Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатика, искусственный интеллект и системы управления

КАФЕДРА Системы обработки информации и управления

Методические указания к домашнему заданию по курсу «Технологии разработки программного обеспечения»

Домашнее задание №3 «Модели проектирования и реализации программных систем»

Виноградова М.В., Волков А.С., Королев С.В.,
Умряев Д.Т., Солохов И.Р., Жизневский П.И.

Под редакцией к.т.н. доц. Виноградовой М.В.

Москва, 2022 г.

ОГЛАВЛЕНИЕ

1. ЗАДАНИЕ

Домашнее задание №3 «Модели проектирования и реализации программных систем» по курсу «Технологии разработки программного обеспечения».

1.1. Цель работы

- Изучить принципы построения моделей проектирования и реализации программных систем на основе подхода RUP;
- Получить практические навыки построения моделей проектирования и реализации в среде моделирования.

1.2. Порядок и время проведения работы

Работа выполняется самостоятельно в часы внеаудиторных занятий. По итогам составляется и защищается отчет в бумажном виде, а также проводится демонстрация работающей программы.

1.3. Средства выполнения

- Среда моделирования: Sparx Enterprise Architect, Star UML и т.д.

1.4. Исходные данные

1. Диаграмма пакетов из ЛР-5 (распределение классов анализа по пакетам).
2. Диаграмма классов проектирования из ДЗ-2.
3. Диаграммы последовательностей из ДЗ-2.

1.5. Задание

1. Определить набор подсистем и распределить по ним классы проектирования (ДЗ-2).
2. Определить зависимости подсистем. Построить диаграмму уровней подсистем.
3. Построить модель трассировки пакетов анализа (из ЛР-5) в подсистемы.
4. Построить модель трассировки классов анализа (ЛР-5) в классы проектирования (ДЗ-2).
5. Построить диаграмму развертывания (узлы, каналы связи и подсистемы).

6. (дополнительно). Определить интерфейсы подсистем. Построить диаграмму последовательностей (из ДЗ-2) в терминах подсистем и их интерфейсов.
7. Определить набор компонентов. Построить модель трассировки подсистем в компоненты.
8. (дополнительно) Построить модель трассировки подсистем в компоненты с сохранением интерфейсов.
9. Построить модель трассировки классов проектирования (ДЗ-2) в исходные файлы.
10. Построить модель зависимостей компонентов от исходных файлов.
11. (дополнительно) Построить диаграмму последовательностей (из ДЗ-2) в терминах компонентов и их интерфейсов.

1.6. К защите

В отчет:

- Титульный лист;
- Цель работы;
- Задание;
- Диаграмма пакетов анализа (из ЛР-5);
- Диаграмма классов и диаграммы последовательностей (из ДЗ-2);
- Диаграмма распределения классов проектирования (ДЗ-2) по подсистемам.
- Диаграмма уровней подсистем.
- Модель трассировки пакетов анализа (из ЛР-5) в подсистемы.
- Модель трассировки классов анализа (ЛР-5) в классы проектирования (ДЗ-2).
- Диаграмма развертывания (узлы, каналы связи и подсистемы).
- (дополнительно) Диаграмма распределения классов проектирования по подсистемам с указанием интерфейсов.
- (дополнительно) Диаграммы последовательностей (из ДЗ-2) в терминах подсистем и их интерфейсов.
- Модель трассировки подсистем в компоненты.
- Модель трассировки классов проектирования (ДЗ-2) в исходные файлы.
- Модель зависимостей компонентов от исходных файлов.
- Список литературы.

2. ТЕОРЕТИКО-ПРАКТИЧЕСКИЙ МАТЕРИАЛ

2.1. Унифицированный процесс разработки ПО RUP

Унифицированный процесс разработки ПО RUP (Rational Unified Process, англ.) — это методология создания программных систем, являющаяся результатом объединения различных подходов к построению программ. В RUP используется визуальное проектирование с применением языка UML и автоматическая генерация кода на основе построенных моделей. RUP определяет методику и процессы разработки программ, а также средства для ее автоматизации. RUP позволяет создавать крупные и сложные системы большой командой разработчиков с различным уровнем квалификации. Он обеспечивает автоматизацию процессов разработки ПО, а также сохранность 6 моделей различного уровня и их согласованность при внесении изменений [1].

RUP вобрал в себя преимущества спиральной и итерационной методологий разработки ПО. Согласно спиральной методологии, требования к программе формируются и уточняются постепенно, в процессе создания программы. В соответствии с инкрементной методологией происходит последовательное приращение функциональности.

RUP имеет следующие особенности:

- управляется прецедентами, т. е. на всех этапах разработки в центре внимания находится прецедент. Прецедент — это прикладная задача, которая подлежит автоматизации при написании программы, например задача добавления сведений о персоне или задача поиска по каталогу;
- ориентирован на архитектуру программы — это означает, что на начальном этапе разработки определяется архитектура будущей системы. На последующих этапах разработки архитектура влияет на принятие проектных решений;
- является итерационно-инкрементным, то есть при разработке происходит декомпозиция всей системы на мини-проекты, называемые итерациями. В результате выполнения очередной итерации появляется инкремент — функциональное приращение.

2.2. Перечень рабочих процессов по RUP

Унифицированный процесс состоит из четырех этапов разработки:

- **Сбор требований** — на этом этапе разработчики должны убедиться, что программный проект осуществим. Для этого они выявляют и анализируют следующие характеристики:
    - область применения будущей системы (функциональное назначение и взаимодействие с внешней средой);
    - основные требования к архитектуре системы (на базе главных функций программы);
    - основные (критические) риски;
    - примерную стоимость и план проекта;
    - начальный макет системы.
- **Проектирование** – создание архитектурного базиса системы и подготовка к выполнению третьего этапа — построению. На данном этапе решают следующие задачи:
    - создают базовый уровень архитектуры системы;
    - определяют существенные риски, методы их отслеживания и устранения или снижения;
    - формируют требования к качеству ПО и процесса его разработки, например стандарты;
    - составляют финансовый план проекта;
    - проводят анализ большинства прецедентов (до 80 %);
    - готовят план итераций для выполнения следующего этапа.
- **Построение** – создание работоспособного программного продукта. В ходе построения решают следующие вопросы:
    - управление ресурсами;
    - оптимизация процессов разработки;
    - отслеживание существенных и критических рисков;
    - оценка качества создаваемого ПО.
- **Внедрение** – готовое и протестированное ПО устанавливают для эксплуатации конечными пользователями и готовят пользователей для работы с ним. На данном этапе выполняют следующие задачи:

- завершение реализации ПО;
- формирование рекомендаций по установке и эксплуатации ПО;
- подготовка программно-аппаратного обеспечения для работы с ПО (конечными пользователями);
- применение ПО в среде конечного пользователя;
- составление документации и руководств (в соответствии с требованиями заказчика);
- проведение бета-тестов, выявление и устранение ошибок ПО;
- создание версии ПО для приемосдаточных испытаний.

Каждый этап выполняется как серия итераций, каждая из которых обеспечивает приращение функциональных возможностей системы.

Итерация — это полный цикл разработки фрагмента ПО, в результате которого формируется промежуточная версия, реализующая некоторый функционал. В ходе разработки происходит постепенное усложнение создаваемого ПО, наращивание его функциональных возможностей.

Каждая итерация состоит из последовательного выполнения пяти рабочих процессов:

1. **сбор требований — определение исходных требований к создаваемой системе (или** ее фрагментам);
2. **анализ (требований)** — преобразование требований в классы и объекты (построение моделей анализа);
3. **проектирование** — создание статического и динамического представлений системы для выполнения исходных требований (построение проектных моделей);
4. **реализация** — процесс генерации программного кода (включая тестирование и отладку модулей);
5. **тестирование** — проверка работоспособности системы в целом.

При каждой итерации осуществляют все рабочие процессы, но удельный вес конкретного рабочего процесса зависит от этапа разработки:

- на первом этапе основное внимание уделяют сбору требований;
- на этапе проектирования — анализу и проектированию;
- на этапе построения — реализации;

- на этапе тестирования — тестированию.

Примерное распределение работ по рабочим процессам в зависимости от этапа разработки приведено на Рисунок 1.

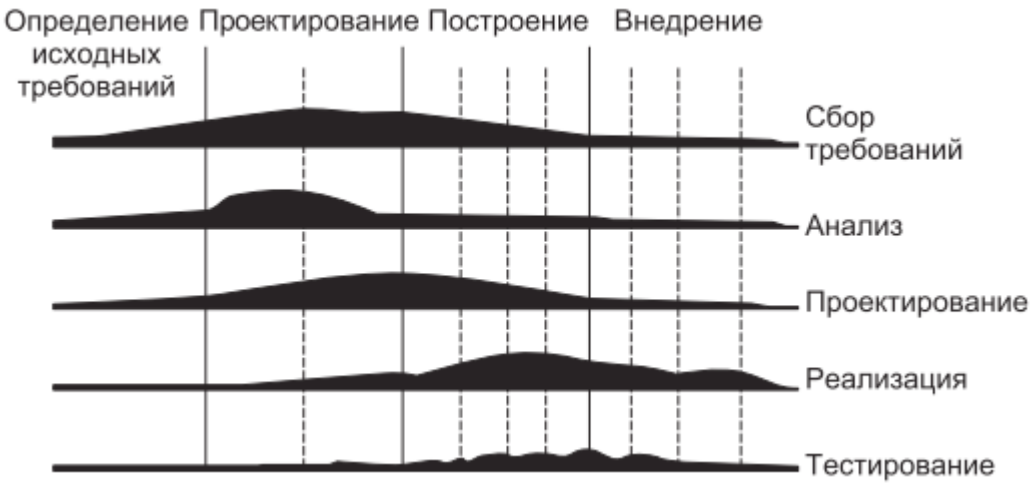

Рисунок 1 – Рабочие процессы на этапах RUP

Существуют множество различных сред для построения модели требований проекта, в настоящей работе мы будем использовать Enterprise Architect.

2.3. Рабочий процесс "Проектирование"

Во время проектирования формируются модели, представляющие разрабатываемую систему с разных точек зрения. Создаются модели, описывающие структуры данных и алгоритмы программы, компоненты системы и связи между ними, сетевую конфигурацию.

В результате проектирования создаются следующие артефакты:

- модель проектирования - содержит систему проектирования, содержащую множество подсистем проектирования, классов проектирования, проектов кооперации и интерфейсов;
- классы проектирования – описания классов в терминах языка реализации;
- проекты кооперации – описания проектных представлений прецедентов, содержащих диаграммы классов, диаграммы взаимодействия и проекты потоков событий для выполнения прецедента;
- подсистемы проектирования и сервисные подсистемы – описывают проекты будущих компонентов системы;
- интерфейсы – содержат перечень доступных извне функций подсистем;

- описание архитектуры – содержит наиболее важные проектные решения, определяет компоненты и технологии, используемые при разработке и функционировании системы;
- модель развертывания – содержит описание сетевой конфигурации.

Процесс проектирования состоит из выполнения следующих действий:

- проектирование архитектуры;
- проектирование прецедентов;
- проектирование классов;
- проектирование подсистем.

Рассмотрим эти действия более подробно.

2.3.1. Проектирование архитектуры

2.3.1.1. Определение узлов и сетевых конфигураций

Проектирование архитектуры начинают с определения физических компонентов и связей между ними. На этом шаге формируют следующие описания системы:

- перечень узлов и их конфигураций (ЦП, ОП и т.д.);
- конфигурация сети с указанием линий связи между узлами, используемых протоколов, характеристики передачи, в том числе скорость и качество передачи;
- дополнительные требования, включая требования к безопасности, достоверности, резервному копированию и т.д.

По итогам определения узлов и сетевых конфигураций составляется диаграмма размещения. Рисунок демонстрирует пример диаграммы размещения для АСУ «Веб-портал поиска пропавших животных».

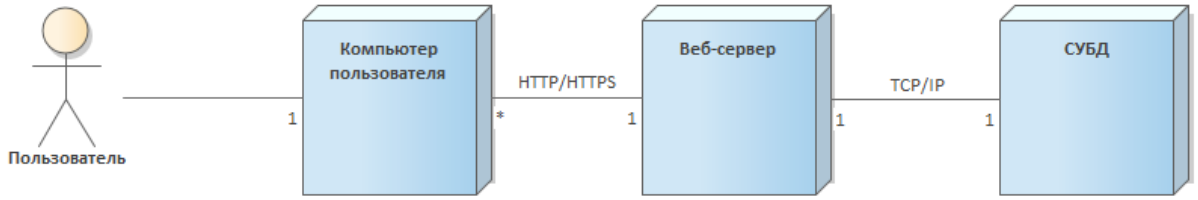

Рисунок 2 – Диаграмма размещения для модели АСУ «Веб-портал поиска пропавших животных»

2.3.1.2. Определение подсистем и их интерфейсов

Следующим шагом проектирования архитектуры является определение подсистем и их интерфейсов.

В приложении выделяют четыре уровня подсистем: специфический и общий прикладной уровни, средний уровень и уровень системного ПО.

Подсистемы прикладного уровня определяются на основе пакетов анализа. Изначально пакеты анализа трассируются в подсистемы проектирования прикладных уровней, сервисные пакеты трассируются в сервисные подсистемы. Далее проводят уточнение и декомпозицию полученных подсистем по следующим правилам:

- часть пакета анализа, совместно используемую несколькими подсистемами, выделяют в отдельную подсистему;
- часть пакета анализа для многократного применения выделяют в отдельную подсистему, в том числе подсистему среднего уровня или уровня системного ПО;
- унаследованные подсистемы выделяют в отдельные подсистемы;
- выполняют декомпозицию подсистем для возможности разделения работ;
- осуществляют декомпозицию подсистем в целях разнесения их по узлам так, чтобы каждая подсистема была на отдельном узле.

В результате трассировки пакетов анализа в подсистемы модели АСУ «Веб-портал поиска пропавших животных» была сформирована диаграмма трассировки (см. Рисунок 3).

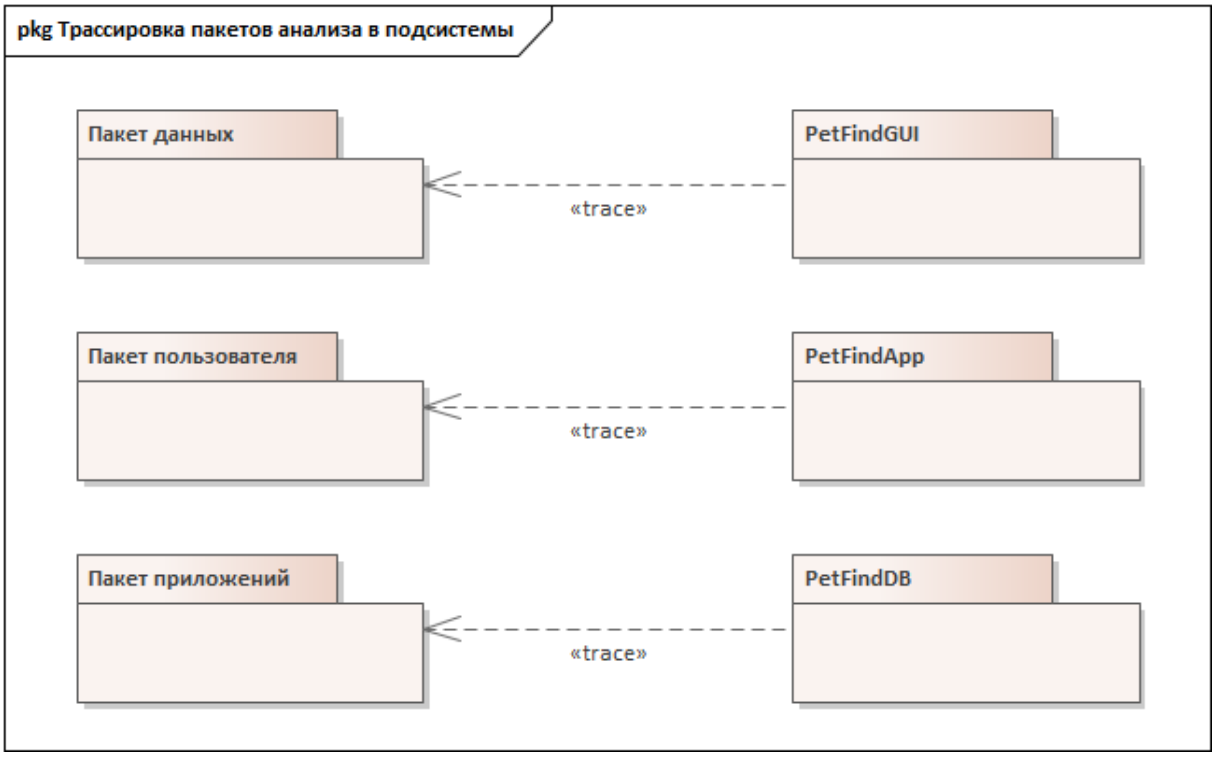

Рисунок 3 – Диаграмма трассировки пакетов в подсистемы для модели АСУ «Веб-портал поиска пропавших животных»

К подсистемам среднего уровня и уровня системного ПО относят следующие компоненты:

- ОС;
- СУБД;
- средства коммуникации;
- средства распределенной обработки;
- средства выполнения транзакций;
- средства разработки интерфейсов.

Поскольку часть из перечисленных компонентов имеет смысл покупать, а не разрабатывать самостоятельно, то при проектировании подсистем среднего уровня и уровня системного ПО осуществляют:

- подбор и интеграцию покупаемых и создаваемых компонентов;
- оценку эффективности и пригодности выбранных компонентов.

Критерием эффективности выбора является уменьшение зависимости от продуктов сторонних производителей и тем самым снижение рисков несовместимости или отсутствия поддержки.

Зависимости между подсистемами проектирования, как и зависимости между пакетами анализа соответствуют вызовам при работе системы. Подсистемы верхних уровней зависят от подсистем нижних уровней, но не наоборот.

Для уменьшения зависимости подсистем проводят замену зависимости подсистем на зависимость их интерфейсов. Это позволит скрыть реализацию подсистемы от зависящих от нее компонентов.

В качестве примера Рисунок 2 демонстрирует диаграмму уровней подсистем для модели АСУ «Веб-портал поиска пропавших животных».

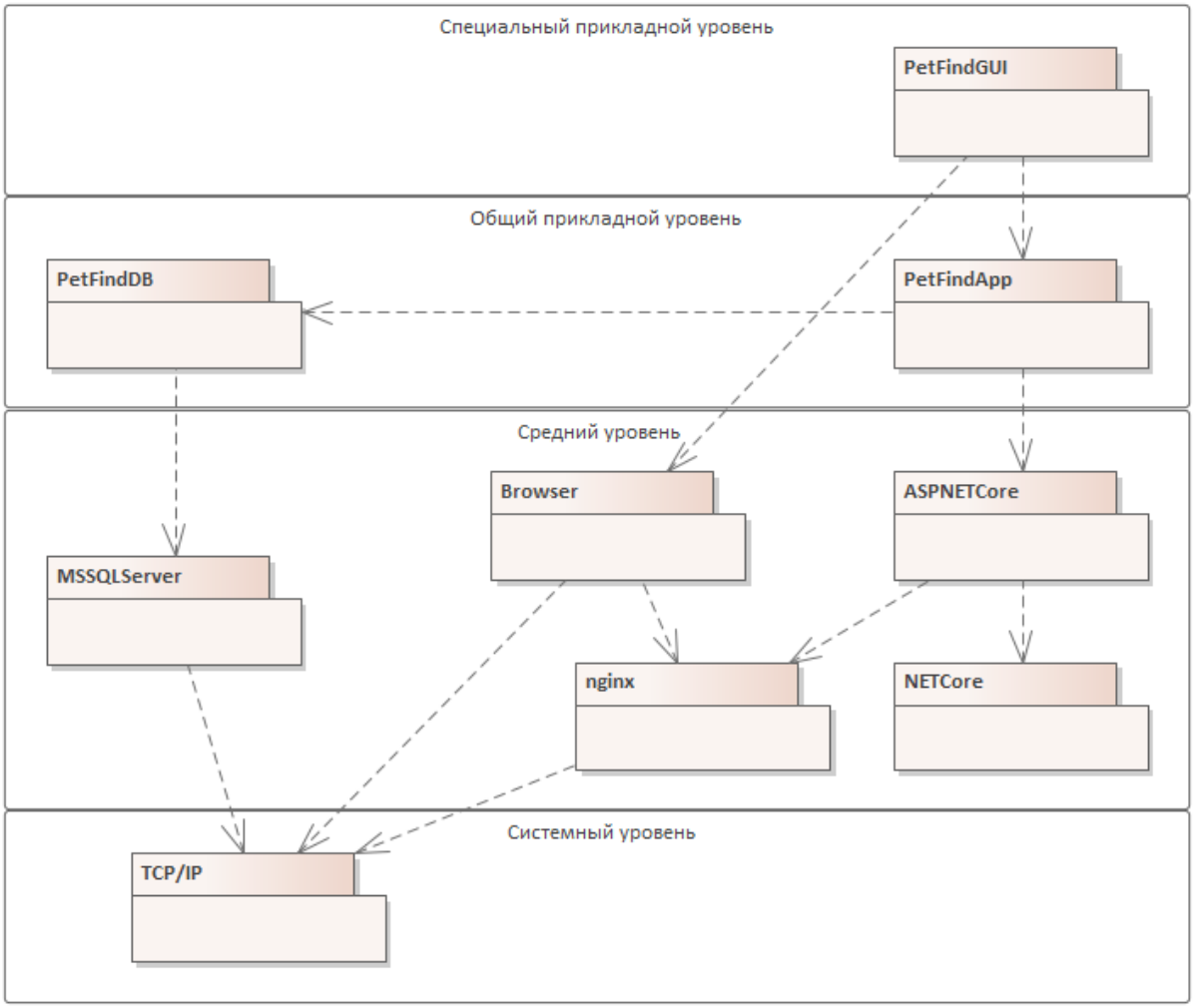

Рисунок 2 – Диаграмма уровней подсистем для АСУ «Веб-портал поиска пропавших животных»

Далее указываем размещение подсисем по узлам.

В качестве примера Рисунок 5 демонстрирует диаграмму развертывания для модели АСУ «Веб-портал поиска пропавших животных».

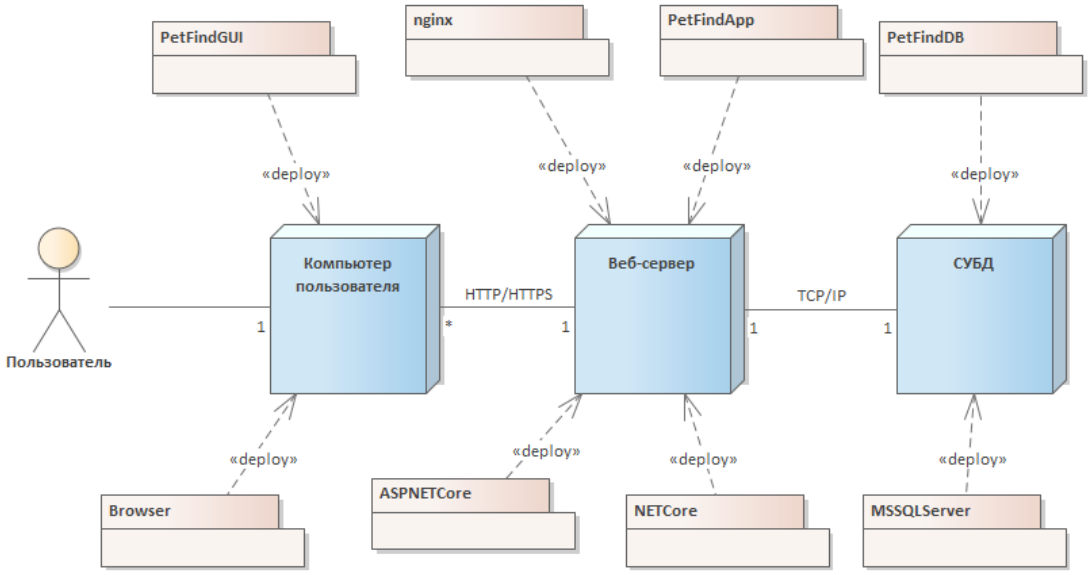

Рисунок 5 – Диаграмма развертывания для модели АСУ «Веб-портал поиска пропавших животных»

2.3.1.3. Определение интерфейсов подсистем

Для каждой подсистемы следует определить ее интерфейс. Интерфейс подсистемы — это перечень операций, выполняемых данной подсистемой и доступных извне. В интерфейсе указывают полные сигнатуры операций.

Интерфейс подсистемы формируется на основе выявленных зависимостей подсистем и определенных ранее классов в пакете анализа. Если подсистема трассирована из пакета анализа, содержащего класс, к которому имеется обращение извне, то интерфейс подсистемы должен обеспечивать внешнюю ответственность этого класса.

Определение интерфейсов начинают с подсистем верхнего уровня. Для подсистем среднего и нижнего уровня интерфейсы могут быть известны заранее, если это существующее ПО.

Интерфейсы влияют на зависимости подсистем, поскольку скрывают их реализацию.

2.3.1.4. Определение архитектурно значимых классов проектирования

Кроме подсистем в описание архитектуры включают описания некоторых классов, имеющих значение для работы всей системы или нескольких ее компонентов. Такие классы называют архитектурно-значимыми.

К архитектурно-значимым классам проектирования относят классы, определенные на основе классов анализа и активные классы. Классы проектирования, определенные на основе классов анализа — это, прежде всего, классы сущностей. Активные классы – это классы, объекты которых имеют свою нить управления или свой процесс.

Привязка активных объектов к узлам проводится с учетом их возможности и конфигурации.

2.3.1.5. Определение обобщенных механизмов проектирования

Проектирование архитектуры завершается определением обобщенных механизмов проектирования. Обобщенные механизмы проектирования – это технологии и средства, которые обеспечивают следующие возможности:

- длительное хранение данных;
- распределенная обработка;
- средства безопасности;
- контроль и исправление ошибок;
- управление транзакциями.

При проектировании архитектуры необходимо сформулировать конкретные требования к системе, а также выбрать используемые технологии и средства для их обеспечения. Например, для длительного хранения можно использовать реляционные или объектные БД или файлы; для распределенной обработки - технологии java.rmi, corba или com.

Поскольку каждая технология имеет свои преимущества и недостатки, то проводится сравнительный анализ с учетом возможного развития системы. Обычно поставляемое и приобретаемое ПО относится к среднему уровню и уровню системного СПО, а создаваемое ПО к прикладному уровню.

Также к обобщенным механизмам проектирования относят применение «обобщенных кооперацией» – паттернов, типовых проектных решений на основе абстрактных кооперацией.

2.3.2. Проектирование прецедентов

При проектировании прецедента определяют классы и подсистемы, необходимые для реализации этого прецедента, а также описывают их взаимодействие.

2.3.2.1. Определение участвующих классов

Классы анализа трассируются в классы проектирования. Если есть специальные требования к кооперации, то создается новый класс проектирования для их реализации. Если требуемых классов проектирования нет, то их надо определить.

Классы анализа трассируются в классы проектирования с учетом выбранных технологий реализации по следующим правилам:

- граничные классы трассируются в классы страниц, форм и их элементов (возможно наследование от стандартных класссов GUI);
- управляющие классы трассируются в классы обработки информации (возможно добавление стандартных классов для встраивание в фреймворк или паттерны);
- классы сущностей трассируются в таблицы и классы работы с базой данных.

Граничные классы в зависимости от выбранной технологии разработки трассируют в следующие классы проектирования:

- классы для создания интерфейса взаимодействия с внешним устройством или системой;
- классы, соответствующие элементам пользовательского интерфейса, например, экранной форме, полю ввода или элементу управления с указанием их стереотипов.

Если применялась среда визуальной разработки, то трассировка проходит автоматически, на основе созданного ранее прототипа пользовательского интерфейса.

Классы сущности в зависимости от выбранной СУБД и технологии работы с БД трассируют в реляционные таблицы или объектные и объектно-реляционные сущности. Возможно применение объектно-реляционного отображения. Часто такая трассировка выполняется автоматически.

Управляющие классы трассируют в классы проектирования с учетом следующих факторов:

- фактор размещения – по одному активному классу на каждый узел;
- производительность – совмещать управляющий класс с граничным классом или классом сущности с целью увеличения скорости работы;
- транзакции – использование существующих технологий управления транзакциями в качестве дополнительных управляющих классов.

2.3.2.2. Определение участвующих классов

Для каждого прецедента составляют диаграмму классов, на которую помещают классы проектирования, учструющие в этом прецеденте и актера - инициатора прецедента.

Связи (направленные ассоциации) показывают порядок взаимодействия классов и передачу управления между ними.

На рис. 6 пример - фрагмент диаграммы классов проектирования.

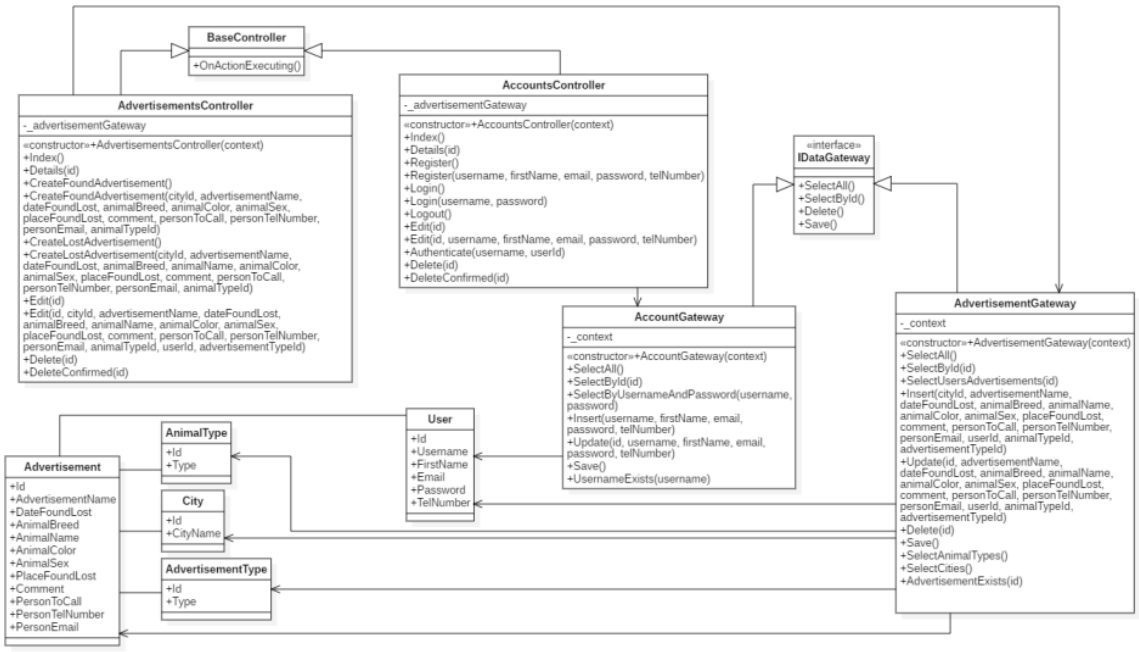

Рисунок 6 – Диаграмма классов для иллюстрации работы паттернов бизнес-логики

2.3.2.3. Описание взаимодействия объектов проектирования

Для описания алгоритма выполнения прецедента составляются диаграммы последовательностей, по одной на каждый поток событий прецедента (возможную последовательность действий). Диаграммы последовательностей содержат объекты классов, участвующих в прецеденте, и поток событий между ними.

На отдельные диаграммы выносят общие последовательности действий или действия для повторного использования.

Правила построения диаграмм последовательностей при проектировании прецедента:

- начальное событие всегда от актера – инициатора прецедента;
- для каждого класса, участвующего в прецеденте, на диаграмму последовательностей помещают его объект или несколько объектов;
- направления сообщений между объектами соответствуют связям между их классами на диаграмме классов и означают передачу управления в ходе выполнения прецедента (вызов методов класса);
- поток управления не может прерываться, в ответ на каждое входное сообщение объект отправляет одно или более сообщений другим объектам или возвращает результат его обработки отправителю;
- при необходимости помещают ссылки на обобщенные диаграммы взаимодействия.

На диаграммах указывают метки для альтернативных вариантов алгоритма, например, тайм-аут соединения, неверный ввод, сообщение об ошибке от внешнего компонента.

На рис. 7-9 примеры диаграмм последовательностей в терминах классов.

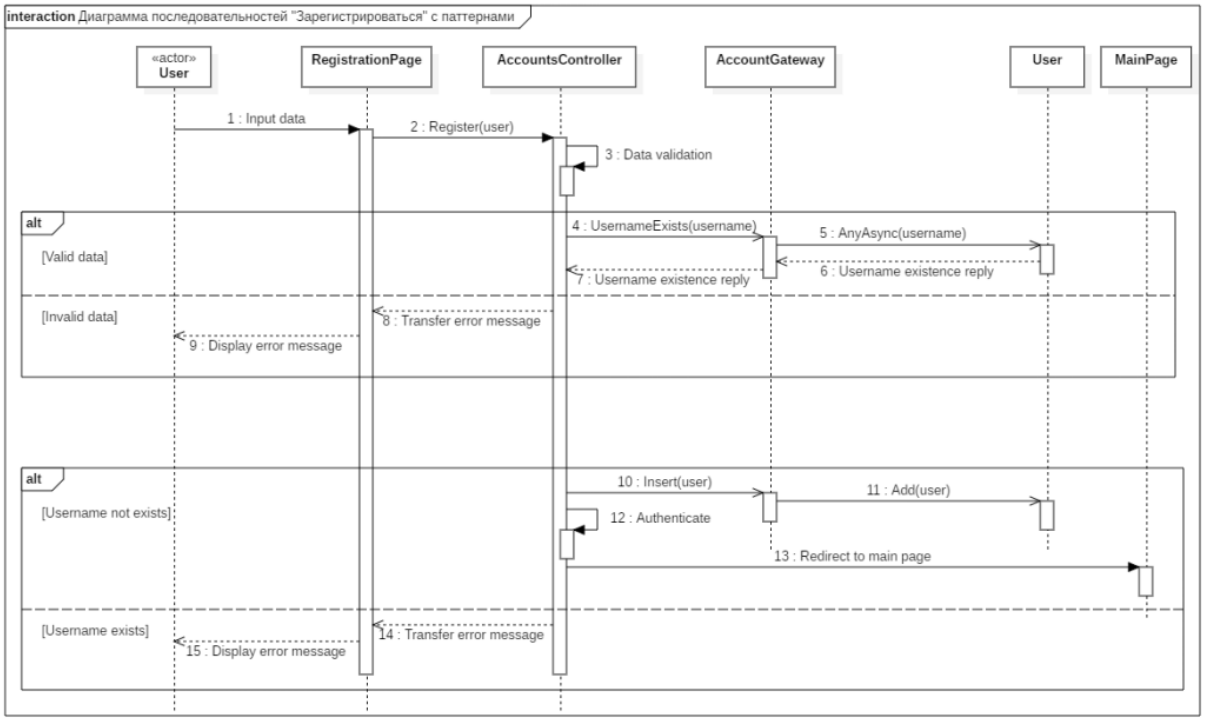

Рисунок 7 – Диаграмма последовательностей «Зарегистрироваться» для иллюстрации работы паттернов бизнес-логики и БД

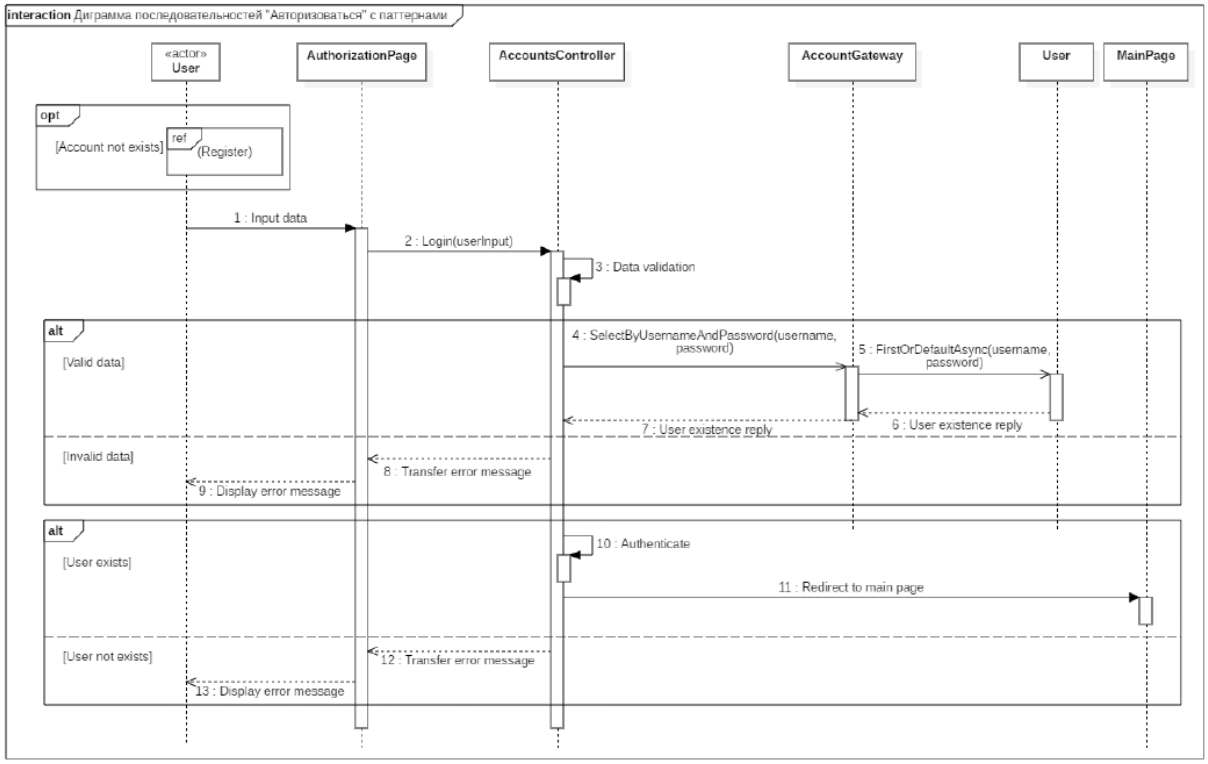

Рисунок 8 – Диаграмма последовательностей «Авторизоваться» для иллюстрации работы паттернов бизнес-логики и БД

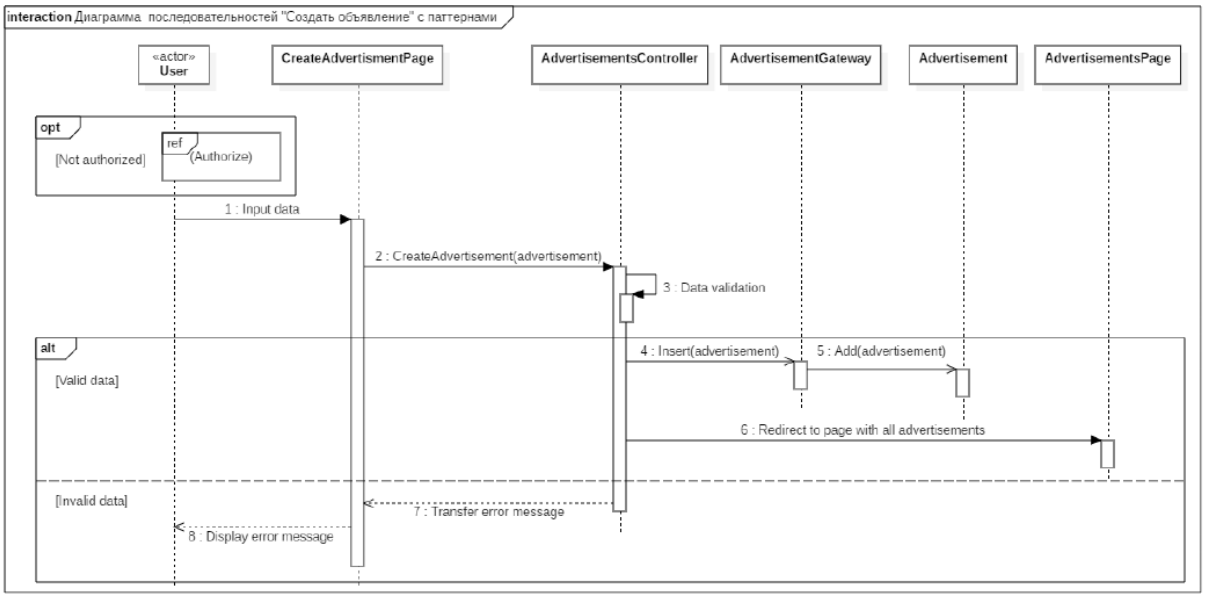

Рисунок 9 – Диаграмма последовательностей «Создать объявление» для иллюстрации работы паттернов бизнес-логики и БД

2.3.2.4. Определение участвующих подсистем и интерфейсов

Кроме классов, участвующих в прецеденте, необходимо определить подсистемы, участвующие в нем. Это с одной стороны, позволяет выполнять проектирование «сверху-вниз», с другой стороны – при необходимости заменить подсистему на аналог.

Подсистемы, участвующие в прецеденте, это подсистемы, которые:

- были транслированы из тех пакетов анализа, в которых содержатся классы анализа, участвующие в данном прецеденте;
- содержат классы проектирования, участвующие в данном прецеденте;
- содержат классы проектирования, полученные для реализации специальных требований к данному прецеденту.

Составляется диаграмма классов, на которую помещают подсистемы, участвующие в прецеденте, их интерфейсы и связи. Связи между подсистемами показывают передачу сообщений между ними в процессе выполнения прецедента. Интерфейсы подсистем содержат названия операций, которые при этом вызываются.

2.3.2.5. Описание взаимодействия подсистем

Кроме диаграммы классов составляется одна или более диаграмм последовательностей, которые показывают выполнение прецедента на уровне подсистем.

Диаграммы последовательностей содержат актеров, подсистемы и сообщения. Линии жизни указываются для подсистем, для каждой подсистемы не менее одной линии жизни. Сообщения отображаются в терминах операций интерфейсов, учитывая, что у подсистемы может быть больше одного интерфейса.

На рис. 10 пример диаграммы последоательностей в терминах подсистем.

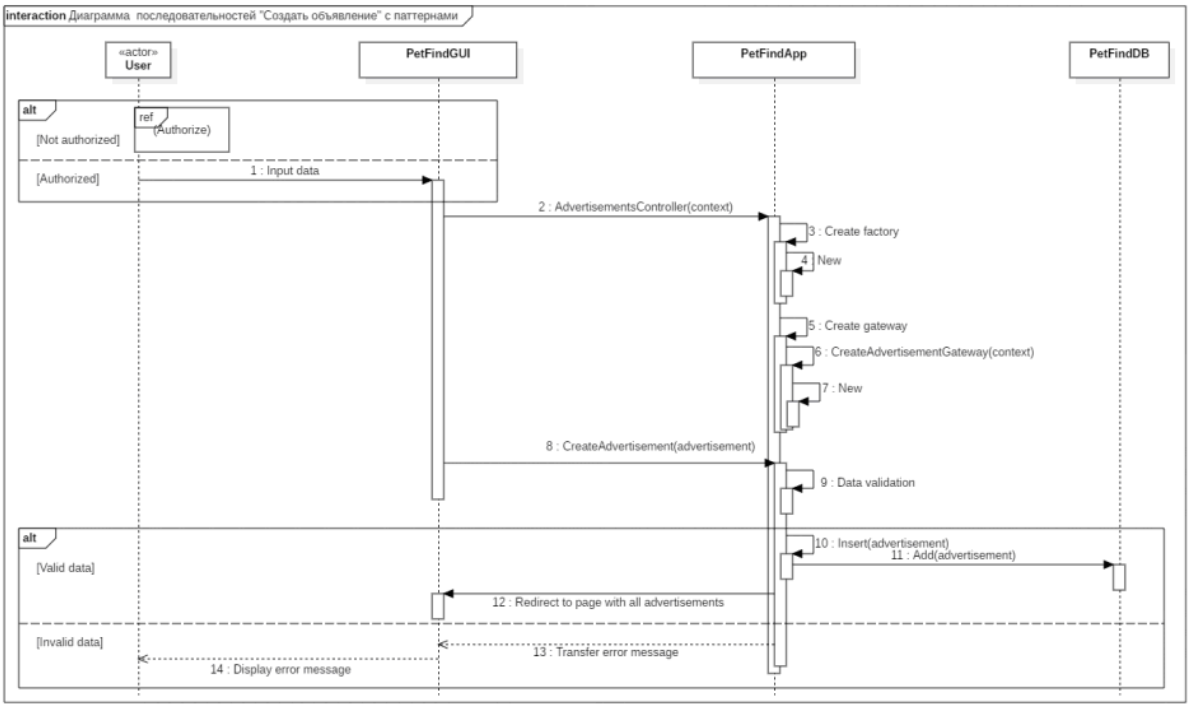

Рисунок 10 – Диаграмма последовательностей «Создать объявление» в терминах подсистем

2.3.2.6. Определение требований к кооперации

Завершается проектирование прецедента определением специальных и нефункциональных требований к нему. Здесь указывают требования, которые необходимо будет реализовать для данного прецедента.

2.3.3. Проектирование класса

Следующим шагом, выполняемым при проектировании, является проектирование класса. Составляется подробное описание класса в терминах выбранного языка программирования.

В результате проектирования класса будут определены следующие его параметры:

- операции (внешний интерфейс);
- атрибуты;
- отношения;
- методы (внутренняя реализация);
- состояния;
- зависимость от обобщенных механизмов проектирования;
- требования к реализации;
- правила реализации всех его интерфейсов.

2.3.3.1. Определение операций классов

Операции классов проектирования определяются на основе описаний классов анализа и их зависимостей по следующим правилам:

- сигнатуры операций записываются с учетом синтаксиса языка программирования;
- указывается область видимости операции;
- для каждой зависимости класса определяется одна или более операций;
- на основе описания ответственности формируется список параметров операции и указывается возвращаемое значение;
- специальные требования должны быть учтены.

В итоге определения операций составляется интерфейс для его последующей реализации в классе.

2.3.3.2. Определение атрибутов класса

Атрибуты классов проектирования также определяются на основе описаний классов анализа и их зависимостей по следующим правилам:

- типы и параметры атрибутов указываются с учетом синтаксиса языка программирования и поддерживаемых типов данных;

- атрибуты являются свойством класса и участвуют в операциях классов;
- на основе каждого атрибута класса анализа определяют один или более атрибутов класса проектирования;
- если несколько классов имеют общий атрибут, то его выделяют в отдельный класс.

При сложной структуре атрибутов класса проводится декомпозиция классов с целью их упрощения. Для лучшего понимания зависимостей между атрибутами класса возможно построение диаграммы классов для его атрибутов.

2.3.3.3. Определение ассоциаций и агрегаций

Если на диаграмме последовательностей объект одного класса отправляет сообщение объекту другого класса, то между классами указывается ассоциация или агрегация. Отправка сообщения соответствует вызову метода класса.

Агрегация означает, что вызывающий класс (его объект) имеет ссылку на объект вызываемого класса и через эту ссылку вызывает его метод.

Направленная ассоциация означает передачу сообщения или вызов метода.

При проектировании ассоциаций и агрегаций определяется тип ссылки: глобальная или локальная переменная, параметр метода или указатель на самого себя. В зависимости от выбранного типа связь помечается стереотипом.

После корректировки набора связей уточняются их роли, арность, множественность, классы ассоциации в соответствии с языком программирования. Например, роль транслируется в имя атрибута, хранящего ссылку на связанный класс; класс ассоциации транслируется в новый класс, реализующий связь типа «многие-ко-многим».

2.3.3.4. Определение обобщений

После проектирования атрибутов и операций классов и связей между ними определяют связи типа обобщений между классами. Обобщения либо трассируются из модели анализа, либо определяются в целях вынесения общего поведения или данных классов в базовый класс. Обобщения проектируют в соответствии с языком программирования. Обычно обобщение трассируется в наследование. При отсутствии в языке конструкций для задания наследования возможна замена обобщения на ассоциацию.

2.3.3.5. Описание методов

Если операции класса задают его внешний интерфейс, то методы определяют внутреннюю реализацию поведения классов. Они реализуют операции класса. Описание метода содержит его алгоритм на естественном языке или псевдокоде. Для тривиальных методов составлять описание необязательно.

2.3.3.6. Описание состояний

Если класс имеет сложное поведение, зависящее от его состояния, то составляется диаграмма состояний. Она отражает, каким образом состояние объекта определяет его поведение при получении сообщения.

2.3.4. Проектирование подсистемы

После проектирования отдельных классов выполняют итоговое (уточняющее) проектирование подсистем. Классы распределяют или перераспределяют по подсистемам с тем, чтобы обеспечить выполнение следующих принципов:

- сильная связность классов внутри подсистемы;
- слабое сцепление подсистемами;
- классы подсистемы обеспечивают выполнение ее задач;
- подсистема имеет правильный интерфейс.

Для достижения перечисленных целей необходимо проверить выполнение следующих факторов:

- сохранение зависимостей между подсистемами;
- сохранение представляемых подсистемой интерфейсов;
- сохранение содержимого подсистемы.

Сохранение зависимостей между подсистемами означает, что зависимость между подсистемами происходит от зависимости их элементов (классов или вложенных подсистем). Если существует связь между элементами разных подсистем, то должна присутствовать связь между самими подсистемами. При этом связь между интерфейсами предпочтительнее, чем связь между подсистемами, поскольку позволяет скрыть внутреннюю реализацию.

Сохранение представляемых подсистемой интерфейсов означает, что интерфейс подсистемы складывается из тех ролей, которые она выполняет в прецедентах. Операции интерфейса подсистемы должны соответствовать всем ролям подсистемы. И

содержать все сообщения (операции), получаемые подсистемой (и ее элементами) извне.

Сохранение содержимого подсистемы означает, что функционал подсистемы определяется поведением ее классов. Для каждого интерфейса подсистемы должен быть класс, который его реализует. Для каждого интерфейса каждого прецедента, в котором участвует подсистема, должна быть определена и описана кооперация в терминах элементов этой подсистемы для его реализации.

2.4. Рабочий процесс «Реализация»

Во время реализации создается программный продукт, готовый к использованию. При этом модели, построенные при проектировании, преобразуются по следующей схеме:

- подсистема проектирования трассируется в подсистему реализации, содержащую исполняемые файлы, исходные тексты и т. д.;
- сервисная подсистема – в сервисную подсистему;
- класс проектирования – в один или более (это зависит от языка программирования) файлов компонентов, содержащих исходный текст программы;
- активный класс - в исполняемый компонент;
- один или более проектов кооперации – в билд (сборку);
- модель развертывания и конфигурация сети – в распределенную систему и развертывание исполняемых компонентов.

Процесс реализации достаточно тривиален, и существенная его часть успешно автоматизирована.

В результате реализации создаются следующие артефакты:

- модель реализации – содержит систему реализации, содержащую множество подсистем, компонентов и интерфейсов реализации;
- описание архитектуры – содержит перечень наиболее важных компонентов системы и описывает их размещения по узлам;
- модель размещения – содержит описание сетевой конфигурации;

- план сборки – содержит перечень прецедентов, которые необходимо реализовать на очередном шаге разработки, и перечень компонентов, которые будут при этом созданы;
- компоненты – физические компоненты программной системы, которые обеспечивают ее работу, например, исполняемые программы, базы данных, файлы настроек и т. д.;
- подсистемы реализации – контейнеры для компонентов, чья конкретная реализация зависит от используемых технологий;
- интерфейсы - содержат перечень доступных извне функций компонентов.

Процесс реализации состоит из выполнения следующих действий:

- реализация архитектуры;
- сборка системы;
- реализация подсистем;
- реализация классов;
- тестирование модулей.

Рассмотрим эти действия более подробно.

2.4.1. Реализация архитектуры

Сначала выполняется определение архитектурно-значимых компонентов, которые обеспечивают выполнение основного функционала системы и которые должны быть реализованы в первую очередь.

Компонент — это физический контейнер для элементов системы, в том числе классов. Их реализация зависит от языка программирования. Компоненты могут иметь в том числе следующие стереотипы:

<<process>> — исполняемый файл;

<<message>> — исходные или выходные данные;

<<service>> — сервис;

<<library>> — библиотека;

<<table>> — таблица БД;

<<specification>> — данные для обработки в определенном формате.

Архитектурно-значимые компоненты реализации трассируются из архитектурно-значимых подсистем и классов проектирования. Далее проводят

определение исполняемых компонентов, как наиболее значимых элементов системы, и распределение компонентов по узлам.

Исполняемые компоненты выявляют на основе моделей проектирования, прежде всего активных классов. Каждый активный класс трассируется в исполняемый компонент (утилиту, приложение или сервис) и располагается в соответствующем узле.

2.4.2. Сборка системы

По методологии унифицированного процесса разработка ПО выполняется итерационно. Каждая итерация дает функциональное приращение программного продукта. Итерация состоит из нескольких билдов.

Билд – это реализация нескольких прикладных функций системы, обычно одного или более прецедентов, а также дополнительных требований к ним. В результате создания билда получается несколько компонентов и/или подсистем, которые реализуют этот функционал.

До начала итерации определяется набор и состав ее билдов. Поскольку разработка ведется несколькими командами разработчиков, то в пределах итерации реализация билдов может происходить последовательно и/или параллельно.

2.4.3. Планирование последующего билда

Выбор прецедентов, которые должны быть реализованы в следующем билде, происходит на основе следующих соображений:

- билд реализует набор связанных прецедентов или сценариев;
- билд должен дать инкремент функциональности;
- за один билд следует реализовывать не слишком много новых компонентов, возможно использование заглушек;
- программный проект развивается вверх и в сторону по иерархии подсистем, начало идет с нижнего уровня.

После отбора прецедентов для следующего билда его планирование выполняется по следующей схеме:

1. определить проекты выбранных прецедентов (их кооперaций);
2. определить классы и подсистемы проектирования, входящие в эти прецеденты;
3. определить подсистемы и компоненты реализации, соответствующие этим классам и подсистемам;

4. определить действия по их реализации и проверить реализуемость билда и его соответствие требованиям к правильному билду (см. выше);
5. если проверка успешна, то перейти к реализации, последующему тестированию и интеграции;
6. иначе – внести корректировку в план билда.

После реализации всех компонентов, входящих в билд, происходит его сборка.

2.4.4. Сборка билда

Сборка билда – это интеграция всех его компонентов с целью получить работающую программную систему.

При сборке билда проводится подбор правильных версий подсистем и компонентов, а затем компиляция компонентов снизу вверх.

Перед проведением сборки билда необходимо реализовать и протестировать все компоненты и подсистемы, входящие в него.

2.4.5. Реализация подсистемы

Подсистема реализации — это контейнер физических компонентов ПО, конкретная структура которого зависит от языка и технологии разработки. Например, подсистема – это проект при разработке на Visual C++ или пакет при разработке на Java.

При реализации подсистемы выполняется работа с ее содержимым. Необходимо обеспечить правильную трассировку подсистемы проектирования в подсистему реализации. Проводится сравнение подсистемы проектирования П и ее реализации Р. Следует отследить соблюдение следующих правил:

- каждый класс проектирования, входящий в подсистему П, имеет реализацию в качестве компонента подсистемы Р;
- каждый интерфейс подсистемы проектирования П должен быть представлен как интерфейс компонента полсистемы Р.

Подсистема реализации содержит компоненты, реализующие классы, полученные во время проектирования.

На рис. 11 пример трассировки подсистем в компоненты (всегда один к одному) с сохранением интерфейсов. При этом сохраняются зависимости использования и реализации.

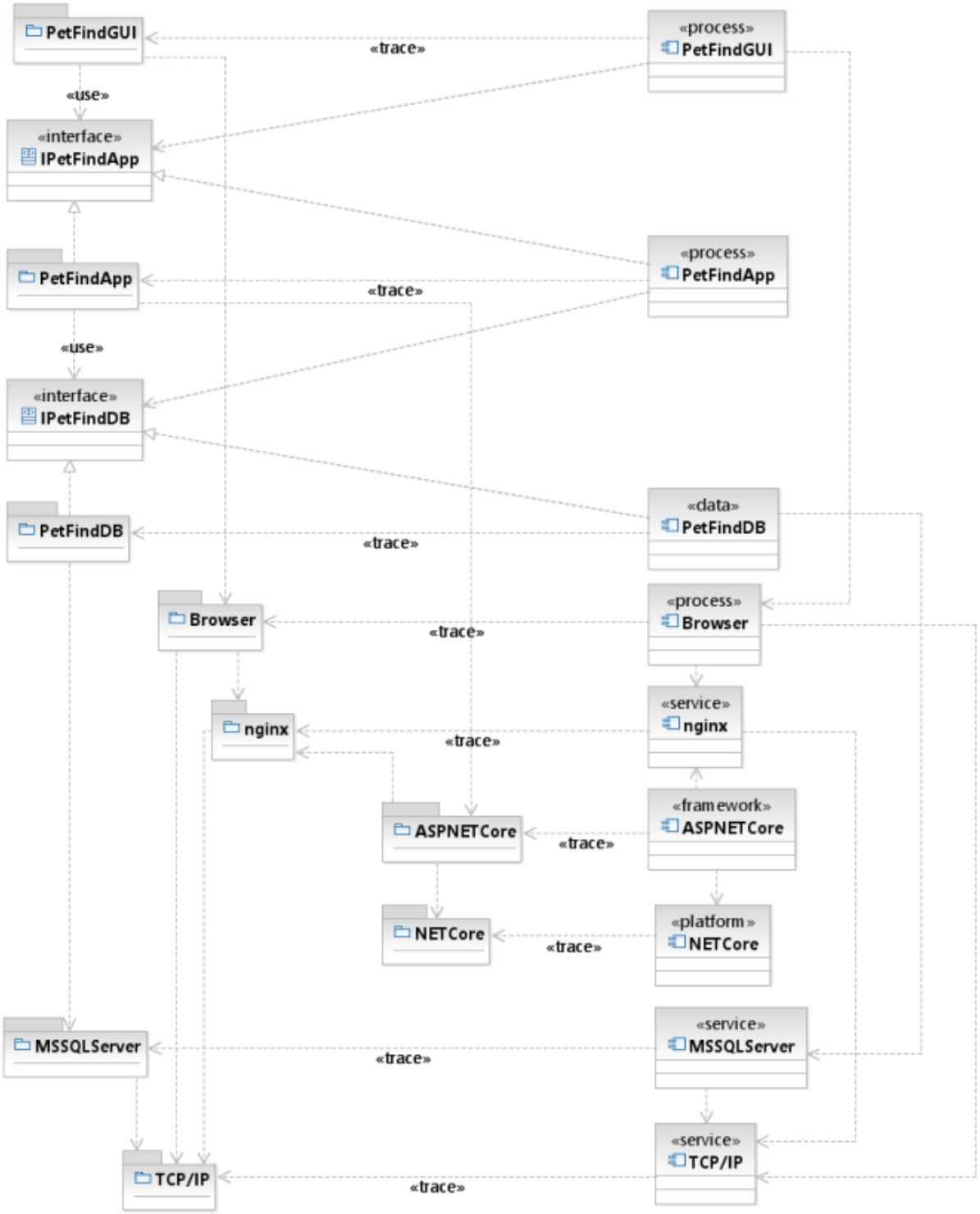

Рисунок 11 – Трассировка подсистем в компоненты

2.4.6. Реализация класса

Реализация класса – это перевод проекта класса в исходный код программы. Реализация класса состоит из выполнения следующих действий:

- описание файловых компонентов;
- генерация кода;
- реализация операций;
- создание компонентов.

Описание файловых компонентов – это оформление будущего компонента, в который трассируется класс проектирования в терминах языка программирования. Описание файловых компонентов зависит от языка программирования и стереотипа класса. Пример на рис. 12.

Рисунок 12 – Трассировка классов проектирования в исходные файлы

Генерация кода для класса проектирования позволяет сформировать команды для его создания. Обычно это действие выполняется автоматически средствами разработки. В результате будет получена конструкция (одна или несколько команд на языке программирования) для создания класса с указанием сигнатур операций и его атрибутов или команды для создания таблицы БД, если класс трассируется в таблицу.

При генерации кода учитываются параметры класса проектирования и возможности языка программирования, например, однонаправленная ассоциации преобразуется в ссылку, а множественная связь – в коллекцию ссылок; роль связи будет преобразована в имя атрибута и т. д.

Реализация операций обычно выполняется программистом вручную. Это кодирование алгоритмов методов.

Завершается реализация классов созданием из них компонентов. Здесь необходимо обеспечить то, что компонент представляет правильные интерфейсы.

Здесь надо указать для каждого компонента требуемые для его сборки файлы - исходники (пример на рис. 13).

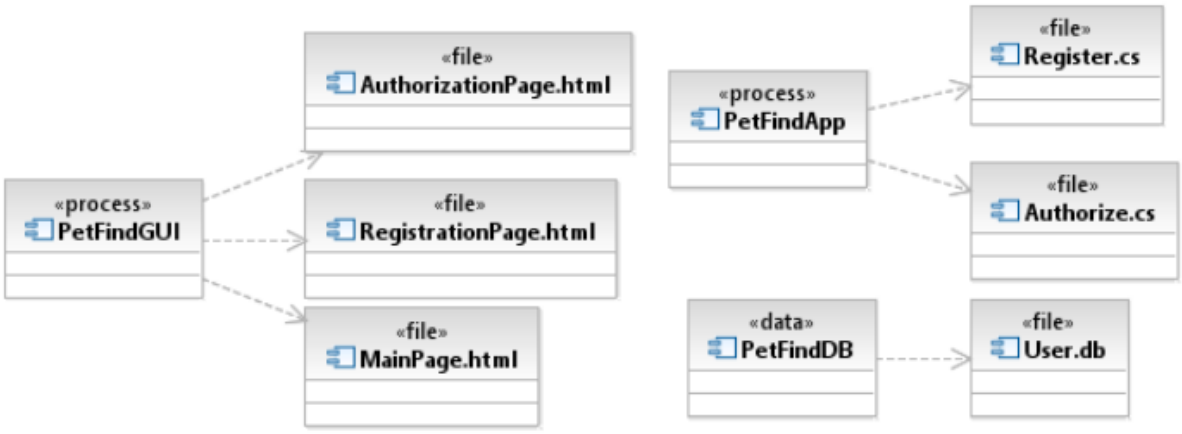

Рисунок 13 – Зависимость компонентов от исходных файлов

Также составляется диаграмма размещения с указанием компонентов (пример на рис. 14).

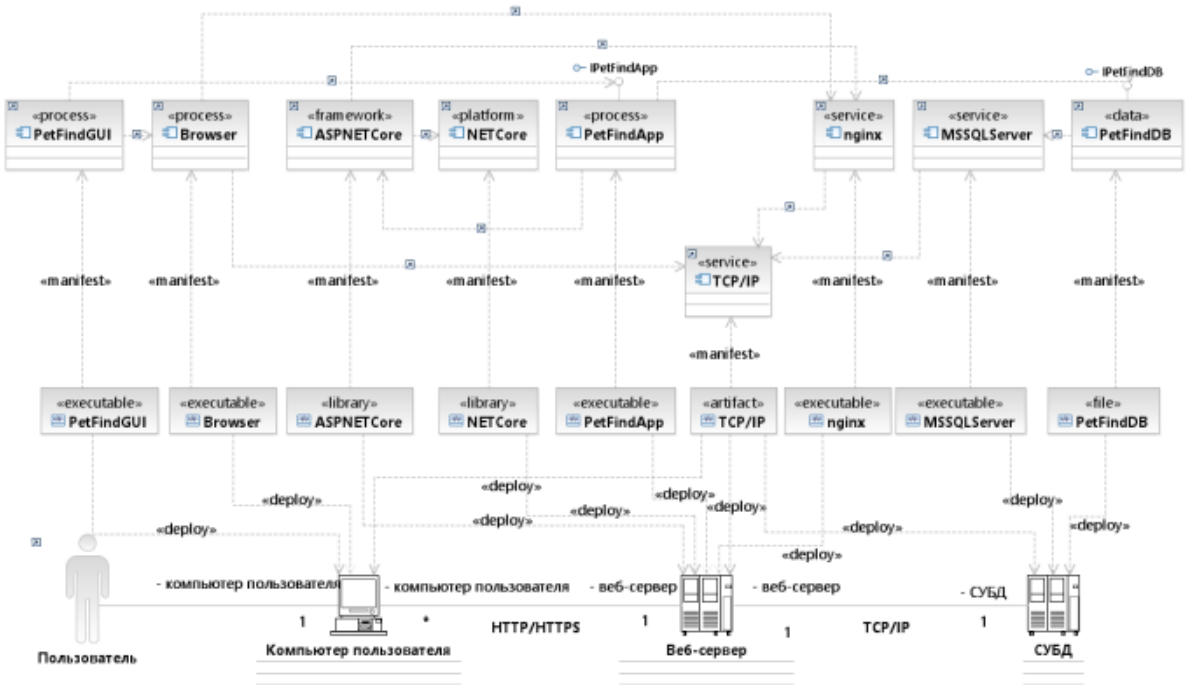

Рисунок 14 – Диаграмма развертывания (с манифестированием)

После реализации классов и подсистем выполняется тестирование.

2.4.7. Тестирование модулей

Тестирование – это проверка правильности работы ПО. Созданные программы запускаются с целью выявить ошибки в их работе. Тестирование модулей позволяет проверить корректность реализации компонентов перед их интегрированием в систему.

Выделяют два вида тестирования модулей:

- тестирование спецификации – «черный ящик»;
- тестирование структуры – «белый ящик».

Тестирование спецификации позволяет на основе описания требований к модулю (прецедентов, которые он реализует) проверить выполнение прикладных функций. Проверяются функциональные и специальные требования. Здесь выявляются ошибки в составлении алгоритмов, передачи входных или выходных данных, инициализации внутренних структур программы. Используются методы функционального тестирования, например методы граничных значений или разбиение на классы эквивалентности.

Тестирование структуры позволяет проверить корректность перевода проекта в программный код, отсутствие синтаксических и семантических ошибок в программе. Здесь выявляются ошибки кодирования алгоритмов и использования переменных, составления условий или задания циклов. Используются методы структурного тестирования, например методы ветвей и границ или проверки циклов.

3. ПРИМЕР ПОСТРОЕНИЯ МОДЕЛЕЙ

Пример для Автоматизированной обучающей системы.

Исходные данные: Классы анализа были объединены в пакеты, итоговая структура пакетов анализа отражена на рисунке 15.

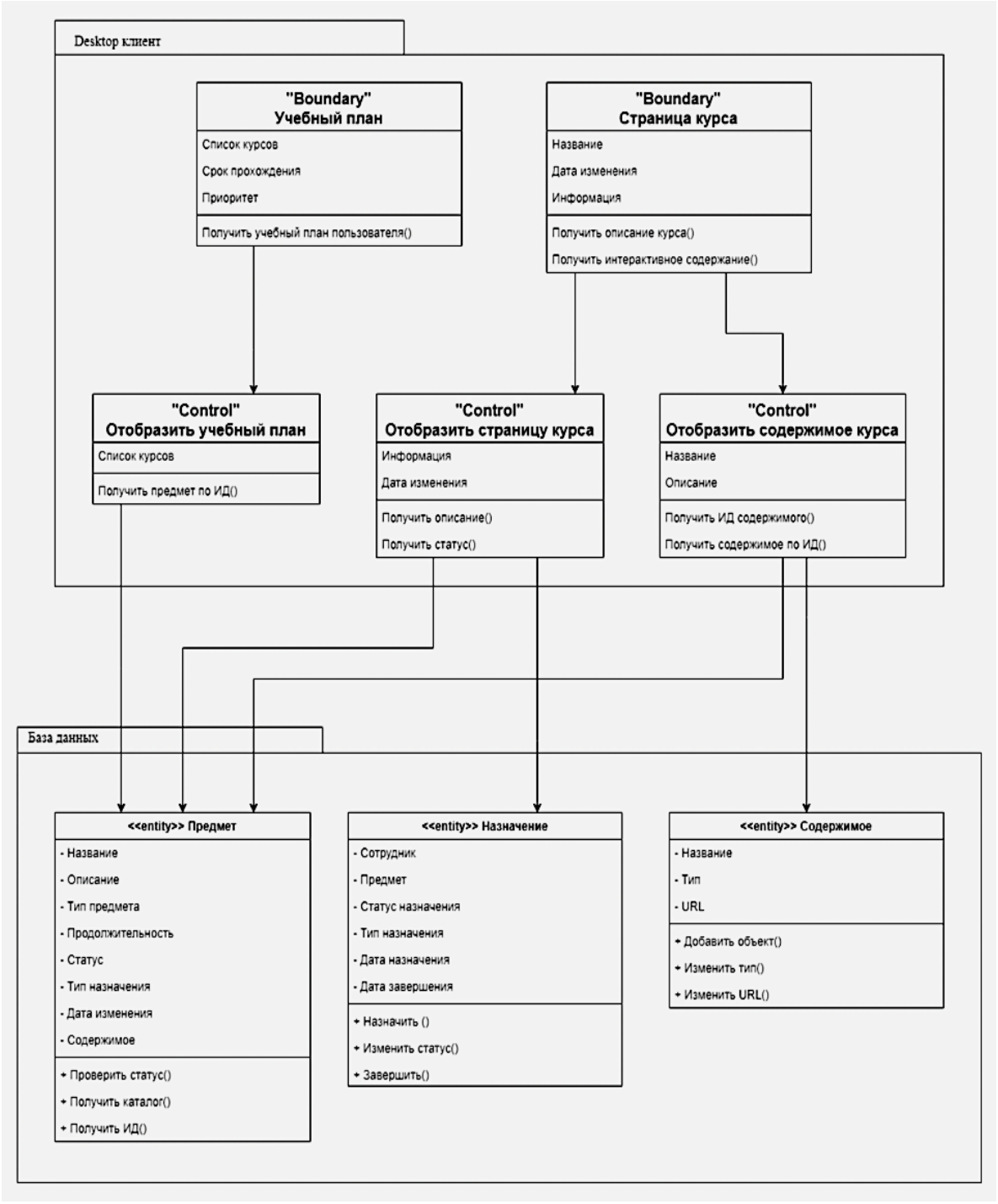

Рисунок 15. Диаграмма классов пакетов анализа и сервисных пакетов

3.1.1. Трассировка элементов моделей анализа в элементы моделей проектирования

После распределения классов анализа по пакетам производится трассировка пакетов анализа в подсистемы. Пакеты трассируются в подсистемы. Классы анализа трассируются в классы проектирования.

Подсистема проектирования – это прообраз будущего программного компонента системы. Она представляет способ группировки элементов системы на этапе проектирования в более крупные блоки.

Пример

В данной предметной области получилось выделить четыре подсистемы – UI_User, User, LMS_DB, Database и представить их на трассировке пакетов анализа в подсистемы на рисунке 16.

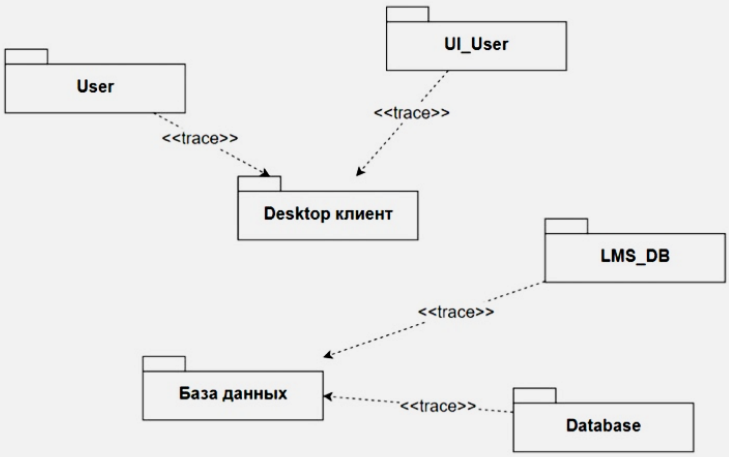

Рисунок 16. Трассировка пакетов в подсистемы

Перенесём на диаграмму классов граничные классы из модели анализа. Добавим новые классы проектирования и свяжем их трассировкой с классами анализа (рисунок 17). Граничные классы трассируются в классы форм и их элементов.

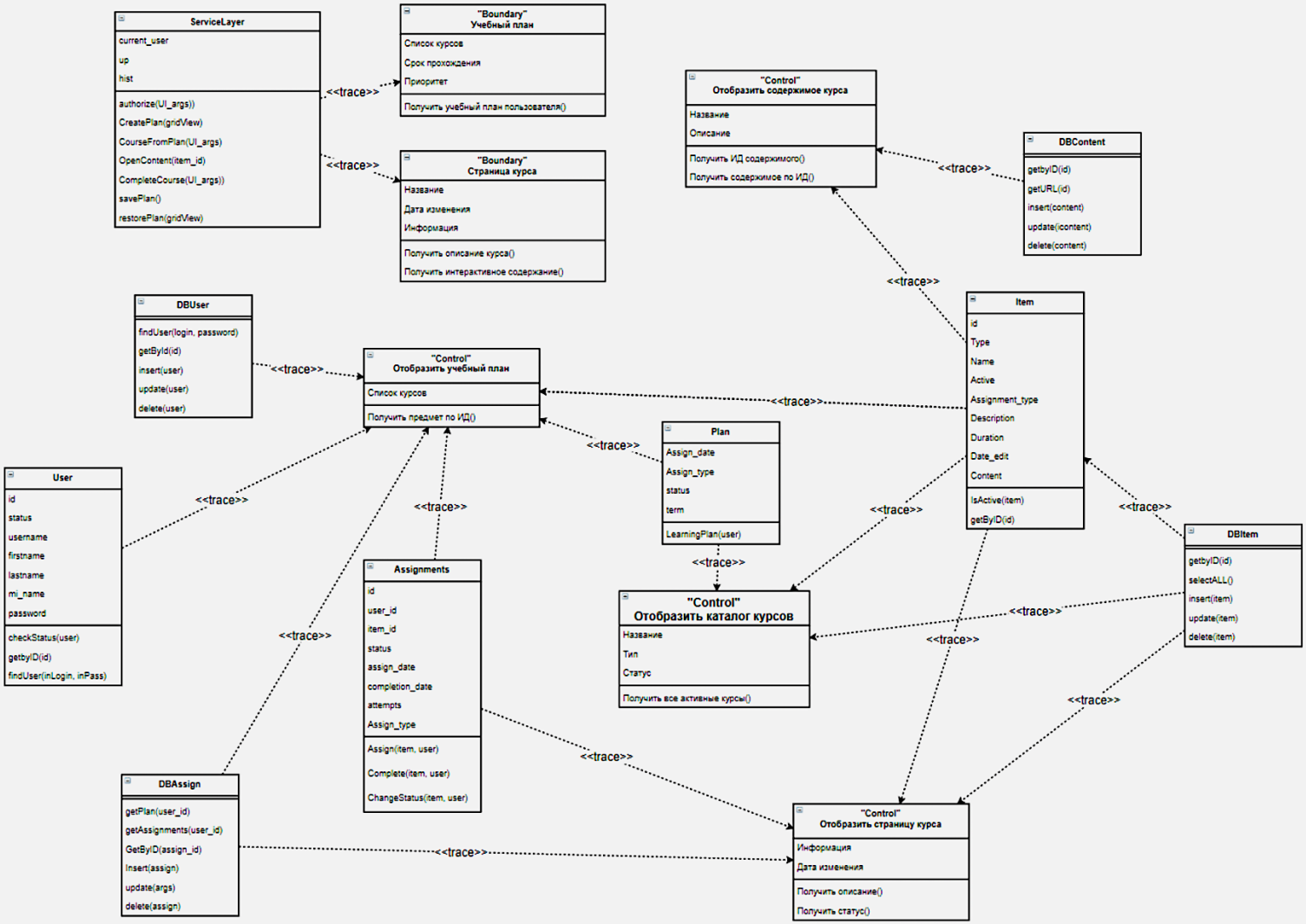

Рисунок 17. Трассировка классов анализа в классы проектирования

Аналогично выполним трассировку классов сущностей из анализа в классы проектирования на рисунке 18. Также классы сущностей обычно трассируются в таблицы, в файлы для хранения информации и в объекты для работы с базой данных.

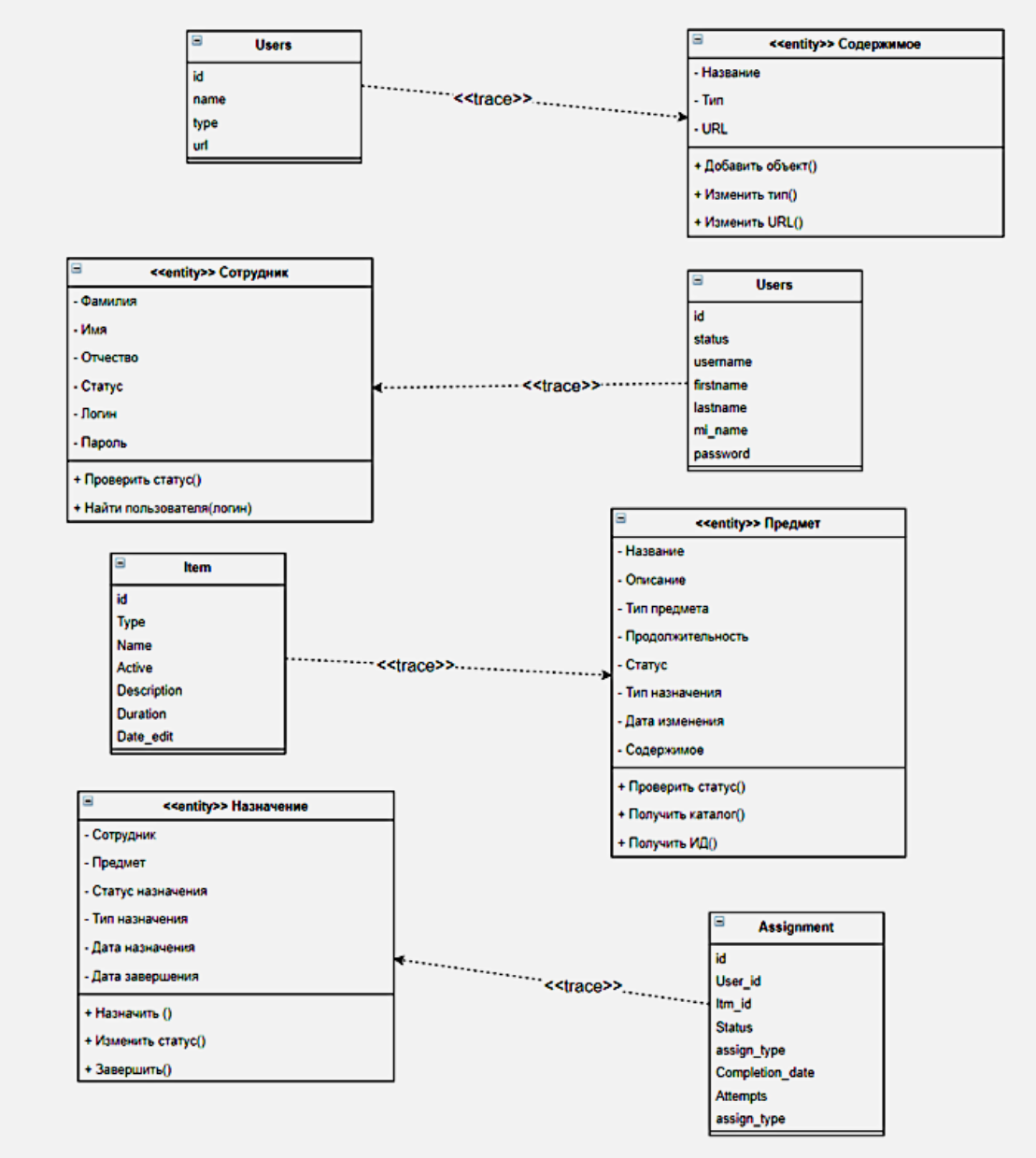

Рисунок 18. Трассировка классов сущностей в классы проектирования

3.1.2. Диаграмма развертывания

Проектирование архитектуры начинают с определения физических компонентов и связей между ними. На этом шаге формируют следующие описания системы:

- перечень узлов и их конфигураций (центрального процесса, оперативной памяти и т. д.);
- конфигурации сети с указанием каналов связи между узлами, используемых протоколов и характеристик передачи, в том числе скорости и качества передачи;
- дополнительные требования, включая требования к безопасности, достоверности, резервному копированию и т. д.

Возможно определение отдельных конфигураций для тестирования или моделирования ПО.

Пример

По итогам определения узлов и сетевых конфигураций составляют диаграмму развертывания (рисунок 19).

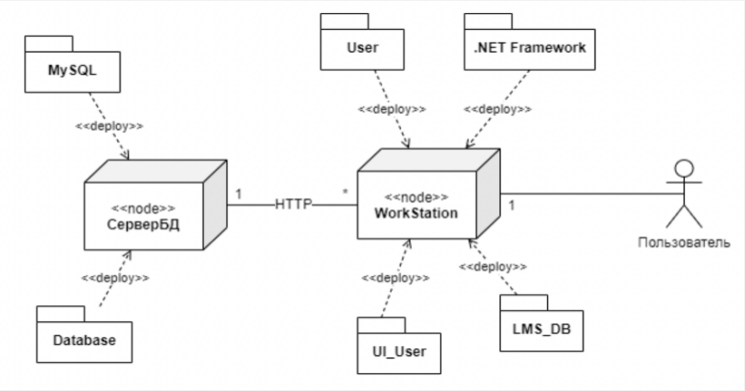

Рисунок 19. Диаграмма развертывания

3.1.3. Определение подсистем и интерфейсов

Подсистемы

Следующим шагом проектирования архитектуры является определение подсистем и их интерфейсов.

Подсистема проектирования — это прообраз будущего программного компонента системы. Она предоставляет способ группировки элементов системы на этапе проектирования в более крупные блоки.

Подсистема проектирования содержит классы проектирования, кооперации, интерфейсы и вложенные подсистемы. При выделении подсистем следует учитывать следующее:

- разработка подсистем может быть параллельной;
- подсистемы имеют сильную связность внутри себя и слабое сцепление между подсистемами;
- подсистема является крупным блоком системы и будет транслирована в исполняемые файлы или библиотеки;
- подсистемы обеспечивают связь с повторно используемыми компонентами;
- подсистемы выполняют обертку унаследованных подсистем.

В приложении выделяют четыре уровня подсистем: специфический и общий прикладной уровни, средний уровень и уровень системного ПО.

Подсистемы прикладного уровня определяют на основе пакетов анализа. Изначально пакеты анализа транслируются в подсистемы проектирования прикладных уровней, сервисные пакеты — в сервисные подсистемы.

Далее проводят уточнение и декомпозицию полученных подсистем по следующим правилам:

- часть пакета анализа, совместно используемую несколькими подсистемами, выделяют в отдельную подсистему;
- часть пакета анализа для многократного применения выделяют в отдельную подсистему, в том числе подсистему среднего уровня или уровня системного ПО;
- унаследованные подсистемы выделяют в отдельные подсистемы;
- выполняют декомпозицию подсистем для возможности разделения работ;
- осуществляют декомпозицию подсистем в целях разнесения их по узлам так, чтобы каждая подсистема была на отдельном узле.

К подсистемам среднего уровня и уровня системного ПО относят следующие компоненты:

- операционную систему;
- СУБД;
- средства коммуникации;
- средства распределенной обработки;
- средства выполнения транзакций;
- средства разработки интерфейсов.

Зависимости подсистем соответствуют зависимостям классов подсистемы или пакетов анализа. Зависимости подсистем проектирования, как и зависимости пакетов

анализа соответствуют вызовам при работе системы. Подсистемы верхних уровней зависят от подсистем нижних уровней, но не наоборот.

Интерфейсы

Для каждой подсистемы следует определить ее интерфейс. Интерфейс подсистемы — это перечень операций, выполняемых данной подсистемой и доступных извне. В интерфейсе указывают полные сигнатуры операций.

Интерфейс подсистемы формируется на основе выявленных зависимостей подсистем и определенных ранее классов в пакете анализа. Если подсистема транслирована из пакета анализа, содержащего класс, к которому имеется обращение извне, то интерфейс подсистемы должен обеспечивать внешнюю ответственность этого класса.

Пример

На рисунках 20 и 22 представлены уровни подсистем и подсистемы и интерфейсы.

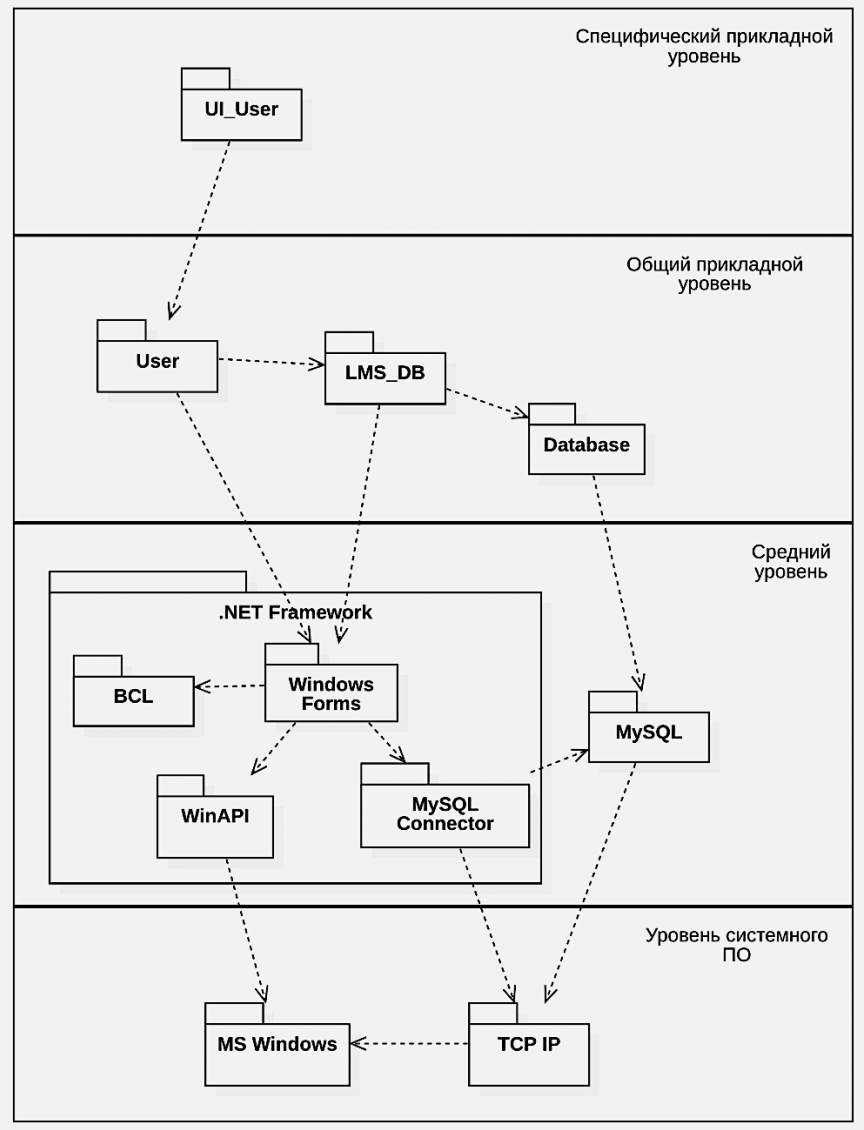

Рисунок 20. Уровни подсистем

Покажем взаимосвязь класса и интерфейса (рисунок 21). Интерфейс – это те же методы и функции в реализуемом пакете.

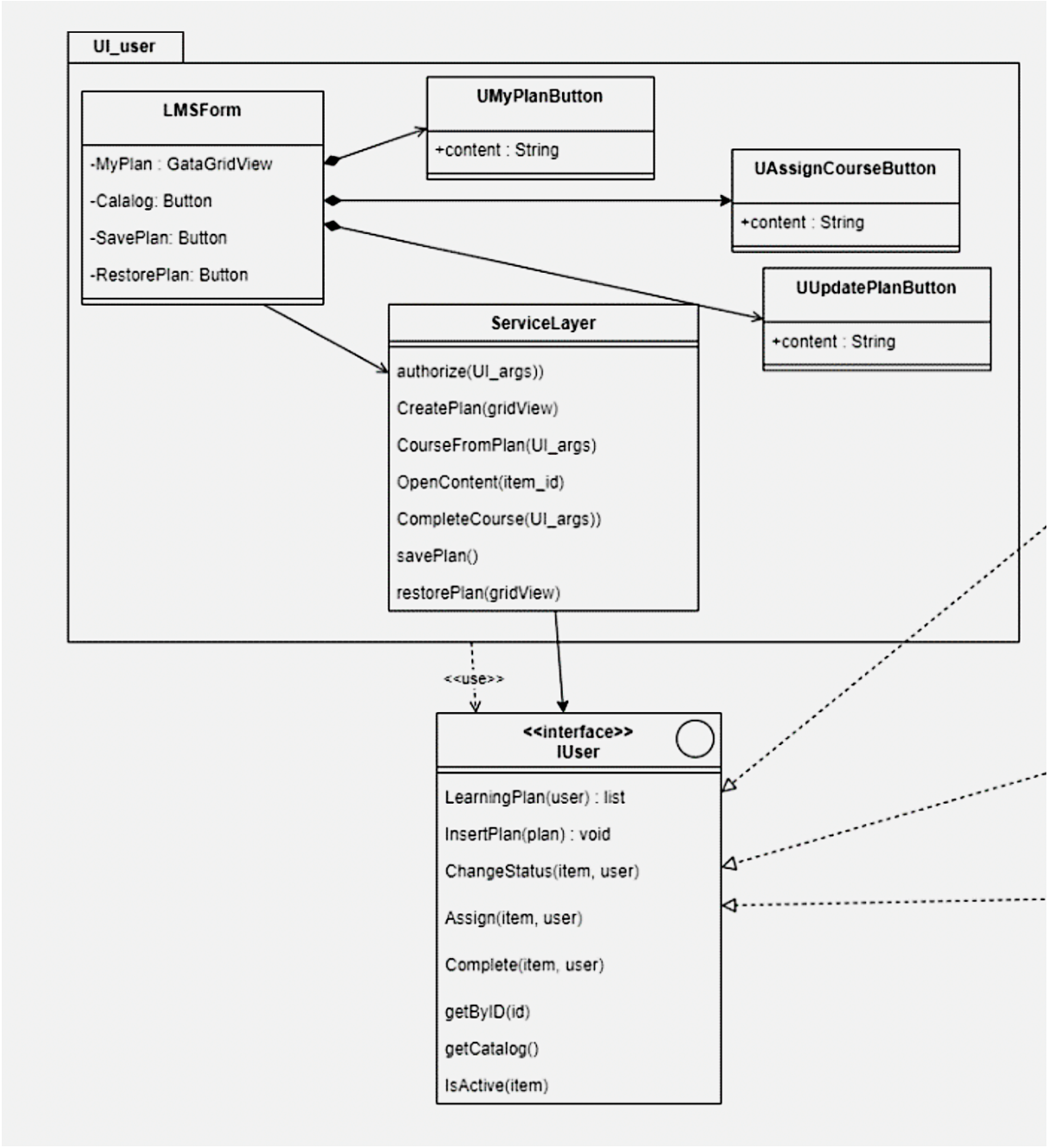

Рисунок 21. Связь класса и интерфейса

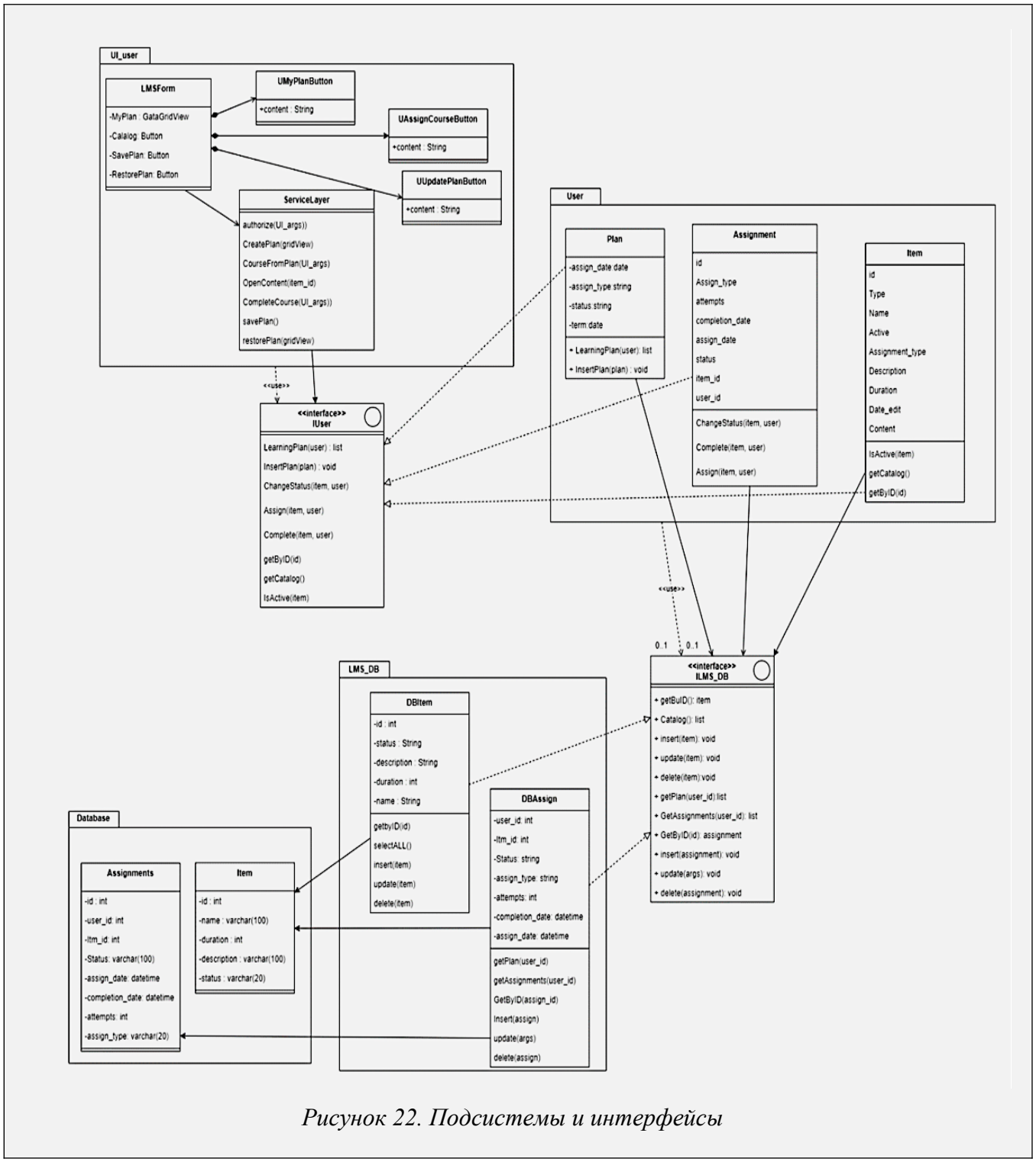

Рисунок 22. Подсистемы и интерфейсы

3.1.4. Описание взаимодействия объектов проектирования

Для описания алгоритма выполнения прецедента составляют диаграммы последовательностей, по одной на каждый поток событий прецедента (возможную последовательность действий). Диаграммы последовательностей содержат объекты классов, участвующих в прецеденте, и поток событий между ними.

Пример

Приведем пример из курсовой работы. На рисунке 23, 24 приведены диаграммы последовательностей.

На рисунке 23 диаграмма последовательностей представлена в терминах класса.

На рисунке 24 диаграмма последовательностей в терминах подсистем

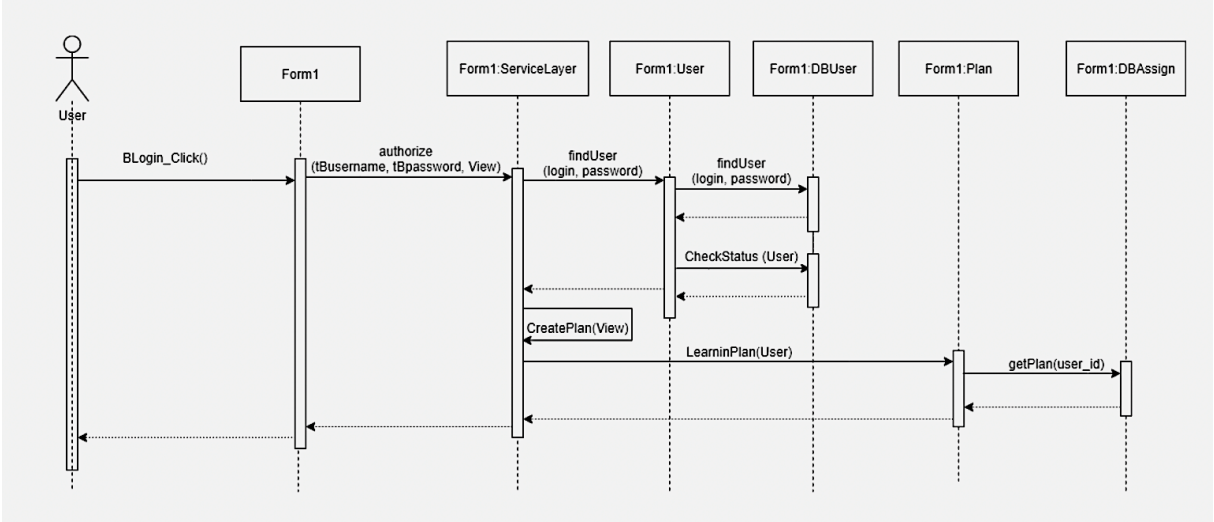

Рисунок 23. Диаграмма последовательностей, реализующая вход пользователя

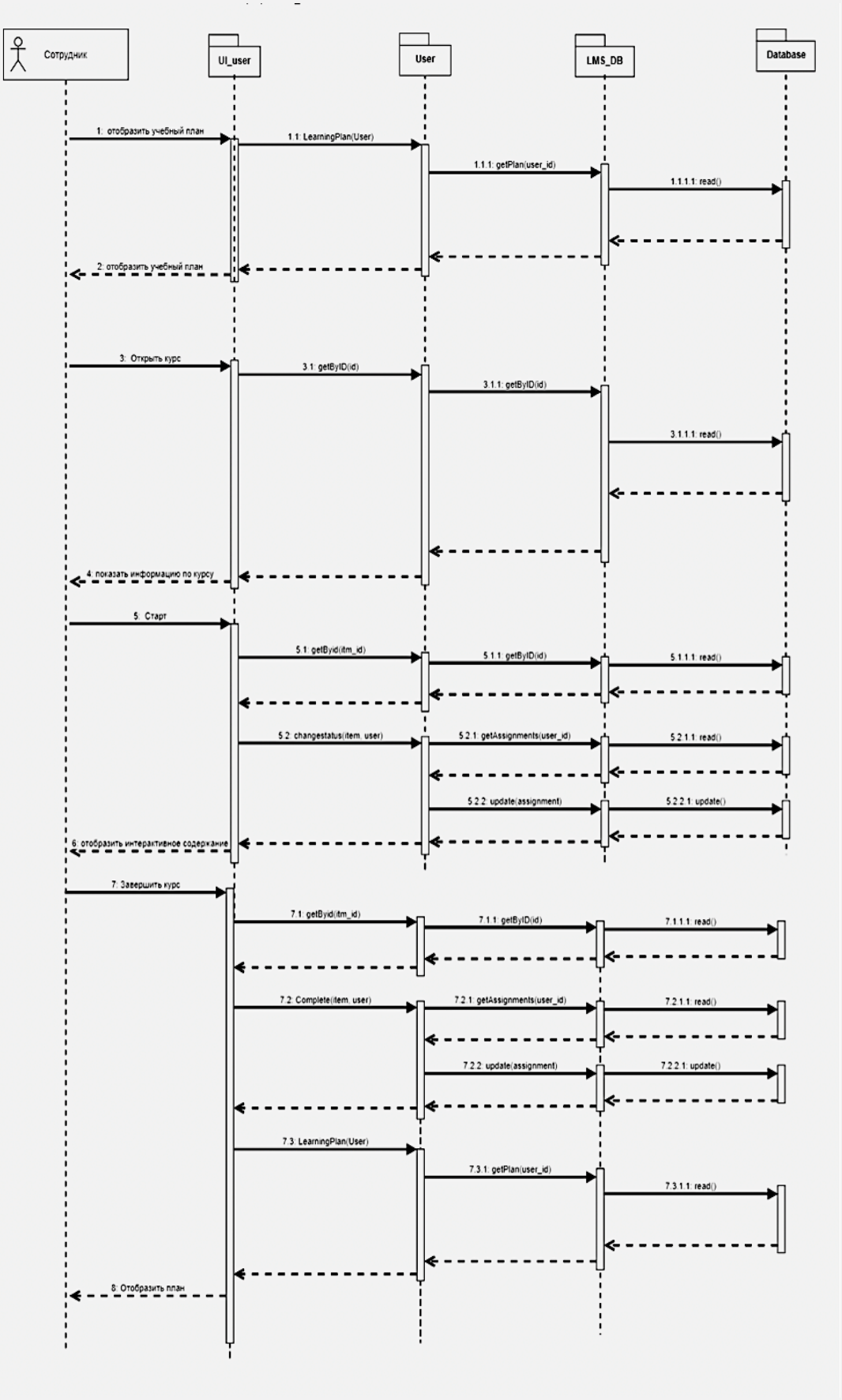

Рисунок 24. Диаграмма последовательностей (подсистемы)

3.1.5. Программная система

Итогом проектирования является множество моделей, подлежащих дальнейшему их преобразованию в работающую программную систему. Во время реализации создается программный продукт, готовый к использованию. При этом модели, построенные при проектировании, трассируются следующим образом:

- подсистема проектирования — в подсистему реализации, содержащую исполняемые файлы, исходные тексты и т. д.;
- сервисная подсистема — в сервисную подсистему;
- класс проектирования — в один или более (это зависит от языка программирования) файлов компонентов, содержащих исходный текст программы;
- активный класс — в исполняемый компонент;
- один или более проектов кооперации — в билд (сборку);
- модель развертывания и конфигурация сети — в распределенную систему и развертывание исполняемых компонентов.

Процесс реализации достаточно тривиален, и существенная его часть успешно автоматизирована. В результате реализации создаются следующие артефакты:

- модель реализации — система реализации, содержащая множество подсистем, компонентов и интерфейсов реализации;
- описание архитектуры — перечень наиболее важных компонентов системы и их размещение по узлам;
- модель размещения — описание сетевой конфигурации;
- план сборки — перечень прецедентов, которые необходимо реализовать на очередном шаге разработки, и перечень компонентов, которые будут при этом созданы;
- компоненты — физические компоненты программной системы, обеспечивающие ее работу, например: исполняемые программы, базы данных, файлы настроек и т. д.;
- подсистемы реализации — контейнеры для компонентов, чья конкретная реализация зависит от используемых технологий;
- интерфейсы — перечень доступных извне функций компонентов.

Модель реализации, описание архитектуры и модель размещения составляет архитектор. План сборки — системный интегратор. За реализацию компонентов, подсистем и интерфейсов отвечает инженер по компонентам.

3.1.6. Диаграмма компонентов и развертывания для реализации архитектуры

Реализацию архитектуры начинают с определения физических компонентов и связей между ними. На этом шаге формируют следующие описания системы:

- перечень узлов и их конфигураций (центрального процесса, оперативной памяти и т. д.);
- конфигурации сети с указанием каналов связи между узлами, используемых протоколов и характеристик передачи, в том числе скорости и качества передачи;
- дополнительные требования, включая требования к безопасности, достоверности, резервному копированию и т. д.

Возможно определение отдельных конфигураций для тестирования или моделирования ПО.

Архитектурно значимые компоненты реализации трассируют из архитектурно значимых подсистем и классов проектирования. Далее проводят определение исполняемых компонентов как наиболее значимых элементов системы и осуществляют распределение компонентов по узлам. .

Пример

На рисунке 25 графически представим данные по трассировке подсистем в компоненты. Подсистемы трассируются в компоненты 1 к 1.

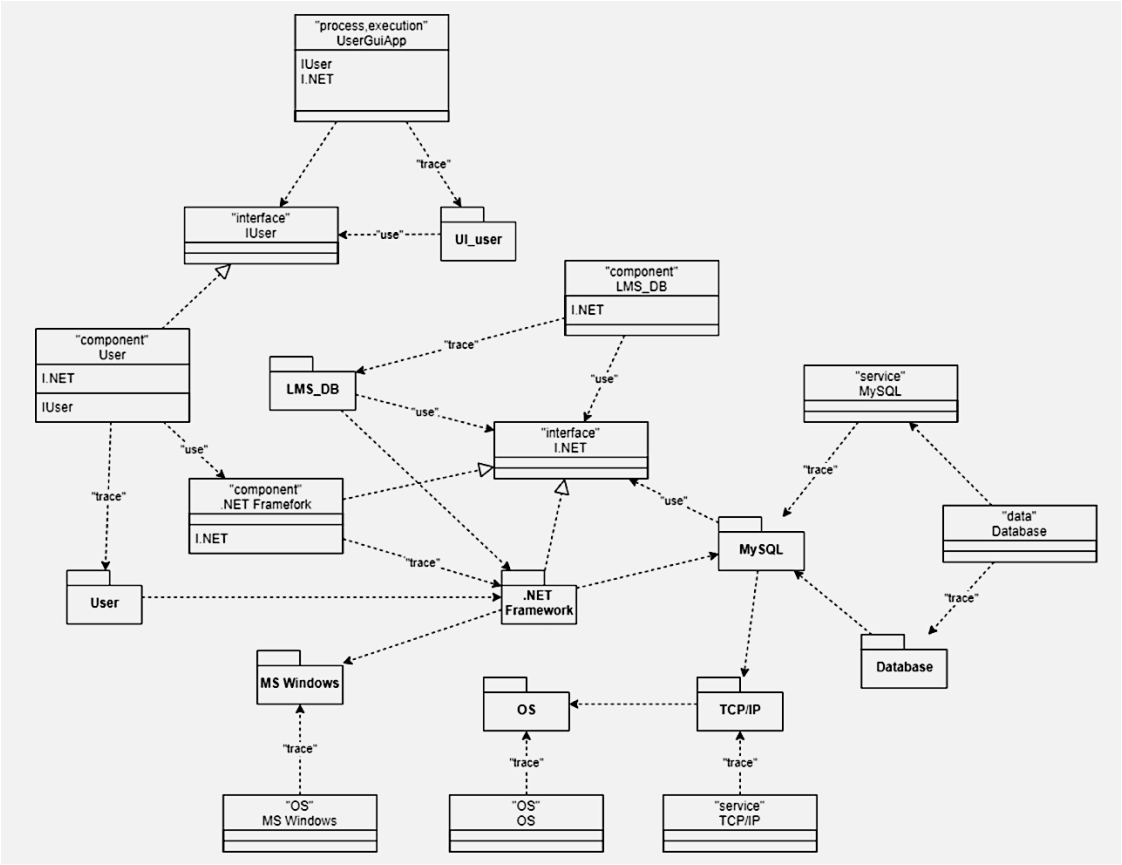

Рисунок 25. Трассировка подсистем в компоненты

По итогам определения узлов и сетевых конфигураций составляют диаграмму развертывания (рисунок 26).

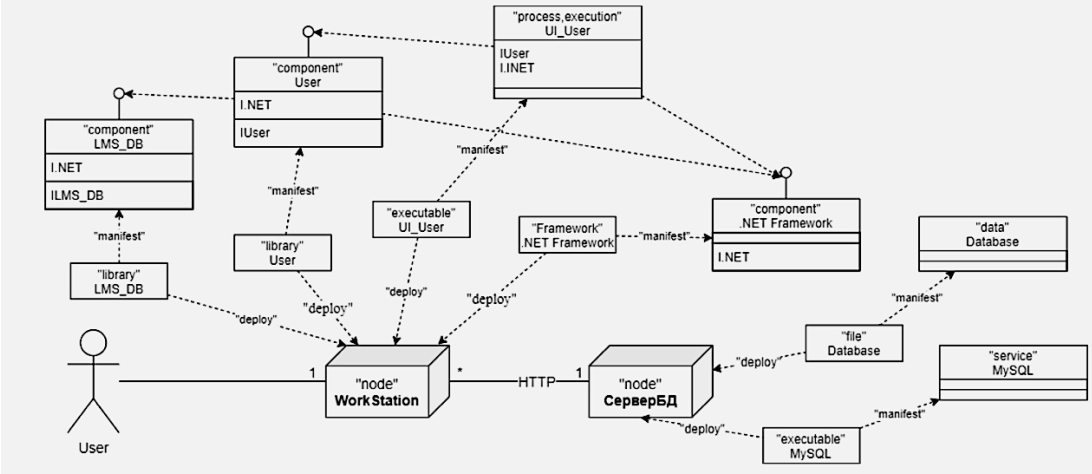

Рисунок 26 Диаграмма компонентов и развертывания для реализации архитектуры

Отразим на рисунке 27 трассировку классов проектирования в исходные файлы в сокращенном варианте.

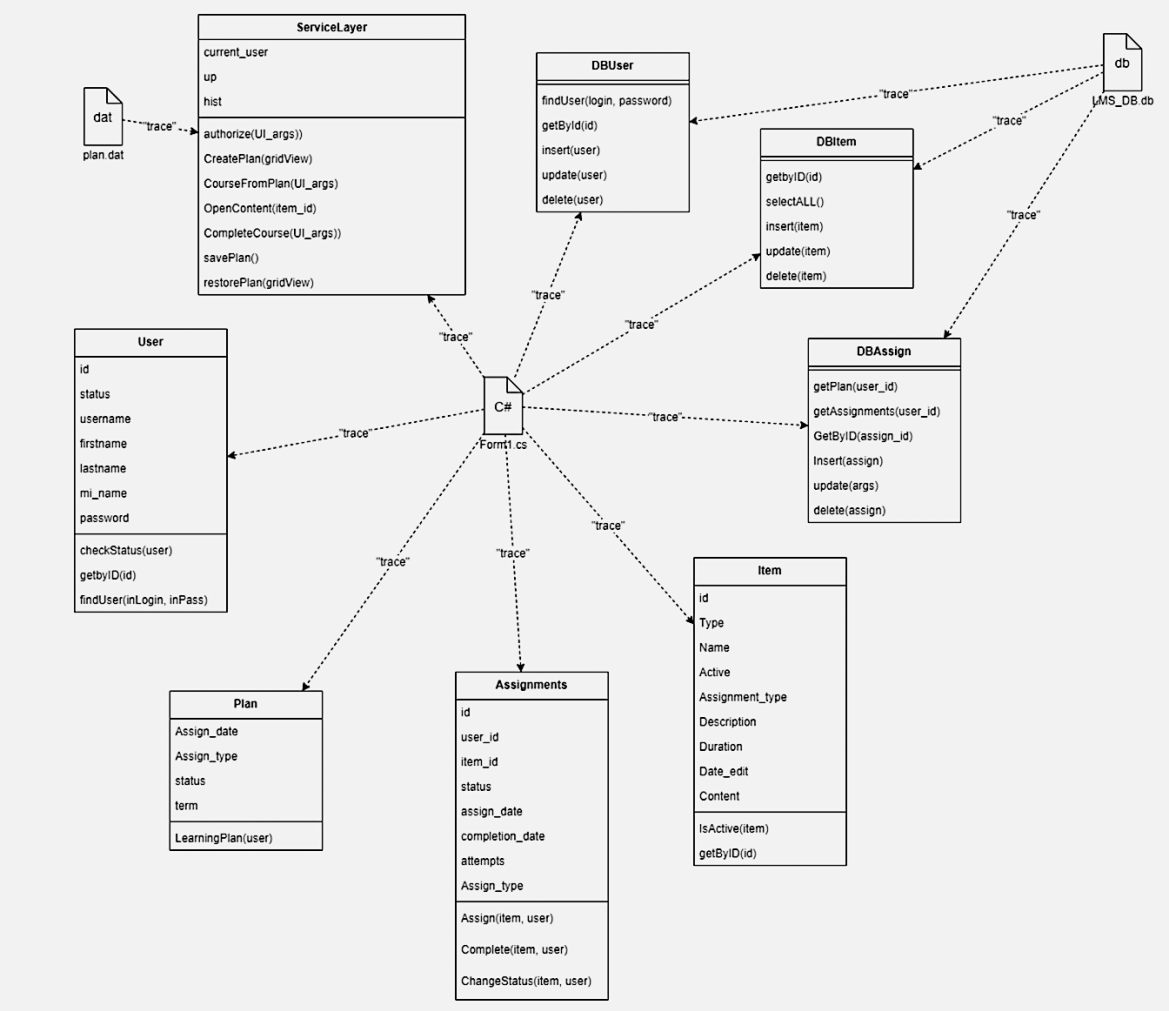

Рисунок 27. Трассировка классов проектирования в исходные файлы

4. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Унифицорованный процесс разработки - RUP. Его особенности, этапы и рабочие процессы.
2. Каково назначение рабочего процесса проектирования?
3. Какие артефакты создаются в процессе его выполнения? Из каких действий он состоит?
4. Что такое архитектура системы? Что входит в ее описание? Как выполняют проектирование архитектуры?
5. Что содержит и как составляется модель развертывания?
6. Как определяют подсистемы проектирования и сервисные подсистемы? Назовите уровни подсистем и охарактеризуйте назначение подсистем каждого уровня.
7. Что такое интерфейс? Как формируют интерфейс подсистемы? Кто реализует интерфейс подсистемы?
8. Что такое активный класс? Каковы правила его выявления?
9. Что такое обобщенные механизмы проектирования? Как их определяют?
10. Как выполняют проектирование прецедентов? Как составляют диаграммы классов-участников, диаграммы участвующих подсистем и диаграммы их взаимодействия?
11. Каким образом пакеты и класса анализа трассируют в подсистемы и классы проектирования? На что влияет стереотип класса анализа?
12. Как выполняют проектирование класса? Что будет его результатом?
13. В чем цель проектирование подсистем? Как определяют зависимости подсистем?
14. Каково назначение рабочего процесса реализации? Из каких действий он состоит? Что содержит модель реализации?
15. Как реализуют подсистему и какие проверки при этом выполняют?
16. Из каких действий состоит реализация классов? Какие из них выполняются автоматически?
17. Во что трассируются классы, активные классы, интерфейсы и подсистемы проектирования при реализации?
18. Как реализуют архитектуру и что будет получено в результате?

19. Что такое компонент? Как определяется его интерфейс?
20. Из каких действий состоит сборка системы? Что такое план сборки? Как и когда его составляют?
21. Что такое билд? Каким образом определяют набор прецедентов для следующего билда?

5. СПИСОК ИСТОЧНИКОВ

1. Виноградова М.В., Белоусова В.И. Унифицированный процесс разработки программного обеспечения: учебное пособие / Виноградова М.В., Белоусова В.И. – М.: МГТУ им.Н.Э. Баумана. – 2015 г. – 82 с. - Текст. Изображение. : электронные // – URL: http://ebooks.bmstu.ru/catalog/193/book1303.html (дата обращения 06.06.2022)
2. Якобсон А, Дуч Г, Рамбо Дж. Унифицированный процесс разработки программного обеспечения. - Спб.: Питер. - 2002 г.
3. Орлов С.А., Цилькер Б.Я. Технологии разработки программного обеспечения. - СПб.: Питер. - 2012 г.