Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Специальность: «Программное обеспечение информационных технологий»

**КОНТРОЛЬНАЯ РАБОТА**

По курсу: «Технология разработки программного обеспечения»

На тему: «Учет транспортных средств ГАИ»

Студент-заочник 2 курса

Группы № 581072

ФИО: Амелькович Никита Александрович

Проверила: Бакунова О.М.

Минск, 2017

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc472637182)

[1 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ 4](#_Toc472637183)

[1.1 Постановка задачи 5](#_Toc472637184)

[1.2 Описание и обоснование выбранного языка программирования 5](#_Toc472637185)

[2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА 7](#_Toc472637186)

[3 ЛОГИЧЕСКОЕ МОДЕЛИРОВАНИЕ 11](#_Toc472637187)

[3.1 Диаграмма вариантов использования 11](#_Toc472637188)

[3.2 Диаграмма классов 12](#_Toc472637189)

[3.3 Диаграмма деятельности 13](#_Toc472637190)

[3.4 Диаграмма коопераций 14](#_Toc472637191)

[3.5 Диаграмма последовательности 15](#_Toc472637192)

[3.6 Диаграмма развёртывания 16](#_Toc472637193)

[3.7 Диаграмма компонентов 17](#_Toc472637194)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 19](#_Toc472637195)

[Генерация кода 20](#_Toc472637196)

# ВВЕДЕНИЕ

Сегодня компьютерные технологии развиваются с очень большой скоростью. Причиной этому стало постоянное увеличение аппаратных мощностей, а также разработка все более технологичного, производительного и оптимизированного программного обеспечения.

Технологии баз данных также не стоят на месте. За последний десяток лет на рынке появились новые СУБД, усовершенствовались старые разработки. Также были придуманы новые концепции хранения, обработки и предоставления данных.

Сегодня базы данных могут хранить огромные массивы данных, распределенные на нескольких системных кластерах и при этом обеспечивать очень высокую скорость записи и чтения данных.

Также стоит отметить тот факт, что системы стали более масштабируемы. Теперь изменение размеров баз данных можно менее затратно отразить на работе всей системы целиком.

Технологии баз данных (БД) можно эффективно применять в учете транспортных средств ГАИ. Учет транспортных средств, паспортов ТС, физических и юридических лиц, сотрудников и отделений ГАИ – обязательные функции, которые должны присутствовать в электронной системе учета транспортных средств ГАИ. Удобнее всего реализовывать хранение всех данных в таблицах БД и обращаться к ним из приложения или формировать на их основе отчеты и графики.

Современные СУБД предоставляют широкий спектр функций, которые позволяют эффективно хранить, а также эффективно и быстро предоставлять данные в нужной ситуации.

Но для того, чтобы система работала действительно эффективно, нужно чтобы она была хорошо спроектирована и реализована.

Хорошее проектировании и продумывание каждой детали программного обеспечения позволят «выжать» всю мощь с аппаратной составляющей системы.

Также, хорошее проектирование позволяет легче поддерживать уже работающую систему при необходимости введения новых функций, так как при добавлении новых функций в хорошей системе меньше будет затронуто сторонних модулей системы, что, в свою очередь, оградит систему от новых ошибок.

# 1 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

Государственная автомобильная инспекция в своей базе хранит полную информацию обо всех транспортных средствах, зарегистрированных в стране. Данная база включает в себя информацию о регистрационном номере транспортного средства, о хозяине транспортного средства, о номерах кузова и двигателя, о цвете транспортного средства, о двигателе, марке и модели, о годе выпуска.

Предметной областью учета транспортных средств государственной автомобильной инспекции являются все транспортные средства, зарегистрированные в стране.

Задачами автоматизации учета транспортных средств ГАИ являются:

* Постановка на учет транспортного средства;
* Учет владельцев ТС;
* Сверка номеров.

Сама регистрация транспортного средства может производиться для физического или юридического лица. В зависимости от этого процедуры постановки на учет отличаются.

Общий вид процедуры постановки на учет транспортного средства:

* Подача заявления установленного образца;
* Отметка военкомата о постановке на учет (юр.л.);
* Сверка номеров;
* Подача документов, подтверждающих законность производства, получения или приобретения транспортного средства (паспорт транспортного средства завода-изготовителя для транспортных средств не бывших в эксплуатации; счет-справка; договор купли-продажи; акт приема-передачи);
* Доверенность на право проведения регистрационных действий в МРО ГАИ(юр.л);
* Страховой полис обязательного страхования гражданской ответственности владельца транспортного средства;
* Платежные поручения.

Процедура проверки номеров включает в себе проверку номеров кузова и двигателя транспортного средства с базой транспортных средств ГАИ. Проводится проверка достоверности этих номеров.

В паспорте транспортного средства указываются идентификационный номер (VIN), марка, модель, наименование и категория транспортного средства, год изготовления, модель и номер двигателя, номера шасси и кузова, цвет кузова, мощность, рабочий объём и тип двигателя, разрешённая максимальная масса, масса без нагрузки, а также сведения об изготовителе, страна вывоза и таможенные ограничения. Каждый паспорт транспортного средства имеет свой идентификационный номер.

Счет-справка хранит в себе туже информацию, что и паспорт транспортного средства. Кроме этого также указана стоимость автомобиля, имя продавца, имя покупателя. Также у счет-справки есть собственный номер и серия.

Договор купли-продажи хранит такие же данные как и счет-справка, но имеет официальную форму, то есть не имеет личного идентификационного номера.

Акт приема-передачи регистрирует факт передачи транспортного средства от продавца к покупателю. Хранит аналогичную информацию договору купли-продажи, но не хранит в себе стоимость.

# Постановка задачи

Чтобы использовать свое транспортное средство (ТС) владелец должен поставить его на учет в государственной автомобильной инспекции (ГАИ). При постановке на учет у водителя в ГАИ затребуют сведения об автомобиле (технический паспорт), сведения о владельце (документ удостоверяющий личность), результаты процедуры сверки номеров, документ, подтверждающий законность приобретения транспортного средства (договори купли-продажи, счет-фактуру и так далее) и иные документы, при возникновении каких-либо нестандартных ситуаций.

Процедура сверки номеров подразумевает под собой сверку всех номеров на кузове автомобиля, а также на двигателе транспортного средства. Удачное прохождение данной процедуры гарнирует, что все детали являются заводскими или же были легально заменены в сервисном центре.

Документ купли-продажи и иные документы данного типа гарантируют факт законного приобретения или принятия в дар транспортного средства.

После взноса всех платежей и предоставления всего необходимого перечня документов владельцу будет выдан новый технический паспорт, который будет подтверждать факт постановки на учет транспортного средства.

Программное средство учета транспортных средств ГАИ должно реализовывать в себе все необходимые процедуры, чтобы в достаточной мере автоматизировать весь процесса постановки на учет.

Саму процедуру постановки на учет и все ее составные субпроцедуры должен проводить сотрудник ГАИ, который обладает достаточными знаниями, навыками и полномочиями для проведения данной операции.

При возникновении внештатных ситуаций, решение которых не может произвести сотрудник ГАИ, в процесс должен вмешаться администратор, который обладает более широкими полномочиями по манипулированию данных.

Владелец (пользователь) имеет самый узкий круг полномочий. Он может только получить сведения по постановке на учет транспортных средств. Полномочий на манипулирование данных он не имеет.

# 1.2 Описание и обоснование выбранного языка программирования

Java — объектно-ориентированный язык программирования, разрабатываемый компанией Sun Microsystems с 1991 года и официально выпущенный 23 мая 1995 года. Изначально новый язык программирования назывался Oak (James Gosling) и разрабатывался для бытовой электроники, но впоследствии был переименован в Java и стал использоваться для написания апплетов, приложений и серверного программного обеспечения

Отличительной особенностью Java в сравнении с другими языками программирования общего назначения является обеспечение высокой продуктивности программирования, нежели производительность работы приложения или эффективность использования им памяти.

В Java используются практически идентичные соглашения для объявления переменных, передачи параметров, операторов и для управления потоком выполнением кода. В Java добавлены все хорошие черты C++.

Три ключевых элемента объединились в технологии языка Java

Java предоставляет для широкого использования свои апплеты (applets) — небольшие, надежные, динамичные, не зависящие от платформы активные сетевые приложения, встраиваемые в страницы Web. Апплеты Java могут настраиваться и распространяться потребителям с такой же легкостью, как любые документы HTML

Java высвобождает мощь объектно-ориентированной разработки приложений, сочетая простой и знакомый синтаксис с надежной и удобной в работе средой разработки. Это позволяет широкому кругу программистов быстро создавать новые программы и новые апплеты

Java предоставляет программисту богатый набор классов объектов для ясного абстрагирования многих системных функций, используемых при работе с окнами, сетью и для ввода-вывода. Ключевая черта этих классов заключается в том, что они обеспечивают создание независимых от используемой платформы абстракций для широкого спектра системных интерфейсов

Огромное преимущество Java заключается в том, что на этом языке можно создавать приложения, способные работать на различных платформах. К сети Internet подключены компьютеры самых разных типов - Pentium PC, Macintosh, рабочие станции Sun и так далее. Даже в рамках компьютеров, созданных на базе процессоров Intel, существует несколько платформ, например, Microsoft Windows версии 3.1, Windows 95, Windows NT, OS/2, Solaris, различные разновидности операционной системы UNIX с графической оболочкой X­Windows. Между тем, создавая сервер Web в сети Internet, хотелось бы, чтобы им могло пользоваться как можно большее число людей. В этом случае выручат приложения Java, предназначенные для работы на различных платформах и не зависящие от конкретного типа процессора и операционной системы.

# 2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

В работе проводится моделирование с использование IDEF0(BPWin), UML (Rational Rose 2000), IDEF1x (ErWin).

1. Важная роль отводится процессу функционального проектирования.

Для регламентирования создания функциональных моделей ПС предназначен стандарт IDEF0 (Integrated Definition Function Modeling), который и реализован в пакете BpWin.

В основе IDEF0 лежит понятие блока, который реализует некую конкретную функцию. Четыре стороны блока имеют разное назначение. Слева отображаются входные данные (исходные данные). Справа – выходные данные (результат выполнения функции). Сверху – управление (управляющие воздействия на функцию). Снизу – механизм (посредством чего реализуется данная функция).

Функция – это управляемое действие над входными данными, результатом которого являются выходные данные, при этом используется некий механизм. Взаимодействие между функциями отображается в виде стрелок. Иногда стороны блока называют направлениями, а стрелки потоками. Стрелки можно подписывать. Подписи связываются с конкретной стрелкой при помощи зигзага.

В основе IDEF0 лежит три базовых принципа:

1) принцип функциональной декомпозиции – любая функция может быть разбита (декомпозирована) на более простые функции (более понятен термин детализация);

2) принцип ограничения сложности – количество блоков на диаграмме должно быть не менее двух, но не более шести (условие удобочитаемости);

3) принцип контекста – моделирование делового процесса начинается с построения контекстной диаграммы, на которой отображается только один блок – главная функция моделирующей системы, ограничивающая область границы моделирующей системы (регламентирует начальный этап построения модели).

Под субъектом понимается сама система, при этом необходимо точно установить, что входит в систему, а что является внешним воздействием на систему. Т. е. первоначально нужно определить область моделирования. Описание области как системы в целом, так и ее компонентов является основой построения модели.

IDEF1X является методом для разработки реляционных баз данных и использует условный синтаксис, специально разработанный для удобного построения концептуальной схемы. Концептуальной схемой мы называем универсальное представление структуры данных в рамках коммерческого предприятия, независимое от конечной реализации базы данных и аппаратной платформы. Будучи статическим методом разработки, IDEF1X изначально не предназначен для динамического анализа по принципу "AS IS", тем не менее, он иногда применяется в этом качестве, как альтернатива методу IDEF1. Использование метода IDEF1X наиболее целесообразно для построения логической структуры базы данных после того, как все информационные ресурсы исследованы (скажем с помощью метода IDEF1) и решение о внедрении реляционной базы данных, как части корпоративной информационной системы, было принято.

Хотя терминология IDEF1X практически совпадает с терминологией IDEF1, существует ряд фундаментальных отличий в теоретических концепциях этих методологий. Сущность в IDEF1X описывает собой совокупность или набор экземпляров похожих по свойствам, но однозначно отличаемых друх от друга по одному или нескольким признакам. Каждый экземпляр является реализацией сущности. Таким образом, сущность в IDEF1X описывает конкретный набор экземпляров реального мира, в отличие от сущности в IDEF1, которая представляет собой абстрактный набор информационных отображений реального мира. Примером сущности IDEF1X может быть сущность "СОТРУДНИК", которая представляет собой всех сотрудников предприятия, а один из них, скажем, Иванов Петр Сергеевич, является конкретной реализацией этой сущности.

Сase-средство BPwin значительно облегчает задачу создания информационной системы, позволяя осуществить декомпозицию сложной системы на более простые с тем, чтобы каждая из них могла проектироваться независимо, и для понимания любого уровня проектирования достаточно было оперировать с информацией о немногих ее частях.

Рассмотрим процесс моделирования в методологии IDEF0 на примере контекстной диаграммы разработанной модели процесса учета кредитных операций в банке.

Цель модели – моделирование процесса учета транспортных средств ГАИ.

Входами модели являются:

1. Авторизация (авторизационные данные) – данные, которые используются при авторизации любого пользователя системы;
2. Информация о машинах – содержит такую информация как год выпуска, мощность, массу, цвет, марку, модель и так далее;
3. Информация о владельцах – та информация, которая используется при регистрации транспортного средства, чтобы обозначить хозяина этого средства;
4. Информация о сотрудниках – информация, которая показывает, который именно сотрудник регистрировал транспортное средство.

Процесс регламентируется законодательством РБ, а также ГАИ УВД.

На выходе получается запись в БД, а также отчетная документация. В качестве пользователей системы могут выступать администратор, который владеет самыми широкими правами, сотрудник, который выполняет функцию регистрации ТС, а также пользователь (владелец), который может посмотреть информацию о регистрации.

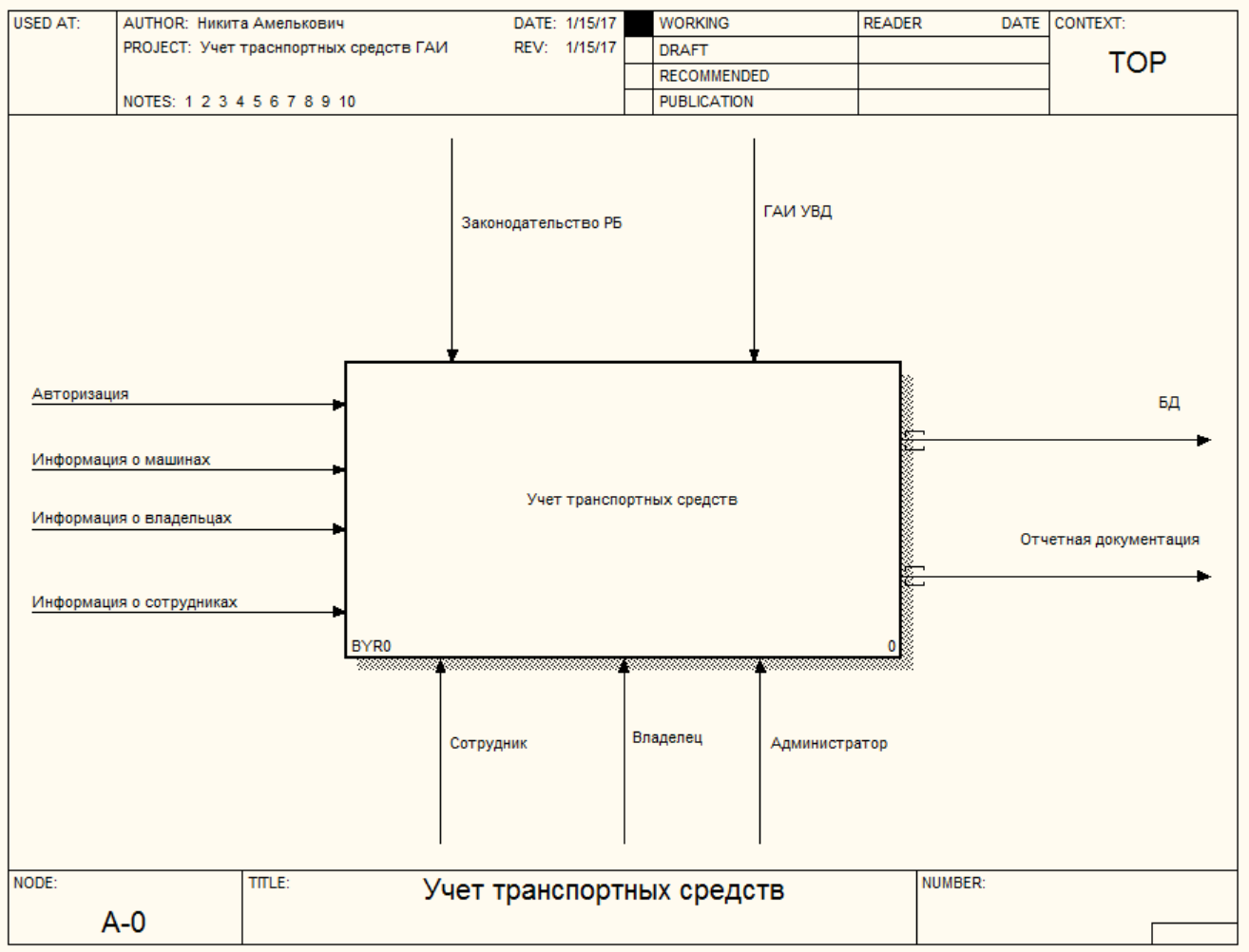


Рисунок 2.1 – Контекстная диаграмма модели

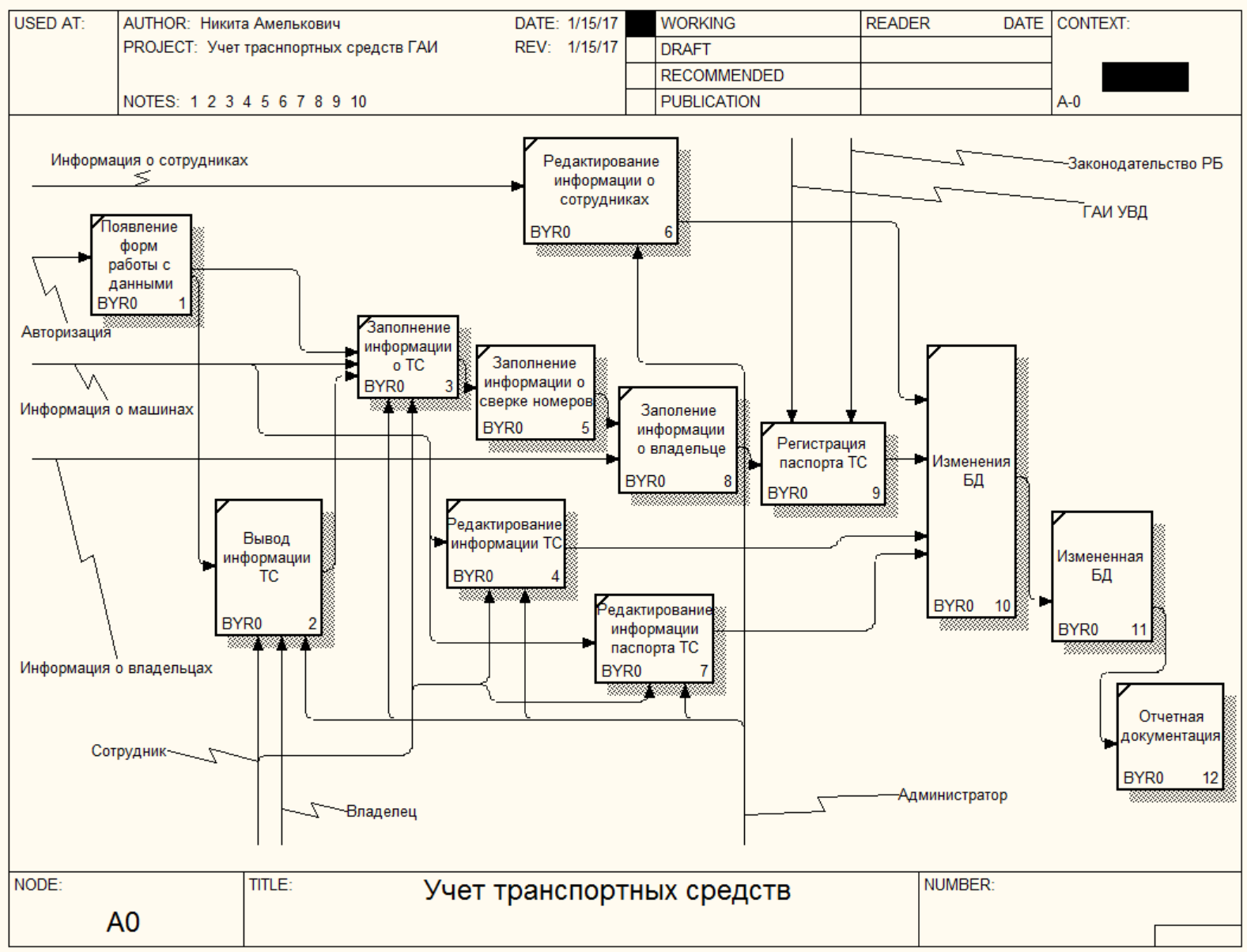


Рисунок 2.2– Декомпозиция контекстной диаграммы

функциональной модели

Декомпозиция модели делится на внутренние процессы такие как:

1. Появление форм работы с данными;
2. Редактирование информации о сотрудниках;
3. Заполнение информации о ТС;
4. Заполнение информации о владельце;
5. Заполнение информации о сверке номеров;
6. Регистрация паспорта ТС;
7. Изменения БД;
8. Измененная БД;
9. Отчетная документация;
10. Вывод информации о ТС;
11. Редактирование информации ТС;
12. Редактирование паспорта ТС.

# 3 ЛОГИЧЕСКОЕ МОДЕЛИРОВАНИЕ

Основное назначение логического представления состоит в анализе структурных и функциональных отношений между элементами модели системы. Различные элементы логического представления, такие как классы, ассоциации, состояния, сообщения, не существуют материально или физически. Они лишь отражают наше понимание структуры физической системы или аспекты ее поведения.

# 3.1 Диаграмма вариантов использования

Модель вариантов использования предназначается для определения требований к системе. Она включает в себя актеров, варианты использования и связи между ними. Для отображения этой модели язык UML предлагает использовать диаграммы Use Case (вариант использования) совместно с моделями State Diagram (диаграммы состояний) и Activity Diagram (диаграммы деятельности/активности). Последние используются для конкретизации вариантов использования системы

Разработка диаграммы вариантов использования преследует цели:

* определить общие границы и контекст моделируемой предметной области на начальных этапах проектирования системы;
* сформулировать общие требования к функциональному поведению проектируемой системы;
* разработать исходную концептуальную модель системы для ее последующей детализации в форме логических и физических моделей;
* подготовить исходную документацию для взаимодействия разработчиков системы с ее заказчиками и пользователями.

Диаграмма вариантов использования представлена на рисунке 3.1.

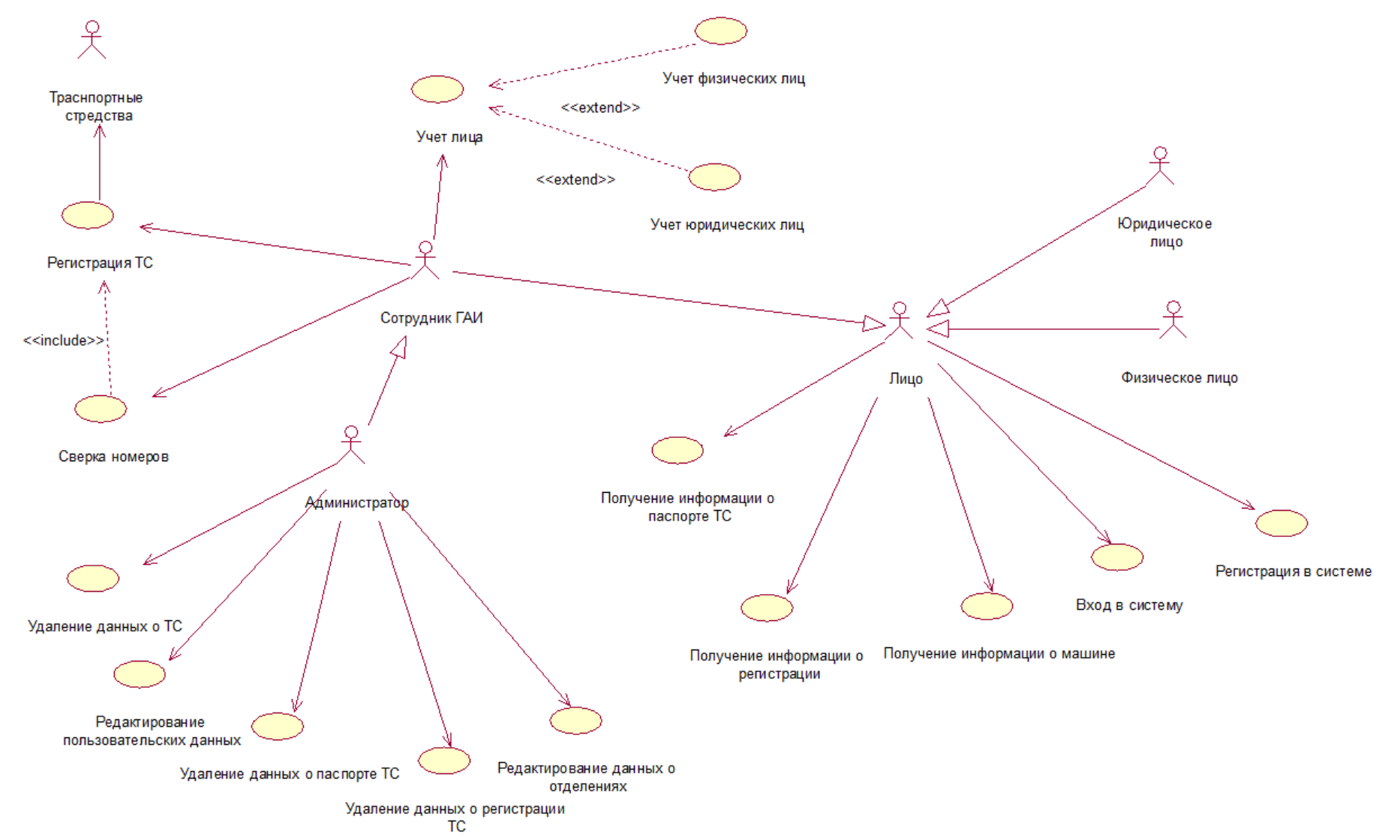


Рисунок 3.1 – Диаграмма вариантов использования

# 3.2 Диаграмма классов

Диаграмма классов — диаграмма, демонстрирующая классы системы, их атрибуты, методы и взаимосвязи между ними. Входит в UML.

Существует два вида:

* статический вид диаграммы рассматривает логические взаимосвязи классов между собой;
* аналитический вид диаграммы рассматривает общий вид и взаимосвязи классов, входящих в систему.

Существуют разные точки зрения на построение диаграмм классов в зависимости от целей их применения:

* концептуальная точка зрения — диаграмма классов описывает модель предметной области, в ней присутствуют только классы прикладных объектов;
* точка зрения спецификации — диаграмма классов применяется при проектировании информационных систем;
* точка зрения реализации — диаграмма классов содержит классы, используемые непосредственно в программном коде (при использовании объектно-ориентированных языков программирования).

Диаграмма классов представлена на рисунке 3.2.

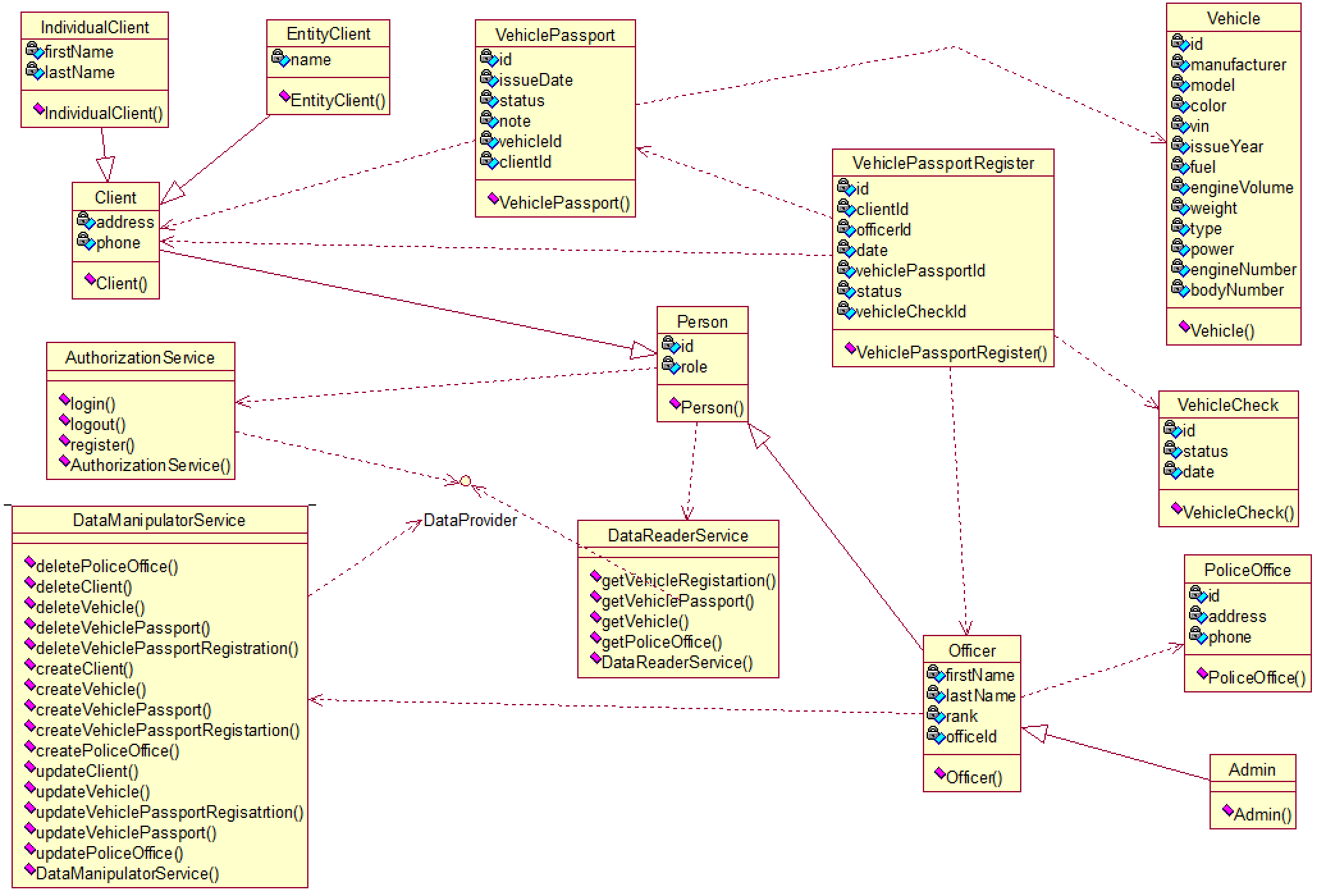


Рисунок 3.2 – Диаграмма классов

# 3.3 Диаграмма деятельности

Диаграмма деятельности — UML-диаграмма, на которой показано разложение некоторой деятельности на её составные части. Под деятельностью (англ. activity) понимается спецификация исполняемого поведения в виде координированного последовательного и параллельного выполнения подчинённых элементов — вложенных видов деятельности и отдельных действий англ. action, соединённых между собой потоками, которые идут от выходов одного узла ко входам другого.

Диаграммы деятельности используются при моделировании бизнес-процессов, технологических процессов, последовательных и параллельных вычислений.

Диаграмма деятельности представлена на рисунке 3.3.

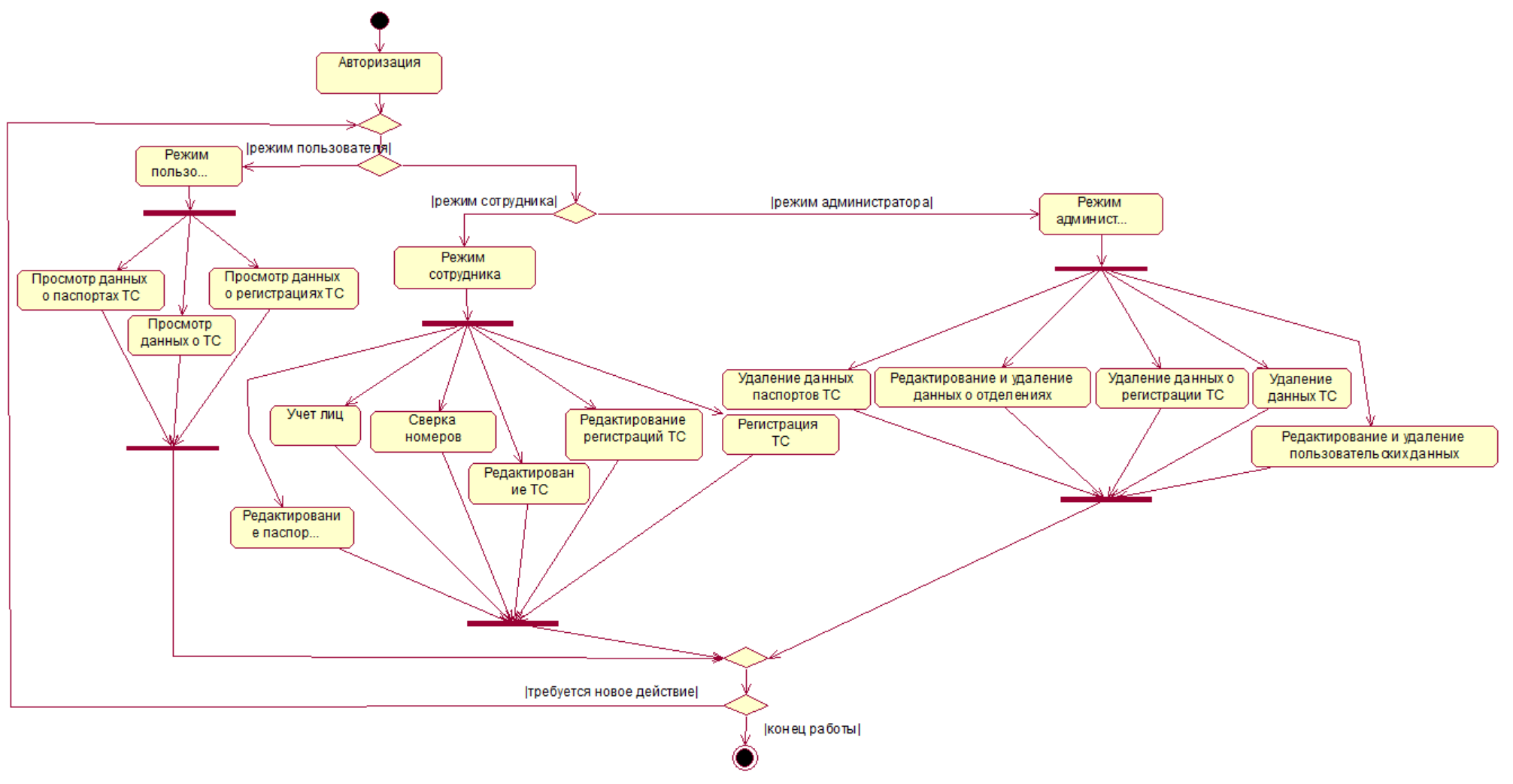


Рисунок 3.3 – Диаграмма деятельности

# 3.4 Диаграмма коопераций

Главная особенность диаграммы кооперации заключается в возможности графически представить не только последовательность взаимодействия, но и все структурные отношения между объектами, участвующими в этом взаимодействии.

Прежде всего, на диаграмме кооперации в виде прямоугольников изображаются участвующие во взаимодействии объекты, содержащие имя объекта, его класс и, возможно, значения атрибутов. Далее, как и на диаграмме классов, указываются ассоциации между объектами в виде различных соединительных линий. При этом можно явно указать имена ассоциации и ролей, которые играют объекты в данной ассоциации. Дополнительно могут быть изображены динамические связи - потоки сообщений. Они представляются также в виде соединительных линий между объектами, над которыми располагается стрелка с указанием направления, имени сообщения и порядкового номера в общей последовательности инициализации сообщений.

В отличие от диаграммы последовательности, на диаграмме кооперации изображаются только отношения между объектами, играющими определенные роли во взаимодействии. На этой диаграмме не указывается время в виде отдельного измерения. Поэтому последовательность взаимодействий и параллельных потоков может быть определена с помощью порядковых номеров. Следовательно, если необходимо явно специфицировать взаимосвязи между объектами в реальном времени, лучше это делать на диаграмме последовательности. Диаграмма коопераций представлена на рисунке 3.4.

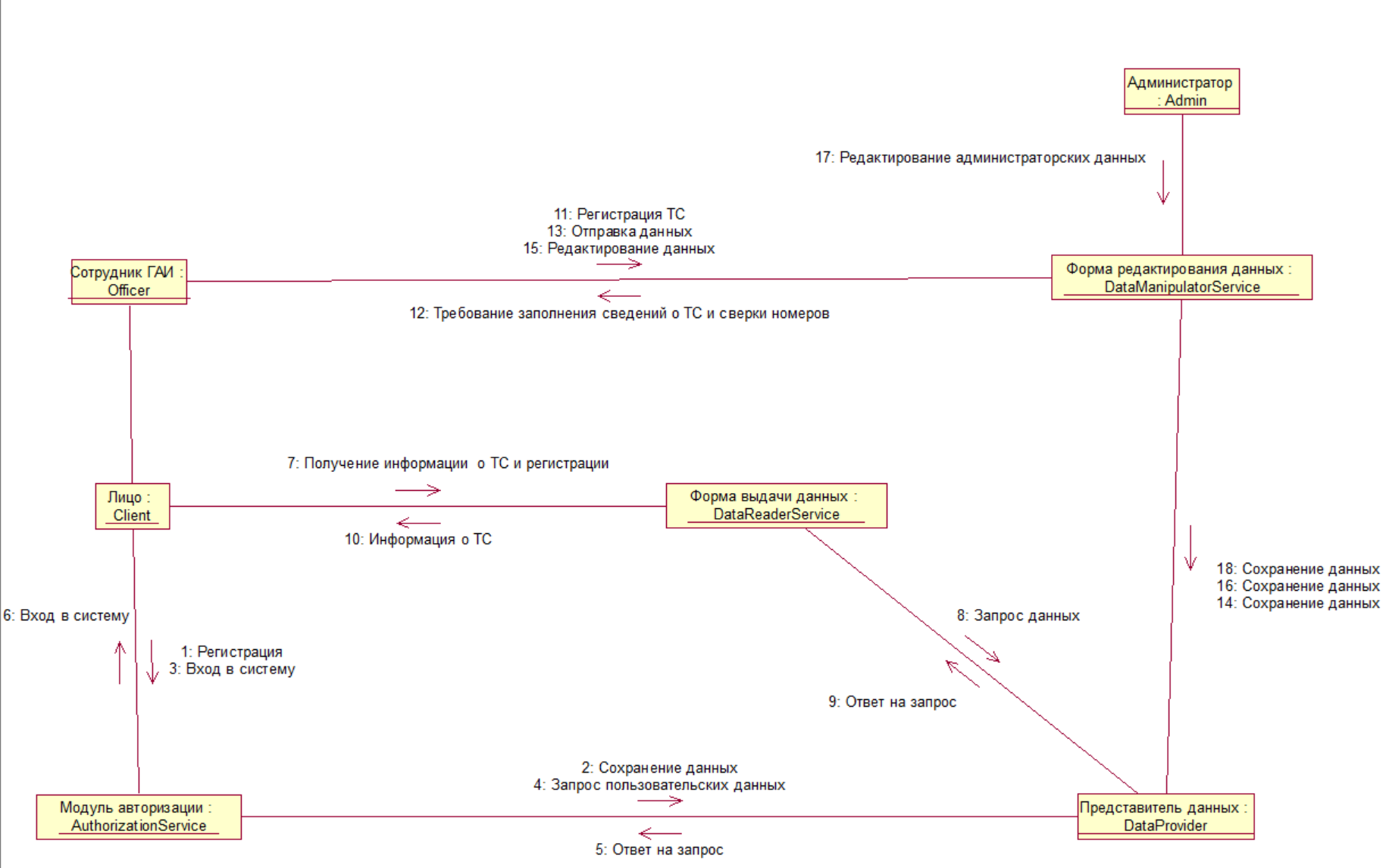


Рисунок 3.4 – Диаграмма коопераций

# 3.5 Диаграмма последовательности

Диаграмма последовательности — диаграмма, на которой показано взаимодействие объектов, упорядоченное по времени, с отражением продолжительности обработки и последовательности их проявления. Используется в языке UML.

Основными элементами диаграммы последовательности являются обозначения объектов (прямоугольники с названиями объектов), вертикальные «линии жизни», отображающие течение времени, прямоугольники, отражающие деятельность объекта или исполнение им определенной функции (прямоугольники на пунктирной «линии жизни»), и стрелки, показывающие обмен сигналами или сообщениями между объектами.

Диаграмма последовательности представлена на рисунке 3.5.

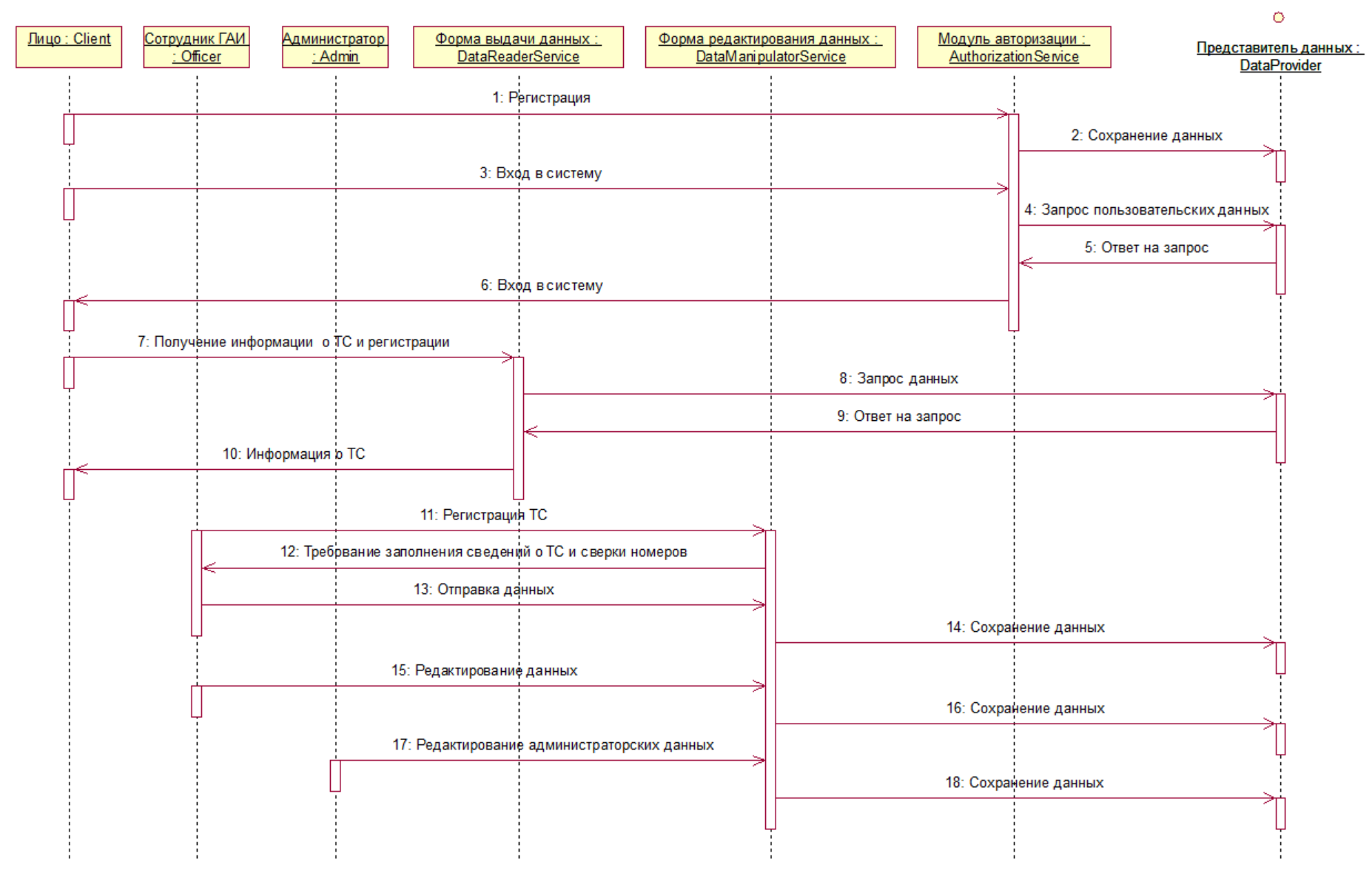


Рисунок 3.5 – Диаграмма последовательности

# 3.6 Диаграмма развёртывания

Диаграмма развёртывания в UML моделирует физическое развертывание артефактов на узлах. Например, чтобы описать веб-сайт диаграмма развертывания должна показывать, какие аппаратные компоненты («узлы») существуют (например, веб-сервер, сервер базы данных, сервер приложения), какие программные компоненты («артефакты») работают на каждом узле (например, веб-приложение, база данных), и как различные части этого комплекса соединяются друг с другом (например, JDBC, REST, RMI).

Узлы представляются как прямоугольные параллелепипеды с артефактами, расположенными в них, изображенными в виде прямоугольников. Узлы могут иметь под узлы, которые представляются как вложенные прямоугольные параллелепипеды. Один узел диаграммы развертывания может концептуально представлять множество физических узлов, таких как кластер серверов баз данных.

Существует два типа узлов:

* узел устройства;
* узел среды выполнения.

Узлы устройств — это физические вычислительные ресурсы со своей памятью и сервисами для выполнения программного обеспечения, такие как обычные ПК, мобильные телефоны. Узел среды выполнения — это программный вычислительный ресурс, который работает внутри внешнего узла и который предоставляет собой сервис, выполняющий другие исполняемые программные элементы. Диаграмма развёртывания представлена на рисунке 3.6.

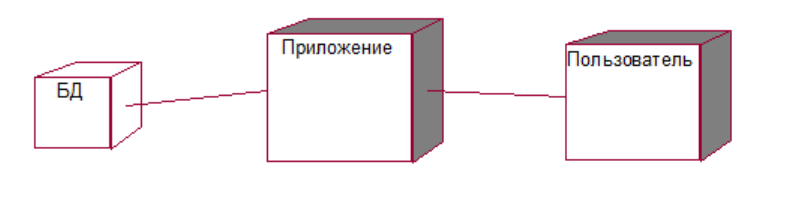


Рисунок 3.6 – Диаграмма развёртывания

# 3.7 Диаграмма компонентов

Диаграмма компонентов — статическая структурная диаграмма, показывает разбиение программной системы на структурные компоненты и связи (зависимости) между компонентами. В качестве физических компонентов могут выступать файлы, библиотеки, модули, исполняемые файлы, пакеты и т. п.

Компоненты связываются через зависимости, когда соединяется требуемый интерфейс одного компонента с имеющимся интерфейсом другого компонента. Таким образом иллюстрируются отношения клиент-источник между двумя компонентами.

Зависимость показывает, что один компонент предоставляет сервис, необходимый другому компоненту. Зависимость изображается стрелкой от интерфейса или порта клиента к импортируемому интерфейсу.

Когда диаграмма компонентов используется, чтобы показать внутреннюю структуру компонентов, предоставляемый и требуемый интерфейсы составного компонента могут делегироваться в соответствующие интерфейсы внутренних компонентов.

Делегация показывается связь внешнего контракта компонента с внутренней реализацией этого поведения внутренними компонентами.

Диаграмма компонентов представлена на рисунке 3.7.

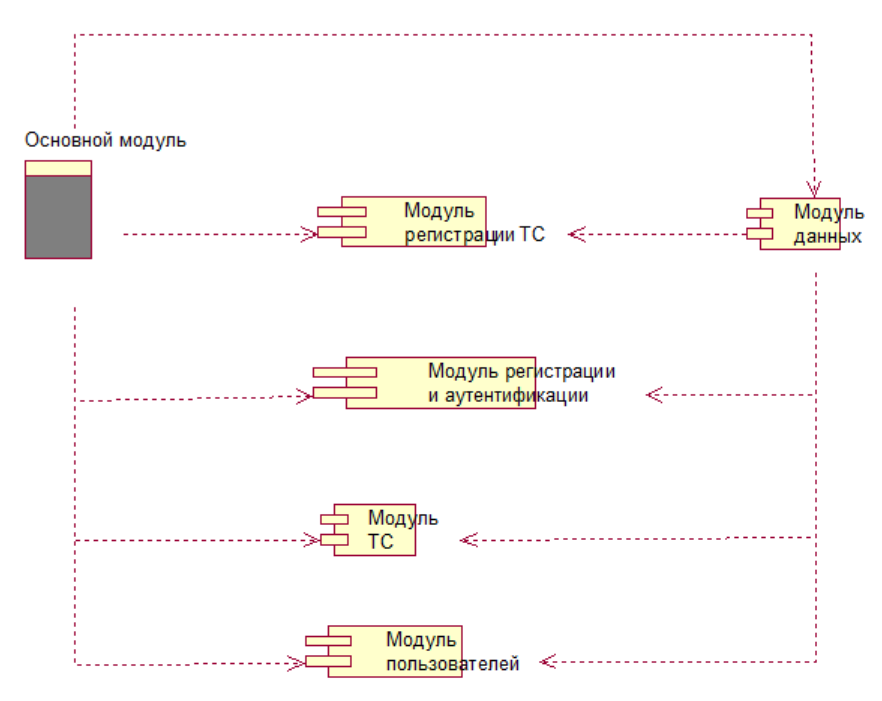


Рисунок 3.7 – Диаграмма компонентов

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ГОСТ 2.105-95 ЕСКД. Введ. 1996–07–01. – М.: Изд-во стандартов, 1996.
2. ГОСТ 19.104–78 – Единая система программной документации. Основные надписи.
3. Маклаков С.В. BPwin и ERwin: CASE - средства для разработки информационных систем.
4. Федотов Д.Э., Семенов Ю.Д., Чижик К.Н. Практикум для высших учебных заведений. CASE-технологии. -157с.
5. Интернет обучение [Электронный ресурс] / Разработка диаграммы кооперации и редактирование свойств ее элементов – Режим доступа: http://www.intuit.ru/ – Дата доступа: 12.01.2017.
6. Интернет обучение [Электронный ресурс] / Информационные системы и технологии – Режим доступа: <http://www.narfu.ru> – Дата доступа: 12.01.2017.
7. Интернет обучение [Электронный ресурс] / Нормализация структурны данных – Режим доступа: http://infostart.ru/public/269803/ – Дата доступа: 12.01.2017.
8. Интернет обучение [Электронный ресурс] / Программа компьютерного моделирования BpWin – Режим доступа: http://bourabai.kz/cm/bpwin.htm – Дата доступа: 12.01.2017.
9. Р. Мюллер. [Базы данных и UML: Проектирование](javascript:parent.descr(window,%22d160587%22)).– Лори, 2002г. 432 с.
10. Фельдман С.К. Система программирования Java без секретов: Как создать безопасное приложение с "нуля". – Новый издательский дом" , 2005 г. , 347 с.
11. Дейтел П.Дж., Дейтел Х.М. Как програмировать на Java. Книга 2. Файлы, сети, базы данных. – "Бином" · 2005 г., 672 с.
12. Х. Гома. UML Проектирование систем реального времени, параллельных и распределенных приложений. – «ДМК». 2016, 700 с.
13. Р. Максимчук. UML для простых смертных. - «Лори». 2016. 268 с.
14. Д. Пайлон. UML 2 для программистов. - «Библиотека O`Reilly». 2012. 240 c.
15. М. Боггс. UML и Rational Rose. - «Лори». 2008. 580 с.

# Генерация кода

//Source file: C:\\USERS\\MIKITA\\YANDEXDISK\\�����\\2016-2017\\����\\NIKITA\\CODE\\Admin.java

public class Admin extends Officer

{

/\*\*

\* @roseuid 587B6E54035C

\*/

public Admin()

{

}

}

//Source file: C:\\USERS\\MIKITA\\YANDEXDISK\\�����\\2016-2017\\����\\NIKITA\\CODE\\AuthorizationService.java

public class AuthorizationService

{

/\*\*

\* @roseuid 587B6E400022

\*/

public AuthorizationService()

{

}

/\*\*

\* @roseuid 5879EF0000FA

\*/

public void login()

{

}

/\*\*

\* @roseuid 5879EF020152

\*/

public void logout()

{

}

/\*\*

\* @roseuid 5879EF060090

\*/

public void register()

{

}

}

//Source file: C:\\USERS\\MIKITA\\YANDEXDISK\\�����\\2016-2017\\����\\NIKITA\\CODE\\Client.java

public class Client extends Person

{

private int address;

private int phone;

/\*\*

\* @roseuid 587B6E730005

\*/

public Client()

{

}

}

//Source file: C:\\USERS\\MIKITA\\YANDEXDISK\\�����\\2016-2017\\����\\NIKITA\\CODE\\DataManipulatorService.java

public class DataManipulatorService

{

/\*\*

\* @roseuid 587B6E61019C

\*/

public DataManipulatorService()

{

}

/\*\*

\* @roseuid 5879FC970203

\*/

public void deletePoliceOffice()

{

}

/\*\*

\* @roseuid 5879FCA3030C

\*/

public void deleteClient()

{

}

/\*\*

\* @roseuid 5879FCAA02E5

\*/

public void deleteVehicle()

{

}

/\*\*

\* @roseuid 5879FCAF034B

\*/

public void deleteVehiclePassport()

{

}

/\*\*

\* @roseuid 5879FCB703A9

\*/

public void deleteVehiclePassportRegistration()

{

}

/\*\*

\* @roseuid 5879FCC40049

\*/

public void createClient()

{

}

/\*\*

\* @roseuid 5879FCD302CB

\*/

public void createVehicle()

{

}

/\*\*

\* @roseuid 5879FCE00092

\*/

public void createVehiclePassport()

{

}

/\*\*

\* @roseuid 5879FCE80234

\*/

public void createVehiclePassportRegistartion()

{

}

/\*\*

\* @roseuid 5879FD1503C6

\*/

public void createPoliceOffice()

{

}

/\*\*

\* @roseuid 5879FD2002B7

\*/

public void updateClient()

{

}

/\*\*

\* @roseuid 5879FD3100C4

\*/

public void updateVehicle()

{

}

/\*\*

\* @roseuid 5879FD400345

\*/

public void updateVehiclePassportRegisatrtion()

{

}

/\*\*

\* @roseuid 5879FD4C03DA

\*/

public void updateVehiclePassport()

{

}

/\*\*

\* @roseuid 5879FD5902C3

\*/

public void updatePoliceOffice()

{

}

}

//Source file: C:\\USERS\\MIKITA\\YANDEXDISK\\�����\\2016-2017\\����\\NIKITA\\CODE\\DataProvider.java

public interface DataProvider

{

}

//Source file: C:\\USERS\\MIKITA\\YANDEXDISK\\�����\\2016-2017\\����\\NIKITA\\CODE\\DataReaderService.java

public class DataReaderService

{

/\*\*

\* @roseuid 587B6E2C01A3

\*/

public DataReaderService()

{

}

/\*\*

\* @roseuid 5879FC610180

\*/

public void getVehicleRegistartion()

{

}

/\*\*

\* @roseuid 5879FC7102A6

\*/

public void getVehiclePassport()

{

}

/\*\*

\* @roseuid 5879FC770301

\*/

public void getVehicle()

{

}

/\*\*

\* @roseuid 5879FC7E02A2

\*/

public void getPoliceOffice()

{

}

}

//Source file: C:\\USERS\\MIKITA\\YANDEXDISK\\�����\\2016-2017\\����\\NIKITA\\CODE\\EntityClient.java

public class EntityClient extends Client

{

private int name;

/\*\*

\* @roseuid 587B6DCA030A

\*/

public EntityClient()

{

}

}

//Source file: C:\\USERS\\MIKITA\\YANDEXDISK\\�����\\2016-2017\\����\\NIKITA\\CODE\\IndividualClient.java

public class IndividualClient extends Client

{

private int firstName;

private int lastName;

/\*\*

\* @roseuid 587B6DBD002B

\*/

public IndividualClient()

{

}

}

//Source file: C:\\USERS\\MIKITA\\YANDEXDISK\\�����\\2016-2017\\����\\NIKITA\\CODE\\Officer.java

public class Officer extends Person

{

private int firstName;

private int lastName;

private int rank;

private int officeId;

/\*\*

\* @roseuid 587B6E20009F

\*/

public Officer()

{

}

}

//Source file: C:\\USERS\\MIKITA\\YANDEXDISK\\�����\\2016-2017\\����\\NIKITA\\CODE\\Person.java

public class Person

{

private int id;

private int role;

/\*\*

\* @roseuid 587B6E350290

\*/

public Person()

{

}

}

//Source file: C:\\USERS\\MIKITA\\YANDEXDISK\\�����\\2016-2017\\����\\NIKITA\\CODE\\PoliceOffice.java

public class PoliceOffice

{

private int id;

private int address;

private int phone;

/\*\*

\* @roseuid 587B6E13022C

\*/

public PoliceOffice()

{

}

}

//Source file: C:\\USERS\\MIKITA\\YANDEXDISK\\�����\\2016-2017\\����\\NIKITA\\CODE\\Vehicle.java

public class Vehicle

{

private int id;

private int manufacturer;

private int model;

private int color;

private int vin;

private int issueYear;

private int fuel;

private int engineVolume;

private int weight;

private int type;

private int power;

private int engineNumber;

private int bodyNumber;

/\*\*

\* @roseuid 587B6E010232

\*/

public Vehicle()

{

}

}

//Source file: C:\\USERS\\MIKITA\\YANDEXDISK\\�����\\2016-2017\\����\\NIKITA\\CODE\\VehicleCheck.java

public class VehicleCheck

{

private int id;

private int status;

private int date;

/\*\*

\* @roseuid 587B6E0B007F

\*/

public VehicleCheck()

{

}

}

//Source file: C:\\USERS\\MIKITA\\YANDEXDISK\\�����\\2016-2017\\����\\NIKITA\\CODE\\VehiclePassport.java

public class VehiclePassport

{

private int id;

private int issueDate;

private int status;

private int note;

private int vehicleId;

private int clientId;

/\*\*

\* @roseuid 587B6DD502A7

\*/

public VehiclePassport()

{

}

}

//Source file: C:\\USERS\\MIKITA\\YANDEXDISK\\�����\\2016-2017\\����\\NIKITA\\CODE\\VehiclePassportRegister.java

public class VehiclePassportRegister

{

private int id;

private int clientId;

private int officerId;

private int date;

private int vehiclePassportId;

private int status;

private int vehicleCheckId;

/\*\*

\* @roseuid 587B6DF80204

\*/

public VehiclePassportRegister()

{

}

}