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 лабораторным работам по курсу «Технологии разработки программного обеспечения»

Лабораторная работа №6

«Разработка модели анализа»

Виноградова М.В., Королев С.В., Умряев Д.Т.

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

Москва, 2022 г.

ОГЛАВЛЕНИЕ

1. ЗАДАНИЕ

Лабораторная работа №6 «Разработка модели анализа» по курсу «Технологии разработки Программного обеспечения».

1.1. Цель работы

- Изучить унифицированный процесс разработки (RUP);
- Приобрести умения построения модели анализа;
- Получить навыки построения модели анализа в среде Sparx Enterprise Architect.

1.2. Средства выполнения

- Sparx Enterprise Architect [1.].

1.3. Исходные данные

- проект Sparx Enterprise Architect, содержащий модель требований (был создан ранее);
- набор актеров и прецедентов (созданы ранее),
- спецификации прецедентов (созданы ранее),
- прототип пользовательского интерфейса (создан ранее).

1.4. Продолжительность работы

Время выполнения лабораторной работы 4 часа.

1.5. Пункты задания для выполнения

1. Открыть в среде Sparx Enterprise Architect проект, созданный ранее. Добавить к нему модель анализа RUP.
2. Добавить в модель анализа кооперации для реализации основных прецедентов. Для каждого прецедента одну кооперацию. Название кооперации совпадают с названием прецедента.

3. Построить «обзорную» диаграмму кооперaций (диаграмма классов).
4. Выполнить анализ архитектуры и построить в пакете Перспектива «обзорную» диаграмму классов сущностей (на основе модели предметной области). Классы сущностей копировать в модель из строительных блоков. Определить для них атрибуты, связи ассоциации, роли, множественность и арность.
5. Добавить в модель управляющие и граничные классы. Управляющий класс — один на каждую кооперацию. Граничный класс — один на каждого актера. Построить в пакете Перспектива «обзорную» диаграмму граничных классов и «обзорную» диаграмму управляющих классов.
6. Выполнить анализ кооперaций. Для каждой кооперации: открыть диаграмму классов кооперации и переместить на нее классы (граничные, управляющий и сущностей), которые участвуют в этой кооперации. Определить ассоциации между классами в соответствии с их взаимодействием для реализации прецедента (общая схема: граничный — управляющий — сущности). Создать диаграмму последовательностей кооперации (для выполнения основного потока алгоритма). В качестве объектов использовать классы-участники кооперации и актера. (Первое действие всегда от актера.) События диаграммы будут прообразом его методов.
7. Проверить модель средствами пакета (валидация модели).
8. Выполнить анализ классов. После построения диаграммы последовательностей уточнить на диаграмме классов кооперации: связи между классами, методы классов (названия и атрибуты), атрибуты классов. Класс сущности: атрибуты — поля хранимых данных, методы — get|set или CRUD. Класс граничный: атрибуты — элементы формы (на основе эскиза пользовательского интерфейса)/параметры протокола, методы — управляющие сигналы от пользователя (их обработчики). Классы управляющие: атрибуты — промежуточные данные алгоритма, методы — шаги алгоритма кооперации.
9. Проверить модель средствами пакета (валидация модели).
10. Построить «обзорную» диаграмму всех классов. Добавить в модель пакеты. Распределить классы анализа по пакетам. Указать зависимости между пакетами.

1.6. Содержание отчета

- Титульный лист;
- Цель работы;
- Задание;
- Описание модели требований:
  - Диаграмма прецедентов;
- Описание модели анализа требований:
  - Трассировка прецедентов в кооперации;
  - Обзорные диаграммы классов сущностей, управляющих и граничных;
  - Диаграмма классов – участников кооперации;
  - Диаграммы последовательностей;
  - Обзорная диаграмма классов;
  - Диаграмма пакетов (с классами);
- Вывод;
- Список используемой 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 [2., 3.].

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 [2., 3.]:

- **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** — 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.

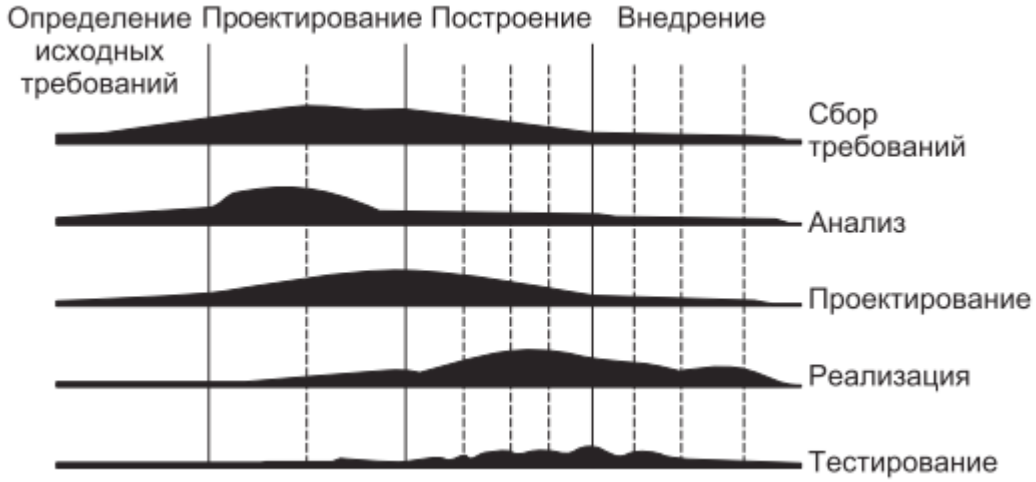

Рисунок 1 – Рабочие процессы на этапах RUP

В данной лабораторной работе мы будем рассматривать второй процесс, а именно **Анализ требований**.

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

2.3. Enterprise Architect

Enterprise Architect представляет собой мощный UML-инструмент для профессионалов, занимающихся разработкой, тестированием и внедрением программного обеспечения. Enterprise Architect полностью поддерживает нотацию языка UML 2.1 и позволяет создавать диаграммы всех типов [5.].

Enterprise Architect (EA) – CASE-инструмент для проектирования и конструирования программного обеспечения. Некоторые из ключевых функций EA, которые используются в текущем курсе:

- создание элементов UML-моделей широкого круга назначения
- размещение этих элементов в диаграммах и пакетах;
- создание связей между элементами;
- документирование созданных элементов;
- генерация кода для конструируемого ПО;
- обратное проектирование имеющегося кода на некоторых языках.

Используя EA, можно выполнять генерацию и обратное проектирование для C++, C#, Delphi, Java, Python классов, синхронизировать код и элементы моделей, проектировать и генерировать элементы баз данных.

2.3.1. Диаграмма трассировки кооперации

Создание Диаграммы кооперации

Рассмотрим основные шаги создания Диаграммы трассировки кооперации.

1) Выбираем любой уже существующий **Пакет** или создаем новый, выбираем «**Add diagram…**» или нажимаем соответствующую кнопку в шапке области **Browser**. В результате откроется окно **New Diagram** (Мастер создания диаграмм).

2) Переходим в категорию «**UML Structural**» и выбираем тип диаграммы «**Object**» (см. Рисунок 2). После этого нажимаем «**Ок**».

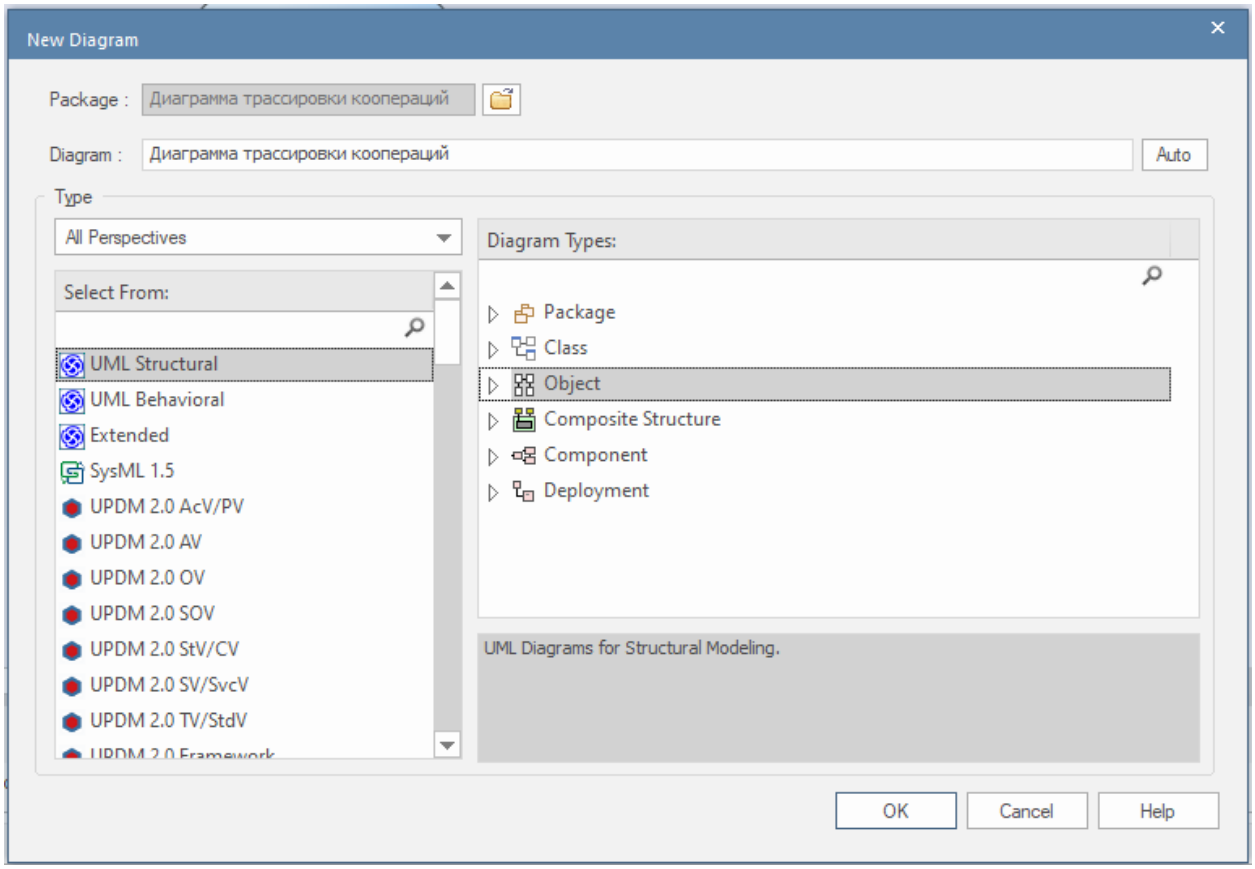

Рисунок 2 – Создание диаграммы трассировки кооперации

В результате в проект добавится новая Диаграмма трассировки кооперации и сразу откроется в области редактирования диаграмм.

Можно заметить, что в области **Toolbox**, появились элементы, специализированные под данный тип диаграмм (см. Рисунок 3).

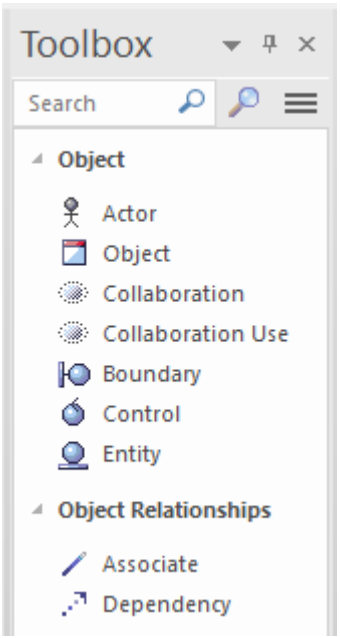

Рисунок 3 – Специализированные элементы области Toolbox для Диаграмм кооперация

Создание кооперации

1) Для создания новой кооперации необходимо в **Toolbox** диаграммы выбрать элемент **Collaboration** (см. Рисунок 3) и перетащить его на рабочую зону.

В результате у нас на рабочей области появится стандартная кооперация с описанием по умолчанию (см. Рисунок 4). После этого можем менять имя кооперации по своему усмотрению.

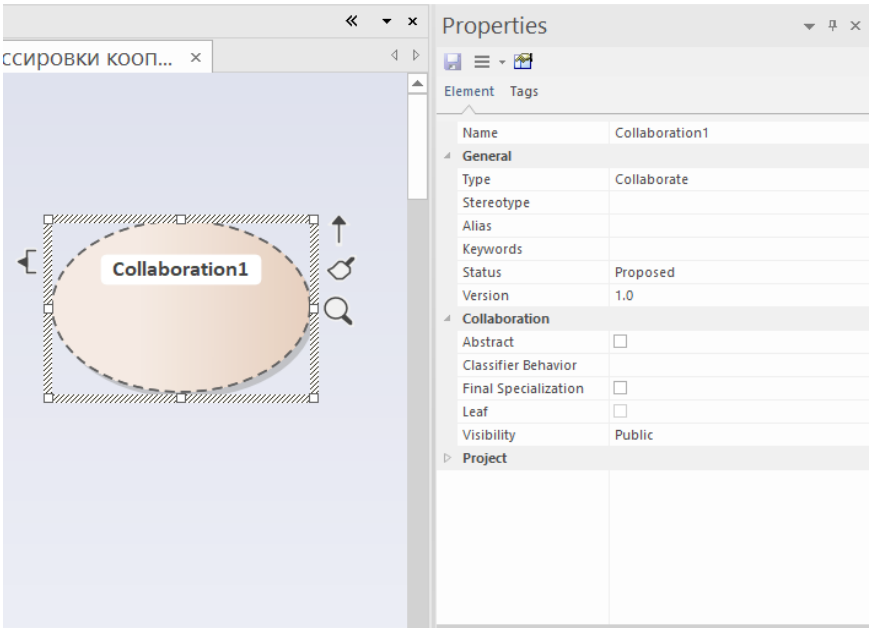

Рисунок 4 – Создание элемента типа Collaboration

2) Перетащим необходимый для нас прецедент из области «Browser» на рабочую область в виде ссылки на соответствующий элемент из модели (см. Рисунок 5).

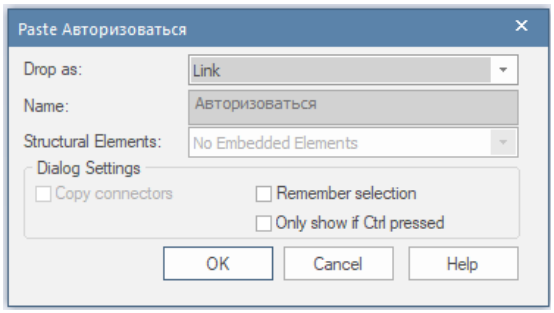

Рисунок 5 – Настройка ссылки на элемент Use Case

3) Для того чтобы скрыть источник прецедента необходимо отключить параметр **«Show Namespace»** в разделе **«Diagram»** в настройках вашей диаграммы кооперации (см. Рисунок 6).

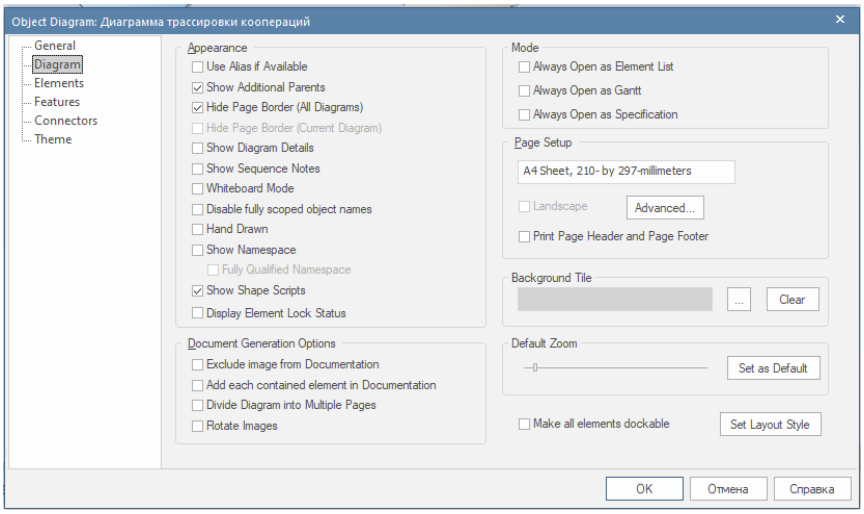

Рисунок 6 – Настройка параметров Диаграммы кооперации

Добавление связей между прецедентом и кооперацией

1) Соединим Кооперацию и Прецедент связью со стереотипом **«trace»** (находится в разделе Common Relationships) (см. Рисунок 7).

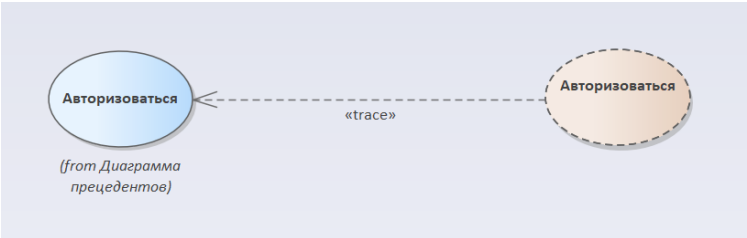

Рисунок 7 – Пример трассировки прецедента в кооперацию

2.3.2. Диаграмма классов

Создание Диаграммы классов

1) Выбираем любой **Пакет** и выбираем «**Add diagram…**» или нажимаем соответствующую кнопку в шапке области **Browser**. В результате откроется окно **New Diagram** (Мастер создания диаграмм).

2) Переходим в категорию «**UML Structural**» и выбираем тип диаграммы «**Class**» (см. Рисунок 8). После этого нажимаем «**Ок**».

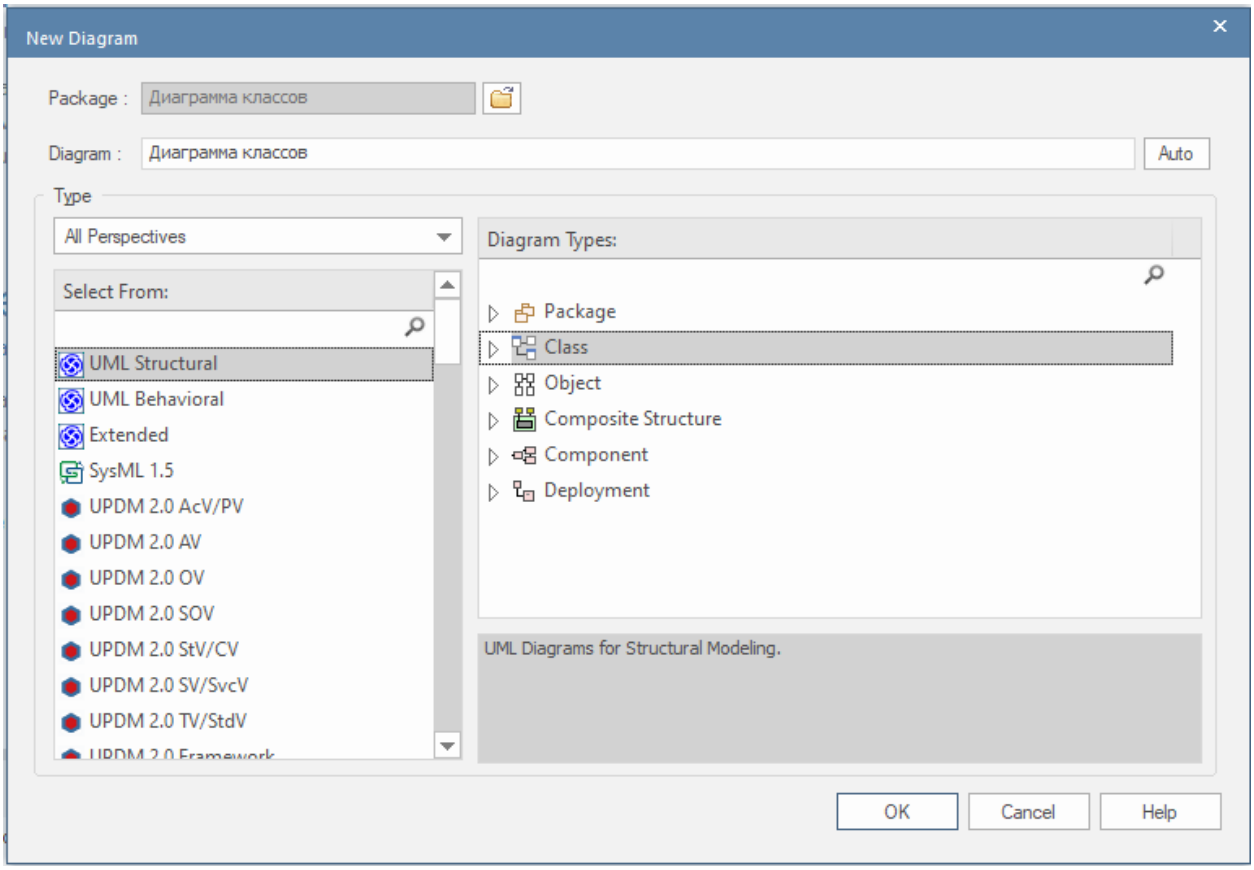

Рисунок 8 – Создание Диаграммы классов

В результате в проект добавится новая Диаграмма классов и сразу откроется в области редактирования диаграмм.

Можно заметить, что в области **Toolbox**, появились элементы, специализированные под данный тип диаграмм (см. Рисунок 9).

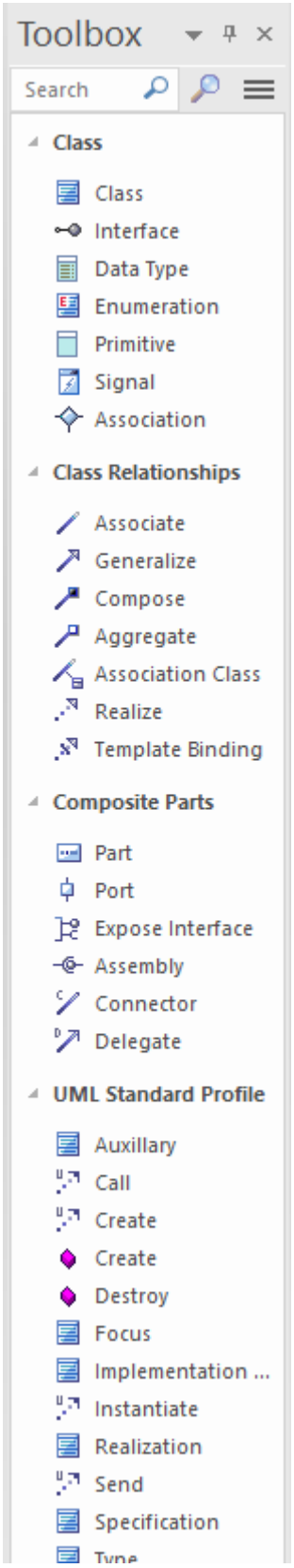

Рисунок 9 – Специализированные элементы области Toolbox для Диаграммы классов

Создание класса

1) Для создания нового класса необходимо в **Toolbox** диаграммы выбрать элемент **Class** (см. Рисунок 9) и перетащить его на рабочую зону.

В результате у нас на рабочей области появится стандартный класс с описанием по умолчанию (см. Рисунок 10).

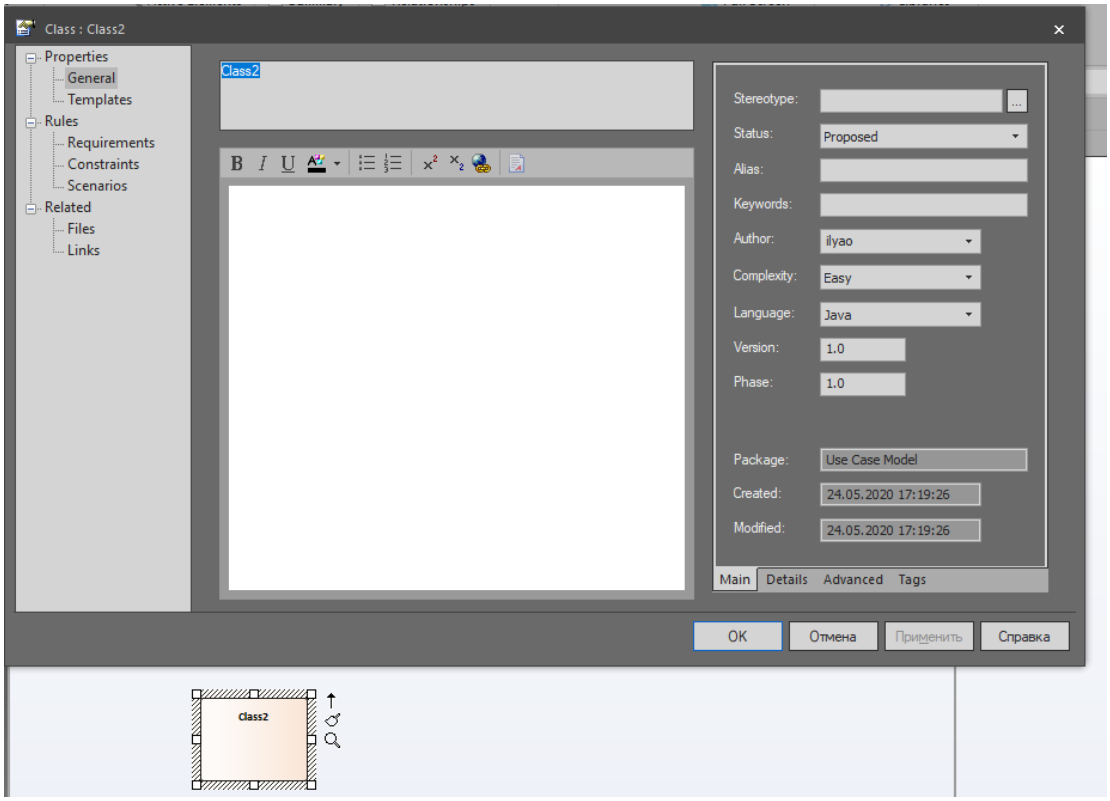

Рисунок 10 – Создание элемента типа Class

2) Для создания классов анализа необходимо установить соответствующий стереотип класса (**«entity»**, **«control»** или **«boundary»**). Для этого на вкладке **Properties**, необходимо найти параметр **«Stereotype»** и нажать на ... в конце строки. В результате откроется окно выбора стереотипа класса.

3) Устанавливаем параметр **«Profile»** на значение **<none>** и выбираем необходимый для нас стереотип (см. Рисунок 11).

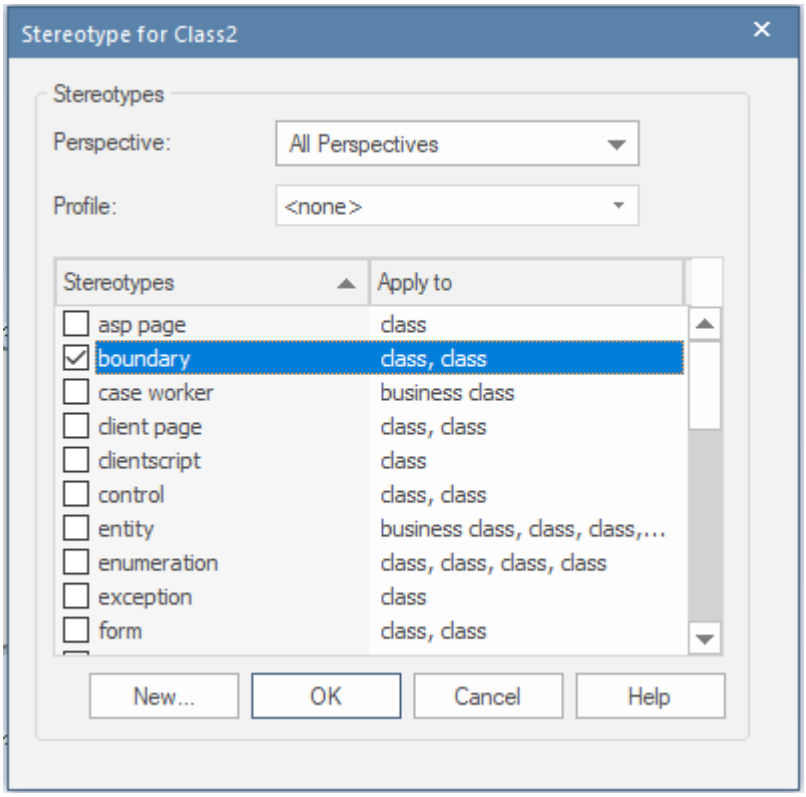

Рисунок 11 – Настройка стереотипа класса

В результате получаем классы необходимых стереотипов (см. Рисунок 12).

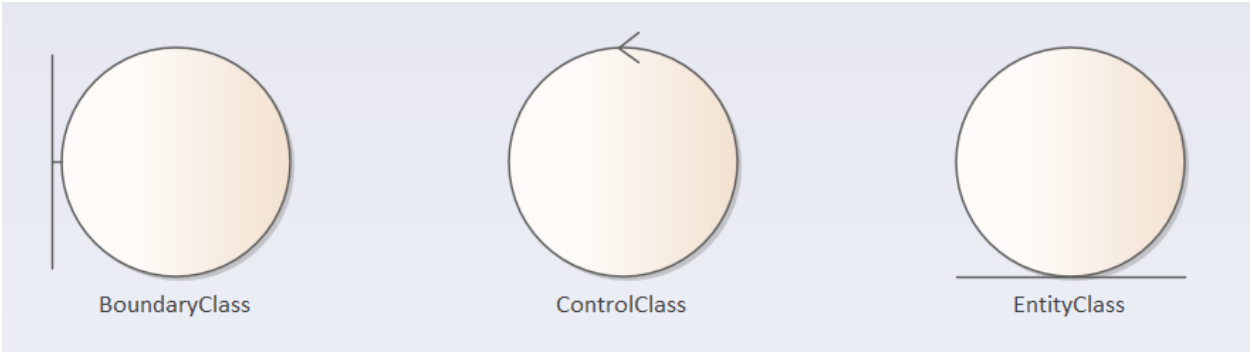

Рисунок 12 – Классы анализа со стереотипами «boundary», «control» и «entity»

4) Для того чтобы изменить вид элементов на классический необходимо отключить параметр **«Use Stereotype Icons»** в разделе **«Elements»** в настройках вашей диаграммы классов (см. Рисунок 13).

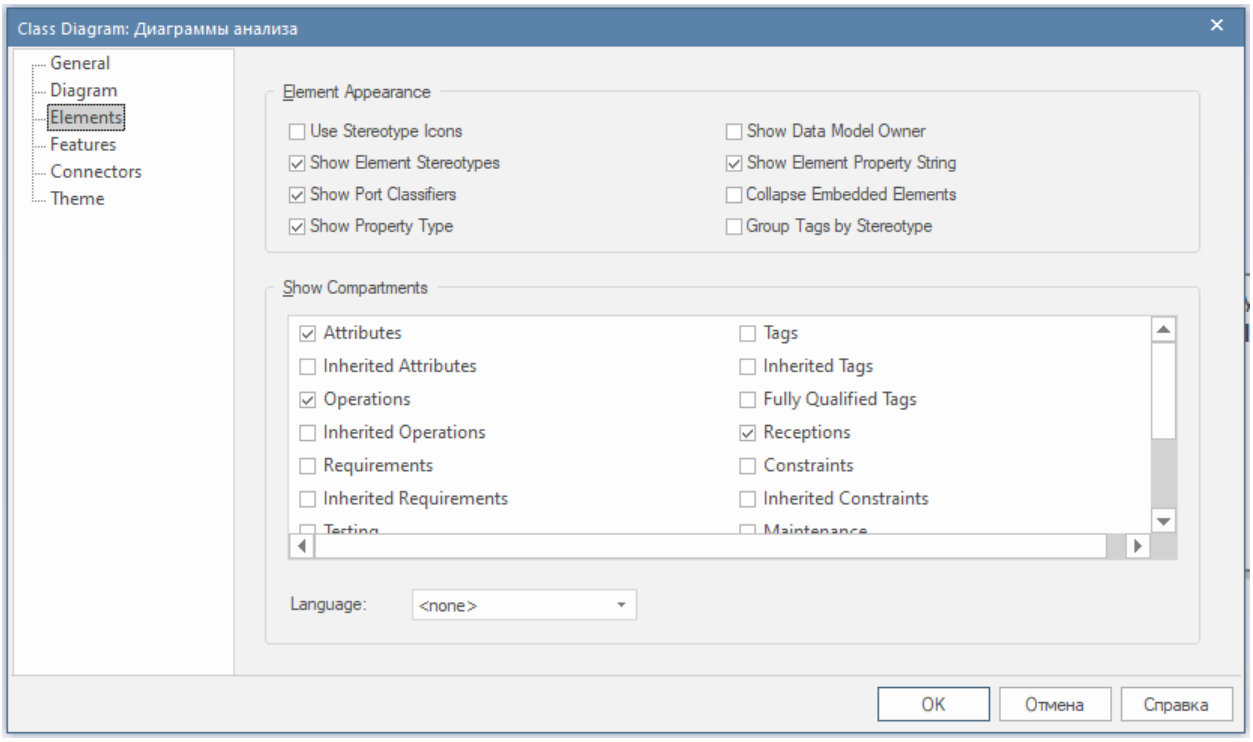

Рисунок 13 – Настройка параметров Диаграммы классов

В итоге получаем привычную нам диаграмму классов с требуемыми для нас стереотипами (см. Рисунок 14).

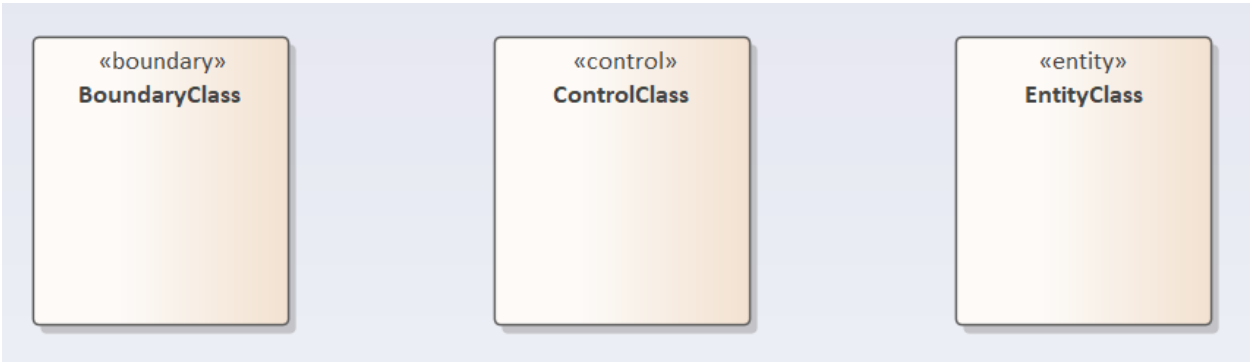

Рисунок 14 – Классический вид классов анализа со стереотипами

После этого зададим методы и атрибуты класса.

Добавление атрибутов и методов

1) Для добавления атрибутов и методов классу необходимо правой кнопкой мыши выбрать класс, открыть в выпадающем меню поле **Features** и выбрать **Attributes** или **Operations** (см. Рисунок 15).

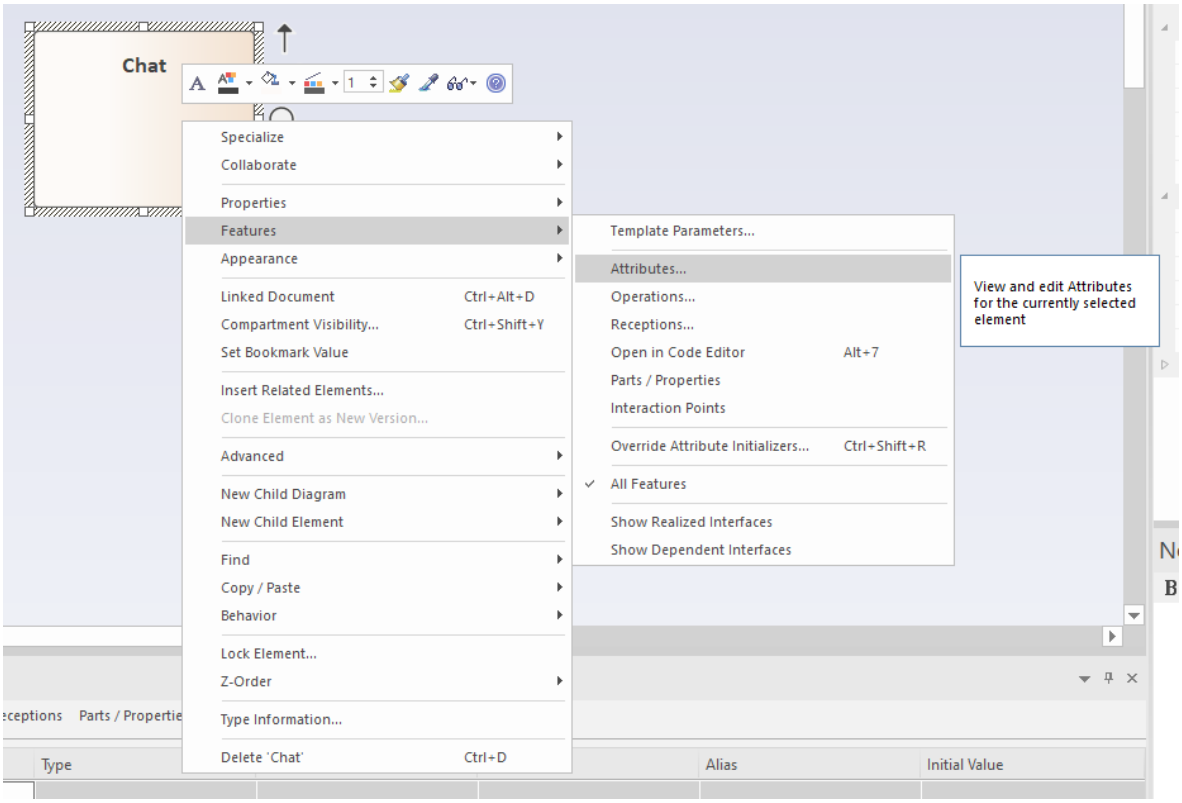

Рисунок 15 – Добавление атрибутов и методов классу

2) В всплывающем диалоговом окне необходимо установить названия атрибутов (см. Рисунок 16) или методов, а также типы возвращаемых параметров или типы атрибутов.

Рисунок 16 – Настройка новых атрибутов класса

3) Для добавления атрибута необходимо указать его имя, тип и идентификатор доступа. При выборе типа можно воспользоваться вариантами из выпадающего списка, которые предлагает EA, если необходимого типа нет среди готовых, его можно ввести с клавиатуры, так в нашем случае не было типа ‘**string’**, и мы его добавили с клавиатуры (см. Рисунок 17).

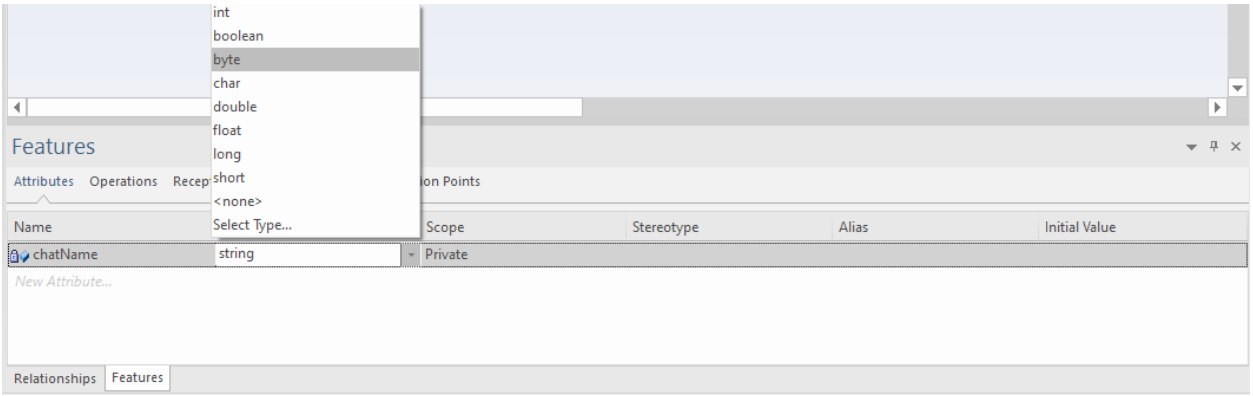

Рисунок 17 – Добавление пользовательского типа атрибута ‘string’

4) Для добавления методов к классу Chat необходимо переключиться на вкладку «**Operations**», в левом верхнем углу вкладки «**Features**».

5) Аналогичным для атрибутов образом необходимо задать имя для метода, принимаемые параметры, возвращаемый тип и идентификатор доступа (см. Рисунок 18).

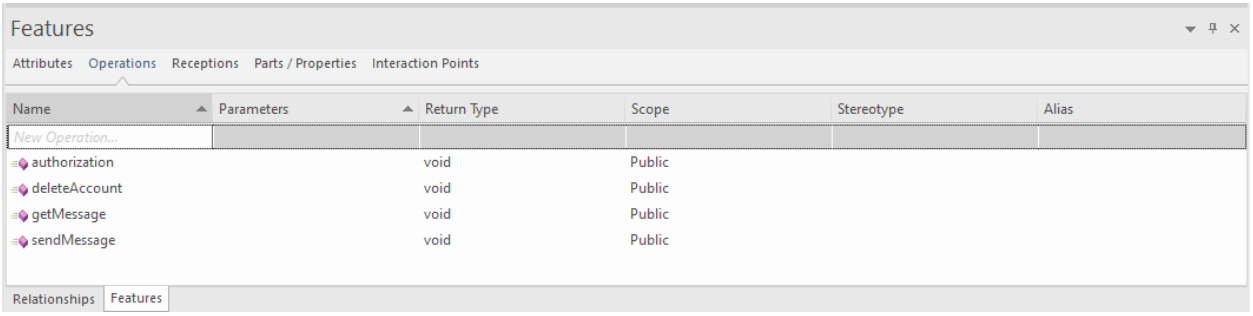

Рисунок 18 – Настройка новых методов класса

6) Повторяя ранее описанные действия создаем описание для классов **ImageMessage** и **Message**.

Добавление связей между классами

Рассмотрим создание связей между классами на примере **ImageMessage** и **Message** (см. Рисунок 19).

Рисунок 19 – Классы ImageMessage и Message

1) Для начала создания связи между классами необходимо выбрать и удерживать левой кнопкой мыши элемент интерфейса программы  (см. Рисунок 20).

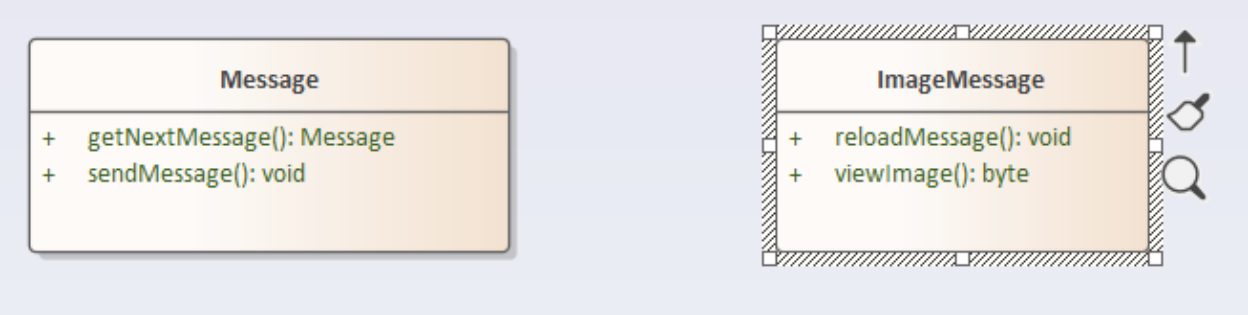

Рисунок 20 – Создание связи между классами

2) Далее необходимо удерживая левую кнопку мыши навестись указателем мыши на класс, с которым мы хотим реализовать связь.

3) После соединения классов появится диалоговое окно, в котором необходимо выбрать тип связи между классами. В нашем случае доступны: ассоциация, направленная ассоциация, агрегация ко всему классу, агрегация к части класса, композиция к целому классу и к части класса, а также наследование, шаблон и реализация (см. Рисунок 21).

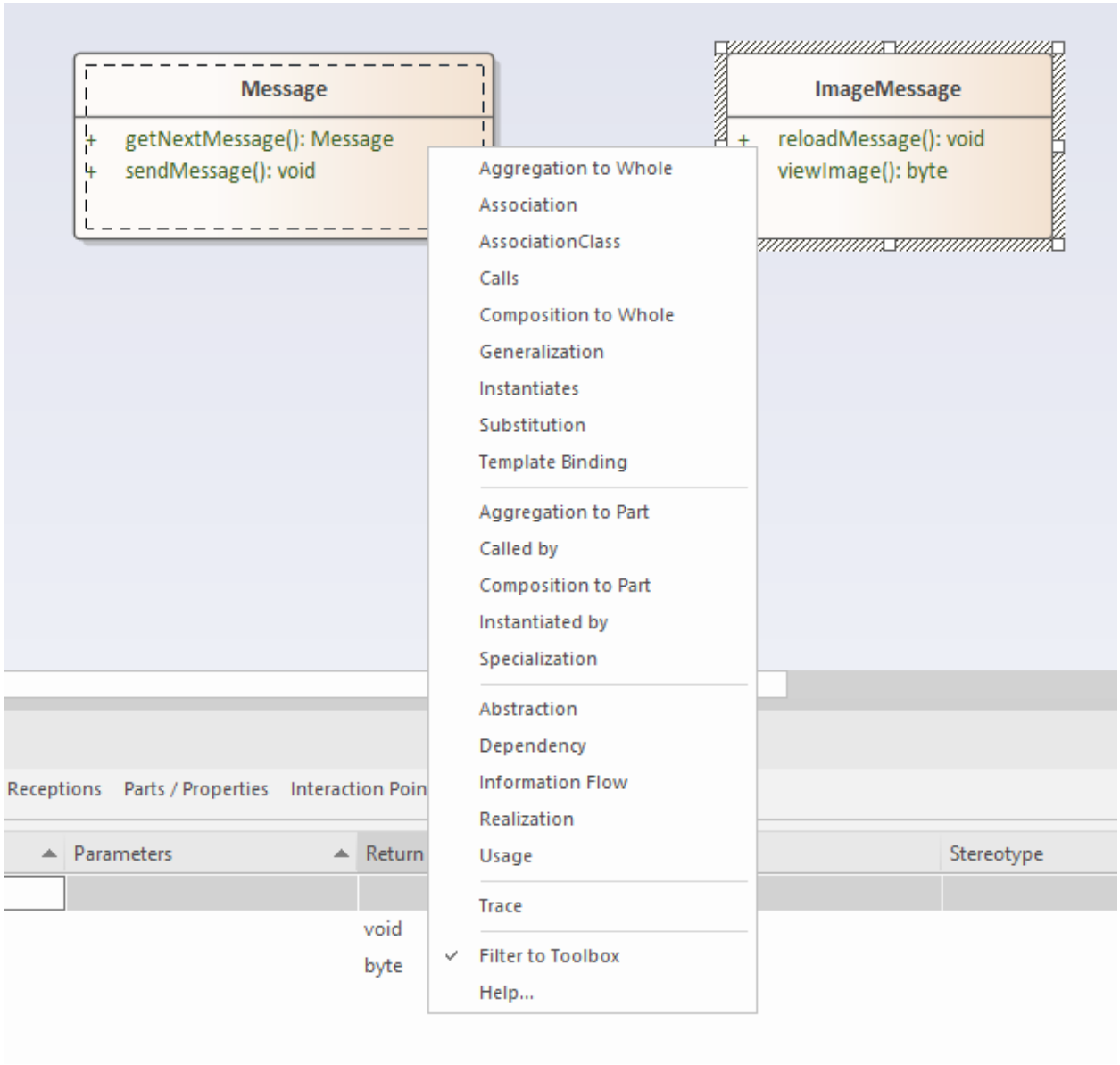

Рисунок 21 – Выбор типа связи

В нашем примере мы рассмотрим **Наследование** (**Generalization**).

4) В появившемся диалоговом окне необходимо выбрать методы и/или параметры класса родителя которые мы хотим унаследовать, в нашем случае это **getNextMessage()** и **sendMessage()** (см. Рисунок 22).

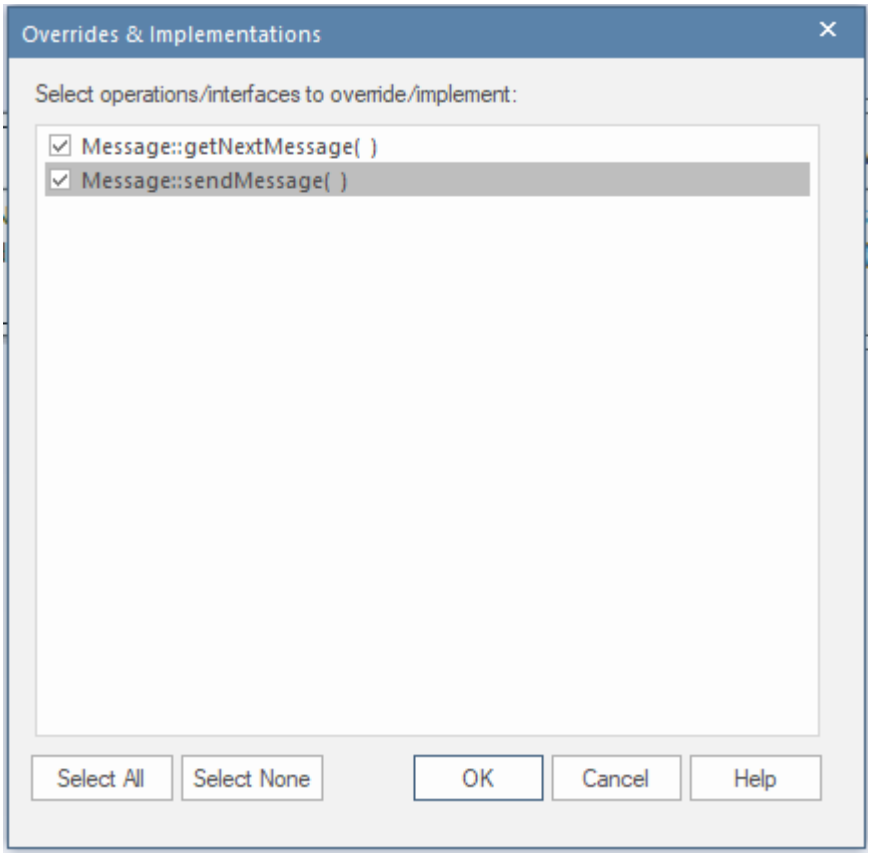

Рисунок 22 – Выбор наследуемых методов классов

Аналогичным образом создаются связи между остальными классами.

2.3.3. Диаграмма последовательностей

Создание Диаграммы последовательностей

1) Выбираем любой **Пакет** и выбираем «**Add diagram…**» или нажимаем соответствующую кнопку в шапке области **Browser**. В результате откроется окно **New Diagram** (Мастер создания диаграмм).

2) Переходим в категорию «**UML Behavioral**» и выбираем тип диаграммы «**Sequence**» (см. Рисунок 23). После этого нажимаем «**Ок**».

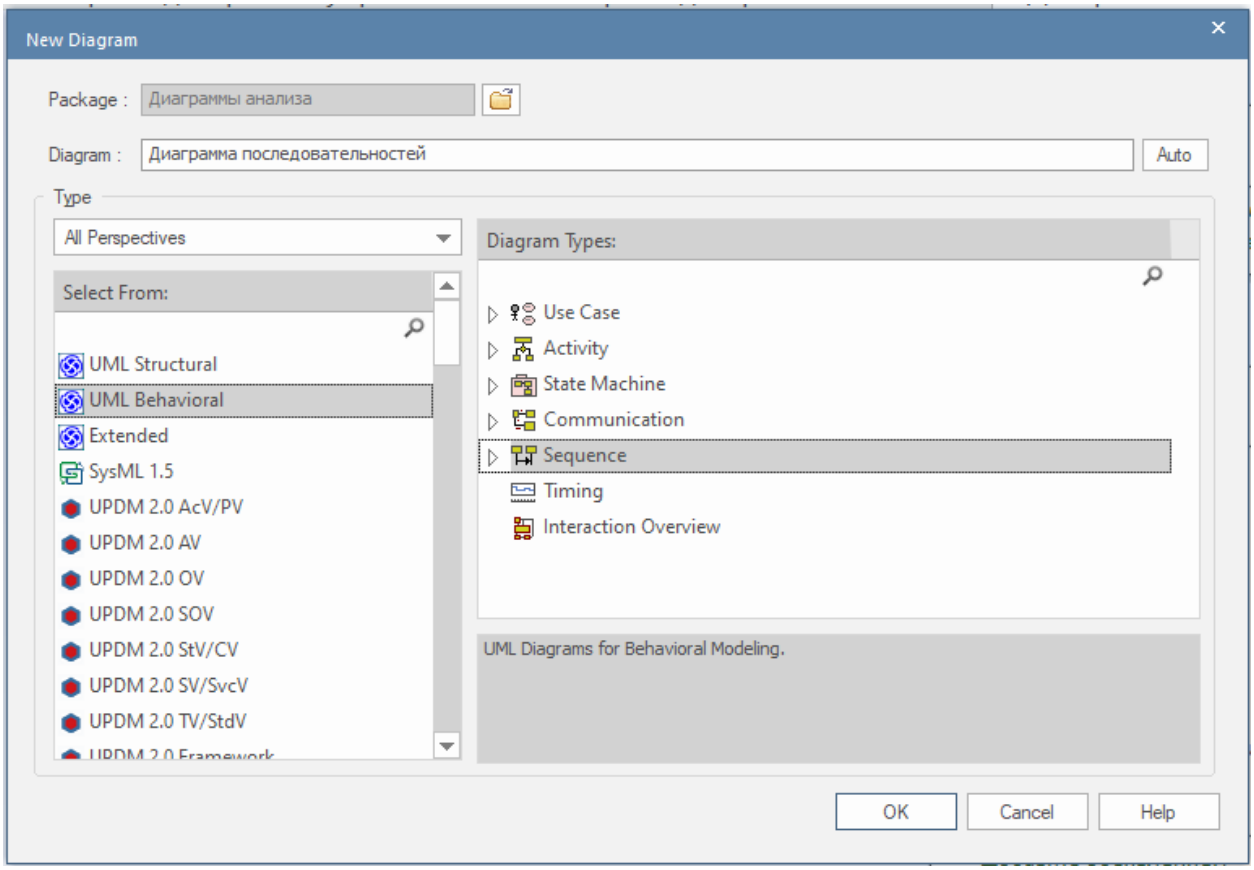

Рисунок 23 – Создание Диаграммы последовательностей

В результате в проект добавится новая Диаграмма последовательностей и сразу откроется в области редактирования диаграмм.

Можно заметить, что в области **Toolbox**, появились элементы, специализированные под данный тип диаграмм (см. Рисунок 24).

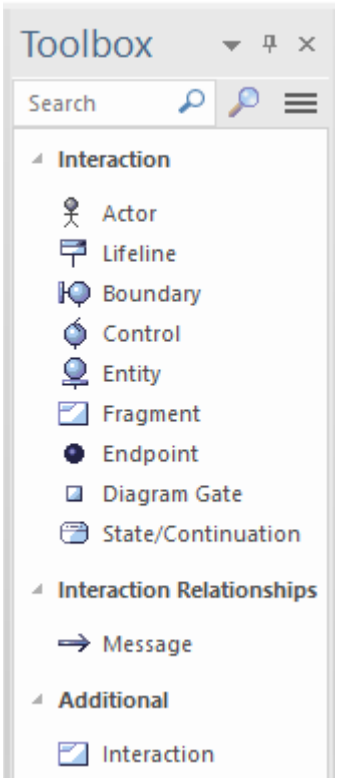

Рисунок 24 – Специализированные элементы области Toolbox для Диаграммы последовательностей

Реализация Диаграммы последовательностей

1) Для реализации диаграммы последовательности необходимо в иерархии проекта переключиться в соответствующую диаграмму (см. Рисунок 25).

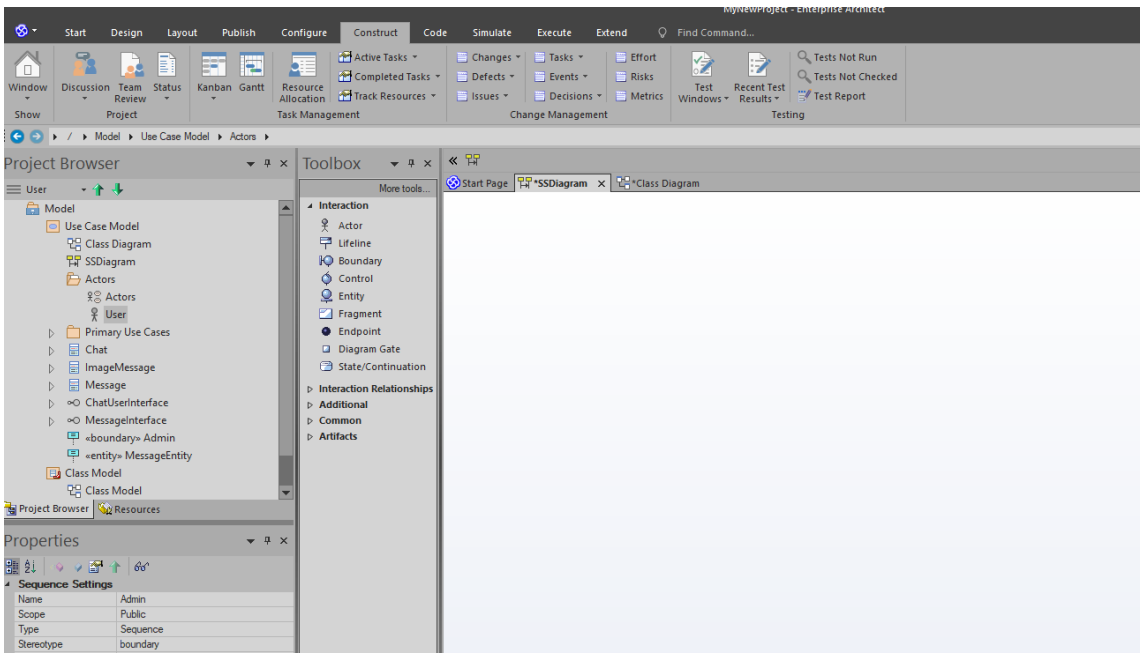

Рисунок 25 – Рабочая область Диаграммы последовательностей

2) После этого из иерархии проекта необходимо выбрать нужного вам актера. Левой кнопкой мыши щелкнуть по нему (в нашем случае это актер **User**) в области «**Browser**» (см. Рисунок 26) и перетащить на рабочую область. Затем необходимо таким же образом перенести ваш класс (в нашем случае это класс **Chat**).

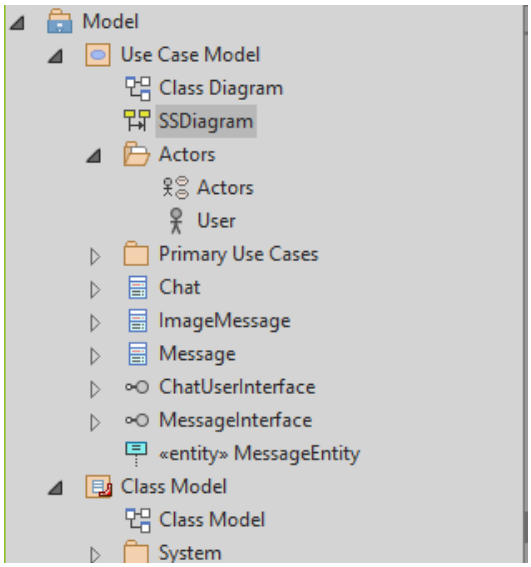

Рисунок 26 – Область Browser с готовыми актерами и классами

3) Также возможно создание нового актера на диаграмме последовательностей. Для этого требуется перенести сущность **Actor** из панели навигации **Toolbox** (см. Рисунок 24) на рабочее пространство диаграммы последовательности, после чего новый актер будет автоматически создан.

Так должна выглядеть диаграмма после перетаскивания на нее актера User и класса Chat (см. Рисунок 27).

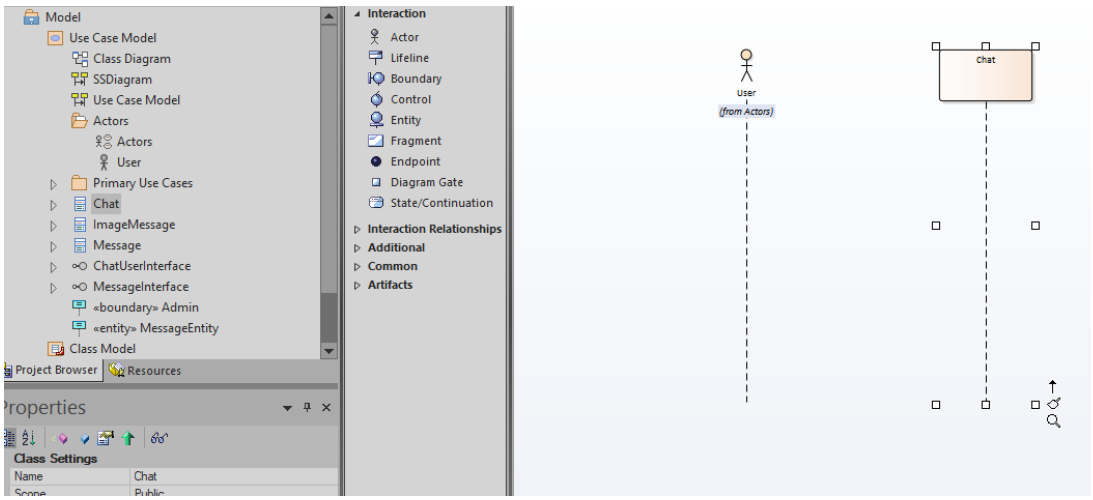

Рисунок 27 – Диаграмма последовательностей с актером и классом

4) После этого из **Toolbox** (см. Рисунок 24) добавим элементы: **Boundary** и **Entity**. После каждого перетаскивания на рабочую область необходимо задать название для элемента в диалоговом окне (см. Рисунок 28).

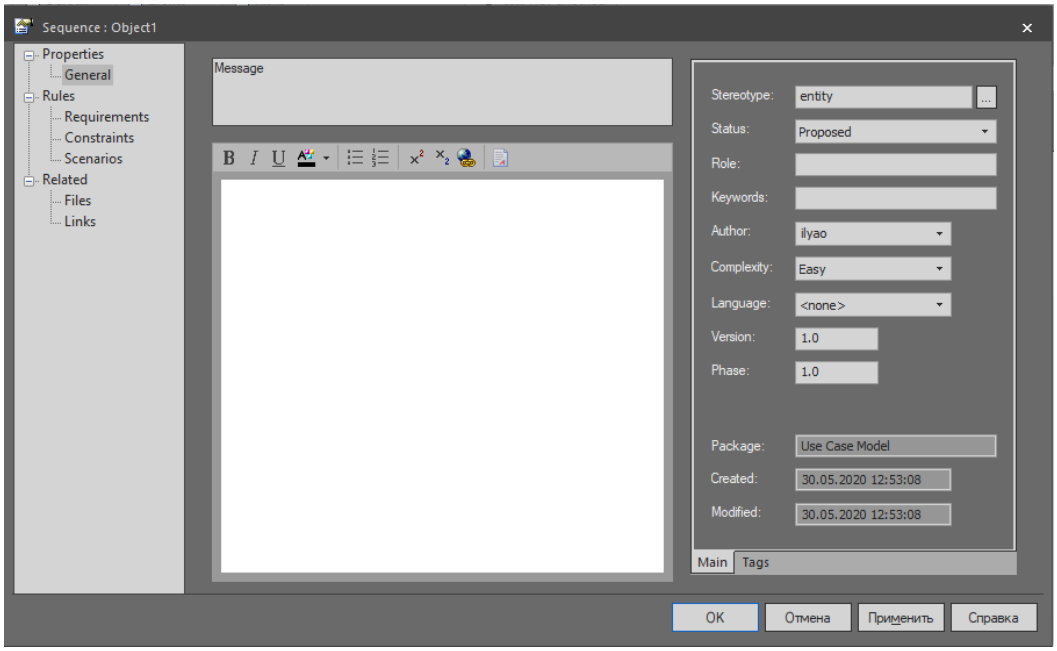

Рисунок 28 – Диалоговое окно настройки классов на Диаграмме последовательностей

Также есть возможность и существующие классы со стереотипами переместить из области **Browser** на рабочую область!

После перетаскивания всех элементов у нас получается следующая диаграмма (см. Рисунок 29).

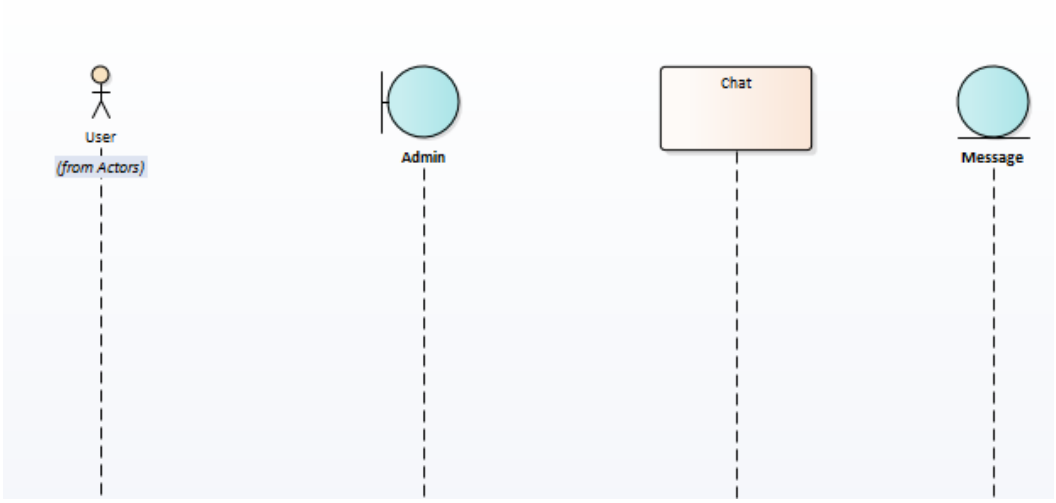

Рисунок 29 – Диаграммы последовательностей со всеми актерами и классами

Создание связей на диаграмме последовательностей:

1) Для создания связей на диаграмме последовательностей необходимо, зажав следующий элемент программного интерфейса (см. Рисунок 30), соединить одну линию жизни с другой (см. Рисунок 31).

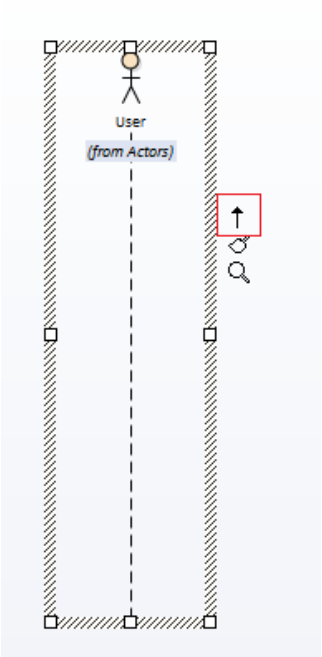

Рисунок 30 – Элемент интерфейса для создания связи на Диаграмме последовательностей

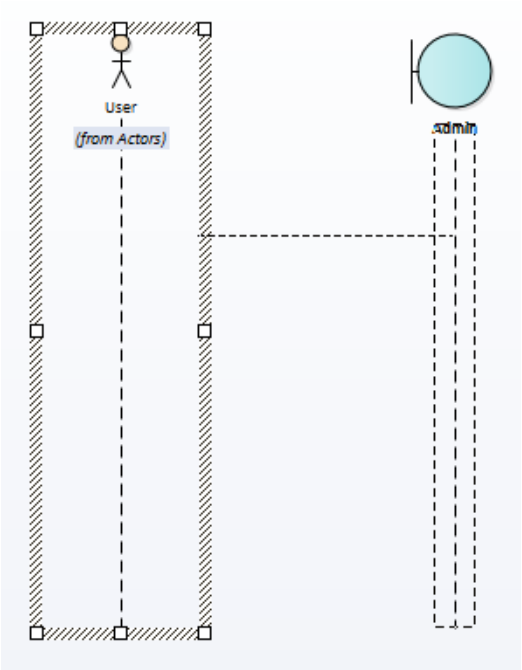

Рисунок 31 – Создание связи на Диаграмме последовательностей

2) В появившемся диалоговом окне (или во вкладке **Properties**) необходимо в поле **Message** задать сообщение, введя его с клавиатуры (см. Рисунок 32). В этом случае, сообщение автоматически становится методом объекта.

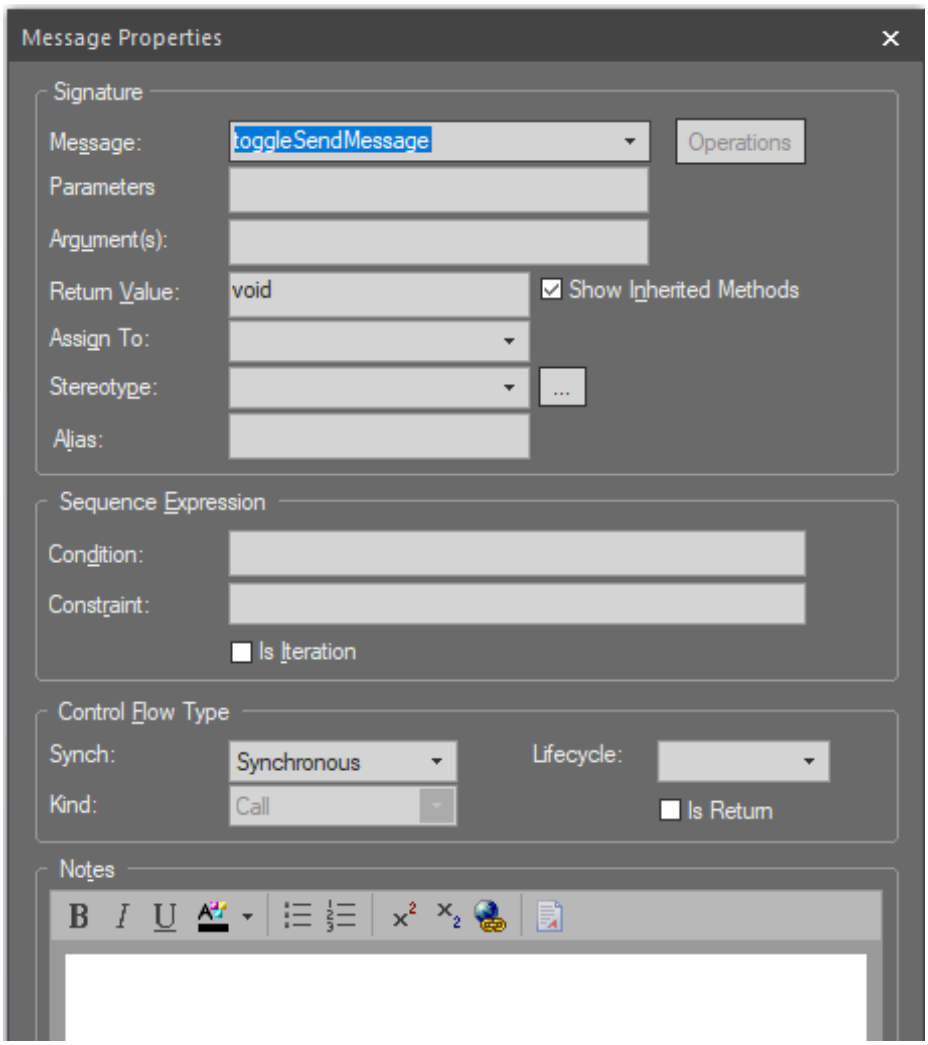

Рисунок 32 – Настройка связи на диаграмме последовательностей

3) Также возможно выбрать сообщение из выпадающего меню (сообщения являются методами в реализации объекта) (см. Рисунок 33).

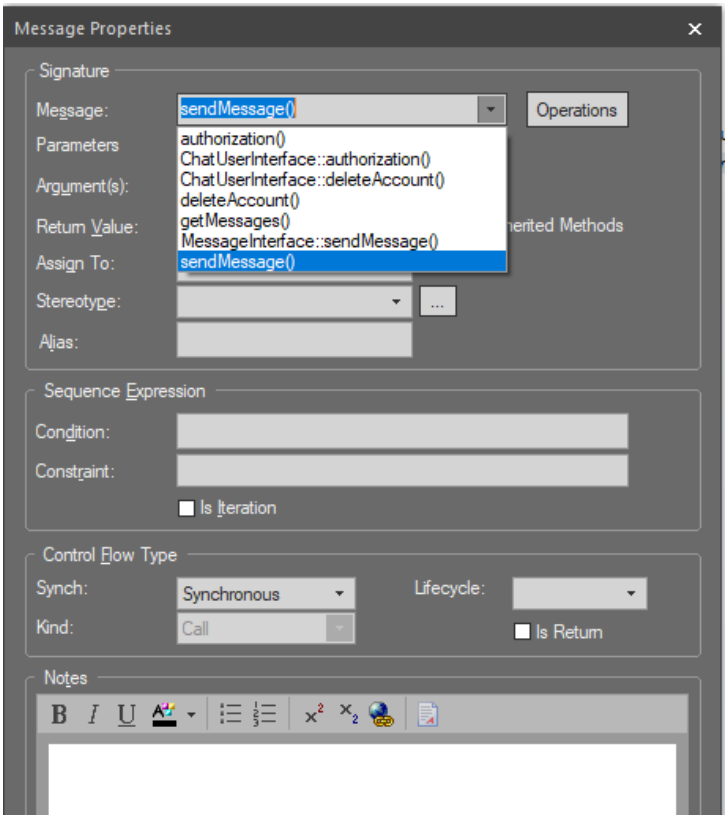

Рисунок 33 – Выбор сообщения из методов класса

4) После всех операций, описанных выше, ваша диаграмма последовательностей может выглядеть следующим образом (см. Рисунок 34).

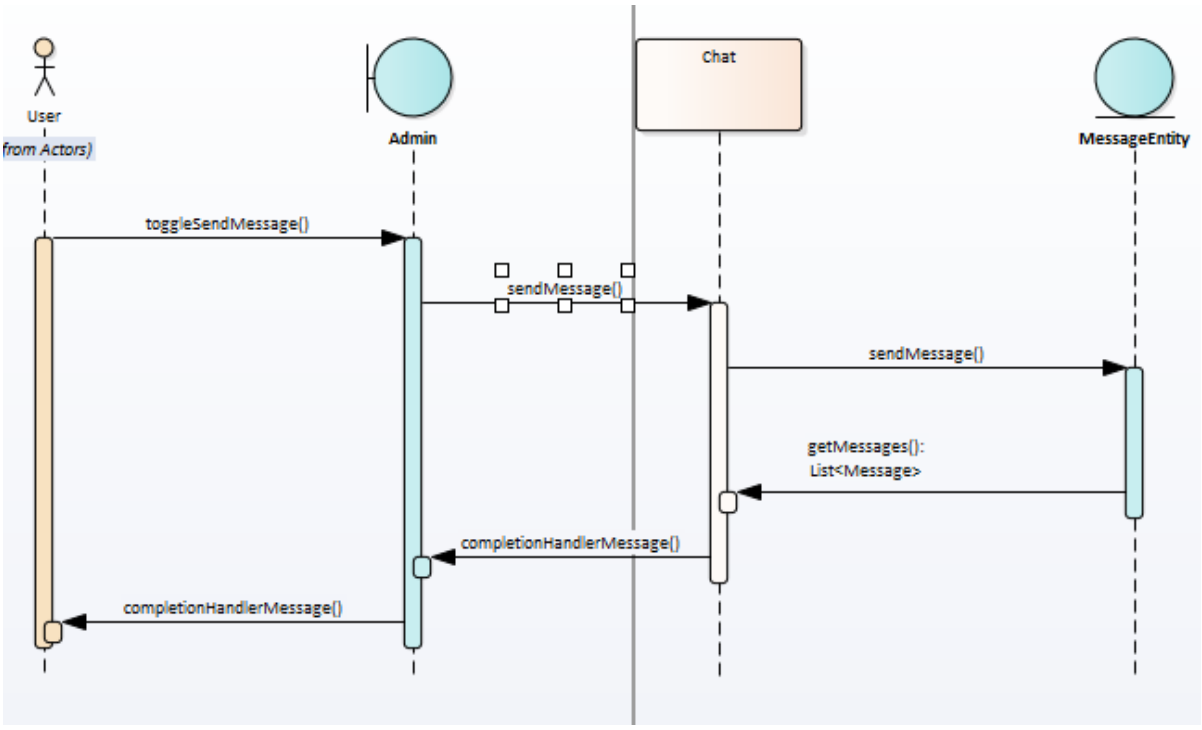

Рисунок 34 – Готовая Диаграмма последовательностей

2.3.4. Диаграмма пакетов

Создание Диаграммы пакетов

3) Выбираем любой **Пакет** и выбираем «**Add diagram…**» или нажимаем соответствующую кнопку в шапке области **Browser**. В результате откроется окно **New Diagram** (Мастер создания диаграмм).

4) Переходим в категорию «**UML Structural**» и выбираем тип диаграммы «**Package**» (см. Рисунок 35). После этого нажимаем «**Ок**».

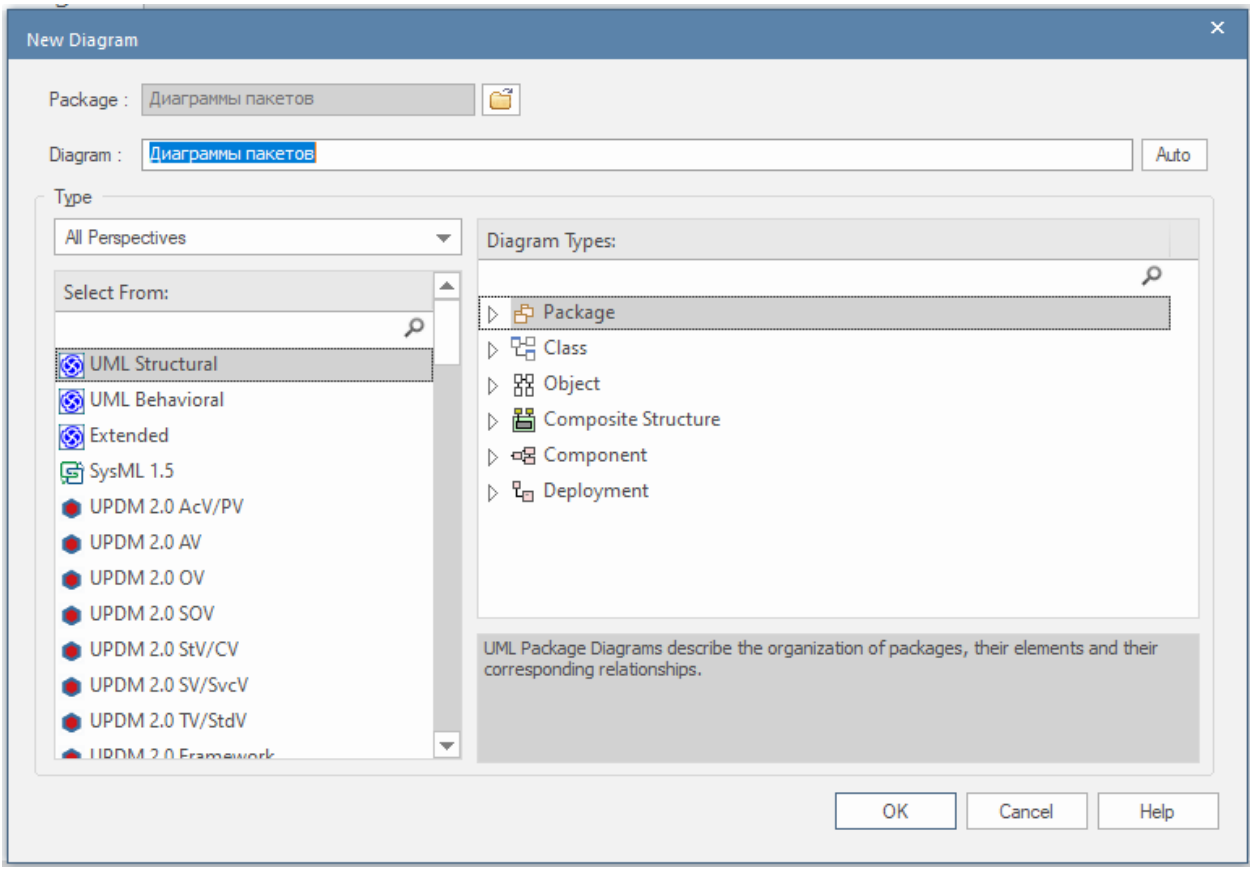

Рисунок 35 – Создание Диаграммы пакетов

В результате в проект добавится новая Диаграмма пакетов и сразу откроется в области редактирования диаграмм.

Можно заметить, что в области **Toolbox**, появились элементы, специализированные под данный тип диаграмм (см. Рисунок 36).

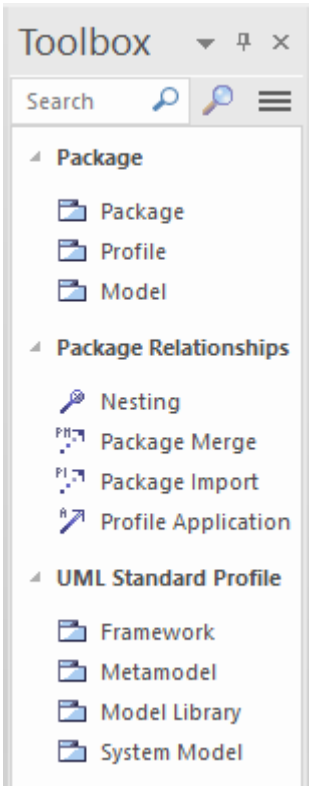

Рисунок 36 – Специализированные элементы области Toolbox для Диаграммы пакетов

Создание пакета

1) Для создания нового пакета необходимо в **Toolbox** диаграммы выбрать элемент **Package** (см. Рисунок 36) и перетащить его на рабочую зону.

В результате у нас на рабочей области появится стандартный пакет с описанием по умолчанию (см. Рисунок 37).

Рисунок 37 – Стандартный пакет на Диаграмме пакетов

2) Также можно переместить на диаграмму пакетов уже существующий пакет из области **Browser** (см. Рисунок 38).

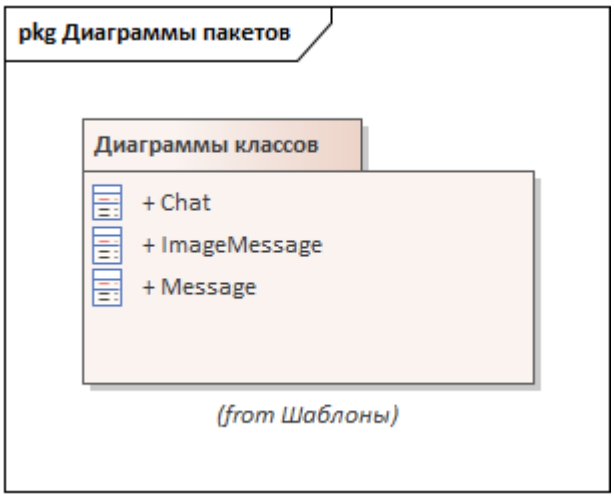

Рисунок 38 – Ранее созданный пакет на Диаграмме пакетов

3) Для того чтобы убрать содержимое пакета необходимо отключить параметр **«Package Contents»** в разделе **«Elements»** в настройках вашей диаграммы пакетов (см. Рисунок 39).

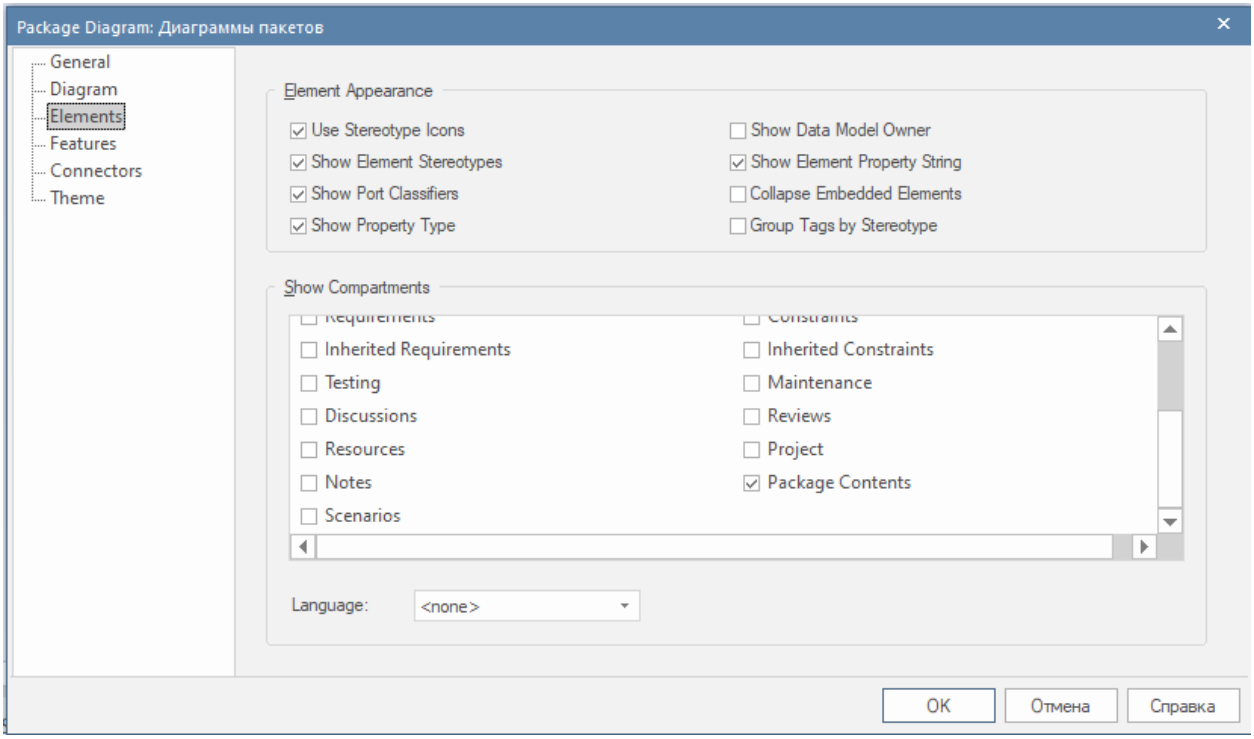

Рисунок 39 – Отключение содержимого пакетов на Диаграмме пакетов

4) Для того чтобы скрыть источник пакета необходимо отключить параметр **«Show Namespace»** в разделе **«Diagram»** в настройках вашей диаграммы пакетов (см. Рисунок 40).

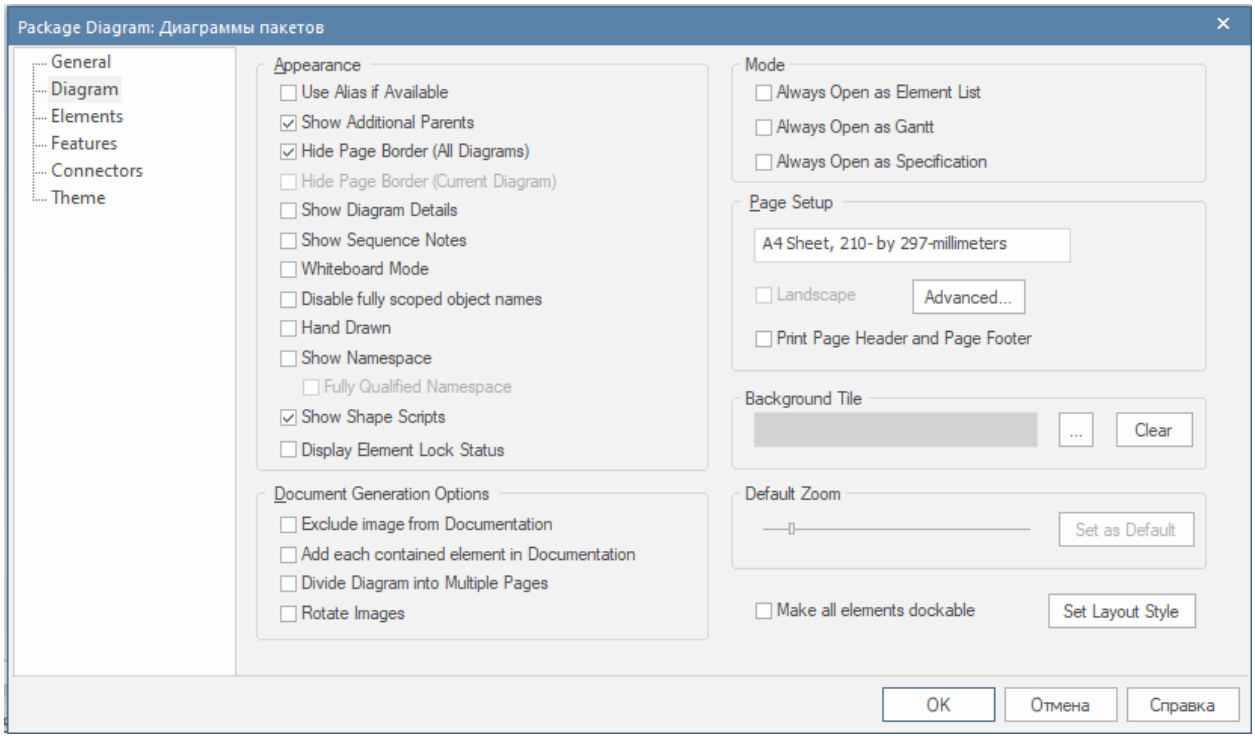

Рисунок 40 – Отключение видимости источников на Диаграмме пакетов

На диаграмме пакетов помимо пакетов могут быть изображены классы и связи между ними.

2.4. Рабочий процесс «Анализ требований»

После того, как мы рассмотрели новые возможности инструмента для построения диаграмм и построили модель требований, необходимо провести анализ требований.

На стадии анализа необходимо выполнить следующие действия:

- анализ архитектуры системы;
- анализ прецедентов;
- анализ классов;
- анализ пакетов.

В результате выполнения анализа требований будут получены следующие артефакты:

- **модель анализа** – содержит систему анализа из множества пакетов, классов и кооперации анализа;
- **классы анализа** – представляют начальное разбиение системы на классы;
- **анализ кооперации** – показывает реализацию прецедентов в терминах классов и связей между ними;
- **пакеты анализа** — логические контейнеры классов анализа;
- **сервисные пакеты** – содержит классы для выполнения сервисных функций.

Далее действия и артефакта анализа рассмотрены подробно.

Процесс анализа требований позволяет:

- уточнить спецификацию требований;
- перейти от языка заказчика к языку разработчика;
- повысить формализм описаний;
- провести анализ внутренних механизмов работы системы;
- структурировать требования для дальнейшей разработки ПО.

2.4.1. Модель анализа

Модель анализа создается на основе модели требований (ее прецедентов).

Прецедент – это некоторая функция, выполняемая системой в интересах ее пользователя. Для реализации прецедента необходим программный компонент, выполняющий данную функцию.

Реализация прецедента называется **кооперацией** и имеет две составляющие: статическую и динамическую. Статическая составляющая задается диаграммой классов, необходимых для выполнения прецедента. Динамическая составляющая определяет алгоритм выполнения прецедента и задается диаграммами взаимодействия.

Модель анализа составляет архитектор, классы и пакеты разрабатывает инженер по компонентам, кооперации анализирует инженер по прецедентам.

2.4.2. Анализ архитектуры

2.4.2.1. Определение кооперaций

В начале анализа следует добавить кооперации для реализации основных прецедентов. Для каждого прецедента одну кооперацию. Название кооперации совпадает с названием прецедента. Составляем «обзорную» диаграмму кооперaций, содержащую прецеденты и реализующие их кооперации, связь между ними типа «реализации» со стереотипом трассировки (отображения) **«trace»**.

Трассировка (Trace) — это специализация зависимости, связывающая элементы модели или наборы элементов, представляющих одну и ту же идею в разных моделях. Трассировки часто используются для отслеживания требований и изменений модели. Поскольку изменения могут происходить в обоих направлениях, порядок этой зависимости обычно игнорируется.

Пример:

В результате определения кооперации АСУ «Веб-портал поиска пропавших животных» была сформирована обзорная диаграмма кооперации и прецедентов (см. Рисунок 41).

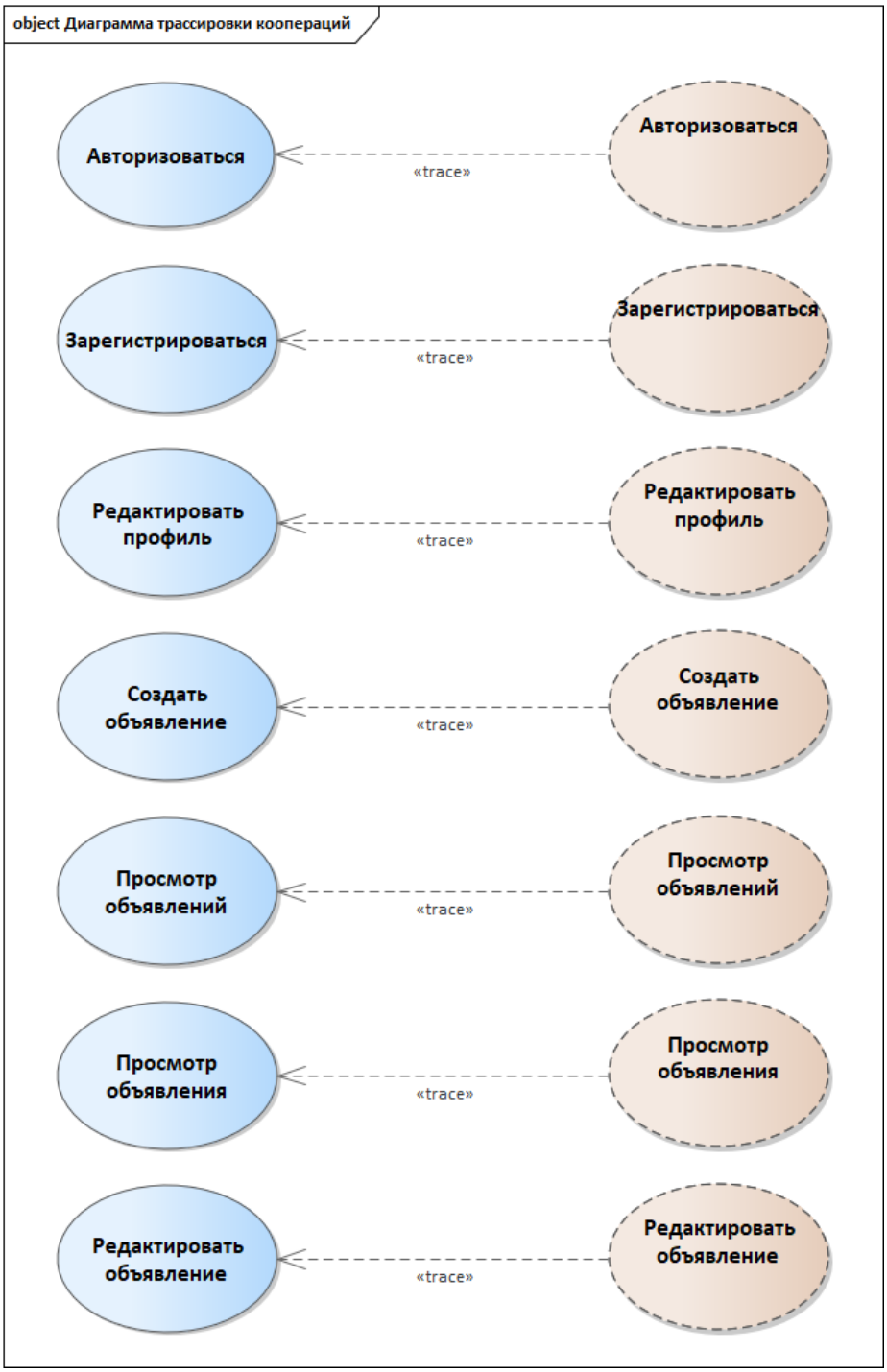

Рисунок 41 – Обзорная диаграмма кооперации и прецедентов для модели АСУ «Веб-портал для поиска пропавших животных»

2.4.2.2. Определение классов анализа

Классы анализа бывают трех типов: **граничный**, **управляющий** или **сущности**.

Классы анализа могут участвовать в нескольких прецедентах. Поэтому в начале анализа прецедентов следует выделить все классы анализа, которые могут быть выделены, а затем определить те из них, которые задействованы в конкретных прецедентах.

Класс анализа является концептуальной единицей, не привязан к конкретному языку программирования, и при его определении следует учитывать, что класс анализа:

- ориентирован на функциональные требования;
- имеет текстовое описание ответственности (перечень выполняемых функций) без сигнатур;
- содержит атрибуты логических типов.

При этом отношения между классами анализа концептуальны.

2.4.2.3. Определение классов сущностей

Классы сущностей показывают логическую структуру данных. Они используются для моделирования долгоживущей, часто сохраняемой информации о человеке, объекте или событии реального мира и могут иметь сложное поведение. Классы сущностей определяют на основе модели предметной области или бизнес-модели. Выделяют те классы, которые участвуют в прецедентах. Обычно около 10-20 штук.

Затем необходимо построить «обзорную» диаграмму классов сущностей (Domain Model) на основе модели предметной области. Определить для них атрибуты, связи ассоциации, роли, множественность и арность. Классы сущностей имеют стереотип **«entity»**.

Пример:

В результате определения классов сущностей АСУ «Веб-портал поиска пропавших животных» была сформирована обзорная диаграмма классов сущностей (см. Рисунок 42).

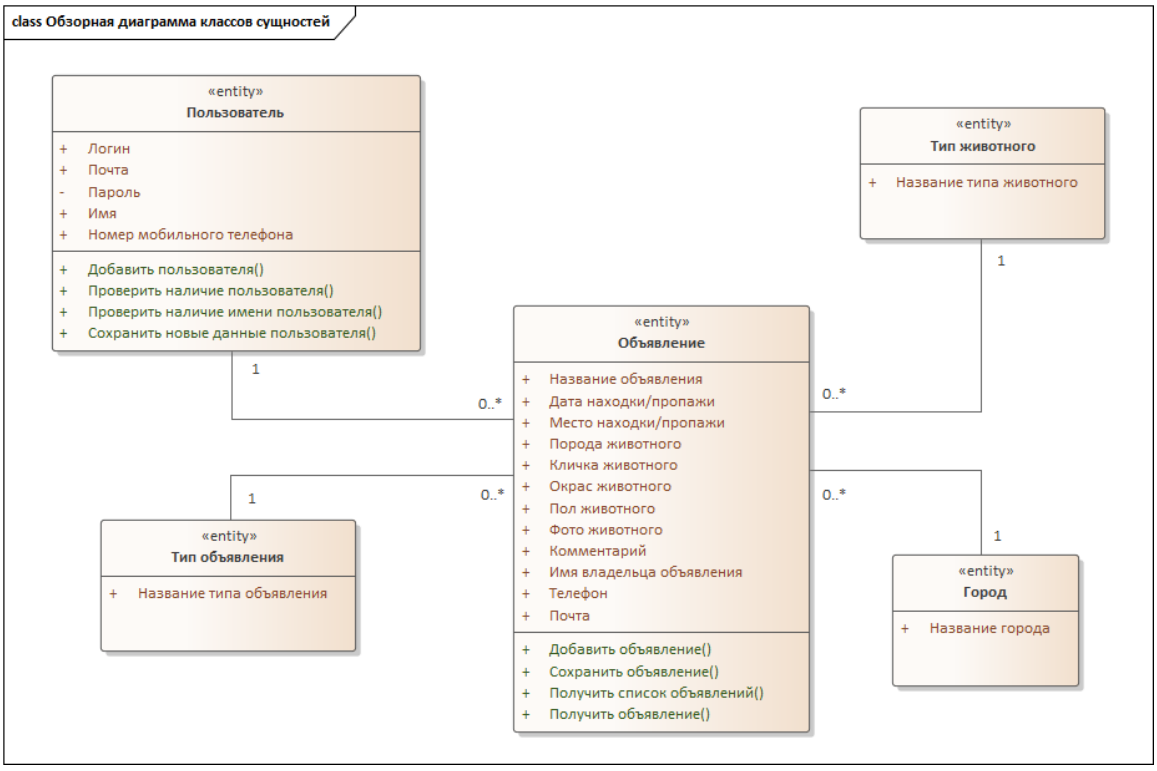

Рисунок 42 – Обзорная диаграмма классов сущностей для модели АСУ «Веб-портал для поиска пропавших животных»

2.4.2.4. Определение граничных классов

Граничный класс – выделяют для моделирования взаимодействия между системой и актерами: получение и передача информации или запроса. Граничный класс соответствует пользовательскому интерфейсу, интерфейсу внешних устройств или коммуникационному протоколу, библиотеке API на высоком уровне абстракции без физической реализации. Граничный класс обозначается стереотипом **«boundary»**.

Принципы выделения граничных классов:

- определить по одному основному граничному классу на каждого актера – человека. Это будет прототипом главной формы его пользовательского интерфейса. Основной граничный класс может быть набором простых граничных классов, соответствующих формам его пользовательского интерфейса;
- определить по одному основному граничному классу на каждого актера – внешнюю систему. Это будет коммуникационным интерфейсом. Если несколько уровней коммуникации, то выделяют по одному граничному классу на каждый уровень.

Пример:

В результате определения граничных классов АСУ «Веб-портал поиска пропавших животных» была сформирована обзорная диаграмма граничных классов (User Interface (UI), англ.) (см. Рисунок 43).

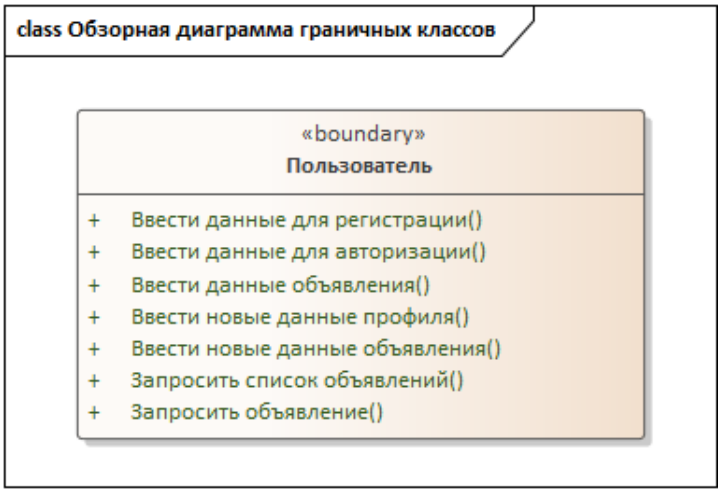

Рисунок 43 – Обзорная диаграмма граничных классов (UI) для модели АСУ «Веб-портал для поиска пропавших животных»

2.4.2.5. Определение управляющих классов

Управляющий класс – выполняет координацию, последовательность, взаимодействие и управление другими объектами для прецедента. Обрабатывает и координирует действия и потоки управления, реализует бизнес-логику. Управляющий класс обозначается стереотипом **«control»**.

Принципы выделения управляющих классов (для нескольких или всех прецедентов):

- определить по одному управляющему классу на каждый прецедент (его кооперацию) для реализации бизнес-логики. В отдельных случаях может быть управляющий класс, включенный в граничный для очень простой бизнес-логики, например, добавления, удаления или просмотра записей. Или два и более управляющих классов для сложного алгоритма управления.

Пример:

В результате определения управляющих классов АСУ «Веб-портал поиска пропавших животных» была сформирована обзорная диаграмма управляющих классов (см. Рисунок 44).

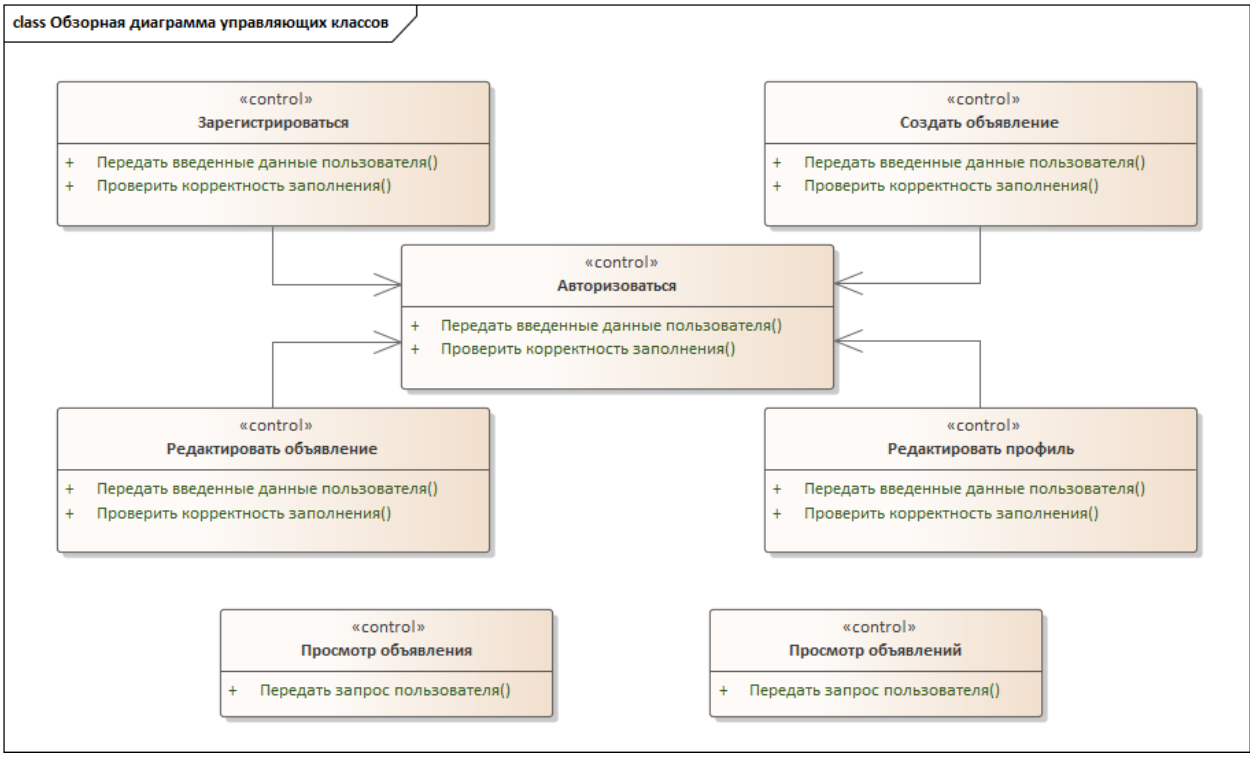

Рисунок 44 – Обзорная диаграмма управляющих классов для модели АСУ «Веб-портал для поиска пропавших животных»

2.4.3. Анализ кооперацій

После анализа архитектуры системы в целом переходят к анализу отдельных прецедентов. При анализе кооперацій необходимо определить те классы анализа, которые участвуют в ее прецедентах, и связи между этими классами, обеспечивающие выполнение конкретных прецедентов. Для каждой кооперации:

- создать диаграмму классов кооперации и переместить на нее классы (граничные, управляющий и сущностей), которые участвуют в этой кооперации.
- определить ассоциации между классами в соответствии с их взаимодействием для реализации прецедента.

2.4.3.1. Определить классы участники кооперацій

Диаграмма классов – участников кооперации содержит классы, участвующие в кооперации. Набор конкретных классов выявляется на основе спецификации прецедента (для данной кооперации). Обычно это:

- граничный класс, соответствующий актеру (актерам), которые вызывают данный прецедент (кооперацию),
- управляющий класс, соответствующий данной кооперации,
- классы сущностей, используемые в данной кооперации.

Возможно добавление других управляющих классов (сервисов), если для данной кооперации ее прецедент включает другой прецедент («include»). Классы, участвующие в кооперации, выбираются из определенных ранее классов анализа.

Классы – участники соединяются направленными ассоциациями. Направление ассоциаций соответствует вызовам между классами:

- от актера к его граничному классу (ввод и вывод через пользовательский интерфейс),
- от граничного – к управляющему (передача вызова функции),
- от управляющего – к сущностям (обращение к хранилищу данных),
- от управляющего – к другому управляющему (если есть вызов сервиса).

Пример:

В результате определения классов участников кооперации «Создать объявление» АСУ «Веб-портал поиска пропавших животных» была сформирована Диаграмма классов – участников кооперации «Создать объявление» (см. Рисунок 45).

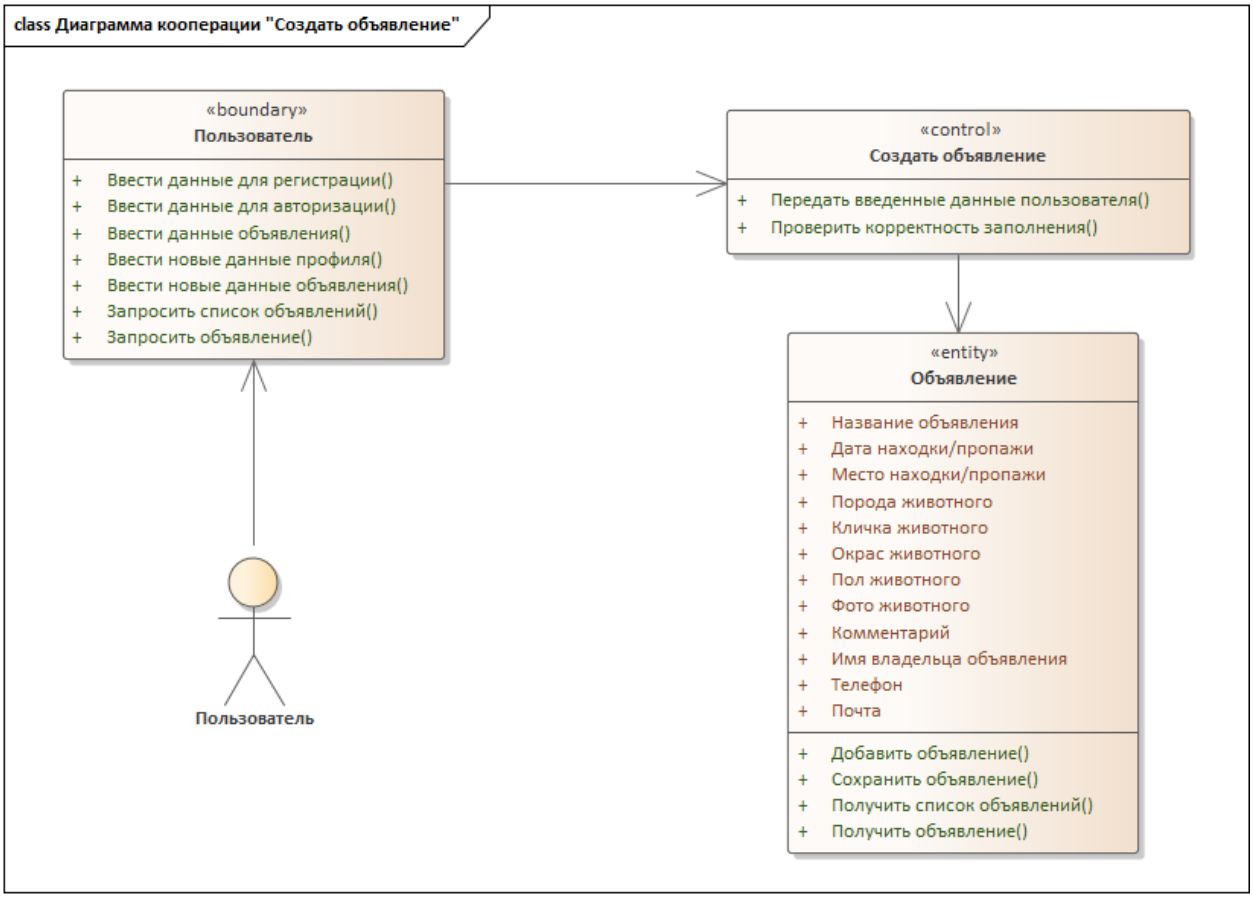

Рисунок 45 – Диаграмма классов – участников кооперации «Создать объявление» для модели АСУ «Веб-портал для поиска пропавших животных»

2.4.3.2. Диаграмма последовательностей

После определения классов – участников кооперации следует создать **диаграмму последовательностей** (**Sequence Diagram**) кооперации. Она создается для описания основного потока алгоритма. Если в алгоритме имеются альтернативные потоки и они тривиальны, то их можно включить в основную диаграмму. В противном случае для альтернативных потоков создают отдельные диаграммы.

На диаграмму последовательностей помещают актера, инициирующего кооперацию, и объекты классов, участвующих в кооперации, и добавляют сообщения, которыми обмениваются объекты при реализации прецедента. Линии и направления сообщений соответствуют ассоциациям на диаграмме классов – участников. Последовательность сообщений определяется порядком взаимодействия объектов. Начинается кооперация всегда сообщением от актера – инициатора. Сообщения указываются без сигнатур, только названием ответственности.

Сообщения между участниками соответствуют вызовам функций классов в процессе выполнения кооперации. Первое сообщение всегда от актера. События диаграммы будут прообразом методов классов.

Все сообщения, которые являются входящими для класса анализа, будут автоматически трансформированы в его методы. Классы анализа пополнятся методами – сообщениями.

Пример:

В результате определения взаимодействия между классами – участниками кооперации «Создать объявление» АСУ «Веб-портал поиска пропавших животных» была сформирована Диаграмма последовательностей «Создать объявление» (см. Рисунок 46).

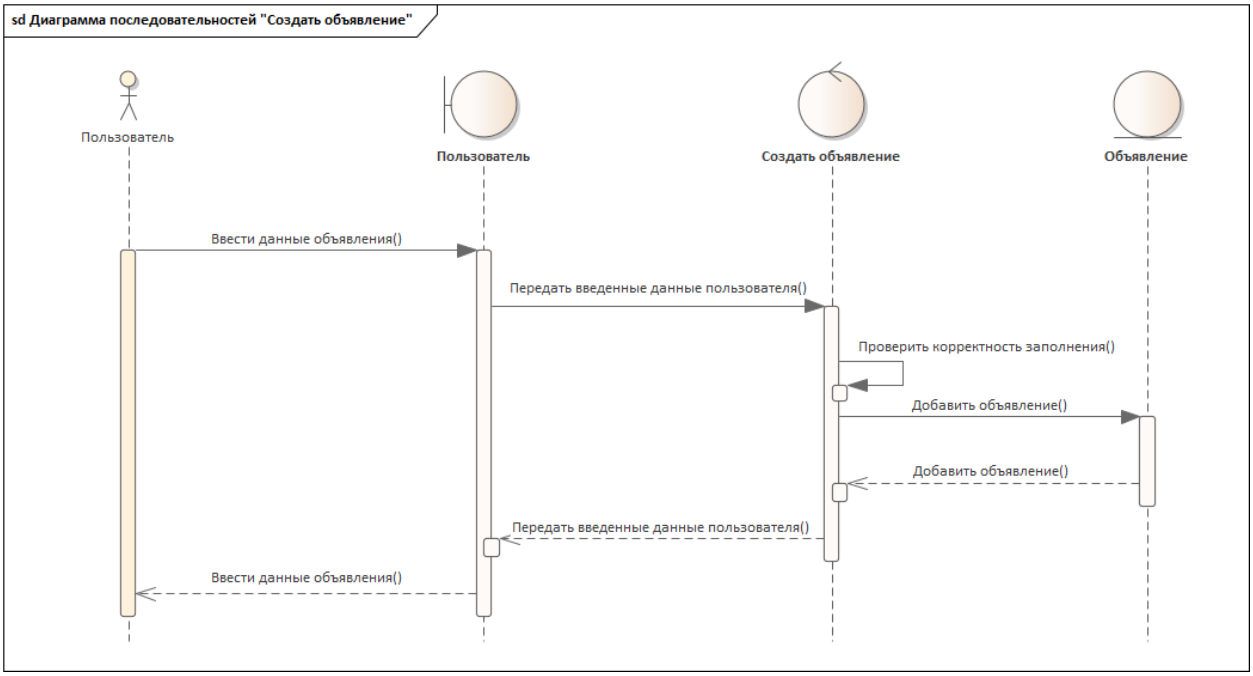

Рисунок 46 – Диаграмма последовательностей кооперации «Создать объявление» для модели АСУ «Веб-портал для поиска пропавших животных»

2.4.4. Анализ классов

После анализа прецедентов (их кооперaций) можно переходить к **анализу классов**. При анализе классов определяют и уточняют их атрибуты, ответственности и связи.

По итогам анализа архитектуры и кооперaций имеется множество классов. Для классов определены их атрибуты и ответственности.

Ответственностью класса анализа является его функциональное назначение, которое определяется на основе совокупности входящих событий на основе всех диаграмм последовательностей, в которых участвует класс (он может участвовать во многих кооперациях). Ответственность — это прообраз будущего метода класса. В отличие от метода она не имеет сигнатуры и не привязана к языку программирования.

Атрибуты определены для классов сущностей и на основе связей ассоциаций. Атрибуты класса должны обеспечивать выполнение его ответственностей.

Для проверки корректности и полноты свойств классов используют следующие принципы:

- для класса сущности: атрибуты — поля хранимых данных, ответственности — get|set или CRUD;
- для класса граничного: атрибуты — элементы формы (на основе эскиза пользовательского интерфейса) или параметры протокола, ответственности — управляющие сигналы от пользователя (обработчики событий, прежде всего кнопок);
- для класса управляющего: атрибуты — промежуточные данные алгоритма, ответственности — шаги алгоритма кооперации.

Также при анализе классов возможно их преобразование, например, разбиение или слияние, выделение базовых и производных классов.

После анализа классов следует внести уточнения во все построенные диаграммы, если они не были скорректированы автоматически.

2.4.5. Анализ пакетов

После окончания анализа классов выполняется пересмотр распределения классов по пакетам с учетом их итогового состава и зависимостей. Эта деятельность называется **анализом пакетов**.

Целью анализа пакетов является достижение следующих итогов:

- независимости или малой зависимости пакета от других пакетов;
- реализации пакетом его прецедентов;
- содержания пакетом функционально ориентированных классов.

Для анализа пакетов необходимо найти их зависимости. Зависимость пакетов определяют на основе зависимостей входящих в них классов.

Например, пакет А зависит от пакета Б, если класс в пакете А зависит от класса в пакете Б.

Реально зависимости отражают трудозатраты на последующее внесение изменений, поэтому их желательно минимизировать. Для достижения поставленных целей возможно перемещение классов между пакетами. Следует помнить, что класс может находиться только в одном пакете, который, в свою очередь, может быть вложен в другой пакет.

Для начала необходимо построить «обзорную» диаграмму всех классов, содержащую все классы и все связи между ними.

2.4.5.1. Обзорная диаграмма всех классов

После анализа отдельных классов составляется одна или более общих диаграмм классов, которые содержат все классы, выявленные при анализе, и связи между ними. Эта диаграмма формируется объединением диаграмм классов, построенных для отдельных кооперaций.

Связи между классами анализа соответствуют либо зависимостям, полученным для предметной области (для классов сущностей), либо линиям и направлениям передачи сообщений между классами в кооперациях. В обоих случаях классы связываются ассоциациями. Ассоциация между классами показывает их взаимодействие в кооперациях. Возможно использование направленной ассоциации.

Связи ассоциации уточняются, и для каждой назначаются параметры: роли классов, видимость связей, арность, множественность, ограничение на количество связей и т. д.

Кроме ассоциации на диаграмме классов указывают связи агрегации:

- композицию для физической вложенности объектов классов, например автомобиль и его детали;
- агрегацию для логической связи, например коллекция объектов, связи родителей и их детей.

Для связей агрегации также задают их параметры.

Для сокращения дублирования поведения или хранимых данных в разных классах используют обобщение. Чтобы выполнить это, разделяемое или общее поведение классов выносят в отдельный класс и определяют связь типа обобщения.

Пример:

В результате предыдущих шагов анализа требований АСУ «Веб-портал поиска пропавших животных» была сформирована Обзорная диаграмма всех классов (см. Рисунок 47).

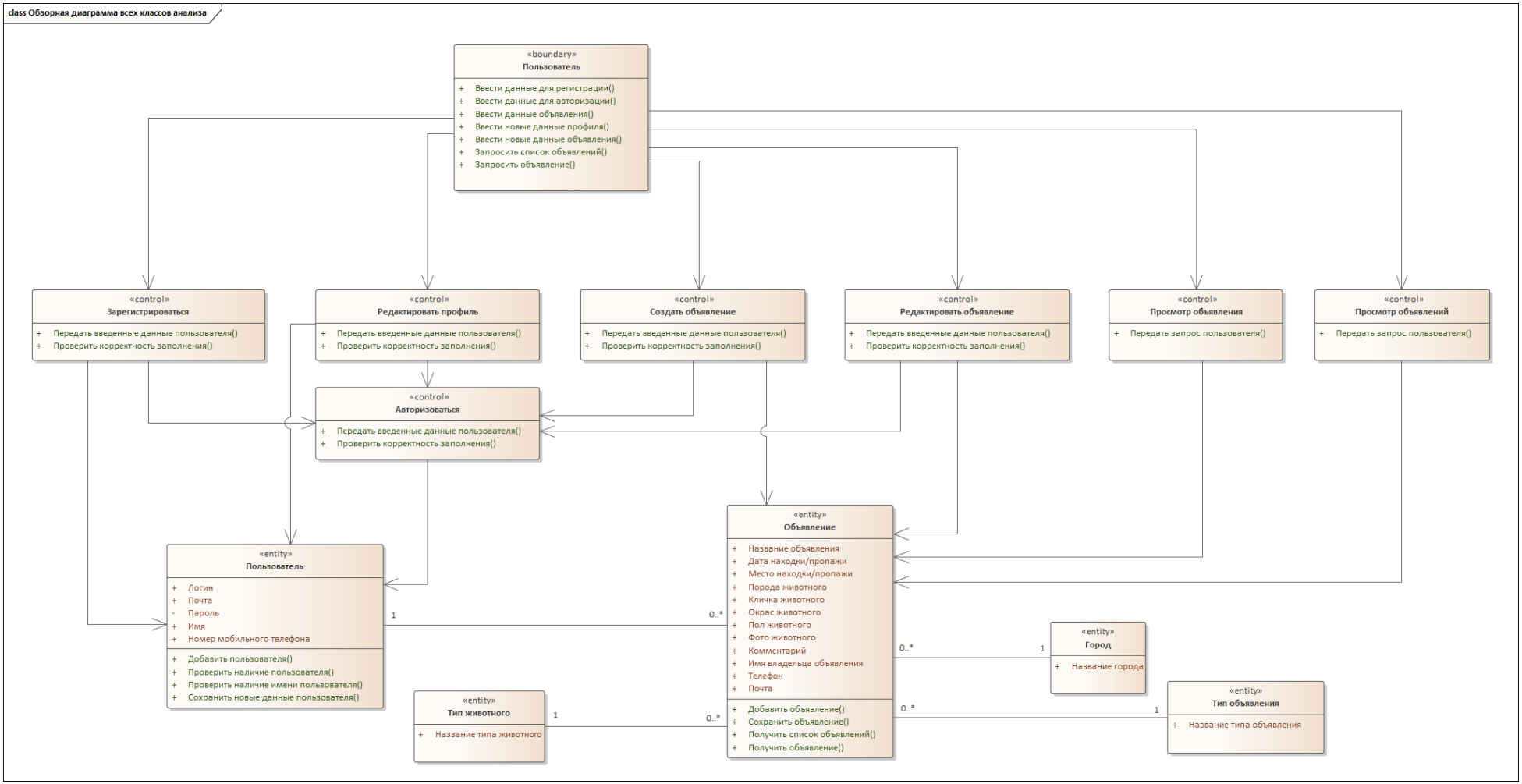

Рисунок 47 – Обзорная диаграмма всех классов для модели АСУ «Веб-портал для поиска пропавших животных»

2.4.5.2. Распределение классов по пакетам

Добавим в модель пакеты и распределим классы анализа по пакетам по принципу функциональной связности.

Пример:

Из Обзорной диаграммы классов анализа АСУ «Веб-портал поиска пропавших животных» была сформирована Диаграмма пакетов (см. Рисунок 48).

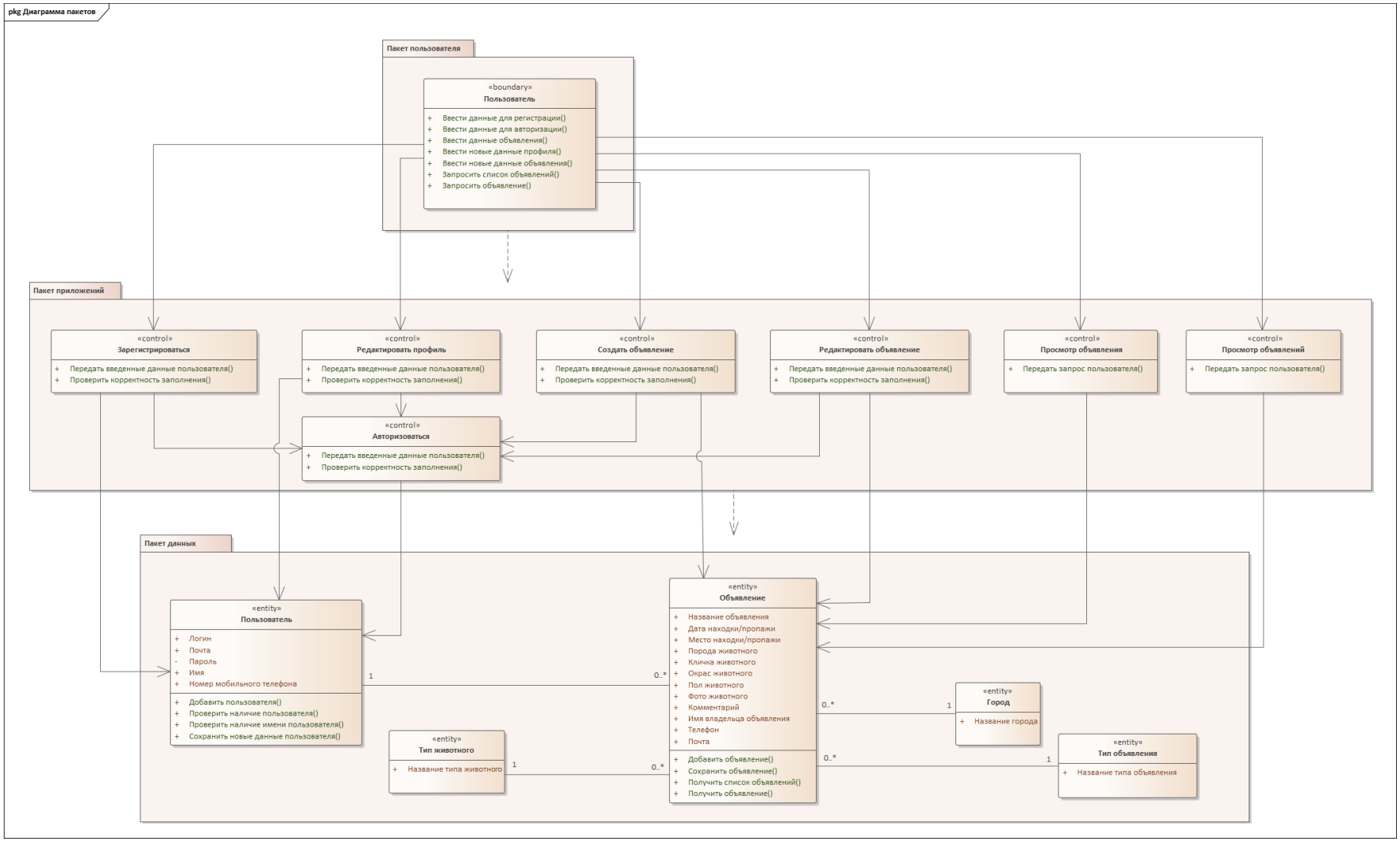

Рисунок 48 – Обзорная диаграмма пакетов (классов) для модели АСУ «Веб-портал для поиска пропавших животных»

3. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Как создать модель анализа в среде Sparx Enterprise Architect?
2. Как добавить модель? Какие типы моделей бывают?
3. Как добавить в модель диаграмму классов?
4. Как добавить в модель диаграмму последовательностей?
5. Какие типы классов анализа выделяют? По каким принципам их определяют?
6. Назовите типы связей на диаграмме классов и их параметры.
7. Перечислите типы связей на диаграмме последовательностей.
8. Назовите стереотипы классов анализа.
9. Как задать методы и свойства класса?
10. Из каких действий состоит рабочий процесс анализа?
11. Как выполняют анализ архитектуры, кооперaций, классов и пакетов?
12. Что такое пакет, кооперация, сервисный пакет? Как их определяют?
13. Как выявляют классы — участники кооперации? Как они связаны?
14. Назовите типовые атрибуты и методы граничных, управляющих и сущностных классов.
15. Как выявляют зависимости пакетов?

4. СПИСОК ИСТОЧНИКОВ

1. Sparx Systems – Текст. Изображение.: электронные // Sparx Systems : [сайт]. – URL: https://sparxsystems.com/ (дата обращения 15.06.2022)
2. Виноградова, М. В. Унифицированный процесс разработки программного обеспечения: учебное пособие / М. В. Виноградова, В. И. Белоусова. — Москва: Издательство МГТУ им. Н.Э. Баумана, 2015. — 80, [2] с.: ил. ISBN 978-5-7038-4265-2
3. Якобсон А., Дуч Г., Рамбо Дж. Унифицированный процесс разработки программного обеспечения. / А. Якобсон, Г. Дуч, Дж. Рамбо. – Спб.: Питер. – 2002.
4. Арлоу Д., Нейштадт И. UML 2 и Унифицированный процесс. Практический объектно-ориентированный анализ и проектирование, 2е издание. – Пер. с англ. – СПб: Символ Плюс, 2007. – 624 с., ил. ISBN13: 9785932860946 ISBN10: 5932860944
5. Руководство пользователя Enterprise Architect 15.1 – Текст. Изображение.: электронные // Sparx Systems : [сайт]. – URL: https://sparxsystems.com/enterprise_architect_user_guide/15.1/index/index.html (дата обращения 15.06.2022)
6. Технологии разработки программного обеспечения: Учебник/ С. Орлов. — СПб.: Питер, 2002. — 464 с.: ил. ISBN 5-94723-145-X