|  |  |
| --- | --- |
|  | **ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО РЫБОЛОВСТВУ**  **ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ**  **ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ**  **«АСТРАХАНСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»** |

**ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И КОММУНИКАЦИЙ**

**КАФЕДРА   
 «АВТОМАТИЗИРОВАННЫЕ СИСТЕМЫ ОБРАБОТКИ ИНФОРМАЦИИ И УПРАВЛЕНИЯ»**

ДОПУЩЕНО К ЗАЩИТЕ

Заведующий кафедрой, доцент

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ М.Б. Френкель

«\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 2011 г.

От \_\_\_\_\_\_\_\_\_\_ № \_\_\_\_\_\_\_\_\_\_\_

**КУРСОВАЯ РАБОТА**

по дисциплине

«Программирование на языке высокого уровня»

на тему:

«Визуализированный графический редактор  
UML-диаграмм»

КР-230100.62-ОПД.В.01.1 - 091198

Студент гр. ДИН-21

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ П.Н. Савченко

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_2011 г.

Руководитель доцент

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.В. Лаптев

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_2011 г.

Защищено с оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Руководитель \_\_\_\_\_\_\_ В.В. Лаптев

Член комиссии \_\_\_\_\_\_\_\_\_\_

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_2011 г.

Астрахань 2011

ОГЛАВЛЕНИЕ

[ВВЕДЕНИЕ 4](#_Toc296087749)

[1 ТЕХНИЧЕСКИЙ ПРОЕКТ 5](#_Toc296087750)

[1.1 Предметная область 5](#_Toc296087751)

[1.1.1 Диаграмма классов 5](#_Toc296087752)

[1.1.2 Диаграмма вариантов использования 10](#_Toc296087753)

[1.1.3 Диаграмма состояний 15](#_Toc296087754)

[1.2 Технология обработки информации 18](#_Toc296087755)

[1.2.1 Каркас графического представления 18](#_Toc296087755)

[1.2.2 Алгоритм рисования 21](#_Toc296087756)

[1.2.3 Сохранение диаграммы в файл 21](#_Toc296087757)

[1.2.4 Чтение диаграммы из файла 22](#_Toc296087758)

[1.2.5 Формат 22](#_Toc296087759)

[1.2.6 Архитектура программы 22](#_Toc296087760)

[1.3 Инфологическая модель 23](#_Toc296087761)

[1.4 Минимальные системные требования 24](#_Toc296087762)

[2 РАБОЧИЙ ПРОЕКТ 26](#_Toc296087763)

[2.1 Общие сведения о работе программы 26](#_Toc296087766)

[2.2 Функциональное назначение 26](#_Toc296087767)

[2.3 Инсталляция и выполнение 27](#_Toc296087768)

[2.4 Общий алгоритм программного продукта 28](#_Toc296087774)

[2.4.1 Структура программного проекта 29](#_Toc296087775)

[2.4.2 Функциональное назначение модулей, классов, объектов, методов 30](#_Toc296087776)

[2.5 Разработанное меню и интерфейсы 38](#_Toc296087781)

[3 ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ 44](#_Toc296087782)

[ЗАКЛЮЧЕНИЕ 45](#_Toc296087783)

[ЛИТЕРАТУРА 46](#_Toc296087784)

### ВВЕДЕНИЕ

Если попытаться охарактеризовать современный уровень развития компьютерных и информационных технологий, то первое, на что следует обратить внимание – это возрастающая сложность не только кода программного обеспечения, но и логической структуры этого программного обеспечения. Еще совсем недавно профессиональному программисту было достаточно в совершенстве владеть одним-двумя языками программирования, чтобы разрабатывать серьезные программные приложения.

В настоящее время разработка программного проекта является достаточно сложным процессом, еще недавно бывшая уделом избранных одиночек, в настоящее время превратилась в высокодоходную сферу бизнеса. В ИT-проектах заняты миллионы программистов и аналитиков, руководителей разного ранга и простых инженеров по всему миру. Таким образом, для упрощения этапа проектирования программного обеспечения был изобретен унифицированный язык моделирования (UML), который получил широкое распространение и динамично развивается. Использование UML не ограничивается моделированием программного обеспечения. Его также используют для [моделирования бизнес-процессов](http://ru.wikipedia.org/wiki/%D0%91%D0%B8%D0%B7%D0%BD%D0%B5%D1%81-%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), [системного проектирования](http://ru.wikipedia.org/w/index.php?title=%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5&action=edit&redlink=1) и отображения [организационных структур](http://ru.wikipedia.org/wiki/%D0%9E%D1%80%D0%B3%D0%B0%D0%BD%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0). UML позволяет также разработчикам программного обеспечения достигнуть соглашения в графических обозначениях для представления общих понятий, и больше сконцентрироваться на проектировании и архитектуре.

Перед разработчиком стоит задача создать приложение, которое позволяет создавать, редактировать и сохранять в файл UML диаграммы.

Данное программное обеспечение имеет большое количество аналогов (ArgoUML, VioletUML). Главная особенность данного графического редактора­­ – минимальный функционал, простота и удобство быстрого созданий простых диаграмм. Функционал программы минимален, но в тоже время весьма практичен и позволяет очень быстро создавать несложные UML диаграммы.

# ТЕХНИЧЕСКИЙ ПРОЕКТ

## 1.1 Предметная область

Перед разработчиком стоит задача – создать программное обеспечение, которое должно предоставлять пользователю визуализированный графический интерфейс. При этом пользователь должен иметь возможность создавать три вида диаграмм с помощью унифицированного языка моделирования (сокр. англ. UML - Unified Modeling Language),   
а именно: диаграмму классов ([сlass diagram](http://ru.wikipedia.org/wiki/%D0%94%D0%B8%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0_%D0%BA%D0%BB%D0%B0%D1%81%D1%81%D0%BE%D0%B2)), диаграмму вариантов использования (use case diagram), диаграмму состояний (state machine diagram), а также редактировать и сохранять их.

UML — язык графического описания для объектного моделирования в области разработки программного обеспечения. UML является языком широкого профиля, это открытый стандарт, использующий графические обозначения для создания абстрактной модели системы, называемой UML-моделью. UML был создан для определения, визуализации, проектирования и документирования в основном программных систем.

### 1 Диаграмма классов

Диаграмма классов (class diagram) служит для представления статической структуры модели системы. Она может отражать, в частности, различные взаимосвязи между отдельными сущностями предметной области, такими как объекты и подсистемы, а также описывает их внутреннюю структуру и типы отношений. На данной диаграмме не указывается информация о временных аспектах функционирования системы. С этой точки зрения диаграмма классов является дальнейшим развитием концептуальной модели проектируемой системы.

Диаграмма классов состоит из множества элементов, которые в совокупности отражают декларативные знания о предметной области. Эти знания интерпретируются в базовых понятиях языка UML, таких как классы, интерфейсы и отношения между ними и их компонентами. При этом отдельные компоненты диаграммы классов могут образовывать пакеты для представления более общей модели системы. Если диаграмма классов является частью некоторого пакета, то ее компоненты должны соответствовать элементам этого пакета, включая возможные ссылки на элементы из других пакетов.

Класс (class) в языке UML служит для обозначения множества объектов, которые обладают одинаковой структурой, поведением и отношениями с объектами из других классов. Графически класс изображается в виде прямоугольника, который дополнительно может быть разделен горизонтальными линиями на разделы или секции (рис. 1.1). В этих разделах могут указываться имя класса, атрибуты (переменные) и операции (методы).

Имя Класса

Имя Класса

Атрибуты Класса

Имя Класса

Атрибуты Класса

Операции Класса

Рис.1.1 Графическое изображение класса на диаграмме классов

Имя класса должно быть уникальным в пределах пакета, который описывается некоторой совокупностью диаграмм классов (возможно, одной диаграммой). Оно указывается в верхней секции прямоугольника. В дополнение к общему правилу наименования элементов языка UML, имя класса записывается по центру секции имени полужирным шрифтом и должно начинаться с заглавной буквы. Рекомендуется в качестве имен классов использовать существительные, записанные по практическим соображениям без пробелов.

Во второй сверху секции прямоугольника класса записываются его атрибуты (attributes) или свойства. В языке UML принята определенная стандартизация записи атрибутов класса, которая подчиняется некоторым синтаксическим правилам. Каждому атрибуту класса соответствует отдельная строка текста, которая состоит из квантора видимости атрибута, имени атрибута, его кратности (общее количество конкретных атрибутов данного типа, входящих в состав отдельного класса), типа значений атрибута и, возможно, его исходного значения:

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

Квантор видимости может принимать одно из трех возможных значений и, соответственно, отображается при помощи специальных символов:

* Символ "+" обозначает атрибут с областью видимости типа общедоступный (public). Атрибут с этой областью видимости доступен или виден из любого другого класса пакета, в котором определена диаграмма.
* Символ "#" обозначает атрибут с областью видимости типа защищенный (protected). Атрибут с этой областью видимости недоступен или невиден для всех классов, за исключением подклассов данного класса.
* Символ " - " обозначает атрибут с областью видимости типа закрытый (private). Атрибут с этой областью видимости недоступен или невиден для всех классов без исключения.

В третьей сверху секции прямоугольника записываются операции или методы класса. Операция (operation) представляет собой некоторый сервис, предоставляющий каждый экземпляр класса по определенному требованию. Совокупность операций характеризует функциональный аспект поведения класса. Запись операций класса в языке UML также стандартизована и подчиняется определенным синтаксическим правилам. При этом каждой операции класса соответствует отдельная строка, которая состоит из квантора видимости операции, имени операции, выражения типа возвращаемого операцией значения и, возможно, строка-свойство данной операции:

* <квантор видимости><имя операции>(список параметров):
* <выражение типа возвращаемого значения>{строка-свойство}

Квантор видимости, как и в случае атрибутов класса, может принимать одно из трех возможных значений и, соответственно, отображается при помощи специального символа.

Кроме внутреннего устройства или структуры классов на соответствующей диаграмме указываются различные отношения между классами. При этом совокупность типов таких отношений фиксирована в языке UML и предопределена семантикой этих типов отношений. Базовыми отношениями или связями в языке UML являются:

* Отношение зависимости (dependency relationship).
* Отношение ассоциации (association relationship).
* Отношение обобщения (generalization relationship).
* Отношение реализации (realization relationship).

Отношение зависимости в общем случае указывает некоторое семантическое отношение между двумя элементами модели. Оно используется в такой ситуации, когда некоторое изменение одного элемента модели может потребовать изменения другого зависимого от него элемента модели.

Каждое из этих отношений имеет собственное графическое представление на диаграмме, которое отражает взаимосвязи между объектами соответствующих классов. Отношение зависимости графически изображается пунктирной линией между соответствующими элементами со стрелкой на одном из ее концов ("->" или "<-").

Класс\_А

Класс\_Б

Рис.1.2. Графическое изображение отношения зависимости на диаграмме классов

Ассоциация — это структурное отношение, показывающее, что объекты одной сущности связаны с объектами другой. Графически ассоциация показывается в виде линии, соединяющей связываемые сущности. Ассоциации служат для осуществления навигации между объектами (рис. 1.3).

Класс\_А

Класс\_В

Рис. 1.3. Графическое изображение отношения бинарной ассоциации между классами

Отношение агрегации имеет место между несколькими классами в том случае, если один из классов представляет собой некоторую сущность, включающую в себя в качестве составных частей другие сущности.

Графически отношение агрегации изображается сплошной линией, один из концов которой представляет собой незакрашенный внутри ромб. Этот ромб указывает на тот из классов, который представляет собой "целое". Остальные классы являются его   
"частями" (рис. 1.4).

Целое

Часть

Рис. 1.4. Графическое изображение отношения агрегации в языке UML

Отношение композиции является частным случаем отношения агрегации. Это отношение служит для выделения специальной формы отношения "часть-целое", при которой составляющие части в некотором смысле находятся внутри целого. Специфика взаимосвязи между ними заключается в том, что части не могут выступать в отрыве от целого, т. е. с уничтожением целого уничтожаются и все его составные части.

Графически отношение композиции изображается сплошной линией, один из концов которой представляет собой закрашенный внутри ромб. Этот ромб указывает на тот из классов, который представляет собой класс-композицию или "целое". Остальные классы являются его "частями" (рис. 1.5).

Целое

Часть

Рис. 1.5. Графическое изображение отношения композиции в языке UML

Отношение обобщения является обычным таксономическим отношением между более общим элементом (родителем или предком) и более частным или специальным элементом (дочерним или потомком). Данное отношение может использоваться для представления взаимосвязей между пакетами, классами, вариантами использования и другими элементами языка UML.

Применительно к диаграмме классов данное отношение описывает иерархическое строение классов и наследование их свойств и поведения. При этом предполагается, что класс-потомок обладает всеми свойствами и поведением класса-предка, а также имеет свои собственные свойства и поведение, которые отсутствуют у класса-предка. На диаграммах отношение обобщения обозначается сплошной линией с треугольной стрелкой на одном из концов (рис. 1.6). Стрелка указывает на более общий класс (класс-предок или суперкласс), а ее отсутствие - на более специальный класс (класс-потомок или подкласс).

Класс-предок

Класс-потомок

Рис. 1.6. Графическое изображение отношения обобщения в языке UML

### 1.1.2 Диаграмма вариантов использования

Визуальное моделирование в UML можно представить, как некоторый процесс спуска от наиболее обшей и абстрактной концептуальной модели исходной системы к логической, а затем и к физической модели соответствующей программной системы. Для достижения этих целей вначале строится модель в форме, так называемой диаграммы вариантов использования (use case diagram), которая описывает функциональное назначение системы или, другими словами, то, что система будет делать в процессе своего функционирования. Диаграмма вариантов использования является исходным концептуальным представлением или концептуальной моделью системы в процессе ее проектирования и разработки.

Суть данной диаграммы состоит в следующем: проектируемая система представляется в виде множества сущностей или актеров, взаимодействующих с системой с помощью, так называемых вариантов использования. При этом актером (actor) или действующим лицом называется любая сущность, взаимодействующая с системой извне. Это может быть человек, техническое устройство, программа или любая другая система, которая может служить источником воздействия на моделируемую систему так, как определит сам разработчик. В свою очередь, вариант использования (use case) служит для описания сервисов, которые система предоставляет актеру. Другими словами, каждый вариант использования определяет некоторый набор действий, совершаемый системой при диалоге с актером. При этом ничего не говорится о том, каким образом будет реализовано взаимодействие актеров с системой.

Конструкция или стандартный элемент языка UML вариант использования применяется для спецификации общих особенностей поведения системы или любой другой сущности предметной области без рассмотрения внутренней структуры этой сущности. Каждый вариант использования определяет последовательность действий, которые должны быть выполнены проектируемой системой при взаимодействии ее с соответствующим актером. Диаграмма вариантов может дополняться пояснительным текстом, который раскрывает смысл или семантику составляющих ее компонентов. Такой пояснительный текст получил название примечания или сценария.

Отдельный вариант использования обозначается на диаграмме эллипсом, внутри которого содержится его краткое название или имя в форме глагола с пояснительными словами (рис. 1.7).

Рис. 1.7. Графическое обозначение варианта использования

Актер представляет собой любую внешнюю по отношению к моделируемой системе сущность, которая взаимодействует с системой и использует ее функциональные возможности для достижения определенных целей или решения частных задач. При этом актеры служат для обозначения согласованного множества ролей, которые могут играть пользователи в процессе взаимодействия с проектируемой системой. Каждый актер может рассматриваться как некая отдельная роль относительно конкретного варианта использования. Стандартным графическим обозначением актера на диаграммах является фигурка "человечка", под которой записывается конкретное имя актера (рис. 1.8).

Рис. 1.8. Графическое обозначение актера

Интерфейс (interface) служит для спецификации параметров модели, которые видимы извне без указания их внутренней структуры. В языке UML интерфейс является классификатором и характеризует только ограниченную часть поведения моделируемой сущности. Применительно к диаграммам вариантов использования, интерфейсы определяют совокупность операций, которые обеспечивают необходимый набор сервисов или функциональности для актеров. Интерфейсы не могут содержать ни атрибутов, ни состояний, ни направленных ассоциаций. Они содержат только операции без указания особенностей их реализации. Формально интерфейс эквивалентен абстрактному классу без атрибутов и методов с наличием только абстрактных операций.

На диаграмме вариантов использования интерфейс изображается в виде маленького круга, рядом с которым записывается его имя (рис. 1.9, а). В качестве имени может быть существительное, которое характеризует соответствующую информацию или сервис (например, "датчик", "сирена", "видеокамера"), но чаще строка текста (например, "запрос к базе данных", "форма ввода", "устройство подачи звукового сигнала"). Если имя записывается на английском, то оно должно начинаться с заглавной буквы I, например, ISecurelnformation, ISensor (рис. 1.9, б).

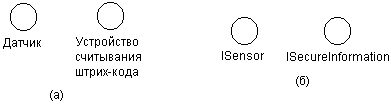


Рис. 1.9. Графическое изображение интерфейсов на диаграммах вариантов использования

Примечания (notes) в языке UML предназначены для включения в модель произвольной текстовой информации, имеющей непосредственное отношение к контексту разрабатываемого проекта. В качестве такой информации могут быть комментарии разработчика (например, дата и версия разработки диаграммы или ее отдельных компонентов), ограничения (например, на значения отдельных связей или экземпляры сущностей) и помеченные значения. Применительно к диаграммам вариантов использования примечание может носить самую общую информацию, относящуюся к общему контексту системы.

Графически примечания обозначаются прямоугольником с "загнутым" верхним правым уголком. Внутри прямоугольника содержится текст примечания. Примечание может относиться к любому элементу диаграммы, в этом случае их соединяет пунктирная   
линия (рис. 1.10). Если примечание относится к нескольким элементам, то от него проводятся, соответственно, несколько линий. Примечания могут присутствовать не только на диаграмме вариантов использования, но и на других канонических диаграммах.

Реализовать в виде отдельной библиотеки стандартных функций

Эта модель разработана О. Бендером после встречи с командой планирования стратегий развития

Рис. 1.10. Примеры примечаний в языке UML

Между компонентами диаграммы вариантов использования могут существовать различные отношения, которые описывают взаимодействие экземпляров одних актеров и вариантов использования с экземплярами других актеров и вариантов. Один актер может взаимодействовать с несколькими вариантами использования. В этом случае этот актер обращается к нескольким сервисам данной системы.

Более того, варианты использования всегда предусматривают некоторые сигналы или сообщения, когда взаимодействуют с актерами за пределами системы. В то же время могут быть определены другие способы для взаимодействия с элементами внутри системы.

В языке UML имеется несколько стандартных видов отношений между актерами и вариантами использования:

* Отношение ассоциации (association relationship).
* Отношение расширения (extend relationship).
* Отношение обобщения (generalization relationship).
* Отношение включения (include relationship).

Отношение обобщения и отношение ассоциации в диаграмме вариантов использования по семантике аналогичны данным отношениям в диаграмме классов.

Отношение расширения определяет взаимосвязь экземпляров отдельного варианта использования с более общим вариантом, свойства которого определяются на основе способа совместного объединения данных экземпляров. В метамодели отношение расширения является направленным и указывает, что применительно к отдельным примерам некоторого варианта использования должны быть выполнены конкретные условия, определенные для расширения данного варианта использования. Так, если имеет место отношение расширения от варианта использования А к варианту использования В, то это означает, что свойства экземпляра варианта использования В могут быть дополнены благодаря наличию свойств у расширенного варианта использования А.

Отношение расширения между вариантами использования обозначается пунктирной линией со стрелкой (вариант отношения зависимости), направленной от того варианта использования, который является расширением для исходного варианта использования. Данная линия со стрелкой помечается ключевым словом "extend" ("расширяет"), как показано на рис. 1.11.

“extend”

Рис. 1.11. Пример графического изображения отношения расширения между вариантами использования

Отношение включения между двумя вариантами использования указывает, что некоторое заданное поведение для одного варианта использования включается в качестве составного компонента в последовательность поведения другого варианта использования. Данное отношение является направленным бинарным отношением в том смысле, что пара экземпляров вариантов использования всегда упорядочена в отношении включения.

Семантика этого отношения определяется следующим образом. Когда экземпляр первого варианта использования в процессе своего выполнения достигает точки включения в последовательность поведения экземпляра второго варианта использования, экземпляр первого варианта использования выполняет последовательность действий, определяющую поведение экземпляра второго варианта использования, после чего продолжает выполнение действий своего поведения. При этом предполагается, что даже если экземпляр первого варианта использования может иметь несколько включаемых в себя экземпляров других вариантов, выполняемые ими действия должны закончиться к некоторому моменту, после чего должно быть продолжено выполнение прерванных действий экземпляра первого варианта использования в соответствии с заданным для него поведением (рис. 1.12).

“include”

Рис. 1.12. Пример графического изображения отношения включения между вариантами использования

### 1.1.3 Диаграмма состояний

Для моделирования поведения на логическом уровне в языке UML могут использоваться сразу несколько канонических диаграмм: состояний, деятельности, последовательности и кооперации, каждая из которых фиксирует внимание на отдельном аспекте функционирования системы. В отличие от других диаграмм диаграмма состояний описывает процесс изменения состояний только одного класса, а точнее - одного экземпляра определенного класса, т. е. моделирует все возможные изменения в состоянии конкретного объекта. При этом изменение состояния объекта может быть вызвано внешними воздействиями со стороны других объектов или извне. Именно для описания реакции объекта на подобные внешние воздействия и используются диаграммы состояний.

Главное предназначение этой диаграммы - описать возможные последовательности состояний и переходов, которые в совокупности характеризуют поведение элемента модели в течение его жизненного цикла. Диаграмма состояний представляет динамическое поведение сущностей, на основе спецификации их реакции на восприятие некоторых конкретных событий. Системы, которые реагируют на внешние действия от других систем или от пользователей, иногда называют реактивными. Если такие действия инициируются в произвольные случайные моменты времени, то говорят об асинхронном поведении модели.

Автомат (state machine) в языке UML представляет собой некоторый формализм для моделирования поведения элементов модели и системы в целом. Он описывает поведение отдельного объекта в форме последовательности состояний, которые охватывают все этапы его жизненного цикла, начиная от создания объекта и заканчивая его уничтожением. Каждая диаграмма состояний представляет некоторый автомат.

Простейшим примером визуального представления состояний и переходов на основе формализма автоматов может служить ситуация с исправностью технического устройства, такого как компьютер. В этом случае вводятся в рассмотрение два самых общих состояния: "исправен" и "неисправен" и два перехода: "выход из строя" и "ремонт". Графически эта информация может быть представлена в виде изображенной ниже диаграммы состояний компьютера (рис. 1.13).



Рис. 1.13. Простейший пример диаграммы состояний для технического устройства типа компьютер

Основными понятиями, входящими в формализм автомата, являются состояние и переход. Главное различие между ними заключается в том, что длительность нахождения системы в отдельном состоянии существенно превышает время, которое затрачивается на переход из одного состояния в другое. Предполагается, что в пределе время перехода из одного состояния в другое равно нулю (если дополнительно ничего не сказано). Другими словами, переход объекта из состояния в состояние происходит мгновенно.

В языке UML под состоянием понимается абстрактный метакласс, используемый для моделирования отдельной ситуации, в течение которой имеет место выполнение некоторого условия. Состояние может быть задано в виде набора конкретных значений атрибутов класса или объекта, при этом изменение их отдельных значений будет отражать изменение состояния моделируемого класса или объекта.

Состояние на диаграмме изображается прямоугольником со скругленными вершинами. Этот прямоугольник, в свою очередь, может быть разделен на две секции горизонтальной линией (рис. 1.14, б). Если указана лишь одна секция, то в ней записывается только имя состояния (рис. 1.14, а). В противном случае в первой из них записывается имя состояния, а во второй - список некоторых внутренних действий или переходов в данном состоянии. При этом под действием в языке UML понимают некоторую атомарную операцию, выполнение которой приводит к изменению состояния или возврату некоторого значения (например, "истина" или "ложь").

Имя состояния

Имя состояния

Список внутренних действий в данном состоянии

(а) с одной секцией (б) с двумя секциями

Рис. 1.14. Графическое изображение состояний на диаграмме состояний

Начальное состояние представляет собой частный случай состояния, которое не содержит никаких внутренних действий (псевдосостояния). В этом состоянии находится объект по умолчанию в начальный момент времени. Оно служит для указания на диаграмме состояний графической области, от которой начинается процесс изменения состояний. Графически начальное состояние в языке UML обозначается в виде закрашенного кружка (рис. 1.15, а), из которого может только выходить стрелка, соответствующая переходу.

Конечное (финальное) состояние представляет собой частный случай состояния, которое также не содержит никаких внутренних действий. В этом состоянии будет находиться объект по умолчанию после завершения работы автомата в конечный момент времени. Оно служит для указания на диаграмме состояний графической области, в которой завершается процесс изменения состояний или жизненный цикл данного объекта. Графически конечное состояние в языке UML обозначается в виде закрашенного кружка, помещенного в окружность (рис. 1.15, б), в которую может только входить стрелка, соответствующая переходу.

(а) начальное состояние (б) конечное состояние

Рис. 1.15. Графическое изображение начального и конечного состояний на диаграмме состояний

Выражение действия (action expression) выполняется в том и только в том случае, когда переход срабатывает. Представляет собой атомарную операцию (достаточно простое вычисление), выполняемую сразу после срабатывания соответствующего перехода до начала каких бы то ни было действий в целевом состоянии. Атомарность действия означает, что оно не может быть прервано никаким другим действием до тех пор, пока не закончится его выполнение. Данное действие может оказывать влияние, как на сам объект, так и на его окружение, если это с очевидностью следует из контекста модели.

## Технология обработки информации

**1.2.**1 Каркас графического представления

Для реализации графического редактора необходимо использовать каркас графического представления определенного, в библиотеке qt-4.7.

Графическое представление обеспечивает поверхность для управления и взаимодействия с большим количеством созданных пользователем двумерных графических объектов, а также область редактирования графического редактора с поддержкой масштабирования и вращения.

Элементы сцены должны будут обрабатывать события клавиш, нажатие, перемещение, отпускание мыши и двойные щелчки.

Графическое представление предоставляет подход к программированию называемый «модель-представление». Несколько представлений могут отображать одну и ту же сцену, а сцена может содержать элементы самой разнообразной геометрической формы.

Сцена графического представления обеспечивает следующие возможности:

* Предоставляет быстрый интерфейс для управления большим количеством элементов.
* Передает сообщения каждому элементу.
* Управляет состояниями элемента, таким как выбор или обработка фокуса.
* Обеспечивает неизменяемую функциональность отрисовки; главным образом для печати.

Сцена является контейнером визуализированных объектов. Представление – это участок области окна, в которой визуализируется содержимое сцены. Визуализированный элемент – объект базового класса для графических элементов сцены. Графическое представление предоставляет несколько стандартных элементов для обычных фигур, но для поставленной задачи необходимо создать пользовательский элемент, сочетающий в себе прямоугольник и текстовые поля.

Пользователь должен иметь возможность выполнять следующие действия: создавать и редактировать диаграммы различных типов (т.е. добавлять на диаграмму элементы, редактировать соответствующие поля этих элементов и соединять их отношениями), а также сохранять диаграммы.

Диаграмма вариантов использования представлена на рис.1.16.

Пользователь

Рис. 1.16. Диаграмма вариантов использования данного программного обеспечения

В соответствии с идеологией Qt нужно создать класс MainWindow, который поместит область редактирования графического редактора на главной форме, а затем передаст ввод данной области сцене. Этот класс должен обновлять область редактирования, когда изменится какой-либо элемент диаграммы или последний добавиться на сцену. Класс также будет удалять визуализированные объекты со сцены и управлять порядком наложения (порядок, в котором элементы отрисовываются, когда они накладываются друг на друга).

В сцене клик мыши должен производить три разных действия:

* элемент под курсором может быть перемещён;
* элемент может быть вставлен;
* отношение может соединить элементы диаграммы.

Каждое действие мыши зависит от режима, в котором находится сцена.

Сцена также устанавливает шрифт её текстовых элементов. Таким образом, класс MainWindow, который будет размещать область редактирования графического редактора на главной форме и сцена делят ответственность за функциональность всего программного обеспечения.

MainWindow предназначен для решения следующих задач:

* удаление элементов, текста и стрелок;
* перемещение элементов диаграммы на заднюю или переднюю область;
* установка масштаба сцены.
* сохранение диаграммы в файл.

На сцене недопустимы текстовые элементы, не содержащие текста, следовательно, необходимо проверять каждый текстовый элемент, добавляемый на сцену на содержание в нем текста, и если проверяемый элемент окажется пустым нужно удалить его. При добавлении нового элемента диаграммы на сцену, пользователь должен зафиксировать его место, кликом мыши в области представления сцены. Чтобы соединить два элемента отношением нужно выбрать тип отношения в меню, затем зафиксировать начало отношения, выбрав начальный элемент на сцене, кликом мыши, и не отпуская левой клавиши мышки переместить курсор на конечный элемент, после чего отпустить клавишу.

Нужно предусмотреть выход из программы, как с сохранением диаграммы в файл, так и без него.

Необходимо создать класс «визуализированный элемент», который будет представлять, какой-либо визуализированный объект на сцене. Данный класс должен иметь список стрелок, которые соединены с его объектом. Это необходимо, так как только элемент знает, когда он удаляется, чтобы во время удаления он смог обновить стрелки.

Также нужно определить отдельный класс «визуализированный текстовый элемент», так как поведение данного объекта отличается от предыдущих элементов. В нем необходимо переопределить событие «двойной клик», чтобы при этом пользователь вошел в режим редактирования текстового поля.

Затем необходимо создать класс «линия», который представляет собой графический элемент, соединяющий между собой два визуализированных объекта. Что бы добиться этого, этот элемент должен отрисовывать себя сам и также переопределить методы, используемые графической сценой для проверки столкновений и выделений графических элементов между собой. При перемещении элементов на сцене, «линия» должна перемещаться вместе с объектами, которые она соединяет.

### 1.2.2 Алгоритм рисования

Функция paint() предназначена для перерисовки отношения, которое производится следующим образом:

Если начальный и конечный элемент сталкиваются, то отношение не рисуется; алгоритм, используемый для нахождения точек стрелки, может выдать некорректные значения, если элементы сталкиваются. Для отрисовки определенного типа отношения, сначала находится позиция, где будет отрисован наконечник. Он должен быть отрисован, где линия и конечный элемент пересекаются. Точка находится путем проверки на пересечение каждой стороны многоугольника составляющего границы конечного элемента на диаграмме. Так как начальная и конечная точки отношения установлены в центр элементов, линия должна пересекать одну и только одну сторону многоугольника. Далее вычисляется угол между осью ОХ и отношением, это делается, потому что нужно направить наконечник стрелки по направлению линии. Затем можно вычислить три точки составляющие многоугольник, который описывает фигуру наконечника. Координаты первой точки (,), координаты второй точки (, ), где - найденный угол. Если пользователь выделил отношение на диаграмме, функция рисует две пунктирные линии чтобы визуально выделить отношение, это так же является переопределением стандартной унаследованной функции, так как её реализация использует функцию boundingRect(), но данный прямоугольник не подходит, так как выделяет область гораздо больше, чем область выделения выделенного отношения.

### 1.2.3 Сохранение диаграммы в файл

Сохранение диаграммы в файл должно производиться следующим образом: необходимо вывести диалоговое окно, где пользователь укажет имя файла и выберет формат сохраняемого файла (в данном случае \*.dia). Затем нужно считать элементы, хранящиеся на сцене и в зависимости от типа элемента сцены записать информацию об этом элементе (позицию элемента на сцене, значение текстовых полей, Z – значение, и т.д.) в бинарный поток QDataStream определенном в стандартной библиотеке Qt-4.7. Затем этот поток записывается в файл.

### 1.2.4 Чтение диаграммы из файла

Чтение диаграммы из файла производится аналогичным образом. Сначала читается тип сохраненного элемента, и в зависимости от типа производится чтение информации об этом элементе (позицию элемента на сцене, значение текстовых полей, Z – значение, и т.д.) и затем он добавляется на сцену.

### 1.2.5 Формат

Сохранение диаграммы в файл предполагается производить следующим образом: запросить у пользователя полное имя файла. Затем нужно записать информацию о каждом элементе сцены (позицию элемента на сцене, значение текстовых полей), в зависимости от его типа в файл формата «dia». В данный формат информация записывается следующим образом:

* сначала записывается идентификатор записываемого объекта (поле типа int);
* позиция данного объекта на сцене (поле типа QPointF);
* тип объекта (поле типа int);
* значение видимости элемента относительно других элементов на сцене (поле типа double);
* значения текстовых полей (поле типа QString).

Если данный элемент отношение то еще записывается начальный и конечный элементы, которые оно соединяет.

### 1.2.6 Архитектура программы

В графическом редакторе должно быть две подсистемы: моделирования и визуализации данных. Каждая из этих подсистем должна иметь в наличии набор встроенных графических объектов, необходимых для решения своей задачи. Подсистема моделирования должна использоваться для интерактивного построения топологии UML диаграммы. Подсистема визуализации данных будет необходима для наглядного отображения информации пользователю.

Подсистема моделирования графического редактора должна выполнять ряд задач для построения топологии диаграммы UML. Начальная стадия построения топологии диаграммы будет состоять в компоновке пользователем элементов, составляющих диаграмму. На этой стадии графический редактор будет выступать в роли конструктора, который позволит посредством встроенных графических объектов моделирования задавать построение диаграммы UML. Рабочая область, в которой произойдет размещение графических объектов, называется областью редактирования графического редактора. Пользователь интерактивно может формировать топологию диаграммы UML посредством распределения и связывания графических объектов друг с другом в области редактирования редактора. По завершении компоновки диаграммы UML в области редактирования формируется графическое представление модели диаграммы UML.

Следующим шагом в построении диаграммы UML будет являться задание отдельных графических элементов, составляющих диаграмму UML. Изменение параметров графических объектов моделирования должно осуществляться непосредственно в области редактирования графического редактора, создание и обработка событий которого возложены на подсистему визуализации данных и отдельные ее компоненты.

Заключительной стадией в построении диаграммы UML будет, является настройка подсистемы визуализации данных, состоящая в размещении графических объектов визуализации в области редактирования графического редактора и задании параметров этих объектов. Параметры графических объектов визуализации можно разделить на две категории: параметры, отвечающие за внешний вид элемента (цвет, размер и т.д.), и параметры, определяющие «внутренние» свойства графического объекта (название полей, методов класса и т.д.).

Подсистема визуализации данных графического редактора должна решать задачу наглядного отображения данных непосредственно в области редактирования изображения. В качестве элементов визуализации в графическом редакторе будут применяться графические объекты визуализации. Эти объекты обладают рядом свойств (размер, цвет), изменение которых позволяет сфокусировать внимание пользователя на наиболее важных данных. Кроме того, графические объекты визуализации могут быть свободно перемещены по области редактирования. Данное свойство графических объектов визуализации позволяет группировать сходные по некоторому признаку объекты в одном месте области редактирования.

### 1.3 Инфологическая модель

По результатам анализа предметной области была разработана инфологическая модель задачи, представленная на соответствующей диаграмме классов (рис. 1.17).

Модель содержит классы «My\_diagram\_item», «My\_diagram\_scene» и «My\_diagram\_arrow», представляющие объекты предметной области. Для обеспечения целостности всего программного продукта создан класс «MainWindow», управляющий согласованностью состояний и поведений всего множества объектов предметной области. Он позволяет учитывать ограничения на значения полей одних объектов в зависимости от состояния других объектов.

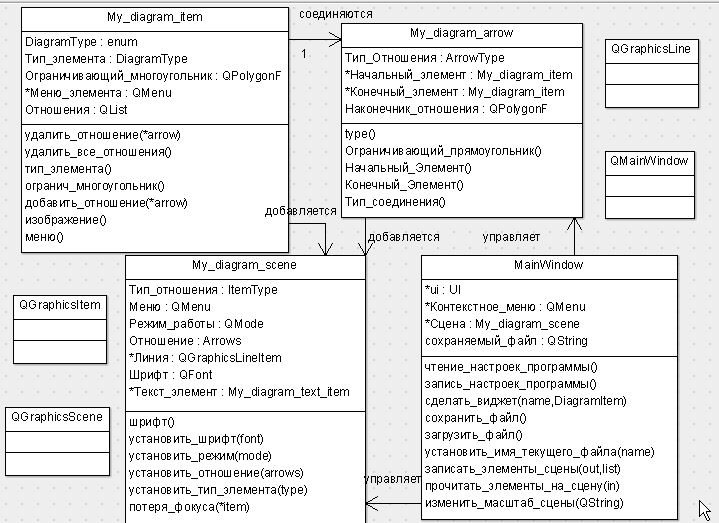


Рис. 1.17. Диаграмма классов данного программного обеспечения

В классе My\_diagram\_item присутствует перечисление DiagramType определяющее тип создаваемого объекта. Аналогично и в классе My\_diagram\_arrow присутствует перечисление ArrowType. Класс My\_diagram\_scene имеет перечисление Mode, которое определяет, в каком режиме находится сцена в определенный момент времени работы программного обеспечения.

### 1.4 Минимальные системные требования

Аппаратные требования:

* Процессор с частотой 1,66 ГГц или более.
* 256 Mb ОЗУ.
* 10 Mb свободного места на жестком диске.
* Монитор Super VGA с разрешением 800х600 точек или более высоким, поддерживающий 256 цветов.

Требования к операционной системе: Microsoft Windows XP/7.

# РАБОЧИЙ ПРОЕКТ



### 2.1 Общие сведения о работе программы

В результате выполнения курсовой работы был создан графический редактор, позволяющий строить UML-диаграммы, и сохранять их в файл. Программа разработана в интегрированной среде программирования Qt Creator 2.0.1 на языке программирования С++ и работает под управлением операционной системы Windows.

### 2.2 Функциональное назначение

Программа обеспечивает следующие функциональные возможности: создание, редактирование и сохранение в файл UML-диаграмм.

Программа имеет следующие функциональные ограничения:

* Невозможность изменять размеры элементов диаграмм.
* Невозможность изменять место прикрепления различных видов отношений.
* Невозможность изгибать линию, представляющую отношения.

## 2.3 Инсталляция и выполнение

Для того чтобы проинсталлировать программу, запустить с диска на выполнение файл “setup\_dia.exe” (рис. 2.1).

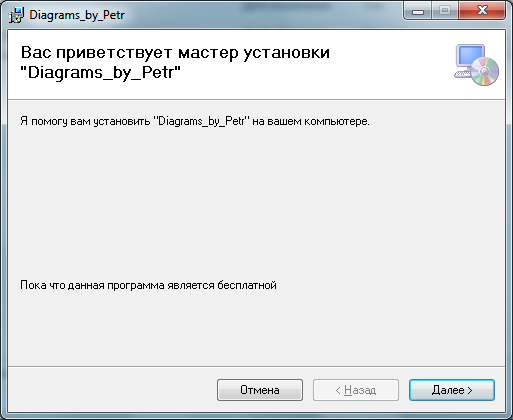


Рис. 2.1. Стартовое окно установщика ПО

Для продолжения установки нажмите по кнопке «Далее» (рис. 2.2), выбрав папку в которую будет установлена программа и режим установки, нажать по кнопке «Далее»

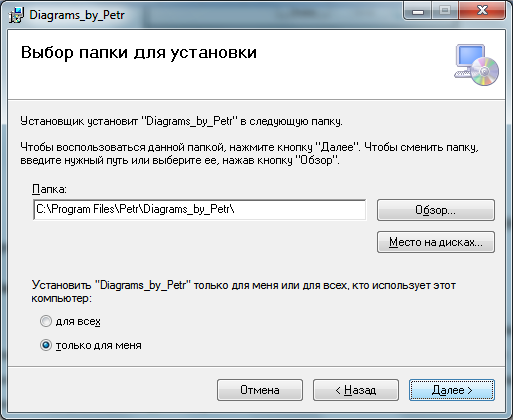


Рис. 2.2. Окно, предлагающее выбрать папку, в которую будет установлена программа

Далее необходимо подождать пока установщик переместит файлы необходимые для корректной работы программы в указанную папку.

После завершения работы установщика будет выведено диалоговое окно (рис. 2.3).

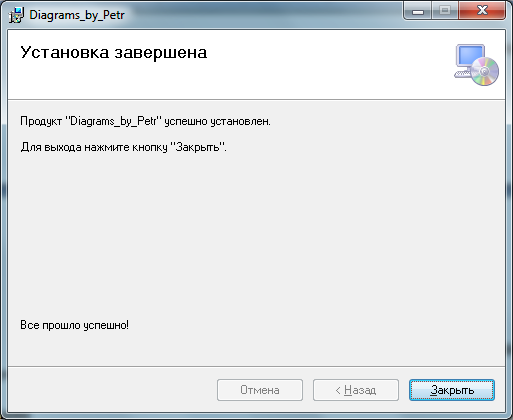


Рис. 2.3 Завершение установки



### 2.4 Общий алгоритм программного продукта

Для решения поставленной задачи был использован каркас графического представления представленный в Qt.

**Сцена**

Сцена выступает в качестве контейнера для элементов, добавляемых на неё. В стандартной библиотеке Qt сцена представлена классом QGraphicsScene, для решения поставленной задачи не достаточно стандартных функций, поэтому необходимо реализовать свою сцену, унаследовав от библиотечной.

**Представление**

Вид (view) собой виджет, который визуализирует содержимое сцены. Представление получает события ввода данных с клавиатуры и мыши и переводит их в события сцены (преобразуя, где необходимо, используемые координаты в координаты сцены), перед отправкой событий в визуализируемую сцену.

**Элемент**

Для реализации элемента, который добавляется на сцену необходимо унаследовать от стандартного класса QGraphicsPoligonItem и переопределить в нем некоторые функции. Класс QGraphicsPoligonItem поддерживает следующие возможности:

События нажатия кнопки мыши, перемещение, отпускание и двойное нажатие кнопки мыши, так же как и события наведения мыши, события колесика мыши и события контекстного меню.

* Фокус ввода с клавиатуры и события клавиш
* Перетаскивание (Drag and Drop)
* Группировку, как через родительско-дочерние отношения, так и через [QGraphicsItemGroup](http://doc.crossplatform.ru/qt/4.5.0/qgraphicsitemgroup.html)
* Обнаружение столкновений

Элементы существуют в локальной системе координат.

### 2.4.1 Структура программного проекта

В ходе работы над данным проектом было принято решение разделить текст программы на отдельные файлы для каждого разработанного класса.

Модули программного продукта и их назначение представлены в табл. 2.1.

Таблица 2.1

|  |  |
| --- | --- |
| **Модули программного продукта** | |
| Имя файла | Назначение |
| my\_diagram\_scene.h | Содержит интерфейс класса My\_diagram\_scene, который представляет собой сцену |
| my\_diagram\_scene.cpp | Содержит реализацию методов класса My\_diagram\_scene |
| Имя файла | Назначение |
| my\_diagram\_arrow.h | Содержит интерфейс класса My\_diagram\_arrow, который представляет собой некоторый отношение |
| my\_diagram\_arrow.срр | Содержит реализацию методов класса My\_diagram\_arrow |
| my\_diagram\_item.h | Содержит интерфейс класса My\_diagram\_item, который представляет собой некоторый элемент диаграммы |
| my\_diagram\_item.cpp | Содержит реализацию методов класса My\_diagram\_item |
| my\_diagram\_text\_item.h | Содержит интерфейс методов класса My\_diagram\_text\_item, который представляет собой текстовый элемент диаграммы |
| my\_diagram\_text\_item.срр | Содержит реализацию методов класса My\_diagram\_text\_item |
| mainwindow.h | Содержит интерфейс методов класса Mainwindow, который представляет собой главное окно программного продукта |
| Mainwindow.cpp | Содержит реализацию методов класса Mainwindow |

### 2.4.2 Функциональное назначение модулей, классов, объектов, методов

Класс MainWindow создаёт и размещает виджеты в главном окне программы, он также передаёт введенные пользователем данные в класс My\_diagram\_scene. Он обновляет свои виджеты, когда изменяется какой-либо элемент диаграммы или текстовый элемент добавляется на сцену.

Данный класс удаляет элементы со сцены и определяет порядок, отрисовки элементов, когда они наложены друг на друга.

В конструкторе вызываются методы для создания виджетов и компоновок, а также создается сцена диаграммы. Панель инструментов должна быть создана после сцены, так как она соединяется с её сигналами.

Каждой кнопке из buttonGroup присваивается уникальный идентификационный номер, равный типу элементов диаграммы, заданных через перечисление DiagramType в классе My\_diagram\_item.

Слот on\_On\_delete\_triggered() удаляет выбранный элемент со сцены. Если удаляемый элемент является классом или интерфейсом, то данный слот удаляет стрелки, которые с ним соединены.

Функция pointerGroupClicked() ставит соотвествующий режим сцене, взависимости от того какая кнопка была нажата.

Описание методов класса MainWindow представлено в табл. 2.2.

Таблица 2.2

|  |  |
| --- | --- |
| **Описание методов класса MainWindow** | |
| Прототип | Назначение |
| void on\_Italic\_triggered() | Делает выделенный текст курсивным |
| void on\_Underline\_triggered() | Делает выделенный текст подчеркнутым |
| void on\_Bold\_triggered() | Делает выделенный текст жирным |
| void on\_On\_front\_triggered() | Ставит выделенный на диаграмме элемент на передний план |
| void on\_On\_back\_triggered() | Ставит выделенный на диаграмме элемент на задний план |
| void on\_On\_delete\_triggered() | Удаляет выбранный элемент с диаграммы |
| void fontSizeChanged(const QString &size) | Изменяет размер шрифта выделенного текста, делегируя свою работу методу handleFontChange() |
| void currentFontChanged(const QFont &font) | Изменяет стиль шрифта выделенного текста, делегируя свою работу методу handleFontChange() |
| Прототип | Назначение |
| void handleFontChange() | Изменяет стиль и размер шрифта выделенного текста, а также делает его курсивным, жирным или подчеркнутым в зависимости от того какое действие выбирал пользователь в момент вызова метода |
| void sceneScaleChanged(QString) | Изменяет масштаб сцены |
| void pointerGroupClicked(int id) | Определяет какой тип отношений сцены будет отрисован в данный момент |
| void buttonGroupClicked(int id) | Определяет режим работы сцены (просто движение, вставка коментария, класса или интерфейса) |
| void itemInserted(My\_diagram\_item \*item) | Вставляет класс или интерфейс, устанавливая сцене сответствующий режим |
| void textInserted(QGraphicsTextItem \*item) | Вставляет коментарий, устанавливая сцене сответствующий режим |
| void itemSelected(QGraphicsItem \*item) | Срабатывает тогда, когда пользователь выделяет элемент на диаграмме, выводит сведения о нем размер, стиль и св-ва шрифта |
| void readSettings() | Считывает предыдущие настройки окна |
| void writeSettings() | Записывает предыдущие настройки окна |
| void add\_in\_button\_group(My\_diagram\_item::DiagramType type) | Осуществляет групировку кнопок |
| Прототип | Назначение |
| QWidget \*createCellWidget(const QString &text,My\_diagram\_item::DiagramType type) | Создаёт виджет, содержащий кнопку с изображением одного из элементов диаграммы и её название. Затем он используется как полноценная кнопка в панели инструментов. |
| bool maybeSave() | Определяет, сохранена ли данная диаграмма |
| bool saveFile(const QString &fileName) | Сохраняет в файл данную диаграмму |
| void loadFile(const QString &fileName) | Загружает из файла диаграмму |
| void setCurrentFile(const QString &fileName) | Устанавливает имя текущего файла |

Класс My\_diagram\_arrow представляет собой графический элемент, соединяющий два элемента диаграммы (либо это класс, либо интерфейс). Он рисует определенный тип отношения, направленного к одному из соединяемых элементов. Класс наследует от стандартного класса QGraphicsLineItem, который рисует линию и перемещается с элементами, которые он соединяет. Чтобы отрисовать My\_diagram\_arrow необходимо переопределить методы, используемые графической сценой для проверки столкновений и выделений.

В конструкторе устанавливается начальный и конечный элемент отношения, наконечник будет отрисован там, где линия пересекается с конечным элементом или с начальным в зависимости от типа отношения.

Функцию boundingRect() необходимо переопределить, так как к линии прибавляется еще наконечник, что больше ограничивающего прямоугольника QGraphicsLineItem. Графическая сцена использует ограничивающий прямоугольник для определения, какие области сцены надо перерисовать.

Функция shape() возвращает фигуру отношения, которая в свою очередь используется для проверки на столкновения и выделения с помощью мышки.

Описание методов класса My\_diagram\_arrow представлено в табл. 2.3.

Таблица 2.3

|  |  |
| --- | --- |
| **Описание методов класса My\_diagram\_arrow** | |
| Прототип | Назначение |
| My\_diagram\_item \*startItem() const | Возвращает объект к которому прикреплено начало линии, изображающую какой-либо тип отношения |
| My\_diagram\_item \*endItem() const | Возвращает объект к которому прикреплен конец линии, изображающую какой-либо тип отношения |
| QRectF boundingRect() const | Предоставляет ограничивающий прямоугольник, который дает возможность графической сцене узнать какие именно области сцены надо перерисовать |
| QPainterPath shape() const | Возвращает точную фигуру линии, которая используется для проверки на столкновения и выделения с помощью мышки. |
| int type() const | Выдает тип отношения между элементами |
| void updatePosition() | Обновляет координаты линии |

My\_diagram\_item представляет собой элементы, добавляемые на сцену. Он является наследником от QGraphicsPolygonItem. Перечисление enum DiagramType предоставляет идентификационное значение для каждого элемента диаграммы. Класс имеет список стрелок, которые с ним соединены это необходимо, чтобы можно было удалить стрелки со сцены, когда удалится элемент.

В конструкторе рисуем форму элемента в соответствии с типом элемента, переданного при создании элемента.

Описание методов класса My\_diagram\_item представлено в табл. 2.4 .

Таблица 2.4

|  |  |
| --- | --- |
| **Описание методов класса My\_diagram\_item** | |
| Прототип | Назначение |
| removeArrow(My\_diagram\_arrow \*arrow) | Используется для удаления линий, когда они или элементы, которые они соединяют, удаляются со сцены. |
| My\_diagram\_item \*endItem() const | Эта функция вызывается, когда элемент удаляется со сцены и удаляет все линии, которые с ним соединены. Линия удалится у начального и конечного элементов. |
| void addArrow(My\_diagram\_arrow \*arrow) | Добавляет стрелку на сцену |
| QPixmap image() const | Рисует изображение элемента в Pixmap который используется для создания иконок кнопок панели инструментов |
| void contextMenuEvent(QGraphicsSceneContextMenuEvent \*event) | Показывает контекстное меню и выделяет элемент для того чтобы можно было осущетвить переход на передний или задний план выбранного элемента относительно случайного перекрывающего данный |
| QVariant itemChange(GraphicsItemChange change, const QVariant &value) | Обновляет координаты линии соединяющую данный элемент |

Класс My\_diagram\_text\_item наследует от стандартного класса QGraphicsTextItem и добавляет возможность перемещать редактируемые текстовые элементы. Стандартный класс спроектирован так, что бы его объект фиксировался на определенном месте на сцене и его редактирование начиналось при одиночном клике пользователя на этом элементе. В My\_diagram\_text\_item редактирование начинается после двойного клика, оставляя одиночный клик для перемещения элемента на сцене.

Функция itemChange() испускает сигнал selectedChanged, когда элемент становится выделенным, MainWindow использует этот сигнал для обновления виджетов, которые отображают свойства шрифта, используя параметры шрифта выделенного текстового элемента.

Функция focusOutEvent() испускает сигнал lostFocus, когда элемент теряет фокус на сцене, данный сигнал использует My\_diagram\_scene, что бы удалить текстовый элемент, если он пустой, т.е. он не содержит текст.

Описание методов класса My\_diagram\_text\_item представлено в   
табл. 2.5.

Таблица 2.5

|  |  |
| --- | --- |
| **Описание методов класса My\_diagram\_text\_item** | |
| Прототип | Назначение |
| My\_diagram\_text\_item(QGraphicsItem \*parent = 0, QGraphicsScene \*scene = 0, ItemTextType itemTextType = Label, NameTextType nameTextType = Class ) | Конструктор, в котором itemTextType – тип текстового поля (комментарий или поле в классе) nameTextType – определяет тип этого поля (имя класса, атрибуты, операции или интерфейс) |
| QVariant itemChange(GraphicsItemChange change, const QVariant &value) | Когда элемент становится выделенным, данный метод испускает сигнал, который принимает слот главного класса для обновления виджетов, которые отображают свойства шрифта, используя параметры шрифта выделенного текстового элемента. |
| void focusOutEvent(QFocusEvent \*event) | Становится активным когда элемент теряет фокус, данный метод испускает сигнал чтобы затем удалить элемент если он не содержит текст |
| void mouseDoubleClickEvent(QGraphicsSceneMouseEvent \*event) | Вызывается при двойном клике мышкой на элементе и делает его редактируемым |

Класс My\_diagram\_scene наследует от стандартного класса QGraphicsScene и добавляет некоторые функции управления элементами диаграммы к функциям, реализованным в стандартном классе. В My\_diagram\_scene клик мыши может произвести три разных действия: элемент под курсором может быть перемещён, элемент может быть вставлен или стрелка может соединить элементы диаграммы. Какое действие будет произведено зависит от режима в котором находится сцена, заданного перечислением Mode.

В конструкторе устанавливаем режим по умолчанию – передвижение элементов.

Функция mousePressEvent() управляет событиями нажатия кнопок мыши в зависимости от того, в каком режиме находится My\_diagram\_scene. Существует несколько вариантов действия функции:

* Если установлен режим «Вставить элемент», просто создается элемент определенного типа и устанавливается на позицию, где был клик мыши.
* Если установлен режим «Вставить линию», рисует линию, её начало фиксируется в точке, где пользователь кликает мышкой, а конец следует за указателем мыши до тех пор, пока кнопка нажата. Когда пользователь отпускает кнопку мыши, линия будет добавлена на сцену в случае, когда под начальной и конечной точкой линии находятся элементы диаграммы, а именно классы или интерфейсы.
* Если установлен режим «Вставить текст», создается комментарий и устанавливается на позицию, где был клик мыши, причем, поверх всех элементов.

Описание методов класса My\_diagram\_scene представлено в   
табл. 2.6.

Таблица 2.6

|  |  |
| --- | --- |
| **Описание методов класса My\_diagram\_scene** | |
| Прототип | Назначение |
| My\_diagram\_scene(QMenu \*itemMenu, QObject \*parent = 0) | Конструктор, в котором по умолчанию устанавливаем режим «передвижение элементов» |
| void setFont(const QFont &font) | Устанавливает шрифт для выбранного текстового элемента |
| Прототип | Назначение |
| void mousePressEvent(QGraphicsSceneMouseEvent \*mouseEvent) | Управляет событием нажатия левой кнопки мыши в зависимости от того в каком режиме находится сцена |
| void mouseReleaseEvent(QGraphicsSceneMouseEvent \*mouseEvent) | Происходит когда пользователь отпускает левую кнопку мыши, в этом случае функция решает возможно ли добавить отношение на сцену, если два выбрано два элемента которые являются классами (или разновидностью классов) и причем это два разных элемента тогда вопрос разрешается положительно |



### Разработанное меню и интерфейсы

При запуске программного обеспечения появляется стартовое окно (рис. 2.4), содержащее главное меню, а также вкладки для построения соответствующих диаграмм.

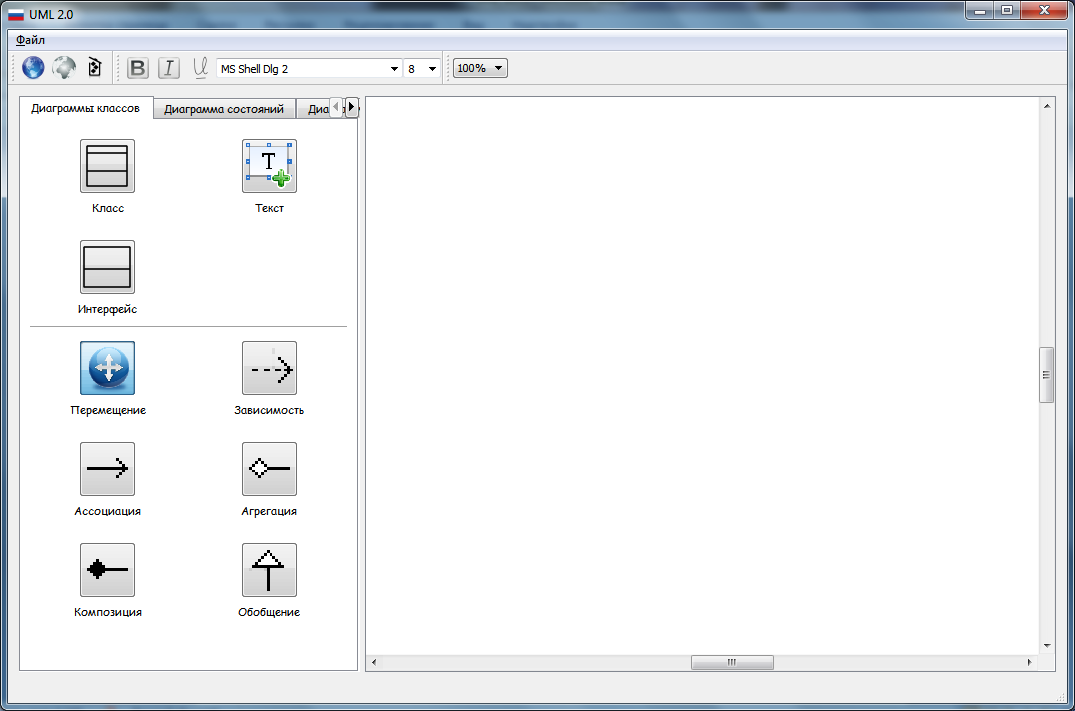


Рис. 2.4 Окно, появляющиеся при запуске программы

В главном меню пользователю предлагается:

* Создать новую диаграмму.
* Сохранить диаграмму.
* Загрузить диаграмму.

В меню создания диаграмм пользователю предоставляется:

* Создать диаграмму классов.
* Создать диаграмму вариантов использования.
* Создать диаграмму состояний.

Для того чтобы постороить диаграмму классов необходимо перейти на вкладку «Диаграмма классов» после чего вниманию пользователя представится стартовое окно  
 (см. рис. 2.4).

Для добавления какого-либо элемента в диаграмму нужно нажать по кнопке соответствующей данному элементу (рис.2.5).

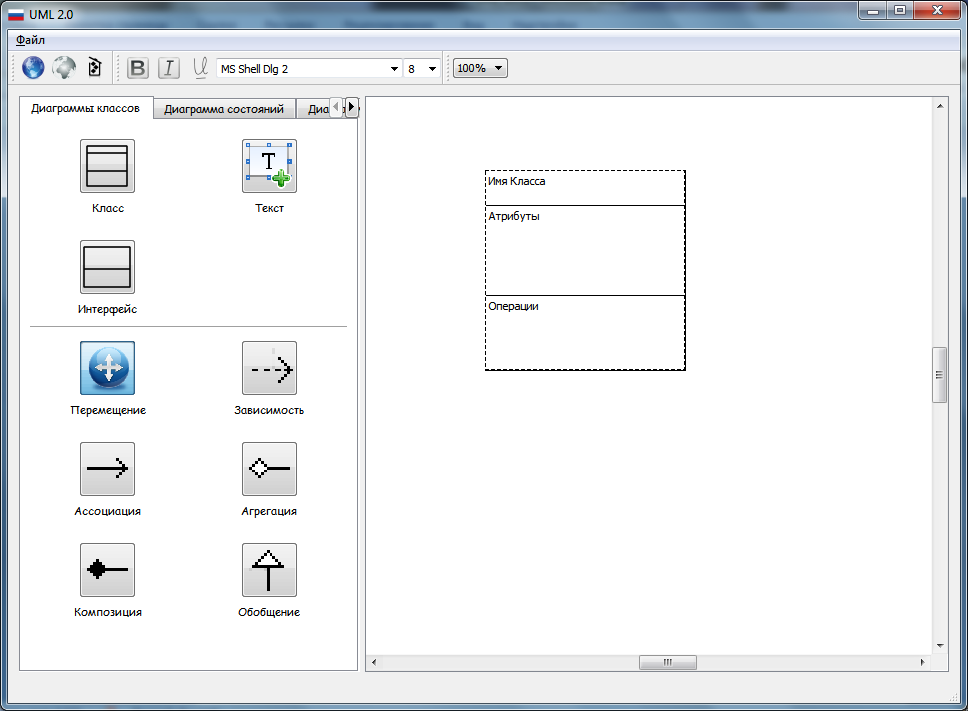


Рис.2.5. Добавление элемента в диаграмму

Для заполнения и редактирования полей элемента необходимо выполнить двойной клик мышкой по редактируемому полю и заполнить соответствующее поле (рис.2.6).

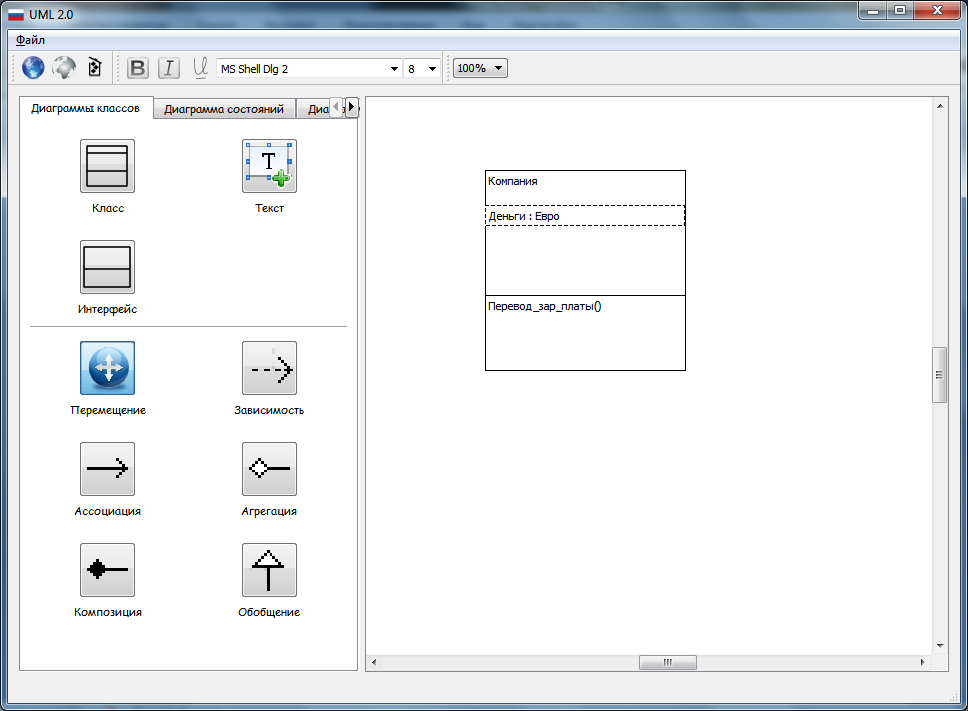


Рис.2.6. Редактирование полей элемента диаграммы

Для того чтобы два элемента диаграммы связать отношением нужно, выбрать соответствующее отношение, а затем провести мышкой воображаемую линию от первого элемента ко второму (рис.2.7).

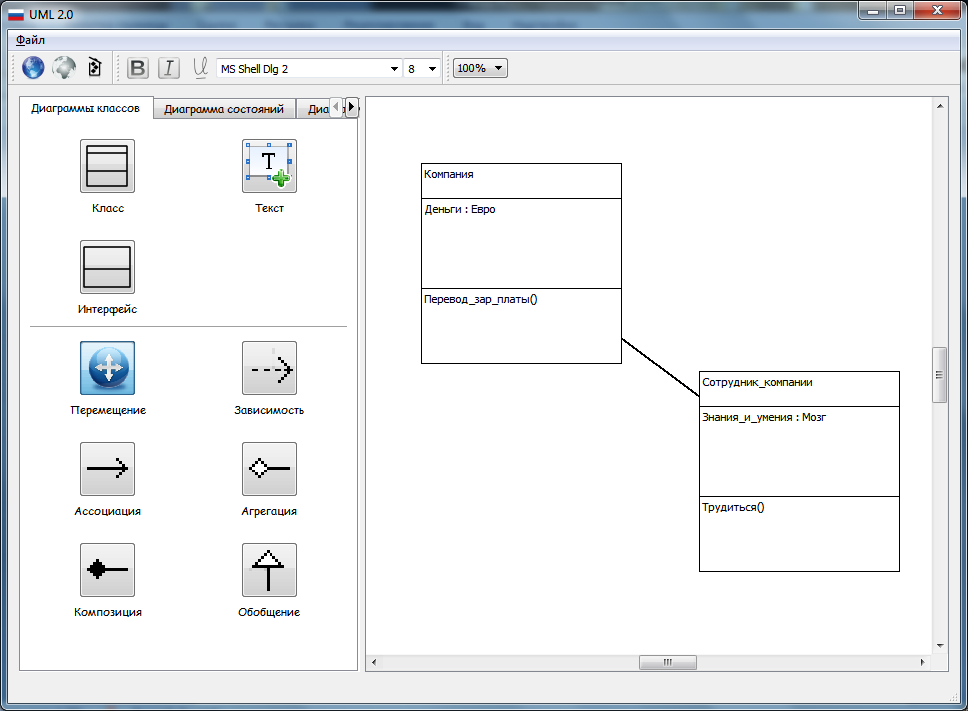


Рис.2.7. Добавление отношения между классами

Для создания диаграммы вариантов использования необходимо переключится на вкладку «Диаграмма вариантов использования» в меню создания диаграмм (рис. 2.8), и затем добавлять элементы в область отображения диаграммы аналогичным способом.

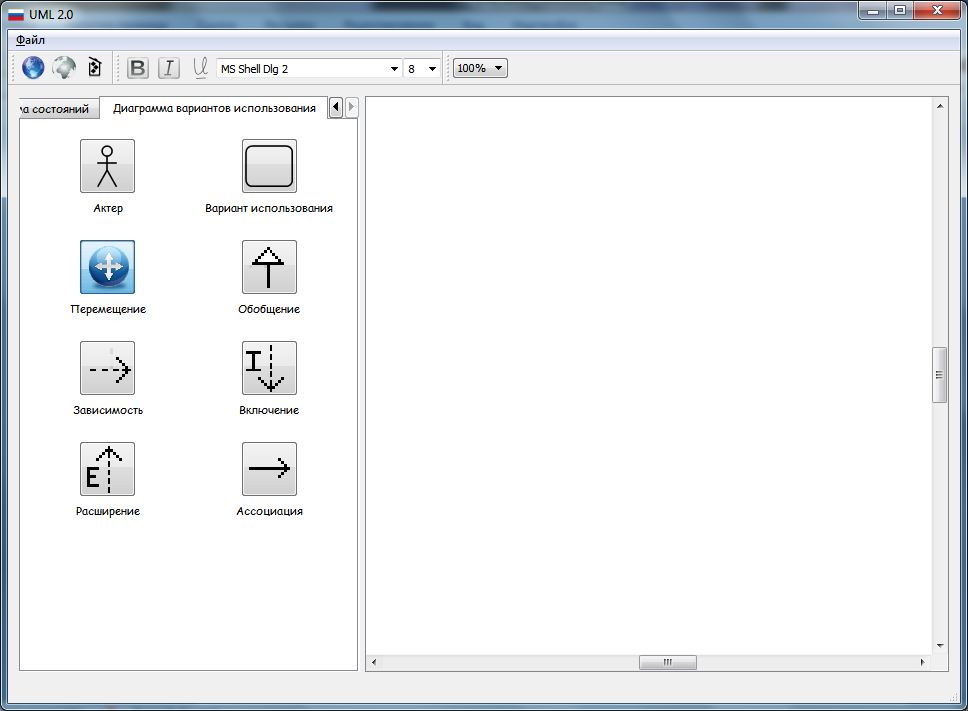


Рис. 2.8. Составление диаграммы вариантов использования

Для создания диаграммы состояний необходимо переключится на вкладку «Диаграмма состояний» в меню создания диаграмм (рис. 2.9), и затем добавлять элементы в область отображения диаграммы аналогичным способом.

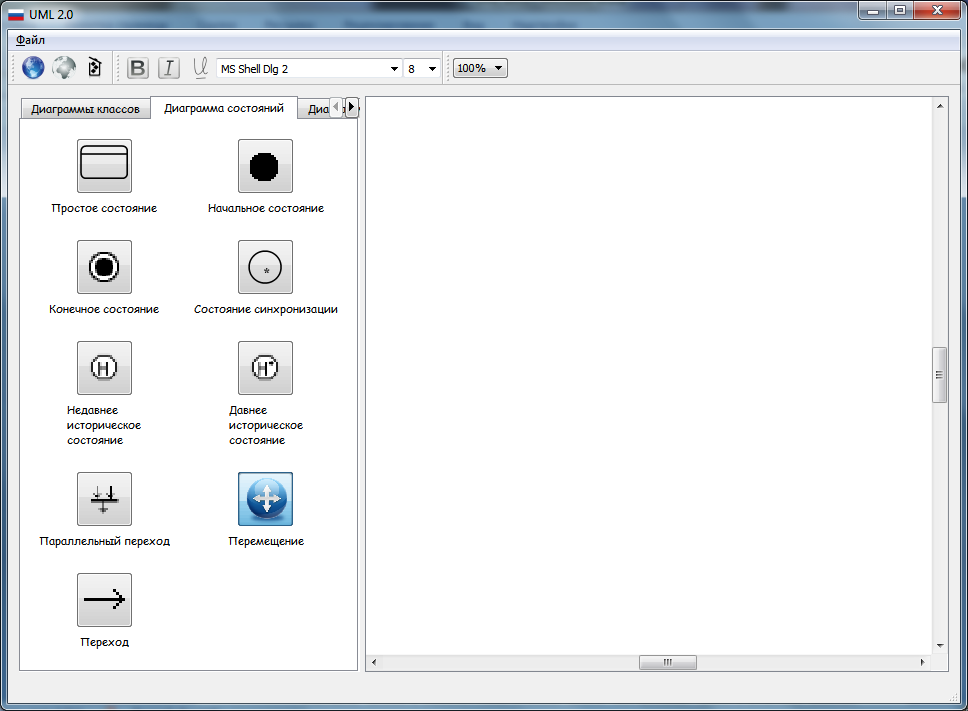


Рис.2.9. Создание диаграммы состояний

Чтобы изменить масштаб диаграммы надо нажать по кнопке изображающей масштаб и выбрать подходящий (рис.2.10).

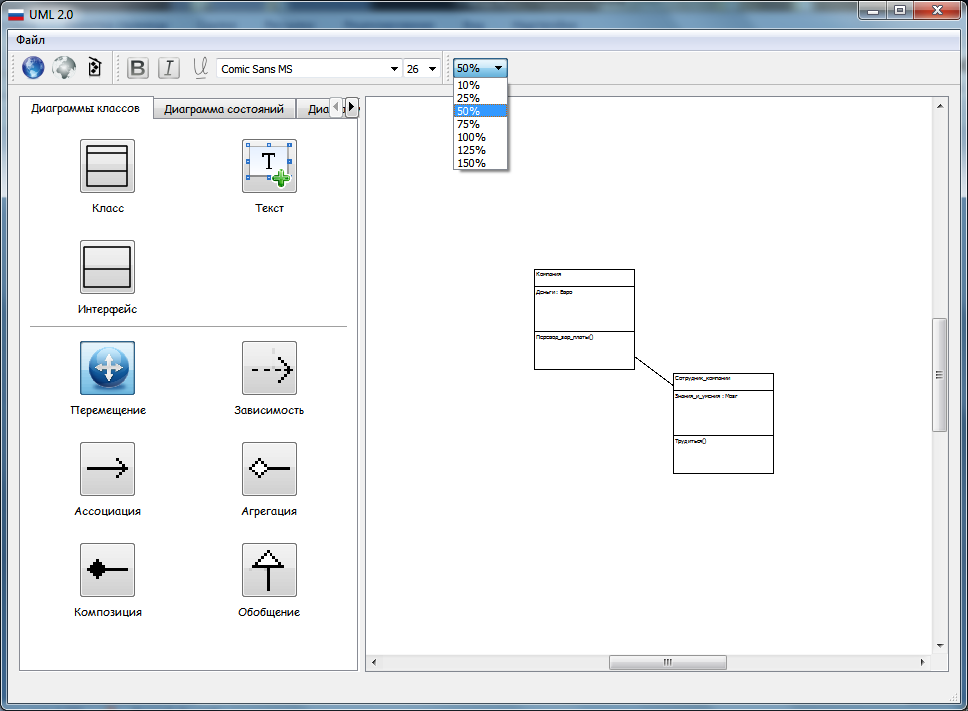


Рис. 2.10. Изменение масштаба диаграммы

Чтобы сохранить созданную диаграмму необходимо в главном меню выбрать пункт «Сохранить» или нажать сочетание клавиш «CTRL + C», затем появится соответствующее диалоговое окно (рис. 2.11).

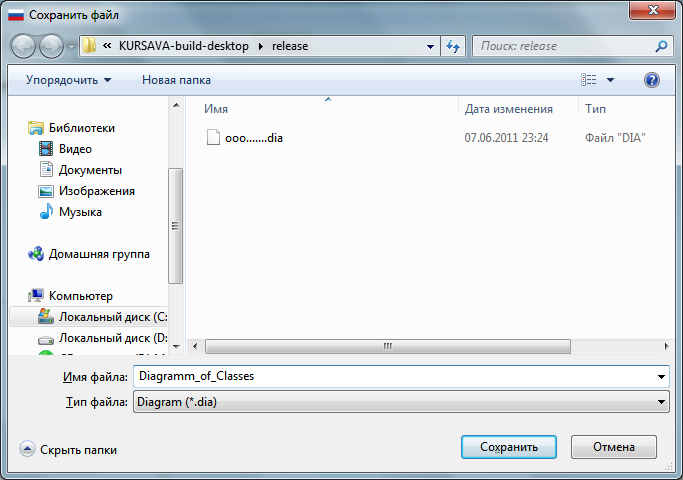


Рис. 2.11. Сохранение диаграммы

При выходе из программы (т.е. при нажатии на кнопку «Х») программа выведет сообщение, в котором попросить пользователя сохранить диаграмму или закончить работу без сохранения или вообще отменить выход из программы (рис.2.12).

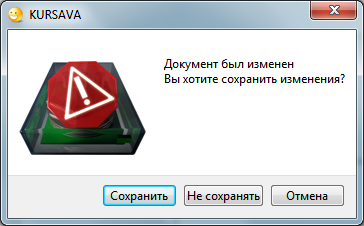


Рис.2.12 Уведомление пользователя о потере данных

1. ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ

При запуске программы появляется стартовое окно (см. рис. 2.1) в котором определено главное меню, а также меню создания диаграмм.

Чтобы проверить корректность программы необходимо сделать следующее:

* Добавить элемент «Класс» в диаграмму (см. рис. 2.5).
* Редактировать поля добавленного элемента, а именно «имя класса», «атрибуты» и «операции» (см. рис. 2.6).
* Добавить еще один элемент под названием «Класс» в диаграмму (см. рис. 2.5).
* Выбрать тип отношения «ассоциация» и последовательно соединить данные классы (см. рис. 2.7).
* Добавить элемент «Текст» и редактировать его поле, затем изменить шрифт и размер текста.
* Удалить элемент, из диаграммы нажав на клавишу «Del» либо выбрав соответствующий пункт в панели инструментов, которая находится под главным меню.
* Добавить элемент «Класс» в диаграмму (см. рис. 2.5).
* Довести элементы до пересечения друг с другом, а затем выделить тот элемент, часть которого перекрывается другим элементом, и затем в панели инструментов (или в контекстном меню элемента) выбрать пункт «На передний план» или нажав по клавише «Q».
* Изменить масштаб диаграммы, для этого надо нажать по кнопке изображающей масштаб и выбрать соответствующий масштаб диаграммы (см. рис.2.10).
* Попытаться завершить работу программы, на что программа ответит диалоговым окном (см. рис. 2.12).
* Выбрать пункт «Сохранить».
* Затем станет активно диалоговое окно, в котором предполагается ввод имени файла для сохранения диаграммы (см. рис. 2.11), ввести имя файла «Курсовая».
* В главном меню выбрать пункт открыть файл или нажать сочетание клавиш «СTRL+O».
* В соответствующем диалоговом окне выбрать файл «Курсовая.dia».
* Завершить работу программы.

### ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы была разработан визуализированный графический редактор с использованием языка программирования высокого уровня С++, в интегрированной среде разработки Qt Creator 2.0.1 – Open Source, отвечающая поставленным требованиям.

Возможности улучшения программы:

* Обеспечить возможность масштабирования фрагментов диаграммы.
* Предоставить возможность изменять место прикрепления различных видов отношений.
* Предоставить возможность изгибать линию, представляющую отношения.

### ЛИТЕРАТУРА

1. Qt Reference Documentation
2. Шлее М. Qt4.5 Профессиональное программирование на С++. – СПб.: БХВ-Петербург, 2010. – 896с.
3. Summerfield, Mark. Advanced Qt programming: creating great software with C++ and Qt 4 /Mark Summerfield.