Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ

КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ И ПРОГРАММНОЙ ИНЖЕНЕРИИ (КАФЕДРА № 43)

Е.О. Пятлина, А.А.Ключарёв, Е.В. Павлов

Объектно-ориентированное проектирование программного обеспечения информационных систем средствами языка UML

Учебное пособие по дисциплине «Объектно-ориентированное проектирование Информационных систем»

Санкт-Петербург

2020

Составители: Пятлина Е.О., Ключарев А.А., Павлов Е.В.

Рецензент: Кафедра «Аэрокосмических компьютерных и программных систем» ГУАП.

Объектно-ориентированное проектирование программного обеспечения информационных систем средствами языка UML.

Учебное пособие предназначено для курсового проектирования по дисциплинам «Технология программирования» направления 09.03.01 «Информатика и вычислительная техника», «Объектно-ориентированное проектирование информационных систем» направлений 09.03.04 «Программная инженерия», 02.03.03 «Математическое обеспечение и администрирование информационных систем» - СПб ГУАП, 2020. – 49 с.

В учебное пособие включен теоретический материал по языку объектно-ориентированного проектирования программного обеспечения UML, необходимый для разработки структуры программного обеспечения информационных систем в рамках курсового проектирования, а также требования к содержанию курсового проекта, пример выполнения курсового проекта.

Подготовлены кафедрой компьютерных технологий и программной инженерии.

1. Общие сведения о методах объектно-ориентированного проектирования информационных систем

1.1. История создания и основные характеристики языка UML

В настоящее время унифицированный язык моделирования UML (англ. *Unified Modeling Language*) является одним из наиболее популярных инструментов разработки программных объектно-ориентированных систем. UML является языком широкого профиля, это — открытый стандарт, использующий графические обозначения для создания абстрактной модели системы, называемой *UML-моделью*. UML не является языком программирования, но на основании UML-моделей возможна генерация кода программ на различных языках высокого уровня.

Первые языки объектно-ориентированного моделирования начали появляться в середине 1970-х годов. В период между 1989 -1994 гг. общее число наиболее известных языков моделирования возросло до пяти десятков. К середине 1990-х наиболее известными методами проектирования, реализуемыми этими языками, становятся:

- Метод Гради Буча (Grady Booch), получивший условное название Booch или Booch'91, Booch Lite (позже - Booch'93)
- Метод Джеймса Румбаха (James Rumbaugh), названный Object Modeling Technique - OMT (позже - OMT-2)
- Метод Айвара Джекобсона (Ivar Jacobson), под названием Object-Oriented Software Engineering - OOSE

История развития языка UML берет начало с октября 1994 года, когда Гради Буч и Джеймс Румбах начали работу по унификации методов Booch и OMT. Их совместная работа была направлена на объединение достоинств известных объектно-ориентированных методов проектирования ПО. Проект так называемого унифицированного метода (Unified Method) версии 0.8 был подготовлен и опубликован в октябре 1995 года.

В этот период поддержка разработки языка UML становится одной из целей консорциума OMG (Object Management Group). Именно в OMG создается команда разработчиков, которая обеспечила дальнейшую работу по унификации и стандартизации языка UML. Усилия группы разработчиков, в которую входили также Г. Буч, Дж. Румбах и А. Джекобсон, привели к появлению первых документов, содержащих собственно описание языка UML версии 0.9 (июнь 1996 г.) и версии 0.91 (октябрь 1996 г.).

Тогда же некоторые компании и организации увидели в языке UML стратегический интерес для своего бизнеса. Компания Rational Software учредила консорциум партнеров UML, в который первоначально вошли такие фирмы, как Digital Equipment Corp., HP, i-Logix, Intellicorp, IBM, ICON Computing, MCI Systemhouse,

Microsoft, Oracle, Rational Software, TI и Unisys. Эти компании обеспечили поддержку последующей работы по более точному определению UML.

В 2005 г. язык UML версии 1.4.2 был принят в качестве международного стандарта ISO/IEC 19501:2005.

Последняя версия языка Unified Modeling Language (UML) Version 2.5 - Beta 2 опубликована на сайте www.omg.org.

В настоящее время на рынке CASE-средств представлены десятки программных инструментов, поддерживающих нотации языка UML, включая прямую и обратную генерацию кода программ, с наиболее распространенными языками и средами программирования, такими как MS Visual C++, Java, Object Pascal/Delphi, Power Builder, MS Visual Basic, Forte, Ada, Smalltalk.

Язык **UML** обладает следующими основными характеРисунок - тиками:

- является языком визуального моделирования, который обеспечивает разработку репрезентативных моделей для организации взаимодействия заказчика и разработчика ИС, а также различных групп разработчиков ИС;
- содержит механизмы расширения и специализации базовых концепций языка.

Пользователям языка предоставлены следующие возможности:

- строить модели на основе средств ядра языка, без использования механизмов расширения;
- добавлять при необходимости новые элементы и условные обозначения, если они не входят в ядро, или специализировать компоненты, систему условных обозначений (нотацию) и ограничения для конкретных предметных областей.

С каждым годом интерес к языку UML со стороны специалистов неуклонно возрастает. Первоначально UML был создан для определения, визуализации, проектирования и документирования, в основном, программных систем. Однако, использование UML не ограничивается моделированием программного обеспечения. Его также используют для моделирования бизнес-процессов, системного проектирования и отображения организационных структур предприятий.

3.4. Преимущества языка UML

1. UML объектно-ориентированный язык, в результате чего методы описания результатов анализа и проектирования семантически близки к методам программирования на современных объектно-ориентированных языках;

2. UML позволяет описать информационную систему практически со всех возможных точек зрения, включая разные аспекты поведения системы;

3. Диаграммы UML сравнительно просты для чтения после достаточно быстрого ознакомления с синтаксисом языка;

4. Сокращение числа возможных ошибок таких как: несогласованные параметры подпрограмм, несогласованное изменение атрибутов;

5. Повторное использование. Предполагается возможность многократного использования уже существующего проекта или его частей в новом проекте;

6. UML расширяет и позволяет вводить собственные текстовые и графические стереотипы, что способствует его применению не только в сфере программной инженерии;

7. UML получил широкое распространение и динамично развивается.

1.2. Основные диаграммы языка UML

Стандарт UML предлагает следующий набор диаграмм для построения модели информационной системы:

- ..***диаграмма вариантов использования*** (другое название - диаграмма сценариев) (use case diagram)– для моделирования бизнес-процессов организации или предприятия и определения требований к создаваемой информационной системе;
- ..***диаграмма классов*** (class diagram) – для моделирования статической структуры классов системы и связей между ними;
- ..***диаграмма поведения системы*** (behavior diagrams);
- ..***диаграмма взаимодействия*** (interaction diagrams);
- ..***диаграмма последовательности*** (sequence diagrams) – для моделирования процесса обмена сообщениями между объектами в рамках одного варианта использования с привязкой к оси времени;
- ..***диаграмма кооперации*** (collaboration diagram) – для моделирования процесса обмена сообщениями между объектами в рамках одного варианта использования без привязки к оси времени;
- ..***диаграмма состояний*** (statechart diagram) – для моделирования поведения объектов системы при переходе из одного состояния в другое;
- ..***диаграмма видов деятельности*** (activity diagram) – для моделирования поведения системы в рамках различных вариантов использования, или моделирования деятельностей;
- ..***диаграмма реализации*** (implementation diagrams):
- ..***диаграмма компонентов*** (component diagrams) – для моделирования иерархии программных компонентов (подсистем) информационной системы;

- ..***диаграмма развертывания*** (deployment diagram) – для моделирования физической архитектуры спроектированной информационной системы.

На Рисунке - 1 представлена интегрированная модель информационной системы, включающая основные диаграммы языка UML, которые должны быть разработаны в курсовом проекте.

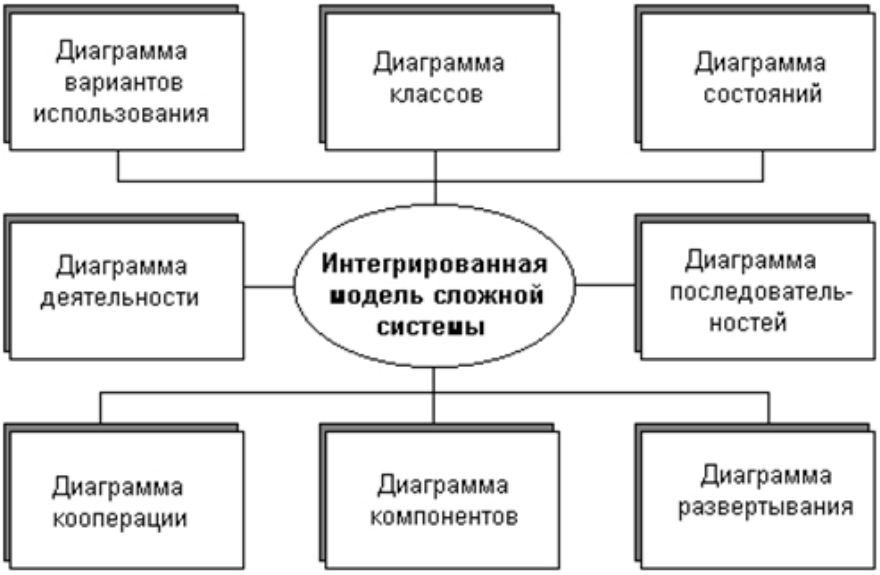


Рисунок - 1 Интегрированная модель информационной системы в нотации языка UML

1.3. Диаграмма вариантов использования

«Вариант использования» (или сценарий) представляет собой последовательность действий, выполняемых системой в ответ на событие, инициируемое некоторым внешним объектом (актером).
В простейшем случае вариант использования определяется в процессе обсуждения разработчика с пользователем тех функций, которые он хотел бы реализовать в данной информационной системе. На языке UML вариант использования изображают следующим образом:

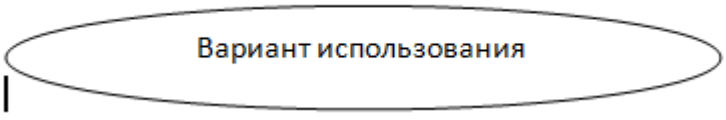


Рисунок - 2 Вариант использования

Актер (actor) – это роль, которую пользователь играет по отношению к информационной системе. Актеры представляют собой роли, а не конкретных людей или наименования работ. Несмотря на то, что на диаграммах вариантов использования они изображаются в виде стилизованных человеческих фигурок, актер может быть не только человеком-пользователем, но также внешней информационной системой, которая является

источником или приемником информации от данной системы. На языке UML актеры представляют в виде фигур:

Рисунок - 3 Действующее лицо (актер)

Актеры делятся на три основных типа:

- пользователи;
- другие информационные системы, взаимодействующие с данной ИС;
- время.

Время становится актером, если от него зависит запуск каких-либо событий в системе.

1.3.1. **Связи между вариантами использования и актерами**

В языке UML на диаграммах вариантов использования поддерживается несколько типов связей между элементами диаграммы:

- коммуникация (communication),
- включение (include),
- расширение (extend),
- обобщение (generalization).

Связь коммуникации – применительно к диаграммам вариантов использования отношение ассоциации может служить только для обозначения взаимодействия актера с вариантом использования. На языке UML связи коммуникации показывают с помощью однонаправленной ассоциации (сплошной линии).

Рисунок - 4 Пример связи коммуникации

Связь включения (include) **определяет** тот факт, что некоторый вариант использования содержит поведение, определенное в другом варианте использования .

Связь расширения (extend) определяет взаимосвязь одного варианта использования с другим вариантом использования, который задействуется первым не всегда, а только при выполнении некоторых дополнительных условий.

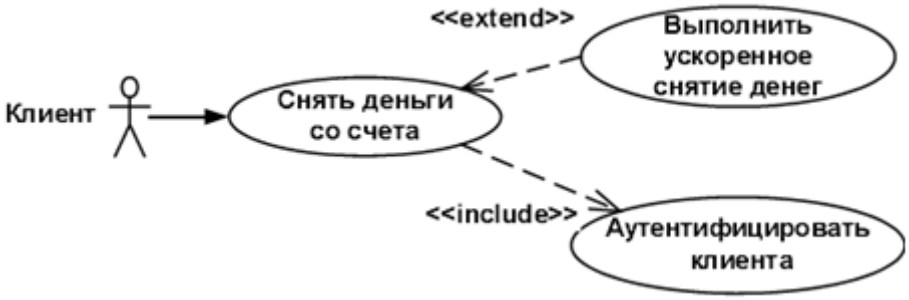


Рисунок - 5 Пример связи включения и расширения

Связь обобщения (generalization) отражает факт, что один элемент модели является специальным или частным случаем другого элемента модели

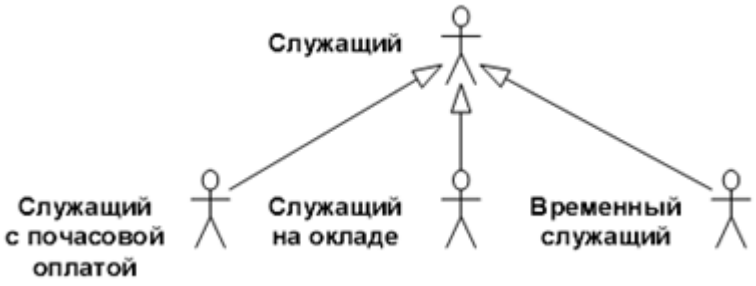


Рисунок - 6 Пример связи обобщения

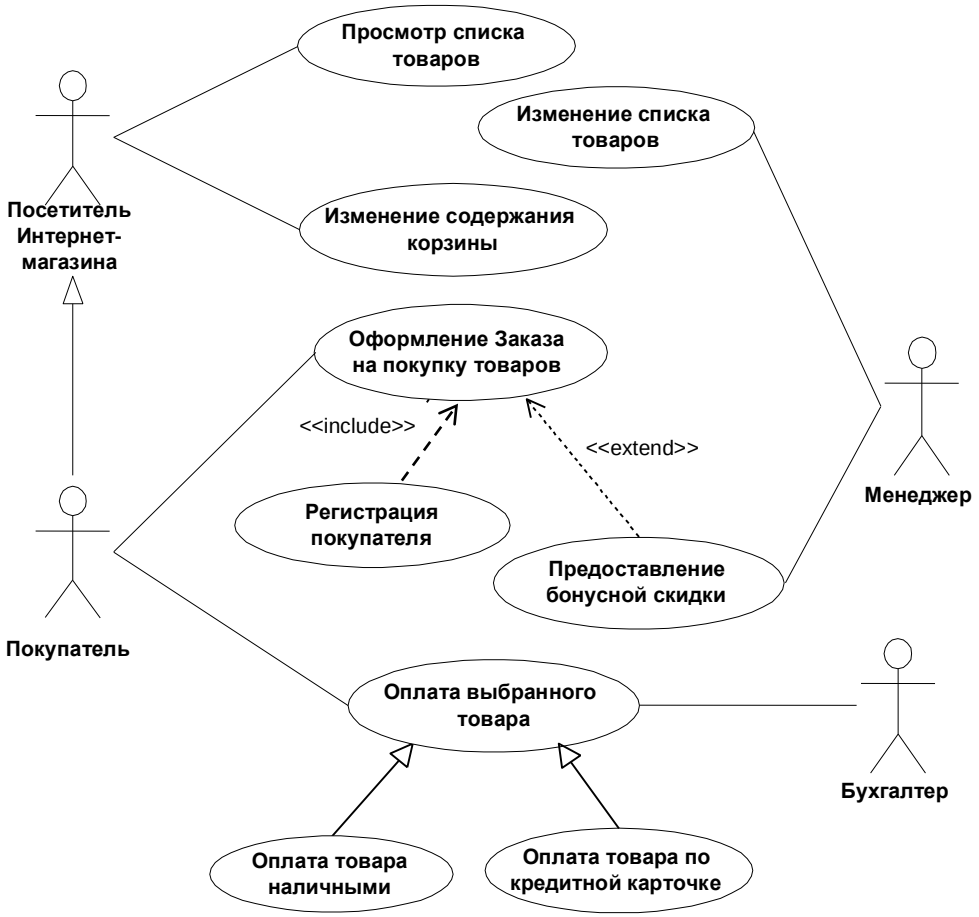


Рисунок - 7 Пример диаграммы вариантов использования

1.4. Диаграммы взаимодействия (interaction diagrams)

Диаграммы взаимодействия (interaction diagrams) описывают поведение взаимодействующих групп объектов. Как правило, диаграмма взаимодействия охватывает поведение объектов в рамках только одного варианта использования. На такой диаграмме отображается ряд объектов и те сообщения, которыми они обмениваются между собой.

Существует два вида диаграмм взаимодействия: ***диаграммы последовательности*** (sequence diagrams) и ***диаграммы кооперации*** (collaboration diagrams).

1.4.1. Диаграмма последовательности (sequence diagrams)

Диаграмма последовательности отражает события, происходящие в рамках одного варианта использования.

Все действующие лица (актеры, классы или объекты), участвующие в данном варианте использования (сценарии), показываются в верхней части диаграммы. Стрелки соответствуют сообщениям, передаваемым между актером и объектом или между объектами для выполнения требуемых функций.

На диаграмме последовательности объект или класс изображается в виде прямоугольника, от которого вниз проведена пунктирная вертикальная линия. Эта линия называется ***линией жизни (lifeline) объекта***. Она представляет собой фрагмент жизненного цикла объекта в процессе взаимодействия.

Каждое сообщение представляется в виде стрелки между линиями жизни двух объектов. Сообщения появляются в том порядке, как они показаны на диаграмме последовательности сверху вниз. Каждое сообщение помечается как минимум именем сообщения. При желании можно добавить также аргументы и некоторую управляющую информацию. Можно показать самоделегирование (self-delegation) – сообщение, которое объект посылает самому себе, при этом стрелка сообщения указывает на ту же самую линию жизни.

Временная последовательность сообщений дополнительно указывается путем нумерации сообщений. Ось времени на диаграмме расположена сверху вниз.

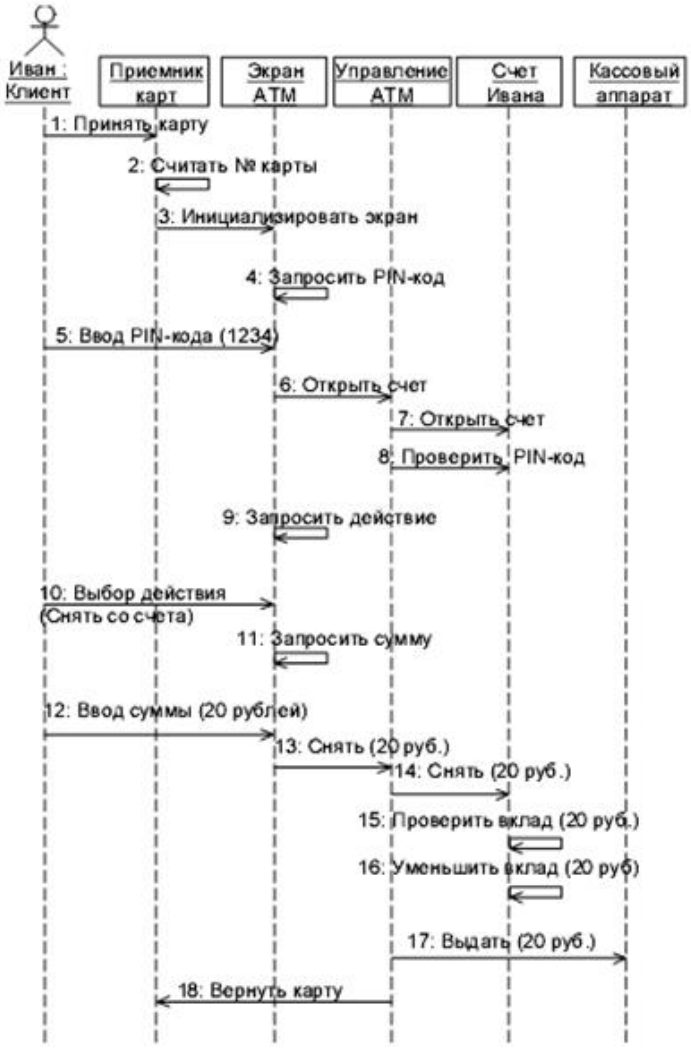


Рисунок - 8 Пример диаграммы последовательности

1.5. **Диаграмма кооперации (collaboration diagram)**

Диаграммы кооперации отображают события в рамках конкретного варианта использования (сценария).
Диаграммы кооперации больше внимания заостряют на связях между объектами. На диаграмме кооперации представлена вся та информация, которая есть и на диаграмме последовательности, но диаграмма кооперации по-другому описывает поток событий. Из нее легче понять связи, существующие между объектами.

На диаграмме кооперации так же, как и на диаграмме последовательности, стрелки обозначают сообщения, обмен которыми осуществляется в рамках данного варианта использования. Их временная последовательность указывается путем нумерации сообщений.

В отличие от диаграммы последовательности, на диаграмме кооперации отсутствует ось времени.

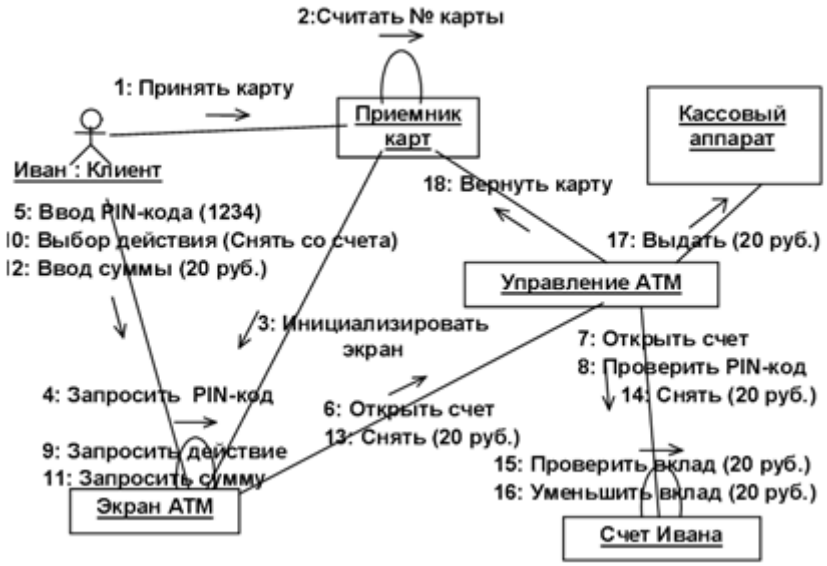


Рисунок - 9 Пример диаграммы кооперации

1.6. **Диаграмма классов**

Диаграмма классов (class diagram) —предназначена для представления модели статической структуры информационной системы в терминологии классов ООП.

Диаграмма классов представляет собой **граф,** вершинами или узлами которого являются элементы типа "**класс**", которые связаны различными типами структурных **отношений.**
На диаграмме классов могут изображаться следующие элементы:

- Класс (class) - описание общих свойств группы сходных объектов;
- Пакет (package) - набор элементов модели, логически связанных между собой;
- Интерфейс (interface) - абстрактный класс, задающий набор операций, которые объект произвольного класса, связанного с данным интерфейсом, предоставляет другим объектам.

Класс - это группа сущностей (объектов), обладающих сходными свойствами, а именно, данными и поведением. Отдельный представитель некоторого класса называется объектом класса или просто объектом.

Под поведением объекта в UML понимаются любые правила взаимодействия объекта с внешним миром и с данными самого объекта.

На диаграммах класс изображается в виде прямоугольника со сплошной границей, разделенного горизонтальными линиями на 3 секции:

Верхняя секция (секция имени) содержит имя класса и другие общие свойства. Например, тип класса (необязательное поле), имя автора и т.п.(необязательное поле). Примерами имен классов могут быть такие существительные, как «Сотрудник», «Компания», «Руководитель», «Клиент», «Продавец», «Менеджер», «Офис» и многие

другие, имеющие непосредственное отношение к моделируемой предметной области и функциональному назначению проектируемой системы.

Классы могут объединяться в более крупные компоненты, называемые пакетами. Поэтому в имени класса может быть указано имя пакета, к которому он принадлежит:

<имя пакета>:: <имя класса>

В средней секции класса содержится список атрибутов данного класса. В нижней - список операций класса, отражающих его поведение (действия, выполняемые классом). Любая из секций атрибутов и операций (а также обе сразу) может отсутствовать. На усмотрение конкретной реализации могут быть введены дополнительные секции, например, исключения.

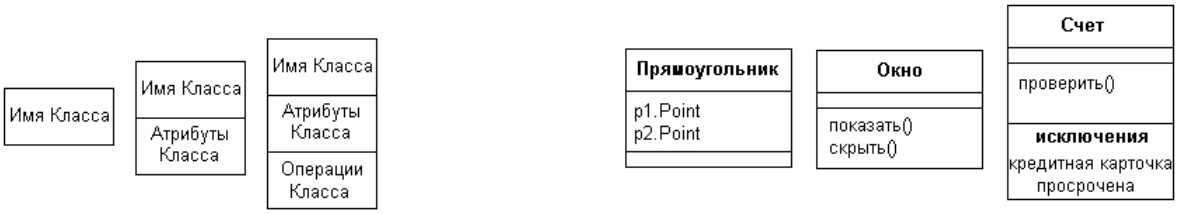


Рисунок - 10 Примеры изображения классов

Атрибут – это элемент информации, связанный с классом. Атрибуты хранят инкапсулированные данные класса.

Каждому атрибуту класса соответствует отдельная строка текста, которая состоит из квантора видимости атрибута, имени атрибута, его кратности, типа значений атрибута и, возможно, его исходного значения. Общий формат записи отдельного атрибута класса следующий:

<квантор видимости> <имя атрибута> [кратность] : <тип атрибута> = <исходное значение> {строка-свойство}

У атрибута можно определить четыре возможных значения квантора видимости:

Public (общий, открытый). Это значение видимости предполагает, что атрибут будет виден всеми остальными классами. Любой класс может просмотреть или изменить значение атрибута. В соответствии с нотацией UML общему атрибуту предшествует знак « + ».

Private (закрытый, секретный). Соответствующий атрибут не виден никаким другим классом. Закрытый атрибут обозначается знаком « – » в соответствии с нотацией UML.

Protected (защищенный). Такой атрибут доступен только самому классу и его потомкам. Нотация UML для защищенного атрибута – это знак « # ».

Package or Implementation (пакетный). Предполагает, что данный атрибут является общим, но только в пределах его пакета.
Этот тип видимости не обозначается никаким специальным значком.

С помощью закрытости или защищенности удается избежать ситуации, когда значение атрибута изменяется всеми классами системы. Вместо этого логика изменения атрибута будет заключена в том же классе, что и сам этот атрибут. Задаваемые параметры видимости повлияют на генерируемый код. В общем случае, атрибуты рекомендуется делать закрытыми или защищенными. Это позволяет лучше контролировать сам атрибут и код. Квантор видимости может быть опущен. Его отсутствие означает, что видимость атрибута не указывается.

Рисунок - 11 Пример обозначения видимости на диаграмме классов

Операции реализуют связанное с классом поведение. Операция включает три части – имя, параметры и тип возвращаемого значения.

Параметры – это аргументы, получаемые операцией «на входе». Тип возвращаемого значения относится к результату действия операции.

На диаграмме классов можно показывать как имена операций, так и имена операций вместе с их параметрами и типом возвращаемого значения. Чтобы уменьшить загруженность диаграммы, полезно бывает на некоторых из них показывать только имена операций, а на других их полную сигнатуру.

В языке UML операции имеют следующую нотацию:

Имя Операции (аргумент: тип данных аргумента, аргумент2:тип данных аргумента2,...): тип возвращаемого значения

Связь представляет собой семантическую взаимосвязь между классами. Она дает классу возможность узнавать об атрибутах, операциях и связях другого класса. Иными словами, чтобы один класс мог послать сообщение другому на диаграмме

последовательности или кооперативной диаграмме, между ними должна существовать связь.

Существуют четыре типа связей, которые могут быть установлены между классами: ассоциации, зависимости, агрегации и обобщения.

Связь ассоциация (association) – это семантическая связь между классами. Их Рисунок - уют на диаграмме классов в виде обыкновенной линии.

Рисунок - 12 Связь ассоциация

Ассоциации могут быть двунаправленными, как в примере, или однонаправленными. На языке UML двунаправленные ассоциации Рисунок - уют в виде простой линии без стрелок или со стрелками с обеих ее сторон. На однонаправленной ассоциации изображают только одну стрелку, показывающую ее направление.

Направление ассоциации можно определить, изучая диаграммы последовательности и кооперативные диаграммы. Если все сообщения на них отправляются только одним классом и принимаются только другим классом, но не наоборот, между этими классами имеет место однонаправленная связь. Если хотя бы одно сообщение отправляется в обратную сторону, ассоциация должна быть двунаправленной.

Связи зависимости (dependency) также отражают связь между классами, но они всегда однонаправлены и показывают, что один класс зависит от определений, сделанных в другом. Например, класс А использует методы класса В. Тогда при изменении класса В необходимо произвести соответствующие изменения в классе А.

Зависимость изображается пунктирной линией, проведенной между двумя элементами диаграммы, и считается, что элемент, привязанный к концу стрелки, зависит от элемента, привязанного к началу этой стрелки.

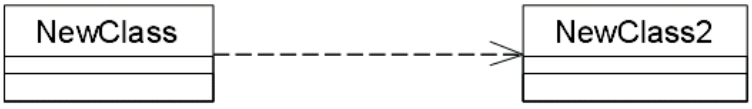


Рисунок - 13 Связь зависимость

При генерации кода для этих классов к ним не будут добавляться новые атрибуты. Однако, будут созданы специфические для языка операторы, необходимые для поддержки связи.

Связь агрегация (aggregations) представляют собой более тесную форму ассоциации. Агрегация – это связь между целым и его частью. Например, у вас может быть класс Автомобиль, а также классы Двигатель, Покрышки и классы для других частей автомобиля. В результате объект класса Автомобиль будет состоять из объекта класса Двигатель, четырех объектов Покрышек и т. д. Агрегации визуализируют в виде линии с ромбиком у класса, являющегося целым:

Рисунок - 14 Связь агрегация

В дополнение к простой агрегации UML вводит более сильную разновидность агрегации, называемую композицией. Согласно композиции, объект-часть может принадлежать только единственному целому, и, кроме того, как правило, жизненный цикл частей совпадает с циклом целого: они живут и умирают вместе с ним. Любое удаление целого распространяется на его части.

Связь обобщение (наследование) - это отношение типа общее-частное между элементами модели. С помощью обобщений (generalization) показывают связи наследования между двумя классами. Большинство объектно-ориентированных языков непосредственно поддерживают концепцию наследования. Она позволяет одному классу наследовать все атрибуты, операции и связи другого. Наследование пакетов означает, что в пакете-наследнике все сущности пакета-предка будут видны под своими собственными именами (т.е. пространства имен объединяются). Наследование показывается сплошной линией, идущей от класса-потомка к классу-предку (в терминологии ООП - от потомка к предку, от сына к отцу, или от подкласса к суперклассу). Со стороны более общего элемента Рисунок - уется большой полый треугольник.

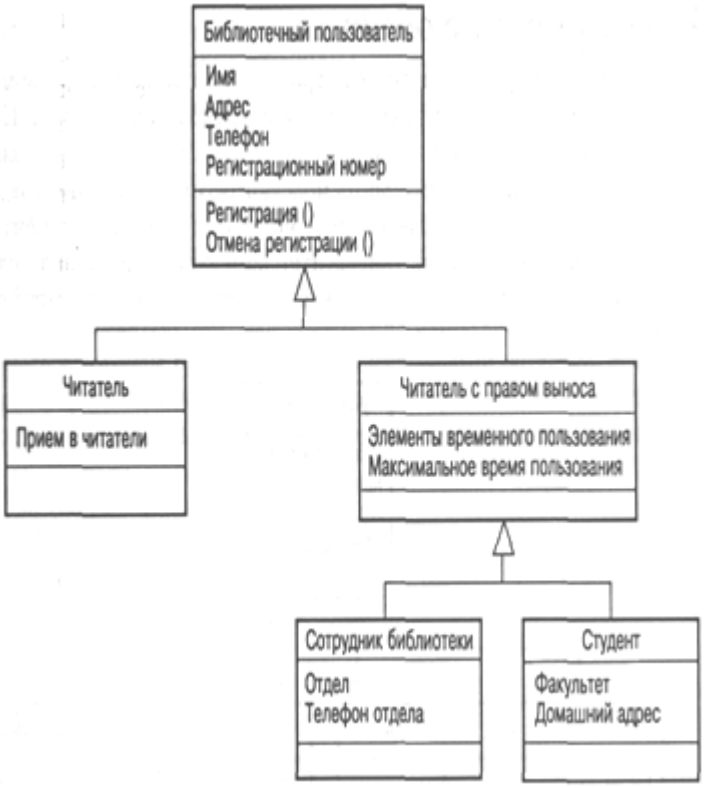


Рисунок - 15 Пример связи наследование

Помимо наследуемых, каждый подкласс имеет свои собственные уникальные атрибуты, операции и связи.

Множественность связи (multiplicity) показывает, сколько экземпляров одного класса взаимодействуют с помощью этой связи с одним экземпляром другого класса в данный момент времени.

В языке UML приняты определенные нотации для обозначения множественности.

Таблица 1 - Обозначения множественности связей в UML

Множественность	Значение
0..*	Ноль или больше
1..*	Один или больше
0..1	Ноль или один
1..1 (сокращенная запись: 1)	Ровно один

Имена связей. Связи можно уточнить с помощью имен связей или ролевых имен. ***Имя связи*** – это обычно глагол или глагольная фраза, описывающая, зачем она нужна. Например, между классом Person (человек) и классом Company (компания) может существовать ассоциация. Можно задать в связи с этим

вопрос, является ли объект класса Person клиентом компании, её сотрудником или владельцем? Чтобы определить это, ассоциацию можно назвать «employs» (нанимает):

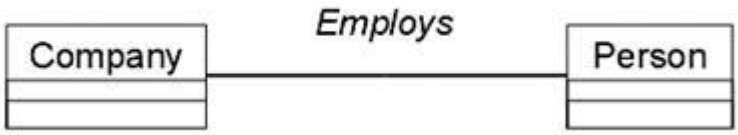


Рисунок - 16 Пример имен связей

Рисунок - 17 Пример диаграммы классов

1.7. Диаграмма пакетов

Пакет - это вместилище для некоторого набора классов и других пакетов. Пакет является самостоятельным пространством имен.

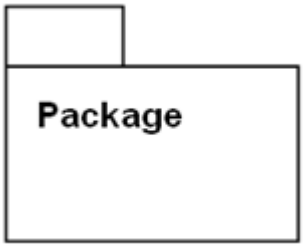


Рисунок - 18 Обозначение пакета в UML

В UML нет каких-либо ограничений на правила, по которым разработчики могут или должны группировать классы в пакеты. Но есть некоторые стандартные случаи, когда такая группировка уместна, например, тесно взаимодействующие классы, или более общий случай - разбиение системы на подсистемы.

Пакет физически содержит сущности, определенные в нем (говорят, что "сущности принадлежат пакету"). Это означает, что если будет уничтожен пакет, то будут уничтожено и все его содержимое.

Существует несколько наиболее распространенных подходов к группировке классов в пакеты.

Во-первых, можно группировать классы по стереотипу. В таком случае получается один пакет с классами-сущностями, один с граничными классами, один с управляющими классами и т.д. Этот подход может быть полезен с точки зрения размещения готовой системы, поскольку все находящиеся на клиентских машинах пограничные классы уже оказываются в одном пакете.

Другой подход заключается в объединении классов по их функциональности. Например, в пакете Security (безопасность) содержатся все классы, отвечающие за безопасность приложения. В таком случае другие пакеты могут называться Employee Maintenance (Работа с сотрудниками), Reporting (Подготовка отчетов) и Error Handling (Обработка ошибок). Преимущество этого подхода заключается в возможности повторного использования.

Зависимость между двумя пакетами существует в том случае, если между любыми двумя классами в пакетах существует любая зависимость.

Таким образом, диаграмма пакетов представляет собой диаграмму, содержащую пакеты классов и зависимости между ними. Строго говоря, пакеты и зависимости являются элементами диаграммы классов, то есть диаграмма пакетов – это форма диаграммы классов.

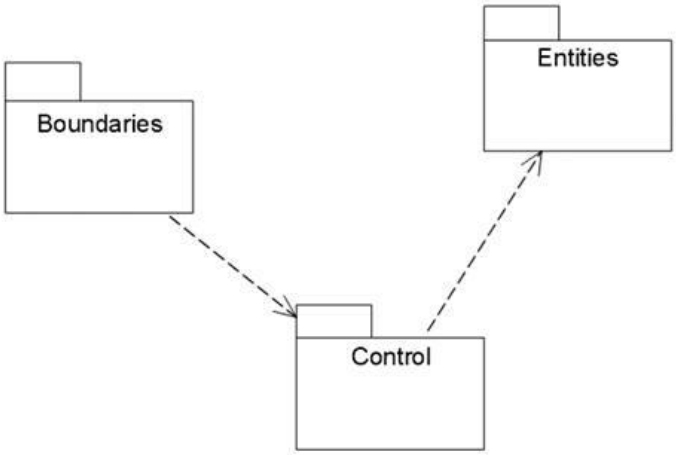


Рисунок - 19 Пример диаграммы пакетов

Зависимость между двумя элементами имеет место в том случае, если изменения в определении одного элемента могут повлечь за собой изменения в другом. Что касается классов, то причины для зависимостей могут быть самыми разными:

- один класс посылает сообщение другому;
- один класс включает часть данных другого класса;
- один класс использует другой в качестве параметра операции.

Диаграммы пакетов можно считать основным средством управления общей структурой системы. Пакеты являются жизненно необходимым средством для больших проектов. Их следует использовать в тех случаях, когда диаграмма классов, охватывающая всю систему в целом и размещенная на единственном листе бумаги формата А4, становится нечитаемой.

1.8. **Диаграмма состояний**

Диаграммы состояний определяют все возможные состояния, в которых может находиться конкретный объект, а также процесс смены состояний объекта в результате наступления некоторых событий.

Существует много форм диаграмм состояний, незначительно отличающихся друг от друга семантикой.

На диаграмме имеются два специальных состояния – начальное (start) и конечное (stop). Начальное состояние выделено черной точкой, оно соответствует состоянию объекта, когда он только что был создан. Конечное состояние обозначается черной точкой в белом кружке, оно соответствует состоянию объекта непосредственно перед его уничтожением. На диаграмме состояний может быть одно и только одно начальное состояние. В то же время, может быть столько конечных состояний, сколько вам нужно, или их может не быть вообще. Когда объект

находится в каком-то конкретном состоянии, могут выполняться различные процессы. Процессы, происходящие, когда
объект находится в определенном состоянии, называются действиями (actions).

С состоянием можно связывать данные пяти типов: деятельность, входное действие, выходное действие, событие и история состояния.

Деятельностью (activity) называется поведение, реализуемое объектом, пока он находится в данном состоянии. Деятельность
– это прерываемое поведение. Оно может
выполняться до своего завершения, пока объект находится в данном состоянии, или может быть прервано переходом объекта в другое состояние. Деятельность изображают внутри самого состояния, ей должно предшествовать слово do (делать) и двоеточие.

Входным действием (entry action) называется поведение, которое выполняется, когда объект переходит в данное состояние. Данное действие осуществляется не после того, как объект перешел в это состояние, а, скорее, как часть
этого перехода. В отличие от деятельности, входное действие рассматривается как непрерываемое. Входное действие также показывают внутри состояния, ему предшествует слово entry (вход) и двоеточие.

Выходное действие (exit action) подобно входному. Однако, оно осуществляется как составная часть процесса выхода из данного состояния. Оно является частью процесса такого перехода. Как и входное, выходное действие является непрерываемым. Выходное действие изображают внутри состояния, ему предшествует слово exit (выход) и двоеточие.

Деятельность может также выполняться в результате получения объектом некоторого события. При получении некоторого события выполняется определенная деятельность.

Переходом (Transition) называется перемещение из одного состояния в другое. Совокупность переходов диаграммы показывает, как объект может перемещаться между своими состояниями. На диаграмме все переходы изображают в виде стрелки, начинающейся на первоначальном состоянии и заканчивающейся последующим.

Переходы могут быть рефлексивными. Объект может перейти в то же состояние, в котором он в настоящий момент находится. Рефлексивные переходы изображают в виде стрелки, начинающейся и завершающейся на одном и том же состоянии.

У перехода существует несколько спецификаций. Они включают события, аргументы, ограждающие условия, действия и посылаемые события.

Событие (event) – это то, что вызывает переход из одного состояния в другое. Событие размещают на диаграмме вдоль линии перехода.

На диаграмме для отображения события можно использовать как имя операции, так и обычную фразу.

Большинство переходов должны иметь события, так как именно они, прежде всего, заставляют переход осуществиться. Тем не менее, бывают и автоматические переходы, не имеющие событий. При этом объект сам перемещается из одного состояния в другое со скоростью, позволяющей осуществиться входным действиям, деятельности и выходным действиям.

Ограждающие условия (guard conditions) определяют, когда переход может, а когда не может осуществиться. В противном случае переход не осуществится.

Ограждающие условия изображают на диаграмме вдоль линии перехода после имени события, заключая их в квадратные скобки.

Ограждающие условия задавать необязательно. Однако если существует несколько автоматических переходов из состояния, необходимо определить для них взаимно исключающие ограждающие условия. Это поможет читателю диаграммы понять, какой путь перехода будет автоматически выбран.

Действием (action), как уже говорилось, является непрерываемое поведение, осуществляющееся как часть перехода. Входные и выходные действия показывают внутри состояний, поскольку они определяют, что происходит, когда объект входит или выходит из него. Большую часть действий, однако, изображают вдоль линии перехода, так как они не должны осуществляться при входе или выходе из состояния.

Действие Рисунок - уют вдоль линии перехода после имени события, ему предшествует косая черта.

Событие или действие могут быть поведением внутри объекта, а могут представлять собой сообщение, посылаемое другому объекту. Если событие или действие посылается другому объекту, перед ним на диаграмме помещают знак « ^ ».

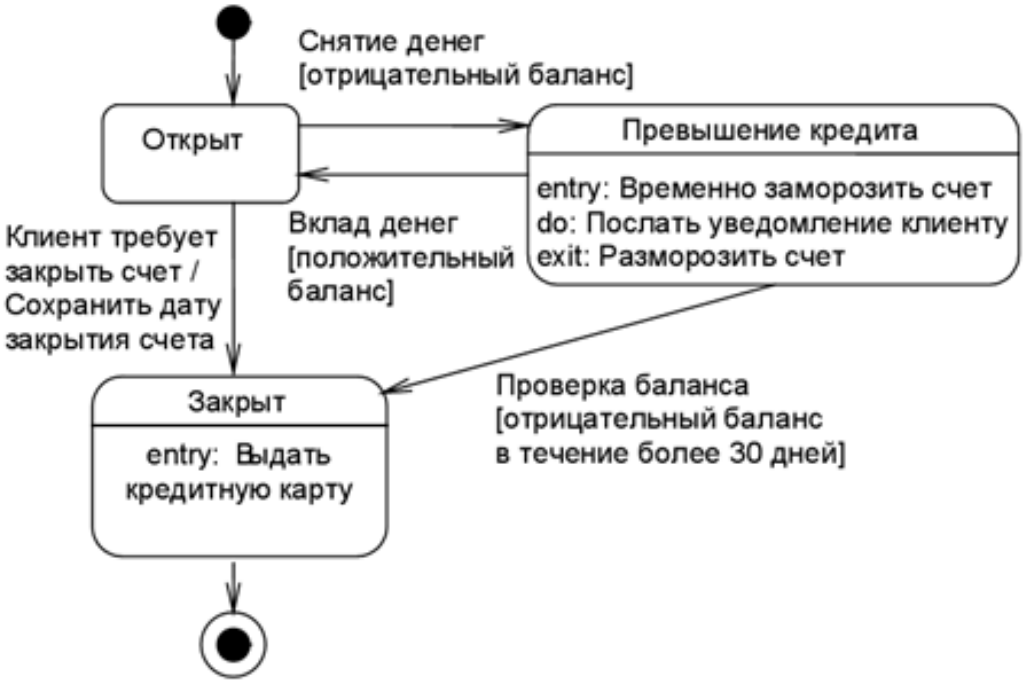


Рисунок - 20 Пример диаграммы состояний

Диаграммы состояний не надо создавать для каждого класса, они применяются только в сложных случаях. Если объект класса может существовать в нескольких состояниях и в каждом из них ведет себя по-разному, для него может потребоваться такая диаграмма.

1.9. **Диаграммы размещения**

Диаграмма размещения (deployment diagram) отражает физические взаимосвязи между программными и аппаратными компонентами системы. Она является хорошим средством для того, чтобы показать маршруты перемещения объектов и компонентов в распределенной системе.

Каждый узел на диаграмме размещения представляет собой некоторый тип вычислительного устройства – в большинстве случаев, часть аппаратуры. Эта аппаратура может быть простым устройством или датчиком.

Диаграмма размещения показывает физическое расположение сети и местонахождение в ней различных компонентов.

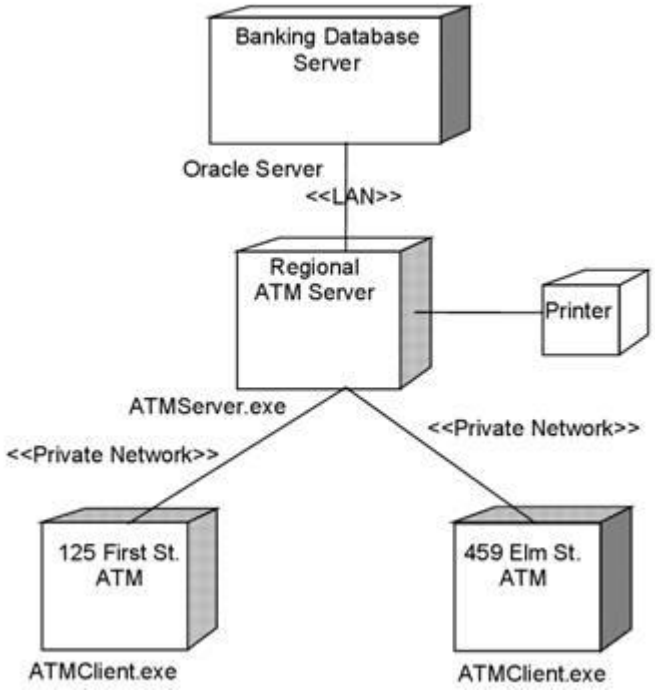


Рисунок - 21 Пример диаграммы размещения

Диаграмма размещения используется менеджером проекта, пользователями, архитектором системы и эксплуатационным персоналом,
чтобы понять физическое размещение системы и расположение её отдельных подсистем.

1.10. **Диаграммы компонентов**

Диаграммы компонентов показывают, как выглядит модель на физическом уровне. На них изображены компоненты программного обеспечения и связи между ними. При этом на такой диаграмме выделяют два типа компонентов: исполняемые компоненты и библиотеки кода.

Каждый класс модели (или подсистема) преобразуется в компонент исходного кода. После создания они сразу добавляются к диаграмме компонентов. Между отдельными компонентами изображают зависимости,
соответствующие зависимостям на этапе компиляции или выполнения программы.

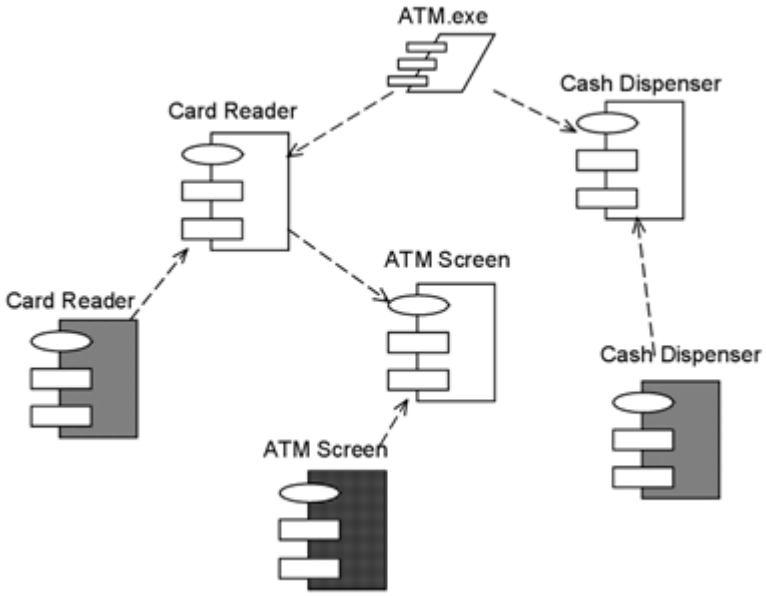


Рисунок -22 Пример диаграммы компонентов

Диаграммы компонентов применяются теми участниками проекта, кто отвечает за компиляцию системы. Из нее видно, в каком порядке надо компилировать компоненты, а также какие исполняемые компоненты будут созданы системой. На такой диаграмме показано соответствие классов реализованным компонентам. Она нужна там, где начинается генерация кода.

В некоторых случаях допускается размещать диаграмму компонентов на диаграмме развертывания. Это позволяет показать какие компоненты выполняются и на каких узлах.

2. Количественная оценка информативности диаграмм UML

2.1 Расчет оценок

Оценка UML диаграммы производится по формуле: $S = \frac{\sum S_{obj} + \sum S_{link}}{1 + Obj + \sqrt{T_{obj} + T_{link}}}$, где

S – оценка диаграммы,

S_{obj} – оценка элементов диаграммы,

S_{link} – оценка связей на диаграмме,

Obj – количество объектов на диаграмме,

T_{obj} – количество типов объектов,

T_{link} – количество типов связей.

Если диаграмма содержит большое число связей одного типа, то их можно не учитывать. В этом случае формула упрощается:

$$S = \frac{\sum S_{obj}}{1 + Obj + \sqrt{T_{obj}}}$$

Если на диаграмме класса указываются для каждого класса атрибуты и операции, то к оценке класса добавляется следующая составляющая:

$S_{cls} = \frac{\sqrt{Op} + \sqrt{Atr}}{0.3 \cdot (Op + Atr)}$, где

Op – количество операций класса,

Atr – количество атрибутов класса.

Таблица 2 **Оценки основных элементовUML**

Тип элемента	**Оценка элемента**
Класс class	5
Интерфейс interface	4
Сценарий use-case	2
Компонент component	4
Узел node	3
Взаимодействие interaction	6
Пакет package	4
Состояние state	4
Примечание note	2

Таблица 3 **Оценки основных типов связей**

Тип связи	**Оценка связи**
Зависимость dependency	2

Ассоциация association	1
Агрегирование aggregation	2
Композиция composition	3
Обобщение generalization	3
Реализация realization	2

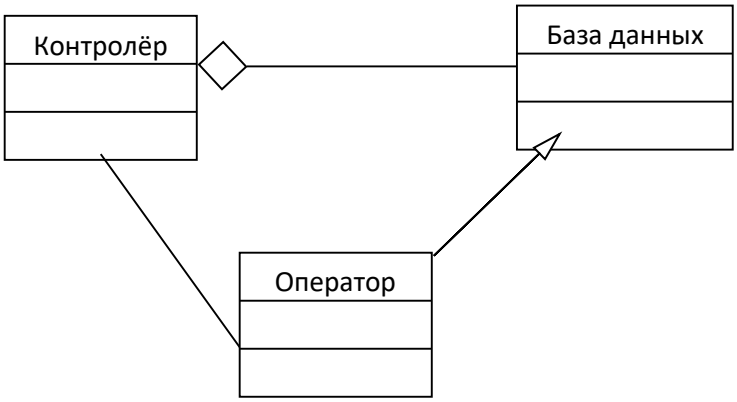
Остальные типы связей рассматриваются как ассоциации.

Оценка диаграммы должна попадать в оптимальный диапазон, иначе диаграмма или слишком краткая, или перегружена информацией.

Таблица 4 **Диапазоны оптимальных оценок диаграмм.**

Название диаграммы	Диапазон оптимальных оценок
Class – диаграмма классов с атрибутами и операциями	5-5,5
Class - диаграмма классов без атрибутов и операций	3-3,5
Component – диаграмма компонентов	3,5-4
Use case - диаграмма вариантов использования	2,5-3
Deployment - диаграмма развёртывания	2-2,5
Sequences - диаграмма последовательности	3-3,5
Cooperative - диаграмма кооперации	3,5-4
Package - диаграмма пакетов	3,5-4
State – диаграмма состояния	2,5-3

2.2. Примеры расчета оценок:



$$S = \frac{\sum S_{obj} + \sum S_{link}}{1 + Obj + \sqrt{T_{obj} + T_{link}}} = \frac{3 \cdot 5 + 6}{1 + 3 + \sqrt{1 + 3}} = 3.5$$

Рисунок – 23 Пример диаграммы классов без атрибутов и операций.

2.2.2. Диаграмма классов с атрибутами и операциями.

Диаграмма классов ИС «Интернет-магазин электроники»

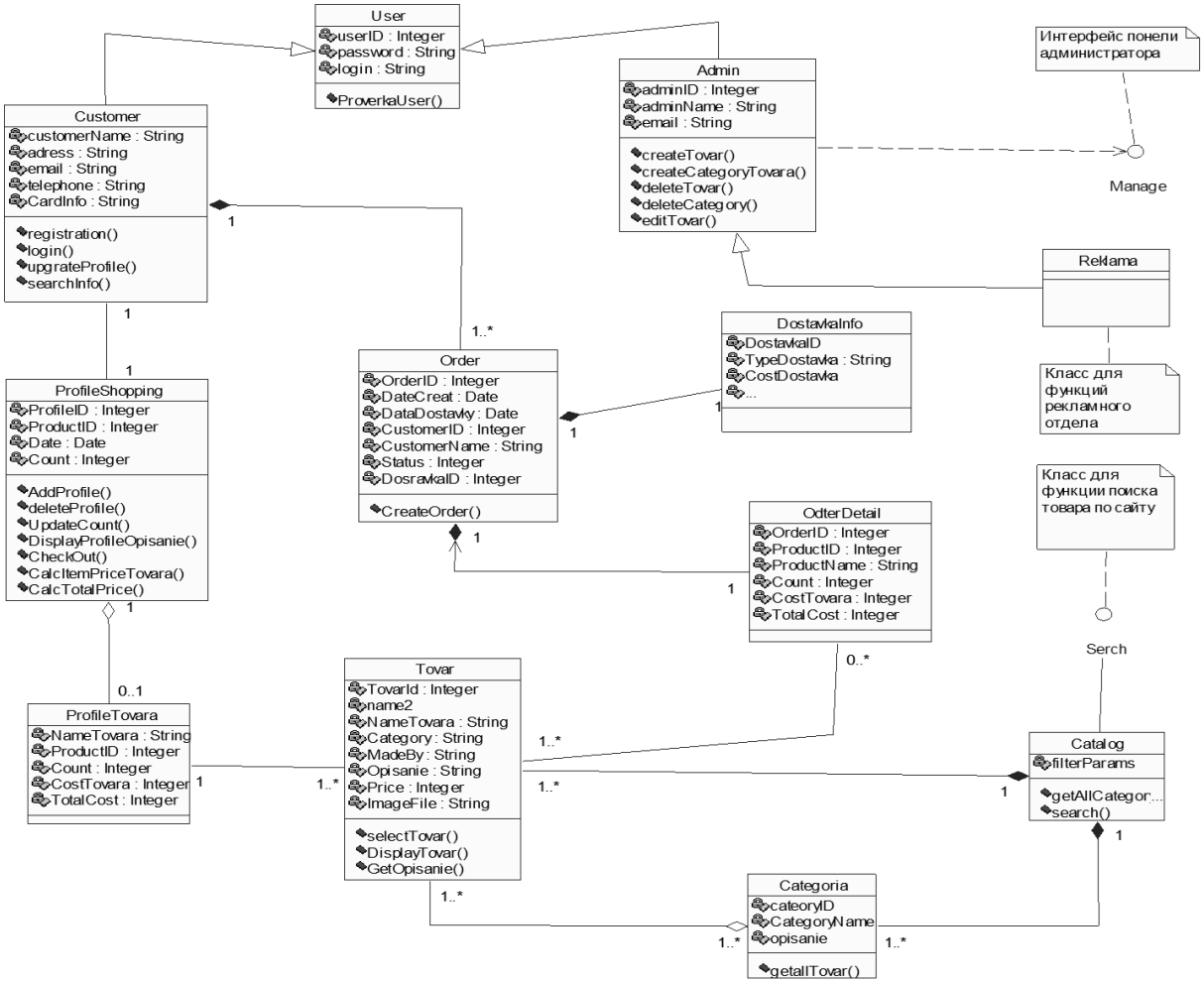


Рисунок – 24 Пример расчета оценки диаграммы классов с атрибутами и операциями

$$\boldsymbol{S_{diagr} = \frac{\sum S_{obj} + \sum S_{link} + S_{clas}}{1 + Obj + \sqrt{T_{obj} + T_{link}}}}$$

где S_{obj}-оценка элемента на диаграмме, S_{link}- оценка связей, Obj- кол-во объектов на диаграмме, T_{obj} –количество типов объектов, T_{link}- количество типов связи.

$$S_{clas} = \frac{\sqrt{Op}+\sqrt{Atr}}{0.3*(Op+Atr)},$$

где Op- количество операций, Atr- количество атрибутов.

Расчет оценки диаграммы классов с атрибутами и операциями:

$$S_{\text{diagr}} = \frac{(12*5+3*2+2*4)+(3*3+5*3+2*2+4*1+4*2)+19{,}62}{1+17+\sqrt{3+5}} = 6{,}42$$

$$S_{(user)} = \frac{\sqrt{3}+\sqrt{1}}{0.3*(3+1)} = 2{,}28$$

$$S_{(customer)} = \frac{\sqrt{5} + \sqrt{4}}{0.3 * (5 + 4)} = 1{,}57$$

$$S_{(ProfileShopping)} = \frac{\sqrt{4} + \sqrt{7}}{0.3 * (4 + 7)} = 1{,}41$$

$$S_{(ProfileTovara)} = \frac{\sqrt{5} + \sqrt{0}}{0.3 * (5 + 0)} = 1{,}49$$

$$S_{(Order)} = \frac{\sqrt{7} + \sqrt{1}}{0.3 * (7 + 1)} = 1{,}52$$

$$S_{(Tovar)} = \frac{\sqrt{8} + \sqrt{3}}{0.3 * (8 + 3)} = 1{,}38$$

$$S_{(Admin)} = \frac{\sqrt{3} + \sqrt{5}}{0.3 * (3 + 5)} = 1{,}65$$

$$S_{(DostavkaInfo)} = \frac{\sqrt{4} + \sqrt{1}}{0.3 * (4 + 1)} = 2$$

$$S_{(OdterDetail)} = \frac{\sqrt{6} + \sqrt{0}}{0.3 * (6 + 0)} = 1{,}36$$

$$S_{(Categoria)} = \frac{\sqrt{3} + \sqrt{1}}{0.3 * (3 + 1)} = 2{,}28$$

$$S_{(Reklama)} = \frac{\sqrt{0} + \sqrt{0}}{0.3 * (0 + 0)} = 0$$

$$S_{(Catalog)} = \frac{\sqrt{1} + \sqrt{2}}{0.3 * (1 + 2)} = 2{,}68$$

Оценка диаграммы не попадает в оптимальный диапазон (5 - 5.5), следовательно, можно сделать вывод, что диаграмма перегружена.

Если оценка не попадает в рекомендуемый диапазон, нужно диаграмму доработать, либо добавить информацию, либо убрать лишнюю, либо разбить диаграмму на части.

3. CASE-средства для проектирования программного обеспечения

Аббревиатура CASE расшифровывается как Computer Aided Software Engineering(Автоматизированная Разработка Программного Обеспечения). CASE средства подразумевают процесс разработки сложных ИС в целом: создание и

сопровождение ИС, анализ, формулировка требований, проектирование прикладного ПО и баз данных, генерацию кода, тестирование, документирование, обеспечение качества, конфигурационное управление и управление проектом, а также другие процессы. Таким образом, CASE-технологии образуют целую среду разработки ИС.

Большинство существующих CASE-средств основано на методологиях структурного или объектно-ориентированного анализа и проектирования, использующих спецификации в виде диаграмм или текстов для описания внешних требований, связей между моделями системы, динамики поведения системы и архитектуры программных средств. Главные составляющие CASE-продукта таковы:

- методология (Method Diagrams), которая задает единый графический язык и правила работы с ним.

- графические редакторы (Graphic Editors), которые помогают Рисунок - овать диаграммы; возникли с распространением PC и GUI, так называемых «upper case технологий

- генератор: по графическому представлению модели можно сгенерировать исходный код для различных платформ (так называемая low case часть CASE-технологии).

- репозиторий, своеобразная база данных для хранения результатов работы программистов.

Разумеется, существуют свои недостатки применения технологий, значимыми являются недостатки со стороны аспектов бизнеса:

- CASE-средства не обязательно дают немедленный эффект; он может быть получен только спустя какое-то время;

- реальные затраты на внедрение CASE-средств обычно намного превышают затраты на их приобретение;

- CASE-средства обеспечивают возможности для получения существенной выгоды только после успешного завершения процесса их внедрения.

Ряд преимуществ созданной системы:

- высокий уровень технологической поддержки процессов разработки и сопровождения ПО;

- положительное воздействие на некоторые или все из перечисленных факторов: производительность, качество продукции, соблюдение стандартов, документирование;

- приемлемый уровень отдачи от инвестиций в CASE-средства.

Rational Rose - CASE-средство фирмы Rational Software Corporation - предназначено для автоматизации этапов анализа и проектирования ПО, а также для генерации кодов на различных языках и выпуска проектной документации. Rational Rose использует

синтез-методологию объектно-ориентированного анализа и проектирования, основанную на подходах трех ведущих специалистов в данной области: Буча, Рамбо и Джекобсона. Разработанная ими универсальная нотация для моделирования объектов (UML - Unified Modeling Language) претендует на роль стандарта в области объектно-ориентированного анализа и проектирования. Конкретный вариант Rational Rose определяется языком, на котором генерируются коды программ (C++, Smalltalk, PowerBuilder, Ada, SQLWindows и ObjectPro). Основной вариант - Rational Rose/C++ - позволяет разрабатывать проектную документацию в виде диаграмм и спецификаций, а также генерировать программные коды на C++. Кроме того, Rational Rose содержит средства реинжиниринга программ, обеспечивающие повторное использование программных компонент в новых проектах.

4. Методические указания к курсовой работе

5. Цель работы:

Целью курсовой работы является ознакомление с основными элементами проектирования больших программных комплексов средствами языка UML, получение навыков по применению данных элементов для построения объектно-ориентированных моделей информационных систем , автоматическая генерация кодов программ на основе спроектированных диаграмм классов.

6. Задание по курсовой работе

Разработать проект программного обеспечения (ПО) заданной информационной системы (ИС), включающий основные диаграммы языка UML.

Список использованных источников

1. Технология программирования: учебник/ Г. С. Иванова. - М.: КноРус, 2011. - 333 с.

2. Вендров А. М. Проектирование программного обеспечения экономических информационных систем: Учебник/ А. М. Вендров. - М.: Финансы и статистика, 2000. - 347 с.

3. Калянов Г. Н. CASE-технологии : Консалтинг в автоматизации бизнес-процессов: учебное пособие/ Г. Н. Калянов. - 2-е изд., перераб. и доп. - М.: Горячая линия - Телеком, 2000. - 317 с.

4. Леоненков А. В. Объектно-ориентированный анализ и проектирование с использованием UML и IBM Rational Rose [Текст] : учебное пособие / А. В. Леоненков. - М. : ИНТУИТ : БИНОМ. Лаборатория знаний, 2013. - 320 с.

5. Буч Г. Введение в UML от создателей языка/ Грэди Буч, Джеймс Рамбо, Айвар Якобсон : Пер. с англ. — ДМК Пресс , 2015 – 469 с.

6. Арлоу Д. UML 2 и Унифицированный процесс. Практический объектно-ориентированный анализ и проектирование / Д. Арлоу, А. Нейштадт. - 2-е изд. - СПб. : Символ, 2014. - 624 с.

7. Фаулер М. UML. Основы. Третье издание. Краткое руководство по стандартному языку объектного моделирования. - Символ, 2013 – 350 с.

8. Нейштадт А. "UML 2 и Унифицированный процесс: практический объектно-ориентированный анализ и проектирование", - Символ-Плюс, 2013. - 624 с.

ПРИЛОЖЕНИЕ 1

Пример курсовой работы на тему:

«Объектно-ориентированное проектирование программного обеспечения информационной системы «Охранная фирма» средствами языка UML»

Введение

В данной работе приведен проект построения программного обеспечения информационной системы. В качестве предметной области рассмотрена «Охранная фирма». Проектирование производилось с помощью специализированного программного обеспечения IBM Rational Rose – программного пакета для создания диаграмм нотации UML.

1. Описание основных функций Информационной системы «Охранная фирма»:

- Получение лицензии
- Сотрудничество с заказчиками
  - Поиск заказчиков
  - Составление договоров
  - Получение объектов охраны
  - Работа на объектах
  - Получение средств на банковский счет
- Сотрудничество с магазином специализированной рабочей одежды
  - Перечисление средств магазину
  - Получение рабочей одежды
- Прием на работу сотрудников
  - Проверка документов
  - Составление договоров
  - Назначение на объекты
  - Охрана объектов
  - Выдача заработной платы сотрудникам
- Составление отчетности по фирме
  - Обработка информации о проделанной работе
  - Составление акта проделанных работ
  - Составление отчета в налоговую службу

2. Разработка ПО информационной системы «Охранная фирма»

2.1. Use-case диаграмма (диаграмма вариантов использования, сценариев, прецедентов). Диаграмма позволяет наглядно представить ожидаемое поведение системы. Элементы, используемые на диаграмме:

- Сценарий (определяет один фрагмент поведения системы, без раскрытия внутреннего содержания)

- Актер (внешняя по отношению к главной системе сущность, которая участвует в сценариях, является инициатором, источником или приемником информации, внутреннее содержание не рассматривается.)

- Интерфейс

- Комментарий

Диаграммы Use Case по концепции напоминают классические DFD диаграммы, также применяемые для структурного анализа. Use Case также отображают границы исследуемой системы, её функциональность и определяют сущности и процессы, а также пользователей системы. Диаграммы Use Case чаще всего используются на этапе трансформации логической архитектуры системы в концептуальную модель, реализуемую при объектном подходе с определением событийной структуры управления будущим программным проектом.

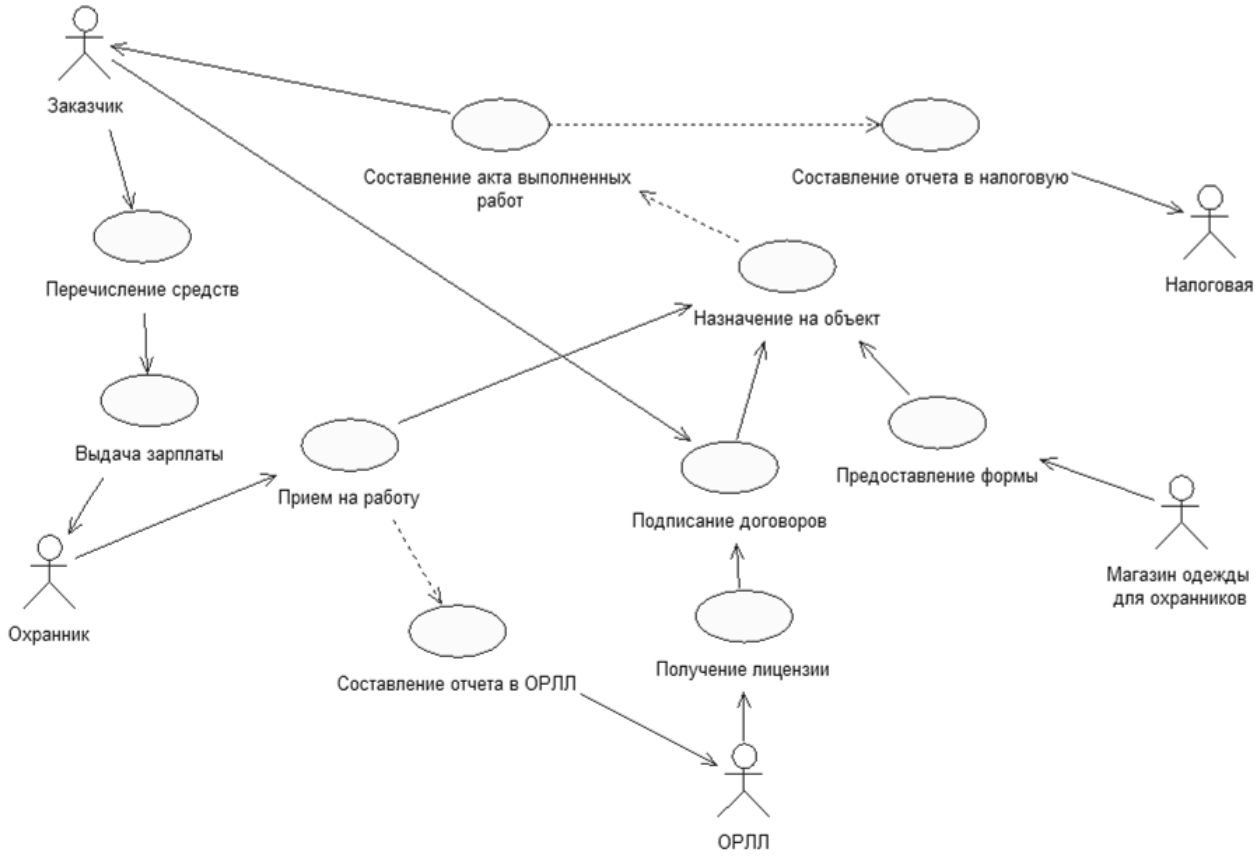


Рисунок - 1 Диаграмма вариантов использования ИС «Охранная фирма»

Расчет оценки диаграммы:

$$S_{diagr} = \frac{\sum Sobj + \sum Slink + Sclas}{1 + OBJ + \sqrt{Tobj + Tlink}},$$

где S_{obj}-оценка элемента на диаграмме, S_{link}- оценка связей, Obj- кол-во объектов на диаграмме, T_{obj} –количество типов объектов, T_{link}- количество типов связи.

$$S_{diagram} = \frac{(10*2+5*2)+(3*2+14*1)}{1+15+\sqrt{2+2}} = 2.8$$

4.2. Диаграмма классов

Диаграмма классов (Class diagram) — статическая структурная диаграмма, описывающая структуру системы, она демонстрирует классы системы, их атрибуты, методы и зависимости между классами.

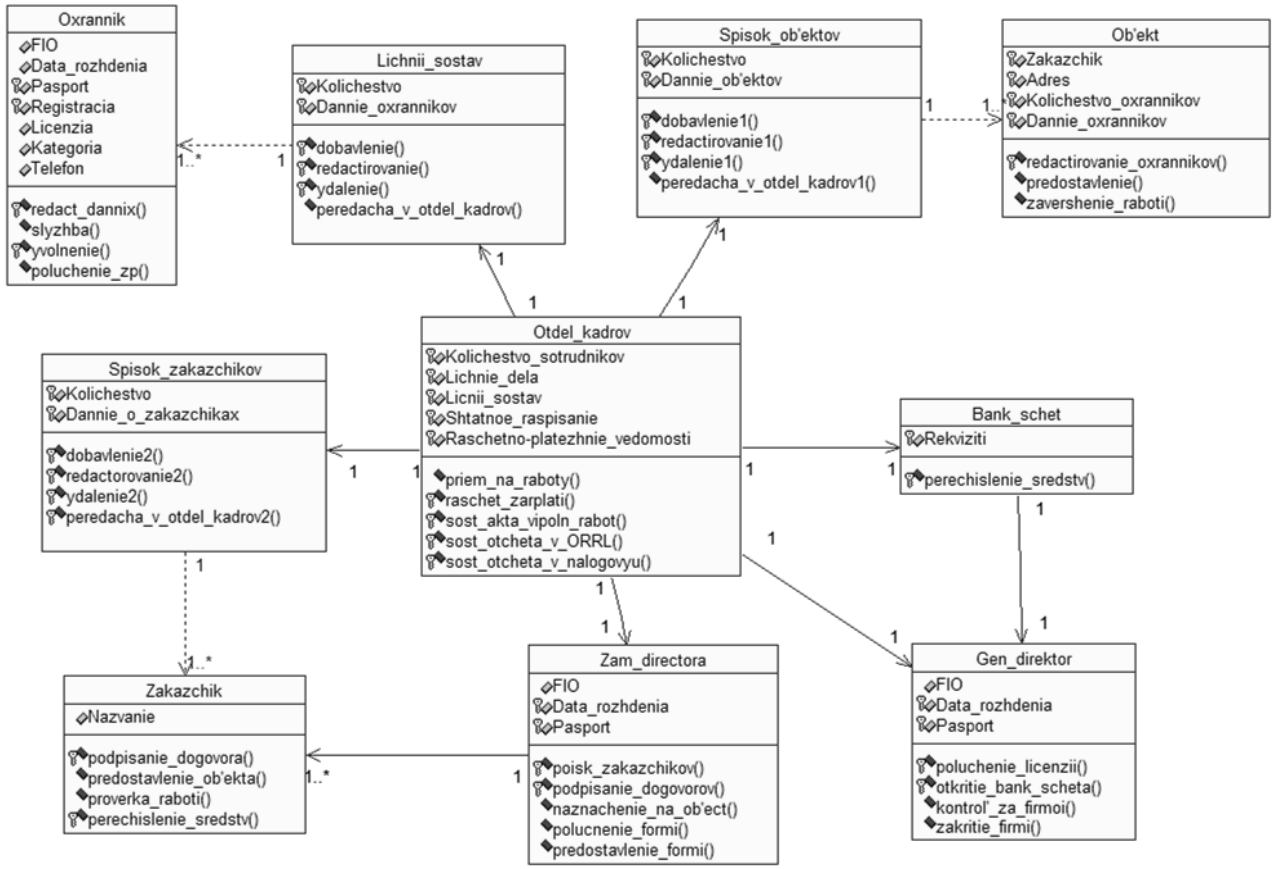


Рисунок - 2 Диаграмма классов ИС «Охранная фирма»

Расчет оценки диаграммы классов с атрибутами и операциями:

$$S_{diagram}=\frac{10*5+(8*1+3*2)+19.1}{1+10+\sqrt{1+2}}=6.5$$

$$S_{(Oxrannik)}=\frac{\sqrt{7}+\sqrt{4}}{0.3*(7+4)}=1.4$$

$$S_{(Lichnii\_sostav)}=\frac{\sqrt{2}+\sqrt{4}}{0.3*(2+4)}=1.9$$

$$S_{(Spisok\_ob'ektov)}=\frac{\sqrt{2}+\sqrt{4}}{0.3*(2+4)}=1.9$$

$$S_{(Ob'ekt)}=\frac{\sqrt{4}+\sqrt{3}}{0.3*(4+3)}=1.8$$

$$S_{(Spisok\_zakazchikov)}=\frac{\sqrt{2}+\sqrt{4}}{0.3*(2+4)}=1.9$$

$$S_{(Otdel\_kadrov)}=\frac{\sqrt{5}+\sqrt{5}}{0.3*(5+5)}=1.5$$

$$S_{(Bank\_schet)}=\frac{\sqrt{1}+\sqrt{1}}{0.3*(1+1)}=3*3$$

$$S_{(Zakazchik)}=\frac{\sqrt{4}+\sqrt{1}}{0.3*(4+1)}=2$$

$$S_{(Zam\_directora)}=\frac{\sqrt{3}+\sqrt{5}}{0.3*(3+5)}=1.6$$

$$S_{(Gen\_direktor)}=\frac{\sqrt{3}+\sqrt{4}}{0.3*(3+4)}=1.8$$

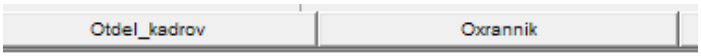
4.3. Диаграммы последовательностей

Диаграмма последовательностей ***(****англ.* Sequence diagram*)*** — диаграмма, на которой показаны взаимодействия объектов, упорядоченные по времени их проявления. Основными элементами диаграммы последовательностей являются обозначения объектов (прямоугольники), вертикальные линии (англ. *lifeline*), отображающие течение времени при деятельности объекта, и стрелки, показывающие выполнение действий объектами.

Объекты - это информационные единицы, участвующие в реализации сценария (варианта использования). Первым указывается объект, который является инициатором

данного сценария. Объекты обмениваются сообщениями, порядок их указывается в виде цифры с двоеточием.

На данной диаграмме объекты располагаются слева направо.



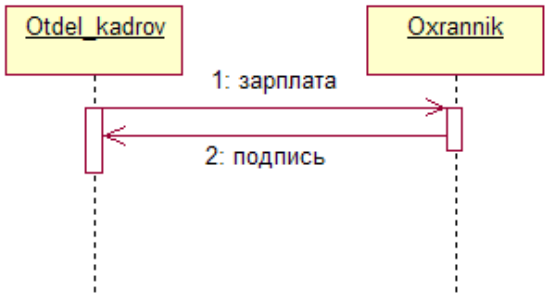
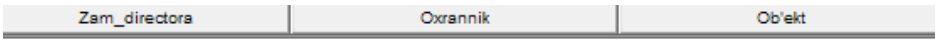


Рисунок - 3 Диаграмма последовательностей для сценария «Выдача зарплаты».

$$S_{diagram} = \frac{2*5+1*2}{1+2+\sqrt{1+1}} = 2.5$$



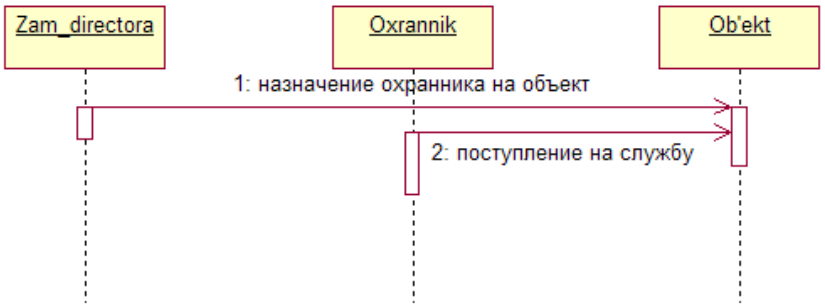


Рисунок - 4 Диаграмма последовательностей для сценария «Назначение на объект».

$$S_{diagram} = \frac{3*5+2*1}{1+3+\sqrt{1+1}} = 3.1$$

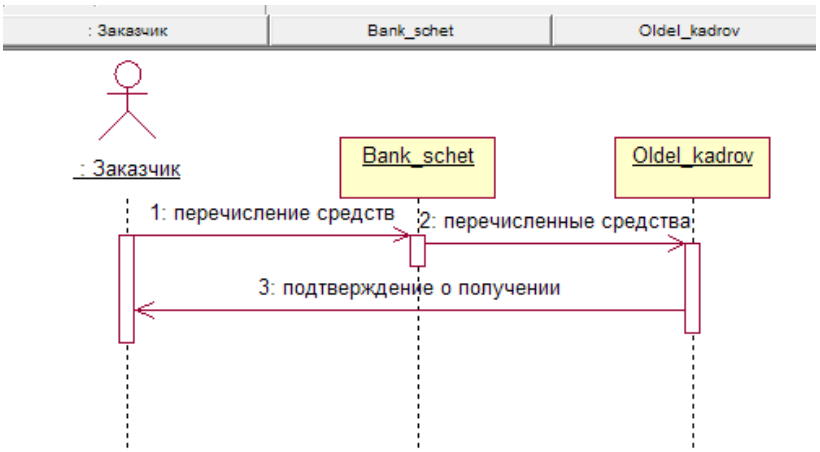


Рисунок - 5 Диаграмма последовательностей для сценария «Перечисление средств».

$$S_{diagram} = \frac{2*5+2+3*1}{1+3+\sqrt{2+1}} = 2.6$$

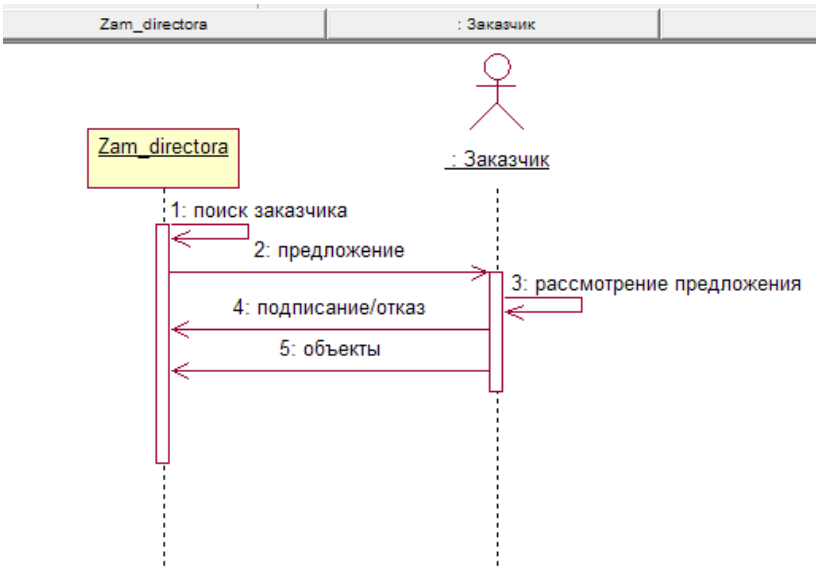


Рисунок - 6 Диаграмма последовательностей для сценария «Подписание договоров».

$$S_{diagram} = \frac{5+2+5*1}{1+2+\sqrt{2+1}} = 2.6$$

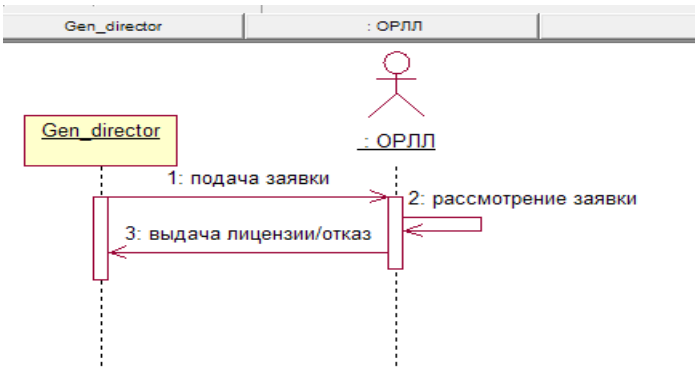


Рисунок - 7 Диаграмма последовательностей для сценария «Получение лицензии».

$$S_{diagram} = \frac{5+2+3}{1+2+\sqrt{2+1}} = 2.1$$

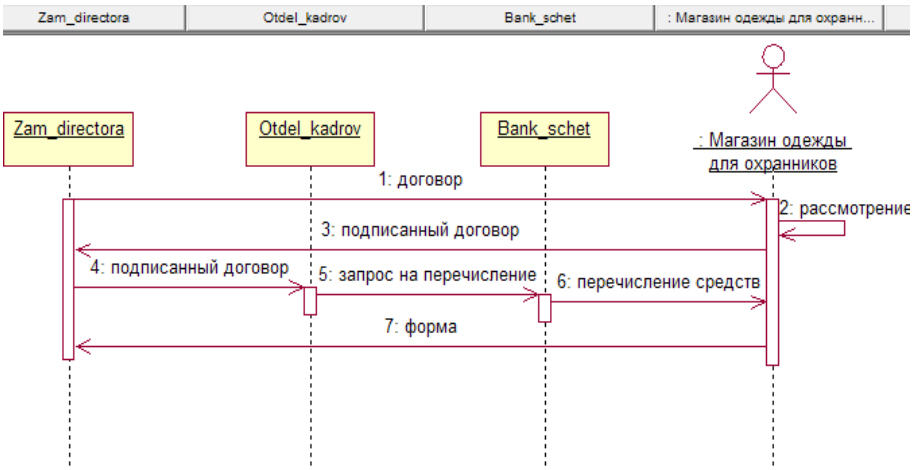


Рисунок - 8 Диаграмма последовательностей для сценария «Предоставление формы».

$$S_{diagram} = \frac{5*3+2+7*1}{1+4+\sqrt{2+1}} = 3.6$$

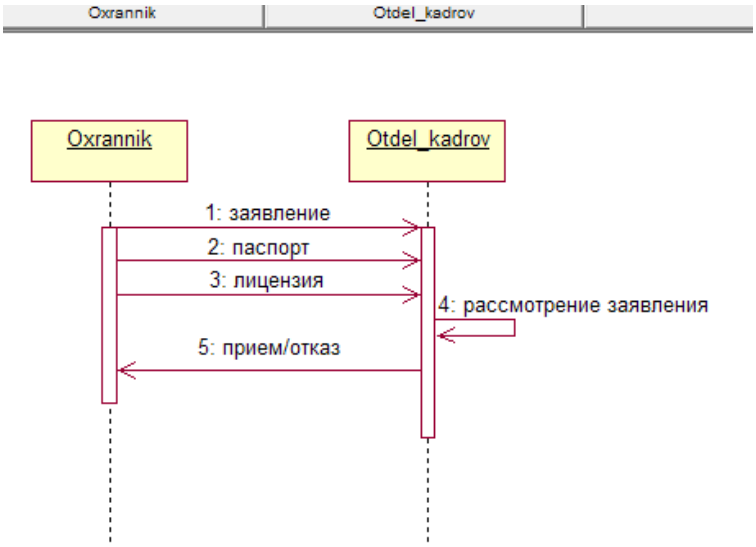


Рисунок - 9 Диаграмма последовательностей для сценария «Прием на работу».

$$S_{diagram} = \frac{5*2+5*1}{1+2+\sqrt{1+1}} = 3.4$$

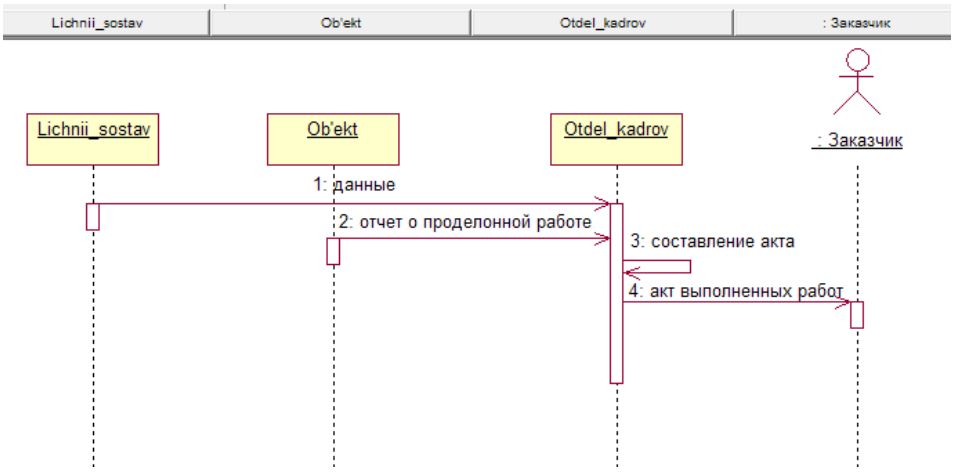


Рисунок - 10 Диаграмма последовательностей для сценария «Составление акта выполненных работ».

$$S_{diagram} = \frac{5*3+2+4*1}{1+4+\sqrt{2+1}} = 3.1$$

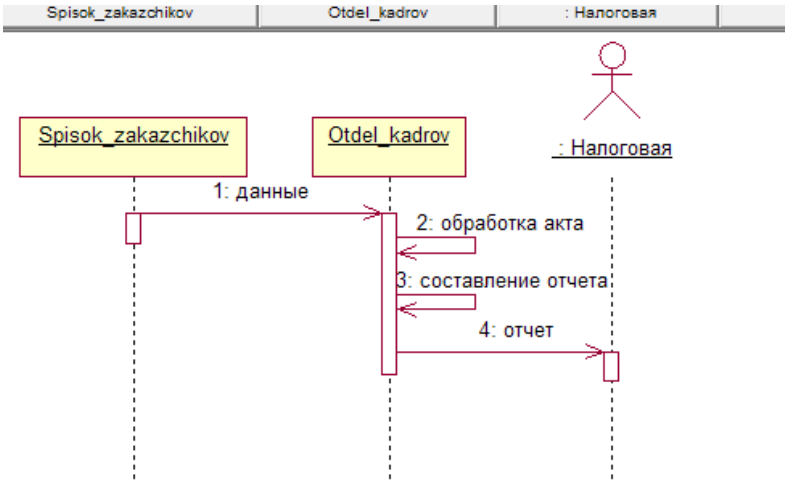


Рисунок - 11 Диаграмма последовательностей для сценария «Составление отчета в налоговую».

$$S_{diagram} = \frac{5*2+2+4*1}{1+3+\sqrt{2+1}} = 2.8$$

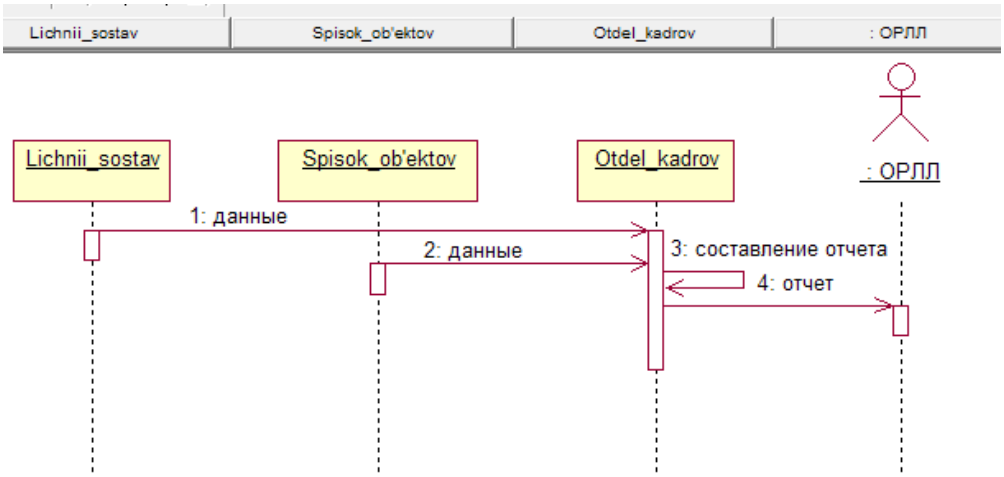


Рисунок - 12 Диаграмма последовательностей для сценария «Составление отчета в ОРЛЛ».

$$S_{diagram} = \frac{5*3+2*1+4*1}{1+4+\sqrt{2+1}} = 3.1$$

4.4. Диаграммы состояний (Statechar diagram)

Диаграмма состояний(Statechar diagram) определяют все возможные состояния, в которых может находиться конкретный объект, а также процесс смены состояний объекта в результате наступления некоторых событий или получения некоторого сообщения.

Существует много форм диаграмм состояний, незначительно отличающихся друг от друга семантикой.

На диаграмме имеются два специальных состояния – начальное (start) и конечное (stop). Начальное состояние выделено черной точкой, оно соответствует состоянию объекта, когда он только что был создан. Конечное состояние обозначается черной точкой в белом кружке, оно соответствует состоянию объекта непосредственно перед его уничтожением. На диаграмме состояний может быть одно и только одно начальное состояние. В то же время, может быть столько конечных состояний, сколько вам нужно, или их может не быть вообще. Когда объект находится в каком-то конкретном состоянии, могут выполняться различные процессы. Процессы, происходящие, когда объект находится в определенном состоянии, называются действиями (actions).

С состоянием можно связывать данные пяти типов: деятельность, входное действие, выходное действие, событие и история состояния.

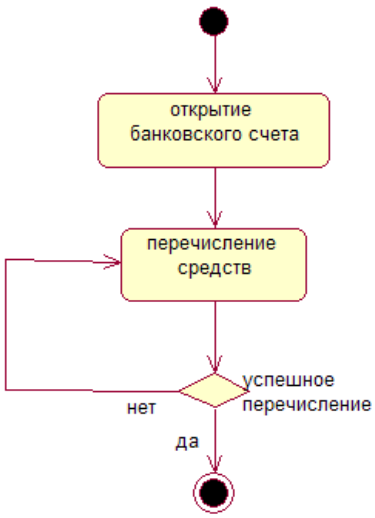


Рисунок - 13 Диаграмма состояний для класса «Bank_schet»

$$S_{diagram} = \frac{2*4+5}{1+3+\sqrt{2+1}} = 2.3$$

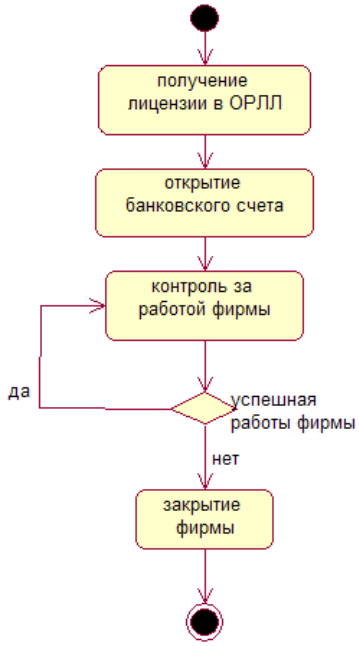


Рисунок - 14 Диаграмма состояний для класса «Gen_director»

$$S_{diagram} = \frac{4*4+7}{1+5+\sqrt{2+1}} = 3$$

Рисунок - 15 Диаграмма состояний для класса «Lichnii_sostav»

$$S_{diagram} = \frac{5*4+10}{1+7+\sqrt{2+1}} = 3.1$$

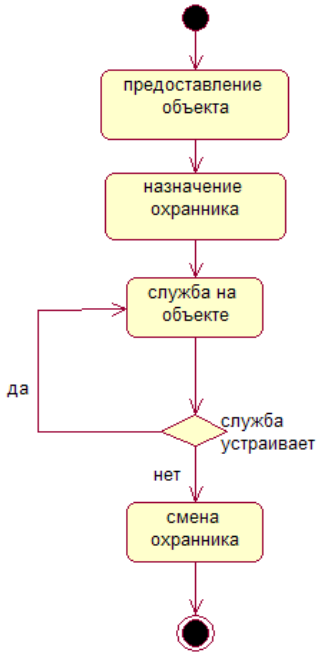


Рисунок - 16 Диаграмма состояний для класса «Ob’ekt»

$$S_{diagram} = \frac{4*4+7}{1+5+\sqrt{2+1}} = 3$$

Рисунок - 17 Диаграмма состояний для класса «Otdel_kadrov»

$$S_{diagram} = \frac{7*4+9*1}{1+8+\sqrt{2+1}} = 3.5$$

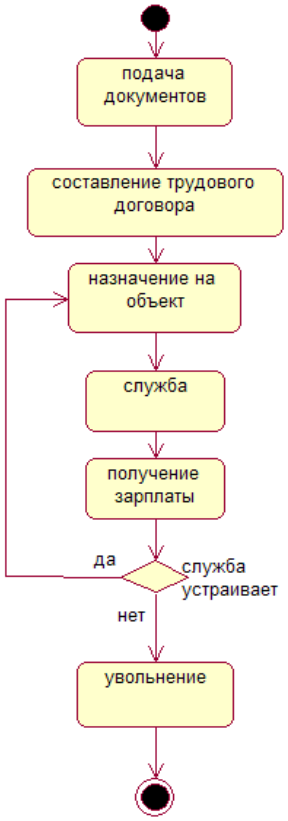


Рисунок - 18 Диаграмма состояний для класса «Oxrannik»

$$S_{diagram} = \frac{6*4+9}{1+7+\sqrt{2+1}} = 3.4$$

Рисунок - 19 Диаграмма состояний для класса «Spisok_ob’ektov»

$$S_{diagram} = \frac{6*4+11}{1+8+\sqrt{2+1}} = 3.3$$

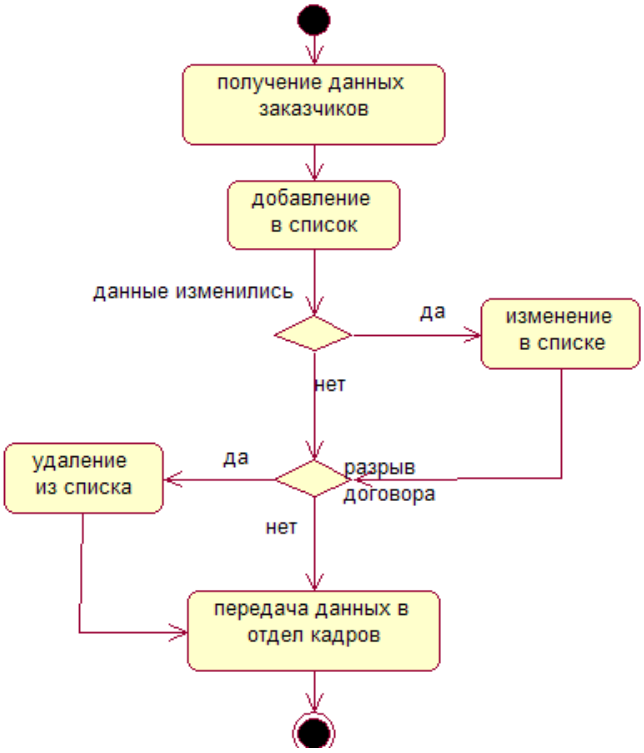


Рисунок - 20 Диаграмма состояний для класса «Spisok_zakazchikov»

$$S_{diagram} = \frac{4*5+10}{1+6+\sqrt{2+1}} = 3.4$$

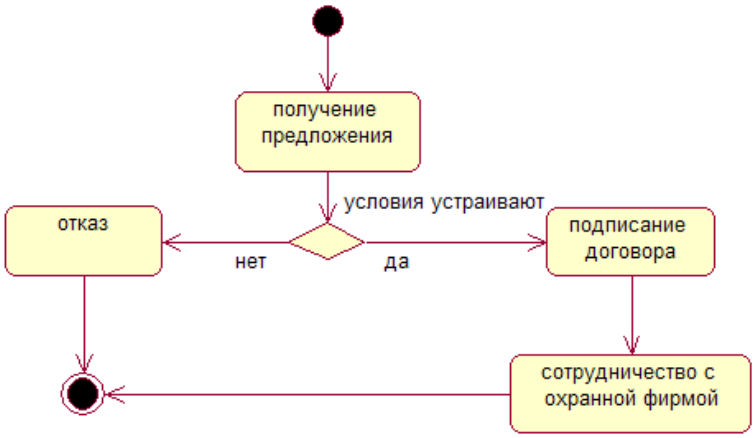


Рисунок - 21 Диаграмма состояний для класса «Zakazchik»

$$S_{diagram} = \frac{4*4+7}{1+5+\sqrt{2+1}} = 3$$

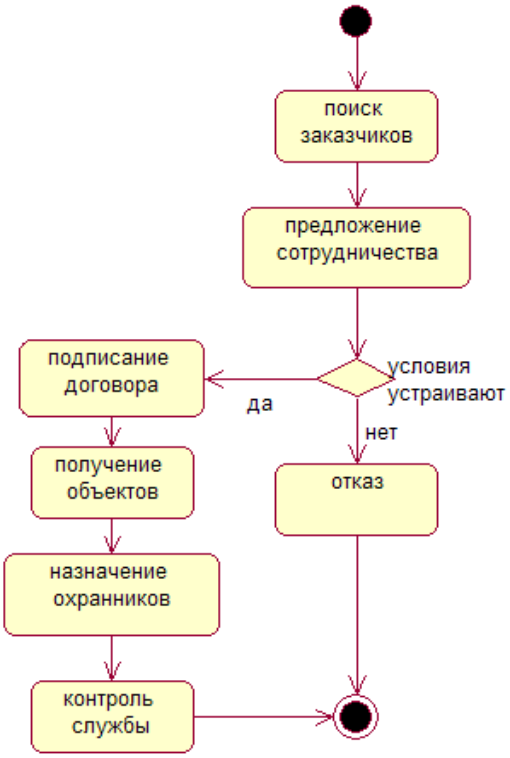
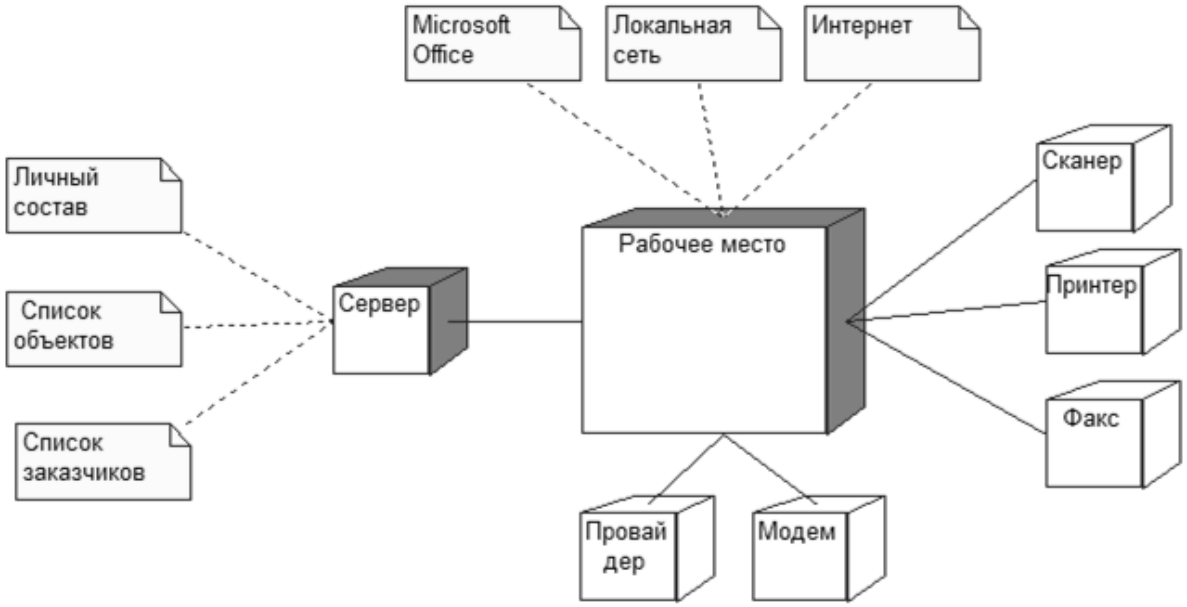


Рисунок - 22 Диаграмма состояний для класса «Zam_directora»

$$S_{diagram} = \frac{7*4+10}{1+8+\sqrt{2+1}} = 3.5$$

4.5. Дразмещений (Диаграмма развертывания).

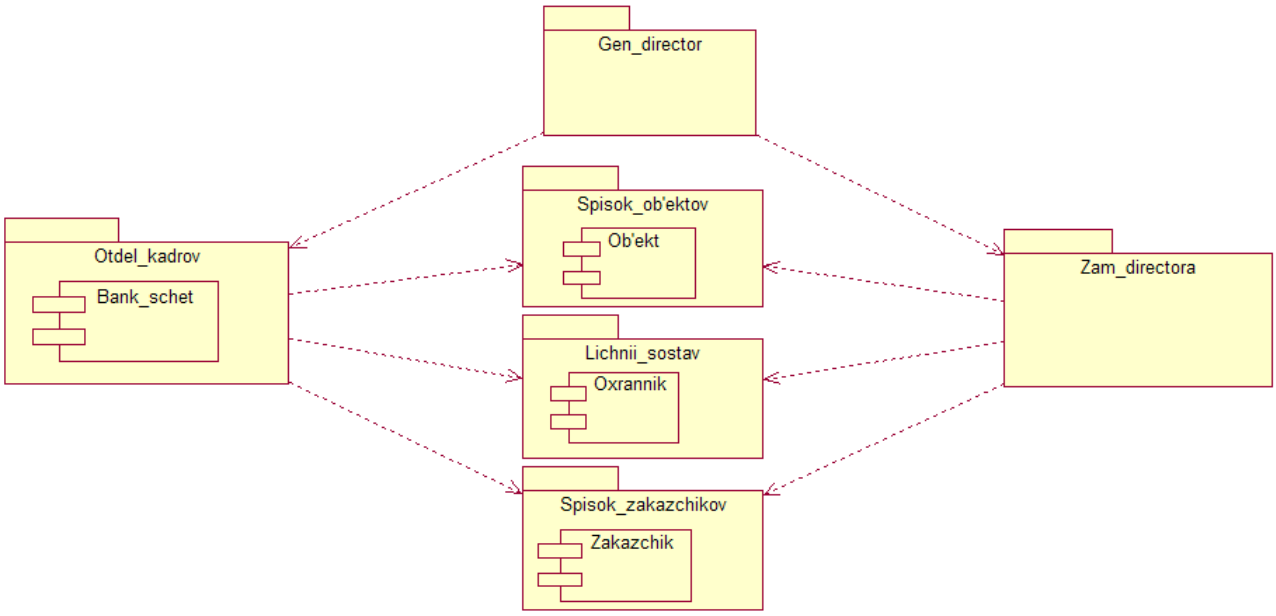
Диаграмма развертывания или размещения (Deployment diagram) - предназначена для визуализации элементов и компонентов программы, существующих лишь на этапе ее исполнения. Диаграмма развертывания отражает физические взаимосвязи между программными и аппаратными компонентами разрабатываемой системы. Каждый узел на диаграмме развертывания представляет собой некоторый тип вычислительного устройства - в большинстве случаев самостоятельную часть аппаратуры.



$$S = \frac{5*3+2*2+6*2+6*1+6*1}{1+13+\sqrt{3+2}} = 2.7$$

4.7. Диаграмма пакетов (Package diagram)

Диаграмма пакетов (Package diagram) - структурная диаграмма, основным содержанием которой являются пакеты и отношения между ними. Диаграммы пакетов служат, в первую очередь, для организации элементов в группы по какому-либо признаку с целью упрощения структуры и организации работы с моделью системы. Строго говоря, пакеты и зависимости являются элементами диаграммы классов, т. е. диаграмма пакетов - это всего лишь форма диаграммы классов.



$$S_{diagram} = \frac{6*4 + 4*4 + 8*2}{1 + 10 + \sqrt{2+1}} = 4{,}4$$

ЗАКЛЮЧЕНИЕ

В данной курсовой работе разработана структура программного обеспечения информационной системы «Охранная фирма» в нотации UML с использованием CASE-средства IBM Rational Rose EnterPrise Edition. В пояснительной записке рассмотрены основные функции, сценарии поведения и алгоритмы изменения состояний главных информационных объектов (классов) данной системы, разработаны диаграммы, описывающие различные аспекты ее функционирования, проведена автоматическая генерация кодов программ на основе разработанной диаграммы классов. Диаграммы сравнительно просты для чтения, методы описания результатов анализа и проектирования семантически близки к методам программирования на современных объектно-ориентированных языках.

Данная работа дает возможность организовать качественное функционирование описанной информационной системы, а также позволяет автоматизировать основные процессы, что, в свою очередь, приведет к сокращению времени и стоимости разработки ПО ИС.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Буч Грэди Введение в UML от создателей языка / Грэди Буч, Джеймс Рамбо, Айвар Якобсон: пер. с англ. – ДМК Пресс, 2015 – 496 с.: ил.

2. Ларман Крэг Применение UML 2.0 и шаблонов проектирования. Введение в объектно-ориентированный анализ, проектирование и итеративную разработку: пер. с англ. – М.: ИД "Вильямс", 2013. – 736 с.: ил.

3. Umbrello UML Modeller [Электронный ресурс]. – KDE, 2018. – URL: https://umbrello.kde.org/

ПРИЛОЖЕНИЕ. Результаты автоматической генерации кода программ на основе спроектированной диаграммы классов