Департамент внутренней и кадровой политики
Белгородской области
Областное государственное автономное
профессиональное образовательное учреждение
«Валуйский колледж»

Методические рекомендации по организации и выполнению практических заданий

МДК 02.01 Технология разработки программного обеспечения

(МДК, УД)

Специальность

09.02.07 Информационные системы и программирование

Рассмотрено на заседании ПЦК предметно-цикловой комиссией математических дисциплин и информационных технологий, протокол

№ 1 от «31» августа 2020 г.

Руководитель: И. В. Крапивина

Методические рекомендации для проведения практических работ студентов по МДК 02.01 Технология разработки программного обеспечения специальности 09.02.07 Информационные системы и программирование

Составитель: Дураков С.Г., преподаватель

Пояснительная записка

К основным видам учебных занятий наряду с другими (урок, лекция, семинар, контрольная, лабораторная работа, консультация, практика, курсовая работа) относится практическое занятие, которое направлено на формирование учебных и профессиональных практических умений.

Состав и содержание практического занятия определяется его ведущей дидактической целью: формирование практических умений:

- профессиональных (выполнять определенные действия, операции, необходимые в последующем в профессиональной деятельности);
- учебных, необходимых в последующей учебной деятельности.

Состав и содержание практических занятий направлены на реализацию государственных требований к минимуму содержания и уровню подготовки студентов. Методические рекомендации предназначены для студентов второго курса многопрофильного отделения специальности 09.02.07 Информационные системы и программирование Практические занятия в среднем профессиональном образовании являются специфическим педагогическим средством организации и управления деятельностью студентов в учебном процессе.

Практическая работа студентов по МДК 02.01 Технология разработки программного обеспечения проводится с целью:

- систематизации и закрепления полученных теоретических знаний и практических умений студентов;
- углубления и расширения теоретических знаний;
- развития познавательных способностей и активности студентов: творческой инициативы, самостоятельности, ответственности и организованности;
- формирования самостоятельности мышления, способностей к саморазвитию, самосовершенствованию и самореализации;
- развития исследовательских умений.

В данных рекомендациях содержатся планы практических занятий с указанием целей, задач, самостоятельных работ, контрольных работ.

Лабораторная работа 1.

Построение диаграммы Вариантов использования и диаграммы Последовательности.

Цель работы: Научиться строить диаграммы Вариантов использования и диаграммы Последовательности.

Теоретическая справка:

Визуальное моделирование в UML можно представить, как некоторый процесс поуровневого 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.

Разработка диаграммы вариантов использования преследует цели:

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

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.

Состав диаграммы Use Case

Диаграмма вариантов использования состоит из актеров, для которых система производит действие, и собственно действие Use Case, которое описывает то, что актер хочет получить от системы. Актер обозначается значком человечка, а Use Case - овалом. Дополнительно в диаграммы могут быть добавлены комментарии.

Виды взаимодействий

Между актерами и вариантами использования могут быть различные виды взаимодействия. Основные виды взаимодействия следующие:

- ***Простая ассоциация*** - отражается линией между актером и вариантом использования (без стрелки). Отражает связь актера и варианта использования. На рисунке между актером *администратор* и вариантом использования *просматривать заказ*.
- ***Направленная ассоциация*** - то же что и простая ассоциация, но показывает, что вариант использования инициализируется актером. Обозначается стрелкой.
- ***Наследование*** - показывает, что потомок наследует атрибуты и поведение своего прямого предка. Может применяться как для актеров, так для вариантов использования.

- ***Расширение*** (extend) - показывает, что вариант использования расширяет базовую последовательность действий и вставляет собственную последовательность. При этом в отличие от типа отношений "включение" расширенная последовательность может осуществляться в зависимости от определенных условий.
- ***Включение*** (include) - показывает, что вариант использования включается в базовую последовательность и выполняется всегда.

Существуют и другие виды взаимодействия, но они менее важны и реже применяются.

Задание 1. Построить диаграмму вариантов использования модели работы банкомата.

Выполните следующие действия:

1. Добавить актера с именем Клиент банкомата.
2. Добавить вариант использования Снятие наличных по кредитной карте.
3. Добавить направленную ассоциацию от бизнес-актера Клиент Банкомата к варианту использования Снятие наличных по кредитной карте
4. Добавить вариант использования Проверка ПИН-кода, как изображено на рисунке 1.

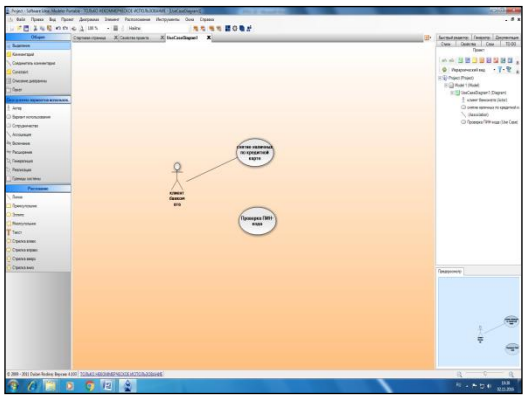

Рисунок 1 – Добавление варианта использования Проверка ПИН-кода

5. Добавить актера с именем Банк.
6. Добавить вариант использования Получение справки о состоянии счета.

7. Добавить вариант использования Блокирование кредитной карточки.

8. Добавить направленную ассоциацию от бизнес-актера Клиент Банкомата к варианту использования Получение справки о состоянии счета.

9. Добавить направленную ассоциацию от варианта использования Снятие наличных по кредитной карточке к сервису Банк.

10. Добавить направленную ассоциацию от варианта использования Получение справки о состоянии счета к сервису Банк.

11. Добавить отношение зависимости со стереотипом «include», направленное от варианта использования Снятие наличных по кредитной карте к варианту использования Проверка Пин-кода.

12. Добавить отношение зависимости со стереотипом «include», направленное от варианта использования Получение справки о состоянии счета к варианту использования Проверка Пин-кода.

13. Добавить отношение зависимости со стереотипом «extend», направленное от варианта использования Блокирование кредитной карточки к варианту использования Проверка Пин-кода.

Задание 2. Построить диаграмму вариантов использования.

Имеются следующие данные:

- четыре действующих лица: Клиента банка, Банк, Кассира и Оператора,
- пять вариантов использования: Снять наличные, Перевести деньги со счета, Положить деньги на счет, Пополнить запас денег и Подтвердить пользователя,
- три зависимости, и отношения между действующими лицами и вариантами использования.

Варианты использования: Снять наличные, Перевести деньги со счета, Положить деньги на счет - требуют включения идентификации клиента в системе. Это поведение может быть выделено в новый вариант использования включения, называемый Подтвердить пользователя. Базовые

варианты использования не зависимы от метода, используемого для идентификации. Поэтому он инкапсулируется (скрывается) в варианте использования включения. С точки зрения базовых вариантов использования не имеет значение производится ли идентификация с помощью магнитной карты или сканированием сетчатки глаза. Они только зависят от результата выполнения варианта использования Подтвердить клиента.

Для создания диаграммы Последовательности необходимо щелкнуть правой кнопкой мыши по пакету Диаграммы взаимодействия и в появившемся меню выбрать пункт New > Sequence Diagram, ввести ее имя, после чего дважды щелкнуть по ней в браузере, как показано на рисунке 2.

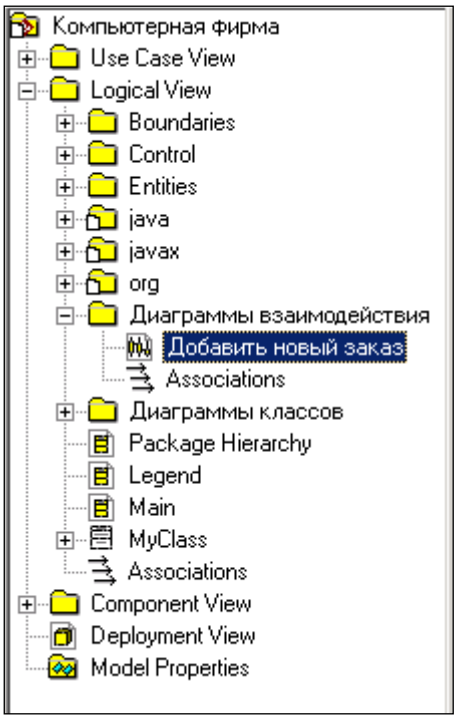

Рисунок 2 – Создание диаграммы последовательности

Построение диаграммы последовательности начинается с размещения на ней объектов, которые будут обмениваться сообщениями. Сначала необходимо разместить объекты, которые посылают сообщения, а потом объекты, получающие их. Инициатором взаимодействия выступает актер *Менеджер по работе с клиентами.* Поэтому на диаграмме он будет находится в левом углу, как показано на рисунке 3.

- объект класса *OrderOptions* (*Параметры работы с заказом*), отвечающий за выбор возможного действия с заказом в рассматриваемом прецеденте;

- объект класса *AddNewOrder* (*Добавление нового заказа*), отвечающий за добавление заказа;

- объект класса *OrderManager* (*Менеджер по работе с заказами*), отвечающий за обработку потока событий рассматриваемого прецедента;

- объект класса *Order* (*Заказ*);

- объект класса *Client* (*Клиент*);

- объект класса *ComponentPart* (*Комплектующее изделие*).

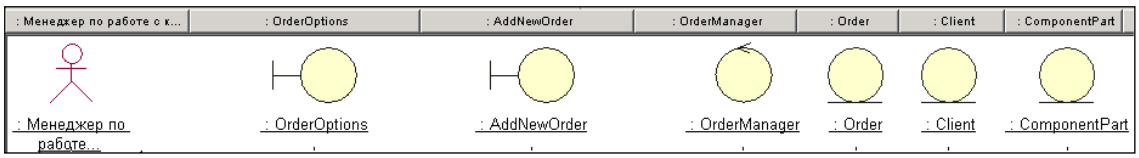

Рисунок 3 – Расположение объектов на диаграмме последовательности

Теперь на диаграмме следует разместить сообщения, которыми будут обмениваться объекты, представленные в таблице 1:

Таблица 1 - Сообщения

№ сообщения	**Объект - отправитель сообщения**	**Объект - получатель сообщения**	**Название**
1	Менеджер по работе с клиентами	OrderOptions	ввод пароля
2	OrderOptions	OrderOptions	проверка пароля
3	Менеджер по работе с клиентами	OrderOptions	выбор операции "добавить"
4	OrderOptions	AddNewOrder	отображение полей ввода
5	Менеджер по работе с клиентами	AddNewOrder	выбор типа компьютера
6	AddNewOrder	OrderManager	получение списка клиентов
7	OrderManager	Client	получение списка клиентов
8	Client	AddNewOrder	список клиентов
9	AddNewOrder	AddNewOrder	отображение списка клиентов

10	Менеджер по работе с клиентами	AddNewOrder	выбор клиента
11	AddNewOrder	OrderManager	получение списка комплектующих
12	OrderManager	ComponentPart	получение списка комплектующих
13	ComponentPart	AddNewOrder	список комплектующих
14	AddNewOrder	AddNewOrder	отображение списка комплектующих
15	Менеджер по работе с клиентами	AddNewOrder	* выбор необходимых комплектующих
16	Менеджер по работе с клиентами	AddNewOrder	сохранить заказ
17	AddNewOrder	OrderManager	передача управления
18	OrderManager	Order	сохранить

В итоге получается диаграмма последовательности, показанная на рисунке 4.

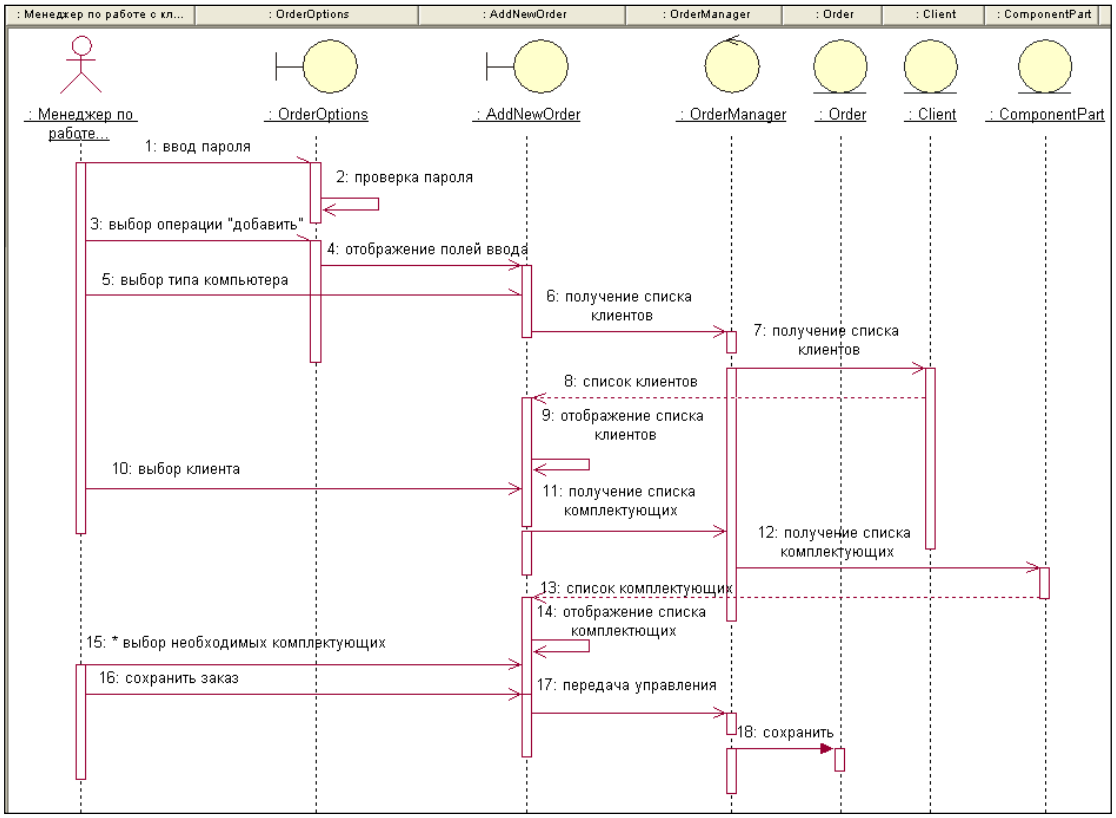

Рисунок 4 – Итоговая диаграмма последовательности.

Лабораторная работа 2.

Построение диаграммы Кооперации и диаграммы Развертывания.

Цель работы: закрепление теоретических сведений о диаграмме кооперации и диаграмме Развертывания; овладение практическими навыками моделирования процессов, описывающих взаимодействие объектов в диаграмме кооперации и диаграмме развертывания.

Теоретические сведения

Диаграмма кооперации (collaboration diagram) предназначена для описания поведения системы на уровне отдельных объектов, которые обмениваются между собой сообщениями, чтобы достичь нужной цели или реализовать некоторый вариант использования.

Кооперация (collaboration) — спецификация множества объектов отдельных классов, совместно взаимодействующих с целью реализации отдельных вариантов использования в общем контексте моделируемой системы.

Кооперация определяет структуру поведения системы в терминах взаимодействия участников этой кооперации.

На диаграмме кооперации размещаются объекты, представляющие собой экземпляры классов, связи между ними, которые в свою очередь являются экземплярами ассоциаций, и сообщения. Связи дополняются стрелками сообщений, а также именами ролей, которые играют объекты в данной взаимосвязи. На диаграмме кооперации показываются структурные отношения между объектами в виде различных соединительных линий и изображаются динамические взаимосвязи — потоки сообщений в форме стрелок с указанием направления рядом с соединительными линиями между объектами, при этом задаются имена сообщений и их порядковые номера в общей последовательности сообщений.

Одна и та же совокупность объектов может участвовать в реализации различных коопераций. В зависимости от рассматриваемой кооперации, могут изменяться как связи между отдельными объектами, так и поток сообщений между ними. Именно это отличает диаграмму кооперации от диаграммы классов, на которой должны быть указаны все без исключения классы, их атрибуты и операции, а также все ассоциации и другие структурные отношения между элементами модели.

Объект (object) — сущность с хорошо определенными границами и индивидуальностью, которая инкапсулирует состояние и поведение. Объект создается на этапе реализации модели или выполнения программы. Он имеет собственное имя и конкретные значения атрибутов.

Для диаграмм кооперации имя объекта – строка текста, разделенная двоеточием: <собственное имя объекта>'/'<Имя роли класса>:<Имя класса>.

Имя роли класса указывается в том случае, когда соответствующий класс отсутствует в модели. Имя класса – это имя одного из классов, представленного на диаграмме классов.

Если указано собственное имя объекта, то оно должно начинаться со строчной буквы. Имя объекта, имя роли с символом "/" или имя класса могут отсутствовать, но ":" всегда должно стоять перед именем класса, а "/" – перед именем роли.

Следующие варианты возможных записей полного имени объекта:

- o : C – объект с собственным именем o, экземпляр класса C.
- : C – анонимный объект, экземпляр класса C.
- o : (или o) – объект-сирота с собственным именем o.
- o / R : C – объект с собственным именем o, экземпляр класса C, играющий роль R.
- / R : C – анонимный объект, экземпляр класса C, играющий роль R.
- o / R – объект-сирота с собственным именем o, играющий роль R.

- / R –анонимный объект и одновременно объект-сирота, играющий роль R.

Составной объект (composite object) или объект-композит предназначен для представления объекта, имеющего собственную структуру и внутренние потоки управления.

Составной объект является экземпляром класса-композита, который связан отношением композиции со своими частями. На диаграммах кооперации составной объект изображается как обычный объект, состоящий из двух секций: верхней и нижней. В верхней секции записывается имя составного объекта, а в нижней – его объекты-части вместо списка атрибутов.

При изображении диаграммы кооперации отношения между объектами описываются с помощью связей, которые являются экземплярами соответствующих ассоциаций.

При изображении диаграммы кооперации отношения между объектами описываются с помощью связей, которые являются экземплярами соответствующих ассоциаций.

Связь (link) — любое семантическое отношение между некоторой совокупностью объектов.

Бинарная связь изображается отрезком сплошной линии, соединяющей два прямоугольника объектов, на концах линии можно указать имена ролей.

Связи на диаграмме кооперации могут быть только анонимными и при необходимости записываются без двоеточия перед именем ассоциации. Имена связей и кратность концевых точек, как правило, на диаграммах кооперации не указываются. Обозначения агрегации и композиции могут присутствовать на отдельных концах связей.

Пример: обобщенная схема компании с именем «с», которая состоит из департаментов (анонимный мультиобъект класса «Департамент»). В последние входят «Сотрудники». Рефлексивная связь указывает на то, что руководитель департамента является одновременно и его сотрудником.

Связь может иметь некоторые стереотипы:

- «association» – ассоциация (предполагается по умолчанию, поэтому может не указываться);
- «parameter» – соответствующий объект может быть только параметром метода;
- «local» – область видимости переменной ограничена только соседним объектом;
- «global» – область видимости переменной распространяется на всю диаграмму кооперации;
- «self» – рефлексивная связь объекта с самим собой, которая допускает передачу объектом сообщения самому себе.

Каждое взаимодействие описывается совокупностью сообщений, которыми участвующие в нем объекты обмениваются между собой.

Сообщение (message) — спецификация передачи информации от одного элемента модели к другому с ожиданием выполнения определенных действий со стороны принимающего элемента.

При этом первый объект предполагает, что после получения сообщения вторым объектом последует выполнение некоторого действия. На диаграмме кооперации сообщение является причиной или стимулом начала выполнения операций, отправки сигналов, создания и уничтожения отдельных объектов. Связь обеспечивает канал для направленной передачи сообщений между объектами от объекта-источника к объекту-получателю.

Иногда отправителя сообщения называют клиентом, а получателя – сервером. При этом сообщение от клиента имеет форму запроса некоторого сервиса, а реакция сервера на запрос после получения сообщения может быть связана с выполнением определенных действий или передачи клиенту необходимой информации тоже в форме сообщения.

Сообщения в языке UML специфицируют роли, которые играют объекты – отправитель и получатель сообщения. Сообщения на диаграмме кооперации изображаются дополнительными стрелками рядом с

соответствующей связью или ролью ассоциации. Направление стрелки указывает на получателя сообщения. На диаграммах кооперации может использоваться один из трех типов стрелок для обозначения сообщений.

Диаграмма кооперации, с одной стороны, обеспечивает концептуально согласованный переход от статической модели диаграммы классов к динамическим моделям поведения, представляемым диаграммами последовательности, состояний и деятельности. С другой стороны, диаграмма этого типа предопределяет особенности реализации модели на диаграммах компонентов и развертывания.

Пример: Программное средство представляет среду для формирования отчетов по лекционному материалу. Пользователь может вводить свою информацию в отчеты, изменять параметры. После оформления отчетов пользователю предоставлена возможность сохранение отчета.

Для построения диаграммы развертывания необходимо:

- изучить на какой платформе и на каких вычислительных средствах реализована ИС;
- рассмотреть возможность отображения физических устройств, которые будут участвовать в работе проектируемой ИС;
- Научиться выявлять узкие места системы и реконфигурировать ее топологию для достижения требуемой производительности.

Содержание работы:

1. Изучить теоретические сведения по теме “Диаграмма кооперации” и «Диаграмма развертывания».
2. Разработать диаграмму кооперации и диаграмму развертывания для произвольной системы индивидуального задания.
3. Оформить отчет, включив в него описание всех компонентов диаграммы кооперации и диаграммы развертывания согласно индивидуальному варианту задания.

Лабораторная работа 3.

Построение диаграммы Деятельности

Цель работы: Научиться строить диаграмму Деятельности.

Теоретические сведения:

При моделировании поведения системы возникает необходимость детализировать особенности алгоритмической и логической реализации выполняемых системой операций. Для моделирования процесса выполнения операций в языке UML используются так называемые диаграммы деятельности. Каждое состояние на диаграмме деятельности соответствует выполнению некоторой элементарной операции, а переход в следующее состояние срабатывает только при завершении этой операции в предыдущем состоянии. Графически диаграмма деятельности представляется в форме графа деятельности, вершинами которого являются состояния действия, а дугами - переходы от одного состояния действия к другому. На диаграмме деятельности отображается логика или последовательность перехода от одной деятельности к другой, при этом внимание фиксируется на результате деятельности. Компонентами диаграммы деятельности являются:

-состояния действия,

-переходы,

-дорожки,

-объекты

Состояние действия. Состояние действия (action state) является специальным случаем состояния с некоторым входным действием и, по крайней мере, одним выходящим из состояния переходом. Этот переход неявно предполагает, что входное действие уже завершилось. Состояние действия не может иметь внутренних переходов, поскольку оно является элементарным. Обычное использование состояния действия заключается в

моделировании одного шага выполнения алгоритма (процедуры) или потока управления.

Внутри фигуры записывается выражение действия (action-expression), которое должно быть уникальным в пределах одной диаграммы деятельности.

Переходы. При построении диаграммы деятельности используются только нетриггерные переходы, т. е. такие, которые срабатывают сразу после завершения деятельности или выполнения соответствующего действия. На диаграмме такой переход изображается сплошной линией со стрелкой.

Если из состояния действия выходит не единственный переход, то сработать может только один из них. Тогда для каждого из таких переходов должно быть явно записано сторожевое условие в форме булевского выражения в прямых скобках.

Дорожки. Диаграммы деятельности могут быть т.к. использованы для моделирования бизнес-процессов. Применительно к бизнес-процессам желательно выполнение каждого действия ассоциировать с конкретным подразделением компании. В этом случае подразделение несет ответственность за реализацию отдельных действий, а сам бизнес-процесс представляется в виде переходов действий из одного подразделения к другому.

Для моделирования этих особенностей в языке UML используется специальная конструкция, получившее название дорожки (swimlanes). При этом все состояния действия на диаграмме деятельности делятся на отдельные группы, которые отделяются друг от друга вертикальными линиями. Две соседние линии и образуют дорожку, а группа состояний между этими линиями выполняется отдельным подразделением.

Названия подразделений явно указываются в верхней части дорожки. Пересекать линию дорожки могут только переходы, которые в этом случае обозначают выход или вход потока управления в соответствующее подразделение компании.

Объекты. В общем случае действия на диаграмме деятельности выполняются над теми или иными объектами. Эти объекты либо инициируют выполнение действий, либо определяют некоторый результат этих действий.

Для графического представления объектов используются прямоугольник класса, с тем отличием, что имя объекта подчеркивается. Далее после имени может указываться характеристика состояния объекта в прямых скобках. Такие прямоугольники объектов присоединяются к состояниям действия отношением зависимости пунктирной линией со стрелкой.

Задание: построить диаграмму Деятельности.

Содержание работы:

1. Изучить теоретические сведения по теме “Диаграмма деятельности».
2. Разработать диаграмму деятельности для произвольной системы индивидуального задания.
3. Оформить отчет, включив в него описание всех компонентов диаграммы деятельности согласно индивидуальному варианту задания.

Лабораторная работа 4.

Построение диаграммы Состояний и диаграммы Классов

Цель работы: закрепление теоретических сведений о диаграмме состояний и диаграмме классов; овладение практическими навыками моделирования процессов, описывающих взаимодействие объектов в диаграмме состояний и диаграмме классов. Ознакомление с методологией и инструментальными средствами моделирования классов на основе языка UML.

Теоретические сведения

Диаграмма классов представляет собой логическую модель статического представления моделируемой системы. Характеристика состояний системы не зависит (или слабо зависит) от логической структуры, зафиксированной в диаграмме классов. Поэтому при рассмотрении состояний системы приходится на время отвлечься от особенностей ее объектной структуры и мыслить совершенно другими категориями, образующими динамический контекст поведения моделируемой системы.

Каждая прикладная система характеризуется не только структурой составляющих ее элементов, но и некоторым поведением или функциональностью. Для общего представления функциональности моделируемой системы предназначены диаграммы вариантов использования, которые на концептуальном уровне описывают поведение системы в целом. Сейчас наша задача заключается в том, чтобы представить поведение более детально на логическом уровне.

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. 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 по существу является графом специального вида, который представляет некоторый автомат. Вершинами этого графа являются состояния и некоторые другие типы элементов автомата (псевдосостояния), которые изображаются соответствующими графическими символами. Дуги графа служат для обозначения переходов из состояния в состояние. Диаграммы состояний могут быть вложены друг в друга, образуая вложенные диаграммы более детального представления отдельных элементов модели. Компонентами диаграммы состояний являются:

-состояния и подсостояния,

-переходы.

Автомат (state machine) в языке UML представляет собой некоторый формализм для моделирования поведения элементов модели и системы в целом. Автомат является пакетом, в котором определено множество понятий, необходимых для представления поведения моделируемой сущности в виде дискретного пространства с конечным числом состояний и переходов. С другой стороны, автомат 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".

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 переход объекта из состояния в состояние происходит мгновенно.

Поведение моделируется как последовательное перемещение по графу состояний от вершины к вершине по связывающим их дугам с учетом их ориентации. Для графа состояний системы можно ввести в рассмотрение специальные свойства.

Одним из таких свойств является выделение из всей совокупности состояний двух специальных: начального и конечного. Предполагается, что последовательность изменения состояний упорядочена во времени. Другими словами, каждое последующее состояние всегда наступает позже предшествующего ему состояния.

Еще одним свойством графа состояний может служить достижимость состояний. Для достижимости состояний необходимо наличие связывающего их ориентированного пути в графе состояний.

Формализм обычного автомата основан на выполнении следующих обязательных условий:

1. Автомат не запоминает историю перемещения из состояния в состояние.

2. В каждый момент времени автомат может находиться в одном и только в одном из своих состояний.

3. Длительность нахождения автомата в том или ином состоянии, а также время достижения того или иного состояния никак не специфицируются, т.е. время на диаграмме состояний присутствует в неявном виде, хотя для отдельных событий может быть указан интервал времени и в явном виде.

4. Количество состояний автомата должно быть обязательно конечным (начальное и конечное состояния).

5. Каждый переход должен обязательно соединять два состояния автомата. Допускается переход из состояния в себя, такой переход еще называют "петлей".

6. Автомат не должен содержать конфликтующих переходов, т. е. таких переходов из одного и того же состояния, когда объект одновременно может перейти в два и более последующих состояния (кроме случая параллельных подавтоматов).

Состояние. 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.

Имя состояния представляет собой строку текста, которая раскрывает содержательный смысл данного состояния. Имя всегда записывается с заглавной буквы. Для идентификации имени состояния рекомендуется использовать глаголы в настоящем времени (звенит, печатает, ожидает) или соответствующие причастия (занят, свободен, передано, получено). Имя у состояния может отсутствовать. В этом случае состояние является анонимным, и если на диаграмме состояний их несколько, то все они должны различаться между собой.

Список внутренних действий. Рассмотрим перечень внутренних действий или деятельностей, которые выполняются в процессе нахождения

моделируемого элемента в данном состоянии. Каждое из действий записывается в виде отдельной строки и имеет следующий формат:

<метка-действия '/' выражение-действия>

Метка действия указывает на обстоятельства или условия, при которых будет выполняться деятельность, определенная выражением действия. Если список выражений действия пустой, то разделитель в виде наклонной черты '/' может не указываться.

Перечень меток действия имеет фиксированные значения в языке UML, которые не могут быть использованы в качестве имен событий. Эти значения следующие:

- entry - эта метка указывает на действие, специфицированное следующим за ней выражением действия, которое выполняется в момент входа в данное состояние (входное действие);
- exit - эта метка указывает на действие, специфицированное следующим за ней выражением действия, которое выполняется в момент выхода из данного состояния (выходное действие);
- do - эта метка специфицирует выполняющуюся деятельность ("do activity"), которая выполняется в течение всего времени, пока объект находится в данном состоянии, или до тех пор, пока не закончится вычисление, специфицированное следующим за ней выражением действия.

В последнем случае при завершении события генерируется соответствующий результат;

- include - эта метка используется для обращения к подавтомату, при этом следующее за ней выражение действия содержит имя этого подавтомата.

Во всех остальных случаях метка действия идентифицирует событие, которое запускает соответствующее выражение действия. Эти события называются внутренними переходами и семантически эквивалентны переходам в само это состояние.

Начальное состояние представляет собой частный случай состояния, которое не содержит никаких внутренних действий. В этом состоянии находится объект по умолчанию в начальный момент времени. Графически начальное состояние в языке UML обозначается в виде закрашенного кружка, 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. Графически конечное состояние в языке UML обозначается в виде закрашенного кружка, помещенного в окружность, в которую может только входить стрелка, соответствующая переходу.

Простой переход (simple transition) представляет собой отношение между двумя последовательными состояниями (исходном и целевом состоянии), которые указывают на факт смены одного состояния другим.

Переход осуществляется при наступлении некоторого события: окончания выполнения деятельности (do activity), получении объектом сообщения или приемом сигнала. На переходе указывается имя события, и могут указываться действия, производимые объектом в ответ на внешние события при переходе из одного состояния в другое. Срабатывание перехода может зависеть не только от наступления некоторого события, но и от выполнения определенного условия, называемого сторожевым условием. Объект перейдет из одного состояния в другое в том случае, если произошло указанное событие и сторожевое условие приняло значение "истина".

Переход может быть направлен в то же состояние, из которого он выходит. В этом случае его называют переходом в себя. Этот переход изображается петлей со стрелкой и отличается от внутреннего перехода. При переходе в себя объект покидает исходное состояние, а затем снова входит в

него. При этом всякий раз выполняются внутренние действия, специфицированные метками entry и exit.

На диаграмме состояний переход изображается сплошной линией со стрелкой, которая направлена в целевое состояние. Каждый переход может быть помечен строкой текста, которая имеет следующий общий формат:

<сигнатура события>'['<сторожевое условие>']' <выражение действия>.

При этом сигнатура события описывает некоторое событие с необходимыми аргументами:

<имя события>'('<список параметров, разделенных запятыми>')'.

Событие. Формально, событие представляет собой спецификацию некоторого факта, имеющего место в пространстве и во времени. Про события говорят, что они "происходят", при этом отдельные события должны быть упорядочены во времени. После наступления некоторого события нельзя уже вернуться к предыдущим событиям, если такая возможность не предусмотрена явно в модели.

В языке UML события играют роль стимулов, которые инициируют переходы из одних состояний в другие. В качестве событий можно рассматривать сигналы, вызовы, окончание фиксированных промежутков времени или моменты окончания выполнения определенных действий. Имя события идентифицирует каждый отдельный переход на диаграмме состояний и может содержать строку текста, начинающуюся со строчной буквы. В этом случае принято считать переход триггерным, т. е. таким, который специфицирует событие-триггер. Например, переходы являются триггерными, поскольку с каждым из них связано некоторое событие-триггер, происходящее асинхронно в момент выхода из строя технического устройства или в момент окончания его ремонта.

Если рядом со стрелкой перехода не указана никакая строка текста, то соответствующий переход является нетриггерным, и в этом случае из контекста диаграммы состояний должно быть ясно, после окончания какой деятельности он срабатывает.

Сторожевое условие (guard condition), если оно есть, всегда записывается в прямых скобках после события-триггера и представляет собой некоторое булевское выражение. Если сторожевое условие принимает значение "истина", то соответствующий переход может сработать. Если же сторожевое условие принимает значение "ложь", то переход не может сработать, и при отсутствии других переходов объект не может перейти в целевое состояние по этому переходу

Примером события-триггера может служить разрыв телефонного соединения с провайдером Интернет-услуг после окончания загрузки электронной почты клиентской почтовой программой (при удаленном доступе к Интернету). В этом случае сторожевое условие есть не что иное, как ответ на вопрос: "Пуст ли почтовый ящик клиента на сервере провайдера?". В случае положительного ответа "истина", следует отключить соединение с провайдером, что и делает автоматически почтовая программа-клиент. В случае отрицательного ответа "ложь", следует оставаться в состоянии загрузки почты и не разрывать телефонное соединение.

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. В качестве примера выражения действия может служить "разорвать телефонное соединение (телефонный номер), которое должно быть выполнено сразу после установления истинности ("истина") сторожевого условия "почтовый ящик на сервере пуст".

Составное состояние (composite state) - такое сложное состояние, которое состоит из других вложенных в него состояний. Последние будут выступать по отношению к первому как **подсостояния** (substate). Хотя между ними имеет место отношение композиции, графически все вершины диаграммы, которые соответствуют вложенным состояниям, изображаются внутри символа составного состояния. В этом случае размеры графического

символа составного состояния увеличиваются, так чтобы вместить в себя все подсостояния. Составное состояние может содержать два или более параллельных подавтомата или несколько последовательных подсостояний. Каждое сложное состояние может уточняться только одним из указанных способов.

Краткое описание методологии моделирования классов в языке UML.

Объект представляет собой экземпляр класса – особую сущность, которая имеет заданные значения атрибутов и операций. Ваша стиральная машина, например, может иметь атрибуты: компания-производитель – Laundatorium, наименование модели – Washmeister, серийный номер изделия – GL57774 и емкость – 16 фунтов.

Атрибуты

Атрибут – это свойство класса. Атрибуты описывают перечень значений, в рамках которых указываются свойства объектов (т.е. экземпляров) этого класса. Класс может не иметь атрибутов или содержать любое их количество. Имена атрибутов, состоящие из одного слова, принято обозначать строчными буквами. Если имя состоит из нескольких слов, то эти слова объединяются, и каждое слово, за исключением первого, начинается с прописной буквы. Список имен атрибутов начинается ниже линии, отделяющей их от имени класса.

UML позволяет отображать дополнительную информацию об атрибутах. В изображении класса можно указать тип для каждого значения атрибута. Перечень возможных типов включает строку, число с плавающей точкой, целое число, логическое значение и другие перечислимые типы. Для отображения типа используется двоеточие, которое отделяет имя атрибута от его типа. Здесь же можно указать значение атрибута по умолчанию.

Операции

Операция – это то, что может выполнять класс, либо то, что вы (или другой класс) можете выполнять над данным классом. Подобно имени

атрибута, имя операции записывается строчными буквами, если это одно слово. Если имя состоит из нескольких слов, они соединяются, и все слова, кроме первого, пишутся с прописной буквы. Список операций начинается ниже линии, отделяющей операции от атрибутов.

Помимо дополнительной информации об атрибутах, можно отобразить дополнительную информацию об операциях. В скобках, следующих за именем операции, можно указать параметр операции и его тип. Один из типов операций, *функция,* по окончании работы возвращает значение. В этом случае можно указать возвращаемое значение и его тип.

Задание: Ознакомиться с методологией моделирования классов на основе языка UML, используя методические указания. Ознакомиться со средствами построения диаграммы состояний и диаграммы классов программного продукта StarUML 5.0. Разработать диаграмму состояний и диаграмму классов автоматизированной системы согласно варианту индивидуального задания, используя инструментальное средство StarUML 5.0. Продемонстрировать результат и защитить работу преподавателю.

Список литературы:

1. Заботина Н. Н. Проектирование информационных систем : учеб. пособие / Н. Н. Заботина. М.: НИЦ «Инфра−М», 2018. 331 с. : ил. ISBN 978−5−16−004509−2.
2. Технология разработки программного обеспечения : учеб. Пособие. Л.Г. Гагарина, Е.В. Кокорева, Б.Д. Виснадул ; под ред. Л.Г. Гагариной. — М. : ИД «ФОРУМ» : ИНФРА-М, 2019.
3. Разработка, внедрение и адаптация программного обеспечения отраслевой направленности : учеб. пособие / Федорова Г. Н.: ИД «КУРС», 2016.
4. Введение в архитектуру программного обеспечения: Учебное пособие / Гагарина Л.Г., Федоров А.Р., Федоров П.А. - М.: ИД ФОРУМ, НИЦ ИНФРА-М, 2016. - 320 с.: 60x90 1/16. - (Высшее образование) (Переплёт 7БЦ) ISBN 978-5-8199-0649-1