейсов по компонентам становится стратегическим вопросом.

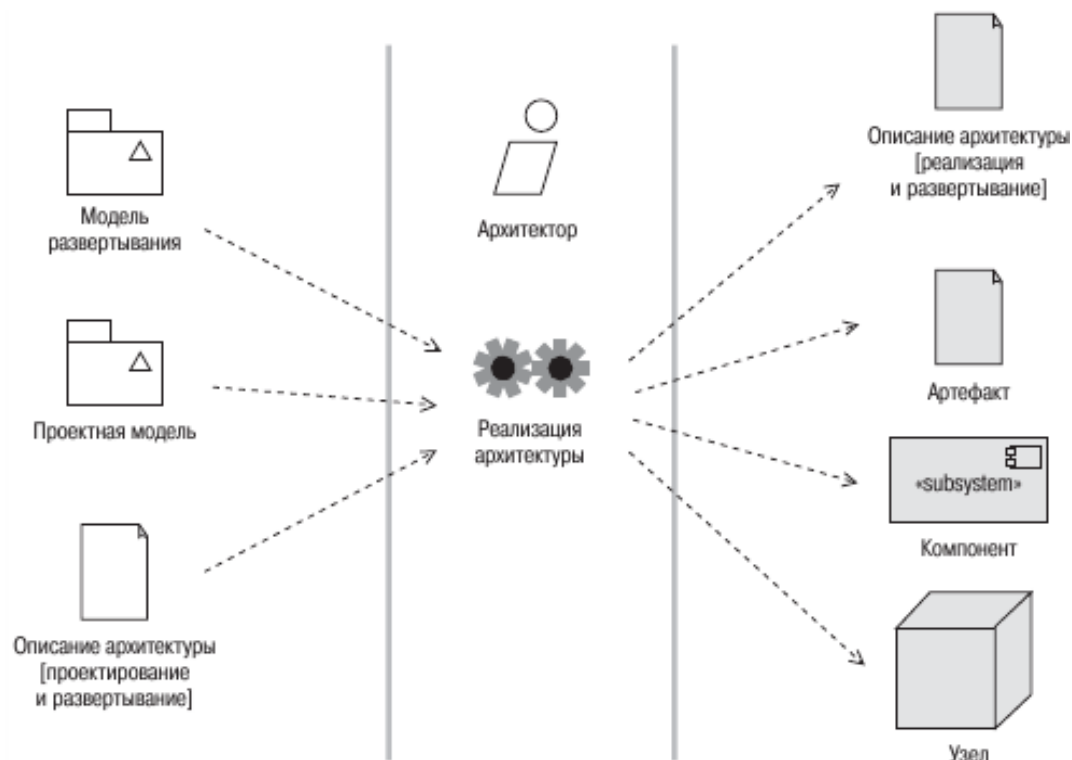


Рис. 2.48 Деятельность UP Реализация архитектуры

Многие инструментальные средства моделирования позволяют создавать модель реализации из исходного кода путем обратного проектирования, что достаточно эффективно выполняет моделирование реализации.

2.3 Структурные методологии

Сущность структурного подхода к разработке системы заключается в декомпозиции на автоматизируемые функции: система разбивается на функциональные подсистемы, которые, в свою очередь, делятся на подфункции, те на задачи и так далее до конкретных процедур. При этом автоматизируемая система сохраняет целостное представление, в котором все составляющие компоненты взаимосвязаны. Базовыми принципами методологии являются:

- принцип “разделяй и властвуй” заключается в построении сложной системы из небольшого количества крупных частей, каждая из которых, в свою очередь, строится из частей меньшего размера и т. д., до тех

пор, пока самые небольшие части можно будет строить из имеющегося материала;

- принцип иерархического упорядочения – принцип организации составных частей системы в иерархические древовидные структуры с добавлением новых деталей на каждом уровне.
- принцип абстрагирования – выделение существенных аспектов системы и отвлечение от несущественных;
- принцип непротиворечивости – обоснованность и согласованность элементов системы;
- принцип структурирования данных – данные должны быть структурированы и иерархически организованы.

2.3.1 Методологии семейства IDEF

Структурный подход лежит в основе методов IDEF (Integrated DEFinition) для моделирования сложных систем. Развитие этих методов привело к появлению семейства методологий стандарта IDEF.

«IDEF (I-CAM DEFinition или Integrated DEFinition) - это методологии семейства ICAM (Integrated Computer-Aided Manufacturing) для решения задач моделирования сложных систем, позволяют отображать и анализировать модели деятельности широкого спектра сложных систем в различных разрезах. При этом ширина и глубина обследования процессов в системе определяется самим разработчиком, что позволяет не перегружать создаваемую модель излишними данными» [4].

В основу была положена технология структурного анализа и проектирования SADT (Structured Analysis and Design Technique), разработанная в начале 70-х годов фирмой SoftTech. В настоящий момент семейство стандартов IDEF состоит из методологий представленных в Таблице 2.3.

Таблица 2.3

Стандарты семейства IDEF

Стандарт	Назначение методологии	Нотации
IDEF0	Function Modeling, методология функционального моделирования, составной частью которой является метод SADT, описывает бизнес-процесс в виде иерархической системы взаимосвязанных функций.	SADT
IDEF1	Information Modeling, методология моделирования информационных потоков внутри системы, позволяющая отображать и анализировать их структуру и взаимосвязи.	DFD, STD
IDEF1X (IDEF1 Extended)	Data Modeling, методология моделирования баз данных на основе модели «сущность-связь». Применяется для построения информационной модели, которая представляет структуру информации, необходимой для поддержки функций производственной системы или среды.	ERD

IDEF2	Simulation Model Design, методология динамического моделирования развития систем, построенная на базе «раскрашенных сетей Петри» (CPN — Color Petri Nets), в связи с весьма серьёзными сложностями анализа динамических систем его развитие приостановилось на самом начальном этапе, но есть алгоритмы и их компьютерные реализации, позволяющие превращать набор статических диаграмм IDEF0 в динамические модели.	CPN
IDEF3	Process Description Capture, документирование технологических процессов, методология документирования процессов, происходящих в системе, описывает сценарий и последовательность операций для каждого процесса. Каждая функция IDEF0 (функциональный блок) может быть представлена в виде отдельного процесса средствами IDEF3.	
IDEF4	Object-Oriented Design, методология построения объектно-ориентированных систем, позволяет отображать структуру объектов и заложенные принципы их взаимодействия и тем самым анализировать и оптимизировать сложные объектно-ориентированные системы.	
IDEF5	Ontology Description Capture, методология онтологического анализа систем, т.е. анализа основных терминов и понятий (словаря), используемых для характеристики объектов и процессов, границ использования, взаимосвязей между ними. На основе этих утверждений формируются выводы о дальнейшем развитии системы и производится её оптимизация.	
IDEF6	Design Rationale Capture, методология обоснования проектных действий. Назначение IDEF6 состоит в облегчении получения «знаний о способе» моделирования, их представления и использования при разработке систем управления предприятиями.	
IDEF7	Information System Auditing, методология аудита информационных систем, определена как востребованная, но разработана не полностью.	
IDEF8	User Interface Modeling, методология разработки интерфейсов взаимодействия оператора и системы (пользовательских интерфейсов), фокусирует внимание на проектировании выполняемых операций интерфейса, на сценарии взаимодействия, на элементах управления и деталях интерфейса.	
IDEF9	Scenario-Driven IS Design (Business Constraint Discovery method), метод исследования бизнес-ограничений, разработан для облегчения обнаружения и анализа ограничений в условиях, в которых действует предприятие.	
IDEF10 - IDEF14	методы определены как востребованные, не были полностью разработаны: IDEF10, Implementation Architecture Modeling, моделирование архитектуры выполнения; IDEF11, Information Artifact Modeling; IDEF12, Organization Modeling, организационное моделирование; IDEF13, Three Schema Mapping Design, трёхсхемное проектирование преобразования данных. IDEF14, Network Design, метод проектирования компьютерных сетей, основанный на анализе требований сетевых компонентов существующих конфигураций сетей.	

Характеристика нотаций.

SADT - метод структурного анализа и проектирования, сочетает процесс моделирования и управление конфигурацией проекта со своим графическим языком.

ERD (Entity Relationship Diagrams, диаграммы "сущность - связь") - способ определения данных и отношений между ними, детализация хранилищ данных проектируемой системы, включая идентификацию объектов (сущностей), свойств этих объектов (атрибутов) и их отношений с другими объектами (связей).

DFD (Data Flow Diagrams, диаграммы потоков данных) – методология графического структурного анализа, описывающая внешние по отношению к системе источники и адресаты данных, логические функции, потоки данных и хранилища данных, к которым осуществляется доступ. Для описания диаграмм DFD используются две нотации: Йордана (Yourdon) и Гейна-Карсон (Gane-Sarson), отличающиеся синтаксисом.

STD (State Transition Diagrams, диаграммы переходов состояний) - методология моделирования последующего функционирования системы на основе ее предыдущего и текущего функционирования.

CPN (Color Petri Nets, раскрашенные сети Петри) - методология создания динамической модели бизнес-процесса, анализирует зависящие от времени характеристики процесса и распределение ресурсов для входящих потоков различной структуры.

2.3.2 Методологии моделирования бизнес-процессов

Бизнес-процесс — это алгоритмическая последовательность действий (операций, процедур), направленных на решение задачи бизнеса. Например, регистрация документа, обслуживание заявки и т.д.

Модель бизнес-процесса — это представление процесса языком определенной нотации. Модель содержит действия (функции, процессы), элементы логики последовательности действий, ответственных лиц, входные и выходные артефакты, регламентирующую документацию, инструменты выполнения. Модель представляет блок-схему бизнес-процесса. Моделирование бизнес-процесса состоит из *разработки модели и ее анализа*.

Основные подходы к моделированию процессов:

а. Функциональный, где бизнес-процесс - это функция, имеющая входные данные и выходной результат.

б. Процессный, где бизнес-процесс - это совокупность действий, приводящих к результату.

EPC (event-driven process chain), BPMN 2.0, IDEF0, UML (диаграмма деятельности) – это наиболее развитые нотации моделирования бизнес-процессов.

Модель в нотации ЕРС разрабатывается в основном вертикально, сверху вниз (но это не обязательно!), основные элементы – это события и функции. События инициируют или завершают функции (действия), которые направлены на достижение определенных результатов. В качестве соединительных элементов используются стрелки и логические элементы И, ИЛИ, ИСКЛЮЧАЮЩЕЕ ИЛИ. Дополнительно функции поясняются набором различных информационных атрибутов, которые отражают их входы и выходы - это бумажные/электронные документы, различного рода товарно-материальные ценности, инструменты, исполнители, источники информации и системы (Рис. 2.49).

Для построения схемы, сначала определяют стартовое/финальное событие, затем – промежуточные события, необходимые для них исполнители, ресурсы (Рис. 2.50). Пример диаграммы бизнес-процесса на Рис. 2.51.

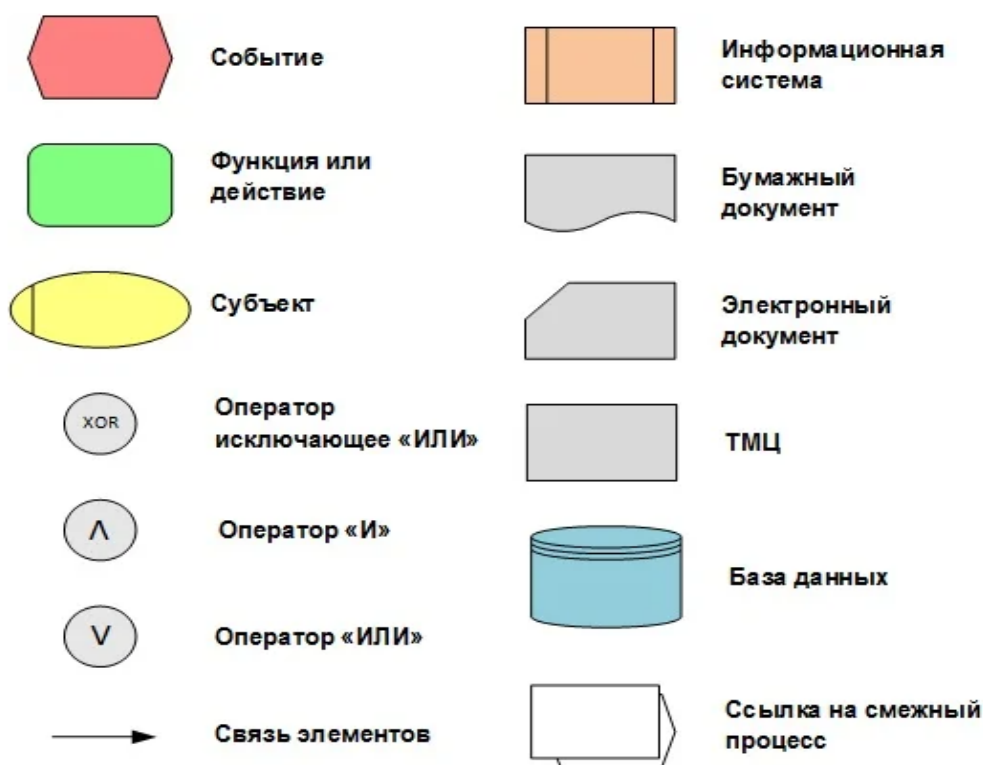


Рис. 2.49 Основные элементы нотации ЕРС

Логические элементы используются следующим образом:

- **ИСКЛЮЧАЮЩЕЕ ИЛИ** - организует ветвление процессного потока только по одному направлению, то есть это условие, которое включает два возможных варианта развития событий.
- **И** - организует слияние процессного потока, то есть необходимо выполнить именно два условия для перехода к дальнейшим действиям.
- **ИЛИ** - организует ветвление, которое осуществляется при выполнении одного или двух условий.

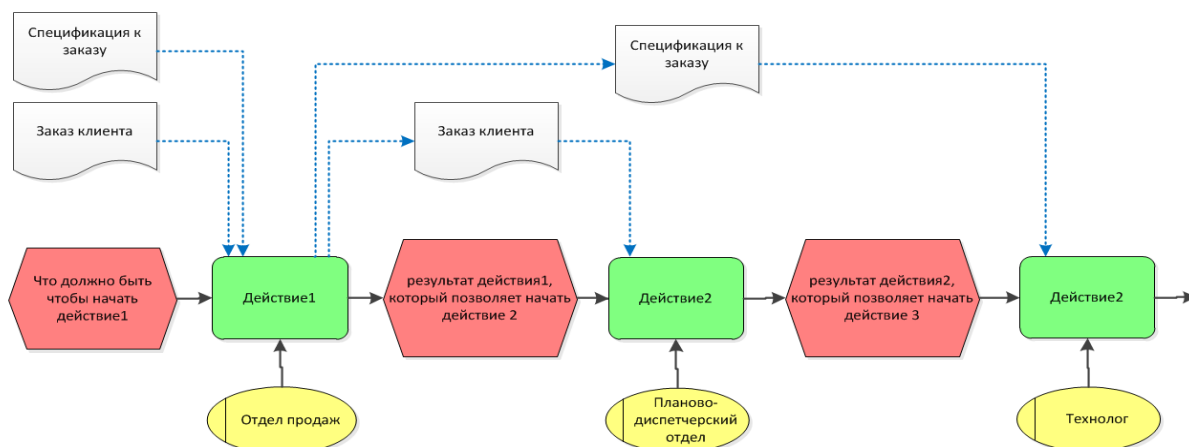


Рис. 2.50 Шаблон диаграммы бизнес-процесса нотации EPC

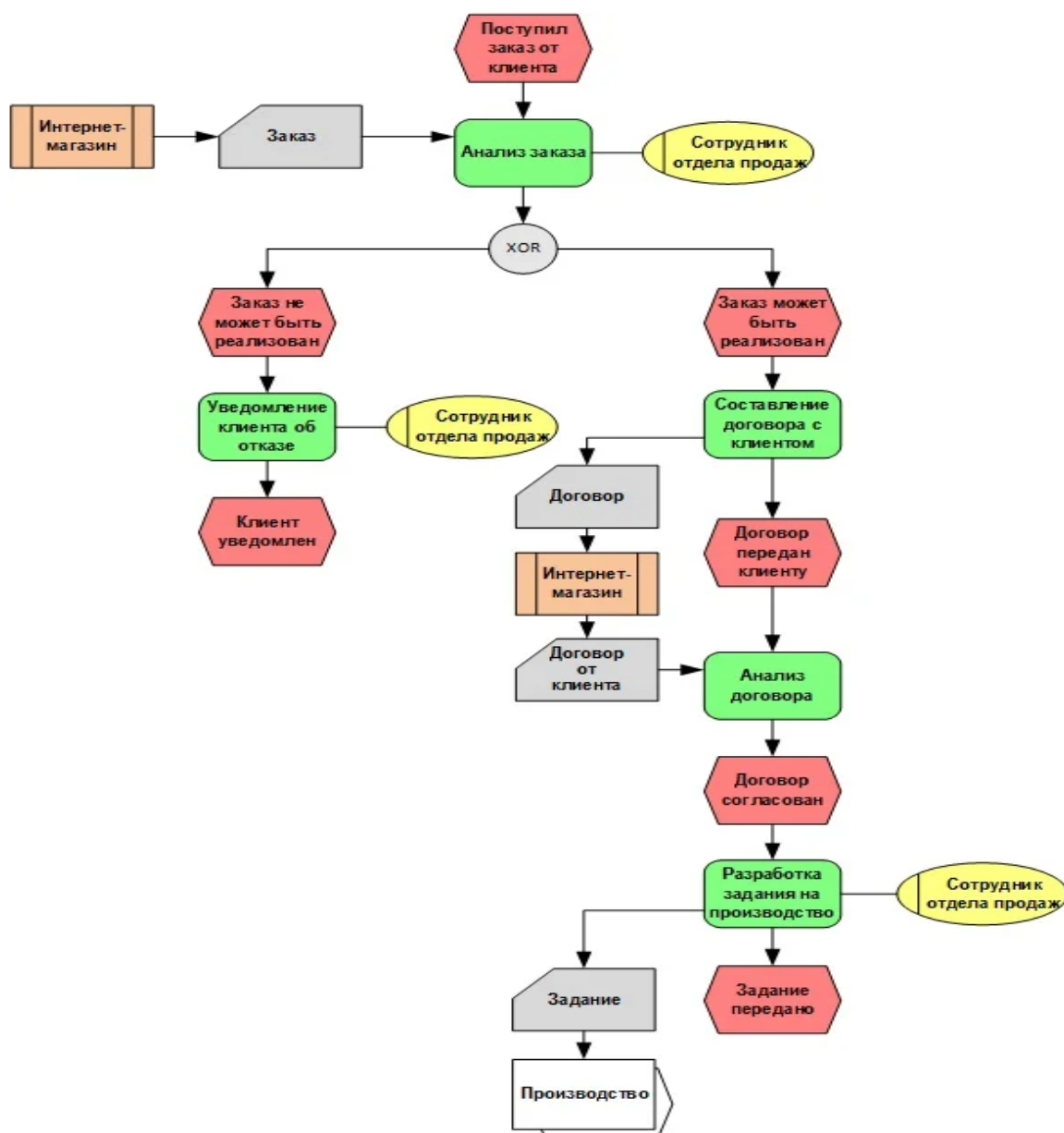


Рис. 2.51 Пример EPC-диаграммы процесса "Интернет-продажи"

Нотация BPMN 2.0. – современная, динамично развивающаяся нотация, которая позволит автоматизировать большинство процессов предприятия. Она появилась относительно недавно и постоянно совершенствуется, последняя версия BPMN 2.0.2 была утверждена в 2014 году.

Нотация BPMN 2.0 моделирует бизнес-процесс в горизонтальной ориентации, он выделен в специальный пул, который разбивается "дорожками" участников процесса. Элементов в BPMN 2.0 больше по сравнению с ЕРС. Основные элементы нотации BPMN (Рис. 2.52) следующие: задача (прямоугольник), событие (круг), развилка (ромб), поток, ход (стрелка), базы данных, документы, сноски, пулы. Логические элементы процесса представлены шлюзами. Расширенный набор содержит более 100 элементов.

В основе моделирование лежит алгоритм бизнес-процесса, схема демонстрирует поток прохождения процесса.

Модель процесса в нотации BPMN помогает описать алгоритм в такой форме, которая будет понятна заинтересованному лицу, не прошедшему специального обучения.

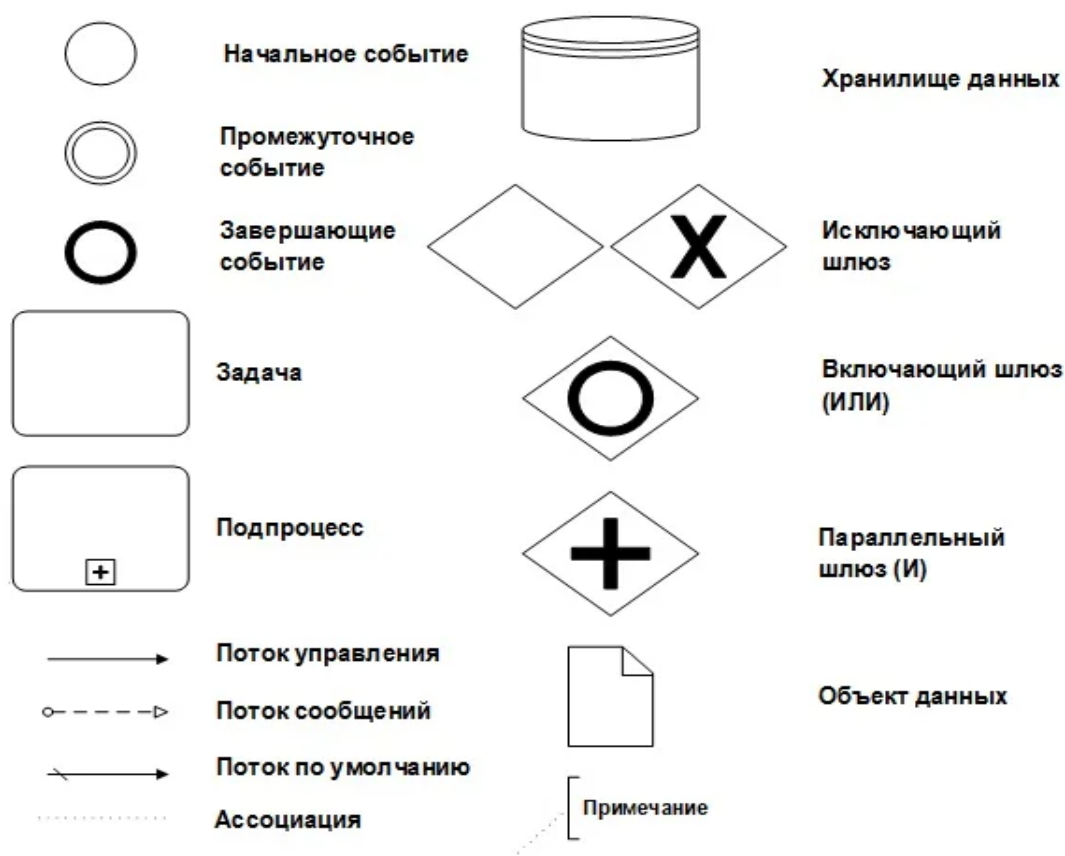


Рис. 2.52 Основные элементы нотации BPMN 2.0

Пример процесса "Интернет-продажи" выполненный в нотации BPMN 2.0 представлен на Рис. 2.53.

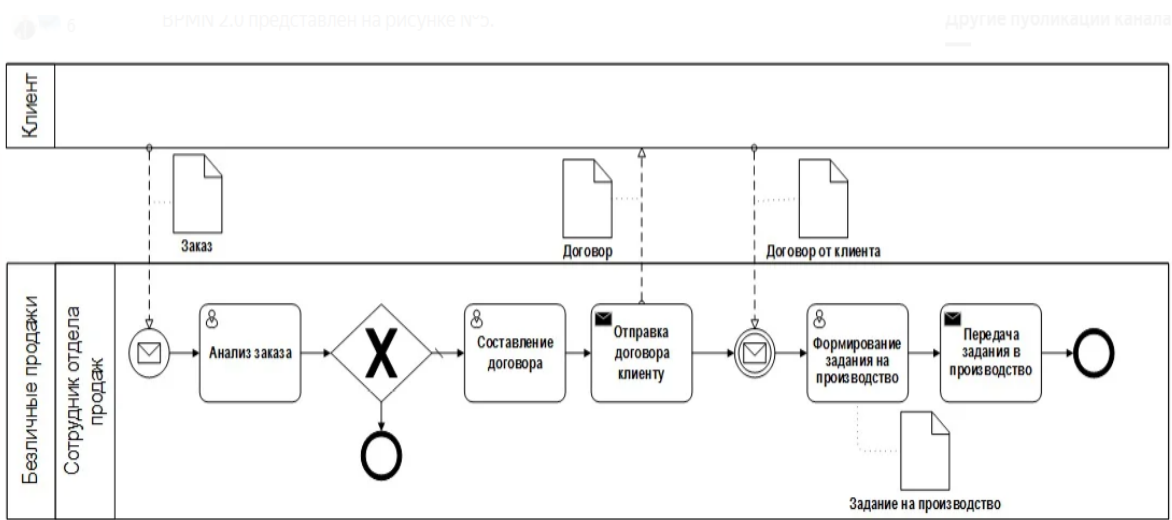


Рис. 2.53 Процесс "Интернет-продажи" выполненный в нотации BPMN 2.0

Представление модели в расширенной нотации усложняется, и понять её сложно без специальных навыков и средств. Чтение таких схем автоматизировано специальными средами разработки и для автоматизации предприятия - это специальный профессиональный инструмент.

Выбор нотации моделирования бизнес-процессов определяется целями. Если цель - внедрить систему по автоматизации и управлению процессами предприятия, то необходима нотация BPNM 2.0, так как все производители и поставщики BPMS систем используют только данную нотацию.

Нотация EPC сложна в чтении из-за своих размеров для больших процессов, а декомпозиция их на подпроцессы запутает пользователей. Но очень наглядна и понятна для обучения и представления небольших, но разветвленных процессов.

Если необходимо исследовать бизнес-процесс для организации работы и управления, обучить новых сотрудников, повысить эффективность и производительность, используя модели бизнес-процессов, то выбор зависит от возможностей заинтересованных лиц.

Деятельности, рекомендованные в проектах автоматизации бизнес-процессов [9]:

1. Определение классов пользователей процессов, нуждающихся в автоматизации.
2. Построение матрицы ролей и разрешений для выявления требований к безопасности автоматизируемых бизнес-процессов.
3. Разработка моделей данных для информации, которая обрабатывается вручную.
4. Определение бизнес-правил, которые нужно автоматизировать, если будут автоматизированы зависящие от них процессы.

5. Моделирование в предпочтительной нотации сценариев текущих ручных процессов сейчас и после автоматизации.

6. Использование диаграмм (DFD, деятельности UML) для определения автоматизируемых процессов и создание новых диаграмм, представляющих взаимодействие новых бизнес-процессов с уже автоматизированными процессами системы предприятия.

7. Адаптация бизнес-процессов к готовым решениям как возможность использования готового программного продукта.

8. Для тестирования моделей - построение матриц сопоставления шагов процесса и бизнес-требований к нему.

2.4 Архитектурные шаблоны

Архитектурный шаблон, паттерн — это обобщенное решение по архитектуре системы для распространённой задачи. Шаблонов разработано много, рассмотрим некоторые из них.

Многоуровневая архитектура. Самый распространенный архитектурный шаблон — многоуровневая архитектура (или N-уровневая).

В программной инженерии многоуровневая архитектура или многослойная архитектура — клиент-серверная архитектура, в которой разделяются функции представления, обработки и хранения данных.

N-уровневая архитектура приложения предоставляет модель, по которой разработчики могут создавать гибкие и повторно-используемые приложения. Разделяя приложение на уровни абстракции, разработчики реализуют возможность внесения изменений в какой-то определённый слой, вместо того, чтобы перерабатывать всё приложение целиком.

Понятия слоя и уровня различают: слой — это механизм логического структурирования компонентов программного решения, а уровень — это механизм физического структурирования инфраструктуры системы.

Шаблон широко применяется архитекторами, проектировщиками и разработчиками систем. Ограничений по количеству и типу уровней никаких нет, однако в большинстве случаев такая архитектура состоит из четырех уровней: представление данных, бизнес-логика, хранение данных и база данных, Рис. 2.54.

Концепция и преимущества: в сложных системах необходимо обеспечивать независимую разработку и развитие отдельных частей. Разработчикам необходимо понятное и хорошо задокументированное разделение задач, чтобы модули системы можно было разрабатывать, поддерживать независимым образом. У каждого из уровней есть определенная роль и ответственность. Например, уровень представления данных отвечает за об-

работку всего пользовательского интерфейса. Такое разделение задач в многоуровневой архитектуре упрощает формирование эффективных ролей и ответственности.

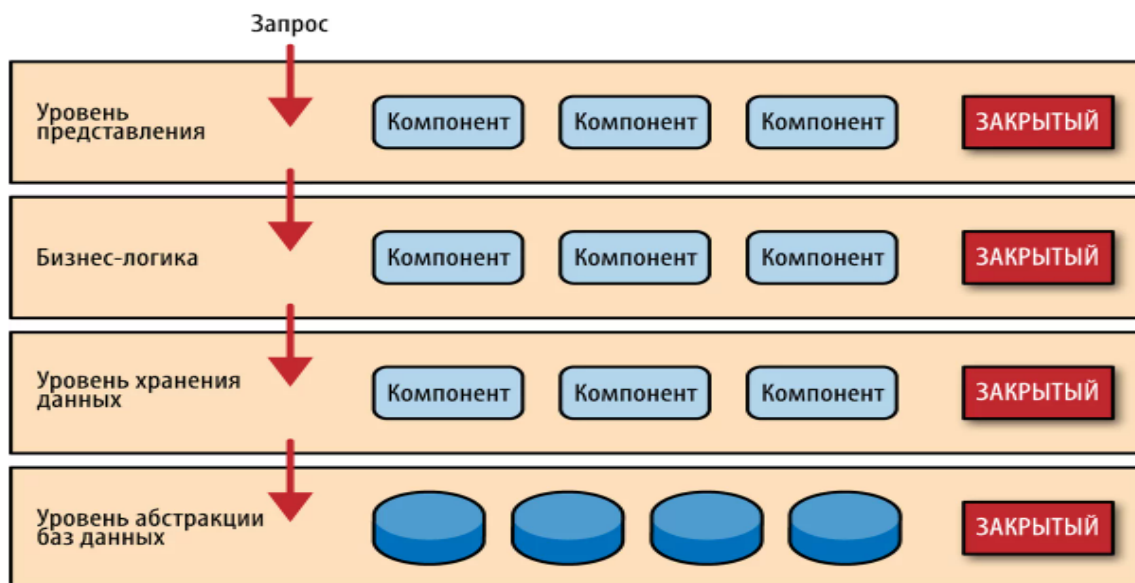


Рис. 2.54 Модель N-уровневой архитектуры

Цель: программное обеспечение необходимо разделить так, чтобы модули можно было разрабатывать, и развивать отдельно — с минимальным взаимодействием между частями, обеспечивая переносимость, модифицируемость и повторное использование.

Решение. Чтобы добиться такого разделения, при использовании многоуровневого шаблона программное обеспечение разделяется на слои/уровни. Каждый слой - это группа модулей, предоставляющих взаимосвязанный набор сервисов. Их применение должно быть однонаправленным. Слои/уровни разделяют систему на подсистемы, причем каждая часть доступна через публичный интерфейс.

Недостатки. Наличие уровней снижает производительность. Этот шаблон не подходит для высокопроизводительных приложений: проходить несколько уровней архитектуры для выполнения бизнес-запроса — это неэффективно. Добавление уровней увеличивает полную стоимость и усложняет систему.

Область применения. Использовать этот подход следует для небольших, простых приложений и веб-сайтов, когда бюджет и время ограничены.

Многоярусный шаблон. При распределенном развертывании часто необходимо разделить инфраструктуру системы на отдельные подмножества, Рис. 2.55.

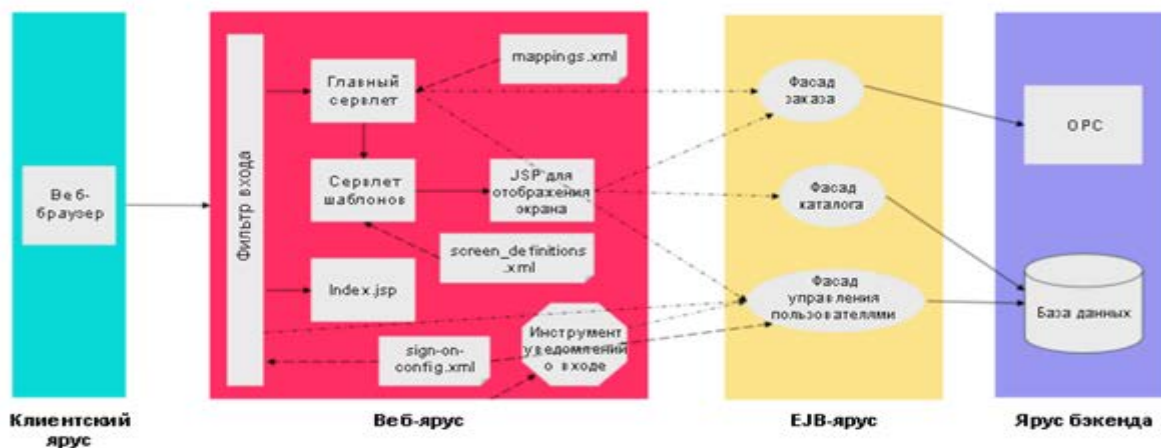


Рис. 2.55 Многоярусная архитектура

Цель. Разделить систему на ряд независимых в вычислительном отношении исполнительных структур - групп программного и аппаратного обеспечения, объединенных каким-нибудь средством связи.

Решение. Исполнительные структуры многих систем организованы как набор логических групп компонентов. Каждая группа называется ярусом.

Недостатки. Высокая стоимость и сложность.

Область применения. Используется в распределенных системах.

Каналы и фильтры. Этот вид архитектуры подходит в том случае, если процесс работы приложения распадается на несколько шагов, которые могут выполняться отдельными обработчиками, Рис. 2.56.

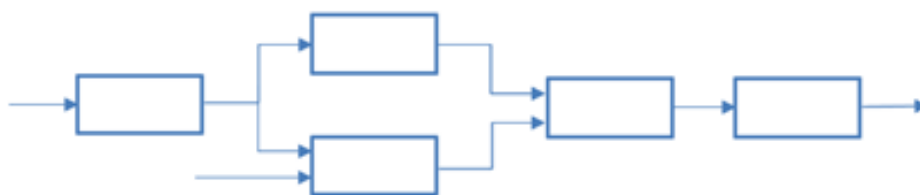


Рис. 2.56 Архитектура каналы и фильтры

Основными компонентами являются «фильтр» (filter) и «канал» (pipe). Иногда дополнительно выделяют «источник данных» (data source) и «потребитель данных» (data sink).

Решение. Каждый поток обработки данных – это серия чередующихся фильтров и каналов, начинающаяся источником данных и заканчивающаяся их потребителем. Каналы обеспечивают передачу данных и синхронизацию. Фильтр же принимает на вход данные и обрабатывает их, трансформируя в некое иное представление, а затем передает дальше.

Недостатки. Архитектура ориентирована на преобразование данных, а не на интерактивные системы. Чрезмерное использование синтаксического анализа и синтеза снижает производительность и усложняет написание самих фильтров.

Область применения. Архитектура каналов и фильтров применяется в самых разных приложениях, особенно при решении задач, обеспечивающих простую одностороннюю обработку — например, инструменты EDI (электронный обмен данными), ETL (извлечение, преобразование и загрузка). Например, компиляторы: последовательно расположенные фильтры выполняют лексический, синтаксический, семантический анализ и создание кода.

Архитектура, управляемая событиями (EDA). Событийно-ориентированная архитектура. Это популярный адаптивный паттерн, широко используемый для создания масштабируемых систем, Рис. 2.57.

Если говорить о программном обеспечении, то в этой схеме существует два варианта событий: инициализирующее событие и событие, на которое реагируют обработчики. Обработчики являются изолированными независимыми компонентами, отвечающими (в идеале) за какую-нибудь одну задачу, и содержат бизнес-логику, необходимую для работы.

Посредник может быть реализован несколькими способами. Самый просто способ — это воспользоваться фреймворками для интеграции Apache Camel, Spring Integration или Mule ESB. Для больших приложений, которым требуется более сложные функции управления, можно реализовать посредника, используя концепцию управления бизнес-процессами (например, движок jBPL).

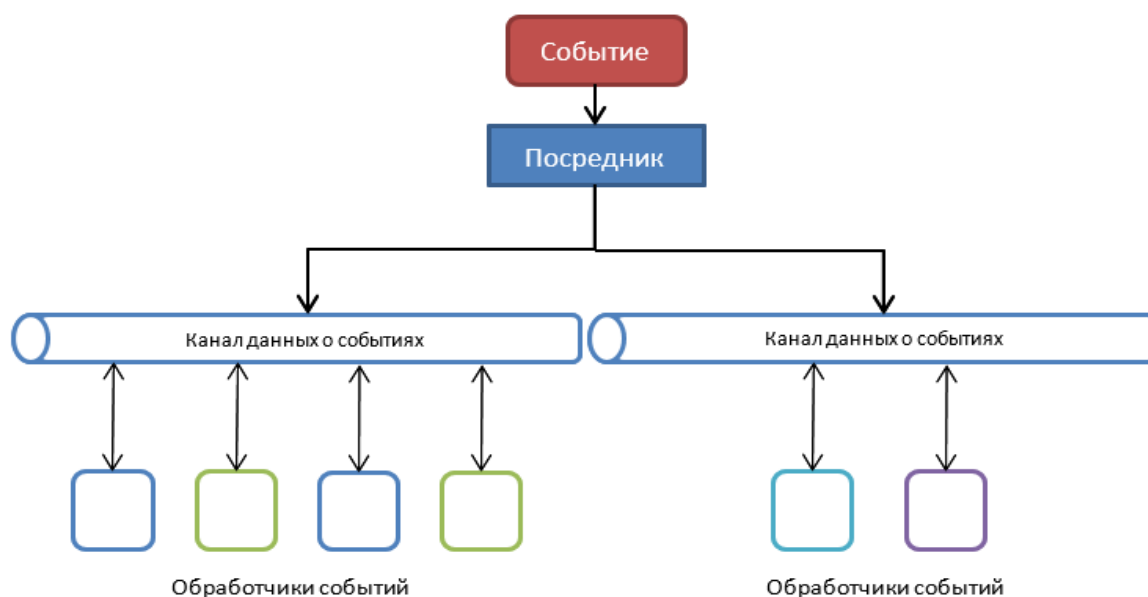


Рис. 2.57 Событийно-ориентированная архитектура

Архитектура, управляемая событиями – это относительно сложный паттерн. Причиной тому – его распределенная и асинхронная природа. Необходимо решить проблемы фрагментации сети, обрабатывать ошибки очереди событий и так далее. Плюсами этой архитектуры могут служить высокая производительность, легкость развертки и широкие возможности масштабирования.

Микроядерная архитектура. Паттерн состоит из двух компонентов: основной системы (ядра) и плагинов. Ядро содержит минимум бизнес-логики, но руководит загрузкой, выгрузкой и запуском необходимых плагинов. Таким образом, плагины оказываются несвязанными друг с другом, Рис. 2.58.

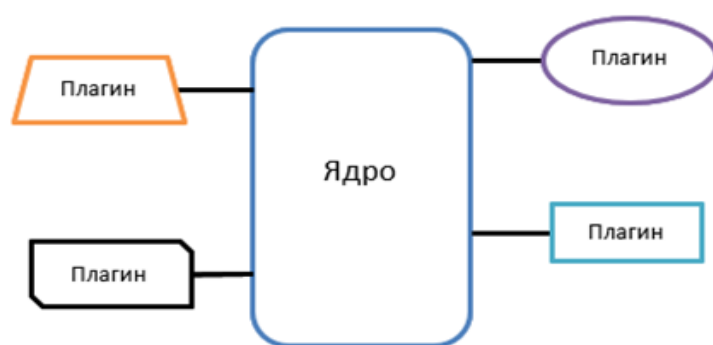


Рис. 2.58 Микроядерная архитектура

Так как плагины могут разрабатываться независимо друг от друга, то системы обладают очень высокой гибкостью и, как следствие, легко тестируются. Производительность приложения напрямую зависит от количества подключенных и активных модулей.

На основе микроядерной архитектуры реализован Eclipse IDE. Eclipse без надстроек – это совершенно пустой редактор. Однако с добавлением плагинов пустой редактор становится в полезным и настраиваемым инструментом разработки. Еще один представитель этой архитектуры – браузер, плагины позволяют расширить его функциональность.

3. ПРИМЕРЫ ПРОЕКТИРОВАНИЯ СИСТЕМ

3.1 Разработка АРМ регистратуры больницы

Описание предметной области задачи

Деятельность любого учреждения во многом зависит от качества обработки, хранения и распространении информации, правильности ведения документооборота и контроля каждой операции. Для регистратуры больницы необходимо разработать систему, способную вести систематический учет пациентов учреждения, отвечающую определенным функциональным требованиям.

Основными нефункциональными требованиями системы будут ее адаптация к изменяющимся условиям, ее пропускная способность, минимальное время реакции системы на запрос, безотказная работа системы, простота эксплуатации и поддержки системы.

Для того чтобы описать предметную область АРМ, необходимо описание структуры данного учреждения, ключевых понятий предмета системы, их характеристик и значений для системы.

На этапе анализа и проектирования необходимо разработать и представить процессы функционирования учреждения с использованием методик DFD и IDEF0, описать процессную модель функционирования предприятия с определением структуры хранимых данных. А также разработать логическую и физическую структуры базы данных с использованием методологии IDEF1.

Цель работы: разработка проекта автоматизированного рабочего места регистратора информационной системы больницы.

Задачи проектирования:

- Описать предметную область.
- Разработать и представить процессы функционирования регистратуры с использованием методик DFD и IDEF0.
- Разработать логическую и физическую структуры базы данных с использованием методологии IDEF1.

Больница – это лечебно-профилактическое учреждение, оказывающее населению стационарную медицинскую помощь, а в случае объединения с поликлиникой и амбулаторно-поликлиническую помощь. В областных (краевых, республиканских) и центральных районных больницах поликлиника является обязательным структурным подразделением; в городских больницах это подразделение может отсутствовать.

Основная задача больниц - оказание медпомощи в тех случаях, когда проведение лечения или сложных диагностических исследований невозможно в амбулаторно-поликлинических условиях. Если выздоровление или улучшение состояния больного может быть быстрее достигнуто в ста-

ционаре, чем при амбулаторном лечении; если пребывание больного дома представляет угрозу для окружающих (при ряде инфекционных и венерических заболеваний, некоторых видах психических расстройств и др.). На Рис. 3.1 представлена структура больницы.

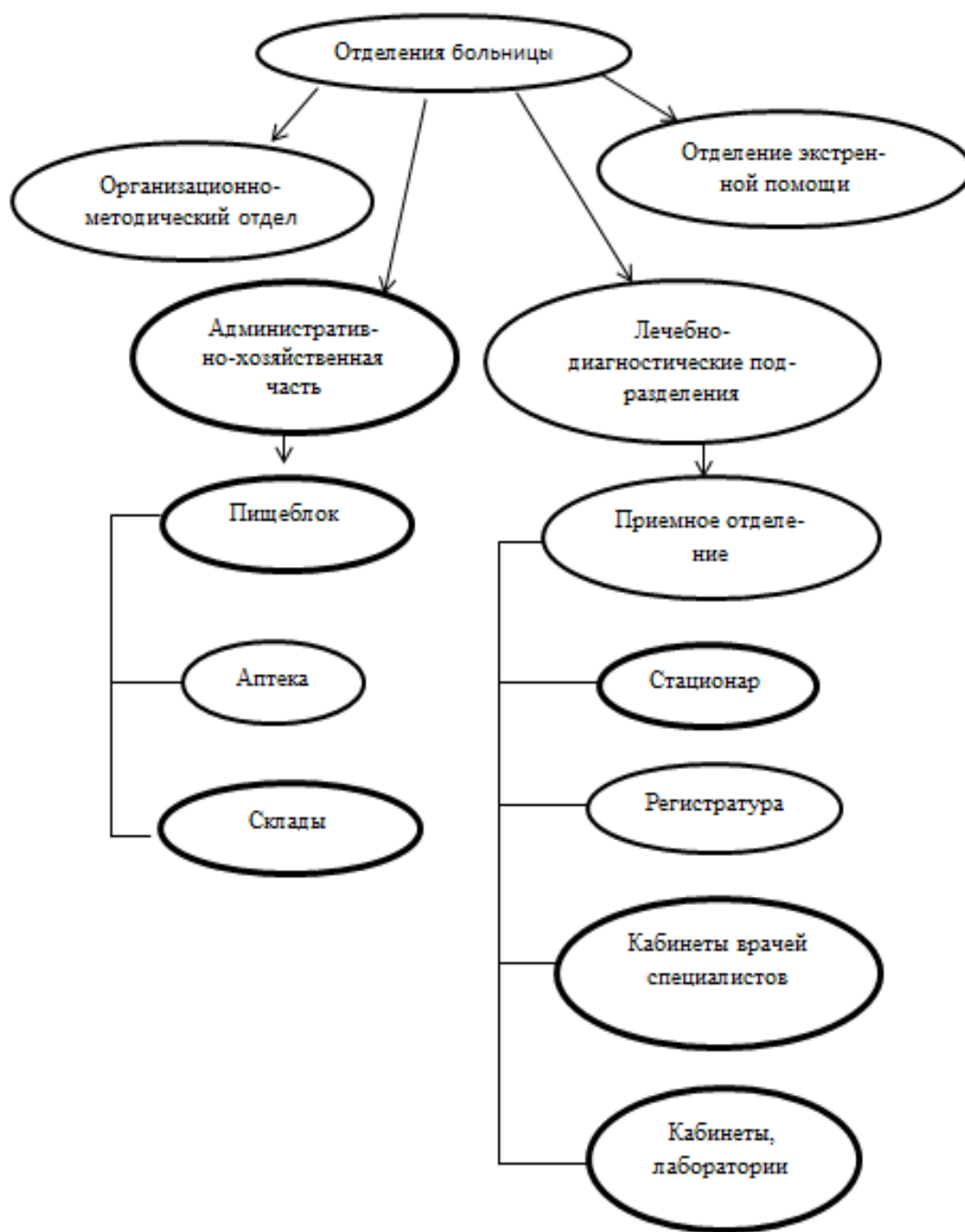


Рис. 3.1 Структура больницы

Автоматизированная система должна позволять регистратору регистрировать пациентов на прием и просматривать информацию о том, к какому участку принадлежит пациент, а еще прикреплять пациентов к участ-

кам. Пациенты имеют возможность сами регистрироваться, выбирая время приема и специалиста. Также пациенты могут зарегистрироваться к специалисту через регистратора.

В больнице существует база данных, в которой содержится информация о графике работы врачей. Регистрация на прием осуществляется следующим образом: пациент звонит или обращается лично в регистратуру с просьбой записать его к определенному врачу. Регистратор открывает расписание данного врача и записывает пациента на подходящую ему дату и время. Если пациент не определился со специалистом, то регистратор выбирает специалиста по участку, к которому принадлежит пациент. Также пациент может сам записаться к специалисту через Интернет, выбрав специалиста, дату и время приема.

Пациент выбирает специалиста, а затем ему предлагается выбрать дату работы специалиста и время. По такому же принципу работает регистратор. Регистратор или пациент могут корректировать запись к специалисту или вообще удалить ее.

Пользователи системы:

- 1 Регистратор – регистрирует на прием, прикрепляет к участку и просматривает информацию о пациентах.
- 2 Пациент – регистрируется на прием.
- 3 Администратор системы – сопровождение и конфигурирование системы.

Постановка задачи на проектирование АРМ регистратора больницы

Общие сведения

- 1) полное наименование системы - «АРМ регистратора больницы»;
- 2) наименование предприятий разработчика и заказчика (пользователя) системы - «Больница»;
- 3) перечень документов, на основании которых создается система - «Положение о работе поликлиники»
- 4) плановые сроки начала и окончания работы по созданию системы - установлены;
- 5) сведения об источниках и порядке финансирования работ - внутренние источники;

Цели и задачи создания (развития) системы

- 1) Цель - создание автоматизированного рабочего места регистратора информационной системы больницы.
- 2) Функциональные характеристики системы: система должна обеспечить возможность пациенту регистрироваться на прием, а регистратору регистрировать пациентов на прием и просматривать информацию о том, к какому участку принадлежит пациент, и прикреплять пациентов к участкам.

Характеристика объектов автоматизации

1) краткие сведения об объекте автоматизации или ссылки на документы, содержащие такую информацию;

Регистрация на прием осуществляется следующим образом: пациент звонит или обращается лично в регистратуру с просьбой записать его к определенному врачу. Регистратор открывает расписание данного врача и записывает пациента на подходящую ему дату и время. Если пациент не определился со специалистом, то регистратор выбирает специалиста по участку, к которому принадлежит пациент. Также пациент может сам записаться к специалисту через Интернет, выбрав специалиста, дату и время приема.

2) сведения об условиях эксплуатации объекта автоматизации.

Требования к системе

1) требования к системе в целом;

- требования к структуре и функционированию системы; система должна быть реализована на архитектуре клиент – сервер, включать в себя БД, пользовательский, системный и аппаратный интерфейсы, обеспечивать координацию работы и доступа всех пользователей в систему.

- требования к численности и квалификации персонала системы и режиму его работы; система должна обеспечивать работу всех пользователей в количестве до 100 клиентов в режиме 24 часа, 7 дней в неделю, квалификация – пользователь ИТ.

- требования к защите информации от несанкционированного доступа; пациенты не имеют доступа к базе данных. Ее редактирование и заполнение осуществляется администратором и/или регистратором. Вход всех пользователей в систему осуществляется по логину и паролю.

- требования по сохранности информации при авариях; осуществляется резервное копирование базы данных раз в сутки.

2) требования к функциям (задачам), выполняемым системой; система должна обеспечивать выполнение следующих функций:

- Входить в систему;
- Просмотр информации о пациентах;
- Прикрепление пациента к участку;
- Регистрация пациента;
- Запись к врачу на прием согласно расписанию;
- Саморегистрация пациента.

3) требования к видам обеспечения.

Для информационного обеспечения требования:

к составу, структуре и способам организации данных в системе; определить входные и выходные данные и структуру данных для хранения;

к контролю, хранению, обновлению и восстановлению данных; база данных должна дублироваться раз в сутки.

Содержание БД – регистратор/администратор, пациент, расчетная система, график работы специалистов, база пациентов.

Входные данные:

1. ФИО пациента
2. Логин регистратора
3. ФИО специалиста
4. Пароли для входа

Выходные данные:

1. Информация о пациенте
2. Информация о прикреплении
2. Запись на прием

Для лингвистического обеспечения системы - требования к способам организации диалога. Диалог должен происходить в интерактивном режиме на русском языке.

Для программного обеспечения системы - перечень покупных программных средств, а также требования:

к независимости программных средств от используемых технических средств и операционной среды; операционная среда Windows 10.

Для технического обеспечения системы - требования:

к видам технических средств, сеть персональных компьютеров клиентской части с единым сервером для Базы данных.

Разработка процессов функционирования регистратуры с использованием методик DFD и IDEF0

SADT (Structured Analysis and Design Technique) или IDEF0 – методология анализа и проектирования систем. Основным рабочим элементом при моделировании является диаграмма.

Построение SADT-модели начинается с представления всей системы в виде простейшего компонента – контекстной диаграммы, состоящей одного блока и дуг, изображающих интерфейсы с функциями вне системы. Поскольку единственный блок отражает систему как единое целое, имя, указанное в блоке, является общим. Это верно и для интерфейсных дуг – они также соответствуют полному набору внешних интерфейсов системы в целом.

Интерфейсы связей в SADT представляют собой наборы предметов и маркируются текстами. Входные связи отражают предметы, используемые и преобразуемые действиями. Управляющие связи обычно изображают информацию, управляющую действиями. Выходные связи изображают предметы, в которые преобразуются входы. Связи механизмов отражают реализацию процессов.

Затем блок, который представляет систему в качестве единого модуля, детализируется на другой диаграмме с помощью нескольких блоков, соединенных интерфейсными дугами связей. Каждая детальная диаграмма является декомпозицией блока из диаграммы предыдущего уровня. Формируется иерархия диаграмм.

На Рис.3.2 представлена контекстная диаграмма IDEF0 для процесса работы регистратуры. Выделены входы, выходы, механизмы и управления.

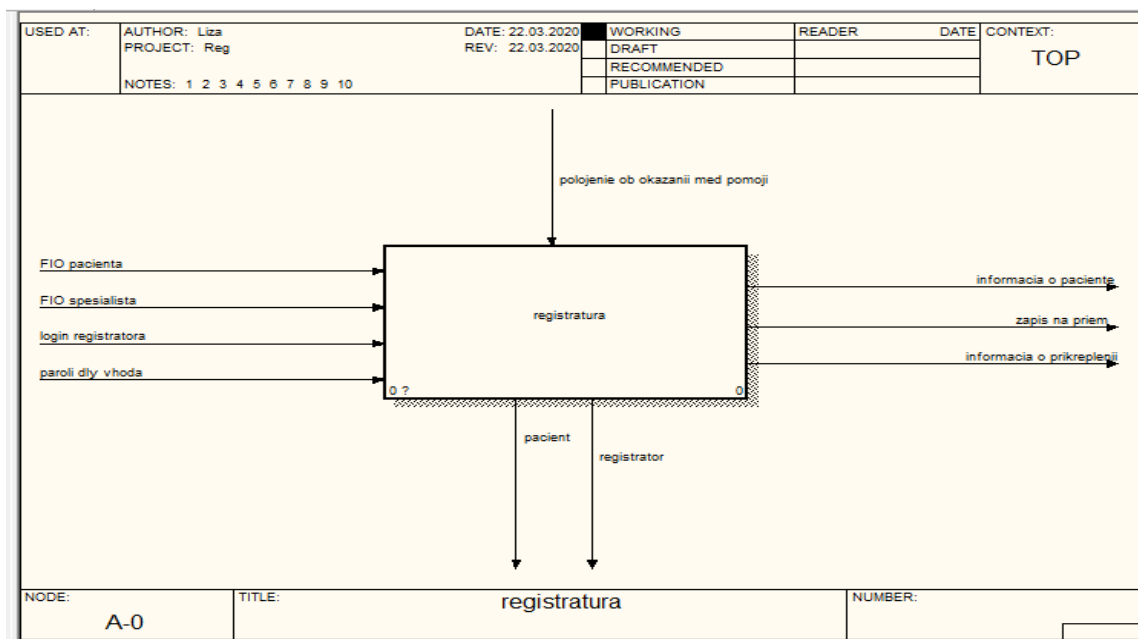


Рис. 3.2 Контекстная диаграмма IDEF0 «Регистратура»

На Рис. 3.3 представлена диаграмма декомпозиции верхнего уровня IDEF0. В ней выделены основные процессы, такие как: открыть регистрацию, найти пациента, открыть график работы специалиста, записать на прием. Все это может быть выполнено как самим пациентом, так и регистратором.

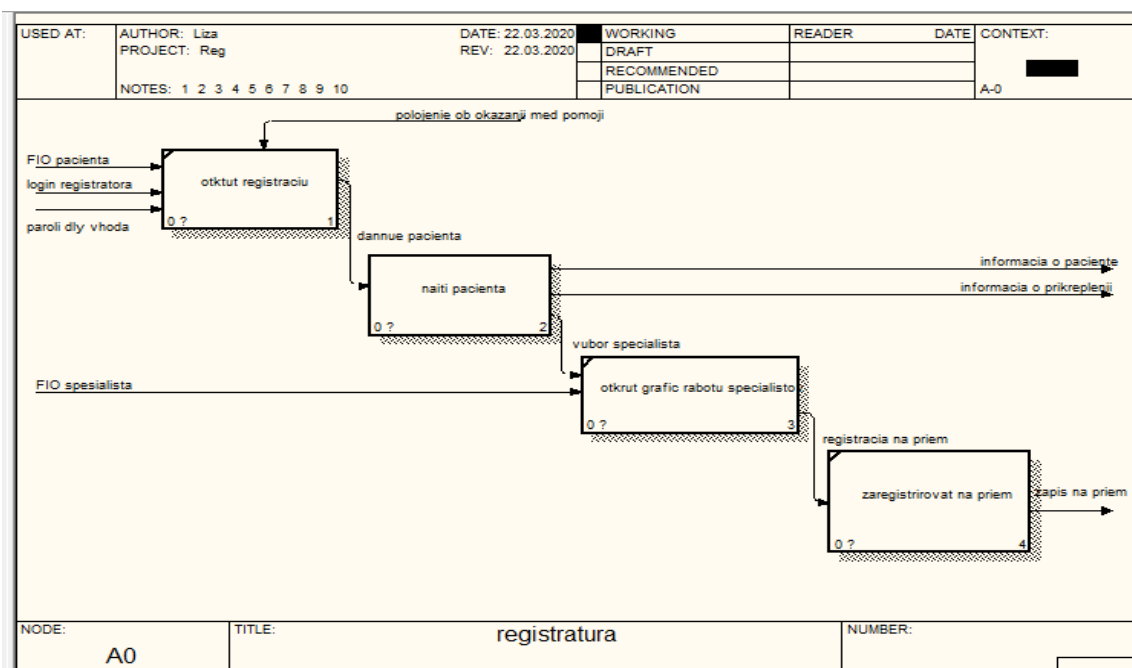


Рис. 3.3 Диаграмма декомпозиции процесса «регистрации пациента на прием»

Следующим этапом идет декомпозиция одного из процессов. Была произведена декомпозиция процесса «найти пациента». В нем было выделено 3 процесса: ввести ФИО пациента, найти прикрепление, прикрепить пациента. В процессе прикрепить пациента была введена обратная связь на случай, если у пациента не было изначально найдено прикрепления (Рис. 3.4).

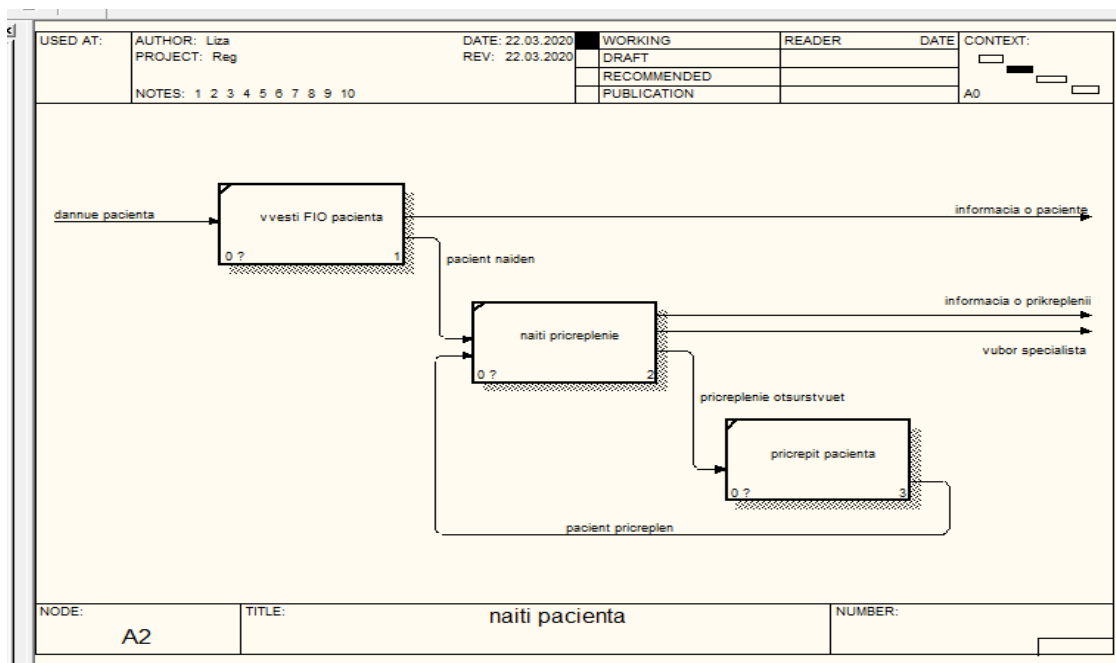


Рис. 3.4 Диаграмма декомпозиции IDEF0 процесса «найти пациента»

Информационная модель представляет собой DFD (Data Flow Diagrams) диаграмму потоков данных. Методология графического структурного анализа, описывающая внешние по отношению к системе источники и адресаты данных, логические функции, потоки данных и хранилища данных, к которым осуществляется доступ. Основные компоненты DFD-модели:

Процесс – функция или последовательность действий, которые нужно предпринять, чтобы данные были обработаны.

Внешние сущности – это любые объекты, которые не входят в саму систему, но являются для нее источником информации либо получателями какой-либо информации из системы после обработки данных.

Хранилище данных – это накопитель информации. В хранилище можно записывать данные и извлекать их.

Связь или поток данных – компонент, который показывается взаимодействие источников информации с приемниками.

На Рис.3.5 представлен топ DFD диаграммы, на котором изображена внешняя сущность и процесс регистрации на прием.

Далее было необходимо декомпонировать топ DFD диаграммы. В данной декомпозиции было выделено 4 процесса: выбор пациента, проверка прикрепления, выбор специалиста, выбор даты. Все процессы связаны между

собой информационными связями. Также были выделены 2 хранилища данных. Первое называется база пациентов. Оно хранит всю информацию о пациентах. Из данного хранилища поступают и извлекаются данные для процесса выбора пациента. Второе хранилище – график работы специалистов. Оно используется для выбора специалиста, даты и времени приема (Рис. 3.6).

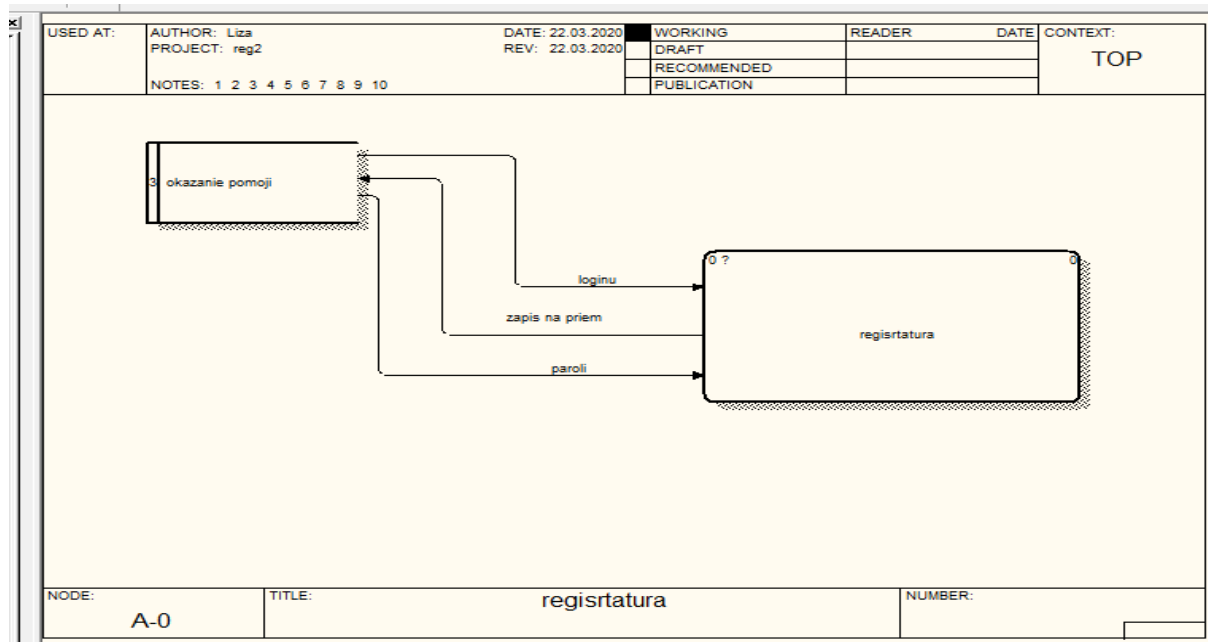


Рис. 3.5 Контекстная диаграмма DFD «Регистратура»

В данном пункте были составлены диаграммы по методологиям IDEF0 и DFD, которые визуализируют процессы, происходящие в работе регистратуры.

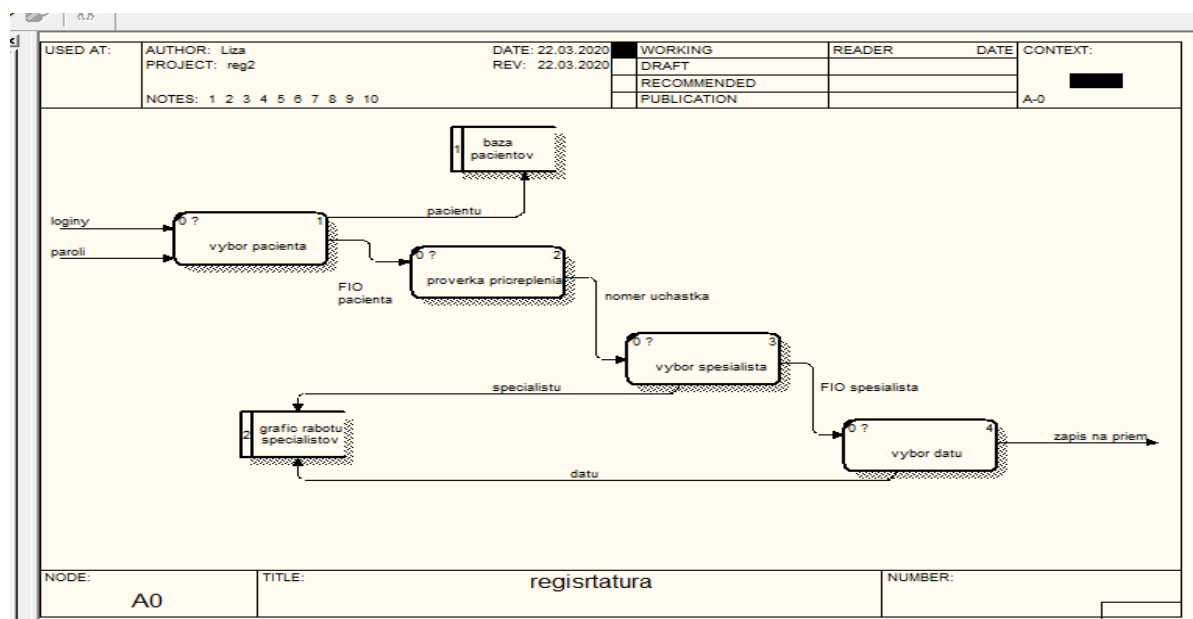


Рис .3.6 Диаграмма декомпозиции DFD процесса «регистрации пациента на прием»

Разработка и описание логической и физической структуры БД

Логическая структура реляционной базы данных - это представление информационных объектов модели данных, каждый объект реализован соответствующей реляционной таблицей. Структура таблицы определяется реквизитным составом соответствующего информационного объекта. Ключевые реквизиты объекта образуют уникальный ключ реляционной таблицы (Рис. 3.7).

Физическая организация БД - это совокупность методов и средств размещения данных во внешней памяти и определяет способы организации данных на носителях и методы доступа к данным. Ниже представлена данная база в Таблицах 3.1 – 3.4.

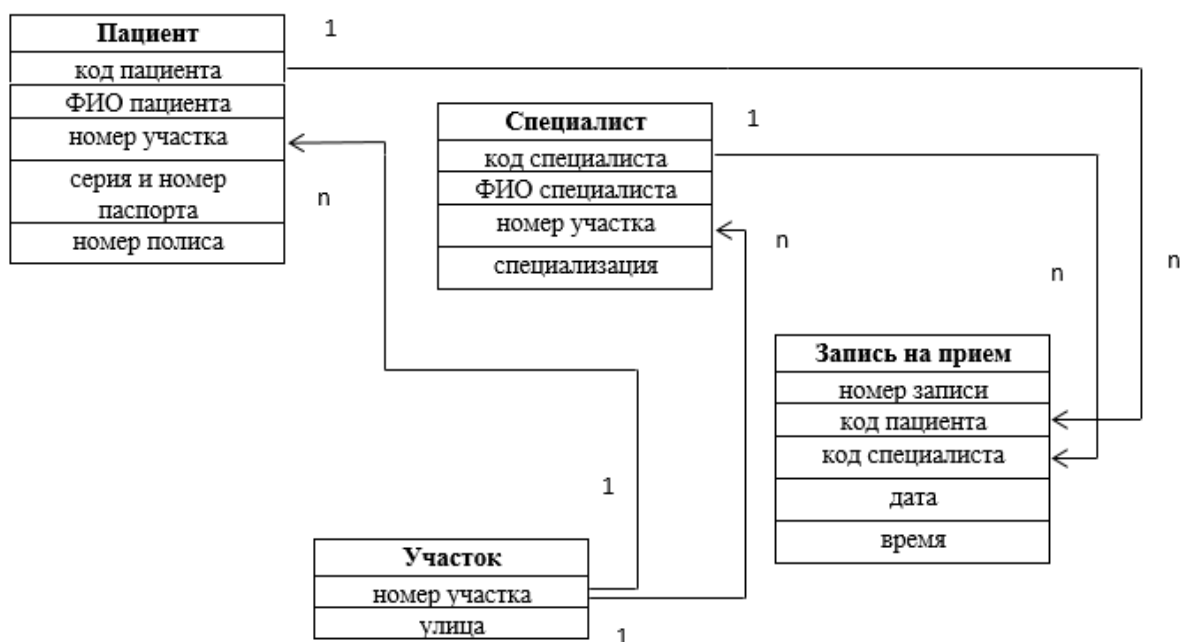


Рис. 3.7 Логическая структура БД

Таблица 3.1

Структура таблицы Пациент

Название	Тип	Размер
Код пациента	int	10
ФИО пациента	char	100
Номер участка	int	4
Серия и номер паспорта	int	15
Номер полиса	boolean	20

Таблица 3.2

Структура таблицы Специалист

Название	Тип	Размер
Код специалиста	int	10
ФИО специалиста	char	100
Номер участка	int	4
Специализация	char	20

Таблица 3.3

Структура таблицы Запись на прием

Название	Тип	Размер
Номер записи	int	10
Код пациента	int	10
Код специалиста	int	10
Дата	date	10
Время	time	8

Таблица 3.4

Структура таблицы Участок

Название	Тип	Размер
Номер участка	int	4
Улица	char	200

Основным структурным компонентом базы данных является таблица. При разработке структуры таблицы, прежде всего, необходимо задать поля, определив их свойства. В таблицах базы данных содержатся все основные сведения и данные.

Описание приложения по реализации методики расчета выбранного процесса

В данной работе в качестве СУБД была выбрана база данных MS Access. Работа начинается с создания таблиц. Были созданы таблицы: запись на прием, пациент, специалист, участок. Был задан тип данных для каждого поля и первичные ключи. На Рис. 3.8 представлена таблица базы данных «запись на прием». На Рис. 3.9 создана таблица «пациент». На рис. 3.10 изображена таблица «специалист», а на рис.11 – таблица «участок».

Имя поля	Тип данных
Номер записи	Счетчик
Код пациента	Числовой
Код специалиста	Числовой
Дата	Дата и время
Время	Дата и время

Рис. 3.8 Таблица базы данных «запись на прием»

Имя поля	Тип данных
Код пациента	Счетчик
ФИО пациента	Короткий текст
Номер участка	Числовой
Серия и номер паспорта	Короткий текст
Номер полиса	Короткий текст

Рис. 3.9 Таблица базы данных «пациент»

Далее была создана схема данных. Схема данных - это графического отображения логической структуры базы данных и система взаимодействия данных. На основе схемы данных работает пользовательский инструмент конструирования многотабличных форм, запросов, отчетов и ра-

ботает поддержка целостности связанных данных при вводе и корректировке данных в таблицах (Рис. 3.10-3.12).

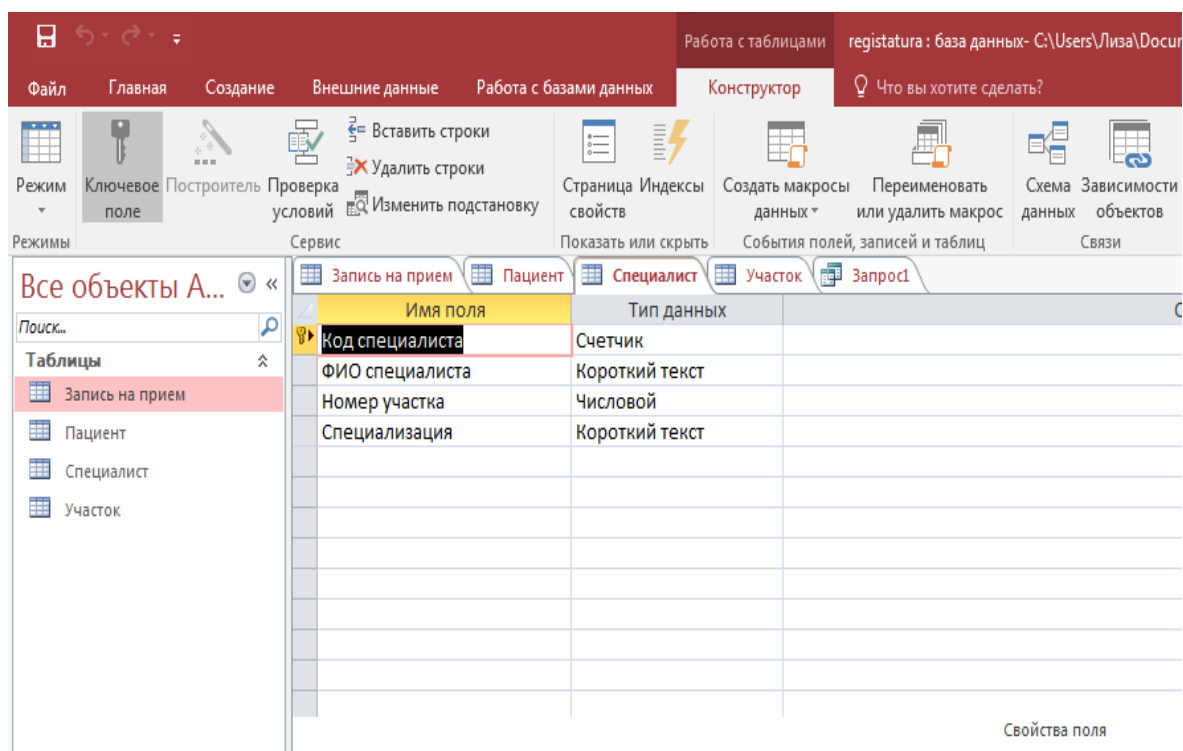


Рис. 3.10 Таблица базы данных «специалист»

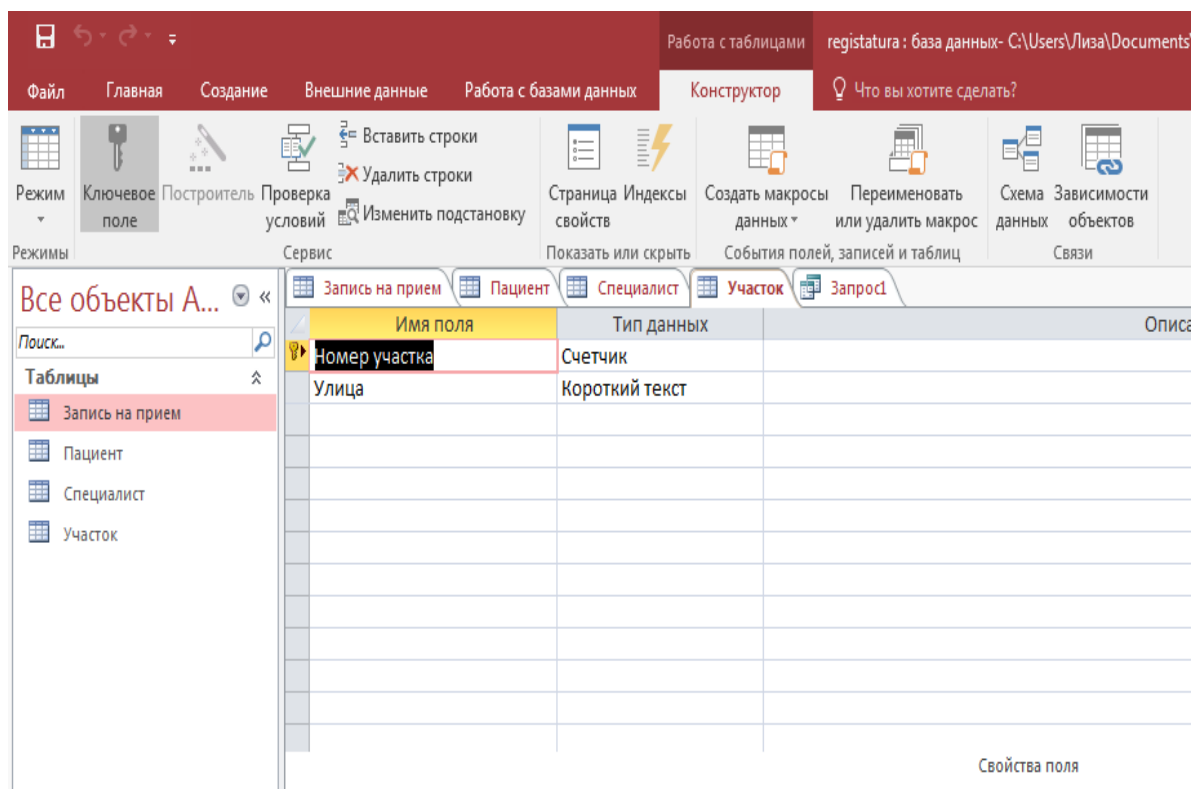


Рис. 3.11 Таблица базы данных «участок»

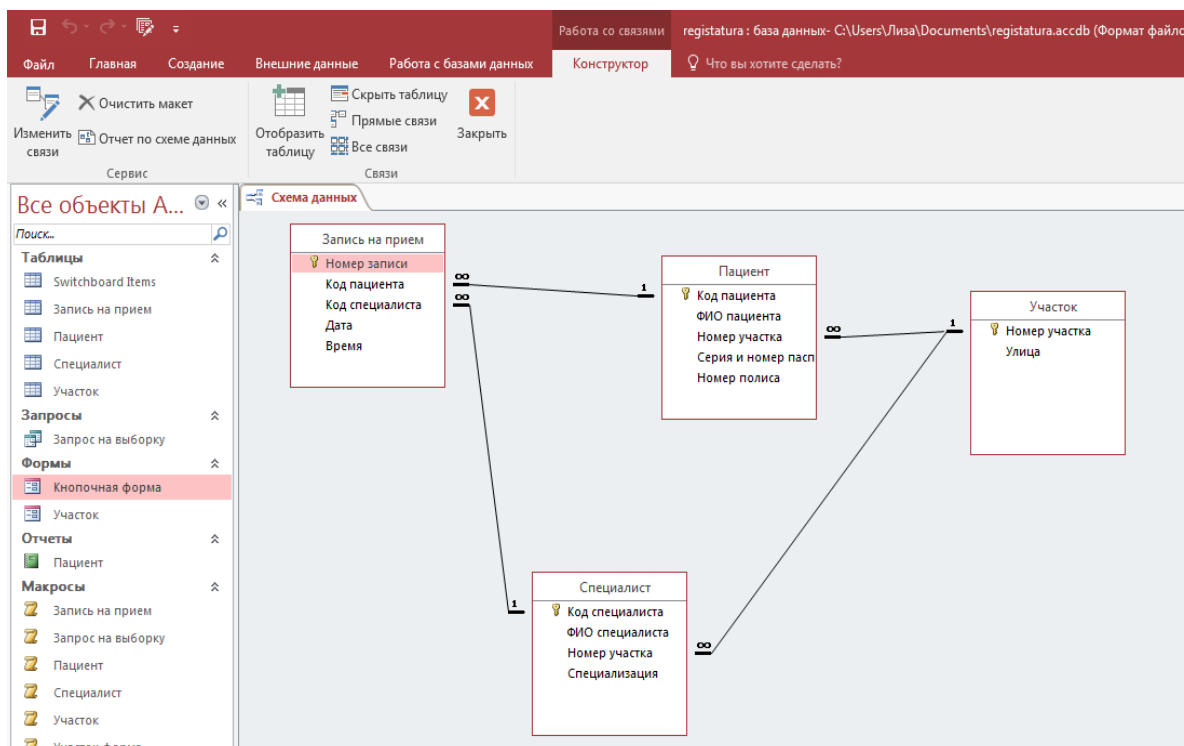


Рис. 3.12 Схема данных

На Рис. 3.13 изображен запрос на выборку. В условиях отбора было названо ФИО специалиста (Рис. 3.14).

Запрос на выборку (Query Design View) в Microsoft Access. Включает следующие таблицы и связи:

- Запись на прием** (Appointment):
 - Номер записи (Primary Key)
 - Код пациента
 - Код специалиста
 - Дата
 - Время
- Пациент** (Patient):
 - Код пациента (Primary Key)
 - ФИО пациента
 - Номер участка
 - Серия и номер пасп
 - Номер полиса
- Специалист** (Specialist):
 - Код специалиста (Primary Key)
 - ФИО специалиста
 - Номер участка
 - Специализация

Связи:

- Запись на прием (1) — Пациент (∞)
- Запись на прием (1) — Специалист (∞)

Условия отбора (Criteria):

- Специалист: ФИО специалиста = Михайловская Вале

Результаты (Results):

Номер записи	Код пациента	Код специалиста	ФИО пациента	ФИО специалиста
1	1	1	Михайловская Вале	Михайловская Вале

Рис. 3.13 Запрос на выборку

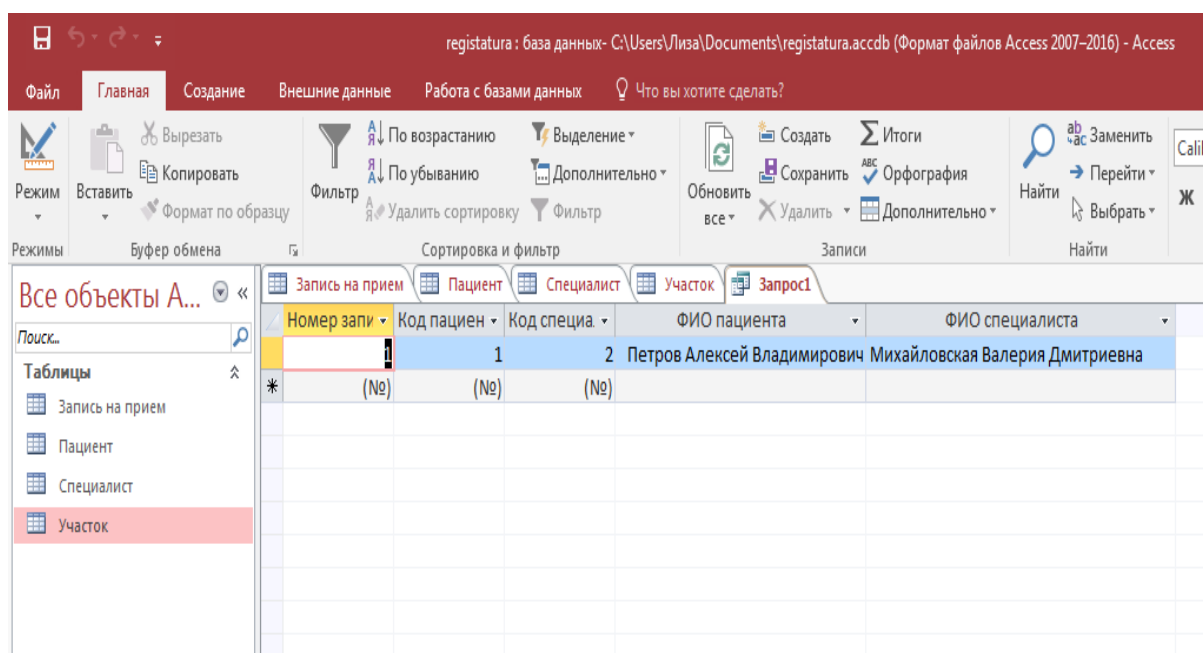


Рис. 3.14 Запрос на выборку по условию

Далее была создана форма участок (Рис. 3.15).

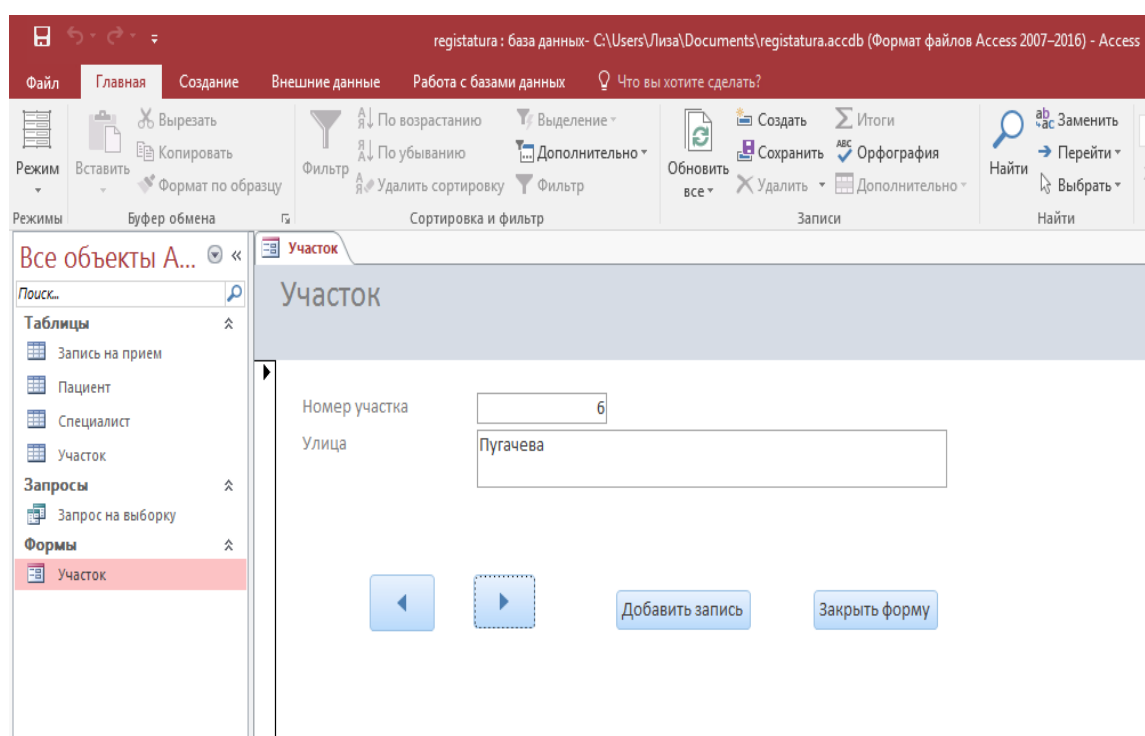


Рис. 3.15 Форма «участок»

На Рис. 3.16 представлен отчет «пациенты».

Далее была создана главная кнопочная форма. Из нее можно открыть таблицы, отчеты, формы и запросы (Рис. 3.17).

Также в каждой кнопочной форме реализованы формы каждой таблицы (Рис. 3.18). С отчетами, формами, запросами была проделана аналогичная работа. В каждой кнопочной форме есть возможность вернуться на шаг назад.

Была создана реляционная база данных MS Access, где реализованы основные компоненты: таблицы, формы, запросы и отчеты. Также для работы с кнопочной формой были созданы макросы, представляющие собой некие контейнеры, в которых хранятся простейшие команды (макрокоманды).

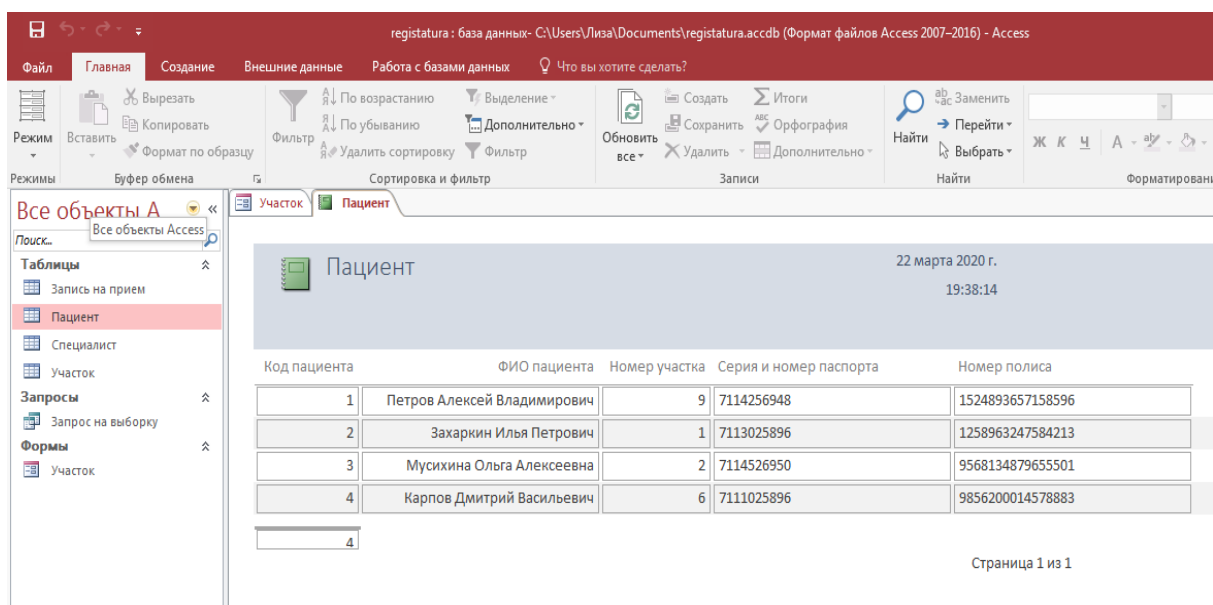


Рис. 3.16 Отчет «пациенты».

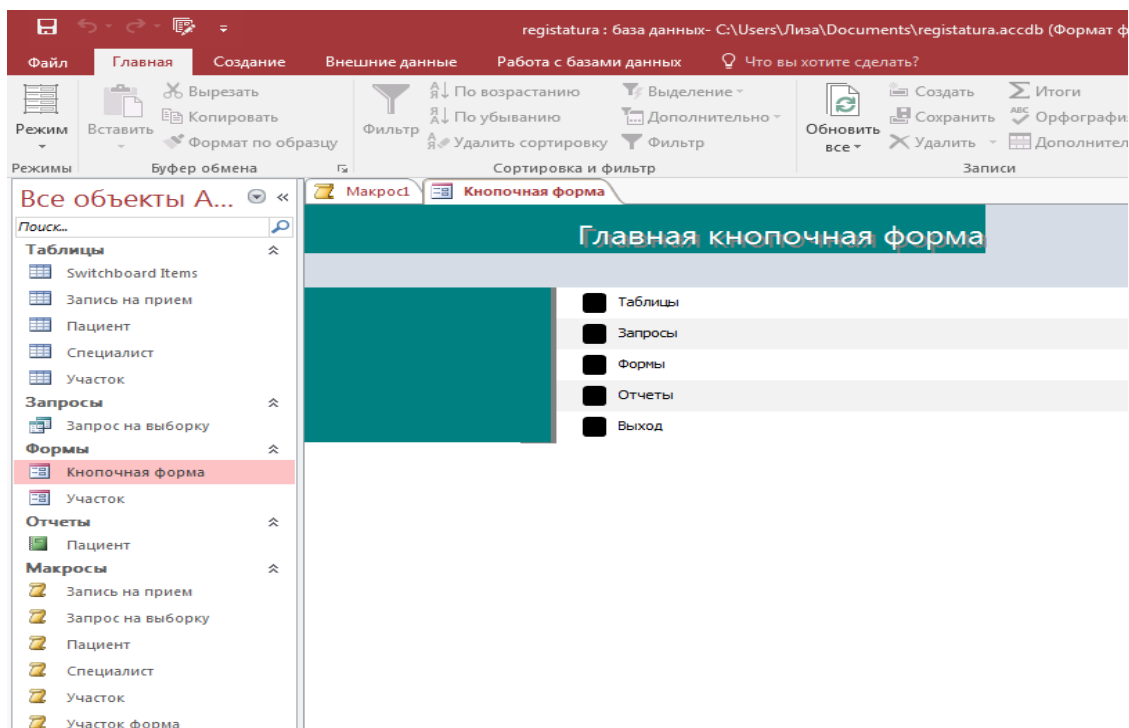


Рис. 3.17 Главная кнопочная форма

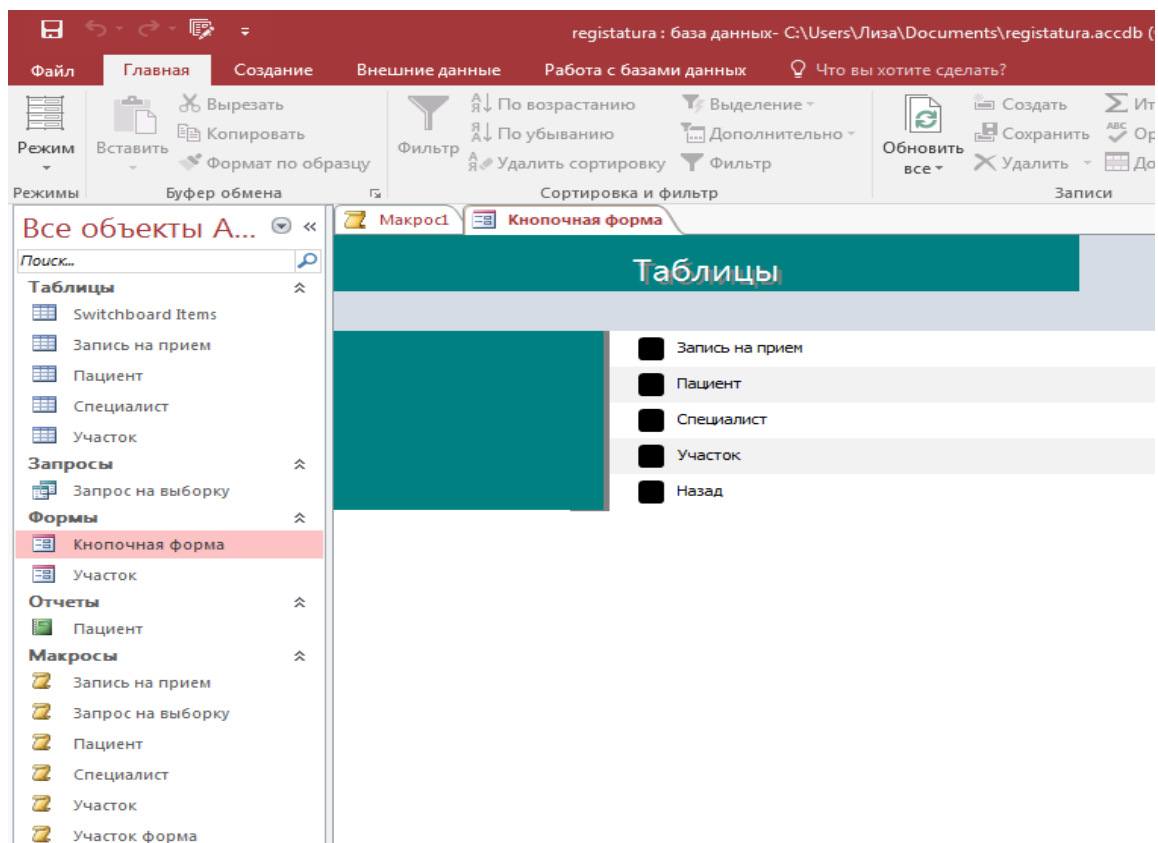


Рис. 3.18 Кнопочная форма таблицы

Была произведено описание процесса работы регистратуры с целью автоматизации обработки информации. Процесс работы включает в себя постановку задачи (словесное описание использования системы) и определение сценариев работы системы. Определены пользователи и артефакты информационной системы: регистратор/администратор, пациент, график работы специалистов, база пациентов.

Составлен словарь предметной области: регистратор, пациент, список пациентов, график работы специалистов, прием, запись.

Для разработки автоматизированной системы были определены характеристики объектов автоматизации и функциональные требования, требования к входным, выходным и хранимым данным, и требования к системе в целом.

Была проведена формализация процесса регистрации пациентов на прием с использованием методологии IDEF0 и DFD. В IDEF0 создана контекстная диаграмма и декомпозирована на четыре блока, а также декомпозированы все процессы в декомпозированной диаграмме. Была создана концептуальная модель DFD и декомпозирована на процессы передачи информации на первом уровне.

Произведено описание сущностей и их атрибутов в модели IDEF1X. Построены логическая и физическая модели данных. Также была создана база данных в среде разработки.

3.2 Проектирование системы «Мобильный сервис вызова на дом врача для поликлиники города»

Описание предметной области системы

Система предназначена для автоматизации записи пациентов на прием к врачу.

При вызове врача на дом пациент может ставить отметку об экстренности. В зависимости от этих отметок система выстраивает для врача оптимальный маршрут посещения пациентов.

При записи на прием к определенному врачу, система указывает на ближайшее свободное время выбранного врача или другого свободного врача той же специализации. При невозможности записи пациент добавляется в лист ожидания, который просматривает администратор и при необходимости сообщает пациенту об освободившемся месте. Если пациент успешно записывается в необходимое время, то он информируется о предстоящей записи.

Категории пользователей:

- администратор городской поликлиники;
- врач;
- пациент.

Основными сценариями в системе являются вызов врача на дом и запись на приём к врачу.

Вызов на дом врача:

При вызове врача на дом, пациент вносит в систему следующую информацию:

1. отметка об экстренности вызова (экстренный - требует незамедлительного выезда врача, несрочный – ставится в очередь пациентов, которых сегодня должен посетить врач);
2. номер телефона, адрес;
3. номер медицинского страхового полиса (если случай несрочный).

Запись на приём в поликлинику:

При записи на приём в поликлинику, пациент вносит в систему следующую информацию:

1. номер медицинского страхового полиса;
2. выбирает нужного врача;
3. дату и время посещения.

Если есть свободное место для записи, то формируется заявка, если нет – пациент добавляется в лист ожидания. В листе ожидания хранится ФИО пациента, номер страхового полиса и номер телефона для оповещения. При появлении свободного места приходит уведомление/сообщение о том, что место освободилось и можно записаться.

Функционирование системы моделирует бизнес-процесс нотации EPC, см. рис. 3.23.

1. Система должна обеспечивать выполнение следующих функций:

- Вызов врача на дом
- Запись на прием в поликлинику
- Координация заявок на вызовы и приемы
- Просмотр графика работы врачей
- Просмотр маршрута врачей
- Выстраивание оптимального маршрута до пациента
- Контроль выезда врача на дом
- Добавление в лист ожидания при занятых местах
- Оповещение пациента о предстоящем приеме
- Хранение данных о заявках

2. Решения по архитектуре системы

Доступ к системе предоставляется следующим образом: на клиентском оборудовании запускается браузер, через который осуществляется доступ к приложению, а физически — обращение к серверу приложений. Передача данных от сервера приложений клиенту осуществляется по этому каналу. Система состоит из следующих основных компонентов: клиентское приложение (браузер), сервер приложений, сервер баз данных.

2. Решения по составу информации, объему, способам ее организации, видам машинных носителей, входным и выходным документам и сообщениям, последовательности обработки информации и другим компонентам

Входными данными в системе являются:

- Информация о пациенте (ФИО, номер страхового полиса, телефон)
- Информация по заявке (дата и время визита к врачу, №каб, фамилия врача, фамилия пациента)
- Информация по вызову (адрес, отметка об экстренности, номер телефона)

Выходными данными в системе являются:

- заявки, в которых хранится информация;
- списки пациентов, которые должны явиться на прием по врачам;
- маршрут врача;
- расписание работы врачей;
- уведомление о вызове на дом;
- уведомление о предстоящем приеме (SMS - оповещение, звонок);
- лист ожидания.

Входные и выходные сообщения описаны в документе «Входные и выходные сообщения» (Рис.3.24), входящем в состав эскизного проекта системы.

3. Решения по составу программных средств, языкам деятельности, алгоритмам процедур и операций и методам их реализации

Система представляет собой web-приложение. Инструменты разработки: среда проектирования StarUML, база данных SQLite, фреймворк Django, язык Python, средства разработки и оформления веб-страниц.

Словарь системы

Лист ожидания - документ, содержащий список пациентов, ожидающих записи на приём к врачу.

Расписание - это таблица, отражающая рабочее время врачей.

Маршрут - электронный онлайн-документ, который отражает порядок посещения пациентов врачом.

Заявка - выражение намерений воспользоваться любыми услугами поликлиники.

Подтверждение заявки - форма веб-интерфейса, уточняющая намерения пациента сделать вызов врача.

Оформление заявки - форма веб-интерфейса, с помощью которой пациент выбирает дату, время и врача или вызывает врача на дом.

Пациент - это человек, пользующийся услугами поликлиники. Абстракция содержит информацию о пациенте, необходимую для организации создания заявок (запись к врачу, вызов врача на дом, добавление в лист ожидания).

Проверка деталей заявки - форма веб-интерфейса, предотвращающая неверный ввод данных.

Просмотр маршрута - карта на форме веб-интерфейса, позволяющая администратору и врачу просматривать маршрут.

Ввод личных данных - форма веб-интерфейса, в которую пациент вводит необходимые данные для создания заявки.

Выбор типа заявки - форма веб-интерфейса, позволяющая выбрать тип заявки: запись к врачу или вызов врача на дом.

Диаграмма прецедентов – эскиз основного функционала системы, представлена на Рис.3.25. Не представлены диаграммы прецедентов Построение маршрута, Работа с Базой данных поликлиники.

Актеры - пользователи системы: Врач, Администратор, Пациент.

Актеры – внешние системы: База данных поликлиники, Система геолокации (для построения маршрута врача)

Поток событий прецедента «Оформление заявки пациентом»:

Основной поток событий:

1. Прецедент начинается с того, что пациент выбирает вызов врача на дом или запись на прием к врачу на форме Оформление заявки.

A1. Пациент выбирает вариант вызова к врачу и запускает поток «Вызов врача на дом».

A2. Пациент выбирает вариант запись на прием к врачу и запускает поток «Запись к врачу».

2. Система добавляет заявки в базу данных заявок.

3. Вариант использования завершается.

Альтернативные потоки:

A1. Вызов врача на дом.

1. Система переходит к первому шагу оформления заявки на вызов: запрашивает у пациента экстренность случая.
 2. Пациент указывает экстренность случая.
 3. Система переходит ко второму шагу оформления заявки на вызов: запрашивает у пациента номер телефона и адрес пациента, при не экстренном вызове система также запрашивает номер страхового медицинского полиса.
- Е1. Пользователь ввел недостаточно данных при вызове врача на дом.
4. Система переходит к третьему шагу оформления заявки на вызов: просит подтверждение вызова.
- А3. Пациент подтверждает запрос вызова врача на дом и запускает поток «Подтверждение запроса».
- А4. Пациент отказывается от вызова врача на дом и запускает поток «Отказ от вызова».
5. Поток возвращается к первому этапу основного потока.
- А2. Запись к врачу.**
1. Система открывает режим просмотра графика работы врачей.
 2. Пациент выбирает нужного врача, дату и время посещения.
- А5. Пациент не смог выбрать нужную ему дату и время и открывает поток «добавление пациента в очередь».
3. Система переходит к первому шагу оформления заявки: запрашивает у пациента ФИО, номер телефона, номер страхового медицинского полиса.
 4. Пациент вводит ФИО, номер телефона и номер страхового полиса
 5. Система переходит ко второму шагу оформления заявки: показывает введенные пациентом данные и выбранного врача с нужной датой и временем посещения и просит подтвердить корректность введенных данных.
- Е2. Пользователь ввел недостаточно данных при записи к врачу.
6. Пациент подтверждает корректность данных.
 7. Система выводит сообщение о том, что заявка успешно обработана.
 8. Поток возвращается к первому этапу основного потока.
- А3. Подтверждение запроса.**
1. Система выводит сообщение о том, что вызов врача на дом обработан.
 2. Если случай был указан, как экстренный, то заявка добавляется после последней экстренной заявки. Если случай был указан, как не экстренный, то заявка добавляется в конец списка.
- Е3. Количество пациентов превышает допустимое значение.
3. Система возвращается ко второму этапу основного потока.
- А4. Отказ от вызова.**
1. Система выводит сообщение о том, что вызов отменен.
 2. Система возвращается к первому этапу основного потока.

A5. Добавление пациента в очередь.

1. Система добавляет пациента в очередь к необходимому ему врачу.
2. Система выводит сообщение о том, что пациент добавлен в лист ожидания и при появившемся свободном месте администратор уведомит его.
3. Поток возвращается к третьему этапу основного потока.

Потоки ошибок.

E1. Пользователь ввел недостаточно данных при вызове врача на дом.

1. Система выводит сообщение о том, что пользователь не указал всю необходимую информацию.
2. Поток возвращается к третьему шагу потока A1.

E2. Пользователь ввел недостаточно данных при записи к врачу.

4. Система выводит сообщение о том, что пользователь не указал всю необходимую информацию.
5. Поток возвращается ко второму шагу потока A2.

E3. Количество пациентов превышает допустимое значение.

1. Система выводит сообщение о том, что превышено допустимое значение и необходимо повторить попытку через некоторое время.
2. Поток возвращается к первому шагу основного потока.

Прототипы форм пользовательского интерфейса в клиентском мобильном приложении, Рис. 3.19-3.22

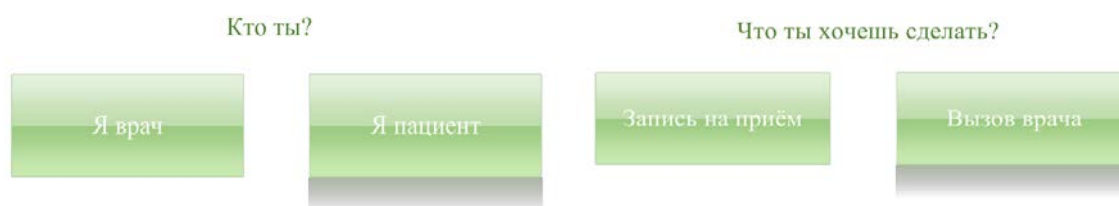


Рис. 3.19 Вход в систему и выбор сценария

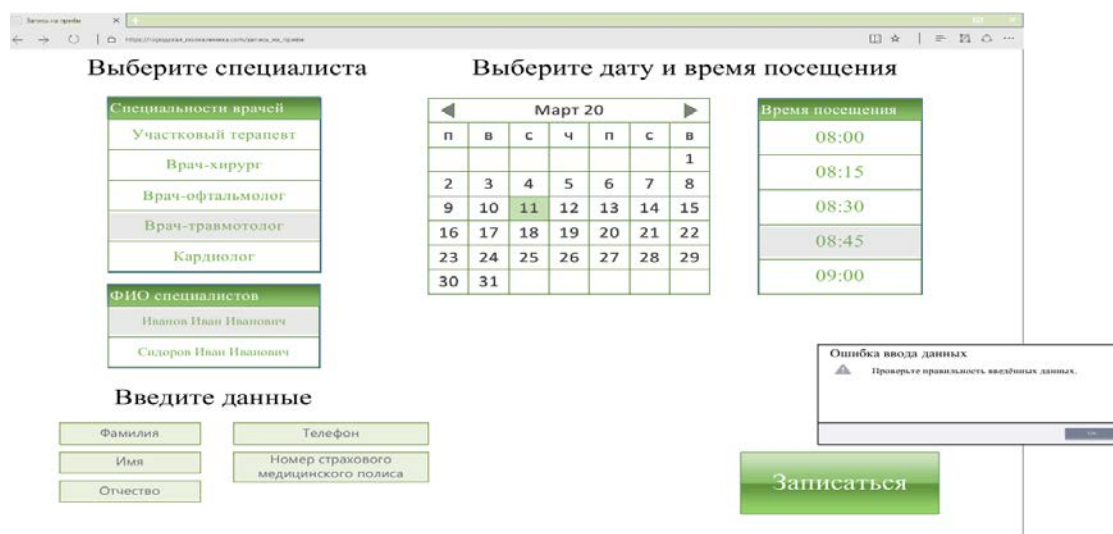


Рис. 3.20 Оформление заявки

ны взаимодействия классов для решения задачи (Рис. 3.29). Выявлены атрибуты и методы классов. Построены диаграммы классов логического уровня. Представлена диаграмма состояний классов. Разработана спецификация атрибутов и операций классов.

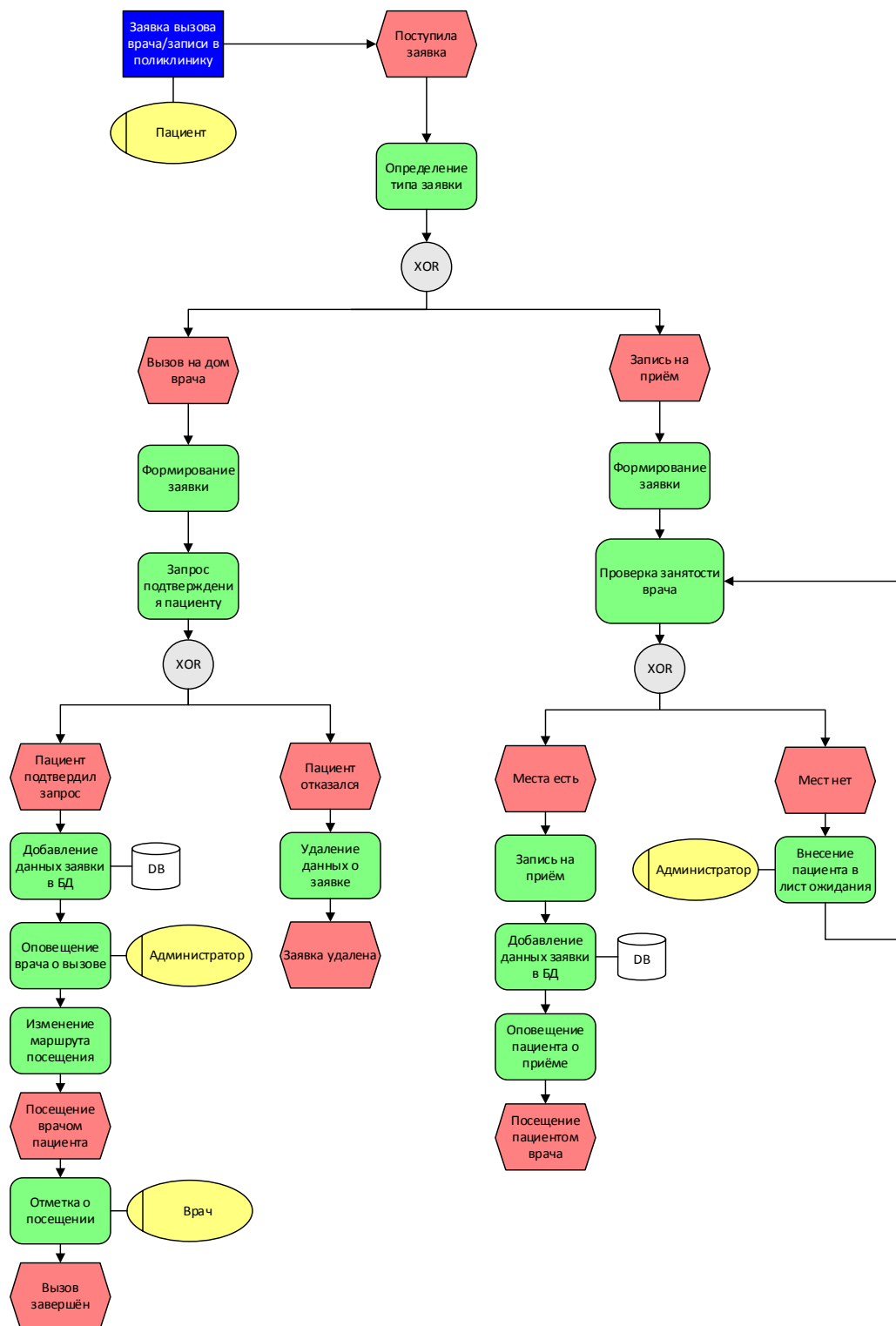


Рис. 3.23 Диаграмма бизнес-процесса в нотации EPC Мобильного сервиса вызова врача на дом

Входные и выходные сообщения системы

Таблица 1. –Параметры выходного сообщения «Заявки на вызов врача на дом»

Получатель информации	Пациенты
Идентификатор	Заявки на вызов врача на дом
Форма представления	Таблица данных
Периодичность выдачи	При поступлении новой заявки
Сроки выдачи и допустимое время задержки	Сразу после поступления новой заявки, без задержек
Атрибуты сообщения	Отметка об экстренности вызова, номер телефона пациента, адрес пациента, номер медицинского страхового полиса (при не экстренном вызове)
Получатель информации	Администратор городской поликлиники

Таблица 2 –Параметры выходного сообщения «Заявки на запись пациента к врачу»

Получатель информации	Пациенты
Идентификатор	Заявки на запись пациента к врачу
Форма представления	Таблица данных
Периодичность выдачи	По требованию
Сроки выдачи и допустимое время задержки	Сразу после поступления новой заявки, без задержек
Атрибуты сообщения	Дата визита, время визита, №каб., фамилия врача, ФИО пациента, номер телефона пациента, номер страхового медицинского полиса
Получатель информации	Администратор городской поликлиники

Таблица 3 –Параметры выходного сообщения «Списки пациентов, которые должны явиться на прием»

Получатель информации	Пациенты
Идентификатор	Списки пациентов
Форма представления	Электронный (doc) печатный документ
Периодичность выдачи	Ежедневно
Сроки выдачи и допустимое время задержки	В 8:00 каждый день + 1 час
Атрибуты сообщения	ФИО пациентов
Получатель информации	Врач

Таблица 4 –Параметры выходного сообщения «Маршрут врача»

Получатель информации	Пациенты
Идентификатор	Маршрут врача
Форма представления	Онлайн-электронный документ
Периодичность выдачи	При поступлении новой заявки
Сроки выдачи и допустимое время задержки	Сразу после поступления новой заявки, без задержек
Атрибуты сообщения	ФИО пациента; отметка об экстренности; адрес пациента
Получатель информации	Врач

Таблица 5 –Параметры выходного сообщения «Расписание работы врачей»

Получатель информации	Пациенты
Идентификатор	Расписание работы врачей
Форма представления	Электронный (doc) печатный документ
Периодичность выдачи	Раз в неделю
Сроки выдачи и допустимое время задержки	Пятница 8:00 + 1 час
Атрибуты сообщения	ФИО врача, №каб., время работы

Таблица 6 –Параметры выходного сообщения «Уведомление о вызове на дом»

Получатель информации	Пациенты
Идентификатор	Уведомление о вызове на дом
Форма представления	SMS – оповещение
Периодичность выдачи	При поступлении новой заявки
Сроки выдачи и допустимое время задержки	Сразу же после поступления новой заявки, без задержек
Атрибуты сообщения	Адрес пациента; отметка об экстренности
Получатель информации	Врач

Таблица 7 –Параметры выходного сообщения «Уведомление о предстоящем приеме»

Получатель информации	Пациенты
Идентификатор	Уведомление о предстоящем приеме
Форма представления	SMS – оповещение
Периодичность выдачи	Каждый день
Сроки выдачи и допустимое время задержки	За день до назначенного приема
Атрибуты сообщения	ФИО врача, время и дата посещения, №каб.
Получатель информации	Пациент

Таблица 8 –Параметры выходного сообщения «Лист ожидания»

Получатель информации	Пациенты
Идентификатор	Лист ожидания
Форма представления	Электронный (doc) документ
Периодичность выдачи	Каждый день
Сроки выдачи и допустимое время задержки	В 8:00 каждый день + 1 час
Атрибуты сообщения	ФИО пациента, ФИО врача, номер телефона пациента
Получатель информации	Администратор городской поликлиники

Таблица 9 –Параметры входного сообщения «Данные пациента для вызова врача на дом»

Получатель информации	Пациенты
Идентификатор	Данные пациента для вызова врача на дом
Форма представления	Таблица данных
Периодичность приема	По запросу пациента
Сроки выдачи и допустимое время задержки	Сразу после поступления
Атрибуты сообщений	Адрес пациента; отметка об экстренности; номер телефона
Источник информации	Пациент

Таблица 10 –Параметры входного сообщения «Данные пациента для записи к врачу»

Получатель информации	Пациенты
Идентификатор	Данные пациента для записи к врачу
Форма представления	Таблица данных
Периодичность приема	По запросу пациента
Сроки выдачи и время задержки	Сразу после поступления
Атрибуты сообщения	Дата визита, время визита, №каб., фамилия врача, ФИО пациента, номер телефона пациента, номер страхового медицинского полиса
Получатель информации	Пациент

Рис. 3.24 Выходные и входные сообщения системы

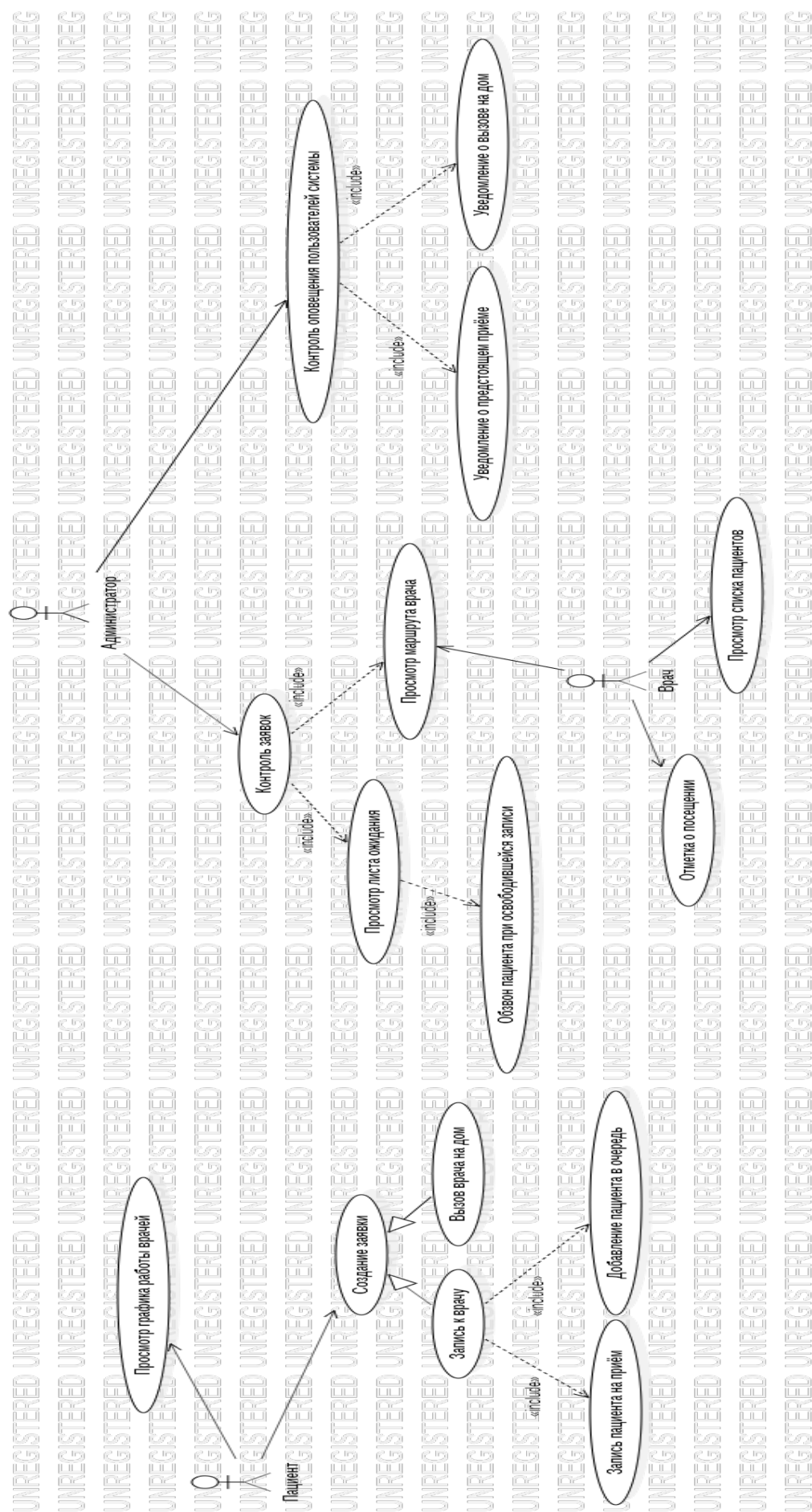


Рис. 3.25 Диаграмма прецедентов Мобильного сервиса вызова врача на дом

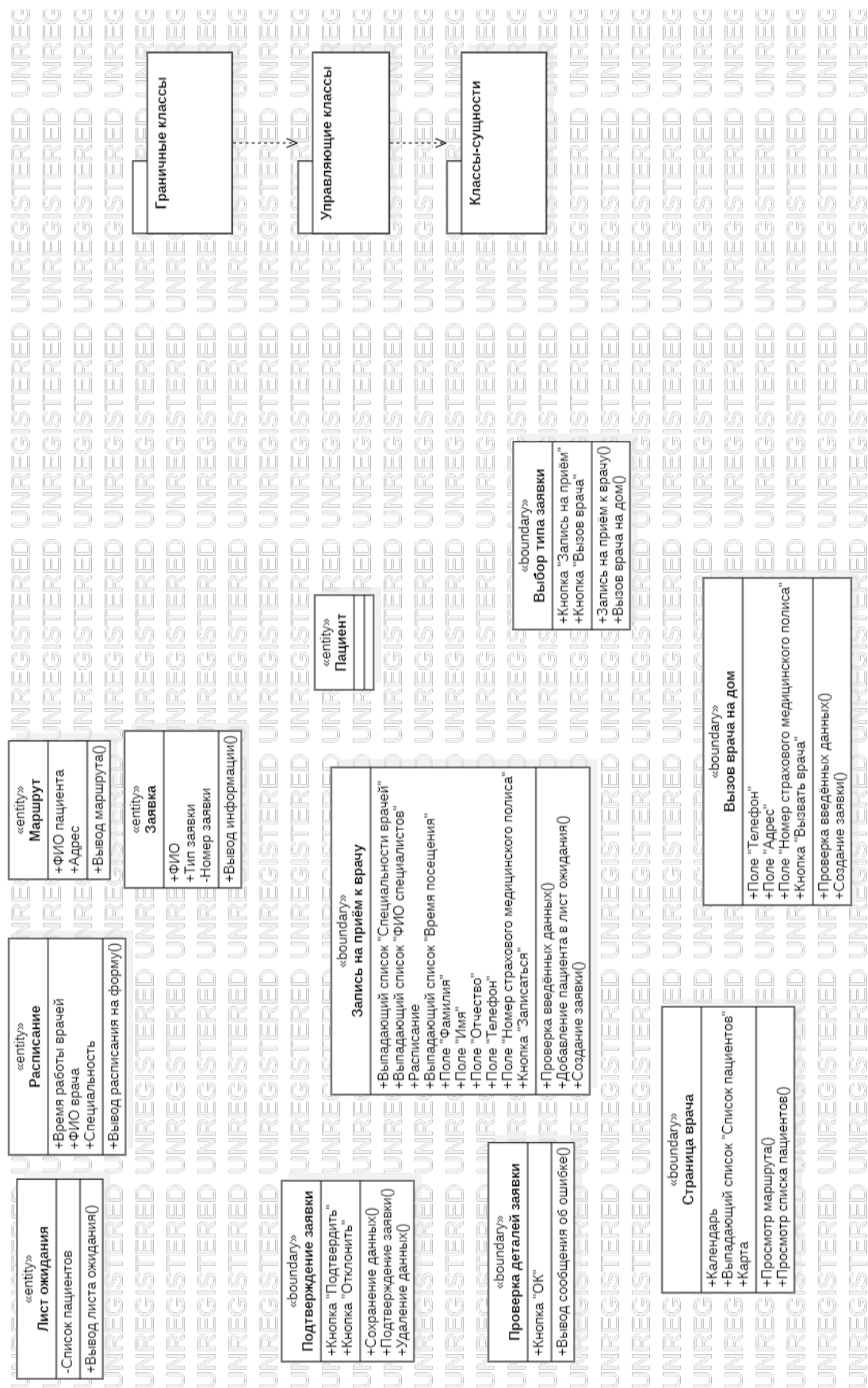


Рис. 3.26 Диаграмма классов ключевых абстракций Мобильного сервиса вызова врача на дом

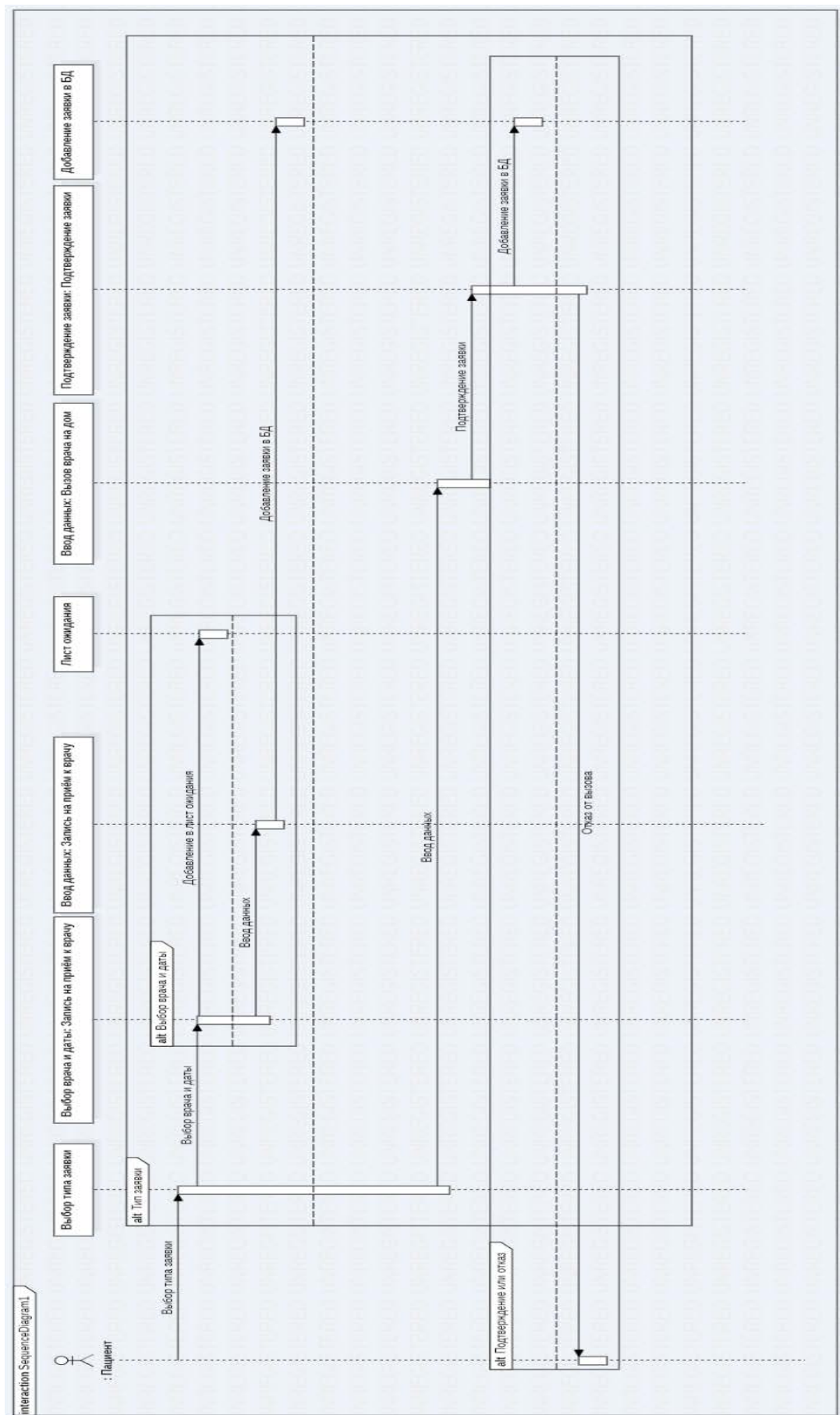


Рис. 3.27 Диаграмма последовательности прецедента Создание заявки
Мобильного сервиса вызова врача на дом

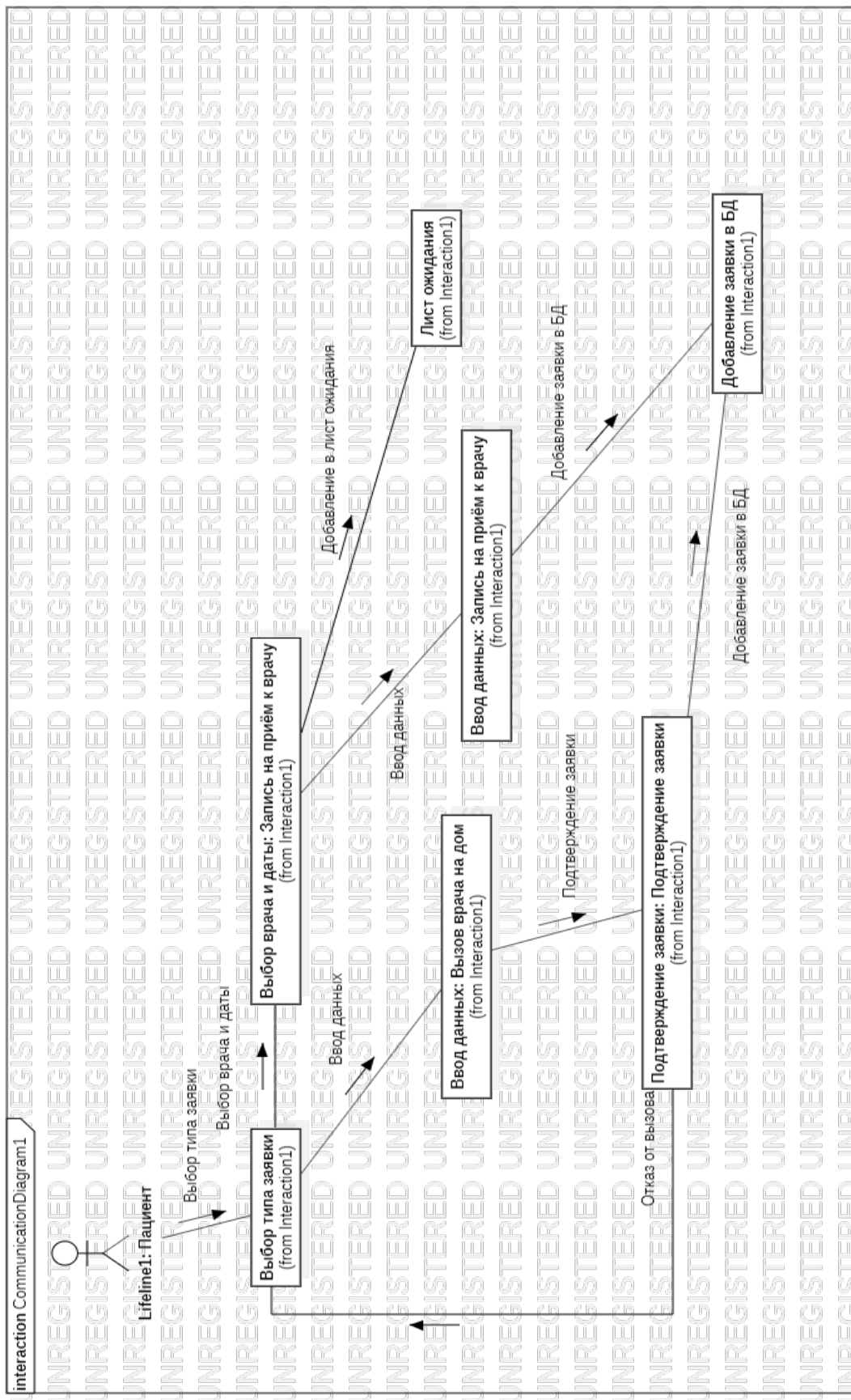


Рис. 3.28 Диаграмма кооперации Мобильного сервиса вызова врача на дом

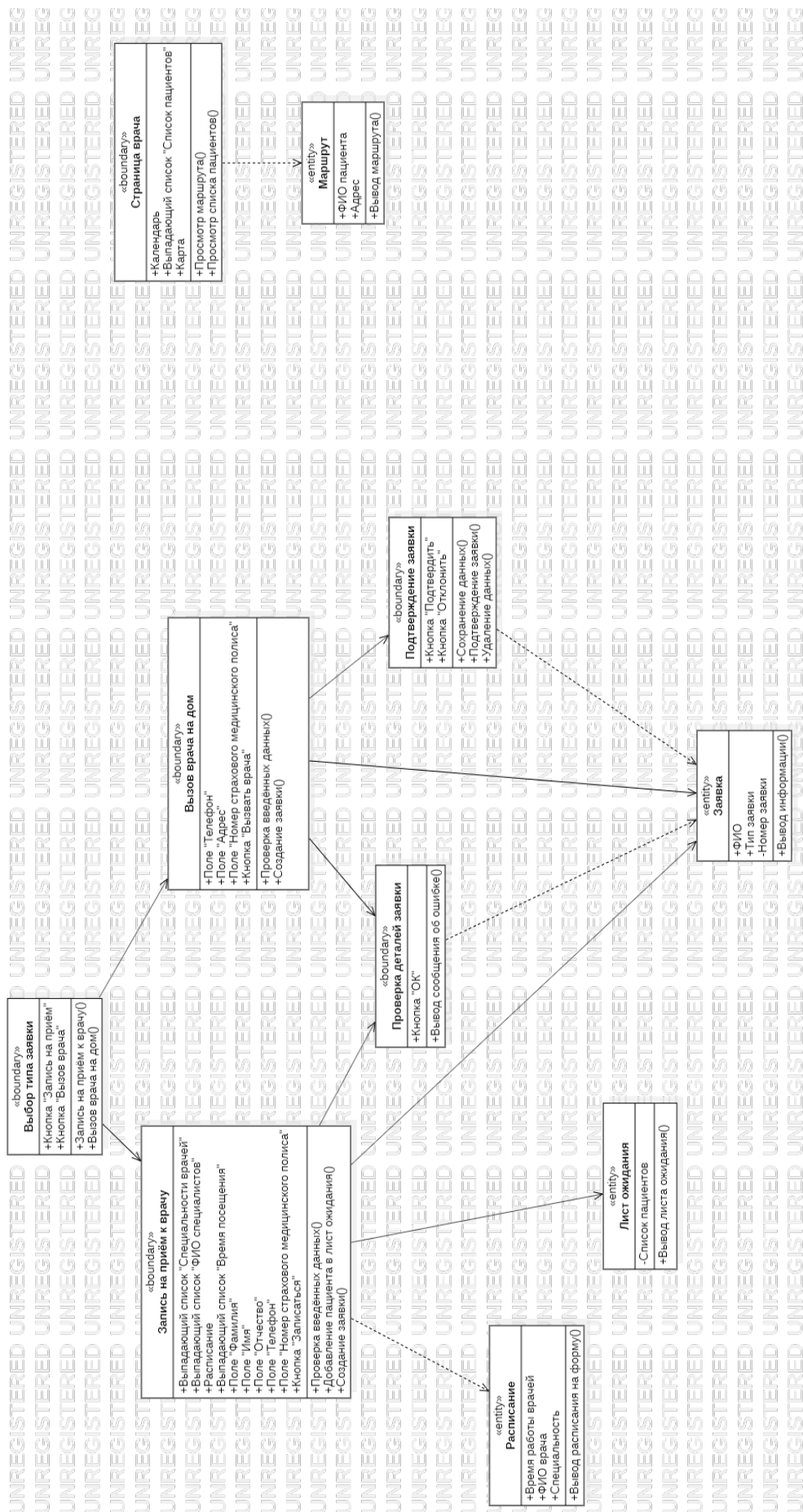


Рис. 3.29 Диаграмма классов аналитической модели Мобильного сервиса вызова врача на дом

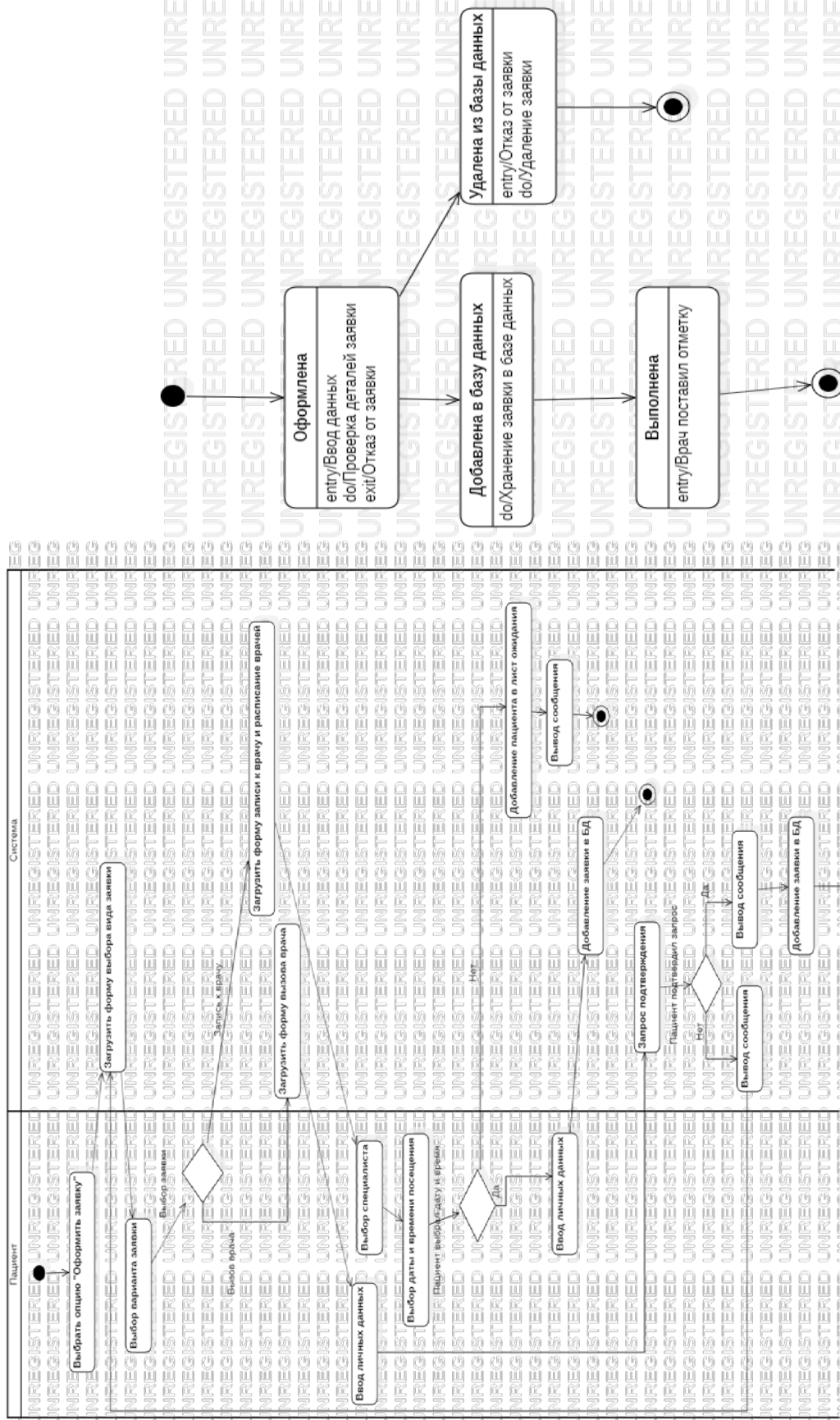


Рис. 3.30 Диаграммы деятельности и состояний класса Заявка Мобильного сервиса вызова врача на дом

БИБЛИОГРАФИЧЕСКИЙ СПИСОК ЛИТЕРАТУРЫ

1. Информатика: энциклопедический словарь для начинающих / ред. Д.А. Поспелов. – Москва: Педагогика-пресс, 1994. – 349 с. – Текст: непосредственный.
2. Волкова В.Н. Моделирование систем и процессов / В.Н. Волкова, В.Н. Козлова. – Москва: Юрайт, 2015. – 132 с. – Текст: непосредственный.
3. Марка Дэвид А. Методология структурного анализа и проектирования SADT / Дэвид А. Марка, Клемент МакГоуэн. – Москва: Весть-МетаТехнология, 1993. – 85 с. – Текст: непосредственный.
4. Калашян А.Н. Структурные модели бизнеса: DFD-технологии / А.Н. Калашян, Г.Н. Калянов. – Москва: Прикладные информационные технологии, 2009. – 256 с. – Текст: непосредственный.
5. Дубейковский В.И. Эффективное моделирование с AllFusion Process Modeler / В.И. Дубейковский. – Москва: Диалог-МИФИ, 2007. – 384 с. – Текст: непосредственный.
6. Бураков П.В. Введение в системы баз данных / П.В. Бураков, В.Ю. Петров – Санкт-Петербург: Лань, 2010. – 129 с. – Текст: непосредственный.
7. Арлоу Д. UML 2 и Унифицированный процесс. Практический объектно-ориентированный анализ и проектирование / Д. Арлоу, И. Нейштадт. – Изд. 2-е. – Санкт-Петербург: Символ Плюс, 2007. – 624 с. – Текст: непосредственный.
8. Буч Г. Объектно-ориентированный анализ и проектирование с примерами приложений / Г. Буч. – Изд. 3-е. – Москва: Издательский дом "Вильямс", 2008. – 720 с. – Текст: непосредственный.
9. Вигерс К. Разработка требований к программному обеспечению / К. Вигерс, Д. Битти. – Изд. 3-е, доп. – Москва: Издательство «Русская редакция», Санкт-Петербург: БХВ-Петербург, 2014. – 736 стр. – Текст: непосредственный.
10. Размышления о разработке программного обеспечения и информационных систем. Процесс работы над требованиями / М.И. Кузнецов. – Текст: электронный // Livejournal.com:[сайт]. – URL: <https://m-i-kuznetsov.livejournal.com/162578.html> (дата обращения: 23.11.2020).
11. Фаулер М. UML. Основы / М. Фаулер. – Изд. 3-е. – Санкт-Петербург: Символ-Плюс, 2004. – 192 с. – Текст: непосредственный.
12. Соммервилл И. Инженерия программного обеспечения / И. Соммервилл. – Изд. 6-е. – Москва: Издательский дом "Вильямс", 2002. – 624 с. – Текст: непосредственный.
13. Kruchten Ph. Architectural Blueprints - The “4+1” View Model of Software Architecture / Ph.Kruchten. – Текст: электронный // Paper published in IEEE Software – 1995. – № 12(6) – С. 42-50. – URL: <https://www.win.tue.nl/~wstomv/edu/2ip30/references/Kruchten-4+1-view.pdf> (дата обращения: 23.11.2020).

Приложение 1 Структура SRS шаблона на основе RUP

1. Введение
 - a. Цели
 - b. Определения, сокращения, термины
 - c. Ссылки
 - d. Обзор
2. Общее описание
 - a. Требования
 - b. Обзор
3. **я**
 - a. Функциональность
 - i. <Функциональное требование первое>
 - ii. ...
 - b. Удобство использования
 - i. <Требование по удобству использования первое>
 - ii. ...
 - c. Надежность
 - i. <Требование к надежности первое>
 - ii. ...
 - d. Производительность
 - i. <Требование к производительности первое>
 - ii. ...
 - e. Поддерживаемость
 - i. <Требование к поддерживаемости первое>
 - ii. ...
 - f. Проектные ограничения
 - i. <Проектное ограничение первое>
 - ii. ...
 - g. Требования по документированности и поддержке пользователей
 - h. Заимствованные компоненты
 - i. Интерфейсы
 - i. Пользовательские интерфейсы
 - ii. Аппаратные интерфейсы
 - iii. Программные интерфейсы
 - iv. Коммуникационные интерфейсы
4. Лицензионные соглашения
5. Необходимые замечания по законодательству, авторским правам и прочие
6. Применяемые стандарты
7. Сопроводительная информация

Приложение 2 Структура SRS на основе стандарта ISO/IEC/IEEE 29148:2011

1. Введение
 - a. Цели
 - b. Соглашения о терминах
 - c. Предполагаемая аудитория и последовательность восприятия
 - d. Масштаб проекта
 - e. Ссылки на источники
2. Общее описание
 - a. Видение продукта
 - b. Функциональность продукта
 - c. Классы и характеристики пользователей
 - d. Среда функционирования продукта (операционная среда)
 - e. Рамки, ограничения, правила и стандарты
 - f. Документация для пользователей
 - g. Допущения и зависимости
3. Функциональность системы
 - a. Функциональный блок X (таких блоков может быть несколько)
 - i. Описание и приоритет
 - ii. Причинно-следственные связи, алгоритмы (движение процессов, workflows)
 - iii. Функциональные требования
4. Требования к внешним интерфейсам
 - a. Интерфейсы пользователя (UX)
 - b. Программные интерфейсы
 - c. Интерфейсы оборудования
 - d. Интерфейсы связи и коммуникации
5. Нефункциональные требования
 - a. Требования к производительности
 - b. Требования к сохранности (данных)
 - c. Требования к качеству программного обеспечения
 - d. Требования к безопасности системы
 - e. Требования на интеллектуальную собственность
6. Прочее
 - Приложение А: Глоссарий
 - Приложение Б: Модели процессов и предметной области и другие диаграммы
 - Приложение В: Список ключевых задач

Приложение 3. Процесс разработки АИС на основе спецификации [10]

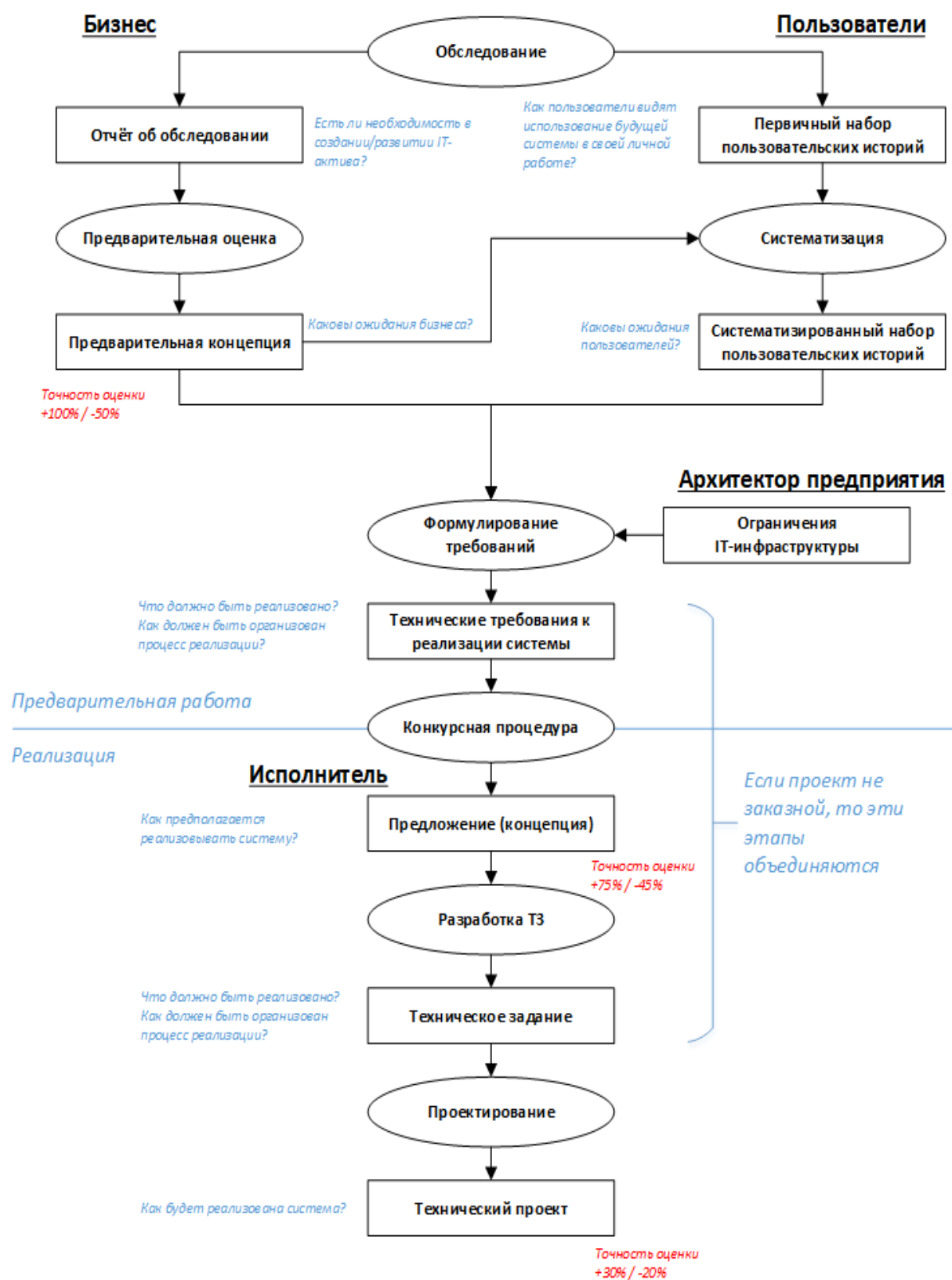




Диаграмма классов: Интерфейсы



Диаграмма классов: Параметризованный класс

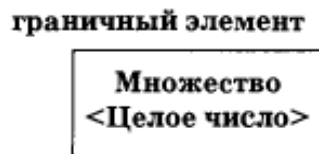
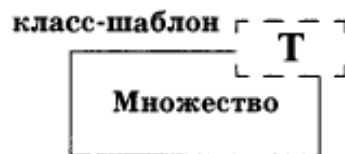


Диаграмма деятельности



ФЗ № 436-ФЗ

Издание не подлежит маркировке
в соответствии с п. 1 ч. 4 ст. 11

Учебное издание

Каратун Сергей Михайлович
Лозикова Инна Олеговна

ПРОЕКТИРОВАНИЕ АВТОМАТИЗИРОВАННЫХ ИНФОРМАЦИОННЫХ СИСТЕМ

В авторской редакции

Подписано в печать 18.06.2021. Формат 60х90 1/16. Печ. л. 7,6.
Тираж 500 экз. Заказ № 2072.

Библиотечно-издательский комплекс
федерального государственного бюджетного образовательного
учреждения высшего образования
«Тюменский индустриальный университет».
625000, Тюмень, ул. Володарского, 38.

Типография библиотечно-издательского комплекса.
625039, Тюмень, ул. Киевская, 52.